Solutions Architect Professional Questions 3

Question 1

A company requires a disaster recovery (DR) solution for its e-commerce website, which is hosted on a fleet of t3.large Amazon EC2 instances and uses an Amazon RDS for MySQL DB instance. The EC2 instances are part of an Auto Scaling group spread across multiple Availability Zones. In a disaster scenario, the company needs the web application to switch to a secondary environment with a Recovery Point Objective (RPO) of 30 seconds and a Recovery Time Objective (RTO) of 10 minutes. The company seeks the most cost-effective solution to achieve these DR objectives.

The proposed solutions for setting up this DR strategy are:

A. Implement infrastructure as code (IaC) for new infrastructure in a DR Region. Create a cross-Region read replica for the RDS DB instance. Use AWS Backup for cross-Region backups of EC2 and DB instances, with a cron job for backups every 30 seconds. Recover EC2 instances from the latest backups. Employ Amazon Route 53 geolocation routing for automatic failover.

B. Use IaC to establish new infrastructure in the DR Region. Create a cross-Region read replica for the RDS DB instance. Set up AWS Elastic Disaster Recovery for continuous replication of EC2 instances to the DR Region. Maintain minimal EC2 instance capacity in the DR Region. Utilize Amazon Route 53 failover routing for automatic failover, increasing Auto Scaling group capacity as needed.

C. Arrange for cross-Region backups of EC2 and DB instances via AWS Backup with a cron job for 30-second backups. Use IaC for new infrastructure in the DR Region. Manually restore backup data on new instances. Implement Amazon Route 53 simple routing for automatic failover.

D. Deploy new infrastructure in the DR Region using IaC. Create an Amazon Aurora global database. Configure AWS Elastic Disaster Recovery for continuous EC2 instance replication to the DR Region. Keep the Auto Scaling group of EC2 instances at full capacity in the DR Region. Use Amazon Route 53 failover routing for automatic failover.

The most suitable solution is B:

- B. IaC, Cross-Region Read Replica, AWS Elastic Disaster Recovery, Minimal EC2 Capacity, Route 53 Failover Routing: This option efficiently meets the RPO and RTO requirements. Using IaC ensures quick and consistent setup in the DR Region. The cross-

Region read replica addresses the RPO for the database. AWS Elastic Disaster Recovery enables continuous replication of the EC2 instances, aligning with the RPO for the application. Running EC2 instances at minimal capacity in the DR Region is cost-effective while allowing for rapid scale-up (meeting the RTO) during a failover event. Route 53 failover routing provides an automated mechanism to switch to the DR environment.

The reasons why the other options are less suitable:

A. **AWS Backup with 30-Second Cron Job, Geolocation Routing**: Taking backups every 30 seconds is impractical and may not be supported by AWS Backup, leading to potential challenges in meeting the RPO. Geolocation routing isn't ideal for DR failover scenarios.

C. **Manual Restore with Simple Routing**: The manual restoration process does not align with the 10-minute RTO requirement. Simple routing in Route 53 doesn't provide automatic failover capabilities.

D. **Full Capacity EC2 Instances, Aurora Global Database**: Running full-capacity EC2 instances in the DR Region can be costly and may not be necessary for maintaining a standby DR site. Also, using an Aurora global database for a MySQL RDS instance adds complexity and may not be directly compatible.

Question 2

A company is planning a one-time migration of its large, 60 TB on-premises MySQL database to Amazon Aurora MySQL, located in the AWS us-east-1 Region. However, the company faces a challenge due to its limited internet bandwidth, estimating that transferring this volume of data over their existing internet connection could take about a month. To expedite the migration process, the company is seeking a faster solution.

The proposed solutions for quickly migrating the database are:

A. Establish a 1 Gbps AWS Direct Connect link between the on-premises data center and AWS, and then use AWS Database Migration Service (AWS DMS) to migrate the MySQL database to Aurora MySQL.

B. Utilize AWS DataSync over the current internet connection to speed up data transfer from the on-premises data center to AWS, and employ AWS Application Migration Service to migrate the MySQL database to Aurora MySQL.

C. Order an AWS Snowball Edge device, transfer the database data to an Amazon S3 bucket using the Snowball Edge's S3 interface, and then use AWS DMS to migrate the data from Amazon S3 to Aurora MySQL.

D. Request an AWS Snowball device, upload the database data to an Amazon S3 bucket using the S3 Adapter for Snowball, and then utilize AWS Application Migration Service to migrate the data from Amazon S3 to Aurora MySQL.

The most effective solution is C:

- C. AWS Snowball Edge, S3, and AWS DMS: Given the size of the database (60 TB) and limited internet bandwidth, using an AWS Snowball Edge device is the most practical and fastest approach. Snowball Edge provides a physical device for data transfer, bypassing bandwidth limitations. After transferring the data to S3 using the device, AWS DMS can efficiently handle the migration from S3 to Aurora MySQL, making the overall process quicker than over-the-internet transfer methods.

The reasons why the other options are less suitable:

A. 1 Gbps AWS Direct Connect and AWS DMS: While Direct Connect provides a dedicated network link, its capacity (1 Gbps in this case) might still result in a lengthy data transfer process for a 60 TB database, not significantly improving on the original one-month estimate.

B. AWS DataSync and Application Migration Service: DataSync can accelerate data transfer, but it's still limited by the company's internet bandwidth. For a database of this size, this approach may not offer a substantial time advantage.

D. AWS Snowball and Application Migration Service: Regular AWS Snowball devices are efficient for large data transfers but are not as versatile as Snowball Edge devices, especially when it comes to direct integration with AWS services like S3. Moreover, the Application Migration Service is more focused on migrating applications rather than large-scale database migrations.

Thus, option C is the most time-efficient method, leveraging Snowball Edge for high-volume data transfer and DMS for effective migration to Aurora MySQL.

Question 3

A company operates an application on AWS, running on 20 Amazon EC2 instances with multiple Amazon Elastic Block Store (EBS) volumes attached. The requirement is to maintain backups in a separate AWS Region, ensuring recovery of both the EC2 instances and their configurations within one business day, with a maximum data loss window of one day. Additionally, the company seeks a backup solution that maximizes operational efficiency and cost-effectiveness, given their limited staffing resources. They already have an AWS CloudFormation template for deploying the necessary network setup in a secondary Region.

The solutions under consideration to meet these backup and recovery requirements are:

A. Develop a second CloudFormation template for recreating EC2 instances in the secondary Region. Use AWS Systems Manager Automation runbooks for daily multivolume EBS snapshots, copying them to the secondary Region. In case of failure, use the CloudFormation templates to restore the EBS volumes from snapshots and switch operations to the secondary Region.

B. Implement Amazon Data Lifecycle Manager (Amazon DLM) to perform daily multivolume EBS snapshots. In a disaster scenario, use the existing CloudFormation template to rebuild the environment in the secondary Region and restore EBS volumes using Amazon DLM.

C. Utilize AWS Backup for a daily scheduled backup plan of the EC2 instances, configuring the backups to be copied to a backup vault in the secondary Region. In case of failure, deploy the CloudFormation template, restore instance volumes and configurations from the backup vault, and shift operations to the secondary Region.

D. Deploy EC2 instances with the same configuration in the secondary Region and employ AWS DataSync to copy data daily from the primary to the secondary Region. In the event of a failure, execute the CloudFormation template and move operations to the secondary Region.

The best solution is C:

- C. AWS Backup with Daily Scheduled Backup and Cross-Region Backup Vault: AWS Backup provides a comprehensive and centralized solution for backing up AWS resources, including EC2 instances and attached EBS volumes. It supports automated daily backups and copying these backups to a vault in a different AWS Region, meeting the recovery objectives with minimal data loss. In case of a disaster, the company can quickly restore the EC2 instances and their configurations from the backup vault in the secondary Region using the CloudFormation template. This approach is operationally efficient, cost-effective, and requires minimal manual intervention.

The reasons why the other options are less suitable:

A. CloudFormation with Systems Manager for Snapshots: While this method can work, manually managing daily snapshots with Systems Manager Automation and a separate CloudFormation template for EC2 instances adds complexity and operational overhead.

