AWS Certification Study Group

Databases

Items in this color have been called out as possible exam topics.

Items in this color were walked through in the console.

Items in this color are exam topics that we walked through in the console.

Domains Covered In This Discussion

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

- 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

Data Security

- Recognize and implement secure practices for optimum cloud deployment and maintenance. Content may include the following:
    - AWS administration and security services
    - Design patterns
- Recognize critical disaster recovery techniques and their implementation.

Essential Databases and Concepts

This chapter covers essential database concepts and three of Amazon's databases – RDS (Relational Database Service), DynamoDB (NoSQL) and data warehouse (Redshift).

We'll discuss the differences between those databases and focus on the following key topics:

- The benefits and tradeoffs of running a database on EC2 or RDS.
- How to deploy database engines to the cloud.
- How to backup and recover and meet RTO and RPO.
- High Availability.
- Secure your databases, tables and clusters.
- Scale database compute and storage vertically and use read replicas to scale horizontally.
- Select the right type of storage volume.
- Design and scale a DynamoDB table, how to read and write from a DynamoDB table and use secondary indexes to speed queries.

- Design a Redshift table and how to load and query a data warehouse

Relational Databases (Amazon RDS)

Primer

- Provides a common interface that lets users read and write from the database using commands or queries written using Structured Query Language (SQL).
- Consists of one or more tables, and a table consists of columns (contains a specific attribute of the record) and rows (comprises an individual record) similar to a spreadsheet.
- 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, depending on how the tables are organized and how the application uses the relational database:
    - Online Transaction Processing (OLTP)
        - Transaction-oriented applications that are frequently writing and changing data (for example, data entry and e-commerce).
    - Online Analytical Processing (OLAP)
        - 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.

RDS Service

- Simplifies the setup, operations, and scaling of a relational database on AWS by offloading common tasks like backups, patching, scaling, and replication. Additionally streamlines installation of database software and infrastructure capacity.
- Exposes and endpoint to which client software can connect and execute SQL. Does not provide shell access to DB instances and restricts access to certain system procedures and tables that require advanced privileges. Can typically use the same tools to query, analyze, modify, and administer the database.

Operational Benefits

- Applies consistent deployment by limiting types of changes that can be made to underlying infrastructure through automation.
- Limits 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

RDS DB Instances

- A DB Instance is an isolated database environment deployed in your private network segments, running and managing any of six relational database engines.
    - MySQL 5.1, 5.5, 5.6 and 5.7 Community Edition
        - Storage Engine = InnoDB
        - Licensing = Open Source
        - High Availability = Multi-AZ
        - Horizontal Scaling = Read Replicas
    - Oracle 11g and 12c on Standard One, Standard and Enterprise
        - Licensing = Included or BYOL
        - High Availability = Multi-AZ (all versions)
    - PostgreSQL 9.3.x, 9.4.x and 9.5.x
        - Licensing = Open Source
        - High Availability = Multi-AZ
        - Horizontal Scaling = Read Replicas
    - Microsoft SQL Server 2008 R2, 2012 and 2014
        - Storage Engine = Express, Web, Standard and Enterprise
        - Licensing = Included or BYOL
        - High Availability = Multi-AZ on Standard and Enterprise only
    - MariaDB 10.0.17
        - Storage Engine = XtraDB
        - Licensing = Open Source
        - High Availability = Multi-AZ
        - Horizontal Scaling = Read Replicas
    - Amazon Aurora
        - Redesigned MySQL to take a more service-oriented approach.
        - Up to 5x performance of MySQL
        - DB cluster has one or more instances and includes a cluster volume that manages data for those instances. Spans multiple AZs, with each containing a copy of cluster data.
        - Primary instance is the main instance which supports both write and read workloads.
        - Aurora Replica is a secondary instance that supports only read. Max of 15 replicas (increases performance) and can be located in multiple AZs (increases availability).

    - 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.
- Launch a new instance by calling the *CreateDBInstance* API or by using the console. Each 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.
- Existing instances can be changed or resized using the *ModifyDBInstance* API or the console.
- DB Instance Class determines compute and memory that best meets your needs.
    - 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.
- Many features and common configuration settings are exposed and managed using DB parameter groups and DB option groups.
    - 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.
    - 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.

Storage Options

- RDS is built using EBS.
- Depending on DB engine and workload, you can scale up to 4-6 TB provisioned storage and up to 30K IOPS.
- Supports three storage types:
    - Magnetic, or standard storage, offers cost-effective for low I/O requirements.
    - General Purpose Solid State , or gp2, provides faster access than magnetic and can provide burst performance to meet spikes for small to medium size DBs.
    - Provisioned IOPS Solid State is designed to meet I/O-intensive workloads sensitive to storage performance and consistency in random I/O throughput.

