Amazon

(AWS Certified Database - Specialty)

AWS Certified Database - Specialty

Total: **359 Questions**

Link:

Question: 1

A company has deployed an e-commerce web application in a new AWS account. An Amazon RDS for MySQL Multi-AZ DB instance is part of this deployment with a database-1.xxxxxxxxxxxx.us-east-1.rds.amazonaws.com endpoint listening on port 3306. The company's Database Specialist is able to log in to MySQL and run queries from the bastion host using these details.
When users try to utilize the application hosted in the AWS account, they are presented with a generic error message. The application servers are logging a `could not connect to server: Connection times out` error message to Amazon CloudWatch Logs.
What is the cause of this error?

A. The user name and password the application is using are incorrect.

B. The security group assigned to the application servers does not have the necessary rules to allow inbound connections from the DB instance.

C. The security group assigned to the DB instance does not have the necessary rules to allow inbound connections from the application servers.

D. The user name and password are correct, but the user is not authorized to use the DB instance.

Answer: C

Explanation:

The correct answer is C: "The security group assigned to the DB instance does not have the necessary rules to allow inbound connections from the application servers."

Here's why:

The error message "could not connect to server: Connection times out" strongly suggests a network connectivity issue. The application servers are unable to reach the RDS instance on the specified port (3306). Since the Database Specialist can connect from the bastion host, the RDS instance itself is likely functioning correctly, and the problem lies in the network path between the application servers and the RDS instance.

Security Groups act as virtual firewalls, controlling inbound and outbound traffic at the instance level. If the Security Group attached to the RDS instance doesn't have a rule allowing inbound traffic on port 3306 (MySQL's default port) from the application servers' Security Group (or specific IP addresses or CIDR blocks associated with the application servers), the connections will be blocked.

Option A is less likely. While incorrect credentials can cause connection errors, they usually result in authentication errors rather than timeouts. The fact that the Database Specialist can connect with their credentials from the bastion host further diminishes this possibility.

Option B is incorrect because inbound rules on the application server's security group dictate what traffic can enter the application servers. The application servers are initiating the connection to the database. Therefore, the outbound rules of the application server security group and the inbound rules of the RDS security group are critical. Since no outbound rules are mentioned, a default rule allowing all outbound traffic likely exists, so the issue is unlikely to be on the outbound side.

Option D is also less probable. Authorization issues usually manifest as authentication errors. The timeout indicates a lack of basic network connection rather than a successful connection followed by a rejected query due to insufficient permissions.

Therefore, the most plausible cause is that the RDS instance's Security Group is missing a rule allowing inbound TCP traffic on port 3306 from the application servers. This rule would specify the source as either the application servers' Security Group ID or the specific IP address range (CIDR block) used by the application servers.

For more information on Security Groups and network connectivity in AWS, refer to the following resources:

AWS Security Groups:https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html
Connecting to an RDS Instance:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html

Question: 2

An AWS CloudFormation stack that included an Amazon RDS DB instance was accidentally deleted and recent data was lost. A Database Specialist needs to add
RDS settings to the CloudFormation template to reduce the chance of accidental instance data loss in the future.
Which settings will meet this requirement? (Choose three.)

A. Set DeletionProtection to True
B. Set MultiAZ to True
C. Set TerminationProtection to True
D. Set DeleteAutomatedBackups to False
E. Set DeletionPolicy to Delete
F. Set DeletionPolicy to Retain

Answer: ADF

Explanation:

Here's a detailed justification for why options A, D, and F are the correct settings to reduce accidental data loss when an AWS CloudFormation stack containing an Amazon RDS DB instance is deleted, and why the other options are not suitable.

A. Set DeletionProtection to True: This is the most crucial setting to prevent accidental deletion. When DeletionProtection is enabled on an RDS instance, it prevents the instance from being deleted directly through the AWS Management Console, CLI, or API, including deletion triggered by CloudFormation stack deletion. This adds a safeguard requiring explicit disabling of DeletionProtection before deletion can occur, significantly reducing the risk of accidental data loss.
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-deletionpolicy.html

D. Set DeleteAutomatedBackups to False: By default, when an RDS instance is deleted, AWS may also delete any automated backups associated with it. Setting DeleteAutomatedBackups to False ensures that these automated backups are retained even after the RDS instance is deleted. This allows for potential restoration of the data from the backup, providing a safety net against permanent data loss. Note: RDS will store backups until the backup retention period expires or until you manually delete them.
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-rds-dbinstance.html#cfn-rds-dbinstance-deleteautomatedbackups

F. Set DeletionPolicy to Retain: The DeletionPolicy attribute in CloudFormation determines what happens to the resource when the CloudFormation stack is deleted. Setting it to Retain will preserve the RDS instance, even if the CloudFormation stack is deleted. The RDS instance will remain available in your AWS account but will no longer be managed by the CloudFormation stack. This protects the data from being lost during stack deletion.
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-deletionpolicy.html

Now, let's look at why the other options are incorrect:

B. Set MultiAZ to True: Setting MultiAZ enables high availability by creating a standby instance in a different Availability Zone. While this enhances resilience against failures, it doesn't prevent accidental deletion of the entire RDS instance, and does not prevent data loss in the event of accidental deletion. This relates to availability, not preventing deletion itself.

C. Set TerminationProtection to True: TerminationProtection is similar to DeletionProtection, but DeletionProtection is the property used by RDS itself. TerminationProtection is a CloudFormation attribute applicable to the stack and not specifically to the RDS resource itself. While setting termination protection on the stack is also a good practice, the question asks for settings related to the RDS instance within the stack.

E. Set DeletionPolicy to Delete: This is the opposite of what is desired. DeletionPolicy set to Delete will cause the RDS instance to be deleted when the CloudFormation stack is deleted, resulting in data loss.

In summary, DeletionProtection = True, DeleteAutomatedBackups = False, and DeletionPolicy = Retain are the appropriate settings within the CloudFormation template to safeguard the RDS instance and its data against accidental deletion and data loss. They provide layers of protection ensuring that even if the stack is mistakenly deleted, the data remains safe.

Question: 3

A Database Specialist is troubleshooting an application connection failure on an Amazon Aurora DB cluster with multiple Aurora Replicas that had been running with no issues for the past 2 months. The connection failure lasted for 5 minutes and corrected itself after that. The Database Specialist reviewed the Amazon
RDS events and determined a failover event occurred at that time. The failover process took around 15 seconds to complete. What is the MOST likely cause of the 5-minute connection outage?

A. After a database crash, Aurora needed to replay the redo log from the last database checkpoint
B. The client-side application is caching the DNS data and its TTL is set too high
C. After failover, the Aurora DB cluster needs time to warm up before accepting client connections
D. There were no active Aurora Replicas in the Aurora DB cluster

Answer: B

Explanation:

The most likely cause of the 5-minute connection outage after a 15-second failover is that the client-side application is caching DNS data with a high TTL (Time-To-Live).

Here's why:

Failover Time: The failover process itself only took 15 seconds. This implies the new primary instance was promoted relatively quickly.

DNS Caching: Applications often cache DNS records to avoid repeatedly querying the DNS server for the database endpoint's IP address. This improves performance but can cause issues after a failover.

TTL Effect: If the DNS record's TTL is set too high (e.g., 5 minutes), the application will continue to use the old, invalid IP address of the previous primary instance for up to 5 minutes after the failover, even though a new primary is already available. This results in connection failures during that period.

Resolution: After the TTL expires, the application queries the DNS server again, receives the new IP address of the new primary instance, and connections are restored.

Why other options are less likely:

A: Aurora is designed for fast failover. Redo log replay happens quickly, and 15 seconds for failover is in line with expectations.

C: Aurora does require some warm-up, but the warm-up period typically doesn't take several minutes. Aurora's storage is distributed, so it's more about caching at the new primary.

D: If there were no active Aurora Replicas, then the failover time would likely be much longer than 15 seconds, as Aurora would have to provision a new instance. The prompt mentions multiple replicas existed before the event.

Therefore, the most plausible explanation aligns with the application's DNS caching behavior and a high TTL setting preventing it from promptly recognizing the new database endpoint.

Supporting Links:

Amazon Aurora Failover:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Concepts.AuroraHighAvailability.html **RDS Connection Management:**https://aws.amazon.com/premiumsupport/knowledge-center/rds-dns-ttl/

Question: 4

A company is deploying a solution in Amazon Aurora by migrating from an on-premises system. The IT department has established an AWS Direct Connect link from the company's data center. The company's Database Specialist has selected the option to require SSL/TLS for connectivity to prevent plaintext data from being set over the network. The migration appears to be working successfully, and the data can be queried from a desktop machine. Two Data Analysts have been asked to query and validate the data in the new Aurora DB cluster. Both Analysts are unable to connect to Aurora. Their user names and passwords have been verified as valid and the Database Specialist can connect to the DB cluster using their accounts. The Database Specialist also verified that the security group configuration allows network from all corporate IP addresses. What should the Database Specialist do to correct the Data Analysts' inability to connect?

A. Restart the DB cluster to apply the SSL change.

B. Instruct the Data Analysts to download the root certificate and use the SSL certificate on the connection string to connect.

C. Add explicit mappings between the Data Analysts' IP addresses and the instance in the security group assigned to the DB cluster.

D. Modify the Data Analysts' local client firewall to allow network traffic to AWS.

Answer: B

Explanation:

The correct answer is B: Instruct the Data Analysts to download the root certificate and use the SSL certificate on the connection string to connect. Here's why:

The scenario explicitly states that the Database Specialist enabled SSL/TLS for connections to the Aurora DB cluster. This means the Aurora instance is configured to require secure connections. While the specialist can connect from their desktop, it's likely their client is already configured to use SSL. The Data Analysts' inability to connect suggests their client tools are not configured to establish an SSL/TLS connection. Simply having valid credentials and network connectivity (verified by the security group rules) is not sufficient when SSL is enforced.

Restarting the DB cluster (option A) is unnecessary. The SSL setting is already in effect, and restarting won't automatically configure client tools to use SSL. Adding specific IP address mappings in the security group (option C) is also incorrect. The problem isn't network access (the security group allows connections from all corporate IPs), but the type of connection. The firewall on the Data Analysts' local machine (option D) is likely not the issue, as the security group already allows traffic from the company's IPs, indicating that network-level access is likely present.

To enforce SSL/TLS connections, the client application connecting to the database needs to be configured to use SSL and trust the certificate authority (CA) that signed the database's SSL certificate. Aurora provides root certificates that clients must download and configure within their connection string or client application.
Without these, the client will attempt a non-SSL connection, which will be rejected by the Aurora instance.
The error messages seen by the analysts will likely indicate a failure to negotiate SSL/TLS, or an untrusted certificate.

Therefore, the Data Analysts must download the appropriate root certificate authority (CA) certificate for the Aurora endpoint, and then specify the certificate location and SSL mode within their connection string or client application. This will ensure that their connections are encrypted and authenticated, allowing them to access the Aurora DB cluster.

Further resources:

Connecting to an Aurora MySQL DB cluster with SSL:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/AuroraMySQL.Security.SSL.html **Connecting to an Aurora PostgreSQL DB cluster with SSL:**
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/AuroraPostgreSQL.Security.SSL.html

Question: 5

A company is concerned about the cost of a large-scale, transactional application using Amazon DynamoDB that only needs to store data for 2 days before it is deleted. In looking at the tables, a Database Specialist notices that much of the data is months old, and goes back to when the application was first deployed.
What can the Database Specialist do to reduce the overall cost?

A. Create a new attribute in each table to track the expiration time and create an AWS Glue transformation to delete entries more than 2 days old.

B. Create a new attribute in each table to track the expiration time and enable DynamoDB Streams on each table.

C. Create a new attribute in each table to track the expiration time and enable time to live (TTL) on each table.

D. Create an Amazon CloudWatch Events event to export the data to Amazon S3 daily using AWS Data Pipeline and then truncate the Amazon DynamoDB table.

Answer: C

Explanation:

The correct answer is C: Create a new attribute in each table to track the expiration time and enable time to live (TTL) on each table. Here's why:

DynamoDB TTL is the most cost-effective and efficient way to automatically remove data from a table after a specified time period. By adding an attribute to store the expiration timestamp (Unix epoch time in seconds) and enabling TTL, DynamoDB automatically deletes expired items in the background without impacting application performance. This eliminates the need for custom cleanup jobs or complex data pipelines.

Option A is incorrect because using AWS Glue for deleting entries older than 2 days is computationally expensive and resource-intensive. Glue transformations would require scanning entire tables regularly, consuming read capacity units (RCUs) and incurring additional costs for Glue runtime.

Option B is incorrect because DynamoDB Streams capture item-level modifications and can trigger other services, but they don't automatically delete expired items. They would require additional logic to be implemented in a Lambda function or other service to process the stream data and delete items, adding complexity and cost.

Option D is also incorrect because using CloudWatch Events, Data Pipeline, and S3 would create a complex and costly solution. Exporting data to S3 and truncating the table daily would be inefficient and potentially disruptive to application operations. Additionally, it doesn't address the existing months-old data.

TTL offers a native, server-side solution optimized for automatic data expiration in DynamoDB. It minimizes operational overhead and reduces costs by freeing up storage space and reducing the need for manual maintenance.Here's a link to the AWS Documentation on DynamoDB TTL:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/TTL.html

Question: 6

A company has an on-premises system that tracks various database operations that occur over the lifetime of a database, including database shutdown, deletion, creation, and backup.
The company recently moved two databases to Amazon RDS and is looking at a solution that would satisfy these requirements. The data could be used by other systems within the company.
Which solution will meet these requirements with minimal effort?

A. Create an Amazon CloudWatch Events rule with the operations that need to be tracked on Amazon RDS. Create an AWS Lambda function to act on these rules and write the output to the tracking systems.

B. Create an AWS Lambda function to trigger on AWS CloudTrail API calls. Filter on specific RDS API calls and write the output to the tracking systems.

C. Create RDS event subscriptions. Have the tracking systems subscribe to specific RDS event system notifications.

D. Write RDS logs to Amazon Kinesis Data Firehose. Create an AWS Lambda function to act on these rules and write the output to the tracking systems.

Answer: C

Explanation:

The correct answer is **C: Create RDS event subscriptions. Have the tracking systems subscribe to specific RDS event system notifications.**

Here's a detailed justification:

RDS Event Notifications are specifically designed for tracking events related to your RDS instances. These notifications provide a near real-time mechanism for receiving alerts about database instance changes, such as creation, deletion, backups, and shutdown. This aligns perfectly with the company's requirement to track database operations.

Option C offers the least effort because it leverages the built-in functionality of RDS. Setting up event subscriptions is straightforward and requires minimal coding or infrastructure. Tracking systems can directly subscribe to these notifications via SNS (Simple Notification Service), simplifying integration.

Option A, using CloudWatch Events, could technically work, but it's less direct and more complex than RDS event subscriptions. It requires configuring rules based on CloudWatch Logs which adds an extra layer of complexity.

Option B, using Lambda triggered by CloudTrail API calls, is also more complex than needed. CloudTrail captures a vast array of AWS API calls, and filtering them to isolate the specific RDS operations is not efficient. Also, CloudTrail is focused on auditing and governance, not real-time event notification for application integration.

Option D, writing RDS logs to Kinesis Data Firehose, is designed for streaming log data for analytics. It's overkill for simple event tracking and requires significant infrastructure and processing overhead. Firehose is better suited for analyzing database performance over time, not for triggering actions based on specific events.

RDS Event Notifications are the most direct, efficient, and cost-effective way to satisfy the company's requirements with minimal effort by using the notification mechanisms already provided by RDS.

Authoritative links:

RDS Event Notifications:https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Events.html

Amazon SNS:https://aws.amazon.com/sns/

Question: 7

A clothing company uses a custom ecommerce application and a PostgreSQL database to sell clothes to thousands of users from multiple countries. The company is migrating its application and database from its on-premises data center to the AWS Cloud. The company has selected Amazon EC2 for the application and Amazon RDS for PostgreSQL for the database. The company requires database passwords to be changed every 60 days. A Database Specialist needs to ensure that the credentials used by the web application to connect to the database are managed securely.
Which approach should the Database Specialist take to securely manage the database credentials?

A. Store the credentials in a text file in an Amazon S3 bucket. Restrict permissions on the bucket to the IAM role associated with the instance profile only. Modify the application to download the text file and retrieve the credentials on start up. Update the text file every 60 days.

B. Configure IAM database authentication for the application to connect to the database. Create an IAM user and map it to a separate database user for each ecommerce user. Require users to update their passwords every 60 days.

C. Store the credentials in AWS Secrets Manager. Restrict permissions on the secret to only the IAM role associated with the instance profile. Modify the application to retrieve the credentials from Secrets Manager on start up. Configure the rotation interval to 60 days.

D. Store the credentials in an encrypted text file in the application AMI. Use AWS KMS to store the key for decrypting the text file. Modify the application to decrypt the text file and retrieve the credentials on start up. Update the text file and publish a new AMI every 60 days.

Answer: C

Explanation:

The correct answer is **C: Store the credentials in AWS Secrets Manager. Restrict permissions on the secret to only the IAM role associated with the instance profile. Modify the application to retrieve the credentials from Secrets Manager on start up. Configure the rotation interval to 60 days.**

Here's a detailed justification:

Secrets Manager is designed specifically for managing secrets like database credentials securely. It provides encryption at rest using KMS and facilitates automatic rotation of credentials, meeting the company's 60-day password change requirement. By restricting permissions to the IAM role associated with the EC2 instances, you ensure only the application running on those instances can access the credentials. The application retrieves credentials dynamically at startup, eliminating the need to hardcode them within the application code or configuration files. This approach adheres to the principle of least privilege.

Option A is less secure because storing credentials in a text file, even in S3 with restricted permissions, increases the risk of accidental exposure or unauthorized access. Also, managing the rotation of credentials becomes a manual, error-prone process.

Option B is not suitable because IAM database authentication is better suited for individual user access and not generally used for application-to-database connections. Creating a separate IAM user for each ecommerce user is not scalable or practical for a large user base, and requiring each user to update their database password every 60 days impacts user experience.

Option D, storing credentials in an encrypted text file within the AMI, also presents challenges. Updating the AMI every 60 days for credential rotation leads to frequent AMI rebuilds and deployments, which impacts operational efficiency and increases the risk of errors. Furthermore, storing the encryption key in KMS and using it within the EC2 instance still poses a security risk, and using the same key to encrypt AMIs, is generally not a recommended practice.

In summary, AWS Secrets Manager is the best approach due to its security features, automatic rotation capabilities, and suitability for application-to-database credential management.

Relevant links:

AWS Secrets Manager: https://aws.amazon.com/secrets-manager/
Secrets Manager Best Practices: https://docs.aws.amazon.com/secretsmanager/latest/userguide/best-practices.html

Question: 8

A financial services company is developing a shared data service that supports different applications from throughout the company. A Database Specialist designed a solution to leverage Amazon ElastiCache for Redis with cluster mode enabled to enhance performance and scalability. The cluster is configured to listen on port 6379.
Which combination of steps should the Database Specialist take to secure the cache data and protect it from unauthorized access? (Choose three.)

A. Enable in-transit and at-rest encryption on the ElastiCache cluster.

B. Ensure that Amazon CloudWatch metrics are configured in the ElastiCache cluster.

C. Ensure the security group for the ElastiCache cluster allows all inbound traffic from itself and inbound traffic on TCP port 6379 from trusted clients only.

D. Create an IAM policy to allow the application service roles to access all ElastiCache API actions.

E. Ensure the security group for the ElastiCache clients authorize inbound TCP port 6379 and port 22 traffic from the trusted ElastiCache cluster's security group.

F. Ensure the cluster is created with the auth-token parameter and that the parameter is used in all subsequent commands.

Answer: ACF

Explanation:

The correct answer is ACF because it addresses different layers of security crucial for protecting sensitive data in a shared ElastiCache for Redis cluster.

A. Enable in-transit and at-rest encryption on the ElastiCache cluster: This protects the data both while it's being transmitted to and from the cache cluster (in-transit) and while it's stored on disk (at-rest). Encryption is fundamental for data privacy, particularly in financial services where sensitive information is common. AWS ElastiCache supports encryption using industry-standard algorithms.

https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/EncryptionAtRest.html and
https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/EncryptionInTransit.html

C. Ensure the security group for the ElastiCache cluster allows all inbound traffic from itself and inbound traffic on TCP port 6379 from trusted clients only: Security Groups act as virtual firewalls, controlling network access to the ElastiCache cluster. Restricting inbound traffic to only trusted clients on the required port (6379) significantly reduces the attack surface. Allowing traffic from itself is necessary for internal cluster communication. This implements the principle of least privilege.

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

F. Ensure the cluster is created with the auth-token parameter and that the parameter is used in all subsequent commands: The auth-token parameter in ElastiCache for Redis implements Redis AUTH. It requires clients to authenticate with a password before they can access data in the cache. This prevents unauthorized access even if the network security is breached. Consistent application of the auth-token in all commands is vital for maintaining this security measure.

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

B is incorrect because CloudWatch metrics primarily focus on monitoring performance and operational health,

not directly securing data. While important for identifying potential issues, they do not prevent unauthorized access.

D is incorrect because while IAM policies are essential for controlling API access, they don't secure the data itself within the ElastiCache cluster. This controls who can manage the ElastiCache cluster, not who can read or write data to it.

E is incorrect because the ElastiCache cluster does not initiate connections to clients, so clients shouldn't need to authorize inbound traffic from the cluster's security group. The client's security group should authorize outbound traffic to the ElastiCache cluster. Furthermore, allowing SSH (port 22) access to the ElastiCache clients unnecessarily widens the attack surface.

Question: 9

A company is running an Amazon RDS for PostgreSQL DB instance and wants to migrate it to an Amazon Aurora PostgreSQL DB cluster. The current database is 1 TB in size. The migration needs to have minimal downtime. What is the FASTEST way to accomplish this?

A.Create an Aurora PostgreSQL DB cluster. Set up replication from the source RDS for PostgreSQL DB instance using AWS DMS to the target DB cluster.

B.Use the pg_dump and pg_restore utilities to extract and restore the RDS for PostgreSQL DB instance to the Aurora PostgreSQL DB cluster.

C.Create a database snapshot of the RDS for PostgreSQL DB instance and use this snapshot to create the Aurora PostgreSQL DB cluster.

D.Migrate data from the RDS for PostgreSQL DB instance to an Aurora PostgreSQL DB cluster using an Aurora Replica. Promote the replica during the cutover.

Answer: D

Explanation:

The fastest method for migrating a 1 TB RDS for PostgreSQL instance to Aurora PostgreSQL with minimal downtime is using an Aurora Replica. Here's why:

Option D leverages the native Aurora Replica functionality, specifically designed for migrations with minimal downtime. Aurora Replicas provide physical replication and are tightly integrated within the AWS ecosystem. Creating an Aurora Replica directly within the Aurora PostgreSQL cluster synchronizes data almost in real-time. Once the replica catches up, it can be promoted to become the primary instance, minimizing the downtime required for the final cutover.

Option A, using AWS DMS (Database Migration Service), while viable, generally introduces more overhead and complexity compared to Aurora Replicas for PostgreSQL-to-PostgreSQL migrations within AWS. DMS is well-suited for heterogeneous migrations (e.g., Oracle to PostgreSQL) or migrations across environments, but it isn't the fastest approach for this specific scenario. DMS involves ongoing data replication, transformation (if needed), and a final cutover.

Option B, using pg_dump and pg_restore, involves a logical backup and restore process. For a 1 TB database, this process can be time-consuming, leading to significant downtime. It requires taking the original database offline or read-only during the dump process to ensure data consistency and the restore process can also take a long time.

Option C, creating a snapshot and restoring it into Aurora, although faster than pg_dump/restore, still involves downtime because the snapshot represents a point-in-time copy. The Aurora cluster is unavailable during the restoration. While it is faster than restoring from a logical dump, it's not as fast as the near real-time replication offered by Aurora Replicas. Additionally, once restored, any changes made after the snapshot are

lost.

Therefore, option D provides the fastest migration path with minimal downtime because Aurora Replicas offer near real-time replication, allowing for a quick promotion and cutover once the replica is synchronized.

References:

Aurora PostgreSQL Replicas
Migrating to Amazon Aurora PostgreSQL using logical replication

Question: 10

A Database Specialist is migrating a 2 TB Amazon RDS for Oracle DB instance to an RDS for PostgreSQL DB instance using AWS DMS. The source RDS Oracle
DB instance is in a VPC in the us-east-1 Region. The target RDS for PostgreSQL DB instance is in a VPC in the use-west-2 Region.
Where should the AWS DMS replication instance be placed for the MOST optimal performance?

A. In the same Region and VPC of the source DB instance
B. In the same Region and VPC as the target DB instance
C. In the same VPC and Availability Zone as the target DB instance
D. In the same VPC and Availability Zone as the source DB instance

Answer: C

Explanation:

Here's a detailed justification for the correct answer (C) regarding AWS DMS replication instance placement:

The optimal placement for the AWS DMS replication instance is **C. In the same VPC and Availability Zone as the target DB instance (RDS for PostgreSQL)**. This is primarily due to reducing latency and costs associated with cross-Region data transfer.

Reducing Latency: Placing the replication instance in the same Region and VPC as the target database minimizes the network latency between the replication instance and the target. Data is transferred more quickly and efficiently within the same network environment.

Reducing Data Transfer Costs: AWS charges for data transfer across Regions. By keeping the DMS instance in the same Region as the target, data transfer charges between the DMS instance and the target database are avoided. While data from the source must cross-region, the bulk of the transformation and loading activity benefits from regional co-location.

Availability Zones: Placing the replication instance in the same Availability Zone as the target DB instance further minimizes latency and ensures the fastest possible data transfer between the two. This is because traffic within the same AZ experiences the lowest possible latency compared to traffic between different AZs in the same region.

Why not the Source Region?: While placing the instance in the source region initially seems logical, the bulk of the data movement occurs after the initial read. The DMS instance is constantly applying changes to the target. Since the target database is in us-west-2, minimizing the latency for the replication instance to apply those changes is paramount for performance.

DMS Architecture: AWS DMS operates by reading data from the source, transforming it as needed, and then loading it into the target. The replication instance acts as the intermediary between the source and target, performing these tasks. Positioning the replication instance near the target ensures that the "last mile" of data transfer (loading into the target) is as efficient as possible.