B. Amazon DLM for EBS Snapshots and CloudFormation: Amazon DLM efficiently automates the EBS snapshot process. However, it does not cover the full backup of EC2 instances, including their configurations, making it less comprehensive than AWS Backup.

D. Secondary EC2 Instances and AWS DataSync: This approach of maintaining parallel instances and using DataSync for daily data transfer is operationally complex and potentially

more costly. It also does not fully address the backup of instance configurations and requires significant manual management.

Therefore, option C, using AWS Backup, is the most suitable choice for meeting the company's requirements for disaster recovery, operational efficiency, and cost-effectiveness.

Question 4

A company is developing a new website for hosting static content, featuring functionality for users to upload and download large files. The company's security standards mandate that all data must be encrypted both in transit and at rest. The solutions architect is tasked with designing this system using Amazon S3 and Amazon CloudFront, ensuring it adheres to the encryption requirements.

The potential steps to ensure compliance with these encryption requirements are (choose three):

A. Activate server-side encryption (SSE) for the Amazon S3 bucket utilized by the web application.

B. Implement a policy attribute of "aws:SecureTransport": "true" for read and write operations in the S3 Access Control Lists (ACLs).

C. Establish a bucket policy for the S3 bucket used by the web application, denying any operations that are not encrypted.

D. Set up encryption at rest on CloudFront using server-side encryption with AWS Key Management Service (KMS) keys (SSE-KMS).

E. Configure CloudFront to redirect all HTTP requests to HTTPS, ensuring encrypted data transfer.

F. Implement the RequireSSL option in the generation of presigned URLs for the S3 bucket used by the web application.

The most suitable combination of steps is A, C, and E:

- A. S3 Server-Side Encryption: Enabling server-side encryption for the S3 bucket ensures that all data stored (at rest) in the bucket is automatically encrypted, meeting the at-rest encryption requirement.
- C. Bucket Policy to Deny Unencrypted Operations: Creating a bucket policy that denies any unencrypted read or write operations to the S3 bucket enforces the encryption-at-rest policy by ensuring that all interactions with the bucket are encrypted.

- E. Redirect HTTP to HTTPS in CloudFront: Configuring CloudFront to redirect HTTP requests to HTTPS ensures that all data in transit between CloudFront and the end-users is encrypted, fulfilling the in-transit encryption requirement.

The reasons why the other options are less suitable:

B. "aws:SecureTransport" in S3 ACLs: While this can enforce HTTPS for access to the S3 bucket, it's generally more effective and easier to manage this at the CloudFront level. Additionally, S3 ACLs are a legacy access control method and are less recommended compared to bucket policies.

D. Encryption at Rest on CloudFront with SSE-KMS: CloudFront does not store data (it's a content distribution network), so configuring encryption at rest on CloudFront is not applicable. The encryption of data at rest is more relevant to the S3 bucket.

F. RequireSSL in Presigned URLs for S3: While using RequireSSL in presigned URLs can enforce HTTPS for specific object access, it doesn't broadly enforce encryption in transit for all access to the bucket. The redirection of HTTP to HTTPS in CloudFront (Option E) is a more comprehensive approach.

Question 5

A company is adopting a serverless architecture using AWS Lambda functions, which necessitate access to a Microsoft SQL Server DB instance hosted on Amazon RDS. There are distinct development and production environments, each with its own database clone. While developers can access the development database credentials, the production database credentials need heightened security. Specifically, these credentials should be encrypted with a key accessible only to the IT security team's IAM user group and regularly rotated.

The solutions architect must ensure this secure access in the production environment. The proposed methods are:

A. Utilize AWS Systems Manager Parameter Store to store the database credentials as a SecureString parameter, encrypted with an AWS Key Management Service (KMS) customer-managed key. Lambda functions would be assigned roles to access this parameter. Access to both the SecureString parameter and the KMS key would be limited to the IT security team.

B. Encrypt the database credentials using the AWS KMS default Lambda key, then store them in the Lambda functions' environment variables. The Lambda code would retrieve credentials from these variables. Access to the KMS key would be restricted to the IT security team.

C. Place the database credentials in the Lambda functions' environment variables and encrypt these variables using an AWS KMS customer-managed key. Access to this key would be restricted to the IT security team.

D. Store the database credentials in AWS Secrets Manager as a secret linked to an AWS KMS customer-managed key. Assign roles to Lambda functions for secret access. Restrict access to both the secret and the KMS key to only the IT security team.

The most appropriate solution is D:

- D. AWS Secrets Manager with KMS Customer-Managed Key: This approach uses AWS Secrets Manager to securely store the database credentials as a secret. The secret is associated with a KMS customer-managed key, ensuring robust encryption. Lambda functions would be given roles to access this secret. Importantly, access to both the secret and the KMS key would be restricted to the IT security team. This setup not only secures the credentials but also facilitates automatic rotation of the credentials, aligning with the company's requirements.

The reasons why the other options are less suitable:

A. Systems Manager Parameter Store with SecureString: While this method securely stores and encrypts credentials, it doesn't inherently offer the same level of management and rotation capabilities for secrets as AWS Secrets Manager.

B. KMS Default Lambda Key with Environment Variables: Storing credentials in environment variables, even when encrypted, is generally less secure and lacks the management and rotation features offered by AWS Secrets Manager.

C. Environment Variables Encrypted with KMS Customer-Managed Key: Like option B, using environment variables for storing credentials, even with encryption, is less secure and efficient compared to using AWS Secrets Manager, which is purpose-built for handling secrets.

Question 6

An online retail company is transitioning its existing on-premises .NET application, which includes a web server tier, an application server tier, and a Microsoft SQL Server database, to AWS. The primary objectives are to address scalability issues, minimize licensing costs, and leverage AWS managed services without rewriting the application.

The proposed solutions for this migration are:

A. Utilize Amazon EC2 instances in Auto Scaling groups behind an Application Load Balancer for both the web and application tiers. Migrate the SQL Server database to Amazon Aurora PostgreSQL, enabling Babelfish to facilitate compatibility with SQL Server.

B. Migrate server images to AWS using AWS Database Migration Service (AWS DMS), then deploy these as Amazon EC2 instances. Set these instances in Auto Scaling groups behind a

Network Load Balancer for both the web and application tiers. Replace the SQL Server database with Amazon DynamoDB.

C. Containerize the web and application tiers, deploying them on an Amazon Elastic Kubernetes Service (Amazon EKS) cluster, with Auto Scaling groups behind a Network Load Balancer. Use Amazon RDS for SQL Server to host the database.

D. Refactor the application into AWS Lambda functions, utilizing Amazon API Gateway for the web and application tiers. Migrate the data from SQL Server to Amazon S3, using Amazon Athena for data querying.

The most suitable solution is A:

- A. EC2 Auto Scaling Groups, Application Load Balancer, Aurora PostgreSQL with Babelfish: This approach addresses scalability by using EC2 Auto Scaling groups for the web and application tiers, ensuring cost-effective scaling in response to demand. The Application Load Balancer offers efficient traffic distribution. Migrating to Aurora PostgreSQL with Babelfish provides SQL Server compatibility, minimizing the need for database code changes and reducing SQL Server licensing costs, making it a cost-effective solution.

The reasons why the other options are less suitable:

B. EC2 Instances with Network Load Balancer, DynamoDB: While this setup uses Auto Scaling for the compute layer, migrating a SQL Server database to DynamoDB (a NoSQL database) would require significant application rewrites, which the company wants to avoid.

C. Containerization with EKS, RDS for SQL Server: Containerizing the application and deploying on EKS could be more complex and might not offer significant cost benefits over EC2 Auto Scaling. Using RDS for SQL Server does not reduce SQL Server licensing costs.

D. Lambda Functions, API Gateway, Data on S3 with Athena: Refactoring the entire application into Lambda functions and migrating the SQL Server database to S3 for querying with Athena would require extensive application rewriting, which is contrary to the company's requirements.