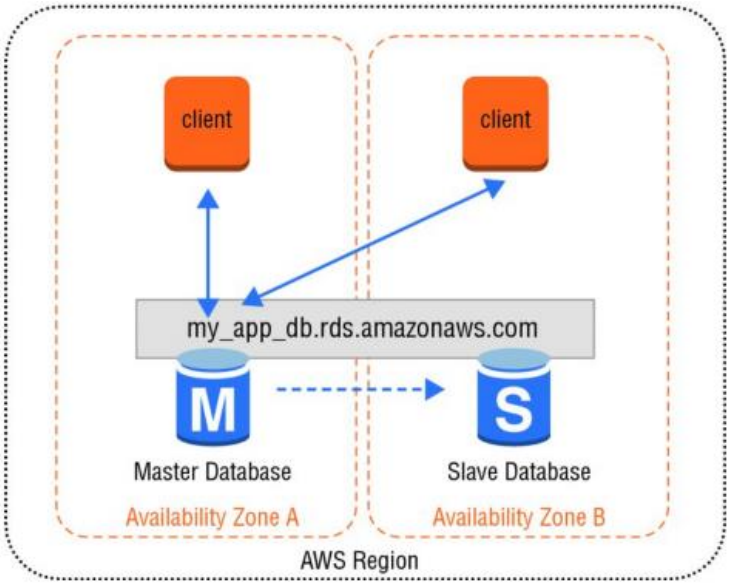
Backup and Recovery

- Concepts
  - Recovery Point Objective = maximum data loss (point in time) acceptable
  - Recovery Time Objective = maximum downtime permitted
  - For busy DBs, use Multi-AZ to minimize the performance impact of a snapshot since storage I/O may be suspended while backing up data (typically the duration of the snapshot).
- Automated backups
  - RDS continuously tracks changes and backs up database.
  - Creates storage volume snapshot, backing up the entire DB instance, not just individual databases.
  - Backup Window = configurable 30-minute maintenance window that occurs daily
  - Backup Retention Period = 1 day default, 35 day max
    - Can restore to any specific time during this period, creating a new DB instance.
  - When you delete a DB instance, all automated backup snapshots are deleted and cannot be recovered.
- Manual Snapshots
  - Can be performed at any time, as frequently as desired.
  - Can restore to specific state within the snapshot.
  - Kept until you explicitly delete them.
  - When you delete a DB instance, manual backup snapshots are maintained.
- Recovery
  - Cannot restore from a DB snapshot to an existing DB instance. A new instance is created when you restore and only the default DB parameter and security groups are associated with the restored instance. You should immediately associate any custom DB parameter or security groups to the restored instance.
  - 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

- Multi-AZ deployments allows you to create a DB cluster across multiple AZs. Amazon RDS Multi-AZ reduces the complexity of setting this up and uses synchronous replication.
- Places a secondary copy of your database in another AZ for DR purposes and available to all RDS DB engines.
- The DB instance endpoint (DNS name) is used when creating the connection to your DB and RDS will automatically failover to the standby instance behind the DNS endpoint (RDS service changes the CNAME). You can also perform a manual failover.
- Both automatic and manual failover takes a matter of seconds to complete. This is to be used for DR only and not to enhance DB performance.
- RDS detects and automatically recovers from most common failure scenarios like:
  - Loss of availability in primary Availability Zone
  - Loss of network connectivity to primary

- Compute unit failure on primary database
- Storage failure on primary database



Auto-scaling

- Scaling Up (Scaling "Vertically")
  - Adding more compute and storage resources by getting a larger machine, allows you to process more transactions, run more queries and store more data.
  - Can be scheduled to occur during next maintenance window or immediately (***ModifyDBInstance*** action).
  - Changing Compute and Memory is done by selecting a different Instance class of the DB. RDS will automate the migration process to a new class with only a short disruption.
  - Can change the amount, class and performance of storage from 5GB up to 6TB (depending on the storage type and engine). Can be increased over time as needed. Supported for all engines but SQL Server.
- Scaling Out (Scaling "Horizontally")
  - Often more difficult than scaling vertically and limited to only some DB engines. Required once vertical capacity limits are reached
  - Sharding
    - Partitioning a large DB into multiple instances (called shards) allows the DB to handle more requests.
    - Requires additional logic at the application layer, since the app will decide how to route requests to the correct shard and becomes limited in the types of queries that can be performed across server boundaries.
    - Supported DBs = NoSQL databases like DynamoDB or Cassandra.
  - Read replicas
    - Offloads read transactions from the primary DB and helps increase the overall number of transactions.

- Designed for read heavy workloads. Updates made to the source DB instance are asynchronously copied to the read replica. Read queries are then routed to your read replica to reduce load on the source DB instance.
- Can create one or more replicas within a single Region or across multiple Regions. Cross-region read replicas serve traffic from a region closest to your users to reduce global latencies. Can also be used to migrate DB across region.
- Supported DBs = RDS for MySQL, PostgreSQL, MariaDB and Aurora.
- Some common read replica scenarios:
    - 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.