Therefore, placing the DMS replication instance in the same VPC and Availability Zone as the target RDS for PostgreSQL DB instance provides the most optimal performance by reducing latency and minimizing data transfer costs.**Authoritative Links:**

AWS DMS Best Practices: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_BestPractices.html AWS DMS Network Infrastructure Considerations: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Security.html#CHAP_Security.Network

Question: 11

The Development team recently executed a database script containing several data definition language (DDL) and data manipulation language (DML) statements on an Amazon Aurora MySQL DB cluster. The release accidentally deleted thousands of rows from an important table and broke some application functionality.
This was discovered 4 hours after the release. Upon investigation, a Database Specialist tracked the issue to a DELETE command in the script with an incorrect
WHERE clause filtering the wrong set of rows.
The Aurora DB cluster has Backtrack enabled with an 8-hour backtrack window. The Database Administrator also took a manual snapshot of the DB cluster before the release started. The database needs to be returned to the correct state as quickly as possible to resume full application functionality. Data loss must be minimal. How can the Database Specialist accomplish this?

A. Quickly rewind the DB cluster to a point in time before the release using Backtrack.

B. Perform a point-in-time recovery (PITR) of the DB cluster to a time before the release and copy the deleted rows from the restored database to the original database.

C. Restore the DB cluster using the manual backup snapshot created before the release and change the application configuration settings to point to the new DB cluster.

D. Create a clone of the DB cluster with Backtrack enabled. Rewind the cloned cluster to a point in time before the release. Copy deleted rows from the clone to the original database.

Answer: B

Explanation:

The correct answer is B because it offers the fastest and most precise recovery method in this scenario. Backtrack, while enabled, can introduce inconsistencies if transactions were in progress during the rewind.

Since the exact time of the error is known (4 hours ago within the 8-hour backtrack window), PITR allows restoring the database to a specific point in time just before the erroneous DELETE statement. This avoids potential inconsistencies of a Backtrack rewind.

Option A, while faster in theory, might not be precise enough. Backtrack rewinds to the latest possible consistent state within the specified window, not necessarily the exact moment before the error. This can include unwanted changes or miss essential transactions before the problem. It would be more applicable if a consistent state was difficult to pinpoint.

Option C involves restoring from a snapshot, which takes longer than PITR. This also necessitates application configuration changes to point to a new DB cluster, adding to the recovery time and complexity. Minimal downtime is a priority, making this option less favorable.

Option D involves creating a clone, which also introduces delay and resource consumption. Cloning with backtrack and rewinding adds unnecessary complexity when PITR is a readily available, faster solution. It doesn't directly address the immediate need to recover the deleted rows to the original database quickly.

Therefore, PITR to a point just before the release provides the most rapid and precise restoration with minimal data loss, fulfilling the requirements. You can then copy the rows from the recovered db to your existing db to minimize changes and restore data quickly.

Relevant Links:

Aurora Backtrack
Aurora Point-in-Time Recovery

Question: 12

A company is load testing its three-tier production web application deployed with an AWS CloudFormation template on AWS. The Application team is making changes to deploy additional Amazon EC2 and AWS Lambda resources to expand the load testing capacity. A Database Specialist wants to ensure that the changes made by the Application team will not change the Amazon RDS database resources already deployed.
Which combination of steps would allow the Database Specialist to accomplish this? (Choose two.)

A. Review the stack drift before modifying the template
B. Create and review a change set before applying it
C. Export the database resources as stack outputs
D. Define the database resources in a nested stack
E. Set a stack policy for the database resources

Answer: BE

Explanation:

The correct answer is **BE**. Here's why:

B. Create and review a change set before applying it: A change set allows you to preview the changes that CloudFormation will make to your stack **before** they are actually applied. This is crucial for understanding the impact of the Application team's changes. By reviewing the change set, the Database Specialist can specifically see if the changes affect the RDS database resources. If the change set doesn't show any modifications to the database resources, they can be confident that the database is not being affected.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html

E. Set a stack policy for the database resources: A stack policy is a JSON document that defines the resource updates you can perform on a specific stack. By setting a stack policy that explicitly denies updates to the database resources, the Database Specialist can prevent accidental or unauthorized changes from being made during the Application team's deployment. This acts as a safety net.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/protect-stack-resources.html

Now, let's discuss why the other options are not the best fit:

A. Review the stack drift before modifying the template: Stack drift detection helps identify if the actual configuration of your stack resources has deviated from what is defined in your CloudFormation template.

While helpful for maintaining consistency, it doesn't proactively prevent unintended changes during an update. Drift is checked after changes might have occurred.

C. Export the database resources as stack outputs: Exporting resources as stack outputs makes those resources discoverable and usable by other CloudFormation stacks (for cross-stack referencing). It doesn't inherently prevent modifications to the database resources in the original stack.

D. Define the database resources in a nested stack: Using nested stacks provides modularity and organization. However, on its own, this doesn't guarantee that the Application team's updates to the main CloudFormation template will not affect the nested database stack. It just compartmentalizes the resources.

The main template could still potentially attempt to modify the nested stack (unless specifically restricted using a stack policy).

In summary, using a change set provides a preview of the impact of modifications, while a stack policy offers a proactive safeguard against unwanted changes to the database resources during deployments.

Question: 13

A manufacturing company's website uses an Amazon Aurora PostgreSQL DB cluster.
Which configurations will result in the LEAST application downtime during a failover? (Choose three.)

A. Use the provided read and write Aurora endpoints to establish a connection to the Aurora DB cluster.

B. Create an Amazon CloudWatch alert triggering a restore in another Availability Zone when the primary Aurora DB cluster is unreachable.

C. Edit and enable Aurora DB cluster cache management in parameter groups.

D. Set TCP keepalive parameters to a high value.

E. Set JDBC connection string timeout variables to a low value.

F. Set Java DNS caching timeouts to a high value.

Answer: ACE

Explanation:

The correct answer is ACE. Here's why:

A. Use the provided read and write Aurora endpoints to establish a connection to the Aurora DB cluster.

Aurora provides a cluster endpoint (write endpoint) and reader endpoint. The cluster endpoint always points to the primary DB instance. In a failover, Aurora automatically promotes a read replica to be the new primary and updates the cluster endpoint to point to the new primary. Using these endpoints abstracts away the need to track individual instance addresses, minimizing application downtime.
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Overview.html

C. Edit and enable Aurora DB cluster cache management in parameter groups.

Aurora provides a cluster cache which enables storing data closer to the database which results in faster reads and writes. This also means that if a cache is enabled correctly and consistently it will take less time to read data after a failover. This also means that if a cache is not enabled correctly then the failover would take a longer time.

E. Set JDBC connection string timeout variables to a low value.

During a failover, existing connections to the primary instance will be lost. A low timeout value in the JDBC connection string ensures the application quickly detects the broken connection and retries to connect to the now-updated cluster endpoint. This rapid detection is crucial for reducing application downtime. A high timeout value would cause the application to wait longer before attempting reconnection.
https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/connecting-to-aws.html

Why the other options are incorrect:

B. Create an Amazon CloudWatch alert triggering a restore in another Availability Zone when the primary Aurora DB cluster is unreachable. Aurora automatically handles failovers to a replica in another Availability Zone. Creating a CloudWatch alert and triggering a restore would be redundant and potentially much slower than Aurora's built-in failover mechanism.

D. Set TCP keepalive parameters to a high value. While TCP keepalive can help detect dead connections, setting it to a high value delays this detection, increasing downtime during a failover. Keepalive is less relevant when using JDBC connection timeouts for failover handling.

F. Set Java DNS caching timeouts to a high value. High DNS caching timeouts can prevent the application from quickly resolving the updated Aurora cluster endpoint after a failover. The application would continue to try connecting to the old (now unavailable) IP address for the duration of the cache timeout, leading to

extended downtime. Lowering the DNS caching timeout is crucial for fast failover.

Question: 14

A company is hosting critical business data in an Amazon Redshift cluster. Due to the sensitive nature of the data, the cluster is encrypted at rest using AWS
KMS. As a part of disaster recovery requirements, the company needs to copy the Amazon Redshift snapshots to another Region.
Which steps should be taken in the AWS Management Console to meet the disaster recovery requirements?

A. Create a new KMS customer master key in the source Region. Switch to the destination Region, enable Amazon Redshift cross-Region snapshots, and use the KMS key of the source Region.

B. Create a new IAM role with access to the KMS key. Enable Amazon Redshift cross-Region replication using the new IAM role, and use the KMS key of the source Region.

C. Enable Amazon Redshift cross-Region snapshots in the source Region, and create a snapshot copy grant and use a KMS key in the destination Region.

D. Create a new KMS customer master key in the destination Region and create a new IAM role with access to the new KMS key. Enable Amazon Redshift cross-Region replication in the source Region and use the KMS key of the destination Region.

Answer: C

Explanation:

The correct answer is C. Let's break down why.

To copy Redshift snapshots encrypted with KMS to another region, you need to explicitly authorize the copy operation using a snapshot copy grant. The core issue is that the KMS key in the source region cannot be directly used in the destination region for decryption due to KMS key region-specific nature and security isolation.

Option C directly addresses this. Enabling cross-Region snapshots in the source Region sets up the fundamental capability to copy snapshots. Critically, creating a snapshot copy grant in the destination region is essential. This grant authorizes Redshift in the source region to use a KMS key in the destination region to re-encrypt the snapshot during the copy process.

Option A is incorrect because you don't need to create a new KMS key in the source region; you're already using one for encryption at rest. Trying to use the KMS key from the source region directly in the destination region isn't possible due to KMS key regionality.

Option B incorrectly suggests using an IAM role for cross-Region replication. While IAM roles are crucial for permissions within a region, cross-Region snapshot copying relies on snapshot copy grants specifically designed for KMS key access in the destination region.

Option D is also incorrect because creating a KMS key and IAM role in the destination region alone is insufficient. A snapshot copy grant is the correct mechanism that connects the source region Redshift with the destination region KMS key, enabling the re-encryption required during the cross-region copy. The grant explicitly gives permission for the source region Redshift service account to use the destination region's KMS key.

In summary, enabling cross-Region snapshots and configuring a snapshot copy grant are the necessary steps for securely copying KMS-encrypted Redshift snapshots to another region.

Supporting Documentation:

Amazon Redshift Snapshot Copy Grants:https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-db-encryption.html#snapshot-copy-grants

Copying snapshots to another AWS Region:https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-snapshots.html#cross-region-snapshots

Question: 15

A company has a production Amazon Aurora Db cluster that serves both online transaction processing (OLTP) transactions and compute-intensive reports. The reports run for 10% of the total cluster uptime while the OLTP transactions run all the time. The company has benchmarked its workload and determined that a six- node Aurora DB cluster is appropriate for the peak workload.
The company is now looking at cutting costs for this DB cluster, but needs to have a sufficient number of nodes in the cluster to support the workload at different times. The workload has not changed since the previous benchmarking exercise.
How can a Database Specialist address these requirements with minimal user involvement?

A. Split up the DB cluster into two different clusters: one for OLTP and the other for reporting. Monitor and set up replication between the two clusters to keep data consistent.

B. Review all evaluate the peak combined workload. Ensure that utilization of the DB cluster node is at an acceptable level. Adjust the number of instances, if necessary.

C. Use the stop cluster functionality to stop all the nodes of the DB cluster during times of minimal workload. The cluster can be restarted again depending on the workload at the time.

D. Set up automatic scaling on the DB cluster. This will allow the number of reader nodes to adjust automatically to the reporting workload, when needed.

Answer: D

Explanation:

Here's a detailed justification for why option D is the best solution, along with explanations of why the other options are less suitable:

Justification for Option D: Set up automatic scaling on the DB cluster.

Automatic scaling is the most effective approach to address the company's need to cut costs while maintaining sufficient capacity for both OLTP and reporting workloads with minimal manual intervention.

Aurora's auto-scaling feature allows the database cluster to dynamically adjust the number of Aurora Replicas based on real-time demand. During the 10% of the time when reports are running, the cluster can automatically scale up the number of reader nodes to handle the increased compute load. When the reporting workload decreases, the cluster will automatically scale down the number of reader nodes, thus reducing costs. This approach requires minimal user intervention once the auto-scaling policy is configured. Auto scaling works with metrics like CPU utilization, connection counts and custom CloudWatch metrics to trigger scaling actions. By monitoring the overall database load, the number of reader nodes can be adjusted accordingly to always maintain sufficient capacity.https://aws.amazon.com/rds/aurora/features/auto-scaling/

Why other options are less suitable:

Option A: Split up the DB cluster into two different clusters: This involves creating two separate Aurora clusters, one for OLTP and one for reporting. While this could technically isolate the workloads, it introduces significant complexity and overhead. Maintaining data consistency between the two clusters requires setting up and managing replication, which can be difficult to implement and maintain, introduce latency, and potentially impact performance. This solution requires a lot more management overhead and is not the best way to reduce costs and minimize user involvement.

Option B: Review the peak combined workload and adjust the number of instances: This is not a dynamically scaling approach, meaning you set a fixed number of instances based on the peak combined workload. While you could ensure sufficient capacity, it doesn't address the requirement to cut costs during the periods when the reporting workload is not running. The cluster would be over-provisioned during the majority of the time, resulting in unnecessary expenses.

Option C: Use the stop cluster functionality: Stopping the entire Aurora cluster during periods of minimal workload is a drastic measure. It would interrupt all database services, including OLTP transactions. This option is not suitable because the OLTP transactions need to run continuously. Stopping the cluster is more appropriate for development or test environments where downtime is acceptable.

In summary, automatic scaling provides the most efficient and cost-effective way to handle fluctuating workloads in Aurora while minimizing user intervention and ensuring continuous availability for critical OLTP transactions.

Question: 16

A company is running a finance application on an Amazon RDS for MySQL DB instance. The application is governed by multiple financial regulatory agencies.
The RDS DB instance is set up with security groups to allow access to certain Amazon EC2 servers only. AWS KMS is used for encryption at rest.
Which step will provide additional security?

A. Set up NACLs that allow the entire EC2 subnet to access the DB instance
B. Disable the master user account
C. Set up a security group that blocks SSH to the DB instance
D. Set up RDS to use SSL for data in transit

Answer: D

Explanation:

The correct answer is **D. Set up RDS to use SSL for data in transit.**

Here's why:

The question emphasizes security for a finance application governed by strict regulations. While the RDS instance already uses security groups for EC2 access control and KMS for encryption at rest, it doesn't address data in transit. SSL (Secure Sockets Layer) encrypts the data exchanged between the application and the database server. This prevents eavesdropping and man-in-the-middle attacks, crucial for protecting sensitive financial data as it moves across the network.

Option A is incorrect because NACLs allowing entire subnet access broadens the attack surface and contradicts the principle of least privilege. Security groups already control access, and NACLs should be more restrictive if anything.

Option B, disabling the master user account, might seem secure initially, but it's problematic. The master user account is essential for administrative tasks, database maintenance, and initial configuration. Disabling it would likely cripple the RDS instance's operability and create significant management difficulties. The master user should be secured with strong passwords and restricted access, not disabled.

Option C, blocking SSH to the DB instance, is misleading. RDS for MySQL doesn't typically use SSH for application connectivity. Applications connect via the MySQL protocol (port 3306 by default). Security groups are used to control access to this port. Therefore, an explicit rule blocking SSH isn't the appropriate security control in this context. It addresses a protocol not normally used for application-database communication.

Therefore, implementing SSL provides an additional layer of security specifically designed to protect sensitive data while it is being transmitted between the application and the database. It addresses a critical security gap not covered by the other options.

Further resources for understanding RDS security and SSL:

AWS RDS Security:https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Security.html **Using SSL with a MySQL DB Instance:**
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html

Question: 17

A company needs a data warehouse solution that keeps data in a consistent, highly structured format. The company requires fast responses for end-user queries when looking at data from the current year, and users must have access to the full 15-year dataset, when needed. This solution also needs to handle a fluctuating number incoming queries. Storage costs for the 100 TB of data must be kept low.
Which solution meets these requirements?

A. Leverage an Amazon Redshift data warehouse solution using a dense storage instance type while keeping all the data on local Amazon Redshift storage. Provision enough instances to support high demand.

B. Leverage an Amazon Redshift data warehouse solution using a dense storage instance to store the most recent data. Keep historical data on Amazon S3 and access it using the Amazon Redshift Spectrum layer. Provision enough instances to support high demand.

C. Leverage an Amazon Redshift data warehouse solution using a dense storage instance to store the most recent data. Keep historical data on Amazon S3 and access it using the Amazon Redshift Spectrum layer. Enable Amazon Redshift Concurrency Scaling.

D. Leverage an Amazon Redshift data warehouse solution using a dense storage instance to store the most recent data. Keep historical data on Amazon S3 and access it using the Amazon Redshift Spectrum layer. Leverage Amazon Redshift elastic resize.

Answer: C

Explanation:

Here's a detailed justification for why option C is the best solution for the given scenario, along with supporting concepts and links:

Option C is the most appropriate solution because it effectively balances performance, cost, and scalability requirements. The key elements are leveraging Amazon Redshift, Redshift Spectrum, and Concurrency Scaling.

First, Redshift provides a columnar data warehouse optimized for analytical queries with a highly structured format, fulfilling the company's initial requirement. A dense storage instance type is used to store the most recent data, ensuring fast query responses for the current year's data.

Second, infrequently accessed historical data is stored on Amazon S3, which is cost-effective for large datasets. This addresses the requirement of low storage costs for the 100 TB dataset. Redshift Spectrum allows querying of this data on S3 without loading it into Redshift, which is critical for accessing the entire 15-year dataset when needed.

Third, Concurrency Scaling handles the fluctuating number of incoming queries. It automatically adds extra compute capacity to handle concurrent queries, ensuring consistent performance even during peak demand. Redshift Concurrency Scaling allows you to support hundreds or thousands of concurrent users and queries with consistently fast query performance.

Option A is less optimal because keeping all data on Redshift local storage, while providing fast access, would significantly increase costs. Also, provisioning instances to handle peak demand permanently leads to underutilization and wasted resources during periods of low demand.

Option B is better than A but lacks scalability. While it separates recent data from historical data, it still requires provisioning instances based on the maximum expected load, leading to potential inefficiencies and cost overruns. It doesn't address fluctuating query demands effectively as Concurrency Scaling does.

Option D uses Redshift Elastic Resize. Elastic Resize involves resizing the Redshift cluster by adding or removing nodes, which can cause downtime. While useful for long-term scaling, Elastic Resize is less suited for handling short-term fluctuations in query concurrency compared to Concurrency Scaling. Elastic Resize takes longer than Concurrency Scaling to scale.

In summary, option C provides a cost-effective, scalable, and performant solution by combining Redshift, Redshift Spectrum, and Concurrency Scaling to meet all the requirements of the data warehouse solution.

Supporting Links:

Amazon Redshift:https://aws.amazon.com/redshift/
Amazon Redshift Spectrum:https://aws.amazon.com/redshift/spectrum/
Amazon Redshift Concurrency Scaling:https://aws.amazon.com/redshift/features/concurrency-scaling/
Amazon S3:https://aws.amazon.com/s3/

Question: 18

A gaming company wants to deploy a game in multiple Regions. The company plans to save local high scores in Amazon DynamoDB tables in each Region. A
Database Specialist needs to design a solution to automate the deployment of the database with identical configurations in additional Regions, as needed. The solution should also automate configuration changes across all Regions.
Which solution would meet these requirements and deploy the DynamoDB tables?

A. Create an AWS CLI command to deploy the DynamoDB table to all the Regions and save it for future deployments.

B. Create an AWS CloudFormation template and deploy the template to all the Regions.

C. Create an AWS CloudFormation template and use a stack set to deploy the template to all the Regions.

D. Create DynamoDB tables using the AWS Management Console in all the Regions and create a step-by-step guide for future deployments.

Answer: C

Explanation:

The correct answer is C, using an AWS CloudFormation template with a stack set. Here's why:

Why C is the best approach:

Infrastructure as Code (IaC): CloudFormation allows you to define your DynamoDB tables (including configurations like indexes, provisioned throughput, and global tables if needed) in a declarative template. This is IaC, ensuring repeatability and consistency.

Stack Sets for Multi-Region Deployment: Stack Sets are a CloudFormation feature specifically designed for deploying and managing stacks across multiple AWS accounts and Regions from a single CloudFormation template. This directly addresses the requirement to automate deployment to additional Regions.

Centralized Management: Stack Sets enable you to centrally manage updates and configuration changes to your DynamoDB tables across all Regions. When you modify the CloudFormation template and update the stack set, the changes are automatically propagated to all target Regions.

Automated Configuration Changes: Modifying the template allows to deploy updates to all regions. This is a fundamental benefit of IaC.

Why other options are not ideal:

A: AWS CLI command: While CLI commands can be used, they lack the declarative nature of CloudFormation. A CLI-based approach would be procedural, making updates and configuration changes across multiple Regions more complex and error-prone. It won't manage interdependencies or ensure consistent state as

effectively as CloudFormation.

B: CloudFormation without Stack Sets: Deploying the same CloudFormation template individually to each Region would be manual and time-consuming. It also wouldn't provide a centralized mechanism for managing configuration changes across all Regions.

D: AWS Management Console: This approach is entirely manual and not scalable. It lacks automation and consistency, making it unsuitable for deploying and managing DynamoDB tables across multiple Regions in a repeatable and reliable way. It does not automate configuration changes.

In Summary:

Using a CloudFormation template with a stack set is the best solution because it provides Infrastructure as Code for consistent deployments, automates deployment across multiple Regions, and enables centralized management of configuration changes. This is a scalable and maintainable approach suitable for a gaming company's multi-region deployment needs.

Authoritative Links:

AWS CloudFormation: Official AWS CloudFormation documentation.
AWS CloudFormation Stack Sets: Documentation on CloudFormation Stack Sets.
DynamoDB: Official DynamoDB documentation.

Question: 19

A team of Database Specialists is currently investigating performance issues on an Amazon RDS for MySQL DB instance and is reviewing related metrics. The team wants to narrow the possibilities down to specific database wait events to better understand the situation.
How can the Database Specialists accomplish this?

A. Enable the option to push all database logs to Amazon CloudWatch for advanced analysis
B. Create appropriate Amazon CloudWatch dashboards to contain specific periods of time
C. Enable Amazon RDS Performance Insights and review the appropriate dashboard
D. Enable Enhanced Monitoring will the appropriate settings

Answer: C

Explanation:

The correct answer is C: Enable Amazon RDS Performance Insights and review the appropriate dashboard.

Performance Insights is a database performance monitoring feature that helps you quickly assess the load on your database and determine when and where to take action. Critically, it allows you to visualize database load and filter it by waits, SQL queries, hosts, or users. This makes it ideally suited for pinpointing specific database wait events that are contributing to performance issues. By enabling Performance Insights for the RDS for MySQL DB instance, the team can use its dashboard to identify the wait events causing the bottlenecks, such as I/O waits or lock contention.

Option A is incorrect because while pushing logs to CloudWatch can provide insights, analyzing raw logs is a manual and time-consuming process, especially when dealing with complex wait events. It doesn't offer the direct, aggregated view that Performance Insights provides.

Option B is incorrect because while CloudWatch dashboards are useful, they are for general metrics and do not have the deep database-specific performance analysis capabilities like wait event analysis that Performance Insights offers. CloudWatch metrics will display CPU utilization, memory, etc., but not the underlying wait events causing these issues.

Option D is incorrect because Enhanced Monitoring provides insights into the operating system level of the

RDS instance, like CPU utilization per core, memory usage, and disk I/O. While useful for overall instance health, it doesn't provide the granular database wait event information needed to pinpoint the root cause of database performance bottlenecks, which Performance Insights is specifically designed for. Enhanced Monitoring observes host metrics, not database metrics.

Therefore, Performance Insights directly addresses the need to analyze specific database wait events for performance troubleshooting.

References:

Amazon RDS Performance Insights
Monitoring OS Metrics with Enhanced Monitoring

Question: 20

A large company is using an Amazon RDS for Oracle Multi-AZ DB instance with a Java application. As a part of its disaster recovery annual testing, the company would like to simulate an Availability Zone failure and record how the application reacts during the DB instance failover activity. The company does not want to make any code changes for this activity. What should the company do to achieve this in the shortest amount of time?

A. Use a blue-green deployment with a complete application-level failover test
B. Use the RDS console to reboot the DB instance by choosing the option to reboot with failover
C. Use RDS fault injection queries to simulate the primary node failure
D. Add a rule to the NACL to deny all traffic on the subnets associated with a single Availability Zone

Answer: B

Explanation:

The correct answer is **B. Use the RDS console to reboot the DB instance by choosing the option to reboot with failover.**

Here's why:

Rebooting with failover is the quickest and most straightforward way to simulate an Availability Zone (AZ) failure in an RDS Multi-AZ environment without code changes. When you initiate a reboot with failover, RDS orchestrates a switchover to the standby DB instance in another AZ. This mimics an AZ failure, allowing the company to observe how their application reacts during the failover process. This process leverages the built-in high availability features of RDS Multi-AZ deployments.

Option A, blue-green deployment, involves creating a duplicate environment and switching traffic, which is a more complex and time-consuming approach than necessary for a simple failover test. It involves significant infrastructure and potentially code changes.

Option C, using RDS fault injection queries, is not a standard or readily available feature within Amazon RDS. It implies custom tooling or scripting that is not natively supported and wouldn't be the quickest way to test.

Option D, adding a rule to the NACL to deny all traffic, could lead to unpredictable network behavior and might not accurately simulate an AZ failure. Network outages can differ from a true DB instance failure, potentially leading to inaccurate test results. Additionally, managing NACLs requires careful planning and can be prone to errors. It doesn't directly trigger the RDS failover mechanism.

Rebooting with failover directly exercises the RDS Multi-AZ failover mechanism, providing a more realistic and quicker simulation of an AZ outage.

Supporting Documentation:

RDS Multi-AZ deployments
Rebooting an RDS DB Instance - specifically mentions "Rebooting with Failover" for Multi-AZ instances.

Question: 21

A company maintains several databases using Amazon RDS for MySQL and PostgreSQL. Each RDS database generates log files with retention periods set to their default values. The company has now mandated that database logs be maintained for up to 90 days in a centralized repository to facilitate real-time and after-the-fact analyses.
What should a Database Specialist do to meet these requirements with minimal effort?

A. Create an AWS Lambda function to pull logs from the RDS databases and consolidate the log files in an Amazon S3 bucket. Set a lifecycle policy to expire the objects after 90 days.

B. Modify the RDS databases to publish log to Amazon CloudWatch Logs. Change the log retention policy for each log group to expire the events after 90 days.

