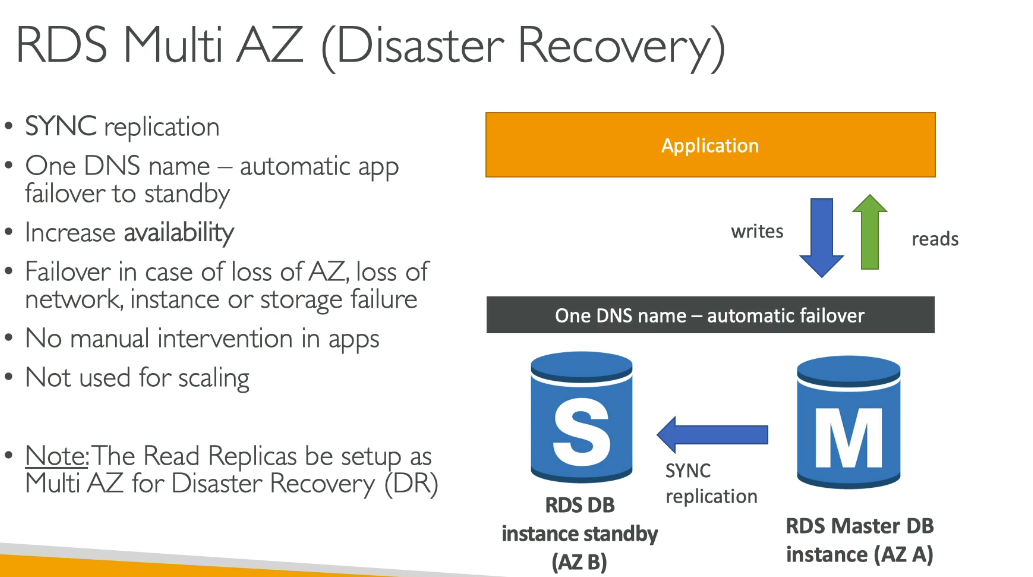
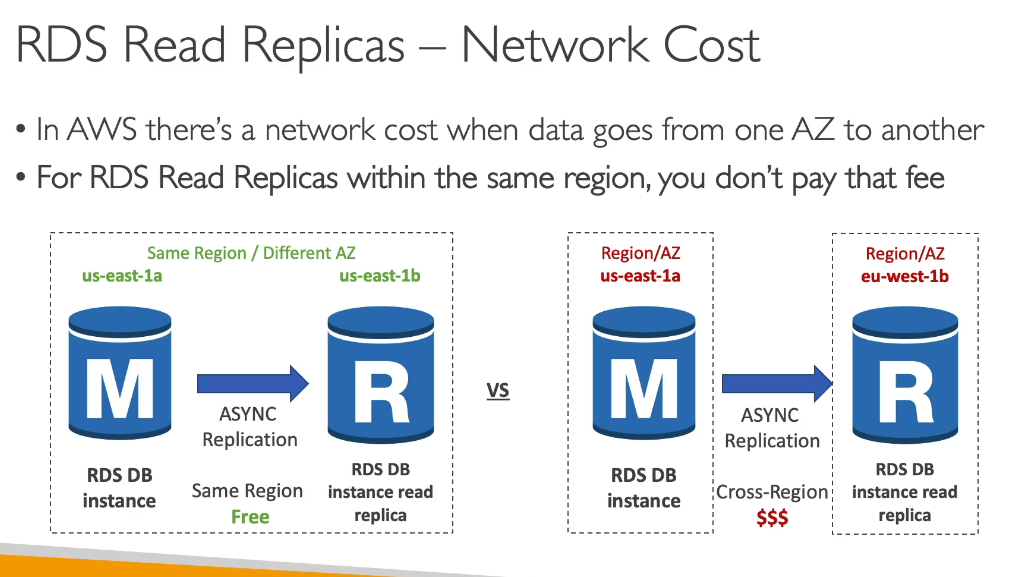
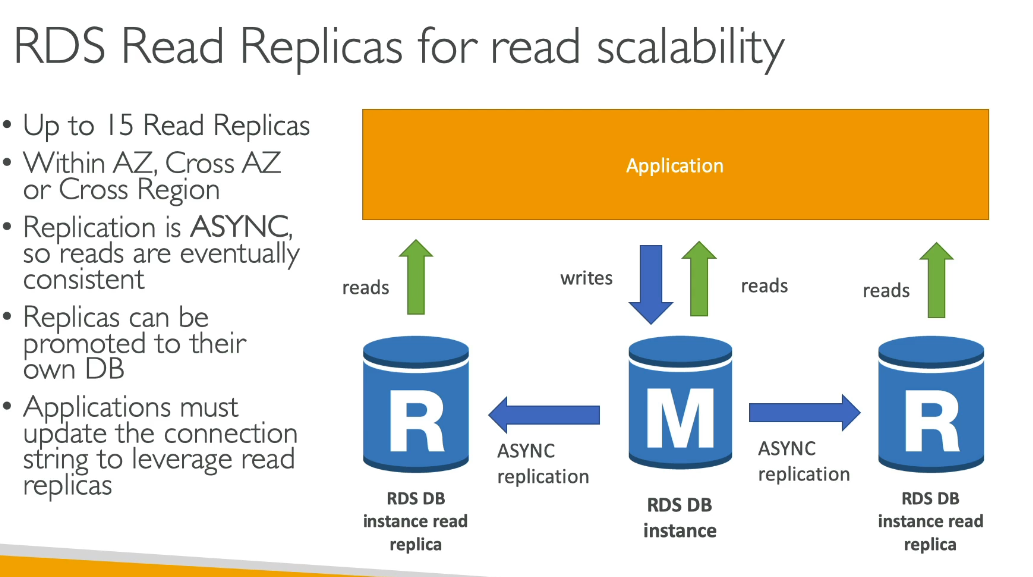
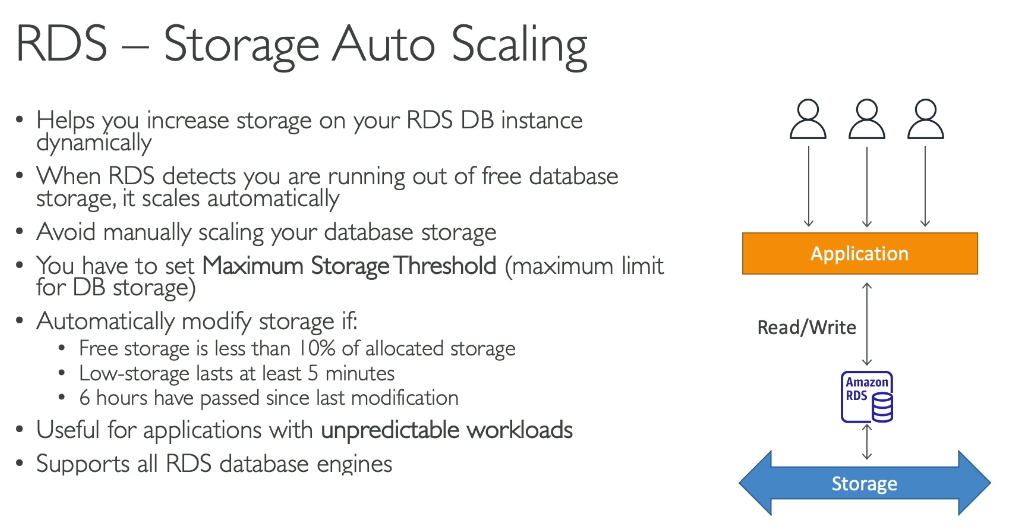
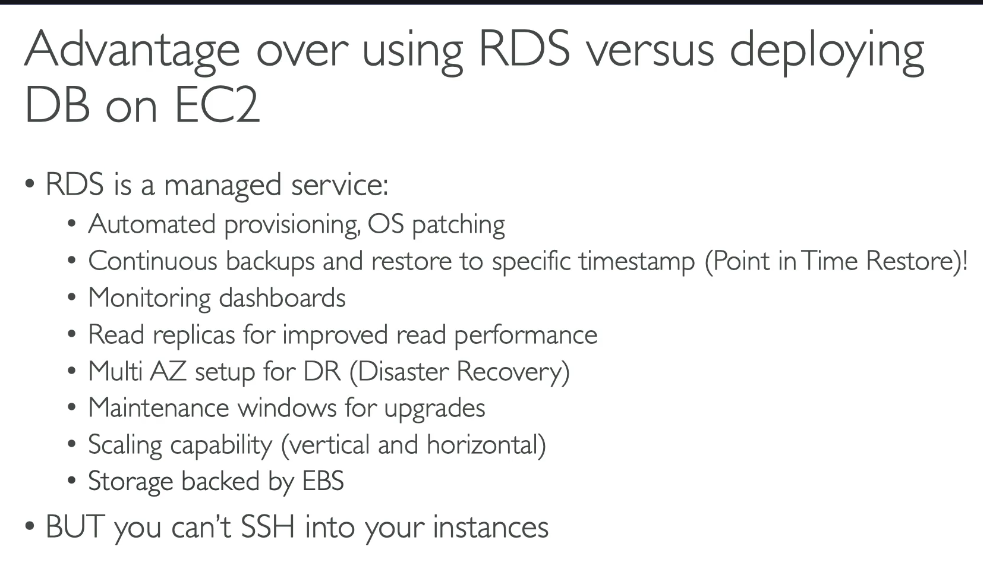
**Overview:**



**Example for RDS Multi-AZ:**

🧠 Think of Multi-AZ like this:

You have two databases:

Primary DB (Master) → The one your app writes to

Standby DB → A hidden backup copy in another Availability Zone

AWS keeps these two DBs perfectly in sync using synchronous replication.

🏗️ What is synchronous replication (in this case)?

When your app writes something to the primary DB:

Primary writes the data

Standby writes the same data

Only after both succeed, AWS returns “write successful”

So both DBs always have the same data.

