**AWS Database Design**

**Database types**

* **Database Types**: AWS offers various types of databases, including relational databases (like Amazon RDS), NoSQL databases (like DynamoDB), and data warehousing solutions (like Redshift). Each type is suited for different use cases based on data structure and query requirements.
* **Database Hosting Methods**: AWS provides multiple hosting options, such as managed services (e.g., Amazon RDS) and self-managed databases on EC2 instances. Managed services handle maintenance tasks like backups and patching, while self-managed options offer more control.
* **High-Availability Solutions**: AWS ensures high availability through features like Multi-AZ deployments for RDS, which replicate data across multiple availability zones to prevent downtime.
* **Scalability Solutions**: AWS databases can scale vertically (by increasing instance size) or horizontally (by adding more instances). Services like Aurora and DynamoDB offer automatic scaling to handle varying workloads.
* **Database Security**: AWS databases include built-in security features such as encryption at rest and in transit, IAM roles for access control, and VPC isolation to protect data.

**Relational databases**

* **Relational Database Structure**: Relational databases store data in tables that are related to each other. For example, you might have a customer table, an order table, and a product table. These tables are connected through unique identifiers like customer IDs and order IDs.
* **Terminology**:
  + **Rows (Tuples)**: Each row in a table represents a single record.
  + **Columns (Attributes/Properties)**: Columns represent the data fields for each record.
  + **Tables (Relations/Entities/Objects)**: Tables store data about a specific entity, like customers or orders.
  + **Views**: A view is a saved SQL query that presents data from multiple tables in a specific layout.
* **Keys**:
  + **Primary Key**: A unique identifier for each record in a table.
  + **Foreign Key**: A reference to a primary key in another table, used to link tables together.
* **Normalization**: This is the process of organizing data to reduce redundancy and improve data integrity. It involves dividing data into multiple tables and defining relationships between them. Higher normalization levels (like 3NF or 4NF) can make reads slower but writes faster.
* **SQL (Structured Query Language)**: SQL is used to interact with relational databases. It allows you to create, read, update, and delete data using queries.

These concepts help in understanding how relational databases are structured and how they function, which is crucial for designing efficient database solutions in AWS.

Top of Form

Bottom of Form

**Database hosting methods**

* **Instance-Based Hosting**:
  + **Setup**: You launch an EC2 instance and install the database service on it. This gives you full control over the database environment.
  + **Security**: You need to manage security groups to ensure the correct ports are open for database connections.
  + **Control**: You have complete control over the database version and configurations, but this also means you handle updates and performance management manually.
* **Service-Based Hosting**:
  + **Setup**: You simply launch the database service (e.g., DynamoDB) and connect to it. AWS manages the underlying infrastructure.
  + **Security**: AWS handles most of the security configurations, making it easier to manage.
  + **Control**: You have less control over the database environment, but AWS provides automatic updates and performance management.
* **Comparison**:
  + **Instance-Based**: Offers complete control but requires more effort to manage.
  + **Service-Based**: Easier to set up and manage, but with less control over specific configurations.

These concepts highlight the trade-offs between having full control over your database environment and the convenience of a managed service. In your field of software development, choosing the right hosting method depends on your specific needs for control, performance, and ease of management.

Top of Form

Bottom of Form

**High-availability solutions**

* **High Availability**: Ensuring that your database remains accessible and operational, even in the event of failures.
* **Clustering**:
  + **Definition**: Multiple servers (instances) work together as a single system.
  + **Redundancy**: Each server in the cluster has a copy of the database, ensuring availability.
  + **Active-Active Cluster**: All servers are active and share the load, providing both performance and failover capabilities.
  + **Active-Passive Cluster**: One server is active, and the other is on standby, ready to take over if the active server fails.
* **Standby Servers**:
  + **Definition**: A secondary server that replicates the primary server's data.
  + **Manual Failover**: Unlike clustering, failover is not automatic and requires manual intervention.
  + **Cost-Effective**: The standby server can be less powerful, reducing costs.
* **Single-AZ vs. Multi-AZ Deployments**:
  + **Single-AZ**: One instance in a single availability zone. Suitable for small to medium-sized businesses with lower fault tolerance needs.
  + **Multi-AZ**: Multiple instances across different availability zones within the same region. Provides fault tolerance and improved performance but at a higher cost.

These concepts help ensure that your databases remain available and resilient to failures, which is crucial in maintaining uninterrupted service.