C. Write a stored procedure in each RDS database to download the logs and consolidate the log files in an Amazon S3 bucket. Set a lifecycle policy to expire the objects after 90 days.

D. Create an AWS Lambda function to download the logs from the RDS databases and publish the logs to Amazon CloudWatch Logs. Change the log retention policy for the log group to expire the events after 90 days.

Answer: B

Explanation:

The correct answer is B because it offers the most straightforward and integrated solution for centralized log management with minimal operational overhead.

Here's why:

CloudWatch Logs Integration: RDS supports direct integration with CloudWatch Logs for MySQL and PostgreSQL. This allows for real-time log streaming without the need for custom scripting or external tools.

Centralized Repository: CloudWatch Logs acts as a centralized repository, simplifying log analysis and correlation across multiple RDS instances.

Retention Policy: CloudWatch Logs provides configurable retention policies, allowing you to store logs for a specified duration (90 days in this case) without manual intervention.

Minimal Effort: This approach minimizes the effort required as it leverages native AWS services designed for log management. No custom Lambda functions or stored procedures are required.

Why other options are less suitable:

A and D (Lambda + S3/CloudWatch Logs): Using Lambda to pull logs introduces complexity and overhead.
You need to manage Lambda function deployments, scaling, error handling, and security. This adds operational burden compared to the direct CloudWatch Logs integration. Moreover, pulling logs from RDS periodically via Lambda can impact database performance.

C (Stored Procedure + S3): Stored procedures are database-specific and tightly coupled to the database engine. Writing and maintaining stored procedures for each RDS instance is less efficient than using the built-in CloudWatch Logs integration. Moreover, pushing logs to S3 from within the database can affect database performance and security.

Therefore, publishing RDS logs directly to Amazon CloudWatch Logs and configuring the retention policy is the most efficient and manageable solution for centralized log management.

Supporting Documentation:

Publishing MySQL Logs to Amazon CloudWatch Logs:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/mysql_cloudwatch_logs.html
Publishing PostgreSQL Logs to Amazon CloudWatch Logs:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/postgresql_cloudwatch_logs.html
Working with Log Groups and Log Streams in CloudWatch Logs:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Working-with-log-groups-and-streams.html

Question: 22

A Database Specialist is setting up a new Amazon Aurora DB cluster with one primary instance and three Aurora Replicas for a highly intensive, business-critical application. The Aurora DB cluster has one medium-sized primary instance, one large-sized replica, and two medium sized replicas. The Database Specialist did not assign a promotion tier to the replicas. In the event of a primary failure, what will occur?

A. Aurora will promote an Aurora Replica that is of the same size as the primary instance
B. Aurora will promote an arbitrary Aurora Replica
C. Aurora will promote the largest-sized Aurora Replica
D. Aurora will not promote an Aurora Replica

Answer: C

Explanation:

The correct answer is C: Aurora will promote the largest-sized Aurora Replica. Here's why:

When an Aurora DB cluster experiences a primary instance failure and no promotion tier is explicitly set for the replicas, Aurora follows a specific logic to determine which replica to promote as the new primary. The primary factors considered are replica size and then the order in which they are created. Aurora prioritizes promoting the replica that offers the best chance of maintaining application performance and availability. In this scenario, the large-sized replica will be promoted over the medium-sized replicas.

Without promotion tiers, Aurora considers the largest instance size among the replicas, which in this case is the "large-sized replica". Aurora favors larger instance sizes because they generally have more resources (CPU, memory) and can better handle the workload previously managed by the primary instance. This strategy aims to minimize the impact of the failover and ensure continued operation with sufficient capacity. While consistency and data loss are prioritized in the failover process, Aurora attempts to minimize service disruption. If multiple replicas had the same largest size, it would prioritize the replica that is highest in the default order in which the replicas were created within the cluster, or available.

For more detailed information, you can refer to the following AWS documentation:

Amazon Aurora High Availability:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Managing.HighAvailability.html **Fault Tolerance for an Aurora DB Cluster:**
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Managing.FaultTolerance.html

Question: 23

A company is running its line of business application on AWS, which uses Amazon RDS for MySQL at the persistent data store. The company wants to minimize downtime when it migrates the database to Amazon Aurora. Which migration method should a Database Specialist use?

A. Take a snapshot of the RDS for MySQL DB instance and create a new Aurora DB cluster with the option to migrate snapshots.

B. Make a backup of the RDS for MySQL DB instance using the mysqldump utility, create a new Aurora DB cluster, and restore the backup.

C. Create an Aurora Replica from the RDS for MySQL DB instance and promote the Aurora DB cluster.

D. Create a clone of the RDS for MySQL DB instance and promote the Aurora DB cluster.

Answer: C

Explanation:

The correct answer is **C. Create an Aurora Replica from the RDS for MySQL DB instance and promote the Aurora DB cluster.** This method offers the least downtime when migrating from RDS for MySQL to Aurora because it leverages native replication capabilities. Here's why:

Option A (snapshot migration) involves taking a snapshot and creating a new Aurora cluster from it. While this works, it requires downtime while the snapshot is created and the new cluster is built, as the snapshot represents a point-in-time copy.

Option B (mysqldump and restore) also leads to significant downtime. The mysqldump process can take a long time for large databases, and the subsequent restore process to the new Aurora cluster adds more downtime.

Option D (cloning and promotion) is not a standard or recommended method for migrating to Aurora. While cloning can create a copy of the RDS instance, it does not directly integrate with Aurora's replication mechanisms. The process of then promoting this clone to an Aurora cluster is not a streamlined or low-downtime procedure.

Option C, creating an Aurora Replica, is the preferred method due to its minimal downtime. Aurora Replicas use MySQL's binary log replication to continuously stream changes from the source RDS for MySQL instance. This process keeps the Aurora Replica nearly synchronized with the source database. Once the replication lag is acceptable, the Aurora Replica can be promoted to become the primary Aurora DB cluster. The downtime during the promotion is significantly reduced to the time it takes to change the DNS or application connection strings to point to the new Aurora cluster.

This method minimizes downtime by performing the majority of the data transfer in the background while the application continues to use the original RDS instance. Promoting the replica involves a brief outage during the cutover.

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/AuroraMySQL.Migrate.htmlhttps://aws.amazon.com/b your-mysql-databases-to-amazon-aurora-with-minimum-downtime/

Question: 24

The Security team for a finance company was notified of an internal security breach that happened 3 weeks ago. A Database Specialist must start producing audit logs out of the production Amazon Aurora PostgreSQL cluster for the Security team to use for monitoring and alerting. The Security team is required to perform real-time alerting and monitoring outside the Aurora DB cluster and wants to have the cluster push encrypted files to the chosen solution.
Which approach will meet these requirements?

A. Use pg_audit to generate audit logs and send the logs to the Security team.

B. Use AWS CloudTrail to audit the DB cluster and the Security team will get data from Amazon S3.

C. Set up database activity streams and connect the data stream from Amazon Kinesis to consumer applications.

D. Turn on verbose logging and set up a schedule for the logs to be dumped out for the Security team.

Answer: C

Explanation:

The correct answer is C: Set up database activity streams and connect the data stream from Amazon Kinesis to consumer applications. Here's why:

Real-time Monitoring and Alerting: Database Activity Streams (DAS) provide a near real-time stream of database operations, fulfilling the Security team's requirement for real-time alerting and monitoring. **Outside the DB Cluster:** DAS streams data outside of the Aurora PostgreSQL cluster, allowing the Security team to process and analyze the data in their chosen solution without impacting the database performance.

Encryption: DAS supports encryption of the data stream at rest and in transit, aligning with the finance company's security requirements. The data can be encrypted using KMS keys controlled by the organization.

Amazon Kinesis Integration: Amazon Kinesis can be used as a conduit to route the DAS stream to the Security team's consumer applications (e.g., SIEM). Kinesis offers scalability, durability, and the ability to process the stream in real time.

Let's analyze why the other options are less suitable:

A. Use pg_audit to generate audit logs and send the logs to the Security team: While pg_audit can generate audit logs, extracting and transmitting them to the Security team in real-time and in an encrypted manner would require custom scripting and infrastructure, making it less efficient and harder to manage compared to DAS. The real-time requirement is not easily met using native PostgreSQL audit logging and requires more engineering effort.

B. Use AWS CloudTrail to audit the DB cluster and the Security team will get data from Amazon S3: CloudTrail primarily focuses on auditing API calls made to AWS services. It doesn't capture the detailed database operations that are required for comprehensive security monitoring inside the database itself.

CloudTrail captures control plane events, whereas the finance company is interested in data plane events.

D. Turn on verbose logging and set up a schedule for the logs to be dumped out for the Security team: Verbose logging can significantly impact database performance and doesn't provide a real-time stream.

Extracting logs on a schedule would introduce delays, making it unsuitable for immediate alerting and monitoring. Furthermore, manually managing the logs and ensuring encryption adds complexity.

Supporting Documentation:

Database Activity Streams:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/DBActivityStreams.html
Amazon Kinesis Data Streams:https://aws.amazon.com/kinesis/data-streams/

Question: 25

A company is using Amazon RDS for MySQL to redesign its business application. A Database Specialist has noticed that the Development team is restoring their
MySQL database multiple times a day when Developers make mistakes in their schema updates. The Developers sometimes need to wait hours for the restores to complete.
Multiple team members are working on the project, making it difficult to find the correct restore point for each mistake.
Which approach should the Database Specialist take to reduce downtime?

A. Deploy multiple read replicas and have the team members make changes to separate replica instances
B. Migrate to Amazon RDS for SQL Server, take a snapshot, and restore from the snapshot
C. Migrate to Amazon Aurora MySQL and enable the Aurora Backtrack feature
D. Enable the Amazon RDS for MySQL Backtrack feature

Answer: A

Explanation:

While both A and C offer potential solutions, option A, deploying multiple read replicas, emerges as the more practical and readily implementable solution in this scenario, especially considering the specifics of frequent schema update mistakes in a development environment.

Here's a detailed justification:

Why A (Deploy multiple read replicas) is preferred:

Parallel Development & Isolation: Creating multiple read replicas allows each developer to work on their own isolated copy of the database. Developers can make schema changes and test them without impacting other team members or the primary database. If a developer makes a mistake, they can simply revert to the original state of their specific read replica or create a new one from the production instance.

Minimizes Restore Time: Restore operations on the main database can be time-consuming, particularly with large databases. Replicas can be created more quickly than full restores. Each developer has the autonomy to manage their replica, reducing dependency and wait times.

Direct Schema Modification: Read replicas allow for testing modifications that go beyond the database content to also include schema modifications, as specified in the question.

Cost-Effectiveness: Read replicas are a relatively cost-effective solution for development environments compared to migrating databases, which can be complicated, introduce compatibility issues, and be expensive.

Low Complexity to Set Up: Setting up read replicas in RDS for MySQL is a straightforward process. Developers can begin utilizing them quickly after initial configuration.

Why C (Migrate to Aurora MySQL and enable Aurora Backtrack) is less preferred:

Migration Overhead: Migrating from RDS for MySQL to Aurora MySQL involves a significant migration effort, including compatibility testing and potential application code changes. This is an unnecessary complication, especially for the short-term need of facilitating faster developer rollback. This is not the primary goal and it does not address the problem.

Backtrack Limitations: While Aurora Backtrack is a valuable feature for rewinding a database to a specific point in time, it might not be granular enough for individual developer mistakes, especially if multiple developers are working concurrently. Backtrack is a single, shared "undo" feature.

Potentially Slower for Schema Changes: Backtrack is optimized for data-level rollbacks and may not be as efficient for reversing schema changes involving large table modifications or complex database operations.

Increased Complexity and Cost: Aurora MySQL can be more expensive than RDS for MySQL, and implementing and managing Aurora Backtrack adds further complexity.

In summary:

While Aurora Backtrack can be a solution, it involves migration costs, potential application compatibility problems and is a single undo function. Deploying multiple read replicas is a far simpler and more targeted solution to reduce downtime, especially during the development phase where multiple developers might be concurrently making changes that occasionally require a rollback. It allows for isolated environments and quick reverts, directly addressing the problem of time-consuming restores.

Supporting Links:

Amazon RDS Read Replicas:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadRepl.html
Amazon Aurora Backtrack:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Managing.Backtrack.html

Question: 26

A media company is using Amazon RDS for PostgreSQL to store user data. The RDS DB instance currently has a publicly accessible setting enabled and is hosted in a public subnet. Following a recent AWS Well-Architected Framework review, a Database Specialist was given new security requirements.
☞ Only certain on-premises corporate network IPs should connect to the DB instance.
☞ Connectivity is allowed from the corporate network only.
Which combination of steps does the Database Specialist need to take to meet these new requirements? (Choose three.)

A. Modify the pg_hba.conf file. Add the required corporate network IPs and remove the unwanted IPs.

B. Modify the associated security group. Add the required corporate network IPs and remove the unwanted IPs.

C. Move the DB instance to a private subnet using AWS DMS.

D. Enable VPC peering between the application host running on the corporate network and the VPC associated with the DB instance.

E. Disable the publicly accessible setting.

F. Connect to the DB instance using private IPs and a VPN.

Answer: BEF

Explanation:

The correct answer is BEF. Here's a detailed justification:

B. Modify the associated security group. Add the required corporate network IPs and remove the unwanted IPs: Security groups act as virtual firewalls controlling inbound and outbound traffic for your RDS instance. By modifying the security group, you can restrict access to the RDS instance to only the specified on-premises corporate network IP addresses. This is a fundamental network security practice.

https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

E. Disable the publicly accessible setting: Enabling the publicly accessible setting on an RDS instance allows it to be accessed from the internet. Disabling this setting ensures that the RDS instance can only be accessed from within the VPC or through allowed network connections like VPN. This prevents unauthorized access from the public internet.

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_CommonTasks.Connect.html

F. Connect to the DB instance using private IPs and a VPN: To allow connectivity from the corporate network (which is now external to the VPC), a secure tunnel is required. Establishing a VPN connection between the on-premises network and the VPC enables secure communication using private IP addresses, protecting data in transit. Because of that, RDS instance is now in private subnet as a result of disabling public accessibility.

https://docs.aws.amazon.com/vpn/latest/s2svpn/VPC_VPN.html

Why the other options are incorrect:

A. Modify the pg_hba.conf file. Add the required corporate network IPs and remove the unwanted IPs: While pg_hba.conf is used for authentication, it's better practice to control network access at the security group level, especially in a cloud environment like AWS. Security groups are the first line of defense. Modifying both Security Groups and pg_hba.conf could add a layer of redundancy, but modifying only pg_hba.conf while leaving security groups open is insufficient.

C. Move the DB instance to a private subnet using AWS DMS: Moving the database to a private subnet is a good security practice, but AWS DMS is not the appropriate tool for this task. AWS DMS is used for database migration between different database engines or versions, not for relocating a database within the same engine and environment. The database will be in a private subnet as a result of disabling public accessibility.

D. Enable VPC peering between the application host running on the corporate network and the VPC

associated with the DB instance: VPC peering connects two VPCs, not an on-premises network to a VPC. While VPC peering would be used if you had another VPC that needed access, in this case, the connection is from an on-premises network, requiring a VPN or Direct Connect.

Question: 27

A company is about to launch a new product, and test databases must be re-created from production data. The company runs its production databases on an
Amazon Aurora MySQL DB cluster. A Database Specialist needs to deploy a solution to create these test databases as quickly as possible with the least amount of administrative effort.
What should the Database Specialist do to meet these requirements?

A. Restore a snapshot from the production cluster into test clusters

B. Create logical dumps of the production cluster and restore them into new test clusters

C. Use database cloning to create clones of the production cluster

D. Add an additional read replica to the production cluster and use that node for testing

Answer: C

Explanation:

The correct answer is **C: Use database cloning to create clones of the production cluster.**

Here's why:

Speed and Efficiency: Database cloning is the fastest and most efficient method for creating test databases from production data. It creates a point-in-time copy of the database without moving actual data blocks, significantly reducing the time required compared to restoring snapshots or logical dumps.

Minimal Administrative Effort: Cloning minimizes administrative overhead because it's a managed feature within Aurora MySQL. The process is initiated with a simple API call or console action, requiring minimal configuration.

Aurora's Cloning Feature: Aurora specifically supports database cloning. Aurora Cloning quickly makes an exact copy of your database at a specific point in time.

Snapshot Restoration (A): While restoring a snapshot is a valid method, it typically takes longer than cloning, especially for large databases, because it involves creating new storage volumes and restoring data onto them.

Logical Dumps (B): Creating logical dumps and restoring them is the slowest option. It involves exporting data into a file and importing it into the new database, which is a time-consuming process and places a load on the production database during the export.

Read Replica (D): Adding a read replica doesn't directly address the requirement of creating test databases.

Read replicas are primarily for read-heavy workloads and high availability, not for creating isolated test environments. Furthermore, testing on a read replica that is still connected to the production database could potentially cause data corruption or other unintended consequences.

In conclusion, database cloning is the ideal solution because it provides the fastest and most efficient way to create test databases from production data with the least amount of administrative effort, perfectly aligning with the requirements outlined in the prompt.

Authoritative Links:

Amazon Aurora Cloning
Cloning DB clusters in Aurora

Question: 28

A company with branch offices in Portland, New York, and Singapore has a three-tier web application that leverages a shared database. The database runs on
Amazon RDS for MySQL and is hosted in the us-west-2 Region. The application has a distributed front end deployed in the us-west-2, ap-southheast-1, and us- east-2 Regions.
This front end is used as a dashboard for Sales Managers in each branch office to see current sales statistics. There are complaints that the dashboard performs more slowly in the Singapore location than it does in Portland or New York. A solution is needed to provide consistent performance for all users in each location.
Which set of actions will meet these requirements?

A. Take a snapshot of the instance in the us-west-2 Region. Create a new instance from the snapshot in the ap-southeast-1 Region. Reconfigure the ap- southeast-1 front-end dashboard to access this instance.

B. Create an RDS read replica in the ap-southeast-1 Region from the primary RDS DB instance in the us-west-2 Region. Reconfigure the ap-southeast-1 front- end dashboard to access this instance.

C. Create a new RDS instance in the ap-southeast-1 Region. Use AWS DMS and change data capture (CDC) to update the new instance in the ap-southeast-1 Region. Reconfigure the ap-southeast-1 front-end dashboard to access this instance.

D. Create an RDS read replica in the us-west-2 Region where the primary instance resides. Create a read replica in the ap-southeast-1 Region from the read replica located on the us-west-2 Region. Reconfigure the ap-southeast-1 front-end dashboard to access this instance.

Answer: B

Explanation:

The correct answer is B: "Create an RDS read replica in the ap-southeast-1 Region from the primary RDS DB instance in the us-west-2 Region. Reconfigure the ap-southeast-1 front-end dashboard to access this instance."

Here's a detailed justification:

The problem is slow dashboard performance for users in Singapore (ap-southeast-1 region) due to the latency of accessing the database in us-west-2. The solution needs to provide consistent performance, implying lower latency access to the database for the Singapore users.

Option B addresses this by creating an RDS read replica in the ap-southeast-1 region. RDS read replicas provide asynchronous replication from the primary database instance. This means data changes in the primary database in us-west-2 are copied to the read replica in ap-southeast-1. By reconfiguring the front-end dashboard in ap-southeast-1 to access the local read replica, the latency for database queries is significantly reduced, leading to improved performance for Singapore users. Read replicas are designed for read-heavy workloads, which aligns with the dashboard application's use case of displaying sales statistics (presumably through read operations). Read replicas are managed by RDS, simplifying maintenance.

Option A is not ideal because creating a new instance from a snapshot creates a standalone database that is not synchronized with the primary. While it puts the data closer, it doesn't keep the data current, negating the dashboard's purpose of showing current sales statistics.

Option C, using AWS DMS with CDC, is more complex than necessary for this scenario. While it does allow for replication, RDS read replicas provide a simpler, more directly integrated solution within the RDS ecosystem.
DMS also incurs additional costs. The extra complexity and cost are unnecessary for the simple goal of improving read performance in a remote region.

Option D introduces an unnecessary intermediate read replica. Creating a read replica from another read replica does not fundamentally improve performance and adds extra complexity. It is best to create read replicas directly from the primary database.In conclusion, creating an RDS read replica in the ap-southeast-1 region and reconfiguring the Singapore front-end to use it provides a simple, cost-effective, and performant solution to the problem of high latency.

Supporting Documentation:

Amazon RDS Read Replicas:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadRepl.html **AWS Database Migration Service (DMS):**https://aws.amazon.com/dms/

Question: 29

A company wants to migrate its existing on-premises Oracle database to Amazon Aurora PostgreSQL. The migration must be completed with minimal downtime using AWS DMS. A Database Specialist must validate that the data was migrated accurately from the source to the target before the cutover. The migration must have minimal impact on the performance of the source database.
Which approach will MOST effectively meet these requirements?

A. Use the AWS Schema Conversion Tool (AWS SCT) to convert source Oracle database schemas to the target Aurora DB cluster. Verify the datatype of the columns.

B. Use the table metrics of the AWS DMS task created for migrating the data to verify the statistics for the tables being migrated and to verify that the data definition language (DDL) statements are completed.

C. Enable the AWS Schema Conversion Tool (AWS SCT) premigration validation and review the premigration checklist to make sure there are no issues with the conversion.

D. Enable AWS DMS data validation on the task so the AWS DMS task compares the source and target records, and reports any mismatches.

Answer: D

Explanation:

Here's a detailed justification for why option D is the most effective solution:

Option D: Enable AWS DMS data validation on the task

This approach directly addresses the core requirement of validating data accuracy after the migration with minimal downtime. AWS DMS data validation compares the source and target records during the migration process. This automated validation pinpoints any discrepancies that may arise during the data transfer, such as data loss or data type conversion errors. This allows for a focused effort on resolving specific data inconsistencies rather than relying on general metrics or pre-migration assessments. This minimizes the cutover time.

The validation happens concurrently with the migration, reducing the impact on the source database. It provides a granular level of validation, ensuring data integrity at the record level. By reporting mismatches, it provides actionable insights for remediation. Data validation confirms data integrity for database migrations or replications done using AWS DMS.
[https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Validating.html]

Why other options are less effective:

Option A: While AWS SCT helps convert schemas, verifying the datatype alone doesn't guarantee data integrity. Data corruption or truncation can still occur without proper validation.

Option B: Table metrics and DDL statement completion verification provide information about the overall migration but don't confirm the accuracy of the actual data transferred. These metrics don't pinpoint specific record-level discrepancies.

Option C: AWS SCT pre-migration validation is useful for identifying potential conversion issues before the migration, but it doesn't validate the data after the migration. It checks compatibility and potential issues but doesn't compare the actual data.

In summary, Option D (AWS DMS data validation) provides the most direct and effective way to validate data accuracy after the migration, ensuring that the target database contains the correct data with minimal downtime. It leverages the capabilities of DMS to provide detailed comparison of data and allows for faster remediation of data errors.

Question: 30

A company is planning to close for several days. A Database Specialist needs to stop all applications along with the DB instances to ensure employees do not have access to the systems during this time. All databases are running on Amazon RDS for MySQL.
The Database Specialist wrote and ran a script to stop all the DB instances. When reviewing the logs, the Database Specialist found that Amazon RDS DB instances with read replicas did not stop.
How should the Database Specialist edit the script to fix this issue?

A. Stop the source instances before stopping their read replicas
B. Delete each read replica before stopping its corresponding source instance
C. Stop the read replicas before stopping their source instances
D. Use the AWS CLI to stop each read replica and source instance at the same time

Answer: B

Explanation:

The correct answer is **B: Delete each read replica before stopping its corresponding source instance.**

Here's a detailed justification:

When stopping RDS DB instances, particularly with read replicas, the order of operations matters due to replication dependencies. Read replicas are copies of the primary/source DB instance and asynchronously receive updates. Therefore, they are dependent on the source instance.

Option A, stopping the source instances before the read replicas, is incorrect because the read replicas rely on the source instance to function. Stopping the source first will likely cause the read replicas to encounter errors and potentially go into a state where they cannot be cleanly stopped.

Option B, deleting the read replicas before stopping the source instance, is the correct approach. By deleting the read replicas first, you break the replication chain. The source instance can then be stopped without being hindered by dependencies on its replicas. The read replicas, once deleted, are no longer dependent on the source. This approach ensures a clean shutdown process.

Option C, stopping the read replicas before their source instances, seems logical on the surface, but stopping a read replica doesn't fundamentally alter the dependency relationship. The replication configuration still exists, and the source database expects the replica to be available (or at least acknowledge its absence).

Option D, stopping read replicas and source instances simultaneously using the AWS CLI, might work in some scenarios. However, it introduces a race condition. The read replicas might still attempt to replicate from the source during the shutdown, leading to inconsistencies or failed shutdowns. Simultaneously stopping via the CLI doesn't guarantee the desired order and introduces unneeded complexity. Also, just "stopping" replicas will only stop them if the source is still up.

In essence, deleting the read replicas severs the dependency, allowing the source instance to be safely stopped afterward. Stopping or attempting to stop the source before deleting its replicas can lead to issues due to their inherent replication relationship. Deleting is a more forceful way to handle the replicas and ensures the desired outcome of fully removing the database access.

Therefore, the Database Specialist should modify the script to first delete all read replicas associated with the DB instances before attempting to stop the corresponding source DB instances. This strategy guarantees that all database access is successfully stopped, fulfilling the requirement of preventing employee access during the company's closure.

For further reading and confirmation of these concepts, refer to the AWS documentation:

Working with Read Replicas:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadRepl.html **Stopping and Starting an Amazon RDS DB Instance**:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_StopInstance.html

Question: 31

A global digital advertising company captures browsing metadata to contextually display relevant images, pages, and links to targeted users. A single page load can generate multiple events that need to be stored individually.
The maximum size of an event is 200 KB and the average size is 10 KB. Each page load must query the user's browsing history to provide targeting recommendations. The advertising company expects over 1 billion page visits per day from users in the United States, Europe, Hong Kong, and India. The structure of the metadata varies depending on the event.
Additionally, the browsing metadata must be written and read with very low latency to ensure a good viewing experience for the users.
Which database solution meets these requirements?