Question 7

A SaaS provider is offering APIs through an Application Load Balancer (ALB) linked to an Amazon Elastic Kubernetes Service (Amazon EKS) cluster in the us-east-1 Region. These APIs employ some unique REST methods like LINK, UNLINK, LOCK, and UNLOCK. However, users located outside the United States are experiencing delayed and inconsistent response times when accessing these APIs. The provider needs a solution that effectively reduces these latency issues while keeping operational complexity to a minimum.

The potential solutions to address this challenge are:

A. Implement an Amazon CloudFront distribution, setting the ALB as its origin.

B. Establish an Amazon API Gateway edge-optimized API endpoint to serve the APIs, with the ALB designated as the target.

C. Create an AWS Global Accelerator and configure it to use the ALB as the origin.

D. Expand the API deployment across two additional AWS Regions (eu-west-1 and ap-southeast-2) and implement latency-based routing using Amazon Route 53.

The most appropriate solution is C:

- C. AWS Global Accelerator with ALB as Origin: AWS Global Accelerator optimizes the routing of user traffic through AWS's extensive global network infrastructure, enhancing the performance of applications for a global user base. By directing traffic to the ALB in the us-east-1 Region, Global Accelerator can significantly reduce latency and provide more consistent response times for users worldwide. This approach is operationally simpler than deploying the application in multiple regions and efficiently handles non-standard REST methods.

The reasons why the other options are less suitable:

A. Amazon CloudFront with ALB as Origin: CloudFront is primarily designed for caching static and dynamic content. While it can accelerate API responses, it might not be fully optimized for non-standard REST methods. CloudFront primarily excels with standard HTTP methods.

B. Amazon API Gateway Edge-Optimized with ALB Target: While API Gateway can improve performance for APIs, edge-optimized endpoints are more tailored for standard RESTful APIs. Handling non-standard methods might require additional configurations, and it may not address global latency issues as effectively as Global Accelerator.

D. Deployment in Additional Regions with Route 53 Latency-Based Routing: Expanding the deployment across multiple regions would increase operational complexity and overhead. It involves managing the application in multiple regions, which contradicts the requirement for minimal operational overhead.

Question 8

A company operating an IoT application on AWS is facing challenges with its current setup, where millions of sensors across the United States collect data and send it to a custom MQTT broker hosted on a single Amazon EC2 instance. This broker, which is the endpoint for sensor

data collection, is identified by the domain iot.example.com, managed via Amazon Route 53. The data is eventually stored in Amazon DynamoDB. However, the broker has been overwhelmed by the data volume, leading to data loss. The company is seeking a solution to enhance the system's reliability and prevent data loss.

The potential solutions for improving the system's reliability are:

A. Implement an Application Load Balancer (ALB) and an Auto Scaling group for the MQTT broker. Set this Auto Scaling group as the ALB's target. Modify the Route 53 DNS record to an alias pointing to the ALB. Continue using the MQTT broker for data storage.

B. Transition to using AWS IoT Core for sensor data reception. Establish and configure a custom domain to connect to AWS IoT Core. Update the Route 53 DNS record to redirect to the AWS IoT Core Data-ATS endpoint. Implement an AWS IoT rule to direct the data into DynamoDB.

C. Deploy a Network Load Balancer (NLB) with the MQTT broker as its target. Set up an AWS Global Accelerator, assigning the NLB as its endpoint. Change the Route 53 DNS record to a multivalue answer record, including the Global Accelerator's IP addresses. Retain the MQTT broker for data storage.

D. Utilize AWS IoT Greengrass for receiving sensor data. Update the Route 53 DNS record to point to the AWS IoT Greengrass endpoint. Configure an AWS IoT rule to trigger an AWS Lambda function for data storage.

The most suitable solution is B:

- **B. AWS IoT Core with Custom Domain and Route 53 Update**: AWS IoT Core is specifically designed to handle large-scale IoT deployments and can efficiently manage millions of device connections, addressing the scalability and reliability issues faced by the current MQTT broker. By directing the sensor data to AWS IoT Core and updating the DNS record in Route 53 to point to the IoT Core Data-ATS endpoint, the solution leverages AWS's managed service capabilities for IoT. An AWS IoT rule can be set up to store the data in DynamoDB, thereby ensuring reliable and scalable data handling.

The reasons why the other options are less suitable:

A. **ALB with Auto Scaling for MQTT Broker**: While this approach introduces scalability, it still relies on a custom MQTT broker, which may not be as reliable or scalable as a managed service like AWS IoT Core. Managing and scaling a custom broker adds operational complexity.

C. **NLB and Global Accelerator for MQTT Broker**: Similar to option A, this method still depends on the custom MQTT broker. While NLB and Global Accelerator provide scalability

and improved global performance, they don't address the fundamental reliability issue inherent in using a single custom broker.

D. AWS IoT Greengrass with Route 53 Update: IoT Greengrass is primarily used for local compute, messaging, data caching, and sync capabilities for connected devices. While it provides edge computing solutions, it is not the optimal choice for centralized data ingestion and management as required in this scenario.

Question 9

A company is operating Linux-based Amazon EC2 instances, each requiring its own unique EC2 SSH key pair for user access. To enhance security, the company plans to implement a policy for automatically rotating these EC2 key pairs, ensuring minimal downtime (less than 1 minute) during the rotation process. Additionally, the company emphasizes securely storing these keys. The company is exploring solutions that can automate key rotation and secure key storage.

The proposed solutions for automating the key rotation and securely storing the keys are:

A. Utilize AWS Secrets Manager to store all the key pairs. Set up a rotation schedule in Secrets Manager to trigger an AWS Lambda function that generates new key pairs, updates the public keys on the EC2 instances, and then stores the new private keys in Secrets Manager.

B. Keep all the keys in AWS Systems Manager Parameter Store as strings. Configure a Systems Manager maintenance window to activate an AWS Lambda function for generating new key pairs, updating public keys on EC2 instances, and storing the new private keys in Parameter Store.

C. Import the EC2 key pairs into AWS Key Management Service (AWS KMS) and enable automatic key rotation for these keys. Use an Amazon EventBridge scheduled rule to trigger a Lambda function that initiates key rotation in AWS KMS.

D. Add all EC2 instances to Fleet Manager, a feature of AWS Systems Manager. Define a Systems Manager maintenance window to execute a Systems Manager Run Command document, which generates new key pairs and rotates the public keys on all instances managed by Fleet Manager.

The most appropriate solution is A:

- A. AWS Secrets Manager for Key Storage and Rotation: Secrets Manager is designed to manage, retrieve, and rotate secrets such as database credentials and SSH keys securely. By storing SSH key pairs in Secrets Manager and defining a rotation schedule that triggers a Lambda function, the company can automate the generation of new key

pairs and the replacement of public keys on the EC2 instances. This solution ensures that the private keys are securely stored and rotated in compliance with the company's policy, and it can be achieved with minimal downtime.

The reasons why the other options are less suitable:

B. Parameter Store for Key Storage and Lambda for Rotation: While Parameter Store can securely store configuration data and secrets, it does not have built-in capabilities for rotating SSH key pairs like Secrets Manager. The process would require more custom implementation compared to Secrets Manager.

C. AWS KMS for Key Rotation: AWS KMS is primarily used for creating and managing cryptographic keys and is not suitable for managing SSH key pairs for EC2 instances. KMS does not directly support the rotation of SSH keys as described in the requirement.

D. Fleet Manager and Run Command for Key Generation and Rotation: Fleet Manager within AWS Systems Manager provides capabilities for managing EC2 instances, but it does not inherently provide a straightforward method for rotating SSH key pairs across multiple instances.

Question 10

A company planning to migrate to AWS is currently operating thousands of VMs within a VMware ESXi environment. However, they lack a configuration management database and have limited insight into the utilization of their VMware portfolio. To facilitate a cost-effective migration, the company requires a comprehensive inventory analysis with minimal operational complexity.

The potential solutions for gathering an accurate inventory and planning the migration are:

A. Implement AWS Systems Manager Patch Manager to deploy Migration Evaluator on each VM. Analyze the data using Amazon QuickSight, focusing on identifying highly utilized servers. Exclude these high-utilization servers from the migration plan and integrate the findings with AWS Migration Hub.