This means zero data loss if the primary fails.

🌩️ What happens during failure?

Let’s say AZ A goes down or your primary instance crashes.

AWS does this automatically:

Promotes the standby in AZ B to become the new primary

Updates the same DNS endpoint to point to the new primary

Your application reconnects automatically (no code changes)

You don’t manually switch anything — AWS handles the whole failover.

**Note:**

### **Aurora**

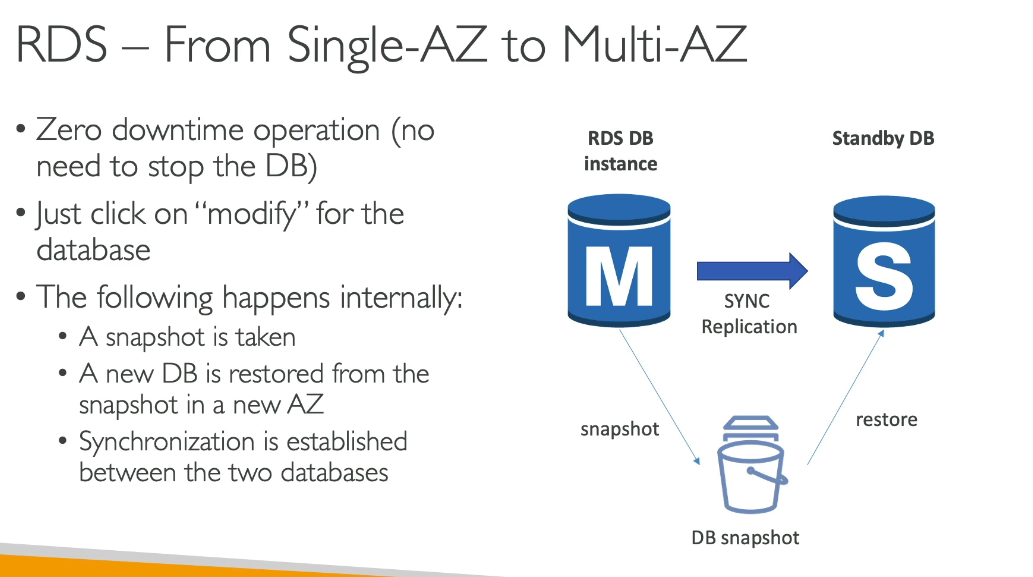
* Uses **synchronous** replication across 3 AZs for storage.
* Reader instances use a proprietary fast commit protocol (quasi-sync).

### **RDS Multi-AZ (MySQL/Postgres)**

* **Synchronous** replication to standby.
* Zero data loss failover.

### **Read replicas (RDS or Aurora)**

* **Asynchronous** replication.
* Can lag behind.

  
**Question:**

Instead of enabling Multi-AZ on an RDS instance, can I use a Multi-AZ read replica and simply promote it to the primary if the main instance or Availability Zone fails?  
  
**Answer:**

**Yes, you *can* do it — but it’s a shitty idea in production and absolutely NOT equivalent to Multi-AZ.**

Let me explain this in a brutally clear way so the reasoning stays in your head forever.

# **🔥 Multi-AZ vs Read Replica Failover — NOT the same thing**

## **🟩 RDS Multi-AZ (the correct HA method):**

* Synchronous replication
* Zero data loss
* Automatic failover
* Automatic DNS switch
* App reconnects instantly
* No intervention required

AWS chooses Multi-AZ for HA for a damn good reason.

## **🟥 Read Replica Promotion (your idea):**

* **Asynchronous** replication → meaning **data loss WILL happen**
* **NO automatic failover**
* **NO automatic DNS switch**
* **YOU must manually promote the replica**

Replication might be minutes behind

* Entire application experiences downtime
* You risk losing the last commits/transactions

AWS never recommends this for HA because it’s unreliable as hell.

# **What happens if the AZ fails?**

Read replica in another AZ still won’t help automatically.

You must manually:

1. Promote replica
2. Change DB endpoint
3. Update app configs or restart your app
4. Hope replication was up-to-date (unlikely)

Meanwhile your system is DOWN.

Multi-AZ handles all of this automatically.

**Types of Log exports in RDS:**

**Audit Log**

* Use: Records database activity for compliance and security monitoring
* Use Case: Track who accessed what data and when, essential for regulatory compliance (HIPAA, PCI-DSS) and security audits. Helps detect unauthorized access attempts.

**Error Log**

* Use: Captures database errors, warnings, and startup/shutdown information
* Use Case: Troubleshoot database crashes, connection issues, or configuration problems. Generated by default and essential for diagnosing operational issues.

**General Log**

* Use: Records all client connections and SQL statements executed
* Use Case: Debug application issues by seeing exact queries being run, identify problematic queries, or understand application behavior. Note: Can impact performance due to high volume.

**iam-db-auth-error Log**

* Use: Tracks errors related to IAM database authentication
* Use Case: Troubleshoot issues when using IAM roles to authenticate to the database instead of traditional passwords. Helps identify permission or configuration problems.

**Slow Query Log**

* Use: Identifies queries that exceed a specified execution time threshold
* Use Case: Performance optimization - find and optimize poorly performing queries that slow down your application. Essential for database tuning and improving response times.

These logs are retained for 24 hours and rotated hourly in RDS.

**Database Authentication Options:**

1. Password Authentication
2. Password and IAM database authentication
3. Password and Kerberos Authentication

**Kerberos Authentication:**

Kerberos authentication is a network authentication protocol that allows you to authenticate database users using Microsoft Active Directory instead of traditional username-password credentials stored in the database.

How It Works:

* Uses tickets and symmetric-key cryptography to eliminate transmitting passwords over the network
* The Kerberos ticket is a certificate issued by an authentication server, encrypted using the server key
* Built into Microsoft Active Directory to authenticate users to network resources

Key Benefits:

* Single Sign-On (SSO): Users authenticate once with Active Directory and can access multiple databases without re-entering credentials
* Centralized Management: All user credentials are stored and managed in one place (Active Directory) rather than in each database
* Enhanced Security: No passwords transmitted over the network, reducing security risks
* Simplified Administration: DBAs don't need to manage individual database credentials

Supported RDS Engines:

* Amazon RDS for PostgreSQL
* Amazon RDS for Oracle
* Amazon RDS for SQL Server
* Aurora PostgreSQL-Compatible Edition When creating a database with Kerberos authentication, you configure external authentication that integrates with your Microsoft Active Directory, allowing users to access the database using their existing AD credentials.

**RDS Connection Types:**

There are several ways to connect to Amazon RDS instances:

1. Database Client Applications

* Use standard client tools like MySQL Workbench, pgAdmin, SQL Server Management Studio, or Oracle SQL Developer
* Configure connection with your DB instance endpoint, port, and credentials

2. Command-Line Tools

* Connect using CLI tools specific to your database engine (mysql, psql, sqlplus, etc.)
* Useful for scripting and automation

3. Application Code

* Connect programmatically using database drivers and connection strings in your application
* AWS provides optimized drivers for enhanced features like failover support

4. Amazon RDS Proxy

* Manage and pool database connections to improve scalability
* Allows applications to share connections efficiently
* Available for MariaDB, MySQL, PostgreSQL, and SQL Server

**RDS Left Side Bar Features:**

**Query Editor**

* Use: Run SQL queries directly in the AWS Console without external tools. Supports only for Aurora Serverless DB.
* Example: Quickly check table data or run ad-hoc queries without connecting via MySQL Workbench

**Performance Insights**

* Use: Monitor database performance and identify bottlenecks
* Example: Identify slow queries consuming CPU or find which queries are causing high database load

**Exports in Amazon S3**

* Use: Export RDS snapshots to S3 in Parquet format for analysis
* Example: Export production database snapshot to S3 for data analytics using Amazon Athena

**Snapshots**

* Use: Manual point-in-time backups of your database
* Example: Create a snapshot before major application deployment for quick rollback if needed

**Automated Backups**

* Use: Automatic daily backups with point-in-time recovery
* Example: Restore database to any point within your retention period (1-35 days) after accidental data deletion

**Reserved Instances**

* Use: Purchase 1 or 3-year commitments for significant cost savings (up to 69%)
* Example: Buy reserved capacity for production databases running 24/7

**Proxies**

* Use: Manage connection pooling for better scalability
* Example: Handle thousands of Lambda connections efficiently without overwhelming your database
* Think of it as a "middleman" between your application and database. Instead of your app directly connecting to the database every time, the proxy manages a pool of connections that can be reused. It's like a taxi stand - instead of calling a new taxi each time, you use taxis waiting at the stand, which is much faster and efficient.

**Parameter Groups**

* Use: Configure database engine settings
* Example: Adjust max\_connections, buffer sizes, or enable slow query logging
* These are configuration settings for your database engine. Think of it like adjusting settings on your phone - you can control things like memory allocation, connection limits, or logging behavior. For example, you might increase max\_connections from 100 to 500 to allow more users to connect simultaneously.

**Option Groups**

* Use: Enable additional database features
* Example: Enable Oracle Transparent Data Encryption or SQL Server audit features
* These add extra features to your database that aren't enabled by default. It's like adding optional accessories to a car - you might add Oracle's Transparent Data Encryption for security, or enable audit logging to track who accessed what data. Not all databases need these features, so they're optional add-ons.

**Custom Engine Versions**

* Use: Create and manage custom database engine versions
* Example: Apply custom patches or maintain specific engine configurations
* This lets you create your own customized version of a database engine. Imagine you need a specific patch or configuration that's unique to your company - you can create a "golden image" with your preferred setup and use it to create new databases consistently.