A. Amazon DocumentDB
B. Amazon RDS Multi-AZ deployment
C. Amazon DynamoDB global table
D. Amazon Aurora Global Database

Answer: C

Explanation:

The correct answer is **C. Amazon DynamoDB global table**. Here's why:

High Throughput and Low Latency: The application requires handling over 1 billion page visits daily, each generating multiple events, demanding extremely high write throughput and low latency reads for a good user experience. DynamoDB is specifically designed for these workloads. It offers consistent, single-digit millisecond latency at any scale. https://aws.amazon.com/dynamodb/

Global Distribution: The application's users are spread across the US, Europe, Hong Kong, and India. DynamoDB Global Tables provide automatic, multi-region replication, enabling low-latency reads and writes for users in different geographical locations. This ensures a consistent and fast experience for users regardless of their location.
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/globaltables.html

Flexible Schema: The metadata structure varies depending on the event. DynamoDB is a NoSQL database that provides a flexible schema, allowing it to accommodate different data structures without requiring predefined schemas, making it suitable for varying event metadata.

Scalability: DynamoDB is a fully managed, serverless database that automatically scales to handle the required throughput. This eliminates the need for manual scaling and provisioning, simplifying operations.

Event Size Compatibility: The event sizes (200KB max, 10KB average) are well within DynamoDB's item size limits (400KB).

Now, let's look at why the other options are less suitable:

A. Amazon DocumentDB: While DocumentDB is suitable for document storage, it is not designed for the massive scale and low latency requirements of this application. It's optimized for MongoDB workloads, not high-velocity, globally distributed data.

B. Amazon RDS Multi-AZ deployment: RDS is a relational database service. While Multi-AZ provides high availability, it does not inherently offer global distribution or the flexible schema needed. Scaling RDS to handle 1 billion visits with low latency would be complex and expensive. Relational databases typically are not as performant as NoSQL databases at this scale.

D. Amazon Aurora Global Database: Aurora Global Database is a great choice for global read replicas, but since the requirement specifies both writes and reads need to be extremely low latency, this option is not optimal. DynamoDB Global Tables are specifically architected for consistent low latency across regions for both reads and writes.

Question: 32

A Database Specialist modified an existing parameter group currently associated with a production Amazon RDS for SQL Server Multi-AZ DB instance. The change is associated with a static parameter type, which controls the number of user connections allowed on the most critical RDS SQL Server DB instance for the company. This change has been approved for a specific maintenance window to help minimize the impact on users.
How should the Database Specialist apply the parameter group change for the DB instance?

A. Select the option to apply the change immediately

B. Allow the preconfigured RDS maintenance window for the given DB instance to control when the change is applied

C. Apply the change manually by rebooting the DB instance during the approved maintenance window

D. Reboot the secondary Multi-AZ DB instance

Answer: C

Explanation:

The correct answer is C: Apply the change manually by rebooting the DB instance during the approved maintenance window. Here's why:

Static parameters in RDS require a reboot of the DB instance for changes to take effect. Unlike dynamic parameters that can be applied immediately, static parameters are loaded only when the instance starts. Therefore, simply waiting for the preconfigured RDS maintenance window (option B) or selecting the "apply immediately" option (option A) will not apply the change to the static parameter.

The question specifies that the modified parameter controls the number of user connections, which directly affects the availability and performance of the critical production database. It is crucial to apply this change during a pre-approved maintenance window to minimize user impact. Therefore, a deliberate, controlled reboot is required.

Rebooting only the secondary Multi-AZ instance (option D) will not propagate the static parameter change. When a failover occurs, the newly promoted primary instance will still retain the old parameter value. Thus, to apply the change, a reboot of the primary DB instance is necessary. In a Multi-AZ setup, this will also trigger a failover to the secondary, resulting in a short period of unavailability, which must be scheduled and communicated.

Therefore, the only method to ensure the static parameter change takes effect on a Multi-AZ RDS SQL Server instance, with minimal impact, is to manually reboot the DB instance (thereby triggering a failover) during the

approved maintenance window. This ensures the updated parameter takes effect during the restart and minimizes disruption to users.

Further research:

AWS RDS Parameter Groups:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html
(Specifically, the section on static vs. dynamic parameters)
AWS RDS Rebooting:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_RebootInstance.html
AWS RDS Multi-AZ:https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html

Question: 33

A Database Specialist is designing a new database infrastructure for a ride hailing application. The application data includes a ride tracking system that stores
GPS coordinates for all rides. Real-time statistics and metadata lookups must be performed with high throughput and microsecond latency. The database should be fault tolerant with minimal operational overhead and development effort. Which solution meets these requirements in the MOST efficient way?

A. Use Amazon RDS for MySQL as the database and use Amazon ElastiCache
B. Use Amazon DynamoDB as the database and use DynamoDB Accelerator
C. Use Amazon Aurora MySQL as the database and use Aurora's buffer cache
D. Use Amazon DynamoDB as the database and use Amazon API Gateway

Answer: B

Explanation:

The correct answer is **B. Use Amazon DynamoDB as the database and use DynamoDB Accelerator (DAX)**. Here's a detailed justification:

DynamoDB is a fully managed NoSQL database service that provides high throughput and low latency performance, making it ideal for applications requiring microsecond response times for high-volume requests, like the ride-tracking system's GPS coordinate storage and real-time statistics. Its fully managed nature minimizes operational overhead.

DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB. DAX delivers up to a 10x performance improvement – from milliseconds to microseconds – even at millions of requests per second. This is crucial for the required microsecond latency. It eliminates the need for developers to manage the caching infrastructure, further reducing operational overhead and development effort.

Option A (RDS for MySQL with ElastiCache) can achieve low latency, but MySQL (a relational database) might not be the most efficient choice for storing and querying GPS coordinates at scale compared to DynamoDB's NoSQL capabilities. Additionally, managing both RDS and ElastiCache adds operational complexity. While ElastiCache would help with caching, the underlying relational database might still become a bottleneck.

Option C (Aurora MySQL with Aurora's buffer cache) improves performance through caching, but doesn't scale as efficiently as DynamoDB for high-throughput, low-latency requirements. Aurora's buffer cache is a database level cache, whereas DAX sits as a layer in front of DynamoDB and reduces the number of direct DynamoDB reads.

Option D (DynamoDB with API Gateway) is incomplete; while API Gateway is useful for managing and securing API endpoints, it doesn't provide caching or accelerate database reads in the same way as DAX. API Gateway

primarily handles API request routing, authentication, and authorization. It isn't a caching solution tailored to DynamoDB's specific needs.

Therefore, DynamoDB with DAX provides the best combination of scalability, low latency, minimal operational overhead, and reduced development effort for this ride-hailing application's requirements.

Authoritative Links:

Amazon DynamoDB:https://aws.amazon.com/dynamodb/
DynamoDB Accelerator (DAX):https://aws.amazon.com/dynamodb/dax/

Question: 34

A company is using an Amazon Aurora PostgreSQL DB cluster with an xlarge primary instance master and two large Aurora Replicas for high availability and read-only workload scaling. A failover event occurs and application performance is poor for several minutes. During this time, application servers in all Availability
Zones are healthy and responding normally.
What should the company do to eliminate this application performance issue?

A. Configure both of the Aurora Replicas to the same instance class as the primary DB instance. Enable cache coherence on the DB cluster, set the primary DB instance failover priority to tier-0, and assign a failover priority of tier-1 to the replicas.

B. Deploy an AWS Lambda function that calls the DescribeDBInstances action to establish which instance has failed, and then use the PromoteReadReplica operation to promote one Aurora Replica to be the primary DB instance. Configure an Amazon RDS event subscription to send a notification to an Amazon SNS topic to which the Lambda function is subscribed.

C. Configure one Aurora Replica to have the same instance class as the primary DB instance. Implement Aurora PostgreSQL DB cluster cache management. Set the failover priority to tier-0 for the primary DB instance and one replica with the same instance class. Set the failover priority to tier-1 for the other replicas.

D. Configure both Aurora Replicas to have the same instance class as the primary DB instance. Implement Aurora PostgreSQL DB cluster cache management. Set the failover priority to tier-0 for the primary DB instance and to tier-1 for the replicas.

Answer: C

Explanation:

The correct answer is C because it addresses the root causes of the observed performance degradation following a failover: instance size mismatch and cache invalidation.

Here's a detailed justification:

1. **Instance Size Mismatch:** The original setup has the primary instance (xlarge) significantly larger than the Aurora Replicas (large). During a failover, one of the large replicas becomes the new primary. This instance downgrade immediately leads to reduced compute and memory resources, causing the application to experience performance issues. Upgrading one replica to xlarge aligns its capacity with the original primary, ensuring a smoother transition and minimizing the impact of failover on performance. The other replica can remain a "large" for cost optimization while maintaining a high-availability footprint.

2. **Cache Warm-up:** After a failover, the newly promoted instance's buffer pool (cache) is cold. This means that frequently accessed data must be read from disk, severely impacting performance.

   Implementing Aurora PostgreSQL DB cluster cache management (or "warm-up") is essential to address this. This can be done by regularly loading frequently accessed data into the replica's buffer pool using tools or queries. When the replica takes over as primary, the warmed-up cache will significantly reduce the performance penalty.

3. **Failover Priority:** Aurora uses a failover priority system to determine which replica to promote first. Setting the primary to tier-0 and one xlarge replica to tier-0 ensures that, whenever possible, that similarly sized instance is promoted. Setting the remaining "large" replica to tier-1 helps improve availability, and prioritizes speed of failover rather than optimized compute capacity after failover.

Options A and D incorrectly advocate for scaling both replicas to xlarge. While this would provide better performance during failover, it introduces unnecessary cost when the replicas are only used for read workloads and potentially only for HA. Option B is an overly complex approach using Lambda to manually promote a replica. Aurora's built-in failover mechanism is much more efficient and reliable.

Here are some helpful links for further study:

Aurora Failover:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Concepts.AuroraHighAvailability.html **Aurora Replica Management:**
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Managing.html
Amazon RDS Event Notifications:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Events.html
Understanding Aurora PostgreSQL buffer pool warmup: https://aws.amazon.com/blogs/database/reduce-postgresql-restart-times-with-the-auto-shared-preload-library/

Question: 35

A company has a database monitoring solution that uses Amazon CloudWatch for its Amazon RDS for SQL Server environment. The cause of a recent spike in
CPU utilization was not determined using the standard metrics that were collected. The CPU spike caused the application to perform poorly, impacting users. A
Database Specialist needs to determine what caused the CPU spike.
Which combination of steps should be taken to provide more visibility into the processes and queries running during an increase in CPU load? (Choose two.)

A. Enable Amazon CloudWatch Events and view the incoming T-SQL statements causing the CPU to spike.
B. Enable Enhanced Monitoring metrics to view CPU utilization at the RDS SQL Server DB instance level.
C. Implement a caching layer to help with repeated queries on the RDS SQL Server DB instance.
D. Use Amazon QuickSight to view the SQL statement being run.
E. Enable Amazon RDS Performance Insights to view the database load and filter the load by waits, SQL statements, hosts, or users.

Answer: BE

Explanation:

The correct answer is **B and E**. Here's a detailed justification:

B. Enable Enhanced Monitoring metrics to view CPU utilization at the RDS SQL Server DB instance level.

Justification: Standard CloudWatch metrics provide a high-level overview of RDS instance performance, including CPU utilization. However, they lack the granularity needed to pinpoint the specific processes or queries consuming the most CPU. Enhanced Monitoring, on the other hand, provides OS-level metrics, including CPU utilization broken down by process. This allows the Database Specialist to see which processes within the SQL Server instance are contributing most to the CPU load. This is essential for diagnosing performance bottlenecks and identifying resource-intensive activities. Enhanced Monitoring metrics are collected at a lower level and with a finer granularity (as low as 1 second) than standard CloudWatch metrics, providing much greater insight.
Relevance: Identifying resource-intensive processes is crucial for understanding why CPU utilization spiked.

Authoritative Link:https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Monitoring.OS.html

E. Enable Amazon RDS Performance Insights to view the database load and filter the load by waits, SQL statements, hosts, or users.

Justification: Amazon RDS Performance Insights is a database performance monitoring and tuning tool that helps you quickly assess the load on your database and determine when and where to take action. It visualizes database load as the Average Active Sessions metric and allows you to drill down to see which SQL statements, waits, hosts, or users are contributing most to the load. This is invaluable for pinpointing the specific queries causing the CPU spike. By filtering by SQL statements, you can directly identify the problematic T-SQL that caused the high CPU usage. Performance Insights allows for proactive performance optimization and real-time problem identification.

Relevance: Allows identifying the exact SQL statement causing the high CPU load and the factors involved.

Authoritative Link:https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PerfInsights.html

Why the other options are incorrect:

A. Enable Amazon CloudWatch Events and view the incoming T-SQL statements causing the CPU to spike. CloudWatch Events reacts to changes in AWS resources and system events. It does not directly capture and analyze the T-SQL statements being executed by RDS in a way that would allow for immediate root cause analysis of a CPU spike. While events related to the spike might be captured, they wouldn't reveal the specific queries causing it.

C. Implement a caching layer to help with repeated queries on the RDS SQL Server DB instance. While caching is a good general practice for improving database performance, implementing it after a CPU spike doesn't help determine the cause of the spike. Caching addresses a symptom, not the root problem. This is a reactive, rather than diagnostic, approach.

D. Use Amazon QuickSight to view the SQL statement being run. QuickSight is a business intelligence service for data visualization and analysis. While QuickSight can analyze data from RDS, it's not designed for real-time monitoring of SQL statement execution or identification of performance bottlenecks within the database itself. You need the low level metrics provided by Enhanced Monitoring and the detailed analysis tools within Performance Insights.

Question: 36

A company is using Amazon with Aurora Replicas for read-only workload scaling. A Database Specialist needs to split up two read-only applications so each application always connects to a dedicated replica. The Database Specialist wants to implement load balancing and high availability for the read-only applications.
Which solution meets these requirements?

A. Use a specific instance endpoint for each replica and add the instance endpoint to each read-only application connection string.

B. Use reader endpoints for both the read-only workload applications.

C. Use a reader endpoint for one read-only application and use an instance endpoint for the other read-only application.

D. Use custom endpoints for the two read-only applications.

Answer: D

Explanation:

The correct answer is **D. Use custom endpoints for the two read-only applications.**

Here's why:

Custom endpoints in Amazon Aurora allow you to define specific subsets of your Aurora Replica instances to

serve particular applications. By creating a custom endpoint for each read-only application, you guarantee that traffic from each application is routed exclusively to its designated replica(s). This achieves the goal of splitting the read-only workloads and dedicating replicas to each application.

Furthermore, custom endpoints support load balancing by distributing connections among the instances associated with the endpoint. If you add multiple replicas to a custom endpoint, Aurora automatically balances the load across those replicas.

High availability is also provided through custom endpoints. If one replica associated with the custom endpoint becomes unavailable, Aurora automatically fails over to another available replica in that endpoint. This ensures continuous service for the read-only applications.

Options A, B, and C are not suitable:

A. Use a specific instance endpoint for each replica and add the instance endpoint to each read-only application connection string: While this dedicates a replica to each application, it lacks load balancing and automatic failover. If the specific replica fails, the application connected to that instance endpoint will also fail.

B. Use reader endpoints for both the read-only workload applications: Reader endpoints automatically balance connections across all available replicas. This will not satisfy the requirement to split the applications so each application always connects to a dedicated replica.

C. Use a reader endpoint for one read-only application and use an instance endpoint for the other read-only application: This mixes the two strategies and introduces the drawbacks of both. One application would be load-balanced across all replicas (not dedicated), while the other lacks load balancing and high availability (single instance endpoint).

In summary, custom endpoints provide the granular control, load balancing, and high availability needed to meet the specific requirements of the scenario. They allow you to isolate read workloads and ensure the resilience of each application.

Further Research:

Amazon Aurora Custom Endpoints

Question: 37

An online gaming company is planning to launch a new game with Amazon DynamoDB as its data store. The database should be designated to support the following use cases:
☞ Update scores in real time whenever a player is playing the game.
☞ Retrieve a player's score details for a specific game session.
A Database Specialist decides to implement a DynamoDB table. Each player has a unique user_id and each game has a unique game_id.
Which choice of keys is recommended for the DynamoDB table?

A. Create a global secondary index with game_id as the partition key
B. Create a global secondary index with user_id as the partition key
C. Create a composite primary key with game_id as the partition key and user_id as the sort key
D. Create a composite primary key with user_id as the partition key and game_id as the sort key

Answer: D

Explanation:

The recommended key structure for the DynamoDB table is a composite primary key with user_id as the

partition key and game_id as the sort key (Option D).

This design effectively addresses the two specified use cases. Using user_id as the partition key allows for efficient retrieval of all game sessions for a specific player, enabling easy access to a player's score details across different games. The sort key game_id then allows efficient querying for a specific game session within that user's data. DynamoDB queries become highly targeted.

For updating scores in real-time, a composite primary key provides efficient write access. Updates will be directed to the correct partition based on the user_id, and then located by game_id.

Option A is incorrect because creating a global secondary index (GSI) with game_id as the partition key would duplicate data and might not be the most efficient choice for retrieving all game sessions for a specific user.
While possible, it is less direct than the composite key approach. Option B suffers the same drawback with user_id as the partition key in the GSI and doesn't cater for retriving a specific game session easily, which is a key requirement.

Option C is less efficient because it would require knowing the game_id first to find the user_id and is not ideal for retriving data for specific users across multiple games. The problem statement emphasizes retrieving a player's score details for a specific game session, thereby prioritizing player-centric queries.

In summary, a composite primary key with user_id as the partition key and game_id as the sort key optimizes both real-time score updates and retrieval of a player's score details for a specific game session.

Further Research:

DynamoDB Primary Keys
DynamoDB Global Secondary Indexes

Question: 38

A Database Specialist migrated an existing production MySQL database from on-premises to an Amazon RDS for MySQL DB instance. However, after the migration, the database needed to be encrypted at rest using AWS KMS. Due to the size of the database, reloading, the data into an encrypted database would be too time-consuming, so it is not an option.
How should the Database Specialist satisfy this new requirement?

A. Create a snapshot of the unencrypted RDS DB instance. Create an encrypted copy of the unencrypted snapshot. Restore the encrypted snapshot copy.
B. Modify the RDS DB instance. Enable the AWS KMS encryption option that leverages the AWS CLI.
C. Restore an unencrypted snapshot into a MySQL RDS DB instance that is encrypted.
D. Create an encrypted read replica of the RDS DB instance. Promote it the master.

Answer: A

Explanation:

The correct answer is **A: Create a snapshot of the unencrypted RDS DB instance. Create an encrypted copy of the unencrypted snapshot. Restore the encrypted snapshot copy.**

Here's a detailed justification:

The core problem is enabling encryption at rest on an existing unencrypted RDS for MySQL instance without incurring a prolonged downtime associated with data reloading. RDS for MySQL does not support in-place encryption of existing unencrypted databases. The only supported way to achieve this is to create a snapshot, encrypt a copy of the snapshot, and then restore the encrypted snapshot to a new RDS instance.

Option A directly addresses this limitation of RDS for MySQL. Taking a snapshot creates a point-in-time backup of the database. Crucially, RDS allows you to create an encrypted copy of an unencrypted snapshot, leveraging AWS KMS. This copy operation is where the encryption happens. Finally, restoring the encrypted snapshot creates a new, encrypted RDS instance containing the data. This approach minimizes downtime compared to alternative methods like reloading the database.

Option B is incorrect because RDS does not offer a direct modification option to enable KMS encryption on an existing unencrypted instance. There's no AWS CLI command or console option to directly "flip a switch" for encryption on an unencrypted DB instance.

Option C is technically possible (restoring an unencrypted snapshot into an encrypted instance), but it doesn't solve the problem of data being unencrypted at rest. The new RDS instance would be encrypted, but the data restored from the unencrypted snapshot would remain unencrypted.

Option D, creating an encrypted read replica, is a viable strategy for migrating to encryption, but requires a final failover. More to the point, creating an encrypted read replica first requires the base instance to support it. RDS does not support creating encrypted replicas from unencrypted instances. The instance must be taken offline and upgraded to support the encrypted read replica creation which is akin to recreating the instance. The snapshot approach detailed in option A is a more direct and preferred method as well as being the official Amazon recommendation.

Therefore, option A is the most efficient and directly addresses the constraints of the problem (avoiding lengthy downtime) while adhering to RDS's encryption capabilities. It's the only option that successfully encrypts the database at rest using supported RDS features with minimal disruption.

Supporting documentation:

RDS Encryption:https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html (Specifically, the section on encrypting existing unencrypted DB instances.)
Creating an Encrypted RDS Snapshot:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CopySnapshot.html#USER_CopySnapshot.Enc
(Describes creating encrypted copies of snapshots.)

Question: 39

A Database Specialist is planning to create a read replica of an existing Amazon RDS for MySQL Multi-AZ DB instance. When using the AWS Management
Console to conduct this task, the Database Specialist discovers that the source RDS DB instance does not appear in the read replica source selection box, so the read replica cannot be created.
What is the most likely reason for this?

A. The source DB instance has to be converted to Single-AZ first to create a read replica from it.
B. Enhanced Monitoring is not enabled on the source DB instance.
C. The minor MySQL version in the source DB instance does not support read replicas.
D. Automated backups are not enabled on the source DB instance.

Answer: D

Explanation:

The most likely reason the RDS for MySQL Multi-AZ DB instance isn't appearing as a source for a read replica is that automated backups are not enabled on the source instance. Read replicas in RDS rely on the underlying database's ability to create and stream changes, which is facilitated through transaction logs.
These logs are maintained and made available precisely because automated backups are turned on.

When automated backups are disabled, RDS doesn't actively retain the transaction logs necessary for creating and maintaining a read replica. The read replica initialization process involves taking a snapshot of the source database and then applying subsequent changes from the transaction logs. Without automated backups enabled (and the corresponding transaction log retention), RDS lacks the mechanism to replicate data to the new read replica.

The other options are less likely:

Multi-AZ instances are perfectly fine as sources for read replicas; there's no need to convert them to Single-AZ.

Enhanced Monitoring is a useful tool for performance insights, but it's not a prerequisite for creating read replicas.

While minor version compatibility is important, it generally doesn't prevent the source instance from appearing in the selection box if backups are enabled. The instance might show up with a warning about potential incompatibility.

Therefore, the absence of automated backups is the direct impediment to creating the read replica in this scenario. RDS requires backups to be enabled to ensure that it can access and stream the changes from the source instance to the read replica.

Relevant links:

Working with MySQL Read Replicas - Amazon RDS
Backing Up and Restoring Amazon RDS DB Instances

Question: 40

A Database Specialist has migrated an on-premises Oracle database to Amazon Aurora PostgreSQL. The schema and the data have been migrated successfully.
The on-premises database server was also being used to run database maintenance cron jobs written in Python to perform tasks including data purging and generating data exports. The logs for these jobs show that, most of the time, the jobs completed within 5 minutes, but a few jobs took up to 10 minutes to complete. These maintenance jobs need to be set up for Aurora PostgreSQL.
How can the Database Specialist schedule these jobs so the setup requires minimal maintenance and provides high availability?

A. Create cron jobs on an Amazon EC2 instance to run the maintenance jobs following the required schedule.
B. Connect to the Aurora host and create cron jobs to run the maintenance jobs following the required schedule.
C. Create AWS Lambda functions to run the maintenance jobs and schedule them with Amazon CloudWatch Events.
D. Create the maintenance job using the Amazon CloudWatch job scheduling plugin.

Answer: C

Explanation:

The optimal solution for scheduling database maintenance jobs for Aurora PostgreSQL, requiring minimal maintenance and high availability, is to use AWS Lambda functions triggered by Amazon CloudWatch Events (Option C).

Here's a breakdown of the justification:

Serverless Architecture: AWS Lambda allows execution of code without provisioning or managing servers. This eliminates the operational overhead associated with managing EC2 instances (Option A) or directly managing cron jobs on the database host (Option B).

Scalability and High Availability: Lambda automatically scales to meet demand, ensuring that the maintenance jobs will run even during peak periods. The inherent high availability of Lambda guarantees minimal downtime.

Event-Driven Scheduling: Amazon CloudWatch Events (now part of Amazon EventBridge) provides a reliable way to schedule Lambda functions based on a cron expression. This offers precise control over the execution timing of the maintenance jobs.

Cost-Effectiveness: Lambda functions are billed only for the compute time consumed, making it a cost-effective solution compared to running a dedicated EC2 instance 24/7 (Option A), especially given the short duration of the maintenance jobs.

Maintenance Isolation: By using Lambda, the maintenance jobs are isolated from the Aurora PostgreSQL instance. Any issues with the maintenance jobs will not directly impact the database's stability.

CloudWatch's Role: While CloudWatch is essential for scheduling (as part of CloudWatch Events), Option D's "job scheduling plugin" is misleading or outdated. CloudWatch Events directly schedules invocations of other AWS services, like Lambda, based on event rules, including cron-based schedules. There isn't a specific "CloudWatch job scheduling plugin" in the typical sense. CloudWatch Logs can be used to monitor Lambda function executions for troubleshooting.

Avoid Direct Database Host Access: Option B is not recommended because directly modifying cron jobs on the database host increases the risk of misconfiguration and potential instability of the database environment. It also adds operational overhead.

In summary, Lambda and CloudWatch Events provide a serverless, scalable, highly available, and cost-effective solution for scheduling database maintenance jobs, minimizing operational overhead and ensuring the reliability of the Aurora PostgreSQL database.

Authoritative Links:

AWS Lambda:https://aws.amazon.com/lambda/
Amazon CloudWatch Events (Amazon EventBridge):https://aws.amazon.com/eventbridge/

Question: 41

A company has an Amazon RDS Multi-AZ DB instances that is 200 GB in size with an RPO of 6 hours. To meet the company's disaster recovery policies, the database backup needs to be copied into another Region. The company requires the solution to be cost-effective and operationally efficient.
What should a Database Specialist do to copy the database backup into a different Region?

A. Use Amazon RDS automated snapshots and use AWS Lambda to copy the snapshot into another Region

B. Use Amazon RDS automated snapshots every 6 hours and use Amazon S3 cross-Region replication to copy the snapshot into another Region

C. Create an AWS Lambda function to take an Amazon RDS snapshot every 6 hours and use a second Lambda function to copy the snapshot into another Region

D. Create a cross-Region read replica for Amazon RDS in another Region and take an automated snapshot of the read replica

Answer: C

Explanation:

The correct answer is C. Here's why:

RPO Requirement: The Recovery Point Objective (RPO) of 6 hours dictates how frequently the database

backup needs to be created. Options A and B utilizing native RDS automated backups might not guarantee a 6-hour RPO. Automated backups are scheduled, not strictly controlled to occur every 6 hours.

Lambda for Scheduling: Lambda functions can be precisely scheduled using Amazon CloudWatch Events (now Amazon EventBridge) to trigger a snapshot creation every 6 hours, satisfying the RPO.