B. Manually export the VMware portfolio details into a .csv file, examining disk utilization for each server. Remove servers with high utilization from the migration consideration and use this data with AWS Application Migration Service. Employ AWS Server Migration Service (AWS SMS) for migrating the remaining servers.

C. Utilize the Migration Evaluator's agentless collector on the ESXi hypervisor to gather data. Review this data within Migration Evaluator, pinpointing inactive servers. Exclude these inactive servers from the migration list and import the refined data into AWS Migration Hub for further migration planning.

D. Install the AWS Application Migration Service Agent on every VM to collect necessary data. Once data collection is complete, use Amazon Redshift to import and analyze this data, employing Amazon QuickSight for visualizing the findings.

The most suitable solution is C:

- C. Migration Evaluator Agentless Collector: The Migration Evaluator's agentless collector is specifically designed for environments like VMware ESXi, enabling comprehensive data collection with minimal intrusion and operational complexity. By deploying it at the hypervisor level, it can efficiently gather detailed utilization and configuration data across all VMs. The analysis within Migration Evaluator helps identify inactive servers, which can be excluded to optimize the migration plan. The data can then be imported to AWS Migration Hub for streamlined migration management.

The reasons why the other options are less suitable:

A. AWS Systems Manager with Patch Manager and Migration Evaluator: While Systems Manager and Patch Manager are powerful AWS tools, deploying Migration Evaluator across each VM individually would be operationally intensive and less efficient compared to an agentless approach.

B. Manual Export to .csv and AWS SMS: Manually exporting data to a .csv file is labor-intensive and prone to errors. This method lacks the depth of automated data collection tools and may not provide a comprehensive view of the environment, leading to less informed migration decisions.

D. AWS Application Migration Service Agent and Amazon Redshift: Deploying individual agents on each VM and using Amazon Redshift for analysis introduces unnecessary complexity. This approach requires significant effort in data handling and analysis, which is not ideal for a company seeking a solution with the least operational overhead.

Question 11

A company operates a microservice using an AWS Lambda function, which is designed to write data to an on-premises SQL database. This database, however, can only handle a limited number of concurrent connections. High volumes of Lambda invocations have led to database crashes and subsequent application downtime. The company's infrastructure includes an AWS Direct Connect linking their VPC to the on-premises data center. The primary goal is to safeguard the database against crashes caused by excessive concurrent connections.

The proposed solutions to address this issue are:

A. Implement Amazon Simple Queue Service (Amazon SQS) to queue the data. Adjust the Lambda function to process data from this queue and write to the current database. Set a reserved concurrency limit on the Lambda function, ensuring it remains below the database's maximum connection capacity.

B. Transition to an Amazon Aurora Serverless DB cluster. Utilize AWS DataSync to transfer data from the existing on-premises database to Aurora Serverless. Update the Lambda function to write data to the new Aurora database.

C. Set up an Amazon RDS Proxy DB instance and link it to the Amazon RDS DB instance. Modify the Lambda function to direct writes to the RDS Proxy instead of directly to the database.

D. Route the data to an Amazon Simple Notification Service (Amazon SNS) topic. Configure the Lambda function to trigger on new messages in the SNS topic, writing to the existing database. Establish provisioned concurrency for the Lambda function to match the database's connection limit.

The most appropriate solution is A:

- A. Amazon SQS Queue and Reserved Lambda Concurrency: This approach effectively decouples the rate of Lambda invocations from direct database writes by using SQS as a buffer. The Lambda function reads from the queue at a controlled rate, dictated by the reserved concurrency setting, which is configured to be lower than the database's maximum connection capacity. This setup prevents the database from being overwhelmed by too many concurrent connections, thereby reducing the risk of crashes.

The reasons why the other options are less suitable:

B. Migrate to Aurora Serverless with DataSync: While Aurora Serverless could potentially handle varying loads, migrating the existing on-premises SQL database to a new Aurora Serverless database involves significant changes in the database infrastructure and potential compatibility issues. It also does not address the immediate issue of managing Lambda function concurrency.

C. Use RDS Proxy with RDS DB Instance: RDS Proxy is designed to work with Amazon RDS and Aurora databases. It is not applicable for on-premises SQL databases, making this option irrelevant for the current setup.

D. Amazon SNS with Provisioned Lambda Concurrency: Using SNS and configuring provisioned concurrency equal to the database's connection limit does not effectively manage the burst of Lambda invocations. It also lacks the queuing mechanism that SQS provides, which is essential for smoothing out bursts in demand.

Question 12

A company currently utilizes a Grafana data visualization tool hosted on a single Amazon EC2 instance to monitor its AWS workloads. They have developed custom dashboards within Grafana that are crucial for their operations and seek to maintain these dashboards. The requirement is for these dashboards to be highly available, with a maximum allowable downtime of 10 minutes. The company aims to achieve this with minimal ongoing maintenance.

The proposed solutions to ensure high availability and ease of maintenance for the Grafana dashboards are:

A. Transition to Amazon CloudWatch dashboards, replicating the existing Grafana dashboards as closely as possible, and utilizing automatic dashboards where feasible.

B. Migrate to Amazon Managed Grafana by setting up a workspace, configuring a new Amazon CloudWatch data source, exporting the current Grafana dashboards, and importing them into the Managed Grafana workspace.

C. Create an Amazon Machine Image (AMI) with Grafana pre-installed, store the existing dashboards on Amazon Elastic File System (Amazon EFS), and set up an Auto Scaling group using this AMI with a configuration of one instance. Pair this with an Application Load Balancer covering at least two Availability Zones.

D. Implement AWS Backup to regularly back up the EC2 instance running Grafana (hourly backups). Restore the EC2 instance from the latest backup to an alternate Availability Zone if needed.

The most suitable solution is B:

- B. Amazon Managed Grafana Workspace: This option involves migrating to Amazon Managed Grafana, a service designed to offer Grafana as a scalable, secure, and managed solution. By exporting the existing dashboards from the current Grafana instance and importing them into a Managed Grafana workspace, the company can maintain its custom dashboards with high availability. Managed Grafana reduces the need for manual maintenance and infrastructure management, aligning with the goal of minimal operational overhead.

The reasons why the other options are less suitable:

A. Transition to Amazon CloudWatch Dashboards: While CloudWatch dashboards are highly available and require minimal maintenance, recreating the Grafana dashboards in CloudWatch could be time-consuming and might not offer the same functionality or visual fidelity as Grafana.

C. AMI with Grafana and Auto Scaling Group: Creating a custom AMI and using Auto Scaling with an Application Load Balancer adds complexity and requires ongoing maintenance of the AMI and the infrastructure. This approach doesn't fully leverage managed services to minimize operational overhead.

D. AWS Backup for EC2 Instance: Regularly backing up the EC2 instance and restoring it to another Availability Zone in case of failure can ensure some level of availability. However, this approach requires manual intervention for restoration and does not guarantee the 10-minute downtime limit.

Question 13

A company is planning to move its customer transactions database, currently hosted on an Oracle DB instance on a Linux server on-premises, to AWS. A new security policy mandates that the database password must be rotated annually. The goal is to find a migration and password management solution that minimizes operational effort.

The suggested solutions to meet these requirements are:

A. Transition the database to Amazon DynamoDB using the AWS Schema Conversion Tool (AWS SCT), store the database password in AWS Systems Manager Parameter Store, and set up an Amazon CloudWatch alarm to trigger an AWS Lambda function for annual password rotation.

B. Migrate the database to Amazon RDS for Oracle, use AWS Secrets Manager to manage the database password, and enable automatic rotation with a yearly schedule.

C. Move the database to an Amazon EC2 instance, utilize AWS Systems Manager Parameter Store for storing and rotating the connection string via an AWS Lambda function on an annual schedule.

D. Migrate the database to Amazon Neptune using AWS SCT, and implement an Amazon CloudWatch alarm to activate an AWS Lambda function for password rotation every year.

The most appropriate solution is B:

- B. Amazon RDS for Oracle with AWS Secrets Manager: Migrating the database to Amazon RDS for Oracle provides a managed database solution, reducing the operational burden associated with database administration. AWS Secrets Manager is specifically designed for managing and automatically rotating secrets like database passwords. It can be configured to rotate the password annually, fulfilling the security requirement with minimal operational overhead. RDS and Secrets Manager integration streamlines this process.