Security

- Use IAM to protect Infrastructure resources.
- Use a private subnet within a VPC to deploy DB instances that limits network access. You must first create a DB subnet group that predefines which subnets are available for RDS deployments.
- Restrict network access using ACLs and Security Groups to limit inbound traffic to a short list of IP addresses.
- At database level (using DB engine-specific access control and user management mechanisms), create users and grant permissions to read and write the DB. Rotate passwords frequently.
- Encryption protects confidentiality of data in transit and at rest. Features vary, but is available, across all engines. Use SSL for data in transit and KMS (Key Management Service) or TDE (Transparent Data Encryption) for data at rest. All logs, backups and snapshots are encrypted at the same time.

Data Warehouses (Redshift)

Primer

- 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.
- Typically used to compile reports and search the database using highly complex queries. 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 (frequently and relatively simple), and the other database as their data warehouse for OLAP (less frequently and more complex).

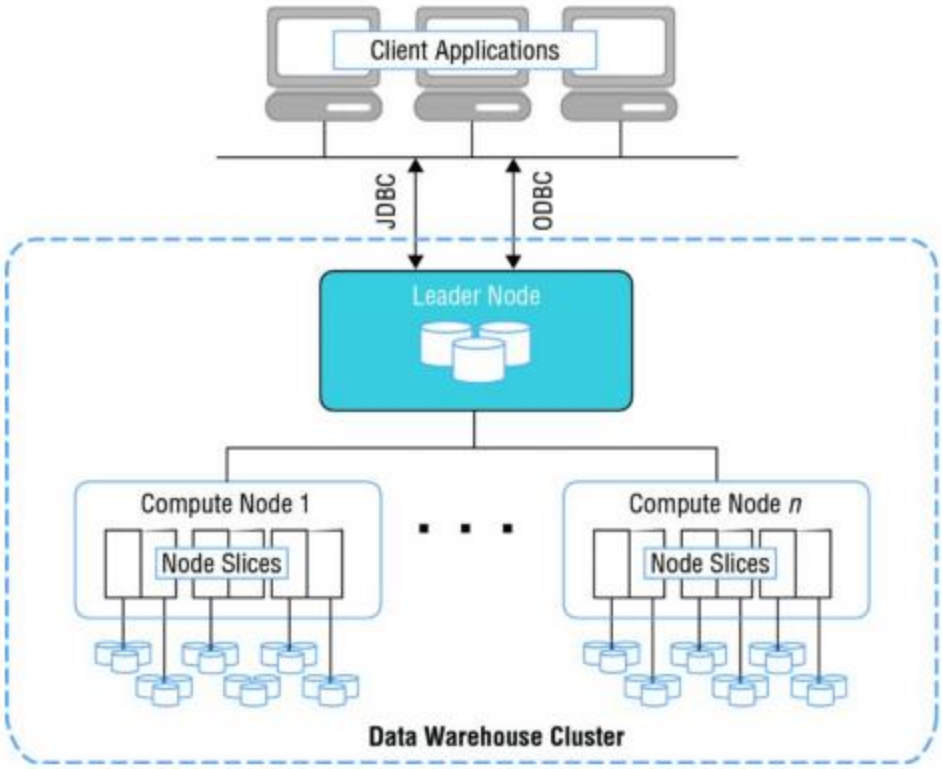
- Amazon RDS is often used for OLTP workloads, but it can also be used for OLAP. Amazon Redshift is a high-performance data warehouse designed specifically for OLAP use cases. It is also common to combine Amazon RDS with Amazon Redshift in the same application and periodically extract recent transactions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

Amazon Redshift Service

- Fast, powerful, fully managed petabyte-scale warehouse service in the cloud. Relational database designed for OLAP scenarios and optimized for high performance analysis and reporting of large datasets.
- Fast querying capabilities over structured data using standard SQL commands to support interactive querying over large datasets. With connectivity via ODBC or JDBC, integrates well with various data loading, reporting, data mining, and analytics tools. Amazon Redshift is based on industry-standard PostgreSQL, so most existing SQL client applications will work with only minimal changes.
- Manages the work needed to set up, operate, and scale a data warehouse. Provisioning the infrastructure capacity, automating backups and patching, automatic monitoring of nodes and drives to help you recover from failures.
- 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.