Cost-Effectiveness: Cross-Region Read Replicas (Option D) continuously replicate data. While offering low RTO (Recovery Time Objective), they are significantly more expensive than snapshot-based approaches as you are paying for a constantly running DB instance in the secondary region. Snapshots only incur storage costs.
S3 Cross-Region Replication is also a good option, but it still needs a trigger to take a snapshot first.

Operational Efficiency: Lambda functions automate the entire process, removing the need for manual intervention or complex custom scripting, promoting operational efficiency. One Lambda function takes the RDS snapshot, and a second Lambda function then copies the snapshot to another region. This minimizes manual operations.

Snapshot-based DR: For disaster recovery scenarios where a few hours of data loss is acceptable (RPO of 6 hours), snapshot-based replication is generally more cost-effective than active replication (e.g., using read replicas).

Why other options are incorrect:

A: While using automated snapshots, AWS Lambda does not have direct integration to copy these snapshots to another region. RDS does have this functionality via copy snapshots and enabling cross-region backup copy.

B: S3 Cross-Region Replication copies objects within S3 buckets. RDS snapshots are not directly stored in S3. Moreover, the snapshot still needs to be manually triggered every 6 hours.

D: Read Replicas offer faster recovery, they are not cost-effective if the primary goal is to meet a 6-hour RPO. This approach keeps a live replica running, incurring substantial ongoing costs compared to using snapshots and Lambda.

Supporting Documentation:

Amazon RDS Snapshots:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html **Amazon RDS Cross-Region Snapshots:**
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CopySnapshot.html **AWS Lambda:**https://aws.amazon.com/lambda/
Amazon CloudWatch Events (Amazon EventBridge):https://aws.amazon.com/eventbridge/

Question: 42

An Amazon RDS EBS-optimized instance with Provisioned IOPS (PIOPS) storage is using less than half of its allocated IOPS over the course of several hours under constant load. The RDS instance exhibits multi-second read and write latency, and uses all of its maximum bandwidth for read throughput, yet the instance uses less than half of its CPU and RAM resources. What should a Database Specialist do in this situation to increase performance and return latency to sub-second levels?

A. Increase the size of the DB instance storage
B. Change the underlying EBS storage type to General Purpose SSD (gp2)
C. Disable EBS optimization on the DB instance
D. Change the DB instance to an instance class with a higher maximum bandwidth

Answer: D

Explanation:

The problem describes an RDS instance with provisioned IOPS experiencing high latency despite low CPU and RAM utilization, while maxing out read bandwidth and using less than half its provisioned IOPS. This suggests the bottleneck is likely related to network bandwidth or instance limitations, not IOPS capacity.

Option A, increasing storage size, might provide more IOPS headroom if the original storage was relatively small, but since the instance isn't fully using its provisioned IOPS, it is unlikely to solve the bandwidth saturation issue. EBS volumes of a sufficient size usually can provide enough IOPS to handle reads even at higher throughput.

Option B, switching to General Purpose SSD (gp2), is counterproductive. PIOPS (io1 or io2) EBS volumes are designed for applications requiring consistent, high IOPS performance. Switching to gp2 might worsen the situation, as gp2 volumes have burst capacity and potentially lower baseline performance for sustained workloads compared to the provisioned performance of an io1/io2 volume.

Option C, disabling EBS optimization, is also incorrect. EBS optimization is designed to minimize contention between EBS I/O and other network traffic on your instance, precisely to avoid bandwidth-related performance issues. Disabling it would almost certainly worsen latency.

Option D, changing to a DB instance class with higher bandwidth, is the most appropriate solution. The scenario clearly indicates that the instance is hitting its maximum bandwidth limit for read throughput. Upgrading to a larger instance class provides higher network bandwidth capacity, which directly addresses the identified bottleneck, allowing the RDS instance to handle the throughput demands without latency spikes. With increased bandwidth, the instance can process more read requests within a given timeframe, reducing the queuing and latency.

Therefore, selecting a DB instance with greater network bandwidth is the best approach to resolve the high latency caused by bandwidth saturation.

Authoritative Links:

Amazon RDS Instance Types:https://aws.amazon.com/rds/instance-types/ (Shows the different instance classes and their specifications including network performance)
Amazon EBS Optimized Instances:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-optimized.html (Explains benefits of EBS optimization)
Amazon EBS Volume Types:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html (Details the characteristics and use cases of different EBS volume types)

Question: 43

After restoring an Amazon RDS snapshot from 3 days ago, a company's Development team cannot connect to the restored RDS DB instance. What is the likely cause of this problem?

A. The restored DB instance does not have Enhanced Monitoring enabled
B. The production DB instance is using a custom parameter group
C. The restored DB instance is using the default security group
D. The production DB instance is using a custom option group

Answer: C

Explanation:

The correct answer is **C. The restored DB instance is using the default security group.**

Here's why:

Security groups act as virtual firewalls for your RDS instances, controlling inbound and outbound traffic.
When an RDS instance is created (or restored), it's associated with one or more security groups. These security groups define which IP addresses or other EC2 instances can connect to the database. If a custom security group was in place before the restore, it likely had specific rules allowing the Development team's access.

Restoring a database instance from a snapshot creates a new RDS instance. By default, restored instances are often associated with the **default security group** in the VPC. The default security group typically restricts inbound access by default, only allowing traffic within the same security group or no inbound access at all, depending on your VPC configuration.

Therefore, if the Development team's IP addresses or the security groups of their EC2 instances are not explicitly authorized in the restored instance's default security group, they will be unable to connect. The original production database instance likely had a custom security group configured to specifically allow their access.

Options A, B, and D are less likely to be the primary cause:

A. The restored DB instance does not have Enhanced Monitoring enabled: Enhanced Monitoring gathers metrics about the operating system that your DB instance runs on. It doesn't directly influence network connectivity. While Enhanced Monitoring is beneficial, its absence wouldn't block the Development team's connection.

B. The production DB instance is using a custom parameter group: Parameter groups define database engine configuration parameters (e.g., buffer pool size, character sets). While different parameter settings could theoretically affect connection behavior in some edge cases (e.g., maximum connections allowed), the default security group being applied is a far more direct and likely reason for connection failures.

D. The production DB instance is using a custom option group: Option groups enable optional database features (e.g., native network encryption, TDE). Similar to parameter groups, these options affect database features but are less likely than security groups to prevent basic connection attempts unless the option group is directly related to network access control (which is rare).

In summary, network access is controlled primarily through security groups. A restored instance reverting to the default security group is the most plausible explanation for the Development team's inability to connect after the restore.

For further research:

Amazon RDS Security Groups:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.RDSSecurityGroups.html **Working with DB Parameter Groups:**
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html
Working with Option Groups:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithOptionGroups.html

Question: 44

A gaming company has implemented a leaderboard in AWS using a Sorted Set data structure within Amazon ElastiCache for Redis. The ElastiCache cluster has been deployed with cluster mode disabled and has a replication group deployed with two additional replicas. The company is planning for a worldwide gaming event and is anticipating a higher write load than what the current cluster can handle.
Which method should a Database Specialist use to scale the ElastiCache cluster ahead of the upcoming event?

A. Enable cluster mode on the existing ElastiCache cluster and configure separate shards for the Sorted Set across all nodes in the cluster.

B. Increase the size of the ElastiCache cluster nodes to a larger instance size.

C. Create an additional ElastiCache cluster and load-balance traffic between the two clusters.

D. Use the EXPIRE command and set a higher time to live (TTL) after each call to increment a given key.

Answer: B

Explanation:

Here's a detailed justification for why increasing the ElastiCache node size is the best approach in this scenario, along with supporting evidence:

The goal is to handle a higher write load on an ElastiCache for Redis cluster that currently has cluster mode disabled and a replication group with two replicas. The data structure being used is a Sorted Set, which is crucial for the leaderboard functionality.

Option B, increasing the size of the ElastiCache cluster nodes to a larger instance size (vertical scaling), is the most suitable solution. Vertical scaling allows the Redis instance to leverage more CPU, memory, and network bandwidth, directly improving its capacity to process more write operations. Since cluster mode is disabled, the data is stored on a single primary node, making increasing its resources the direct way to improve performance. This is the simplest and quickest way to increase throughput without any application-level code changes.

Option A, enabling cluster mode, requires significant application changes to handle data sharding. The prompt states there is an impending gaming event, so time is of the essence. Enabling cluster mode is time-consuming, and there is no guarantee that it can be thoroughly tested ahead of the event. Also, enabling cluster mode when it was previously off can involve downtime for the application.

Option C, creating an additional ElastiCache cluster and load-balancing traffic, introduces significant complexity. The application would need to implement custom logic to determine which cluster to write to and potentially handle data consistency issues between the clusters. This is essentially manual sharding, which is more complex than the built-in sharding from ElastiCache cluster mode. This is not feasible, because the cluster must guarantee high-availability in order to avoid impacting the gaming experience.

Option D, using the EXPIRE command to increase the TTL, is not relevant to write capacity. EXPIRE is used to manage data retention, not to reduce the load on the Redis server. The command will impact the lifetime of the data and impact the validity of the leaderboard.

Therefore, increasing the ElastiCache instance size directly addresses the need for higher write throughput on the existing primary node, without the complexity and time investment of implementing sharding or introducing application-level load balancing.

Supporting evidence:

AWS ElastiCache Documentation on Scaling: https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/Scaling.html
AWS Whitepaper on Redis Best Practices: (Search AWS Whitepapers for "Redis Best Practices") This whitepaper details strategies for optimal Redis performance on AWS.

Question: 45

An ecommerce company has tasked a Database Specialist with creating a reporting dashboard that visualizes critical business metrics that will be pulled from the core production database running on Amazon Aurora. Data that is read by the dashboard should be available within 100 milliseconds of an update.
The Database Specialist needs to review the current configuration of the Aurora DB cluster and develop a cost-effective solution. The solution needs to accommodate the unpredictable read workload from the reporting dashboard without any impact on the write availability and performance of the DB cluster.

Which solution meets these requirements?

A. Turn on the serverless option in the DB cluster so it can automatically scale based on demand.

B. Provision a clone of the existing DB cluster for the new Application team.

C. Create a separate DB cluster for the new workload, refresh from the source DB cluster, and set up ongoing replication using AWS DMS change data capture (CDC).

D. Add an automatic scaling policy to the DB cluster to add Aurora Replicas to the cluster based on CPU consumption.

Answer: D

Explanation:

The correct answer is **D. Add an automatic scaling policy to the DB cluster to add Aurora Replicas to the cluster based on CPU consumption.**

Here's a detailed justification:

The primary requirement is to provide a reporting dashboard with near real-time data (within 100 milliseconds) without impacting the performance of the production Aurora database. The solution must also be cost-effective and handle unpredictable read workloads.

Aurora Replicas are designed for read scaling. They share the same underlying storage as the primary instance, ensuring data is virtually up-to-date with minimal replication lag, typically within milliseconds. This meets the 100ms latency requirement.

Automatic scaling policy allows the number of Aurora Replicas to adjust based on demand, specifically CPU consumption in this case. When the dashboard workload increases, more Replicas are added automatically, distributing the read load and preventing the primary instance from being overwhelmed. When the load decreases, Replicas are removed, optimizing costs.

Option A (Serverless): While Aurora Serverless can scale automatically, it's designed for infrequent, intermittent, or unpredictable workloads. For a continuously running reporting dashboard, especially one requiring near real-time data, provisioning Aurora Replicas is generally more cost-effective and predictable. Serverless also involves a brief activation time when scaling that can exceed the 100ms latency requirement. **Option B (Clone):** Cloning creates a point-in-time copy. It doesn't offer continuous data replication, making it unsuitable for near real-time data requirements. Also, the read latency will be affected.

Option C (Separate Cluster with DMS CDC): AWS DMS CDC (Change Data Capture) can replicate data but typically has higher latency than Aurora Replicas (seconds to minutes). Additionally, setting up and maintaining DMS CDC adds complexity and cost.

Therefore, adding Aurora Replicas with an automatic scaling policy offers the best balance of near real-time data, cost-effectiveness, and scalability for unpredictable read workloads, making it the optimal solution.

Supporting Links:

Aurora Replicas
Aurora Auto Scaling
AWS DMS CDC

Question: 46

A retail company is about to migrate its online and mobile store to AWS. The company's CEO has strategic plans to grow the brand globally. A Database
Specialist has been challenged to provide predictable read and write database performance with minimal operational overhead.
What should the Database Specialist do to meet these requirements?

A. Use Amazon DynamoDB global tables to synchronize transactions
B. Use Amazon EMR to copy the orders table data across Regions
C. Use Amazon Aurora Global Database to synchronize all transactions
D. Use Amazon DynamoDB Streams to replicate all DynamoDB transactions and sync them

Answer: A

Explanation:

The best solution is **A. Use Amazon DynamoDB global tables to synchronize transactions.**

Here's why:

Predictable Read and Write Performance: DynamoDB is a NoSQL database known for its consistent and predictable performance at any scale. DynamoDB global tables provide active-active replication across multiple AWS Regions, ensuring low latency reads and writes for users globally. Each Region has a read/write replica of the table, allowing clients to access the nearest copy.

Minimal Operational Overhead: DynamoDB is a fully managed service. Global tables automate the replication of data across Regions, removing the burden of managing complex replication configurations and infrastructure.

Global Growth: DynamoDB global tables are designed to support global applications by providing low-latency access to data for users in different geographic locations. As the retail company expands its online and mobile store globally, global tables will ensure that users have a consistent and responsive experience.

Aurora Global Database (Option C) would be suitable for a relational database which is not specified in the question and often requires more operational overhead than DynamoDB. It's also more complex to set up.

Amazon EMR (Option B) is for big data processing, not transactional database replication. It is not an appropriate solution for the stated requirement.

DynamoDB Streams (Option D) can replicate data, but it requires custom code to consume and apply the changes, resulting in more operational overhead. Global tables offer a managed solution. DynamoDB Streams can be integrated with custom applications to perform actions on the data, such as invoking a Lambda function.

In summary, DynamoDB global tables provide the best balance of predictable performance, minimal operational overhead, and global scalability for the retail company's needs. It leverages the strengths of DynamoDB as a NoSQL database, fully managed service, and a simple, scalable, global data replication solution.

Supporting Documentation:

Amazon DynamoDB Global Tables: How It Works
Best Practices for DynamoDB

Question: 47

A company is closing one of its remote data centers. This site runs a 100 TB on-premises data warehouse solution. The company plans to use the AWS Schema
Conversion Tool (AWS SCT) and AWS DMS for the migration to AWS. The site network bandwidth is 500 Mbps. A Database Specialist wants to migrate the on- premises data using Amazon S3 as the data lake and Amazon Redshift as the data warehouse. This move must take place during a 2-week period when source systems are shut down for maintenance. The data should stay encrypted at rest and in transit.
Which approach has the least risk and the highest likelihood of a successful data transfer?

A. Set up a VPN tunnel for encrypting data over the network from the data center to AWS. Leverage AWS SCT and apply the converted schema to Amazon Redshift. Once complete, start an AWS DMS task to move the data from the source to Amazon S3. Use AWS Glue to load the data from Amazon S3 to Amazon Redshift.

B. Leverage AWS SCT and apply the converted schema to Amazon Redshift. Start an AWS DMS task with two AWS Snowball Edge devices to copy data from on-premises to Amazon S3 with AWS KMS encryption. Use AWS DMS to finish copying data to Amazon Redshift.

C. Leverage AWS SCT and apply the converted schema to Amazon Redshift. Once complete, use a fleet of 10 TB dedicated encrypted drives using the AWS Import/Export feature to copy data from on-premises to Amazon S3 with AWS KMS encryption. Use AWS Glue to load the data to Amazon redshift.

D. Set up a VPN tunnel for encrypting data over the network from the data center to AWS. Leverage a native database export feature to export the data and compress the files. Use the aws S3 cp multi-port upload command to upload these files to Amazon S3 with AWS KMS encryption. Once complete, load the data to Amazon Redshift using AWS Glue.

Answer: B

Explanation:

Option B is the most suitable solution for migrating the 100 TB on-premises data warehouse to Amazon Redshift within the given constraints (2-week downtime, limited bandwidth, encryption requirements). Here's why:

Schema Conversion: AWS SCT is correctly used to convert the schema to be compatible with Amazon Redshift.

Data Transfer with Snowball Edge: Given the limited bandwidth (500 Mbps) and the large data volume (100 TB), transferring data over the network within the 2-week window is highly unlikely. AWS Snowball Edge provides a secure and efficient offline data transfer method. Since each Snowball Edge device offers storage, using two devices allows for parallel data copying, reducing transfer time. The AWS KMS encryption ensures data at rest on the devices is secure.

Calculating Network Transfer Time: 100 TB is approximately 800 Terabits. At 500 Mbps, transferring 800 Terabits would take 1600 seconds, nearly 28 minutes.

Data Lake and Warehousing: Utilizing Amazon S3 as a data lake is a valid approach. AWS Snowball Edge can directly import the data into S3.

DMS Completion: After the bulk of the data is loaded via Snowball Edge, AWS DMS can be used to replicate any changes that might have occurred during the Snowball transfer, ensuring data consistency. It then finishes loading the data to Amazon Redshift.

Option A is problematic due to the slow network transfer, making it improbable to complete within the 2-week timeframe.

Option C is not practical; AWS Import/Export is outdated and less efficient than Snowball Edge for this size data. Managing multiple drives also introduces complexity and potential failure points.

Option D involves potential network bandwidth issues and uses AWS S3 cp multi-part upload command, which may be complicated and inefficient for such a large dataset compared to Snowball.

Authoritative Links:

AWS Snowball Edge: https://aws.amazon.com/snowball/
AWS DMS: https://aws.amazon.com/dms/
AWS SCT: https://aws.amazon.com/dms/schema-conversion-tool/

Question: 48

A company is looking to migrate a 1 TB Oracle database from on-premises to an Amazon Aurora PostgreSQL DB cluster. The company's Database Specialist discovered that the Oracle database is storing 100 GB of large binary objects (LOBs) across multiple tables. The Oracle database has a maximum LOB size of
500 MB with an average LOB size of 350 MB. The Database Specialist has chosen AWS DMS to migrate the data with the largest replication instances.
How should the Database Specialist optimize the database migration using AWS DMS?

A. Create a single task using full LOB mode with a LOB chunk size of 500 MB to migrate the data and LOBs together

B. Create two tasks: task1 with LOB tables using full LOB mode with a LOB chunk size of 500 MB and task2 without LOBs

C. Create two tasks: task1 with LOB tables using limited LOB mode with a maximum LOB size of 500 MB and task 2 without LOBs

D. Create a single task using limited LOB mode with a maximum LOB size of 500 MB to migrate data and LOBs together

Answer: B

Explanation:

The correct answer is B. Here's a detailed justification:

The problem involves migrating an Oracle database with significant LOB data to Aurora PostgreSQL using AWS DMS. The goal is to optimize the migration process. AWS DMS offers two LOB handling modes: Full LOB mode and Limited LOB mode.

Full LOB mode is suitable for large LOBs but retrieves the entire LOB from the source database and stores it in memory before writing it to the target. Using a very large LOB chunk size, like 500 MB, can lead to excessive memory consumption on the DMS replication instance, potentially causing performance issues or even crashes, especially when dealing with multiple concurrent LOBs.

Limited LOB mode retrieves only the LOB data up to the specified MaxLobSize. Data exceeding this size is truncated, which is unacceptable for the current use case.

The best approach is to split the migration into two tasks:

Task 1 (LOB Tables): Migrate tables containing LOBs using Full LOB mode. This ensures the entire LOB data is migrated without truncation. Setting the LOB chunk size to the maximum (500MB in this case) is acceptable here because the question specifically asks for the "optimization" of this migration, not for the most resource-efficient setup. Using the maximum chunk size leverages parallelism.

Task 2 (Non-LOB Tables): Migrate the remaining tables without LOBs. This reduces the memory pressure on the DMS replication instance, as the LOB processing is isolated to a separate task. Since the task only manages tables without LOBs, it will be less resource-intensive.

By separating LOB and non-LOB tables into different tasks, we optimize resource utilization and potentially reduce the overall migration time. The choice of "largest replication instances" also supports handling potentially large LOBs.

Links for further research:

AWS DMS LOB Handling: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Tasks.LOBSupport.html AWS DMS Best Practices: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_BestPractices.html

Question: 49

A Database Specialist is designing a disaster recovery strategy for a production Amazon DynamoDB table. The table uses provisioned read/write capacity mode, global secondary indexes, and time to live (TTL). The Database Specialist has restored the latest backup to a new table.
To prepare the new table with identical settings, which steps should be performed? (Choose two.)

A. Re-create global secondary indexes in the new table
B. Define IAM policies for access to the new table
C. Define the TTL settings
D. Encrypt the table from the AWS Management Console or use the update-table command
E. Set the provisioned read and write capacity

Answer: BC

Explanation:

The correct answer is BC. Let's justify why and why the other options are incorrect in the context of disaster recovery for a DynamoDB table using a restored backup.

A. Re-create global secondary indexes in the new table: Global secondary indexes are not automatically restored with the table data when using DynamoDB backups. After restoring the backup, you must manually recreate these indexes. This is because the backup primarily focuses on the table's core data, not its associated secondary indexes and their configurations. Therefore, this step is necessary to replicate the original table's functionality.

B. Define IAM policies for access to the new table: IAM policies determine who and what can access the DynamoDB table. These policies are not part of the table backup itself. Once the restored table is in place, you must re-establish or create new IAM policies to govern access. Without correct IAM policies, applications and users won't be able to interact with the restored table, rendering the disaster recovery process incomplete.

C. Define the TTL settings: Time-to-live (TTL) settings define when items in the table are automatically deleted. Like global secondary indexes and IAM policies, TTL settings are also not automatically restored from the table backup. Therefore, the TTL configuration must be re-established on the restored table.

D. Encrypt the table from the AWS Management Console or use the update-table command: DynamoDB backups maintain the encryption status of the source table. If the original table was encrypted, the restored table will also be encrypted using the same key by default. Therefore, manually re-encrypting the table after the restore is generally not required, unless you specifically want to change the encryption key or type (e.g., switch from AWS-owned KMS key to a customer-managed key).

E. Set the provisioned read and write capacity: When you restore a DynamoDB table from a backup, the provisioned read/write capacity settings of the original table are preserved in the restored table. You do not need to manually reconfigure the provisioned capacity. It's automatically inherited from the backup.

In summary, after restoring a DynamoDB table from a backup, you must manually recreate global secondary indexes, redefine IAM policies, and redefine the TTL settings to ensure the restored table behaves identically to the original. You do not need to re-encrypt the table (as the encryption is preserved) or manually reset the provisioned throughput (as it's also preserved).

Authoritative Links:

DynamoDB On-Demand Backup and Restore
Managing Global Secondary Indexes
IAM and Policy
DynamoDB Time To Live (TTL)

Question: 50

A Database Specialist is creating Amazon DynamoDB tables, Amazon CloudWatch alarms, and associated infrastructure for an Application team using a development AWS account. The team wants a deployment method that will standardize the core solution components while managing environment-specific settings separately, and wants to minimize rework due to configuration errors.
Which process should the Database Specialist recommend to meet these requirements?

A. Organize common and environmental-specific parameters hierarchically in the AWS Systems Manager Parameter Store, then reference the parameters dynamically from an AWS CloudFormation template. Deploy the CloudFormation stack using the environment name as a parameter.

B. Create a parameterized AWS CloudFormation template that builds the required objects. Keep separate environment parameter files in separate Amazon S3 buckets. Provide an AWS CLI command that deploys the CloudFormation stack directly referencing the appropriate parameter bucket.

C. Create a parameterized AWS CloudFormation template that builds the required objects. Import the template into the CloudFormation interface in the AWS Management Console. Make the required changes to the parameters and deploy the CloudFormation stack.

D. Create an AWS Lambda function that builds the required objects using an AWS SDK. Set the required parameter values in a test event in the Lambda console for each environment that the Application team can modify, as needed. Deploy the infrastructure by triggering the test event in the console.

Answer: A

Explanation:

The best approach to meet the requirements of standardizing deployments while managing environment-specific settings separately and minimizing configuration errors is **A**. Here's why:

AWS CloudFormation for Standardization: CloudFormation enables Infrastructure as Code (IaC), allowing the Database Specialist to define the core DynamoDB tables, CloudWatch alarms, and associated infrastructure in a template. This ensures consistency across environments.

AWS Systems Manager Parameter Store for Configuration Management: Parameter Store provides a centralized, secure, and hierarchical repository for storing configuration data and secrets. It supports versions and encryption, which reduces the risk of configuration errors.

Dynamic Parameter Referencing: CloudFormation can dynamically reference parameters stored in Parameter Store. This means the same CloudFormation template can be used across multiple environments, with environment-specific configurations pulled dynamically from Parameter Store during deployment.

Environment Parameterization: By deploying the CloudFormation stack with the environment name as a parameter, the correct set of parameters for that environment can be retrieved from Parameter Store. This isolates environment configurations and prevents cross-environment contamination.

Why other options are less suitable:

B: Storing parameter files in S3 introduces more overhead and complexity in managing permissions and potentially accessing the right bucket based on environments. Parameter store gives a more straightforward way to manage parameters and use them dynamically.

C: Manually modifying parameters in the CloudFormation interface is error-prone and doesn't promote automation or repeatability. It's also hard to track versions and maintain consistency across deployments. **D:** Using a Lambda function to build infrastructure is less declarative and maintainable than CloudFormation. Also, putting parameters in test events is not suitable for production systems. It's not designed for managing environment-specific configurations. It is not an effective method of doing infrastructure as code.

In summary: Option A leverages the strengths of CloudFormation for infrastructure standardization and Parameter Store for environment-specific configuration management, resulting in a robust and easily manageable deployment process.

Supporting Links:

AWS CloudFormation
AWS Systems Manager Parameter Store

Question: 51

A company runs online transaction processing (OLTP) workloads on an Amazon RDS for PostgreSQL Multi-AZ DB instance. Tests were run on the database after work hours, which generated additional database logs. The free storage of the RDS DB instance is low due to these additional logs.
What should the company do to address this space constraint issue?