**Scalability solutions**

* **Scalability**: The ability to increase your database's capacity to handle more data and operations. This can involve scaling storage, processing power, or network throughput.
* **Instance Scaling**:
  + **Vertical Scaling**: Increasing the size of your instance, such as moving from an instance with 32GB of memory to one with 128GB.
  + **Auto-Scaling**: Automatically adjusting the number of instances based on demand. Note that auto-scaling is not supported in RDS but can be scripted using AWS CLI commands.
* **Read Replicas**:
  + **Definition**: A read-only copy of your database that helps offload read traffic from the main database.
  + **Usage**: Useful for scenarios where many users need to read data, such as generating reports, without affecting the performance of write operations.
  + **Geographic Distribution**: You can have read replicas in different regions to improve performance for users in various locations.
* **Scripting for Automation**:
  + **Batch Jobs**: Using AWS CLI commands in batch processes (e.g., cron jobs on Linux or scheduled tasks on Windows) to automate scaling based on performance thresholds.

These concepts are essential for ensuring that your AWS databases can handle increased loads efficiently.

**Database security**

* **Encryption**:
  + **At-Rest Encryption**: Encrypts data stored on the disk. This protects data if someone physically steals the storage device. In AWS, this must be enabled when creating the database.
  + **In-Transit Encryption**: Encrypts data as it moves between your database and other services or users, protecting it from interception.
* **Permissions**:
  + **IAM Permissions**: Use AWS Identity and Access Management (IAM) to control who can manage databases. Only grant necessary permissions to minimize security risks.
  + **Database Permissions**: Control who can access and modify data within the database itself. Use roles and permissions to restrict access based on the principle of least privilege.
* **SQL Injection Attacks**:
  + **Definition**: A type of attack where malicious SQL code is inserted into queries to manipulate the database.
  + **Prevention**: Ensure developers write secure code that validates and sanitizes user inputs to prevent these attacks.

These concepts are crucial for maintaining the security of your AWS databases.

**Aurora**

* **Aurora Overview**:
  + **Relational Database**: Aurora is a relational database created by Amazon, compatible with MySQL.
  + **Optimized for OLTP**: Designed for online transaction processing, meaning it handles fast writes and updates efficiently.
* **Performance**:
  + **Improved Performance**: Offers up to five times better performance than standard MySQL.
  + **Scalability**: Starts at 10GB and scales in 10GB increments up to 64TB.
* **High Availability**:
  + **Multiple Copies**: By default, Aurora maintains six copies of your data across three availability zones.
  + **Fault Tolerance**: Can continue to write even if two copies are lost and read even if three copies are lost.
* **Replicas**:
  + **Aurora Replicas**: Supports up to 15 replicas with automatic failover.
  + **MySQL Read Replicas**: Supports up to five MySQL read replicas, but without automatic failover.
* **Compute Resources**:
  + **Virtual CPUs**: Supports up to 32 virtual CPUs.
  + **Memory**: Supports up to 244 GiB of RAM.

These features make Aurora a powerful and scalable database solution suitable for high-performance applications.

**Redshift**

* **OLAP Database**:
  + **Definition**: Redshift is an Online Analytical Processing (OLAP) database designed for fast read operations and data analysis.
  + **Data Warehouse**: It aggregates data from multiple sources into a central repository for analysis.
* **Cost Efficiency**:
  + **Pricing**: Starts at $0.25 per hour or $1,000 per terabyte per year, which is cost-effective compared to building and maintaining your own data warehouse.
* **Scalability**:
  + **Single Node**: Supports up to 160GB.
  + **Multi-Node**: Uses a leader node for query management and compute nodes for data storage and processing.
* **Performance**:
  + **Columnar Data Store**: Optimized for fast sequential reads.
  + **Data Compression**: Speeds up data retrieval by reducing the amount of data read from disk.
  + **Massively Parallel Processing (MPP)**: Allows multiple processors to work on queries simultaneously.
* **Security**:
  + **SSL Transit Encryption**: Protects data in transit.
  + **At-Rest Encryption**: Uses AES-256 encryption for stored data.
* **Availability**:
  + **Snapshots**: Can be restored to different availability zones for increased availability.
* **Quickstarts**:
  + **Deployment**: AWS offers quickstart guides to help deploy Redshift efficiently.

These features make Redshift a powerful and cost-effective solution for data warehousing and analytics.

Top of Form

**DynamoDB**

* **DynamoDB Overview**: DynamoDB is a NoSQL database service provided by AWS. Unlike traditional relational databases, NoSQL databases like DynamoDB are designed to handle large volumes of data and provide high performance for specific use cases.
* **Key Features**:
  + **Millisecond Latency**: DynamoDB offers very fast read and write operations, even at large scales. This is achieved by storing data on SSDs and distributing it across multiple data centers.
  + **Consistency Types**: There are two read consistency types in DynamoDB:
    - **Eventual Consistent Reads**: These may have a slight delay in synchronization across data copies.
    - **Strongly Consistent Reads**: These ensure data is synchronized within milliseconds.
* **Use Cases**: DynamoDB is ideal for applications that require quick access to small amounts of data, such as mobile apps and online stores. For example, it can quickly determine which advertisements to show based on user interactions.
* **Cost**: Storage starts at $0.25 per gigabyte per month, with additional costs for read and write throughput.