Clusters and Nodes

- A cluster is composed of a leader node and one or more compute nodes. The client application interacts directly only with the leader node, and the compute nodes are transparent to external applications.
- Currently supports six different node types and each has a different mix of CPU, memory, and storage. The six node types are grouped into two categories: Dense Compute and Dense Storage.
    - Dense Compute node types support clusters up to 326TB using fast SSDs
    - Dense Storage nodes support clusters up to 2PB using large magnetic disks.
    - Each cluster consists of one leader node and one or more compute nodes.
- Each cluster contains one or more databases. User data for each table is distributed across the compute nodes. Your application or SQL client communicates with Redshift using standard JDBC or ODBC connections with the leader node, which in turn coordinates query execution with the compute nodes. Your application does not interact directly with the compute nodes.
- Disk storage for a compute node is divided into a number of slices. The number of slices per node depends on the node size of the cluster and typically varies between 2 and 16. The nodes all participate in parallel query execution, working on data that is distributed as evenly as possible across the slices.



Performance and Scaling

- You can increase query performance by adding multiple nodes to a cluster. When you submit a query, Amazon Redshift distributes and executes the query in parallel across all of a cluster's compute nodes. Amazon Redshift also spreads your table data across all compute nodes in a cluster based on a distribution strategy that you specify. This partitioning of data across multiple compute resources allows you to achieve high levels of performance.
- You can resize a cluster to add storage and compute capacity over time as needs evolve. You can also change the node type of a cluster and keep the overall size the same. Whenever you perform a resize operation, Amazon Redshift will create a new cluster and migrate data from the old cluster to the new one. During a resize operation, the database will become read-only until the operation is finished.

Table Design & Strategy

- Each cluster can support one or more databases, and each database can contain many tables.
- Like most SQL-based databases, you can create a table using the ***CREATE TABLE*** command. This command specifies the name of the table, the columns, and their data types. It also supports specifying compression encodings, distribution strategy, and sort keys.
- Data Types
    - Columns support a wide range of data types such as common numeric data types like INTEGER, DECIMAL, and DOUBLE, text data types like CHAR and VARCHAR, and date data types like DATE and TIMESTAMP. Additional columns can be added to a table using the ***ALTER TABLE*** command; however, existing columns cannot be modified.

- Compression Encoding
    - Data compression happens when loading data for the first time into an empty table, Amazon Redshift will automatically sample your data and select the best compression scheme for each column. Alternatively, you can specify compression encoding on a per-column basis as part of the ***CREATE TABLE*** command.
- Distribution Strategy
    - One of the primary decisions when creating a table in Amazon Redshift is how to distribute the records across the nodes and slices in a cluster. You can configure the distribution style of a table to give Amazon Redshift hints as to how the data should be partitioned to best meet your query patterns.
    - When you run a query, the optimizer shifts the rows to the compute nodes as needed to perform any joins and aggregates. The goal in selecting a table distribution style is to minimize the impact of the redistribution step by putting the data where it needs to be before the query is performed.
    - The data distribution style that you select for your database has a big impact on query performance, storage requirements, data loading, and maintenance. By choosing the best distribution strategy for each table, you can balance your data distribution and significantly improve overall system performance.
    - When creating a table, you can choose between one of three distribution styles:
        - EVEN distribution is the default and data is distributed across the slices in a uniform fashion regardless of the data.
        - KEY distribution distributes the rows according to the values in one column. The leader node will store matching values close together and increase query performance for joins.
        - ALL distribution distributes a full copy of the entire table to every node. This is useful for lookup tables and other large tables that are not updated frequently.
- Sort Keys
    - Sorting enables efficient handling of range-restricted predicates. If a query uses a range-restricted predicate, the query processor can rapidly skip over large numbers of blocks during table scans. The sort keys for a table can be either compound or interleaved.
    - A compound sort key is more efficient when query predicates use a prefix, which is a subset of the sort key columns in order.
    - An interleaved sort key gives equal weight to each column in the sort key, so query predicates can use any subset of the columns that make up the sort key, in any order.

Loading Data

- Redshift supports standard SQL commands like ***INSERT*** and ***UPDATE*** to create and modify records in a table. For bulk operations, however, Amazon Redshift provides the ***COPY*** command as a much more efficient alternative than repeatedly calling ***INSERT***.
    - A ***COPY*** command can load data into a table in the most efficient manner, and it supports multiple types of input data sources. The fastest way to load data into Amazon Redshift is doing bulk data loads from flat files stored in an S3 bucket or from an Amazon DynamoDB table.

- When loading data from Amazon S3, the ***COPY*** command can read from multiple files at the same time. Amazon Redshift can distribute the workload to the nodes and perform the load process in parallel. Instead of having one single large file with your data, you can enable parallel processing by having a cluster with multiple nodes and multiple input files. After each bulk data load that modifies a significant amount of data, you will need to perform a ***VACUUM*** command to reorganize your data and reclaim space after deletes. It is also recommended to run an ***ANALYZE*** command to update table statistics. Data can also be exported out of Amazon Redshift using the ***UNLOAD*** command. This command can be used to generate delimited text files and store them in Amazon S3.