The reasons why the other options are less suitable:

A. DynamoDB with Parameter Store and Lambda: Converting an Oracle database to DynamoDB, a NoSQL database, could involve significant changes to the database schema and application code. Additionally, using Parameter Store and a custom Lambda function for password rotation adds complexity compared to the automated rotation capabilities of Secrets Manager.

C. EC2 Instance with Parameter Store and Lambda: Hosting the database on an EC2 instance requires more manual management compared to using RDS. Additionally, using Parameter Store and a Lambda function for password rotation is less streamlined than the automated process offered by Secrets Manager.

D. Neptune with SCT and Lambda: Migrating an Oracle database to Amazon Neptune, a graph database, likely requires significant changes to the database schema and application logic. This option, like option A, also involves more complexity in password rotation management.

Question 14

A company with multiple teams operating in a single AWS Region requires an efficient AWS account and network design. The key requirement is to establish a Virtual Private Cloud (VPC) for these teams, ensuring connectivity to their existing on-premises network. The anticipated data transfer rate to and from this on-premises network is below 50 Mbps. The goal is to achieve this setup in the most cost-effective manner, addressing both the need for a VPC and the connectivity to the on-premises network.

The potential solutions to meet these criteria are:

A. Individual Deployment of CloudFormation Template: Develop a CloudFormation template to set up a VPC and necessary subnets, and deploy it within each AWS account separately. This method ensures every team has its dedicated VPC, but it could lead to duplicated efforts and increased complexity in managing multiple VPCs.

B. Centralized CloudFormation Template with Subnet Sharing: Create a single CloudFormation template to establish a VPC and subnets in a central shared services account. Utilize AWS Resource Access Manager to share these subnets with different team accounts. This approach centralizes network management and reduces redundancy.

C. AWS Transit Gateway with Site-to-Site VPN: Implement AWS Transit Gateway in combination with an AWS Site-to-Site VPN. Share the Transit Gateway across accounts using AWS Resource Access Manager. While Transit Gateway provides a scalable and efficient

network architecture, it may be more complex and costly than necessary for the stated requirements.

D. AWS Site-to-Site VPN: Deploy an AWS Site-to-Site VPN to connect the VPC to the on-premises network. This solution is straightforward and cost-effective, especially suitable for the expected traffic volume of less than 50 Mbps.

E. AWS Direct Connect: Implement AWS Direct Connect for the on-premises to AWS connection. Although Direct Connect offers a consistent and low-latency link, it might be excessive for the expected low traffic volume and more costly compared to a Site-to-Site VPN.

Considering the need for cost-effectiveness and simplicity in the setup, the most suitable solutions are: B, and E

- B. Centralized CloudFormation Template with Subnet Sharing: This method promotes efficient resource use and easier network management. By having a central VPC and sharing its subnets, the company can avoid unnecessary duplication and simplify the network setup for all teams.
- D. AWS Site-to-Site VPN: Given the low expected traffic volume, a Site-to-Site VPN is an appropriate choice. It provides the necessary connectivity to the on-premises network without incurring the higher costs and complexities of solutions like AWS Direct Connect.

The reasons why the other options are less suitable:

A. Individual Deployment of CloudFormation Template in Each Account: Deploying a separate VPC and subnets in each team's AWS account through individual CloudFormation templates can lead to increased administrative overhead. Managing multiple VPCs across several accounts can be complex and inefficient, especially when all teams operate within the same AWS Region. This approach lacks the centralization that could simplify network management and reduce redundancy.

C. AWS Transit Gateway with Site-to-Site VPN: Using AWS Transit Gateway in conjunction with a Site-to-Site VPN offers a robust solution for managing network traffic across multiple VPCs and the on-premises network. However, for a company that expects less than 50 Mbps of traffic and is looking for the most cost-effective solution, Transit Gateway might be an overkill. It's designed for more complex networking requirements and higher traffic volumes, and it may introduce unnecessary costs and complexity for this scenario.

E. AWS Direct Connect: While AWS Direct Connect provides a dedicated and reliable connection between on-premises networks and AWS, it's generally more suitable for scenarios requiring consistent high throughput and low latency. For the company's expected traffic of less than 50 Mbps, Direct Connect may not be the most cost-effective choice due to

its higher setup and ongoing costs compared to a Site-to-Site VPN. It's typically more beneficial for larger-scale operations with greater bandwidth requirements.

Question 15

In a large organization with over 100 AWS accounts connected via a centralized AWS Transit Gateway, there's a need to implement network security for all outbound internet traffic. The organization, operating solely in a single AWS Region, requires centrally managed, rule-based filtering of this traffic. Additionally, the peak outbound traffic across each Availability Zone is expected not to exceed 25 Gbps.

The task is to determine the most effective solution to achieve this centralized filtering for all accounts in the organization. The options considered are:

A. Set Up a Central VPC with a Custom Proxy: This involves creating a new VPC dedicated to handling outbound internet traffic. The Transit Gateway would connect to this VPC, which would include a NAT gateway and an Auto Scaling group of EC2 instances running an open-source internet proxy for filtering. All default routes in the accounts would be modified to direct traffic through this proxy setup.

B. Central VPC with AWS Network Firewall: Establish a new VPC for internet-bound traffic and link it to the existing Transit Gateway. Incorporate AWS Network Firewall for rule-based filtering, setting up Network Firewall endpoints in each Availability Zone. Modify the default routes in all accounts to direct traffic through these Network Firewall endpoints.

C. Individual AWS Network Firewall in Each Account: Deploy an AWS Network Firewall in every AWS account for filtering, adjusting the default routes in each account to route through these individual firewalls.

D. Decentralized EC2-based Proxy in Each Account: Implement an Auto Scaling group of network-optimized EC2 instances in each AWS account, each running an open-source internet proxy for filtering. Alter all default routes to direct through these proxies.

The most suitable solution is B:

- B. Central VPC with AWS Network Firewall: This option aligns with the requirement for centralized management of rule-based filtering across all accounts. By creating a dedicated VPC for internet traffic and using AWS Network Firewall, the organization can efficiently manage and apply consistent security policies across all accounts. The use of Network Firewall endpoints in each Availability Zone ensures high availability and the capacity to handle the peak traffic load. This setup simplifies management by centralizing the filtering solution instead of deploying it individually across accounts.

The reasons why the other options are less suitable:

A. Custom Proxy in Central VPC: While this method centralizes traffic management, running and maintaining a custom proxy solution introduces significant operational overhead and complexity. It also lacks the robustness and integration of a managed service like AWS Network Firewall.

C. Individual Firewalls in Each Account: Deploying individual firewalls in each account increases operational complexity and makes it challenging to enforce consistent security policies across the entire organization.

D. Decentralized EC2-based Proxies: This approach leads to a decentralized model, requiring the management of proxy solutions in every account. It's operationally intensive and less efficient compared to a centralized AWS Network Firewall solution.

Question 16

A company currently routes its traffic to Amazon EC2 instances in a single Availability Zone through a load balancer. They are focused on enhancing security and have outlined specific requirements for a new architecture. These requirements include filtering inbound requests for common vulnerabilities, directing rejected requests to a third-party auditing application, and ensuring high availability of all resources.

The proposed solutions to address these requirements are:

A. ALB with Multi-AZ Auto Scaling, Amazon Inspector, and AWS WAF: This involves setting up a Multi-AZ Auto Scaling group with the application's AMI, an Application Load Balancer (ALB), and using Amazon Inspector for traffic monitoring. AWS WAF is integrated with the ALB and a Lambda function is used to send Inspector reports to the third-party auditing application.

B. ALB with AWS WAF and CloudWatch Logs: This option includes configuring an ALB with EC2 instances as targets, creating an AWS WAF attached to the ALB with logging enabled through Amazon CloudWatch Logs, and using a Lambda function to send logs to the third-party application.