A. Log in to the host and run the rm $PGDATA/pg_logs/* command
B. Modify the rds.log_retention_period parameter to 1440 and wait up to 24 hours for database logs to be deleted
C. Create a ticket with AWS Support to have the logs deleted
D. Run the SELECT rds_rotate_error_log() stored procedure to rotate the logs

Answer: B

Explanation:

The correct answer is **B. Modify the rds.log_retention_period parameter to 1440 and wait up to 24 hours for database logs to be deleted.**

Here's a detailed justification:

The problem describes a low storage issue on an RDS for PostgreSQL instance caused by excessive logs. The key to solving this issue is managing log retention. RDS provides built-in mechanisms for managing logs, and manually deleting files on the underlying host (Option A) is **strongly discouraged** and unsupported by AWS. Modifying the underlying OS in such a way violates AWS's managed service principles.

Option C, creating a support ticket, is unnecessary for routine tasks like log management which are well-documented and manageable by the user. AWS Support typically assists with problems beyond the scope of normal operational activities.

Option D, running SELECT rds_rotate_error_log(), rotates the error log file, creating a new error log. While this can temporarily free up space from the current error log, it doesn't address the underlying issue of overall log volume taking up storage. This action only addresses the current error log size and doesn't influence the retention policy.

Option B directly addresses the core problem: excessive log retention. The rds.log_retention_period parameter controls how long RDS retains database logs. The value is specified in minutes. Setting it to 1440 minutes is equivalent to setting it to 24 hours. RDS automatically manages log deletion according to this retention policy. After changing the parameter, it may take up to 24 hours for RDS to initiate the log deletion process. This is the intended and supported way to manage log storage in RDS. Decreasing this value would ensure the older logs are deleted, freeing up valuable storage space.

Key Concepts and Relevant Links:

Amazon RDS Log Management: RDS provides mechanisms for managing logs for different database engines, including PostgreSQL.
rds.log_retention_period Parameter: This parameter specifically controls the retention period for database logs.
Managed Service Principles: AWS RDS is a managed service, meaning AWS handles the underlying

infrastructure and operational tasks. Users should interact with the service through the provided APIs and parameters.

RDS for PostgreSQL Documentation:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_LogAccess.Concepts.PostgreSQL.html **RDS Parameter Groups:**
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html

Changing the rds.log_retention_period is the proper, supported, and direct approach to addressing the database log space constraint issue.

Question: 52

A user has a non-relational key-value database. The user is looking for a fully managed AWS service that will offload the administrative burdens of operating and scaling distributed databases. The solution must be cost-effective and able to handle unpredictable application traffic.
What should a Database Specialist recommend for this user?

A. Create an Amazon DynamoDB table with provisioned capacity mode
B. Create an Amazon DocumentDB cluster
C. Create an Amazon DynamoDB table with on-demand capacity mode
D. Create an Amazon Aurora Serverless DB cluster

Answer: C

Explanation:

The correct answer is **C. Create an Amazon DynamoDB table with on-demand capacity mode**. Here's a detailed justification:

The user requires a fully managed, cost-effective, and scalable solution for a key-value database handling unpredictable traffic. Let's analyze each option:

A. Create an Amazon DynamoDB table with provisioned capacity mode: While DynamoDB is fully managed and suitable for key-value data, provisioned capacity requires predefining read/write capacity units. This is inefficient and potentially costly for unpredictable traffic, as you either over-provision and waste resources or under-provision and face throttling.

B. Create an Amazon DocumentDB cluster: Amazon DocumentDB is a fully managed document database service compatible with MongoDB workloads. It's not optimized for simple key-value use cases and is more complex and costly than necessary. The user specifies a key-value database requirement, making DocumentDB less suitable.

C. Create an Amazon DynamoDB table with on-demand capacity mode: On-demand capacity mode in DynamoDB is perfectly suited for unpredictable traffic. It automatically scales up or down based on actual request volume, eliminating the need to pre-provision capacity. The pricing model reflects actual usage, making it highly cost-effective for variable workloads. DynamoDB, in general, addresses the fully managed aspect and supports key-value storage natively.

D. Create an Amazon Aurora Serverless DB cluster: Amazon Aurora Serverless is for relational databases (compatible with MySQL and PostgreSQL). It automatically scales, but it's designed for transactional workloads and is an overkill for a simple key-value requirement. It's also relational and thus doesn't fit the no-relational key-value database requirement.

Therefore, option C best addresses all the user's needs: it's a fully managed DynamoDB solution optimized for

key-value storage, uses on-demand capacity for cost-effective handling of unpredictable traffic, and removes the administrative burden of capacity planning.

Relevant links for further research:

Amazon DynamoDB on-demand capacity:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html
Amazon DynamoDB:https://aws.amazon.com/dynamodb/
Amazon DocumentDB:https://aws.amazon.com/documentdb/
Amazon Aurora Serverless:https://aws.amazon.com/rds/aurora/serverless/

Question: 53

A gaming company is designing a mobile gaming app that will be accessed by many users across the globe. The company wants to have replication and full support for multi-master writes. The company also wants to ensure low latency and consistent performance for app users.
Which solution meets these requirements?

A. Use Amazon DynamoDB global tables for storage and enable DynamoDB automatic scaling
B. Use Amazon Aurora for storage and enable cross-Region Aurora Replicas
C. Use Amazon Aurora for storage and cache the user content with Amazon ElastiCache
D. Use Amazon Neptune for storage

Answer: A

Explanation:

The correct answer is A: Use Amazon DynamoDB global tables for storage and enable DynamoDB automatic scaling. Here's why:

DynamoDB global tables are specifically designed for globally distributed applications requiring low-latency access to data from multiple regions and supporting multi-master writes. This directly addresses the gaming company's requirement for a mobile app accessed globally with full multi-master write support. Global tables replicate data across multiple AWS regions, allowing users to access data from the nearest region, thus reducing latency. They are also designed for high availability and disaster recovery.

The requirement for consistent performance is met by DynamoDB's automatic scaling capability. Automatic scaling dynamically adjusts the read and write capacity of the tables based on actual traffic patterns. This ensures that the application maintains consistent performance even during peak usage periods, eliminating manual intervention and preventing performance bottlenecks.
[https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.html]

Option B is incorrect because Aurora cross-Region Replicas, while providing read replicas, do not inherently support multi-master writes. Writes would typically be directed to a single primary region. While one could potentially build a complex application to manage cross-region writes with Aurora, it would not be as straightforward or natively supported as DynamoDB global tables.

Option C is also incorrect. While caching with ElastiCache can improve read performance, it doesn't solve the core requirement of multi-master writes and global data distribution. ElastiCache sits in front of a database; it doesn't replace the database's core functionalities for distributed writes.

Option D, using Amazon Neptune, is designed for graph databases, which may not be the optimal data model for the gaming application's data storage needs. While Neptune can be replicated across regions, it is not typically chosen for general-purpose transactional data with multi-master write requirements in the way DynamoDB global tables are. Neptune is best suited when the relationship between data elements are critical.

[https://aws.amazon.com/neptune/]

In summary, DynamoDB global tables with automatic scaling provide the best solution because they inherently support multi-master writes across regions, ensure low latency through regional data replication, and maintain consistent performance with automatic capacity scaling, perfectly aligning with the gaming company's stated requirements.

Question: 54

A Database Specialist needs to speed up any failover that might occur on an Amazon Aurora PostgreSQL DB cluster. The Aurora DB cluster currently includes the primary instance and three Aurora Replicas.
How can the Database Specialist ensure that failovers occur with the least amount of downtime for the application?

A. Set the TCP keepalive parameters low
B. Call the AWS CLI failover-db-cluster command
C. Enable Enhanced Monitoring on the DB cluster
D. Start a database activity stream on the DB cluster

Answer: A

Explanation:

The correct answer is **A. Set the TCP keepalive parameters low.**

Here's why:

Minimizing downtime during a failover in Aurora PostgreSQL is critical. The goal is to quickly detect a primary instance failure and promote one of the Aurora Replicas to become the new primary. TCP keepalive parameters play a crucial role in this detection process.

TCP Keepalive: These parameters configure how frequently and aggressively the operating system checks if a TCP connection is still active. By lowering these parameters (e.g., tcp_keepalives_idle, tcp_keepalives_interval, tcp_keepalives_count), the system detects broken connections to the primary instance more rapidly. This faster detection allows Aurora to initiate the failover process sooner, reducing downtime. The application experiences a quicker interruption, and Aurora can initiate a faster failover.

Why other options are incorrect:

B. Call the AWS CLI failover-db-cluster command: This command forces a failover. While useful for testing, it's not a proactive measure to speed up automatic failover detection in production. Manually intervening adds delay.

C. Enable Enhanced Monitoring on the DB cluster: Enhanced Monitoring provides detailed metrics about the instance, including CPU utilization, memory usage, and disk I/O. Although useful for performance analysis and troubleshooting, it doesn't directly influence failover detection speed.

D. Start a database activity stream on the DB cluster: Database Activity Streams is an audit feature which capture database activities for security and regulatory compliance. It doesn't contribute to faster failover detection.

In summary, adjusting TCP keepalive parameters allows for faster detection of connection loss to the primary instance, which prompts Aurora to initiate a failover quickly. The quicker Aurora can identify the issue, the more rapidly a Replica is promoted and the faster the application gets back up.

Supporting Documentation:

Amazon Aurora Failover
PostgreSQL TCP Keepalive (Refer to the parameters that control connection timeout.)

Question: 55

A Database Specialist needs to define a database migration strategy to migrate an on-premises Oracle database to an Amazon Aurora MySQL DB cluster. The company requires near-zero downtime for the data migration. The solution must also be cost-effective.
Which approach should the Database Specialist take?

A. Dump all the tables from the Oracle database into an Amazon S3 bucket using datapump (expdp). Run data transformations in AWS Glue. Load the data from the S3 bucket to the Aurora DB cluster.

B. Order an AWS Snowball appliance and copy the Oracle backup to the Snowball appliance. Once the Snowball data is delivered to Amazon S3, create a new Aurora DB cluster. Enable the S3 integration to migrate the data directly from Amazon S3 to Amazon RDS.

C. Use the AWS Schema Conversion Tool (AWS SCT) to help rewrite database objects to MySQL during the schema migration. Use AWS DMS to perform the full load and change data capture (CDC) tasks.

D. Use AWS Server Migration Service (AWS SMS) to import the Oracle virtual machine image as an Amazon EC2 instance. Use the Oracle Logical Dump utility to migrate the Oracle data from Amazon EC2 to an Aurora DB cluster.

Answer: C

Explanation:

The correct approach is C: Utilize AWS SCT for schema conversion and AWS DMS for data migration. Here's why:

AWS Schema Conversion Tool (AWS SCT) automates the complex process of converting database schemas from one engine (Oracle) to another (MySQL). It assesses the source database schema and creates a report highlighting compatibility issues, suggesting remediation steps, and automating the conversion of database objects like tables, indexes, views, and stored procedures. This reduces manual effort and ensures a more accurate and efficient schema migration.

AWS Database Migration Service (AWS DMS) enables near-zero downtime migration. DMS supports heterogeneous database migrations and facilitates change data capture (CDC). During the initial full load, DMS copies the entire data from the Oracle database to the Aurora MySQL DB cluster. Subsequently, through CDC, DMS continuously replicates ongoing changes from the source database to the target database. This keeps the Aurora cluster synchronized, minimizing downtime when switching over. This meets the
requirement of near-zero downtime.

Option A is less ideal because dumping into S3, transforming with Glue, and then loading is an ETL process that requires significant downtime during the full load.

Option B, using Snowball, is better suited for large datasets or situations where network bandwidth is limited. However, it doesn't inherently offer a near-zero downtime migration strategy. The downtime would occur during the data load from S3 to Aurora.

Option D, using SMS and EC2, introduces unnecessary complexity and cost. Running Oracle on EC2 for migration purposes is less cost-effective than leveraging the purpose-built DMS. Also, simply performing a logical dump from Oracle to Aurora would still incur downtime for the full data load. Oracle Data Pump on EC2 is also an option, but ultimately offers no advantage over DMS and is likely more complex.

DMS is designed for minimizing downtime during data migration, and SCT helps simplify and automate the

schema conversion, making option C the most cost-effective and suitable approach for migrating an Oracle database to Amazon Aurora MySQL with near-zero downtime.

Supporting Documentation:

AWS Schema Conversion Tool:https://aws.amazon.com/sct/ **AWS Database Migration Service:**https://aws.amazon.com/dms/

Question: 56

A marketing company is using Amazon DocumentDB and requires that database audit logs be enabled. A Database Specialist needs to configure monitoring so that all data definition language (DDL) statements performed are visible to the Administrator. The Database Specialist has set the audit_logs parameter to enabled in the cluster parameter group. What should the Database Specialist do to automatically collect the database logs for the Administrator?

A. Enable DocumentDB to export the logs to Amazon CloudWatch Logs

B. Enable DocumentDB to export the logs to AWS CloudTrail

C. Enable DocumentDB Events to export the logs to Amazon CloudWatch Logs

D. Configure an AWS Lambda function to download the logs using the download-db-log-file-portion operation and store the logs in Amazon S3

Answer: A

Explanation:

The correct answer is **A. Enable DocumentDB to export the logs to Amazon CloudWatch Logs.** and **C. Enable DocumentDB Events to export the logs to Amazon CloudWatch Logs**.

Here's a detailed justification:

The requirement is to capture and make visible all Data Definition Language (DDL) statements executed within the Amazon DocumentDB database for audit purposes. DocumentDB audit logs, once enabled via the audit_logs parameter, contain information about these DDL operations. The most direct and efficient way to collect and centralize these logs is to export them to Amazon CloudWatch Logs.

Why CloudWatch Logs? CloudWatch Logs is a centralized service for collecting, monitoring, and archiving log files. By exporting DocumentDB audit logs to CloudWatch Logs, the administrator can easily search, filter, and analyze the DDL statements performed within the database using CloudWatch Logs' powerful querying capabilities. CloudWatch Logs also allows you to set up alarms based on specific patterns found in the logs, providing real-time alerting of critical DDL changes.

Why other options are not optimal:

B. Enable DocumentDB to export the logs to AWS CloudTrail: While CloudTrail tracks API calls made to AWS services (including DocumentDB), it primarily focuses on control plane operations (e.g., creating a DocumentDB cluster, modifying a security group). It doesn't directly capture the DDL statements executed within the database itself. CloudTrail is helpful for auditing who made changes to your AWS infrastructure and resources, but is not the primary way to log audit information within DocumentDB.

D. Configure an AWS Lambda function to download the logs using the download-db-log-file-portion operation and store the logs in Amazon S3: This approach is overly complex and less efficient. It involves creating and maintaining a Lambda function, managing its permissions, and handling the log downloading and storage process manually. DocumentDB offers a built-in integration with CloudWatch Logs, making this manual approach unnecessary. While storing logs in S3 is a valid archiving strategy, it's best done after initially exporting to CloudWatch Logs for real-time monitoring and analysis. The Lambda function is also not

the primary way to log audit information within DocumentDB.

C. Enable DocumentDB Events to export the logs to Amazon CloudWatch Logs: DocumentDB Events track the status of different events in the DB instance or cluster, such as creation, deletion, failover, etc. These are not the same as DDL or DML commands. Although there are benefits of seeing these events, they do not satisfy the requirements of the problem. It would be beneficial to see some of these events, but not all of them.

Therefore, directly exporting DocumentDB audit logs to Amazon CloudWatch Logs is the recommended approach for automatically collecting and centralizing DDL statements for auditing purposes.

Supporting Links:

Amazon DocumentDB Auditing:https://docs.aws.amazon.com/documentdb/latest/developerguide/event-auditing.html
Amazon CloudWatch Logs:https://aws.amazon.com/cloudwatch/features/

Question: 57

A company is looking to move an on-premises IBM Db2 database running AIX on an IBM POWER7 server. Due to escalating support and maintenance costs, the company is exploring the option of moving the workload to an Amazon Aurora PostgreSQL DB cluster.
What is the quickest way for the company to gather data on the migration compatibility?

A. Perform a logical dump from the Db2 database and restore it to an Aurora DB cluster. Identify the gaps and compatibility of the objects migrated by comparing row counts from source and target tables.

B. Run AWS DMS from the Db2 database to an Aurora DB cluster. Identify the gaps and compatibility of the objects migrated by comparing the row counts from source and target tables.

C. Run native PostgreSQL logical replication from the Db2 database to an Aurora DB cluster to evaluate the migration compatibility.

D. Run the AWS Schema Conversion Tool (AWS SCT) from the Db2 database to an Aurora DB cluster. Create a migration assessment report to evaluate the migration compatibility.

Answer: D

Explanation:

The correct answer is **D: Run the AWS Schema Conversion Tool (AWS SCT) from the Db2 database to an Aurora DB cluster. Create a migration assessment report to evaluate the migration compatibility.**

Here's why:

AWS SCT is designed for database migrations: AWS SCT specifically analyzes the source database schema (Db2 in this case) and automatically converts it to a compatible schema for the target database (Aurora PostgreSQL). It significantly reduces the time and effort involved in manually rewriting database code.

Migration Assessment Reports provide compatibility insights: AWS SCT generates comprehensive migration assessment reports that detail the compatibility of database objects, potential conversion issues, and recommended solutions. This report allows the company to quickly identify the scope and complexity of the migration, pinpointing which database objects can be converted automatically and which require manual intervention. This proactive assessment is vital for planning the migration and estimating the required resources.

Logical Dumps and Restoration (Option A) are time-consuming and less informative for initial assessment: While a logical dump and restore can eventually identify compatibility issues, it's a much more labor-intensive and less efficient approach for the initial assessment. The process involves physically moving data before understanding the compatibility, which is not ideal. Comparing row counts is useful after migration, not for the initial assessment of schema compatibility.

AWS DMS (Option B) is primarily for data migration, not schema conversion assessment: AWS DMS can migrate data between different database engines, but it's not its primary function to assess schema compatibility before the migration. While DMS might highlight some issues during the migration, it's not the fastest way to get a comprehensive compatibility assessment. Furthermore, AWS DMS depends on data types that are common between the source and target. If there are incompatible data types, AWS DMS will not be able to migrate the data.

Native PostgreSQL replication (Option C) is incompatible with Db2: PostgreSQL replication techniques are designed to replicate data from one PostgreSQL instance to another. It cannot be directly used with a Db2 database.

In summary: AWS SCT provides the quickest and most efficient way to gather data on migration compatibility because it is a dedicated tool for schema conversion and assessment, providing detailed reports that allow for informed decision-making before any data movement occurs. The other options are either incompatible or less efficient for the initial compatibility assessment.

Supporting Links:

AWS Schema Conversion Tool (SCT):https://aws.amazon.com/dms/schema-conversion-tool/ **AWS Database Migration Service (DMS):**https://aws.amazon.com/dms/

Question: 58

An ecommerce company is using Amazon DynamoDB as the backend for its order-processing application. The steady increase in the number of orders is resulting in increased DynamoDB costs. Order verification and reporting perform many repeated GetItem functions that pull similar datasets, and this read activity is contributing to the increased costs. The company wants to control these costs without significant development efforts.
How should a Database Specialist address these requirements?

A. Use AWS DMS to migrate data from DynamoDB to Amazon DocumentDB

B. Use Amazon DynamoDB Streams and Amazon Kinesis Data Firehose to push the data into Amazon Redshift

C. Use an Amazon ElastiCache for Redis in front of DynamoDB to boost read performance

D. Use DynamoDB Accelerator to offload the reads

Answer: D

Explanation:

The best solution to reduce DynamoDB costs related to repeated read operations for order verification and reporting, without extensive development, is to use DynamoDB Accelerator (DAX). DAX is a fully managed, highly available, in-memory cache for DynamoDB. It's designed to improve read performance from milliseconds to microseconds, reducing the load on the DynamoDB tables. By caching frequently accessed data, DAX minimizes the number of direct reads from DynamoDB, thus reducing the read capacity unit (RCU) consumption and associated costs.

Option A is incorrect because migrating to DocumentDB is a major development effort and a complete database change, which goes against the requirement to minimize development. Option B, involving DynamoDB Streams and Kinesis Data Firehose, is designed for real-time data ingestion into analytical databases like Redshift, and isn't suitable for addressing repeated read requests in the order-processing application itself. It adds complexity, increases costs, and involves more development. Option C, using ElastiCache for Redis, while a valid caching option, would require significant code changes to integrate it with the DynamoDB application to manage cache invalidation and consistency, which contradicts the "without significant development efforts" requirement.

DAX, on the other hand, integrates seamlessly with DynamoDB. It requires minimal code changes as the application continues to use the DynamoDB API. DAX sits in front of the DynamoDB table and caches the

results of GetItem operations, thus dramatically reducing the load on DynamoDB and its associated costs for repeated reads. DAX also handles cache invalidation automatically. DAX is specifically designed for this kind of scenario where you want to improve read performance and reduce DynamoDB costs without substantial changes to your application code.

Further Reading:

DynamoDB Accelerator (DAX)
DAX Best Practices

Question: 59

An IT consulting company wants to reduce costs when operating its development environment databases. The company's workflow creates multiple Amazon
Aurora MySQL DB clusters for each development group. The Aurora DB clusters are only used for 8 hours a day. The DB clusters can then be deleted at the end of the development cycle, which lasts 2 weeks.
Which of the following provides the MOST cost-effective solution?

A. Use AWS CloudFormation templates. Deploy a stack with the DB cluster for each development group. Delete the stack at the end of the development cycle.

B. Use the Aurora DB cloning feature. Deploy a single development and test Aurora DB instance, and create clone instances for the development groups. Delete the clones at the end of the development cycle.

C. Use Aurora Replicas. From the master automatic pause compute capacity option, create replicas for each development group, and promote each replica to master. Delete the replicas at the end of the development cycle.

D. Use Aurora Serverless. Restore current Aurora snapshot and deploy to a serverless cluster for each development group. Enable the option to pause the compute capacity on the cluster and set an appropriate timeout.

Answer: D

Explanation:

The best solution is D (Aurora Serverless) because it directly addresses the needs for cost reduction in development environments used only 8 hours a day for 2 weeks, with an understanding that they will be deleted afterwards. Aurora Serverless automatically starts, scales, and shuts down based on application needs, incurring costs only when the database is in use. By restoring from a snapshot, the environment can be quickly provisioned, and enabling the pause compute capacity option ensures that no costs are incurred during periods of inactivity. This aligns with the requirement to minimize cost during downtime.

Option A (CloudFormation) is a valid Infrastructure-as-Code (IaC) approach that helps automate DB cluster deployments, and deleting the stack when finished could lower costs. However, it doesn't inherently address periods of inactivity, incurring costs even when the DB cluster isn't being used. CloudFormation also requires more intervention and setup.

Option B (Aurora DB cloning) can provide a fast way to create database copies, but it's more suitable for creating test environments that reflect production data. It does not automatically scale down or pause when inactive, leading to higher costs. It also has some potential implications for data management and version control that are not discussed in the prompt.

Option C (Aurora Replicas) provides read scalability but isn't designed for development environments with short lifecycles. Promoting each replica to master adds complexity and operational overhead and doesn't solve the problem of cost optimization during inactive periods. Creating and managing replicas just to be used temporarily doesn't add real value in this case.

In summary, Aurora Serverless is most cost-effective since it only charges for actual usage and automatically

handles scaling, which aligns with the scenario's goal of minimizing costs during inactivity. The other options do not automatically manage costs during periods of inactivity or address the scenario requirements efficiently.

Aurora Serverless v1Aurora Serverless v2CloudFormation

Question: 60

A company has multiple applications serving data from a secure on-premises database. The company is migrating all applications and databases to the AWS
Cloud. The IT Risk and Compliance department requires that auditing be enabled on all secure databases to capture all log ins, log outs, failed logins, permission changes, and database schema changes. A Database Specialist has recommended Amazon Aurora MySQL as the migration target, and leveraging the Advanced Auditing feature in Aurora.
Which events need to be specified in the Advanced Auditing configuration to satisfy the minimum auditing requirements? (Choose three.)

A. CONNECT
B. QUERY_DCL
C. QUERY_DDL
D. QUERY_DML
E. TABLE
F. QUERY

Answer: ABC

Explanation:

The correct answer is ABC (CONNECT, QUERY_DCL, QUERY_DDL) because these events directly address the IT Risk and Compliance department's auditing requirements for the secure on-premises database migration to Aurora MySQL. Let's break down why:

A. CONNECT: This event captures all log ins and log outs, as well as failed login attempts, directly satisfying the requirement to audit connection activities. Monitoring connection events is crucial for identifying unauthorized access or potential security breaches.

B. QUERY_DCL: Data Control Language (DCL) commands are used to control access to data. Auditing these events allows tracking of permission changes, which is specifically required by the IT Risk and Compliance department. Examples include GRANT and REVOKE statements.

C. QUERY_DDL: Data Definition Language (DDL) commands are used to define the database schema. Auditing these events enables the tracking of database schema changes, such as creating, altering, or dropping tables. This is crucial for maintaining data integrity and complying with audit requirements.

Now let's analyze why the other options are not ideal:

D. QUERY_DML: Data Manipulation Language (DML) commands (e.g., SELECT, INSERT, UPDATE, DELETE) are used to manipulate data. While auditing DML can be useful, it's not specifically required by the prompt, which focuses on logins, permission changes, schema changes, and failed logins. Auditing all DML statements could create excessively large audit logs without addressing the core compliance needs.

E. TABLE: While related to the database, simply auditing tables does not give insights into the specific events requested by IT Risk and Compliance. Knowing a table exists, or has been accessed, does not tell you who logged in, when permissions changed, or how the table schema was altered.

F. QUERY: This option is too generic. While it would capture all queries, including DDL, DCL, and DML, it would lead to excessive logging and make it difficult to focus on the specific audit requirements of the prompt.

Therefore, the combination of CONNECT, QUERY_DCL, and QUERY_DDL events provides the minimum set of events needed to satisfy the audit requirements related to logins, permission changes, failed logins, and database schema changes. This selective auditing optimizes log volume while ensuring compliance.

Further Research:

Amazon Aurora MySQL Auditing:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Auditing.html
MySQL Auditing:https://dev.mysql.com/doc/refman/8.0/en/audit-log.html

Question: 61

A gaming company has recently acquired a successful iOS game, which is particularly popular during the holiday season. The company has decided to add a leaderboard to the game that uses Amazon DynamoDB. The application load is expected to ramp up over the holiday season.
Which solution will meet these requirements at the lowest cost?

A. DynamoDB Streams
B. DynamoDB with DynamoDB Accelerator
C. DynamoDB with on-demand capacity mode
D. DynamoDB with provisioned capacity mode with Auto Scaling

Answer: D

Explanation:

The correct answer is **D: DynamoDB with provisioned capacity mode with Auto Scaling**. Here's why:

Cost Optimization: The key requirement is the lowest cost. DynamoDB offers two capacity modes: on-demand and provisioned. On-demand is excellent for unpredictable workloads but is often more expensive for predictable or ramping workloads.