Understanding these concepts can help you appreciate how DynamoDB can be used effectively in various scenarios, especially in the software development industry.

**Data security policies**

* **Compliance vs. Policies**:
  + **Compliance**: External requirements imposed by government regulations, industry standards, or partner agreements. It defines what you need to do.
  + **Policies**: Internal guidelines created to meet compliance requirements. They define how you will achieve compliance within your organization.
* **Data Privacy**:
  + **Protecting Personal Information**: Ensure the privacy of individuals' data you handle.
  + **Key Rules**:
    - Do what is required by regulations.
    - Fulfill promises made in privacy statements.
    - Report incidents quickly and accurately.
* **Cloud Data Security Concerns**:
  + **External Network**: Data is stored and managed on someone else's network, accessed through the internet.
  + **Concentration of Data**: Large volumes of data in one place can increase vulnerability.
  + **Improper Configuration**: Misconfigured services can introduce security risks.
  + **Remote Access**: Data accessed by remote users needs careful security management.
* **Solutions**:
  + **AWS Tools**: Use AWS tools like Control Tower and Security Hub to monitor and enforce compliance.
  + **Education**: Ensure admins and users are educated about proper security practices.
  + **Monitoring**: Continuously monitor for compliance and enforce policies using AWS tools.

These concepts are crucial for maintaining robust data security policies in the cloud.

**AWS Database Deployment**

**DynamoDB tables lab**

* **DynamoDB Overview**:
  + DynamoDB is a fully managed NoSQL database service provided by AWS, designed for fast, small transactions, such as those needed for mobile apps or websites.
* **Creating Tables**:
  + **Create Table**: Instead of creating a full database, you create tables within DynamoDB. These tables are hosted and managed by AWS.
  + **Table Name**: You need to give your table a name (e.g., "My fast table").
  + **Partition Key**: This acts as the primary key for the table, often a number (e.g., UserID).
* **Default Settings**:
  + **Provisioned Capacity**: Default settings include five reads and five writes per second.
  + **Basic Alarms**: Basic monitoring features are included, such as alarms for 80% usage thresholds.
* **Customization**:
  + You can customize settings like read/write capacity, auto-scaling, and encryption if needed.
* **Adding Items**:
  + **User ID**: When adding items, you start with the UserID.
  + **Appending Fields**: You can add more fields (e.g., Boolean values like "Active" which can be true or false).
* **Flexibility**:
  + **Dynamic Columns**: You can add new columns at any time without modifying the table structure, making it very flexible for evolving applications.
* **Access Control**:
  + **Policies**: Access control is managed through policies, ensuring secure access to the data.

These points should help you understand the process and features of creating and managing tables in DynamoDB.

**MySQL lab**

* **Why MySQL?**: MySQL is a popular database that is free to use, making it a good choice for learning and practice. Unlike Aurora DB, which has costs associated with it, MySQL can be implemented without incurring charges.
* **Creating a MySQL Database**:
  + Go to the RDS (Relational Database Service) section in AWS.
  + Click on "Create database" and select MySQL from the options.
  + Choose the "General public license" for the license model.
  + Select the version of MySQL you need. For example, MySQL 8.0.11.
  + Configure the instance settings, such as storage type (SSD) and storage allocation (20 GB in this case).
* **Advanced Settings**:
  + **Virtual Private Cloud (VPC)**: Decide which VPC your database will be in. This ensures it is in the right place within your virtual infrastructure.
  + **Public Accessibility**: Determine if the database should be accessible from outside the VPC. For example, if you have a web server that needs to access the database, you might need to make it publicly accessible.
  + **Security Groups**: These act like firewalls, controlling what can access your database. Let AWS create a new security group for you to ensure the correct ports are open.
  + **Backup and Monitoring**: Set up automated backups and decide on the retention period (e.g., 7 days). You can also enable enhanced monitoring for additional insights, though this may incur extra costs.
* **Final Steps**: After configuring all settings, click "Create database." AWS will manage the database instances for you, ensuring they run smoothly and efficiently.