Querying Data

- Redshift allows you to write standard SQL commands to query your tables. By supporting commands like ***SELECT*** to query and join tables, analysts can quickly become productive using Amazon Redshift or integrate it easily.
- For complex queries, you can analyze the query plan to better optimize your access pattern. You can monitor the performance of the cluster and specific queries using Amazon CloudWatch and the Amazon Redshift web console.
- For large Amazon Redshift clusters supporting many users, you can configure Workload Management (WLM) to queue and prioritize queries. WLM allows you define multiple queues and set the concurrency level for each queue. For example, you might want to have one queue set up for long-running queries and limit the concurrency and another queue for short-running queries and allow higher levels of concurrency.

Backups and Restores

- Concepts
  - You can create point-in-time snapshots of your Amazon Redshift cluster. A snapshot can then be used to restore a copy or create a clone of your original Amazon Redshift cluster.
  - Durably stored internally in Amazon S3 by Amazon Redshift.
- Automated snapshots
  - Periodically take snapshots of your cluster and keep a copy for a configurable retention period.
- Manual snapshots
  - Create and share across regions or even with other AWS accounts.
  - Manual snapshots are retained until you explicitly delete them.

Security

- With IAM, you can create policies that grant other AWS users the permission to create and manage the lifecycle of a cluster, including scaling, backup, and recovery operations.
- At the network level, Amazon Redshift clusters can be deployed within the private IP address space of your Amazon VPC to restrict overall network connectivity. Fine-grained network access can be further restricted using security groups and network ACLs at the subnet level.
- When you initially create an Amazon Redshift cluster, you will create a master user account and password.

    - The master account can be used to log in to the Amazon Redshift database and to create more users and groups.
    - Each database user can be granted permission to schemas, tables, and other database objects. These permissions are independent from the IAM policies used to control access to the infrastructure resources and the Amazon Redshift cluster configuration.
- Supports encryption of data in transit using SSL-encrypted connections, and also encryption of data at rest by integrating with KMS and AWS CloudHSM for encryption key management services.

NoSQL Databases (DynamoDB)

Primer

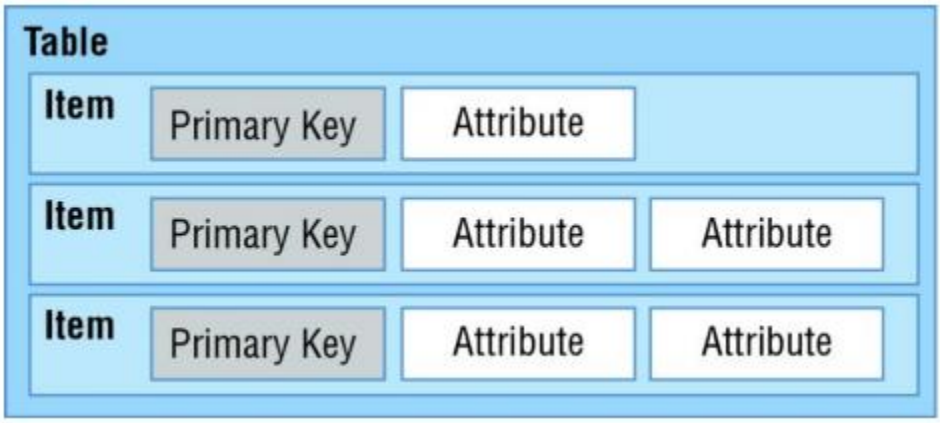
- Simple to use, more flexible, and can achieve performance levels that are difficult or impossible with traditional relational databases. Traditional relational databases are difficult to scale beyond a single server without significant engineering and cost, but a NoSQL architecture allows for horizontal scalability on commodity hardware.
- Non-relational and do not have the same table and column semantics of a relational database. 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).
- 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.

DynamoDB Service

- Provides consistent performance levels by automatically distributing the data and traffic for a table over multiple partitions. After configuring a certain read or write capacity, Amazon DynamoDB will automatically add enough infrastructure capacity to support the requested throughput levels. As your demand changes over time, you can adjust the read or write capacity after a table has been created, and Amazon DynamoDB will add or remove infrastructure and adjust the internal partitioning accordingly.
- All table data are stored on high-performance SSD drives to help maintain consistent, fast performance levels.
- Performance metrics, including transactions rates, can be monitored using Amazon CloudWatch.
- Provides automatic high-availability and durability protections by replicating data across multiple Availability Zones within an AWS Region.

Data Model

- The basic components of the Amazon DynamoDB data model include tables, items, and attributes.
- A table is a collection of items and each item is a collection of one or more attributes. Each item also has a primary key that uniquely identifies the item.