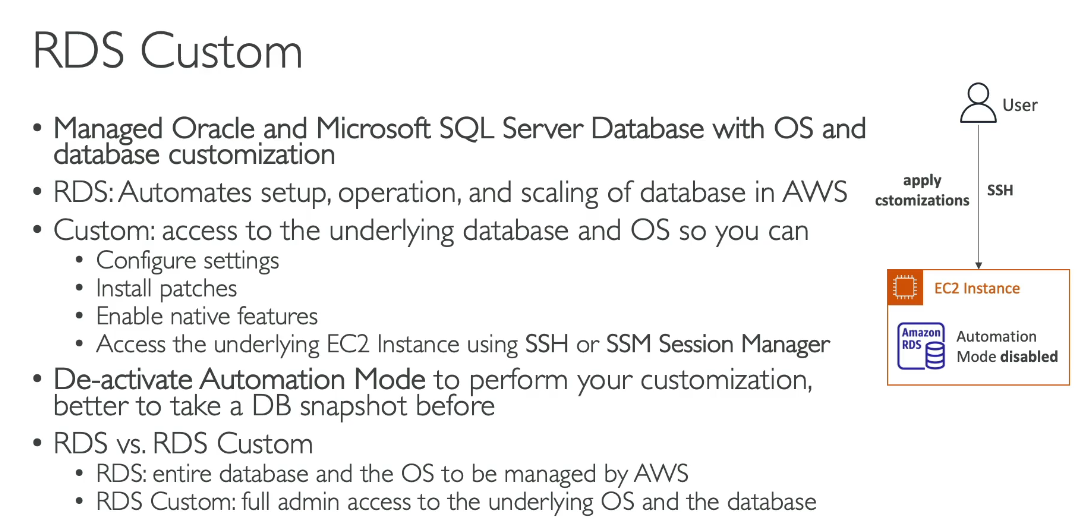
**Zero-ETL Integrations**

* Use: Automatically replicate data to Amazon Redshift or SageMaker for analytics
* Example: Real-time data replication from RDS MySQL to Redshift without ETL pipelines
* Extract, Transform, Load - the traditional process of copying data from one system to another for analysis. Zero-ETL means you skip this complex process. Your RDS data automatically replicates to analytics services like Amazon Redshift in near real-time, without writing code or managing pipelines. It's like having a live mirror of your data ready for analysis.

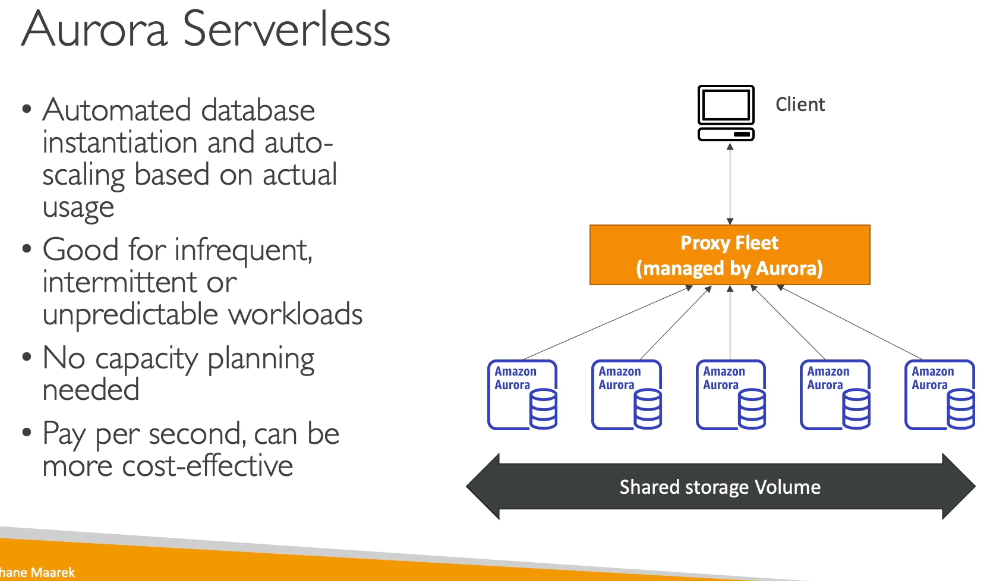
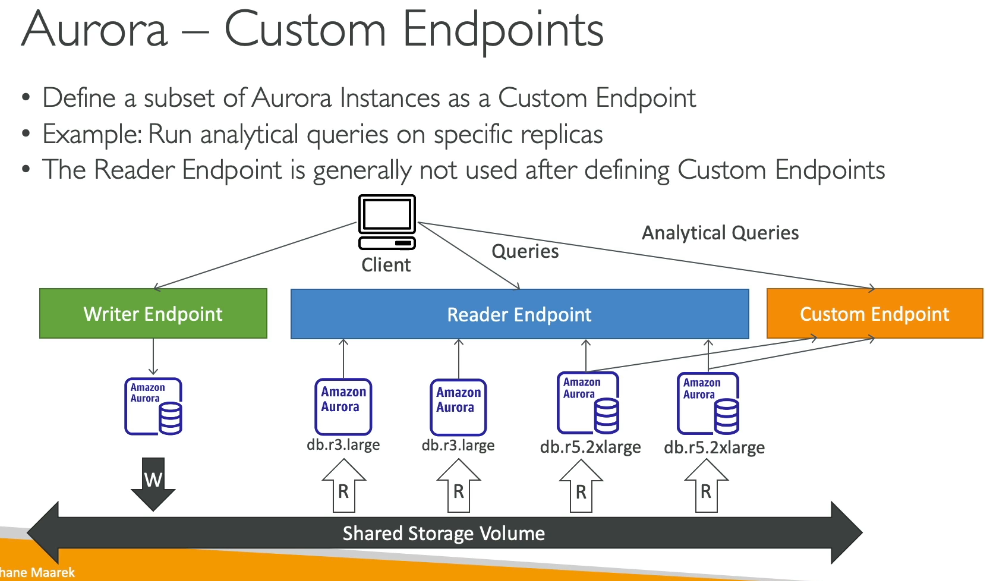
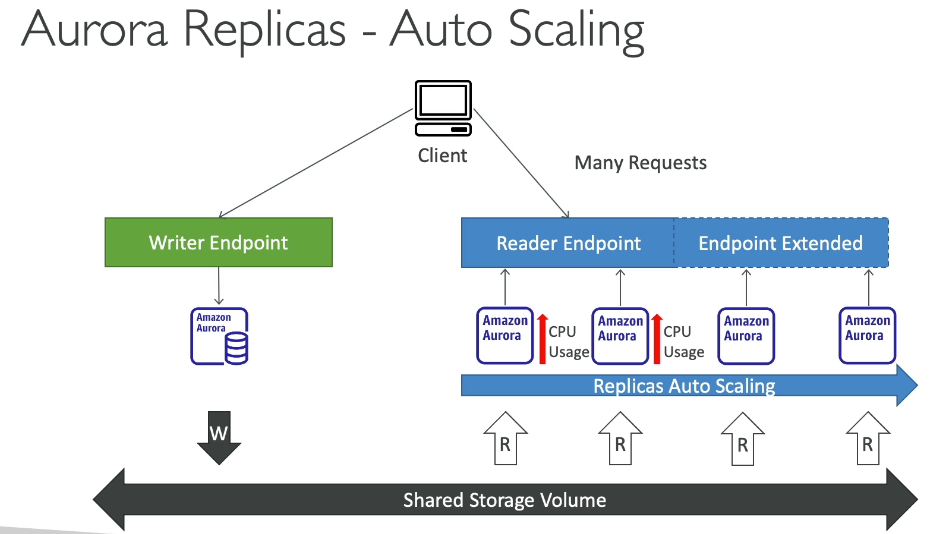
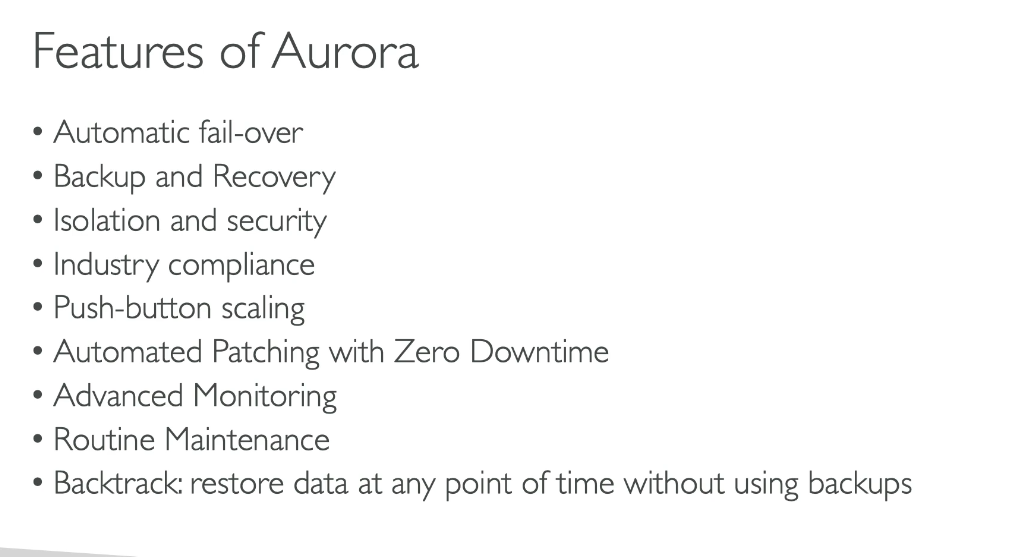
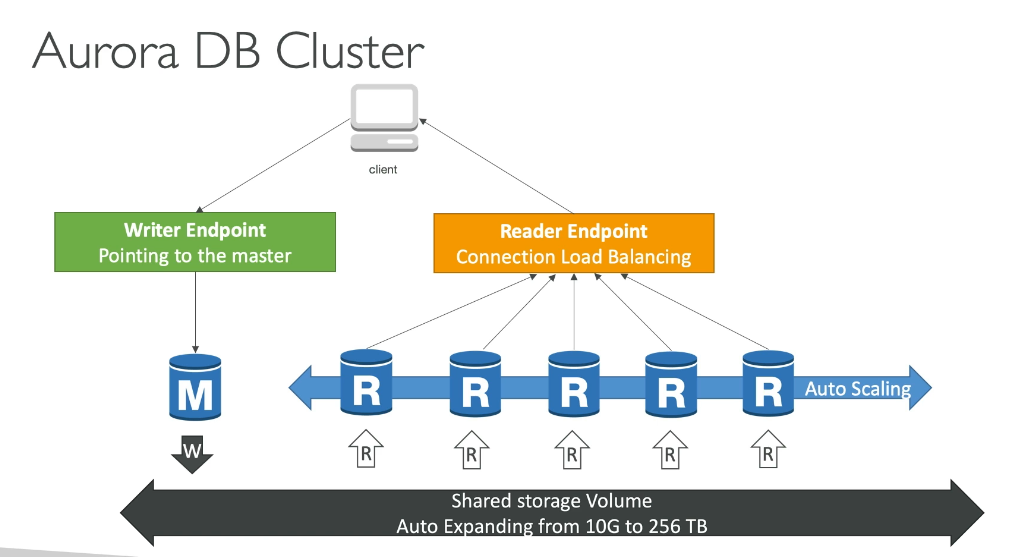
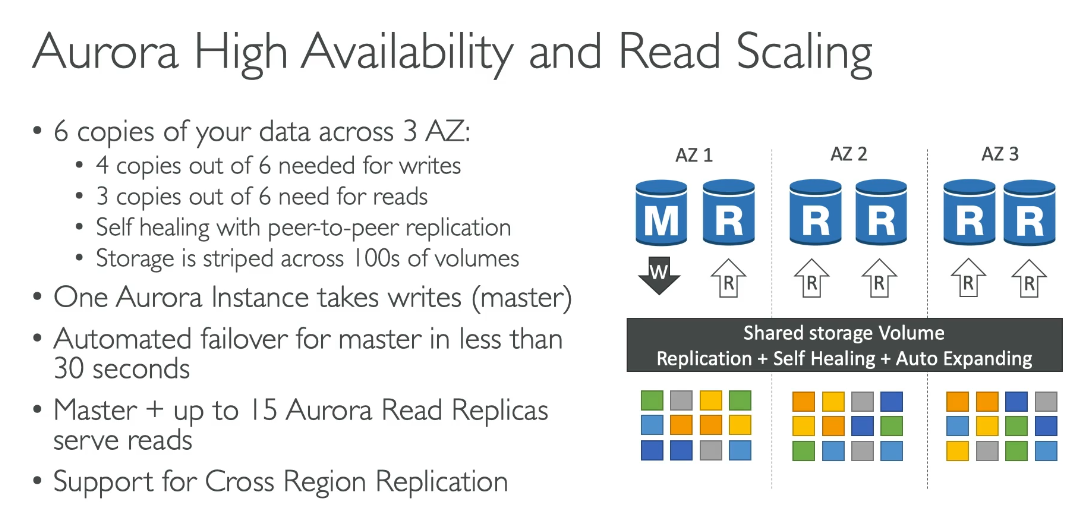
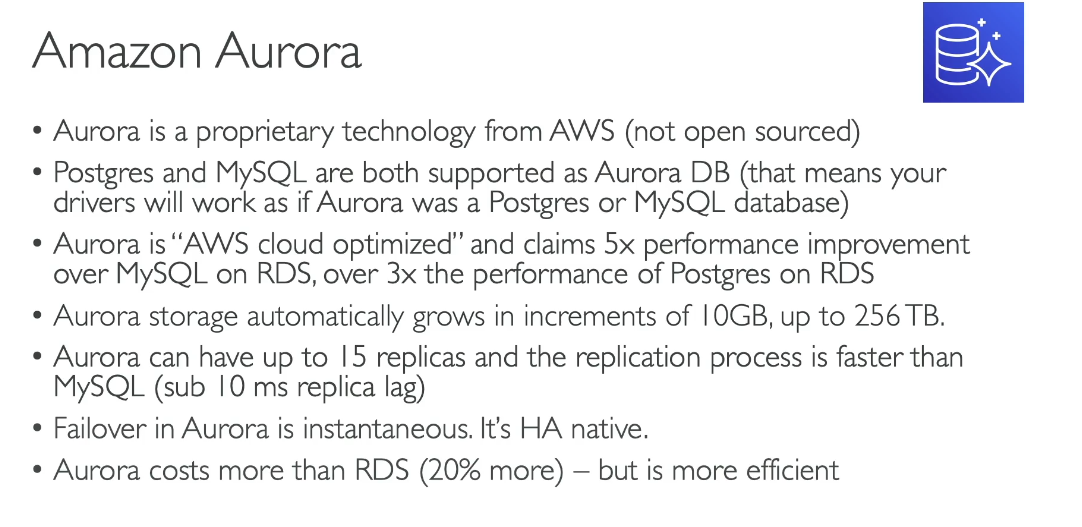
**Certificate Update**