**Configuration lab**

1. **Database Instance Management**:
   * AWS RDS manages your database instances, not EC2. You won't find your database instance in the EC2 dashboard; instead, you manage it through the RDS dashboard.
2. **Modifying Database Configurations**:
   * You can modify various parameters of your database instance, such as enabling multi-AZ deployment for high availability or adjusting storage capacity. Be mindful that changes like converting to multi-AZ can impact performance temporarily.
3. **Creating Read Replicas**:
   * Read replicas help improve performance by offloading read operations to a replica database, while the master database handles write operations. This setup enhances scalability and ensures that read operations do not affect write performance.
4. **DNS vs. IP for Connections**:
   * Always use the DNS name for database connections instead of the IP address. This ensures that if the primary instance fails, the DNS will redirect to the failover instance automatically.

These concepts are crucial for managing and optimizing your AWS database instances effectively.

**Backups lab**

1. **Simplicity of AWS Backups**:
   * AWS automates the backup process, making it simpler compared to traditional methods that require extensive planning. You only need to set the parameters, and AWS handles the rest.
2. **Backup Parameters**:
   * The key parameter is the retention period, which determines how long automated backups are kept. Setting this to zero disables backups, which is generally not recommended unless you have a specific reason.
3. **Performance Impact**:
   * Backups can affect database performance. For instance, if you have a database used solely for analytics and can be rebuilt easily, you might choose to disable backups to avoid performance hits.
4. **Retention Period**:
   * The maximum retention period for AWS automated backups is 35 days. This is important for point-in-time recovery, allowing you to restore your database to a specific point within that timeframe.

These concepts are crucial for effectively managing your AWS database backups and ensuring data integrity and availability.

Top of Form

Bottom of Form

Top of Form

**Restore lab**

* **Restoring Databases**:
  + **Automated Backups**: You can restore your database from automated backups created by AWS.
  + **Snapshots**: Another option is to restore from snapshots, which will be covered in the next video.
* **Restoration Process**:
  + **New Instance**: When restoring, AWS creates a new database instance rather than restoring to the existing one. This allows you to make changes, such as enabling encryption, during the restore process.
  + **Point-in-Time Restore**: You can restore your database to a specific point in time, which is useful if you need to recover from data corruption or accidental deletions.
* **Steps to Restore**:
  + Select your MySQL database instance.
  + Go to "Instance actions" and choose "Restore to point in time."
  + Choose the time you want to restore to, either the latest restorable time or a custom time.
  + Configure the new instance settings and launch the new DB instance.
* **Importance for Exam**:
  + Understanding the restore process is crucial for the AWS Certified Solutions Architect – Associate exam, as it ensures you can implement the right backup and restore procedures.

These points should help you understand the essential concepts of restoring databases in AWS RDS.

**Snapshot lab**

* **Snapshots Overview**:
  + Snapshots are point-in-time copies of your database, allowing you to create manual backups.
* **Creating Snapshots**:
  + You can create snapshots through the AWS graphical interface or command line.
  + To create a snapshot, select your database instance, go to "Instance actions," and choose "Take snapshot."
  + Naming convention: Use a clear naming format like mysqldb/ss/yyyy-mm-dd to easily identify when the snapshot was created.
* **Restoring from Snapshots**:
  + Snapshots can be used to restore your database to a specific point in time.
  + Restoring from a snapshot creates a new database instance based on the snapshot.
* **Additional Options**:
  + **Copying Snapshots**: You can copy snapshots to different regions or accounts.
  + **Sharing Snapshots**: Snapshots can be shared with other AWS accounts, either privately or publicly.
  + **Deleting Snapshots**: You can delete old snapshots to manage storage and keep your snapshot list organized.

These points should help you understand the process and benefits of using snapshots in AWS.

**Monitoring lab**

* **Monitoring Basics**:
  + AWS provides basic monitoring for any RDS deployment, including metrics like CPU utilization, database connections, free storage space, free memory, write IOPS, and read IOPS.
* **Performance Troubleshooting**:
  + By monitoring these metrics, you can identify performance issues. For example, high CPU utilization and low free memory during peak times might indicate the need for a read replica or clustering to improve performance.
* **Read Replicas**:
  + Read replicas can offload read operations from the main database, improving performance for write operations and batch processes.
* **Clustering**:
  + Clustering involves using multiple databases to handle write operations, which can significantly enhance performance during high-demand periods.
* **Advanced Monitoring**:
  + For more detailed insights, AWS offers Performance Insights, a paid service that provides enhanced performance metrics. This is useful for large databases with many users.
* **Architect's Perspective**:
  + As an architect, it's crucial to design effective database implementation plans, backup plans, recovery plans, and monitoring solutions that meet the needs of your AWS deployment.

These points should help you understand the essential concepts of monitoring databases in AWS.

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Bottom of Form

Top of Form

Bottom of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form