- Unlike a Relational database that has a predefined schema, DynamoDB only requires that a table have a primary key, but it does not require you to define all of the attribute names and data types in advance.
    - Individual items in an Amazon DynamoDB table can have any number of attributes, although there is a limit of 400KB on the item size.
    - Each attribute in an item is a name/value pair. An attribute can be a single-valued or multi-valued set.
        - For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed.
- Applications can connect to the Amazon DynamoDB service endpoint and submit requests over HTTP/S to read and write items to a table or even to create and delete tables. DynamoDB provides a web service API that accepts requests in JSON format. While you could program directly against the web service API endpoints, most developers choose to use the AWS Software Development Kit (SDK) to interact with their items and tables.

Data Types

- Unlike a traditional relational database that requires you to define your column types ahead of time, DynamoDB only requires a primary key attribute. Each item that is added to the table can then add additional attributes.
- This gives you flexibility over time to expand your schema without having to rebuild the entire table and deal with record version differences with application logic. When you create a table or a secondary index, you must specify the names and data types of each primary key attribute (partition key and sort key).
- The four formats of a NoSQL database are:
    1. Document Databases
    2. Graph Stores
    3. Key/Value stores
    4. Wide-column stores

- Amazon DynamoDB supports a wide range of data types for attributes. Data types fall into three major categories:
    1. A scalar type represents exactly one value. Amazon DynamoDB supports the following five scalar types:
        - String
            - Text and variable length characters up to 400KB. Supports Unicode with UTF8 encoding.
        - Number
            - Positive or negative number with up to 38 digits of precision
        - Binary
            - Binary data, images, compressed objects up to 400KB in size
        - Boolean
            - Binary flag representing a true or false value
        - Null
            - Represents a blank, empty, or unknown state. String, Number, Binary, Boolean cannot be empty.
    2. Set Data Types are useful to represent a unique list of one or more scalar values. Each value in a set needs to be unique and must be the same data type. Sets do not guarantee order. Amazon DynamoDB supports three set types:
        - String Set
            - Unique list of String attributes
        - Number Set
            - Unique list of Number attributes
        - Binary Set
            - Unique list of Binary attributes
    3. Document Data Types are useful to represent multiple nested attributes, similar to the structure of a JSON file. Amazon DynamoDB supports two document types and can be combined and nested to create complex structures:
        - List
            - Each List can be used to store an ordered list of attributes of different data types.
        - Map
            - Each Map can be used to store an unordered list of key/ value pairs. Maps can be used to represent the structure of any JSON object.

Primary Key

- When creating a table, you must specify the primary key of the table in addition to the table name. Like a relational database, the primary key uniquely identifies each item in the table.
- A primary key will point to exactly one item.
- Amazon DynamoDB supports two types of primary keys, and this configuration cannot be changed after a table has been created:
    1. Partition Key

        - The primary key is made of one attribute, a partition (or hash) key. Amazon DynamoDB builds an unordered hash index on this primary key attribute.
    2. Partition and Sort Key
        - The primary key is made of two attributes. The first attribute is the partition key and the second one is the sort (or range) key. Each item in the table is uniquely identified by the combination of its partition and sort key values.
        - It is possible for two items to have the same partition key value, but those two items must have different sort key values. Furthermore, each primary key attribute must be defined as type string, number, or binary. Amazon DynamoDB uses the partition key to distribute the request to the right partition.
- If you are performing many reads or writes per second on the same primary key, you will not be able to fully use the compute capacity of the Amazon DynamoDB cluster. A best practice is to maximize your throughput by distributing requests across the full range of partition keys.

Provisioned Capacity

- You are required to provision a certain amount of read and write capacity to handle your expected workloads. Based on your configuration settings, DynamoDB will then provision the right amount of infrastructure capacity to meet your requirements with sustained, low-latency response times.
- Overall capacity is measured in read and write capacity units. These values can later be scaled up or down by using an ***UpdateTable*** action.
- Each operation against an Amazon DynamoDB table will consume some of the provisioned capacity units. The specific amount of capacity units consumed depends largely on the size of the item, but also on other factors. For read operations, the amount of capacity consumed also depends on the read consistency selected in the request.
    - For example, given a table without a local secondary index, you will consume 1 capacity unit if you read an item that is 4KB or smaller. Similarly, for write operations you will consume 1 capacity unit if you write an item that is 1KB or smaller. This means that if you read an item that is 110KB, you will consume 28 capacity units, or 110 / 4 = 27.5 rounded up to 28. For read operations that are strongly consistent, they will use twice the number of capacity units, or 56 in this example.
- You can use Amazon CloudWatch to monitor your Amazon DynamoDB capacity and make scaling decisions. There is a rich set of metrics, including *ConsumedReadCapacityUnits* and *ConsumedWriteCapacityUnits*. If you do exceed your provisioned capacity for a period of time, requests will be throttled and can be retried later. You can monitor and alert on the *ThrottledRequests* metric using Amazon CloudWatch to notify you of changing usage patterns.