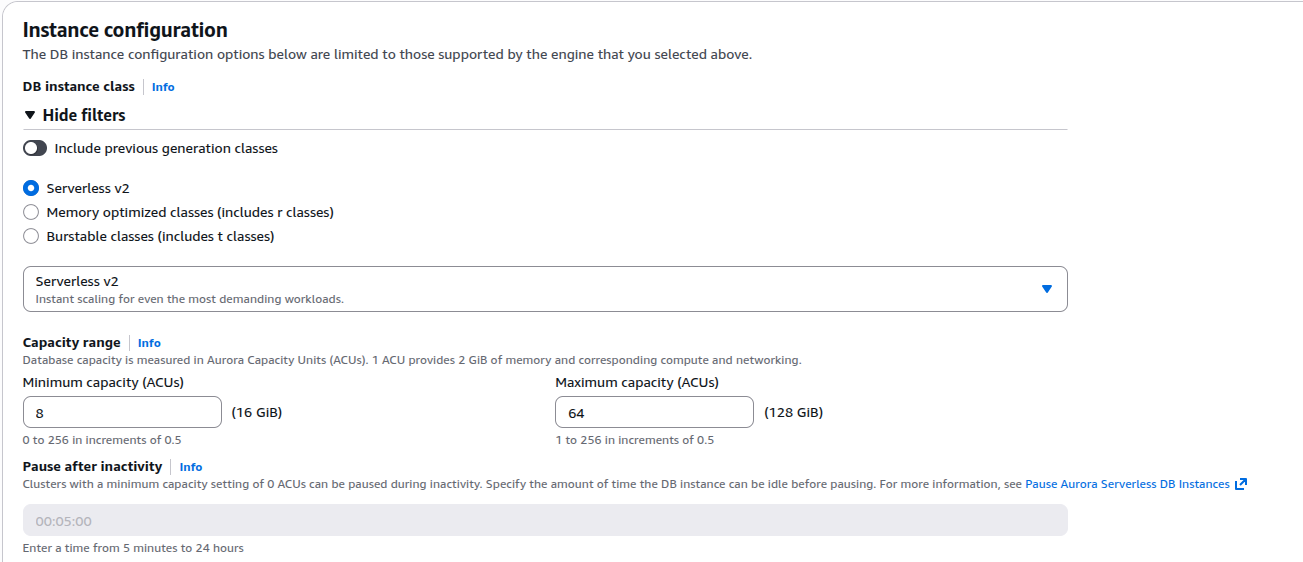
* Use: Manage SSL/TLS certificates for encrypted connections
* Example: Update expiring certificates to maintain secure database connections
* SSL/TLS certificates are like digital passports that encrypt the connection between your application and database. These certificates expire periodically (like a passport), so you need to update them to maintain secure connections. AWS provides tools to rotate these certificates without downtime.

---------------------------------------------------------------------------------------------------------------------

**RDS Custom: (Installing/using database in EC2 instance):**  


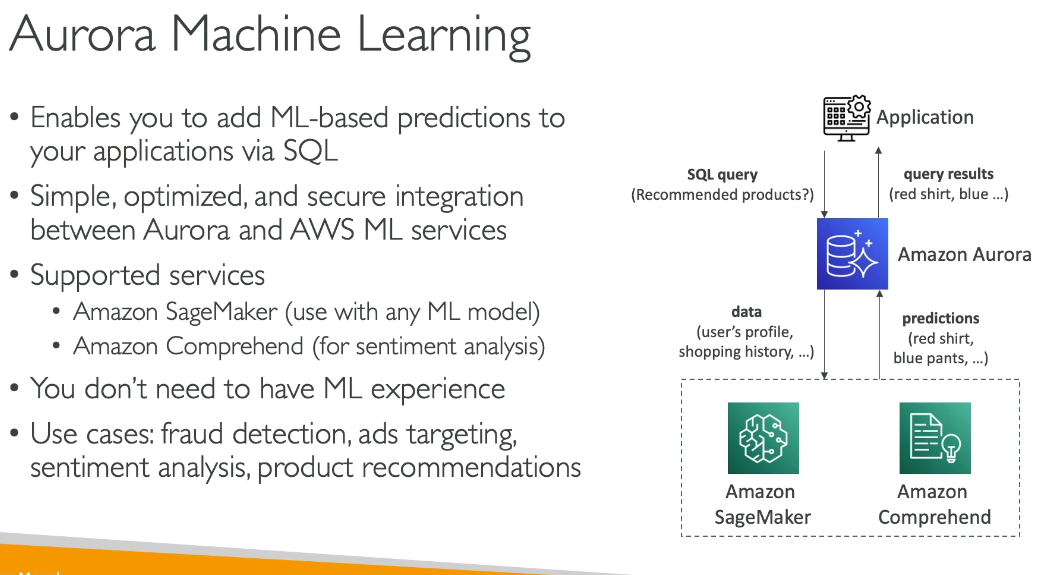
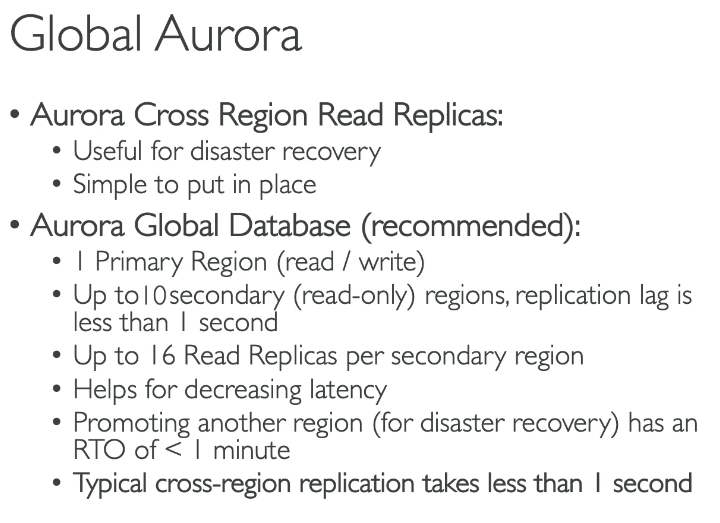
**Amazon Aurora:**