Provisioned Capacity: Provisioned capacity allows you to specify the read and write capacity units (RCUs and WCUs) your application requires. This gives you direct control over costs.

Auto Scaling for Elasticity: The workload is expected to ramp up during the holiday season. DynamoDB Auto Scaling dynamically adjusts provisioned capacity based on real-time application traffic. This ensures optimal performance during peak periods and reduces costs during off-peak periods by scaling down capacity.

Why other options are less optimal:

A. DynamoDB Streams: DynamoDB Streams captures data modification events in near real-time. This isn't related to the question's core requirement of cost-effectively managing fluctuating leaderboard traffic. It would add cost without solving the capacity management issue.

B. DynamoDB with DynamoDB Accelerator (DAX): DAX is an in-memory cache for DynamoDB. It's helpful for read-heavy workloads requiring microsecond latency, but it adds cost. While DAX could improve leaderboard read performance, it doesn't address the fundamental need to scale capacity cost-effectively with predictable ramps. Furthermore, caching is often better implemented at the application layer or with a service like ElastiCache.

C. DynamoDB with on-demand capacity mode: While easy to use, on-demand capacity mode scales automatically but charges per request. For a workload that predictably ramps up, this is less cost-effective than provisioning capacity and using Auto Scaling because you pay for every single request, even if that request could have been served with cheaper, provisioned capacity.

In summary: Provisioned capacity with Auto Scaling provides the best balance between performance, cost

control, and automatic scaling for the specified workload. It allows for fine-grained control over resource allocation while dynamically adjusting to changing demand.

Authoritative Links:

DynamoDB Capacity Modes:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html
DynamoDB Auto Scaling:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.html

Question: 62

A company's Security department established new requirements that state internal users must connect to an existing Amazon RDS for SQL Server DB instance using their corporate Active Directory (AD) credentials. A Database Specialist must make the modifications needed to fulfill this requirement.
Which combination of actions should the Database Specialist take? (Choose three.)

A. Disable Transparent Data Encryption (TDE) on the RDS SQL Server DB instance.

B. Modify the RDS SQL Server DB instance to use the directory for Windows authentication. Create appropriate new logins.

C. Use the AWS Management Console to create an AWS Managed Microsoft AD. Create a trust relationship with the corporate AD.

D. Stop the RDS SQL Server DB instance, modify it to use the directory for Windows authentication, and start it again. Create appropriate new logins.

E. Use the AWS Management Console to create an AD Connector. Create a trust relationship with the corporate AD.

F. Configure the AWS Managed Microsoft AD domain controller Security Group.

Answer: BCF

Explanation:

The correct answer is BCF. Here's a detailed justification:

B. Modify the RDS SQL Server DB instance to use the directory for Windows authentication. Create appropriate new logins. This is a core step in enabling AD authentication for RDS SQL Server. RDS allows you to associate a directory with your SQL Server DB instance, allowing users from the domain to authenticate. After associating the directory, you need to create SQL Server logins mapped to the AD users or groups that require access.

C. Use the AWS Management Console to create an AWS Managed Microsoft AD. Create a trust relationship with the corporate AD. To enable internal users to authenticate using their corporate AD credentials, you need a mechanism to bridge the gap between AWS and the on-premises or hosted corporate AD. AWS Managed Microsoft AD is a fully managed service for running Active Directory on AWS. By creating an AWS Managed Microsoft AD and establishing a trust relationship with the corporate AD, users in the corporate AD can be authenticated within the AWS environment. This eliminates the need to create and manage a separate AD instance in AWS.

F. Configure the AWS Managed Microsoft AD domain controller Security Group. The security group associated with the AWS Managed Microsoft AD domain controllers controls the network traffic allowed to and from those controllers. To allow RDS SQL Server instances to communicate with the AD for
authentication purposes, the security group needs to be configured to allow traffic from the RDS instance to the AD domain controllers (specifically, DNS and Kerberos traffic). This enables the RDS instance to contact the AD for authentication requests.

Why the other options are incorrect:

A. Disable Transparent Data Encryption (TDE) on the RDS SQL Server DB instance: TDE is a data-at-rest encryption feature and has no direct relationship to Active Directory authentication. Disabling TDE does not contribute to the required outcome.

D. Stop the RDS SQL Server DB instance, modify it to use the directory for Windows authentication, and start it again. Create appropriate new logins: While the directory association does sometimes require a restart depending on changes being made, a simple directory association should not require stopping the instance completely. This approach adds unnecessary downtime.

E. Use the AWS Management Console to create an AD Connector. Create a trust relationship with the corporate AD: AD Connector is used to proxy requests to your existing on-premises Active Directory, while Managed AD gives you a standalone AD that you can trust with your existing on-premises AD. Since the prompt implies users connect to the corporate AD, it is better to use a managed directory in this situation and create a trust.

Supporting Documentation:

Using Windows Authentication with an RDS for SQL Server DB Instance:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/SQLServer.Procedural.UsingWithSQLServer.IAM.html **AWS Managed Microsoft AD:**https://aws.amazon.com/directoryservice/managed-microsoft-ad/
AD Connector:https://aws.amazon.com/directoryservice/ad-connector/

Question: 63

A Database Specialist is performing a proof of concept with Amazon Aurora using a small instance to confirm a simple database behavior. When loading a large dataset and creating the index, the Database Specialist encounters the following error message from Aurora:
ERROR: cloud not write block 7507718 of temporary file: No space left on device
What is the cause of this error and what should the Database Specialist do to resolve this issue?

A. The scaling of Aurora storage cannot catch up with the data loading. The Database Specialist needs to modify the workload to load the data slowly.

B. The scaling of Aurora storage cannot catch up with the data loading. The Database Specialist needs to enable Aurora storage scaling.

C. The local storage used to store temporary tables is full. The Database Specialist needs to scale up the instance.

D. The local storage used to store temporary tables is full. The Database Specialist needs to enable local storage scaling.

Answer: C

Explanation:

The error message "ERROR: could not write block 7507718 of temporary file: No space left on device" in the context of Aurora, particularly during index creation or large dataset loading, indicates that the instance's local storage, which is used for temporary files, has run out of space. Aurora's storage scaling primarily refers to the underlying data volume and doesn't directly impact the local temporary storage attached to the instance.

Option A is incorrect because Aurora storage scaling does catch up, but it's irrelevant to the error at hand. The error relates to local temporary storage. Option B is incorrect for the same reason: enabling Aurora storage scaling won't solve a local storage issue. Option D is also incorrect because Aurora doesn't offer local storage scaling as a feature. The local storage space is tied to the instance size.

Option C correctly identifies the problem. Aurora (and PostgreSQL in general) uses local storage for temporary tables and files during operations like index creation or complex queries that require sorting or intermediate result sets. When creating an index on a large dataset, the database engine often creates temporary tables to efficiently build the index. If the local storage on the instance is insufficient, this error will occur.

The solution is to scale up the instance to a larger size. Larger instances come with more local storage. This provides the temporary space needed to complete the index creation or large data load.

For further research, refer to the AWS Aurora documentation regarding instance types and their specifications, which include details on local storage:

Amazon Aurora Instance Types:https://aws.amazon.com/rds/aurora/instance-types/
PostgreSQL Temporary Files: This isn't AWS-specific, but explains the general concept of temporary files used by PostgreSQL, the engine often used by Aurora PostgreSQL:
https://www.postgresql.org/docs/current/runtime-config-resource.html (Search for "temp_file_limit").

Question: 64

A financial company wants to store sensitive user data in an Amazon Aurora PostgreSQL DB cluster. The database will be accessed by multiple applications across the company. The company has mandated that all communications to the database be encrypted and the server identity must be validated. Any non-SSL- based connections should be disallowed access to the database.
Which solution addresses these requirements?

A. Set the rds.force_ssl=0 parameter in DB parameter groups. Download and use the Amazon RDS certificate bundle and configure the PostgreSQL connection string with sslmode=allow.

B. Set the rds.force_ssl=1 parameter in DB parameter groups. Download and use the Amazon RDS certificate bundle and configure the PostgreSQL connection string with sslmode=disable.

C. Set the rds.force_ssl=0 parameter in DB parameter groups. Download and use the Amazon RDS certificate bundle and configure the PostgreSQL connection string with sslmode=verify-ca.

D. Set the rds.force_ssl=1 parameter in DB parameter groups. Download and use the Amazon RDS certificate bundle and configure the PostgreSQL connection string with sslmode=verify-full.

Answer: D

Explanation:

Here's a detailed justification for why option D is the correct answer:

The financial company requires two key security measures: encrypted communication to the Aurora PostgreSQL database and validation of the server's identity. To enforce encryption, all non-SSL connections must be disallowed.

rds.force_ssl=1: This setting within the DB parameter group is crucial. Setting it to 1 forces all connections to the Aurora PostgreSQL instance to use SSL/TLS encryption. Any connection attempt that doesn't use SSL will be rejected.

Amazon RDS Certificate Bundle: This bundle contains the root certificate authority (CA) certificates that your client applications need to verify the authenticity of the RDS instance (in this case, the Aurora PostgreSQL cluster). Without this, your application can't be sure it's actually talking to the legitimate database and not a malicious imposter.

sslmode=verify-full: This setting in the PostgreSQL connection string is the most secure option for SSL/TLS connections. Here's why:

sslmode=verify-ca: This option verifies that the server certificate is signed by a trusted CA (one included in your certificate bundle). However, it doesn't verify that the hostname in the certificate matches the hostname you're connecting to. This leaves you vulnerable to man-in-the-middle attacks if DNS is compromised.

sslmode=verify-full: This option adds hostname verification to the checks performed by verify-ca. It ensures that the hostname in the server's certificate exactly matches the hostname you specified in the connection string. This provides the strongest level of protection against man-in-the-middle attacks.

Options A and C are incorrect because they set rds.force_ssl=0, which means SSL is not required. This directly contradicts the company's requirement to disallow non-SSL connections. Option B forces SSL but uses sslmode=disable, meaning SSL is disabled. This defeats the purpose of requiring encrypted connections.

Option D, using rds.force_ssl=1 and sslmode=verify-full, correctly addresses both requirements: it enforces encryption for all connections and verifies the server's identity to prevent man-in-the-middle attacks, offering the highest level of security.

Authoritative Links:

Using SSL/TLS to Encrypt a Connection to a DB Instance:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html
PostgreSQL SSL Support:https://www.postgresql.org/docs/current/libpq-ssl.html
Setting up TLS/SSL on your cluster:https://aws.amazon.com/blogs/database/setting-up-tls-ssl-on-your-cluster/

Question: 65

A company is using 5 TB Amazon RDS DB instances and needs to maintain 5 years of monthly database backups for compliance purposes. A Database
Administrator must provide Auditors with data within 24 hours.
Which solution will meet these requirements and is the MOST operationally efficient?

A. Create an AWS Lambda function to run on the first day of every month to take a manual RDS snapshot. Move the snapshot to the company's Amazon S3 bucket.

B. Create an AWS Lambda function to run on the first day of every month to take a manual RDS snapshot.

C. Create an RDS snapshot schedule from the AWS Management Console to take a snapshot every 30 days.

D. Create an AWS Lambda function to run on the first day of every month to create an automated RDS snapshot.

Answer: A

Explanation:

The correct answer is **A**. Here's why:

Explanation:

Compliance Requirement: The company needs to maintain 5 years of monthly backups, which translates to 60 backups per database. They also need to restore within 24 hours.

Option A (Lambda + S3): This option leverages a serverless approach using Lambda to automate the snapshot creation process on the first day of each month. Storing snapshots in S3 provides durable, scalable, and cost-effective storage for long-term retention, meeting the 5-year requirement. S3's versioning and lifecycle policies can be used to manage older snapshots. Moving snapshots to S3 ensures that the primary RDS instance isn't overloaded with holding a huge backup catalogue, improving operational efficiency.

Option B (Lambda only): While Lambda is used for snapshot creation, it doesn't mention where the snapshots are stored. RDS automated backups are also snapshots; however, RDS has limits on the number of stored

automated backups. Storing them directly in RDS would quickly reach storage limits and hinder the 5-year retention requirement.
Option C (RDS snapshot schedule): The question specifies that the snapshots should be monthly, and the snapshots should be taken on the first day of every month. RDS snapshot schedules allow for daily, weekly, or monthly snapshots. However, creating RDS snapshots every 30 days wouldn't necessarily run on the first day of the month.
Option D (Lambda + automated snapshots): RDS offers automated backups (which are also snapshots). The question specifies that the snapshots should be monthly, and the snapshots should be taken on the first day of every month. The granularity of automated backups is based on days, and not a specified day of the month. Also, the user is restricted to storing only a limited number of automated backups in RDS.

Why A is the most operationally efficient:

Automation: Lambda automates the entire backup process, reducing manual effort and potential errors.
Scalability: S3 provides virtually unlimited storage capacity, easily accommodating the growing number of backups over the 5-year period.
Cost-effectiveness: S3's storage tiers (e.g., Glacier) can be utilized for older backups to further optimize storage costs.
Restore speed: Restoring a database from S3 requires creating an RDS instance from the snapshot in S3, which could be done relatively quickly to satisfy the 24-hour restoration requirement.

Authoritative Links:

AWS Lambda: AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers.
Amazon S3: Amazon S3 is object storage built to store and retrieve any amount of data from anywhere.
Amazon RDS Backups: Documentation on automated backups and manual snapshots in Amazon RDS.

Question: 66

A company wants to automate the creation of secure test databases with random credentials to be stored safely for later use. The credentials should have sufficient information about each test database to initiate a connection and perform automated credential rotations. The credentials should not be logged or stored anywhere in an unencrypted form. Which steps should a Database Specialist take to meet these requirements using an AWS CloudFormation template?

A. Create the database with the MasterUserName and MasterUserPassword properties set to the default values. Then, create the secret with the user name and password set to the same default values. Add a Secret Target Attachment resource with the SecretId and TargetId properties set to the Amazon Resource Names (ARNs) of the secret and the database. Finally, update the secret's password value with a randomly generated string set by the GenerateSecretString property.

B. Add a Mapping property from the database Amazon Resource Name (ARN) to the secret ARN. Then, create the secret with a chosen user name and a randomly generated password set by the GenerateSecretString property. Add the database with the MasterUserName and MasterUserPassword properties set to the user name of the secret.

C. Add a resource of type AWS::SecretsManager::Secret and specify the GenerateSecretString property. Then, define the database user name in the SecureStringTemplate template. Create a resource for the database and reference the secret string for the MasterUserName and MasterUserPassword properties. Then, add a resource of type AWS::SecretsManagerSecretTargetAttachment with the SecretId and TargetId properties set to the Amazon Resource Names (ARNs) of the secret and the database.

D. Create the secret with a chosen user name and a randomly generated password set by the GenerateSecretString property. Add an SecretTargetAttachment resource with the SecretId property set to the Amazon Resource Name (ARN) of the secret and the TargetId property set to a parameter value matching the desired database ARN. Then, create a database with the MasterUserName and MasterUserPassword properties set to the previously created values in the secret.

Answer: C

Explanation:

The correct answer is C because it utilizes AWS Secrets Manager to generate and manage database credentials securely within a CloudFormation template, addressing the key requirements of the question.

Here's a detailed breakdown:

AWS Secrets Manager: Secrets Manager is specifically designed for managing secrets like database credentials. It allows you to generate, rotate, and retrieve credentials securely. This directly addresses the requirement to store credentials safely and prevent logging in unencrypted form.

AWS::SecretsManager::Secret resource: This resource allows you to define a secret within your CloudFormation template. The critical part is the GenerateSecretString property, which automatically generates a random, strong password for the database.

SecureStringTemplate: This element enables you to embed the database username (defined separately for customization if needed) within the automatically generated secret. This allows Secrets Manager to create a complete, properly formatted credential string that the database can understand.

Referencing the Secret in the Database Resource: By referencing the secret ARN and the SecretString output (which contains both username and password in JSON format) in the database resource's MasterUserName and MasterUserPassword properties, you automatically configure the database to use the generated credentials. CloudFormation handles the resolution of these references during stack creation.

AWS::SecretsManagerSecretTargetAttachment: This resource associates the secret with the target resource, which in this case is the database instance. This association enables Secrets Manager to perform tasks such as automatic secret rotation.

Let's address why the other options are less suitable:

Option A: Using default values initially is a significant security risk and contradicts the requirement of not storing credentials in an unencrypted form. It creates a window of vulnerability before the password is rotated.

Option B: Introducing a Mapping property from the database ARN to the secret ARN is an unnecessary complexity. CloudFormation allows directly referencing secret values when creating the database resource. Also, it doesn't leverage SecretTargetAttachment for rotation.

Option D: Creating the secret first and then using the ARN as a parameter is not wrong, but is less efficient than using a secret from the beginning. Most importantly, it neglects to use AWS::SecretsManagerSecretTargetAttachment, which is crucial for database rotations.

In summary, Option C offers the most secure and automated approach by leveraging AWS Secrets Manager to generate and manage database credentials directly within the CloudFormation template, ensuring credentials are never stored or logged in plain text and facilitating automated credential rotation.

Authoritative Links:

AWS Secrets Manager:https://aws.amazon.com/secrets-manager/
AWS::SecretsManager::Secret CloudFormation Documentation:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-secretsmanager-secret.html
AWS::SecretsManager::SecretTargetAttachment CloudFormation Documentation:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-secretsmanager-secrettargetattachment.html

Question: 67

A company is going to use an Amazon Aurora PostgreSQL DB cluster for an application backend. The DB cluster contains some tables with sensitive data. A
Database Specialist needs to control the access privileges at the table level.
How can the Database Specialist meet these requirements?

A. Use AWS IAM database authentication and restrict access to the tables using an IAM policy.
B. Configure the rules in a NACL to restrict outbound traffic from the Aurora DB cluster.
C. Execute GRANT and REVOKE commands that restrict access to the tables containing sensitive data.
D. Define access privileges to the tables containing sensitive data in the pg_hba.conf file.

Answer: C

Explanation:

The correct answer is C: Execute GRANT and REVOKE commands that restrict access to the tables containing sensitive data. This is because Aurora PostgreSQL, like standard PostgreSQL, provides robust role-based access control within the database system itself. The GRANT and REVOKE commands are the standard SQL commands for assigning and removing privileges to database objects, including tables, to specific database roles or users. These commands allow fine-grained control over which users or roles can SELECT, INSERT, UPDATE, DELETE, or perform other actions on specific tables.

Option A is incorrect. While IAM database authentication allows users to authenticate using IAM roles, IAM policies primarily control access to AWS services and resources outside of the database. They do not directly control table-level privileges within the Aurora PostgreSQL database itself. IAM policies can authenticate a user to the database but the GRANT and REVOKE commands must be used within the database to control table level access.

Option B is incorrect. NACLs (Network Access Control Lists) control network traffic at the subnet level and are not used for managing database access privileges. NACLs operate at layer 3 and 4 of the OSI model and don't understand SQL or table structures.

Option D is incorrect. The pg_hba.conf file configures client authentication methods (e.g., password, IAM) and connection parameters, but it does not define specific privileges for tables. It is more concerned with how a user authenticates to the database, rather than what they can do once authenticated. Table-level permissions are managed separately using GRANT and REVOKE.

In summary, to control table-level access privileges within an Aurora PostgreSQL DB cluster, the best and most direct method is to use SQL GRANT and REVOKE commands. This adheres to standard database security practices and offers granular control over user access to sensitive data.

Further Research:

PostgreSQL GRANT Command:https://www.postgresql.org/docs/current/sql-grant.html
PostgreSQL REVOKE Command:https://www.postgresql.org/docs/current/sql-revoke.html
IAM Database Authentication for PostgreSQL:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.html
Controlling Database Access with PostgreSQL Roles:https://www.postgresql.org/docs/current/tutorial-privileges.html

Question: 68

A Database Specialist is working with a company to launch a new website built on Amazon Aurora with several Aurora Replicas. This new website will replace an on-premises website connected to a legacy relational database.
Due to stability issues in the legacy database, the company would like to test the resiliency of Aurora.

Which action can the Database Specialist take to test the resiliency of the Aurora DB cluster?

A. Stop the DB cluster and analyze how the website responds
B. Use Aurora fault injection to crash the master DB instance
C. Remove the DB cluster endpoint to simulate a master DB instance failure
D. Use Aurora Backtrack to crash the DB cluster

Answer: B

Explanation:

The correct answer is **B. Use Aurora fault injection to crash the master DB instance.**

Here's a detailed justification:

Aurora fault injection is specifically designed to test the resiliency of Aurora clusters. This feature allows administrators to deliberately introduce failures into the Aurora DB cluster in a controlled manner. By crashing the master DB instance, the Database Specialist can observe how Aurora handles the failure and whether the application automatically fails over to a read replica, promoting it to become the new master. This directly tests the High Availability (HA) capabilities of Aurora and verifies the failover mechanisms are functioning correctly. It demonstrates if the website will remain available (though potentially with a temporary write interruption) in the event of a primary DB instance failure.

Option A is incorrect because simply stopping the DB cluster doesn't simulate a real-world failure. It's a controlled shutdown and doesn't test the automated failover capabilities.

Option C is not ideal. Removing the DB cluster endpoint will disrupt connectivity but does not truly mimic a master DB instance failure. The endpoint automatically updates during failover and can lead to confusion when troubleshooting. Aurora fault injection provides a more realistic scenario.

Option D, using Aurora Backtrack, is for rewinding the database to a specific point in time to recover from data corruption or incorrect transactions. It doesn't test the HA and failover aspects of the Aurora cluster.

Therefore, using Aurora fault injection is the most effective and direct way to test the resiliency of the Aurora DB cluster by simulating a master DB instance failure and verifying the automated failover to a read replica.

This approach provides insights into the system's behavior under stress and ensures the website remains available during disruptions.

Further research:

AWS Documentation on Aurora Fault Injection:https://aws.amazon.com/blogs/database/simulating-failures-in-amazon-aurora-to-build-resilient-applications/
Aurora High Availability:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Concepts.AuroraHighAvailability.html

Question: 69

A company just migrated to Amazon Aurora PostgreSQL from an on-premises Oracle database. After the migration, the company discovered there is a period of time every day around 3:00 PM where the response time of the application is noticeably slower. The company has narrowed down the cause of this issue to the database and not the application.

Which set of steps should the Database Specialist take to most efficiently find the problematic PostgreSQL query?

A. Create an Amazon CloudWatch dashboard to show the number of connections, CPU usage, and disk space consumption. Watch these dashboards during the next slow period.
B. Launch an Amazon EC2 instance, and install and configure an open-source PostgreSQL monitoring tool that

will run reports based on the output error logs.

C. Modify the logging database parameter to log all the queries related to locking in the database and then check the logs after the next slow period for this information.

D. Enable Amazon RDS Performance Insights on the PostgreSQL database. Use the metrics to identify any queries that are related to spikes in the graph during the next slow period.

Answer: D

Explanation:

The best approach to identify the problematic PostgreSQL query causing performance slowdowns after the migration to Aurora PostgreSQL is **D. Enable Amazon RDS Performance Insights on the PostgreSQL database. Use the metrics to identify any queries that are related to spikes in the graph during the next slow period.**

Here's why:

Performance Insights is designed for database performance analysis: Amazon RDS Performance Insights is a database performance monitoring and tuning tool specifically created to analyze database load. It visualizes database load, enabling you to quickly identify performance bottlenecks and take informed actions.

https://aws.amazon.com/rds/performance-insights/

Granular query-level analysis: Performance Insights goes beyond general resource utilization metrics. It can pinpoint the exact SQL queries that are consuming the most database resources (CPU, I/O), making it easy to find the culprit. It captures the top SQL queries by wait events, providing actionable insights.

Time-series visualization: Performance Insights displays database load as a time-series graph, making it easy to correlate performance spikes with specific queries during the 3:00 PM slowdown.

Ease of Use: Enabling Performance Insights is simple with a few clicks in the AWS Management Console. It automatically collects and analyzes performance data without requiring complex configuration.

Why other options are less efficient:

A. CloudWatch dashboards: While useful for general monitoring, CloudWatch provides system-level metrics (CPU, memory, disk) but lacks query-level detail. You might see high CPU usage but not know which queries are causing it.

B. EC2 instance with PostgreSQL monitoring tool: This approach is complex and time-consuming. It involves setting up an EC2 instance, installing and configuring a monitoring tool, and analyzing logs manually. There's a learning curve and ongoing maintenance overhead.

C. Logging all locking queries: While useful in specific locking scenarios, this approach generates a large volume of logs that are difficult to sift through. It doesn't give you an overall view of the database load during the slowdown. Also, the performance issue might be related to a different problem than locking.

Therefore, using Amazon RDS Performance Insights is the fastest and most targeted way to identify the slow-performing PostgreSQL query, making it the most efficient solution.

Question: 70

A company has a web-based survey application that uses Amazon DynamoDB. During peak usage, when survey responses are being collected, a Database
Specialist sees the ProvisionedThroughputExceededException error.
What can the Database Specialist do to resolve this error? (Choose two.)

A. Change the table to use Amazon DynamoDB Streams

B. Purchase DynamoDB reserved capacity in the affected Region

C. Increase the write capacity units for the specific table

D. Change the table capacity mode to on-demand

E. Change the table type to throughput optimized

Answer: CD

Explanation:

The ProvisionedThroughputExceededException in DynamoDB indicates that the application is attempting to read or write data faster than the provisioned capacity allows. The goal is to enable the application to handle peak loads without encountering this exception.

Option C, "Increase the write capacity units for the specific table," directly addresses the problem. DynamoDB uses write capacity units (WCUs) to measure the write throughput. By increasing the WCUs, the table can handle more write requests per second, thus resolving the ProvisionedThroughputExceededException during write-heavy periods like survey response collection. This is a fundamental scaling strategy for provisioned capacity mode.

Option D, "Change the table capacity mode to on-demand," provides an alternative solution. DynamoDB on-demand capacity mode automatically scales capacity to accommodate the application's workload. Switching to on-demand eliminates the need to manually provision and manage capacity, as DynamoDB handles scaling up and down based on actual usage. This removes the risk of exceeding provisioned throughput, as the system dynamically allocates resources.

Option A, "Change the table to use Amazon DynamoDB Streams," is incorrect. DynamoDB Streams captures data modification events in a table. While Streams can be used for other purposes like data replication or triggering Lambda functions, it does not directly address the ProvisionedThroughputExceededException. It's a way to react to data changes, not a way to increase write throughput.