Secondary Indexes

- When you create a table with a partition and sort key (formerly known as a hash and range key), you can optionally define one or more secondary indexes on that table. A secondary index lets you query the data in the table using an alternate key, in addition to queries against the primary key.
- Amazon DynamoDB supports two different kinds of indexes:
    1. Global Secondary Index

        - An index with a partition and sort key that can be different from those on the table. You can create or delete a global secondary index on a table at any time.
    2. Local Secondary Index
        - An index that has the same partition key attribute as the primary key of the table, but a different sort key. You can only create a local secondary index when you create a table.
2. Secondary indexes allow you to search a large table efficiently and avoid an expensive scan operation to find items with specific attributes. These indexes allow you to support different query access patterns and use cases beyond what is possible with only a primary key.
3. While a table can only have one local secondary index, you can have multiple global secondary indexes. Amazon DynamoDB updates each secondary index when an item is modified. These updates consume write capacity units. For a local secondary index, item updates will consume write capacity units from the main table, while global secondary indexes maintain their own provisioned throughput settings separate from the table.

Writing and Reading Data

- After you create a table with a primary key and indexes, you can begin writing and reading items to the table.
- Amazon DynamoDB provides multiple operations that let you create, update, and delete individual items. It also provides multiple querying options that let you search a table or an index or retrieve back a specific item or a batch of items.

Eventual Consistency

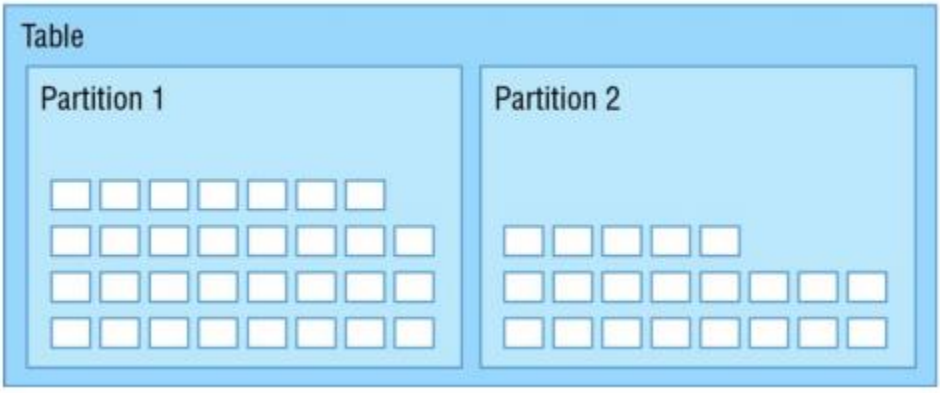
- When reading items from Amazon DynamoDB, the operation can be either eventually consistent or strongly consistent. Amazon DynamoDB is a distributed system that stores multiple copies of an item across an AWS Region to provide high availability and increased durability. When an item is updated in Amazon DynamoDB, it starts replicating across multiple servers. Because Amazon DynamoDB is a distributed system, the replication can take some time to complete. Because of this we refer to the data as being eventually consistent, meaning that a read request immediately after a write operation might not show the latest change. In some cases, the application needs to guarantee that the data is the latest and Amazon DynamoDB offers an option for strongly consistent reads.
- Eventually Consistent Reads
    - When you read data, the response might not reflect the results of a recently completed write operation. The response might include some stale data. Consistency across all copies of the data is usually reached within a second; if you repeat your read request after a short time, the response returns the latest data.
- Strongly Consistent Reads
    - When you issue a strongly consistent read request, Amazon DynamoDB returns a response with the most up-to-date data that reflects updates by all prior related write operations to which Amazon DynamoDB returned a successful response. A strongly consistent read might be less available in the case of a network delay or outage. You can request a strongly consistent read result by specifying optional parameters in your request.

Searching Items

- Two operations, Query and Scan, can be used to search a table or an index.
- A Query operation is the primary search operation you can use to find items in a table or a secondary index using only primary key attribute values.
    - Each Query requires a partition key attribute name and a distinct value to search. You can optionally provide a sort key value and use a comparison operator to refine the search results. Results are automatically sorted by the primary key and are limited to 1MB.
- A Scan operation will read every item in a table or a secondary index. By default, a Scan operation returns all of the data attributes for every item in the table or index.
    - Each request can return up to 1MB of data. Items can be filtered out using expressions, but this can be a resource-intensive operation.
- If the result set for a Query or a Scan exceeds 1MB, you can page through the results in 1MB increments.

Scaling and Partitioning

- An Amazon DynamoDB table can scale horizontally through the use of partitions to meet the storage and performance requirements of your application.
    - Each individual partition represents a unit of compute and storage capacity. A well-designed application will take the partition structure of a table into account to distribute read and write transactions evenly and achieve high transaction rates at low latencies.
    - Amazon DynamoDB stores items for a single table across multiple partitions. Amazon DynamoDB decides which partition to store the item in based on the partition key. The partition key is used to distribute the new item among all of the available partitions, and items with the same partition key will be stored on the same partition.