C. ALB with AWS WAF, Kinesis Data Firehose, and AWS Managed Rules: This setup also involves configuring an ALB with EC2 instances in the target group. AWS WAF is set up with the ALB and logging is enabled, directing logs to a Kinesis Data Firehose that feeds into the third-party auditing application. Additionally, it includes subscribing to AWS Managed Rules in the AWS Marketplace.

D. Multi-AZ Auto Scaling, ALB with AWS WAF, Kinesis Data Firehose, and AWS Managed Rules: This solution proposes a Multi-AZ Auto Scaling group using the application's AMI, an ALB targeting this Auto Scaling group, and an AWS WAF integrated with the ALB. Logging is

enabled via Kinesis Data Firehose to the third-party auditing application, and the setup subscribes to AWS Managed Rules in the AWS Marketplace.

The most suitable solution is D:

- **D. Multi-AZ Auto Scaling, ALB with AWS WAF, Kinesis Data Firehose, and AWS Managed Rules**: This approach effectively meets all the company's requirements. The Multi-AZ Auto Scaling group ensures high availability across multiple Availability Zones. The ALB distributes traffic across these instances. AWS WAF provides the necessary filtering of inbound requests for vulnerabilities and integrates with the ALB. The logging of rejected requests is efficiently managed through Kinesis Data Firehose, which directly sends the data to the third-party auditing application. Subscribing to AWS Managed Rules enhances the security posture by leveraging pre-configured rules that address common vulnerabilities.

The reasons why the other options are less suitable:

A. **ALB with Amazon Inspector and AWS WAF**: While it uses a Multi-AZ Auto Scaling group and ALB for high availability, Amazon Inspector is not the ideal tool for real-time monitoring of web traffic or filtering requests based on WAF rules.

B. **ALB with AWS WAF and CloudWatch Logs**: This option lacks the high availability setup provided by a Multi-AZ Auto Scaling group. CloudWatch Logs isn't as direct or efficient for integrating with third-party applications compared to Kinesis Data Firehose.

C. **ALB with AWS WAF, Kinesis Data Firehose, and AWS Managed Rules**: This solution misses the high availability component that the Multi-AZ Auto Scaling group offers.

Question 17

A company's AWS-hosted application, comprising microservices running on a fleet of Amazon EC2 instances across multiple Availability Zones and managed by an Application Load Balancer, has recently expanded to include a new REST API developed using Amazon API Gateway. Some existing microservices on EC2 instances need to interact with this new API. The company's key stipulations are that the API should not be publicly accessible and that sensitive data should not travel over the public internet.

To address these requirements, the proposed solutions are:

A. **Site-to-Site VPN with API Gateway**: Establish a Site-to-Site VPN between the VPC (where EC2 instances reside) and API Gateway. Use API Gateway to issue unique API keys for each microservice and configure the API methods to mandate these keys.

B. Interface VPC Endpoint for API Gateway: Implement an interface VPC endpoint specifically for API Gateway, setting an endpoint policy to limit access to the particular API. Apply a resource policy on API Gateway to restrict access exclusively to the VPC endpoint, and change the API Gateway endpoint to private.

C. IAM Authentication and Transit Gateway: Adapt API Gateway to utilize IAM authentication and modify the IAM policy for the EC2 instances' IAM role to permit API Gateway access. Relocate the API Gateway into a new VPC, and connect the VPCs via a transit gateway.

D. AWS Global Accelerator with API Gateway: Create an AWS Global Accelerator, linking it to API Gateway. Update the VPC route tables to direct traffic to the Global Accelerator's endpoint IP address and integrate API keys for authentication.

The most appropriate solution is B:

- B. Interface VPC Endpoint for API Gateway: This option aligns with the company's requirements by creating a private connection between the EC2 instances and the API Gateway. The interface VPC endpoint enables the EC2 instances to access the API Gateway without routing traffic through the public internet, maintaining the desired security posture. Additionally, the endpoint policy and resource policy on API Gateway ensure that only specified traffic from within the VPC can access the API, thereby preventing public internet access.

The reasons why the other options are less suitable:

A. Site-to-Site VPN with API Gateway: Using a Site-to-Site VPN and API keys does not inherently prevent the API from being accessed over the public internet. It also adds complexity in terms of VPN management.

C. IAM Authentication and Transit Gateway: While IAM authentication adds a layer of security, moving the API Gateway into a new VPC and connecting via a transit gateway is unnecessary for the given scenario. This approach does not address the requirement to avoid public internet exposure for the API.

D. AWS Global Accelerator with API Gateway: Global Accelerator is designed to optimize and secure global traffic access to your services, but it does not inherently restrict API access to internal networks or prevent data from traversing the public internet.

Question 18

A company operating on AWS has its ecommerce website on Amazon EC2 and stores static data in Amazon S3. They have a single AWS account managed by three engineers. The issue arises when an engineer inadvertently modifies an EC2 security group setting, leading to

compliance problems. The company needs a system to monitor and alert on noncompliant changes to EC2 security group configurations.

The proposed solutions to track these changes and alert on noncompliance are:

A. Implement AWS Organizations and Use Service Control Policies (SCPs): This would involve setting up AWS Organizations and applying SCPs to monitor and control changes to security group configurations in the AWS account.

B. Utilize AWS CloudTrail and Amazon CloudWatch Rules: This involves enabling AWS CloudTrail to log changes to EC2 security groups and configuring CloudWatch rules to trigger alerts when noncompliant security settings are detected.

C. Enable SCPs on the AWS Account for Alerts: This option suggests using SCPs within the existing AWS account to alert on noncompliant security group changes.

D. Activate AWS Config and Use Amazon SNS for Alerts: This entails enabling AWS Config to monitor EC2 security groups for noncompliant changes and using Amazon Simple Notification Service (Amazon SNS) to send alerts when such changes occur.

The most appropriate solution is D:

- D. Enable AWS Config and Use Amazon SNS for Alerts: AWS Config is specifically designed to track configurations and changes in AWS resources, including EC2 security groups. It can evaluate these changes against compliance rules and report any noncompliance. Using Amazon SNS for alerts ensures timely notification to the engineers when noncompliant changes are made. This solution is direct and efficient for monitoring specific resource configurations and compliance enforcement.

The reasons why the other options are less suitable:

A. Implement AWS Organizations and Use SCPs: While AWS Organizations with SCPs can govern account-level permissions and actions, they are not designed to track and alert on specific resource-level changes like alterations to EC2 security group configurations.

B. Utilize AWS CloudTrail and Amazon CloudWatch Rules: Although CloudTrail captures all account activity, including changes to security groups, and CloudWatch can alert on specific events, this setup is more complex for tracking compliance of resource configurations compared to AWS Config.

C. Enable SCPs on the AWS Account for Alerts: SCPs are not the right tool for tracking and alerting on specific configuration changes. They are more suited for setting broad permissions and controls at the account or organization level.

Question 19

A company utilizes IoT sensors to monitor urban traffic patterns. The sensors feed data into Amazon Kinesis Data Streams, which is then accessed by multiple applications for analysis and aggregation. However, some of these applications are experiencing throttling issues, specifically encountering the `ReadProvisionedThroughputExceeded` error while reading from the stream. The solutions architect needs to address this issue effectively.

The potential actions to resolve this throttling problem are:

A. Increase Shard Count in the Stream: Resharding to increase the number of shards in the Kinesis Data Stream can enhance the stream's capacity to handle more read and write operations, thereby reducing throttling.

B. Adjust Kinesis Producer Library (KPL) Polling Frequency: While KPL is used to efficiently produce data to Kinesis Streams, adjusting its polling frequency is more related to the data production side and won't directly address the read throughput issue experienced by the consumers.

C. Utilize Enhanced Fan-Out Feature: Enhanced fan-out provides a dedicated throughput capacity for each consumer, enabling higher performance and eliminating contention between consumers. This can significantly reduce throttling problems.

D. Decrease Shard Count in the Stream: Reducing the number of shards would decrease the stream's overall capacity for read and write operations, potentially exacerbating the throttling issue.

E. Implement Error Retry and Exponential Backoff: Incorporating error handling with retry logic and exponential backoff in the consumer applications can help manage read requests more effectively, particularly in situations where intermittent throttling occurs.