Option B, "Purchase DynamoDB reserved capacity in the affected Region," is also not a direct solution to the problem. Reserved capacity provides discounted pricing for a pre-determined amount of throughput but does not automatically scale to handle unexpected peaks beyond the reserved amount. It requires planning and may still lead to throughput exceptions if not correctly provisioned.

Option E, "Change the table type to throughput optimized," is misleading. DynamoDB doesn't have different table types based on throughput optimization. The throughput capacity of DynamoDB tables is determined by the chosen capacity mode (provisioned or on-demand) and configured WCUs or RCUs, not by a predefined "table type."

Therefore, the correct options are C and D. They either increase the available write capacity (C) or switch to a mode where capacity is automatically scaled on demand (D), both effectively resolving the ProvisionedThroughputExceededException.

Relevant Links:

DynamoDB Provisioned Capacity Mode
DynamoDB On-Demand Capacity Mode
DynamoDB Streams

Question: 71

A company is running a two-tier ecommerce application in one AWS account. The web server is deployed using an Amazon RDS for MySQL Multi-AZ DB instance. A Developer mistakenly deleted the database in the production environment. The database has been restored, but this resulted in hours of downtime and lost revenue. Which combination of changes in existing IAM policies should a Database Specialist make to prevent an error like this from happening in the future? (Choose three.)

A. Grant least privilege to groups, users, and roles

B. Allow all users to restore a database from a backup that will reduce the overall downtime to restore the database

C. Enable multi-factor authentication for sensitive operations to access sensitive resources and API operations

D. Use policy conditions to restrict access to selective IP addresses

E. Use AccessList Controls policy type to restrict users for database instance deletion

F. Enable AWS CloudTrail logging and Enhanced Monitoring

Answer: ACD

Explanation:

The correct answer is ACD. Here's why:

A. Grant least privilege to groups, users, and roles: The principle of least privilege is a core security practice. By only granting the necessary permissions to perform a specific task, you minimize the risk of accidental or malicious actions. In this case, the Developer deleting the database likely had more permissions than required.
Restricting permissions to only what is absolutely needed for their role would prevent accidental deletions.
https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html

C. Enable multi-factor authentication for sensitive operations to access sensitive resources and API operations: Multi-Factor Authentication (MFA) adds an extra layer of security. Requiring MFA for database deletion or other critical operations makes it significantly harder for unauthorized actions to occur, even if someone's credentials are compromised. This provides an additional check before a destructive operation like deleting a database can proceed.
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa.html

D. Use policy conditions to restrict access to selective IP addresses: Restricting access based on IP addresses limits who can access the AWS resources based on where the request is coming from. If database administration should only be done from the corporate network or a designated jump server, enforcing this condition can block deletion requests originating from unauthorized locations or personal devices that may be compromised. This adds a location-based security control.
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-keys.html#condition-keys-sourceip

Let's look at why the other options are not the best choices:

B. Allow all users to restore a database from a backup that will reduce the overall downtime to restore the database: While faster restoration is desirable, allowing all users to restore a database introduces significant security risks. Restoration can overwrite existing data or create new attack vectors if not properly controlled. It violates the principle of least privilege.

E. Use AccessList Controls policy type to restrict users for database instance deletion: "AccessList Controls policy type" is not a standard IAM policy type in AWS. IAM policies are based on JSON documents defining permissions. While you can control who can delete database instances, it's done through standard IAM policies, not a specific "AccessList Controls" type.

F. Enable AWS CloudTrail logging and Enhanced Monitoring: While enabling AWS CloudTrail logging is a crucial security practice for auditing and identifying who deleted the database (and is always a good practice), it doesn't prevent the deletion from happening in the first place. Enhanced Monitoring provides performance insights but also doesn't help prevent actions. They are reactive, not proactive measures for preventing deletions.

Question: 72

A company is building a new web platform where user requests trigger an AWS Lambda function that performs an insert into an Amazon Aurora MySQL DB cluster. Initial tests with less than 10 users on the new platform yielded successful execution and fast response times. However, upon more extensive tests with the actual target of 3,000 concurrent users, Lambda functions are unable to connect to the DB cluster and receive too many connections errors.
Which of the following will resolve this issue?

A. Edit the my.cnf file for the DB cluster to increase max_connections
B. Increase the instance size of the DB cluster
C. Change the DB cluster to Multi-AZ
D. Increase the number of Aurora Replicas

Answer: B

Explanation:

Here's a detailed justification for why increasing the instance size of the Aurora MySQL DB cluster (Option B) is the most appropriate solution to the "too many connections" error in this scenario, along with why the other options are less effective or inappropriate:

The core issue is that the Aurora MySQL DB cluster is being overwhelmed by the number of concurrent connections initiated by the Lambda functions. Each concurrent user triggers a Lambda function, and each Lambda function attempts to establish a new connection to the database to perform the insert operation. With 3,000 concurrent users, the database instance is simply running out of resources to handle all those connection requests.

Why Option B (Increase the instance size of the DB cluster) is the best solution:

Increased Resources: Increasing the instance size of the Aurora DB cluster directly addresses the resource constraint. A larger instance has more CPU, memory, and network bandwidth. These increased resources allow the database to handle more concurrent connections without becoming overloaded. Specifically, memory is a key factor as each connection consumes memory for buffers and other overhead.

Connection Capacity: Larger instances generally have higher connection limits than smaller instances. While the exact connection limits depend on the instance type, upgrading to a larger instance provides the necessary headroom to accommodate the 3,000 concurrent users.

Performance Improvement: A larger instance also improves overall database performance, leading to faster query execution and reduced latency. This contributes to a better user experience and reduces the likelihood of connection timeouts.

Why other options are less effective or inappropriate:

Option A (Edit the my.cnf file for the DB cluster to increase max_connections): While increasing max_connections appears to solve the immediate problem, it's often a superficial fix. Simply raising the connection limit without addressing the underlying resource limitations can lead to other problems, such as:

Resource Contention: The database server may become overloaded and unresponsive as it tries to manage too many connections with limited resources. This can lead to performance degradation and even crashes. **Memory Exhaustion:** Each connection consumes memory. Simply increasing max_connections can lead to memory exhaustion, which can cripple the database.

Therefore, it's generally preferable to address resource constraints first before tweaking configuration settings like max_connections.

Option C (Change the DB cluster to Multi-AZ): Multi-AZ deployments enhance availability and provide failover capabilities. They create a standby replica in a different Availability Zone. However, Multi-AZ does not address the issue of the database instance being overwhelmed by the number of concurrent connections. It's primarily for disaster recovery and fault tolerance, not for scaling connection capacity. The standby replica doesn't handle the primary workload in a healthy cluster.

Option D (Increase the number of Aurora Replicas): Aurora Replicas are read-only replicas of the primary DB instance. They can be used to offload read traffic, but they do not handle write operations (like the INSERT in this scenario). Since the Lambda functions are performing insert operations, Aurora Replicas are irrelevant to resolving the "too many connections" error. Furthermore, adding replicas doesn't increase write capacity of the primary node that handles the INSERT operations.

In summary, increasing the instance size is the most direct and effective way to provide the resources needed to handle the increased number of concurrent database connections caused by the high volume of Lambda function invocations.

Supporting Documentation:

Amazon Aurora Connection Management:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/AuroraMySQL.Managing.Connections.html
(Explains Aurora's approach to connection handling and its relationship to instance size).
Choosing the Right Instance Size:https://aws.amazon.com/rds/instance-types/ (Provides guidance on selecting the appropriate instance size based on workload requirements).

Question: 73

A company is developing a multi-tier web application hosted on AWS using Amazon Aurora as the database. The application needs to be deployed to production and other non-production environments. A Database Specialist needs to specify different MasterUsername and MasterUserPassword properties in the AWS
CloudFormation templates used for automated deployment. The CloudFormation templates are version controlled in the company's code repository. The company also needs to meet compliance requirement by routinely rotating its database master password for production.
What is most secure solution to store the master password?

A. Store the master password in a parameter file in each environment. Reference the environment-specific parameter file in the CloudFormation template.

B. Encrypt the master password using an AWS KMS key. Store the encrypted master password in the CloudFormation template.

C. Use the secretsmanager dynamic reference to retrieve the master password stored in AWS Secrets Manager and enable automatic rotation.

D. Use the ssm dynamic reference to retrieve the master password stored in the AWS Systems Manager Parameter Store and enable automatic rotation.

Answer: C

Explanation:

The most secure solution is **C. Use the secretsmanager dynamic reference to retrieve the master password stored in AWS Secrets Manager and enable automatic rotation.**

Here's why:

Secrets Manager is designed for secrets: AWS Secrets Manager is specifically built to securely store and manage secrets, including database credentials. It provides encryption at rest and in transit, access control policies, and auditing capabilities. https://aws.amazon.com/secrets-manager/
Automatic Rotation: Secrets Manager allows for automatic rotation of database passwords. This significantly

reduces the risk associated with compromised credentials as the password is automatically changed on a schedule without manual intervention. This satisfies the compliance requirement for routine password rotation.
https://docs.aws.amazon.com/secretsmanager/latest/userguide/rotating-secrets.html

Dynamic References: CloudFormation supports dynamic references to Secrets Manager. This means you can reference a secret stored in Secrets Manager directly within your CloudFormation template without hardcoding the password or storing it in plain text. The password will only be accessible during stack creation or update and retrieved by AWS CloudFormation service.

Parameter Store vs. Secrets Manager: While AWS Systems Manager Parameter Store can store secrets, it's primarily designed for storing configuration data. Secrets Manager provides more robust security features, including encryption, access control, auditing, and automated rotation, making it the superior choice for sensitive credentials like database passwords. Though Parameter Store can store secrets, it is not designed as a secrets management system.

Parameter Files Weakness: Storing passwords in parameter files, even if they are environment-specific, still poses a significant risk. Parameter files can be accidentally committed to version control, exposed through misconfigured deployments, or accessed by unauthorized personnel.

Encryption in CloudFormation Template Weakness: Encrypting the password in the CloudFormation template itself provides limited security. The encryption key would also need to be managed, and if that key is compromised, the password becomes easily accessible. Storing encrypted passwords in templates doesn't offer automatic rotation or centralized management.

In summary, using Secrets Manager with dynamic references and automatic rotation provides the most secure and compliant way to manage database master passwords in a CloudFormation-based deployment. It ensures that the password is never stored in plain text, access is tightly controlled, and the password is automatically rotated to mitigate the risk of compromise.

Question: 74

A company is writing a new survey application to be used with a weekly televised game show. The application will be available for 2 hours each week. The company expects to receive over 500,000 entries every week, with each survey asking 2-3 multiple choice questions of each user. A Database Specialist needs to select a platform that is highly scalable for a large number of concurrent writes to handle the anticipated volume.
Which AWS services should the Database Specialist consider? (Choose two.)

A. Amazon DynamoDB
B. Amazon Redshift
C. Amazon Neptune
D. Amazon Elasticsearch Service
E. Amazon ElastiCache

Answer: AE

Explanation:

Here's a detailed justification for choosing Amazon DynamoDB and Amazon ElastiCache as the appropriate services for the survey application scenario:

Amazon DynamoDB: DynamoDB is a NoSQL database service designed for high scalability and performance.
Its key-value and document data model, coupled with its ability to handle millions of requests per second, makes it well-suited for high-volume write operations like receiving survey entries. DynamoDB's auto-scaling capabilities can automatically adjust the throughput capacity to handle the anticipated load of 500,000+ entries within a 2-hour timeframe. DynamoDB's built-in support for concurrent writes ensures that data integrity is maintained even with a large number of users submitting data simultaneously. The pay-per-request pricing model can be cost-effective when the application is only used for 2 hours a

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

Amazon ElastiCache: ElastiCache offers in-memory data caching services that can significantly reduce the load on the database. In this scenario, ElastiCache can be used to cache frequently accessed survey questions or intermediate results, leading to faster response times and reduced costs. This improves the application's scalability. ElastiCache can handle millions of requests per second, which is critical for the application's peak usage during the game show. Furthermore, it helps alleviate the database from handling the full load, allowing it to focus on the storing and analyzing the survey data.https://aws.amazon.com/elasticache/

Here's why the other options are less suitable:

Amazon Redshift: Redshift is primarily designed for analytical workloads and data warehousing, optimized for complex queries on large datasets. It is not ideal for handling a high volume of concurrent write operations in real-time.

Amazon Neptune: Neptune is a graph database service and is not an appropriate choice for storing simple survey responses.

Amazon Elasticsearch Service: Elasticsearch is a search and analytics engine that is not suitable for transaction processing.

Question: 75

A company has migrated a single MySQL database to Amazon Aurora. The production data is hosted in a DB cluster in VPC_PROD, and 12 testing environments are hosted in VPC_TEST using the same AWS account. Testing results in minimal changes to the test data. The Development team wants each environment refreshed nightly so each test database contains fresh production data every day.
Which migration approach will be the fastest and most cost-effective to implement?

A. Run the master in Amazon Aurora MySQL. Create 12 clones in VPC_TEST, and script the clones to be deleted and re-created nightly.

B. Run the master in Amazon Aurora MySQL. Take a nightly snapshot, and restore it into 12 databases in VPC_TEST using Aurora Serverless.

C. Run the master in Amazon Aurora MySQL. Create 12 Aurora Replicas in VPC_TEST, and script the replicas to be deleted and re-created nightly.

D. Run the master in Amazon Aurora MySQL using Aurora Serverless. Create 12 clones in VPC_TEST, and script the clones to be deleted and re-created nightly.

Answer: A

Explanation:

Here's a breakdown of why option A is the most suitable solution for refreshing the test databases, considering speed, cost-effectiveness, and the context provided:

Aurora Cloning for Speed: Aurora cloning creates a point-in-time, copy-on-write clone of the database. This is significantly faster than traditional backups and restores because it doesn't involve copying the entire dataset upfront. Only when a block is modified in the clone does it get physically copied. This speed is crucial for a nightly refresh requirement.

Cost-Effectiveness of Cloning: Aurora clones share the underlying storage with the original database. This minimizes storage costs, particularly because the test environments experience only minimal data changes. The clones only consume additional storage for the differences between the clone and the original database. **Ease of Automation:** Creating and deleting Aurora clones can be easily scripted using the AWS CLI or SDKs. This allows for the automation of the nightly refresh process.

Aurora Serverless Inefficiency: While using aurora serverless seems like it would reduce costs, it is not a

good fit for this use case. Aurora Serverless is designed for infrequent, intermittent, or unpredictable workloads, which is not the case here. The test databases are being used nightly and it will have to be spooled up every night which takes time and money. It is not a good fit for this use case.

Aurora Replicas Limitation: Aurora Replicas are designed for read-only workloads and cannot be directly written to. Therefore, they are not suitable for testing environments where data modifications are required.

Network Configuration: Aurora DB clusters can span multiple VPCs. This is important as the test environments are in a different VPC (VPC_TEST) from the production DB cluster (VPC_PROD). Ensure proper network peering or connectivity is in place between the VPCs to facilitate the creation of clones.

Option A leverages the strengths of Aurora's cloning feature to achieve fast and cost-effective refreshes of the test environments. It's also easily automatable, making it a practical solution.

Authoritative Links:

Amazon Aurora Cloning:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Managing.Clone.html
Automating Amazon RDS Tasks using AWS CLI:https://aws.amazon.com/blogs/database/automating-amazon-rds-tasks-using-aws-cli/

Question: 76

A large ecommerce company uses Amazon DynamoDB to handle the transactions on its web portal. Traffic patterns throughout the year are usually stable; however, a large event is planned. The company knows that traffic will increase by up to 10 times the normal load over the 3-day event. When sale prices are published during the event, traffic will spike rapidly. How should a Database Specialist ensure DynamoDB can handle the increased traffic?

A. Ensure the table is always provisioned to meet peak needs

B. Allow burst capacity to handle the additional load

C. Set an AWS Application Auto Scaling policy for the table to handle the increase in traffic

D. Preprovision additional capacity for the known peaks and then reduce the capacity after the event

Answer: D

Explanation:

The best approach to handle the predicted 10x traffic increase during the 3-day event is **D. Preprovision additional capacity for the known peaks and then reduce the capacity after the event.**

Here's why:

Predictability: The event is planned and the traffic increase is anticipated. This allows for proactive capacity planning.

Cost Optimization: Provisioning only for peak needs continuously (Option A) is very expensive. The high traffic is temporary, so temporarily increasing capacity makes more sense.

AWS Application Auto Scaling Effectiveness: While Option C is useful generally, DynamoDB auto scaling might not react quickly enough to the rapid spikes described in the question, especially when dealing with a 10x increase. Auto scaling policies need some time to kick in and adjust capacity based on observed metrics. If the initial spike exceeds the provisioned capacity before auto scaling can adjust, it could lead to throttling and a poor user experience.

Burst Capacity Limitations: Burst capacity (Option B) is designed for handling small, infrequent spikes. A 10x increase for three days is far beyond what burst capacity is intended to handle and is not a reliable solution.

Pre-Provisioning Advantages: By pre-provisioning the capacity, you guarantee that DynamoDB can handle the expected load immediately when the sale prices are published and traffic surges. You have full control and avoid the risk of throttling during the crucial initial phase of the event. After the event, the capacity can

be scaled down to save costs.

In essence, pre-provisioning offers the optimal balance of performance and cost-efficiency for a predictable, large, and temporary increase in traffic. Auto scaling would complement this strategy in the long term but is less suitable to rely solely upon for abrupt and significant surges.**Supporting Resources:**

DynamoDB Capacity Planning
DynamoDB Auto Scaling
Understanding DynamoDB Throughput

Question: 77

A Database Specialist is migrating an on-premises Microsoft SQL Server application database to Amazon RDS for PostgreSQL using AWS DMS. The application requires minimal downtime when the RDS DB instance goes live. What change should the Database Specialist make to enable the migration?

A. Configure the on-premises application database to act as a source for an AWS DMS full load with ongoing change data capture (CDC)

B. Configure the AWS DMS replication instance to allow both full load and ongoing change data capture (CDC)

C. Configure the AWS DMS task to generate full logs to allow for ongoing change data capture (CDC)

D. Configure the AWS DMS connections to allow two-way communication to allow for ongoing change data capture (CDC)

Answer: A

Explanation:

The correct answer is A. Here's why:

Minimal Downtime Requirement: The primary goal is to minimize downtime when switching to the RDS PostgreSQL instance. This implies using a strategy that avoids a prolonged outage during the final cutover.

AWS DMS and Change Data Capture (CDC): AWS DMS supports two primary migration methods: full load and change data capture (CDC). A full load copies all data from the source to the target. CDC continuously replicates changes from the source to the target after the initial full load. CDC is the key to minimal downtime.

Full Load + CDC: By configuring DMS to perform a full load followed by ongoing CDC, the majority of the data is initially copied over. Once the full load is complete, DMS starts replicating only the changes made to the on-premises database in real-time (or near real-time) to the RDS PostgreSQL instance.

Cutover Process: When it's time to switch over to the RDS PostgreSQL instance, the application can be briefly paused to allow DMS to replicate any remaining changes that occurred since the last CDC event. This greatly reduces the outage window compared to a strategy that requires copying all data during the cutover.

Why other options are incorrect:

B: Configuring the replication instance to allow full load and CDC doesn't address the source database configuration. The source database needs to be set up to capture changes that DMS can then replicate. The replication instance configuration is assumed to be proper if using DMS.

C: DMS task logs do not enable CDC. CDC is an independent function. Although logs can aid debugging, they are not fundamentally necessary for CDC.

D: Two-way communication is not required for this migration scenario. DMS requires a one-way flow of data from the source to the target.

Authoritative Links:

AWS DMS Documentation: Change Data Capture (CDC)
AWS DMS Documentation: Using AWS DMS for database migration

Question: 78

A financial company has allocated an Amazon RDS MariaDB DB instance with large storage capacity to accommodate migration efforts. Post-migration, the company purged unwanted data from the instance. The company now want to downsize storage to save money. The solution must have the least impact on production and near-zero downtime. Which solution would meet these requirements?

A. Create a snapshot of the old databases and restore the snapshot with the required storage

B. Create a new RDS DB instance with the required storage and move the databases from the old instances to the new instance using AWS DMS

C. Create a new database using native backup and restore

D. Create a new read replica and make it the primary by terminating the existing primary

Answer: B

Explanation:

The correct answer is **B: Create a new RDS DB instance with the required storage and move the databases from the old instances to the new instance using AWS DMS.**

Here's why:

AWS DMS (Database Migration Service) is specifically designed for migrating databases with minimal downtime. It supports homogeneous migrations (MariaDB to MariaDB) and allows for continuous data replication between the source and target databases. This ensures that the target database stays synchronized with the source during the migration process. https://aws.amazon.com/dms/

Near-Zero Downtime: With DMS, the target instance can catch up with the source instance before a final cutover. This cutover can be scheduled during a maintenance window, resulting in minimal downtime.

Storage Reduction: Creating a new instance allows you to specify the exact storage size needed. Once DMS completes replication and you've verified the new instance, you can switch traffic and decommission the old, oversized instance.

Let's analyze why the other options are not ideal:

A. Create a snapshot of the old databases and restore the snapshot with the required storage: RDS doesn't support modifying storage size during restore from snapshots for shrinking. Storage is either the same as the snapshot or larger, not smaller.

C. Create a new database using native backup and restore: While native backup and restore is a valid migration method, it involves a significant downtime window. The database needs to be taken offline for backup, and there's downtime while the restore process completes on the new instance. This isn't near-zero downtime.

D. Create a new read replica and make it the primary by terminating the existing primary: RDS read replicas inherit the storage size of the primary instance. You cannot create a read replica with a smaller storage size than the primary. Therefore, this option doesn't address the need to downsize storage.

In summary, AWS DMS provides the most effective way to migrate databases while minimizing downtime and allows you to provision a new instance with the desired storage capacity.

Question: 79

A large financial services company requires that all data be encrypted in transit. A Developer is attempting to connect to an Amazon RDS DB instance using the company VPC for the first time with credentials provided by a Database Specialist. Other members of the Development team can connect, but this user is consistently receiving an error indicating a communications link failure. The Developer asked the Database Specialist to reset the password a number of times, but the error persists. Which step should be taken to troubleshoot this issue?

A. Ensure that the database option group for the RDS DB instance allows ingress from the Developer machine's IP address

B. Ensure that the RDS DB instance's subnet group includes a public subnet to allow the Developer to connect

C. Ensure that the RDS DB instance has not reached its maximum connections limit

D. Ensure that the connection is using SSL and is addressing the port where the RDS DB instance is listening for encrypted connections

Answer: D

Explanation:

The correct answer is D. Here's a detailed justification:

The core issue is a "communications link failure" when connecting to an RDS instance with a requirement for data encryption in transit (SSL). The fact that other developers can connect suggests the problem is specific to this particular developer's connection setup, not a widespread outage or RDS configuration error.

Option D directly addresses the encryption requirement. If the developer's connection isn't configured to use SSL/TLS, or if it's attempting to connect on the non-SSL port, a communications failure is expected. RDS enforces encrypted connections when configured to do so, resulting in refused connections from clients that don't support or enable encryption. It's essential to ensure that the connection string or client configuration specifies the correct port and encryption protocol (usually TLS 1.2 or higher) for the RDS instance. The error suggests a mismatch between what the RDS expects (an encrypted connection) and what the client is providing (potentially an unencrypted one).

Option A is less likely. Database option groups configure features like native encryption or extensions, not IP-based access control. Security groups, not option groups, control network access.

Option B is incorrect because RDS instances should ideally be in private subnets for enhanced security.
Requiring a public subnet for a secure database connection contradicts security best practices.

Option C is plausible but less likely given that other team members can connect. The connection limit would affect all users, not a single user consistently. The error message explicitly indicates a link failure, pointing to a problem with the connection parameters.

Therefore, verifying SSL configuration is the most pertinent first step in troubleshooting this issue. The developer should confirm their client is configured to use SSL and is connecting to the port where RDS is listening for SSL connections (usually 3306 for MySQL, 5432 for PostgreSQL, with a specific port for the SSL connection when it is different).

Here are some helpful resources from AWS:

Using SSL/TLS to Encrypt a Connection to a DB Instance:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html
Security in Amazon RDS:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Security.html

Question: 80

A company is running Amazon RDS for MySQL for its workloads. There is downtime when AWS operating system patches are applied during the Amazon RDS- specified maintenance window.
What is the MOST cost-effective action that should be taken to avoid downtime?

A. Migrate the workloads from Amazon RDS for MySQL to Amazon DynamoDB
B. Enable cross-Region read replicas and direct read traffic to them when Amazon RDS is down
C. Enable a read replica and direct read traffic to it when Amazon RDS is down
D. Enable an Amazon RDS for MySQL Multi-AZ configuration

Answer: D

Explanation:

The correct answer is **D. Enable an Amazon RDS for MySQL Multi-AZ configuration**.

Here's why:

Multi-AZ Configuration: Amazon RDS Multi-AZ deployments provide enhanced availability and durability for database instances. When enabled, RDS automatically provisions and maintains a synchronous standby replica in a different Availability Zone. This means data is automatically replicated to the standby. During planned maintenance, like OS patching, RDS automatically fails over to the standby instance, minimizing downtime.

Cost-Effectiveness: Multi-AZ is the most cost-effective solution in this scenario. While DynamoDB (option A) provides high availability, migrating the entire workload to a NoSQL database would be a significant and costly effort, requiring application changes and potentially performance tuning. Cross-Region read replicas (option B) introduce more complexity and cost than Multi-AZ because of the need to manage cross-region replication and potential latency issues for write operations. A single read replica (option C) won't eliminate downtime during maintenance, as a failover is not automatic, and the read replica would experience the maintenance event after the primary instance.

Maintaining Consistency: Synchronous replication in Multi-AZ ensures that the standby instance is always up-to-date, preventing data loss during failover. This is crucial for maintaining data consistency and integrity, which is paramount for many database workloads.

RDS Managed Failover: Amazon RDS handles the failover process automatically, reducing the operational overhead and simplifying database administration.

In summary, enabling Multi-AZ configuration is the most straightforward, cost-effective, and minimally disruptive approach to avoid downtime during RDS maintenance windows. It leverages RDS's built-in high availability features without requiring significant application changes or increased operational complexity.

Supporting Links:

Amazon RDS Multi-AZ: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html Amazon RDS Maintenance:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_UpgradeDBInstance.Maintenance.html