- Provisioned throughput allocated to a partition is entirely dedicated to that partition, and there is no sharing of provisioned throughput across partitions.
- When a table is created, Amazon DynamoDB configures the table's partitions based on the desired read and write capacity.
    - One single partition can hold about 10GB of data and supports a maximum of 3,000 read capacity units or 1,000 write capacity units.

    - For partitions that are not fully using their provisioned capacity, Amazon DynamoDB provides some burst capacity to handle spikes in traffic. A portion of your unused capacity will be reserved to handle bursts for short periods.
    - As storage or capacity requirements change, Amazon DynamoDB can split a partition to accommodate more data or higher provisioned request rates. After a partition is split, however, it cannot be merged back together. Keep this in mind when planning to increase provisioned capacity temporarily and then lower it again. With each additional partition added, its share of the provisioned capacity is reduced.
- To achieve the full amount of request throughput provisioned for a table, keep your workload spread evenly across the partition key values.
- To maximize Amazon DynamoDB throughput, create tables with a partition key that has a large number of distinct values and ensure that the values are requested fairly uniformly. Adding a random element that can be calculated or hashed is one common technique to improve partition distribution.

Security

- Use IAM policies that allow or deny specific operations on specific tables. You can also use conditions to restrict access to individual items or attributes.
- All operations must first be authenticated as a valid user or user session. Applications that need to read and write from Amazon DynamoDB need to obtain a set of temporary or permanent access control keys. Use IAM Amazon EC2 instance profiles to manage credentials. This allows you to avoid storing sensitive keys in configuration files that must then be secured.
- Fine-grained access control can restrict access to specific items within a table or even specific attributes within an item. Using conditions in an IAM policy allows you to restrict which actions a user can perform, on which tables, and to which attributes a user can read or write.

DynamoDB Streams

- A common requirement for many applications is to keep track of recent changes and then perform some kind of processing on the changed records. Amazon DynamoDB Streams makes it easy to get a list of item modifications for the last 24-hour period.
- Amazon DynamoDB Streams allows you to extend application functionality without modifying the original application. By reading the log of activity changes from the stream, you can build new integrations or support new reporting requirements that weren't part of the original design.
- Each item change is buffered in a time-ordered sequence or stream that can be read by other applications. Changes are logged to the stream in near real-time and allow you to respond quickly or chain together a sequence of events based on a modification.
- Streams can be enabled or disabled for an Amazon DynamoDB table using the AWS Management Console, Command Line Interface (CLI), or SDK. A stream consists of stream records. Each stream record represents a single data modification in the Amazon DynamoDB table to which the stream belongs. Each stream record is assigned a sequence number, reflecting the order in which the record was published to the stream.
- Stream records are organized into groups, also referred to as shards. Each shard acts as a container for multiple stream records and contains information on accessing and iterating

through the records. Shards live for a maximum of 24 hours and, with fluctuating load levels, could be split one or more times before they are eventually closed.

- To build an application that reads from a shard, it is recommended to use the Amazon DynamoDB Streams Kinesis Adapter. The Kinesis Client Library (KCL) simplifies the application logic required to process reading records from streams and shards.

Exercises - RDS

1. Create an RDS instance.
    i. Select Aurora.
    ii. Use Multi-AZ and GP storage.
    iii. DB Instance identifier and database name to MySQL123 (admin123).
    iv. Launch the instance.
2. Simulate a Failover from One AZ to Another
    i. Select the instance, and issue a Reboot command from the actions menu. You have now simulated a failover from one Availability Zone to another using Multi-AZ failover.
3. Create a read replica
    i. Issue a Create Read Replica command from the list of actions.
    ii. Configure the name of the read replica and any other settings. Create the replica.
    iii. When it is complete, delete both the MySQL123 and MySQLReadReplica databases by clicking the checkboxes next to them, clicking the Instance Actions drop-down box, and then clicking Delete.
4. Delete the instance.

Exercises - DynamoDB

1. Read and Write from a DynamoDB Table.
    i. Create a new table named UserProfile with a partition key of userID of type String.
    ii. Create and save a new item in the table. Set the userID to U01, and append another String attribute called **name** with a value of **Joe**.
    iii. Perform a scan on the table to retrieve the new item.
2. Delete the table.

Exercises - Redshift

1. Launch a Redshift cluster.
    i. Create a new cluster (training), configuring the database name (traindb), username, and password (Admin123).
    ii. Configure the cluster to be single node using one SSD-backed storage node.
    iii. Launch the cluster into an Amazon VPC using the appropriate security group.
    iv. Using SQL Workbench on your local computer, connect to the new cluster.
    v. Create a new table and load data using the COPY command.

2. Delete the cluster, VPC and Security Group.