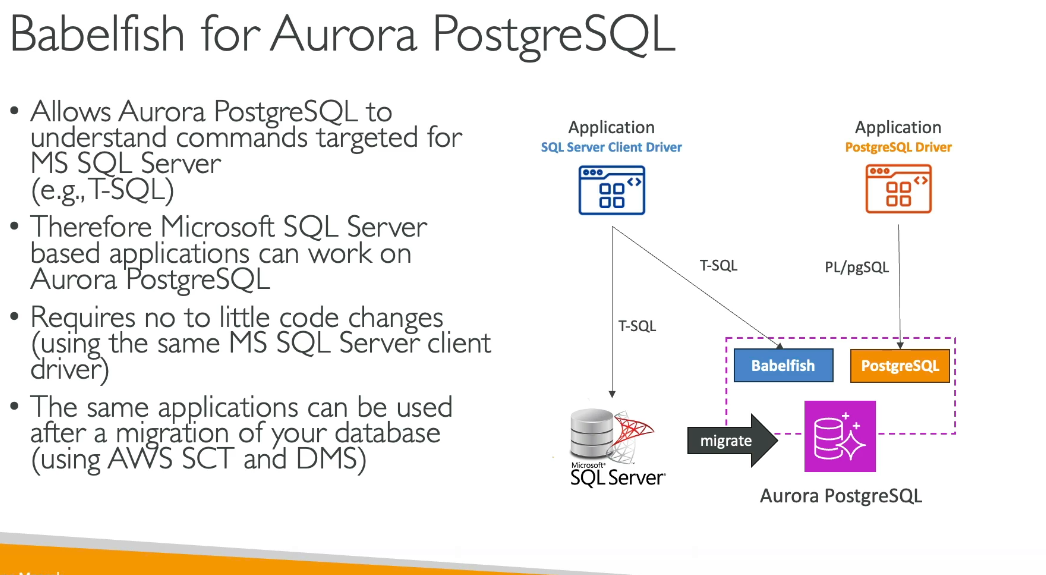




|  |  |  |
| --- | --- | --- |
| **Area** | **Aurora (Provisioned)** | **Aurora Serverless v2** |
| Capacity | Fixed instances | Auto-scaling ACUs |
| Scaling speed | Minutes | Seconds |
| Idle cost | Full price | Lower, not zero |
| Cold starts | No | No |
| Performance predictability | High | Medium-High |
| Ops effort | Medium | Low |
| Cost efficiency | Best at steady load | Best at spiky/unpredictable load |

**Aurora advance concepts:**





**Difference between AWS SCT and AWS DMS:**

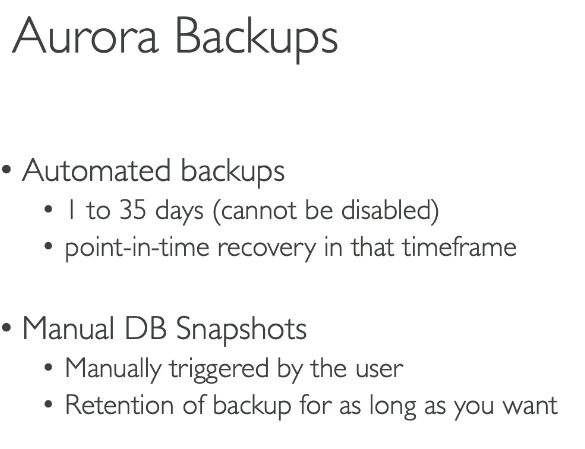
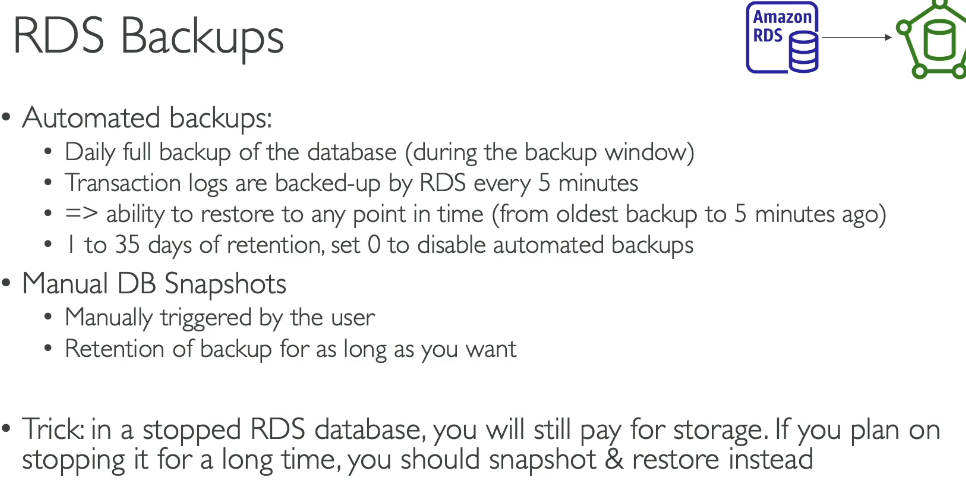
**AWS SCT** converts database schemas and code between different database engines, generates migration assessment reports, and is best for workloads over 10 TB.

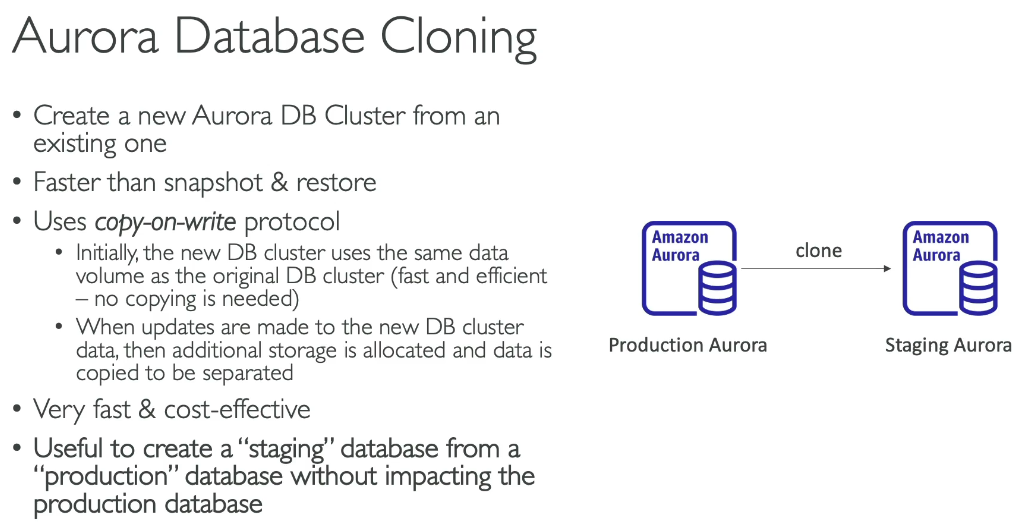
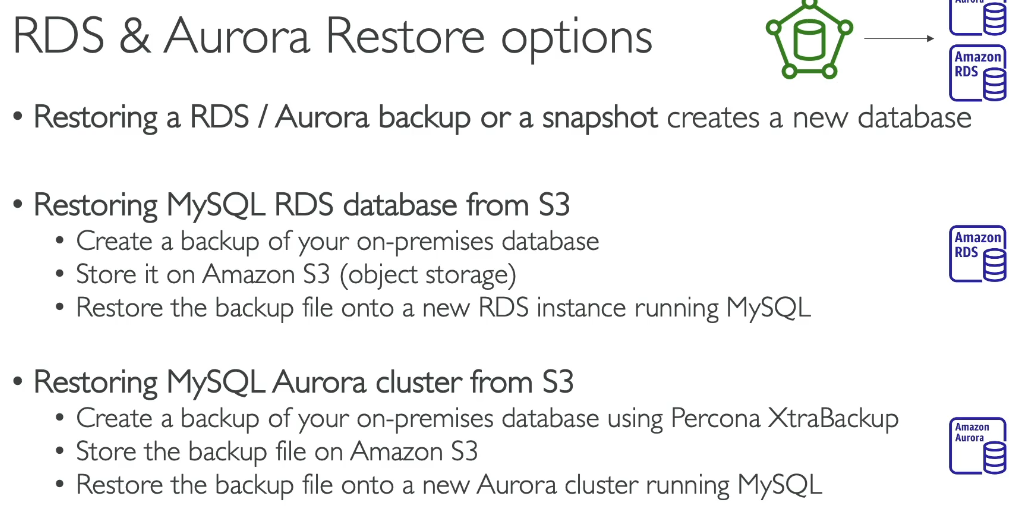
**AWS DMS** migrates the actual data between databases with minimal downtime, supports continuous replication, and is typically used for workloads under 10 TB.