F. Configure Dynamic Partitioning: Dynamic partitioning is a feature used to optimize the write operation into Kinesis Data Streams and does not directly address read throughput issues.

The best actions to take are: A, C, and E.

- A. Increase Shard Count in the Stream: This directly addresses the throughput limitations by increasing the stream's capacity to handle more data, thereby alleviating throttling on the read side.
- C. Utilize Enhanced Fan-Out Feature: By providing dedicated throughput to consumers, this feature directly tackles the issue of throttling and ensures more consistent and higher performance data consumption.

- E. Implement Error Retry and Exponential Backoff: This is a best practice in application development, especially in distributed systems where intermittent failures like throttling can occur. It helps to gracefully handle such errors and reduces the impact of throttling.

The reasons why the other options are less suitable:

B. Use the Kinesis Producer Library (KPL) and Adjust Polling Frequency: The Kinesis Producer Library (KPL) is designed to efficiently send data to a Kinesis Data Stream, and its primary function is related to the writing (producing) side of Kinesis operations. Adjusting the polling frequency in KPL impacts how data is ingested into the stream, not how it is read by consumers. The issue at hand is related to reading data (consumers experiencing throttling), so adjustments in the KPL's polling frequency won't directly solve the problem of `ReadProvisionedThroughputExceeded` errors.

D. Reshard the Stream to Reduce the Number of Shards in the Stream: Resharding to reduce the number of shards in a Kinesis Data Stream would decrease the overall capacity of the stream to handle read and write operations. In a scenario where throttling is already occurring due to high read demand, reducing the number of shards would likely exacerbate the issue by further limiting the stream's capacity to handle simultaneous read requests. The goal should be to increase, not decrease, the stream's capacity to handle higher loads.

F. Configure the Stream to Use Dynamic Partitioning: Dynamic partitioning in Kinesis Data Streams is a feature that helps optimize the distribution of incoming data records across shards when writing to the stream. It's designed to improve the ingestion side of Kinesis operations by automatically adjusting the number of shards to match the volume of incoming data records. However, this feature does not directly address or resolve issues related to reading data from the stream, such as the throttling experienced by consumers. The issue in question is related to the consumption (reading) of data, not its ingestion (writing).

Question 20

A company managing AWS accounts through AWS Organizations seeks an efficient way to identify and downsize Amazon EC2 instances with low CPU and memory usage. They need a solution that provides a comprehensive list of underutilized instances and actionable recommendations for downsizing them.

The potential solutions are:

A. Use AWS Marketplace Tool for Monitoring; Store Data in S3; Python Script for Analysis: This involves installing a CPU and memory monitoring tool from AWS Marketplace on all EC2 instances, storing the data in Amazon S3, and writing a Python script to analyze the data and determine downsizing options based on EC2 instance pricing. This approach requires

significant manual effort in setting up and maintaining the monitoring system, scripting for analysis, and integrating with pricing information.

B. Install CloudWatch Agent via Systems Manager; Use Cost Explorer in Management Account: This option proposes installing the Amazon CloudWatch agent on all EC2 instances using AWS Systems Manager, which simplifies the deployment process. It then suggests using AWS Cost Explorer in the organization's management account to get resource optimization recommendations. Cost Explorer can provide insights into resource utilization and suggest downsizing options, minimizing the need for manual analysis.

C. Install CloudWatch Agent via Systems Manager; Use Cost Explorer in Each Account: Similar to option B, but instead of aggregating the data in the management account, it retrieves optimization recommendations from AWS Cost Explorer in each individual account. This method is less efficient than B, as it requires managing recommendations across multiple accounts instead of centrally.

D. Install CloudWatch Agent via Systems Manager; Lambda Function for Data Extraction; Use Athena for Analysis: This involves using AWS Systems Manager to install CloudWatch agents, creating an AWS Lambda function to extract utilization data, storing this data in S3, and analyzing it with Amazon Athena. It also involves manual referencing of EC2 instance pricing for downsizing recommendations. This solution is complex and requires significant effort to set up and maintain the data pipeline and analysis process.

The best solution is B:

- B. Install CloudWatch Agent via Systems Manager; Use Cost Explorer in Management Account: This solution streamlines the process of identifying underutilized instances. By using AWS Systems Manager to deploy the CloudWatch agent, the company can easily collect necessary utilization metrics. AWS Cost Explorer then analyzes this data and provides downsizing recommendations. Utilizing the management account for Cost Explorer allows for an aggregated view across all accounts in AWS Organizations, offering a centralized and efficient way to manage and analyze resource utilization and optimization opportunities.

Reasons why the other options are less suitable:

A. Use AWS Marketplace Tool for Monitoring; Store Data in S3; Python Script for Analysis: This approach requires significant manual effort and technical expertise. First, it involves selecting and installing a third-party monitoring tool from AWS Marketplace on all EC2 instances, which can be complex and time-consuming. Storing the findings in Amazon S3 and then writing and maintaining a Python script for analysis adds additional layers of complexity. This process not only demands continuous management but also necessitates expertise in scripting and data analysis. Moreover, integrating and constantly updating EC2 instance

pricing information for downsizing recommendations would require manual intervention and constant upkeep.

C. Install CloudWatch Agent via Systems Manager; Use Cost Explorer in Each Account: While this option correctly suggests using the CloudWatch agent for gathering metrics and AWS Cost Explorer for optimization recommendations, it loses efficiency by requiring the retrieval of these recommendations from each individual AWS account within the organization. This fragmented approach is less efficient compared to aggregating and analyzing data at the organization's management account level. It increases operational complexity by necessitating the management of recommendations across multiple accounts, which can be cumbersome for a large organization with numerous accounts.

D. Install CloudWatch Agent via Systems Manager; Lambda Function for Data Extraction; Use Athena for Analysis: This solution is overly complex for the task at hand. It involves multiple steps: installing the CloudWatch agent on all EC2 instances, creating and maintaining a Lambda function to extract the utilization data, storing this data in Amazon S3, and then using Amazon Athena to analyze the data. Additionally, manually referencing EC2 instance pricing for downsizing adds further complexity. This method requires significant effort in setting up and maintaining the data extraction and analysis pipeline. It also demands proficiency in AWS Lambda and Amazon Athena, as well as the ability to effectively write and manage the queries needed for Athena.

Question 21

A company is using a custom network analysis software on Amazon EC2 instances within a VPC, managed by AWS CloudFormation and an Auto Scaling group. The software analyzes incoming and outgoing VPC traffic. However, when the software fails, the Auto Scaling group replaces the faulty instance, but doesn't update the network routes accordingly. The task is to determine the best combination of steps to resolve this issue from the following options:

A. Establish alarms based on EC2 instance health metrics, triggering the Auto Scaling group to replace any non-functional instance. B. Modify the CloudFormation template to install the Amazon CloudWatch agent on EC2 instances, set to monitor and report the application's process metrics. C. Revise the CloudFormation template to deploy AWS Systems Manager Agent on EC2 instances, aimed at monitoring and reporting the application's process metrics. D. Set up a CloudWatch alarm for a custom metric that detects software failure, and configure it to notify an Amazon SNS topic. E. Develop an AWS Lambda function that acts on Amazon SNS notifications to decommission the failing instance and reroute network traffic to the new instance. F. Include a condition in the CloudFormation template to automatically update network routes when a new instance is launched.

The most suitable combination of steps to address the issue is B, D, and E:

- B (Correct): This step involves modifying the CloudFormation template to ensure that the CloudWatch agent is installed on the EC2 instances. The agent's role is to monitor the software's process metrics, which is crucial for early detection of software failures.
- D (Correct): Setting up a CloudWatch alarm for custom metrics specific to software failure allows for timely identification of issues. Configuring this alarm to send notifications to an SNS topic ensures that appropriate automated responses are triggered when the software stops working.
- E (Correct): Creating an AWS Lambda function to respond to SNS messages facilitates the automated removal of the failed instance from the service. It also ensures that the network routes are updated to direct traffic to the newly launched instance, maintaining the continuity and effectiveness of the network analysis.