**Together**: Use SCT first to convert schemas, then DMS to migrate data and maintain ongoing synchronization.

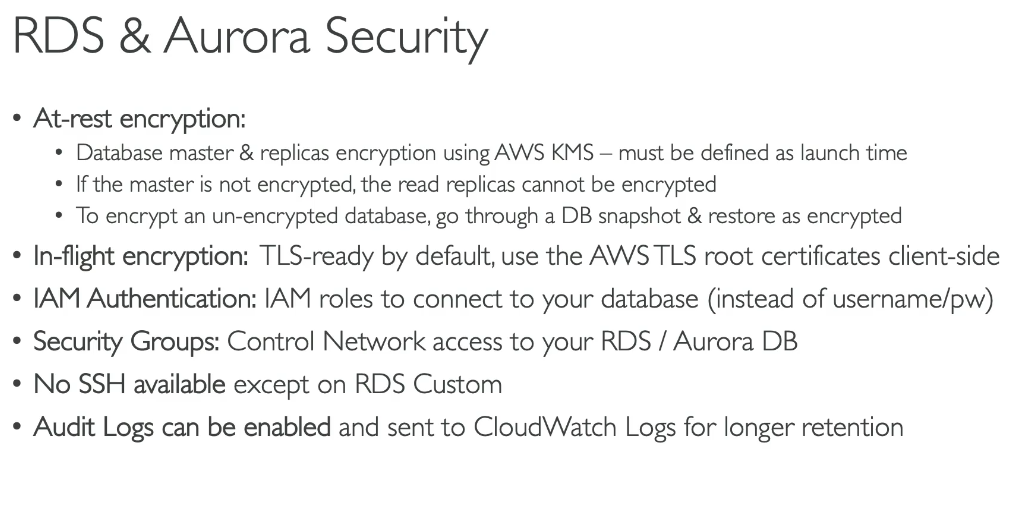
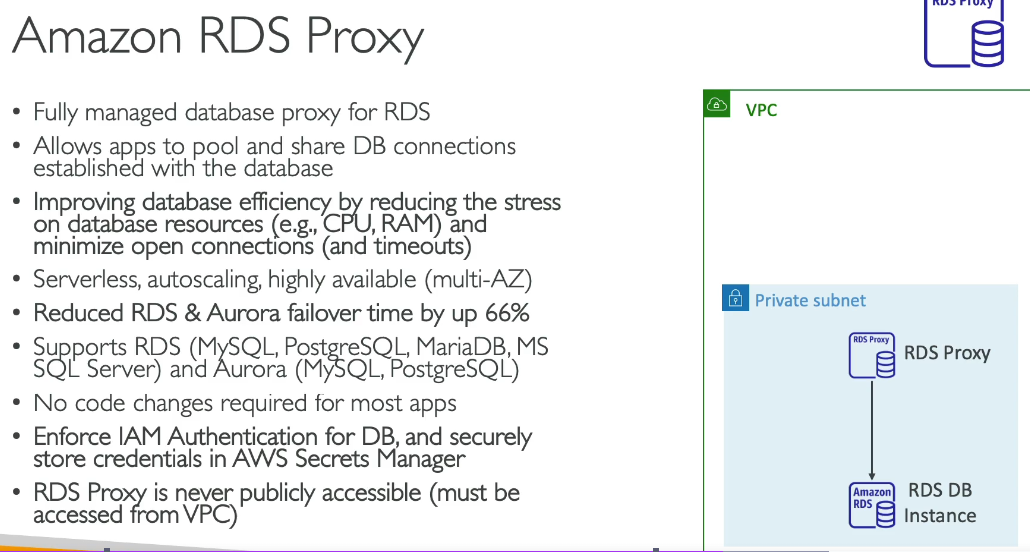
---------------------------------------------------------------------------------------------------------------------

**RDS & Aurora Backups and Monitoring:**





**Security:**

**SSH available for RDS Custom and even SSM too, the key pair will get stored in AWS Secret manager after when you create the RDS Custom database**  
  
RDS Proxy is available for:

* Amazon RDS for MariaDB
* Amazon RDS for MySQL
* Amazon RDS for PostgreSQL
* Amazon RDS for SQL Server

RDS Proxy is NOT available for:

* Amazon RDS for Oracle
* Amazon RDS for Db2

**Ports:**

List of Ports to be familiar with

Here's a list of **standard** ports you should see at least once. You shouldn't remember them (the exam will not test you on that), but **you should be able to differentiate between an Important (HTTPS - port 443) and a database port (PostgreSQL - port 5432)**

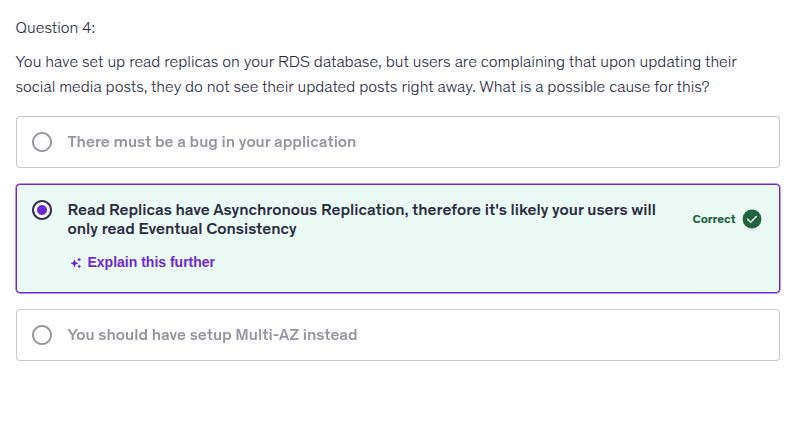
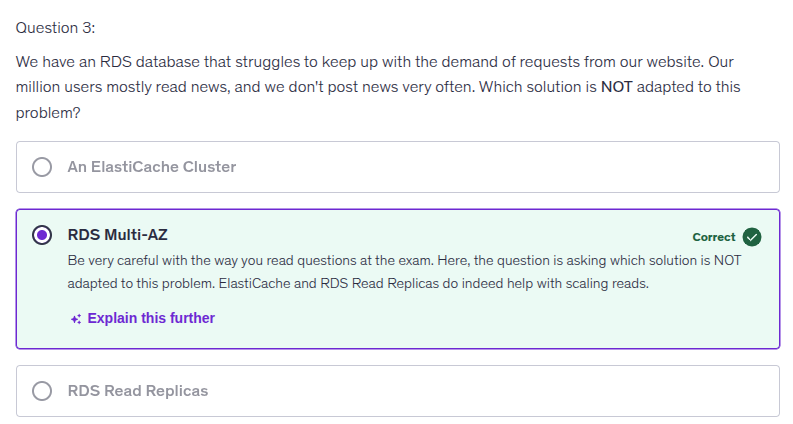
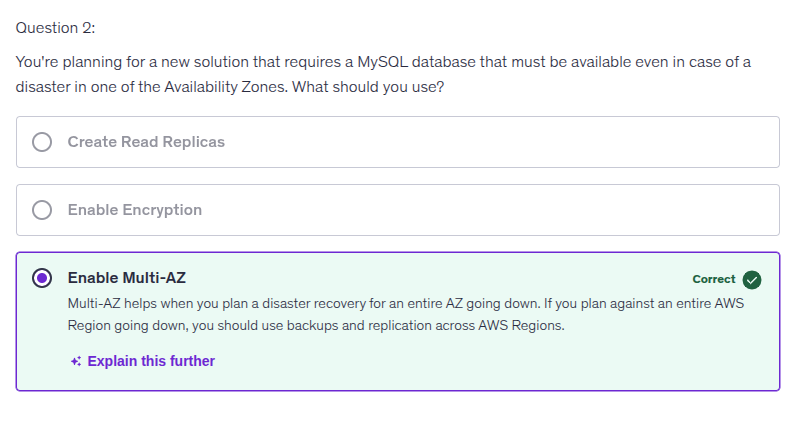
**Important ports:**

* FTP: 21
* SSH: 22
* SFTP: 22 (same as SSH)
* HTTP: 80
* HTTPS: 443

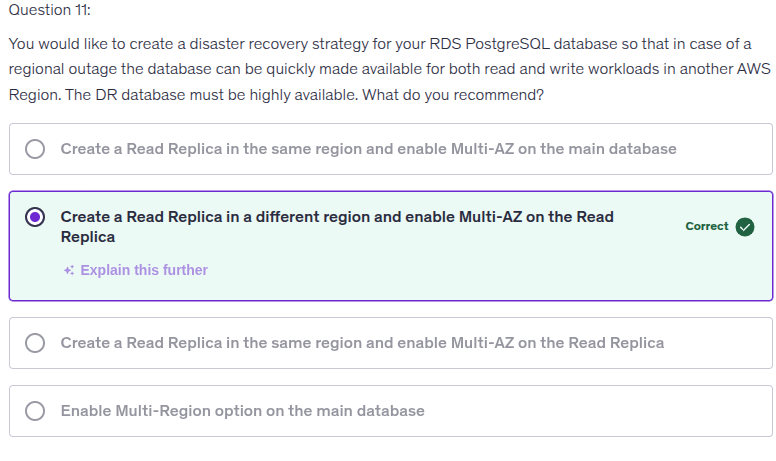
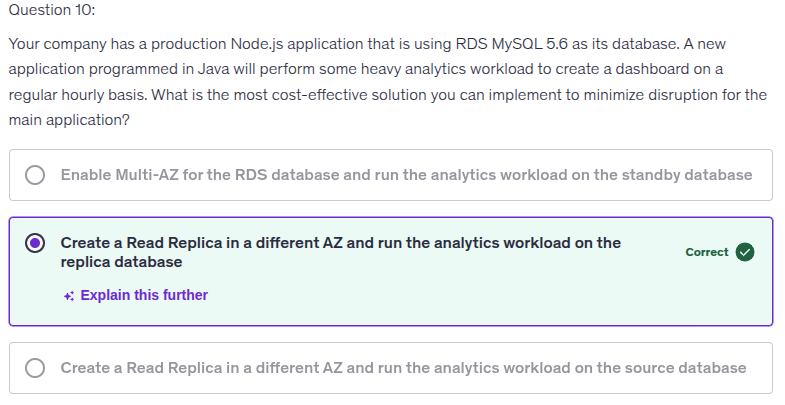
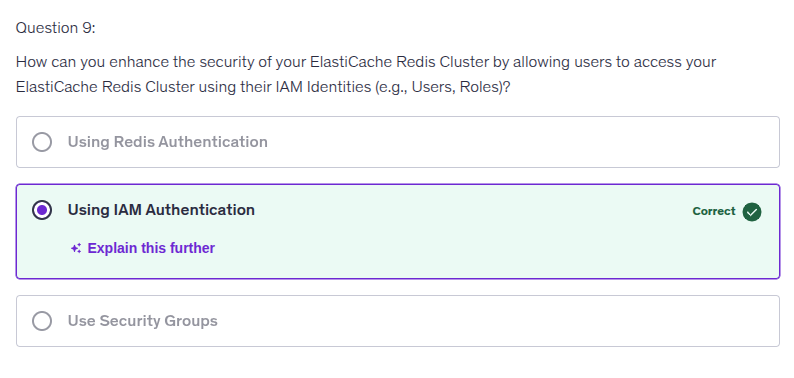
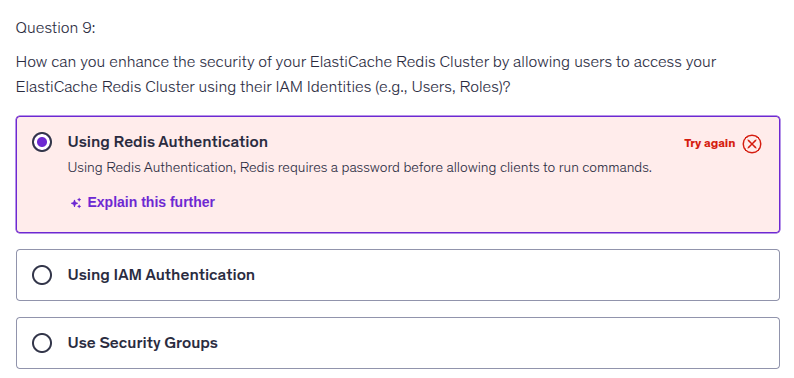
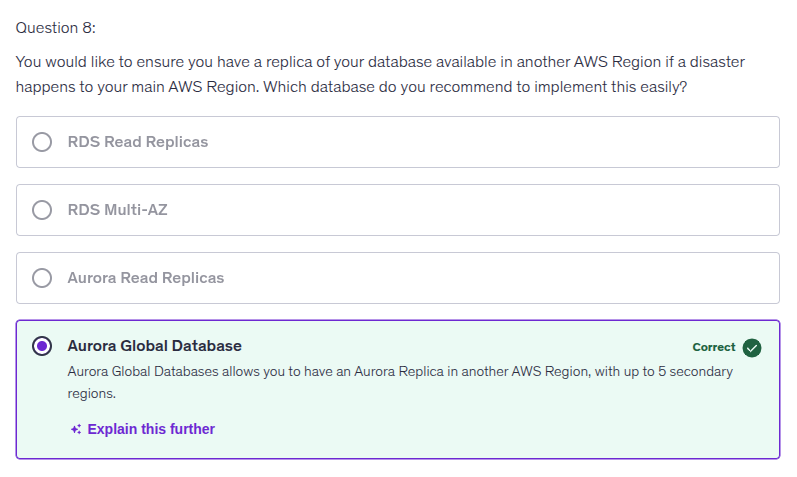
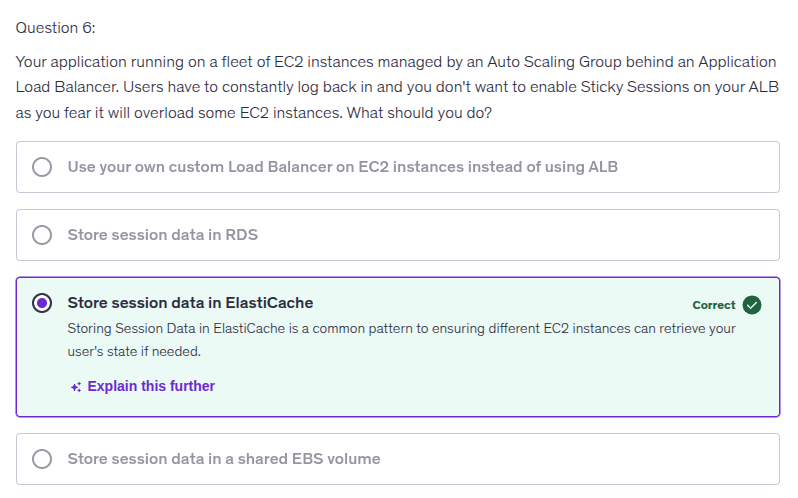
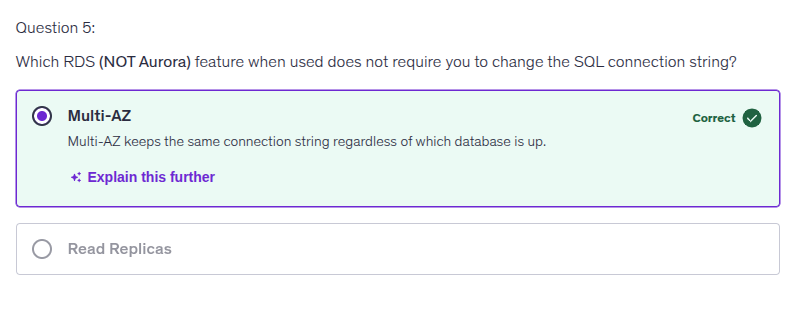
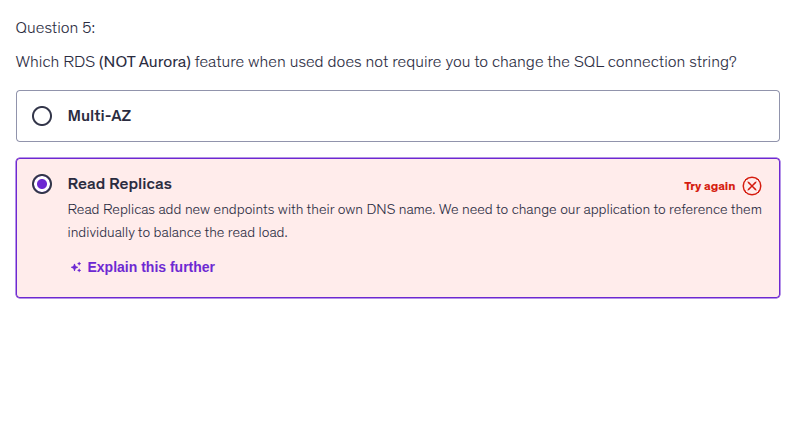
**vs RDS Databases ports:**

* PostgreSQL: 5432
* MySQL: 3306
* Oracle RDS: 1521
* MSSQL Server: 1433
* MariaDB: 3306 (same as MySQL)
* Aurora: 5432 (if PostgreSQL compatible) or 3306 (if MySQL compatible)

**Udemy Quiz:**



Your selection is correct because read replicas in RDS utilize asynchronous replication, meaning there is a lag in data propagation from the primary database to the replicas. This can lead users to experience eventual consistency, where they may not see their updates immediately after posting, as the replicas might still be refreshing with the latest information from the primary database.

The correct option is: Create a Read Replica in a different region and enable Multi-AZ on the Read Replica

Here's why:

Requirements Analysis:

* Regional outage protection → Need cross-Region solution
* Quick availability for read AND write → Need promotable replica
* High availability → Need Multi-AZ

Why this option works:

1. Cross-Region Read Replica provides disaster recovery in case of regional outage
2. Multi-AZ on the Read Replica ensures the DR database is highly available
3. When promoted during a disaster, the Read Replica becomes a standalone database that supports both read and write operations
4. After promotion, it's already Multi-AZ enabled, providing immediate high availability

Why other options don't work:

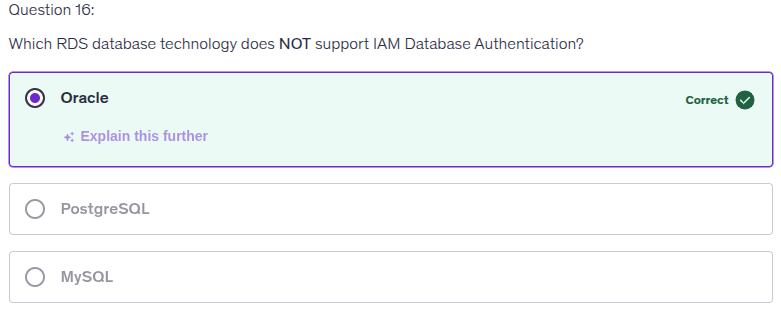
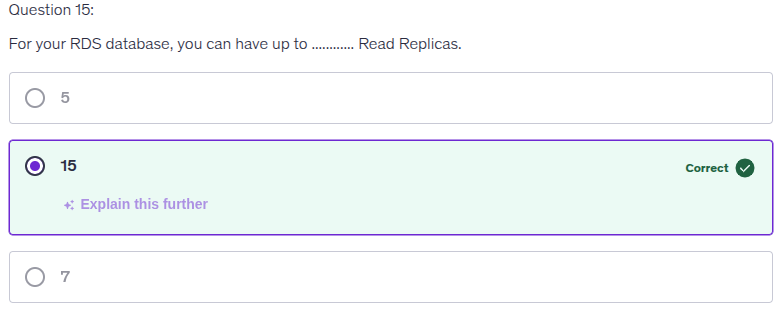
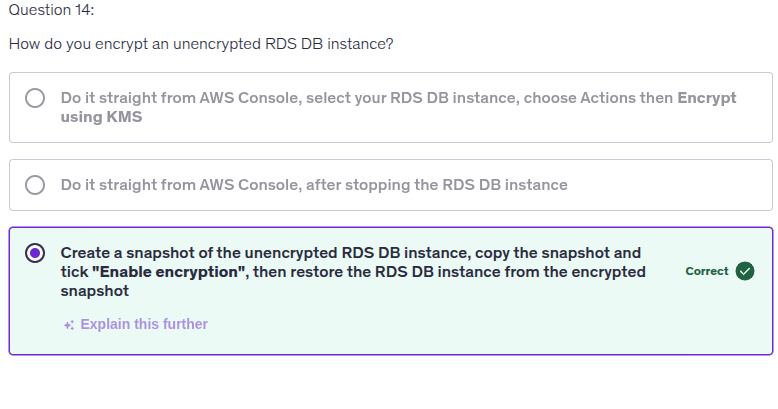
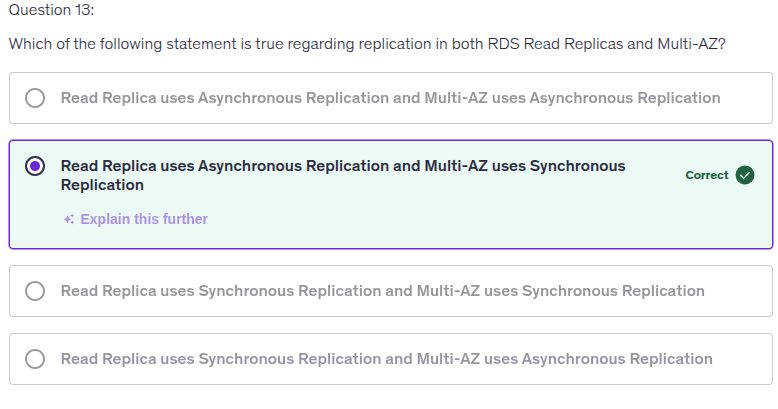
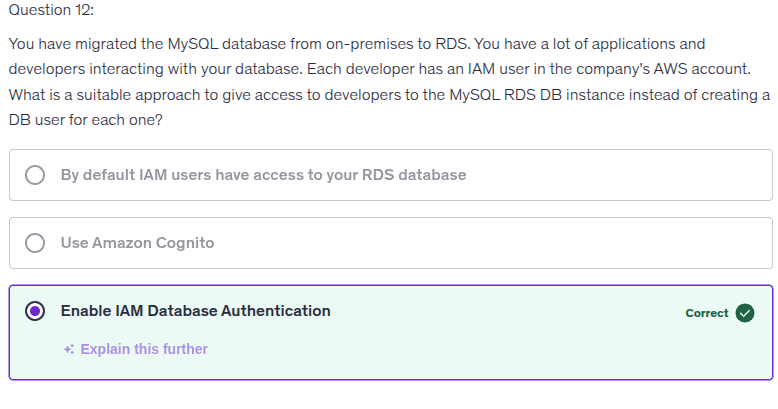
* Same region replica - Doesn't protect against regional outage
* Multi-AZ on main only - Doesn't provide cross-Region DR
* Multi-Region option - This doesn't exist for RDS PostgreSQL (it's available for Aurora as Global Database) This solution provides low RTO (Recovery Time Objective) for disaster recovery while maintaining high availability in the DR region.

Amazon RDS does not have a "Multi-Region option" for traditional RDS database engines (MySQL, PostgreSQL, MariaDB, Oracle, SQL Server).

For multi-Region capabilities, you need to use:

Amazon Aurora Global Database - which supports:

* Amazon Aurora MySQL-Compatible Edition
* Amazon Aurora PostgreSQL-Compatible Edition

IAM Database Authentication is supported only for:

* Amazon RDS for MySQL
* Amazon RDS for PostgreSQL
* Amazon RDS for MariaDB

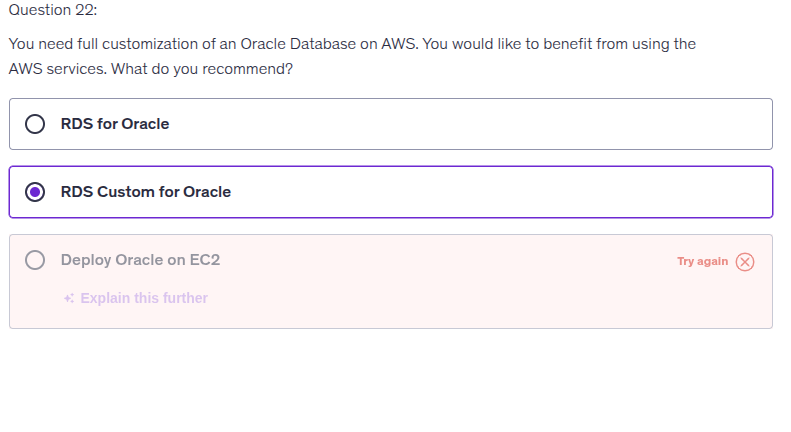
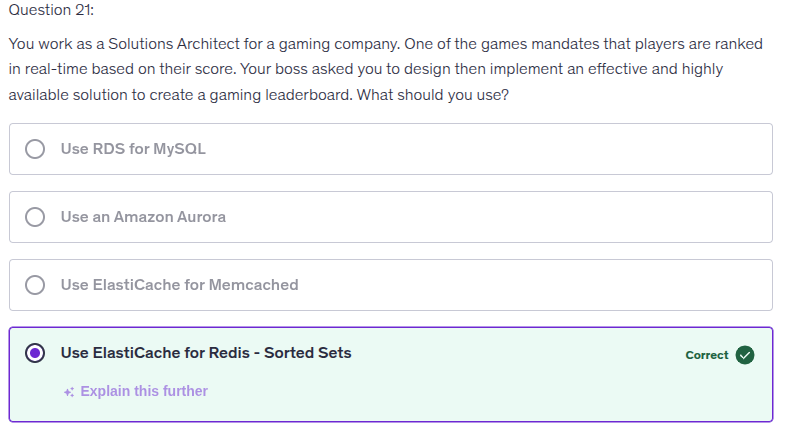
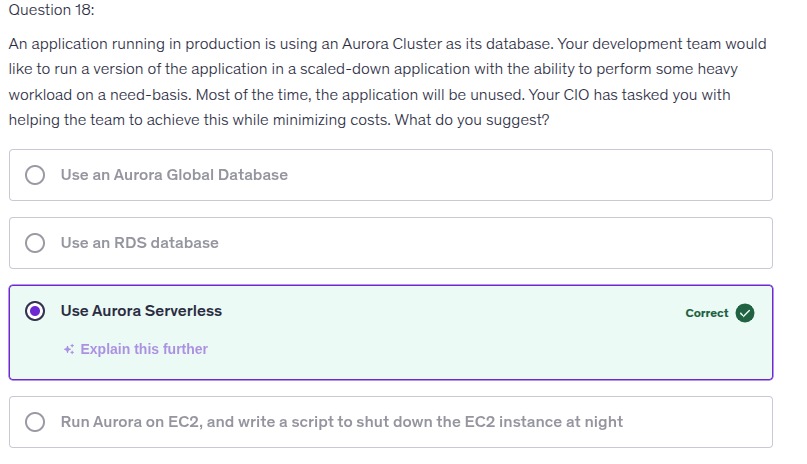
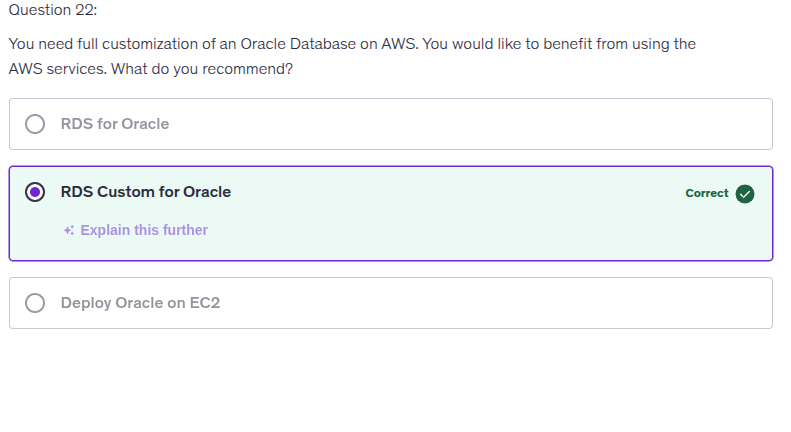
Not supported for:

* Amazon RDS for Oracle
* Amazon RDS for SQL Server
* Amazon RDS for Db2

Why Oracle doesn't support it:

Oracle databases use different authentication mechanisms. Oracle has its own authentication methods including:

* Database authentication (username/password)
* Operating system authentication
* External authentication through third-party software
* Enterprise directory services (like Active Directory)

  
RDS Custom for Oracle

Here's why:

RDS Custom for Oracle provides the best balance between full customization and AWS managed services benefits:

Advantages:

* Full OS and database access - Customize settings, install patches, enable native Oracle features
* AWS automation - Automated setup, operation, scaling, backups, and monitoring
* Flexibility - Access to underlying operating system for deep customization
* Managed infrastructure - AWS handles hardware, OS maintenance, and failure detection
* Supports Oracle Enterprise Edition with BYOL (Bring Your Own License)

Why not the other options:

RDS for Oracle:

* Limited customization - no OS-level access
* Cannot install custom patches or modify OS settings
* Best for standard workloads without special customization needs

Deploy Oracle on EC2:

* Full control but no AWS managed services benefits
* You handle ALL management tasks (setup, patching, backups, scaling, monitoring)
* More operational overhead