Reasons why the other options are less suitable:

- A (Incorrect): While creating alarms based on EC2 status check metrics is useful for monitoring instance health, it doesn't address the specific challenge of updating network routes when an instance is replaced.
- C (Incorrect): Installing the AWS Systems Manager Agent, though beneficial for instance management, does not directly contribute to monitoring the specific process metrics of the network analysis software, which is critical in this scenario.
- F (Incorrect): Writing a condition in the CloudFormation template to update network routes could be less effective and less responsive compared to a Lambda function. The latter offers more flexibility and immediacy in handling dynamic changes in the infrastructure.

Question 22

A company is rolling out a new microservices-based on-demand video application, projected to attract 5 million users at launch and grow to 30 million in six months. This application is hosted on Amazon Elastic Container Service (Amazon ECS) running on AWS Fargate and uses HTTPS protocol. To manage updates effectively, a solutions architect is tasked with implementing blue/green deployments. The chosen solution must ensure traffic distribution across ECS services via a load balancer and enable automatic task number adjustments in response to Amazon CloudWatch alarms.

The potential solutions for meeting these requirements are:

A. Set Up ECS with Blue/Green Deployments and a Network Load Balancer: This option involves configuring the ECS services for blue/green deployments and using a Network Load Balancer. It also suggests requesting increases in the service quota for the number of tasks per service as user demand grows.

B. ECS Blue/Green Deployments with Network Load Balancer and Cluster Autoscaler: This solution configures ECS services for blue/green deployments with a Network Load Balancer and proposes implementing an Auto Scaling group for each ECS service, utilizing the Cluster Autoscaler.

C. Blue/Green Deployments in ECS with Application Load Balancer and Cluster Autoscaler: This approach configures the ECS services for blue/green deployments, uses an Application Load Balancer, and recommends setting up an Auto Scaling group for each ECS service with the Cluster Autoscaler.

D. ECS Blue/Green Deployments with Application Load Balancer and Service Auto Scaling: This method involves configuring ECS services for blue/green deployments, using an Application Load Balancer, and implementing Service Auto Scaling for each ECS service.

The most appropriate solution is D:

- D. ECS Blue/Green Deployments with Application Load Balancer and Service Auto Scaling: This configuration is ideal for the company's requirements. The Application Load Balancer is well-suited for handling HTTPS traffic and distributing it efficiently across the ECS services. The key advantage is the implementation of Service Auto Scaling for each ECS service, which allows the application to automatically adjust the number of tasks based on CloudWatch alarm triggers. This ensures that the application can scale dynamically in response to user demand, crucial for managing the anticipated rapid growth in user base.

The reasons why the other options are less suitable:

- A. Set Up ECS with Blue/Green Deployments and a Network Load Balancer: While this option provides for blue/green deployments and uses a Network Load Balancer, the manual request for increasing service quotas doesn't offer the automatic scalability needed to handle rapid changes in demand.
- B. ECS Blue/Green Deployments with Network Load Balancer and Cluster Autoscaler: The use of Cluster Autoscaler is more aligned with Kubernetes environments and may not provide the optimal scalability solution for ECS services running on AWS Fargate.
- C. Blue/Green Deployments in ECS with Application Load Balancer and Cluster Autoscaler: Similar to option B, the inclusion of Cluster Autoscaler is not the most effective choice for ECS on Fargate, despite the use of an Application Load Balancer.

Question 23

A company operates a containerized application on the AWS Cloud, utilizing Amazon Elastic Container Service (Amazon ECS) on Amazon EC2 instances grouped in an Auto Scaling group. Container images for this application are stored in Amazon Elastic Container Registry

(Amazon ECR). Each new image version uploaded to the registry is tagged uniquely. The company seeks a system that can inspect these new image versions for vulnerabilities and exposures, specifically aiming to automatically remove tags of new images with Critical or High severity findings and to alert the development team about such deletions.

The potential solutions for this requirement are:

A. **Enable Scan-on-Push and Use EventBridge with Step Functions**: Activate the 'scan on push' feature for the repository. Employ Amazon EventBridge to trigger an AWS Step Functions state machine when an image scan completes and reveals Critical or High severity findings. This state machine should be responsible for removing the image tag of such images and alerting the development team via Amazon Simple Notification Service (Amazon SNS).

B. **Scan-on-Push with SQS Queue and Lambda Function**: Implement 'scan on push' for the repository and configure scan results to be sent to an Amazon Simple Queue Service (Amazon SQS) queue. Utilize an AWS Lambda function triggered by new messages in the SQS queue to delete image tags with Critical or High severity findings, and inform the development team using Amazon Simple Email Service (Amazon SES).

C. **Scheduled Lambda Function for Manual Scans and EventBridge Integration**: Set a schedule for an AWS Lambda function to initiate manual image scans every hour. Use Amazon EventBridge to activate another Lambda function upon scan completion. This second function should delete image tags of images with Critical or High severity findings and notify the development team through Amazon Simple Notification Service (Amazon SNS).

D. **Periodic Image Scan with SQS Queue and Step Functions**: Configure a periodic image scan for the repository. Set scan results to be forwarded to an Amazon SQS queue. Trigger an AWS Step Functions state machine when new messages appear in the SQS queue, programmed to remove the image tag of images with Critical or High findings and inform the development team via Amazon Simple Email Service (Amazon SES).

The most suitable solution is A:

- **A. Enable Scan-on-Push and Use EventBridge with Step Functions**: This solution is the most efficient and effective. Enabling 'scan on push' ensures immediate scanning of new image versions as they are pushed to the registry. Using Amazon EventBridge to initiate an AWS Step Functions state machine allows for a seamless and automated workflow to handle the deletion of non-compliant image tags and the notification process via Amazon SNS. This method provides immediate detection and response, aligning perfectly with the company's requirements.

The reasons why the other options are less suitable:

- B. Scan-on-Push with SQS Queue and Lambda Function: While this option involves scanning on image push and responding to vulnerabilities, the use of a Lambda function for deletion and Amazon SES for notification introduces additional complexity and potential points of failure compared to the more streamlined solution A.
- C. Scheduled Lambda Function for Manual Scans and EventBridge Integration: This option is less efficient as it relies on manual scans scheduled every hour, which could delay the detection of vulnerabilities in newly pushed images compared to the immediate scanning in solution A.
- D. Periodic Image Scan with SQS Queue and Step Functions: Similar to C, this solution's periodic scanning could result in delays in identifying vulnerabilities. Immediate scanning upon image push, as in solution A, offers a more timely and effective approach.

Question 24

An enterprise operating numerous workloads on AWS uses AWS Organizations for account management. These workloads, hosted on Amazon EC2, AWS Fargate, and AWS Lambda, exhibit fluctuating demand, with some months showing high usage and others much lower. The company is looking to maximize its computing cost efficiency over the next three years. To achieve this, a solutions architect has analyzed a six-month average usage across all accounts within the organization.

The available strategies to enhance cost savings for the organization's overall compute usage are:

A. Acquire Reserved Instances for the Organization: This involves purchasing Reserved Instances for the organization based on the size and quantity of the most frequently used EC2 instances in the member accounts.

B. Invest in a Compute Savings Plan from the Management Account: Opt for a Compute Savings Plan for the entire organization, making the purchase from the management account, guided by the recommendations at the management account level.

C. Buy Reserved Instances for High-Usage Member Accounts: This option suggests purchasing Reserved Instances specifically for each member account that demonstrated high EC2 usage, as per the last six months' data.

D. Procure an EC2 Instance Savings Plan for Each Member Account: This involves buying an EC2 Instance Savings Plan for each member account from the management account, based on the EC2 usage data from the past six months.

The most effective solution is B:

- B. Invest in a Compute Savings Plan from the Management Account: A Compute Savings Plan offers the most flexibility and cost savings for an organization with varying compute needs across different services (EC2, Fargate, and Lambda). By purchasing from the management account, the plan applies to the entire organization, allowing all accounts to benefit from the savings regardless of fluctuations in usage. The plan's ability to adapt to different compute usage patterns across various AWS services makes it the most comprehensive and cost-effective option.

The reasons why the other options are less suitable:

- A. Acquire Reserved Instances for the Organization: While Reserved Instances can offer savings for EC2, they lack the flexibility needed for varying usage patterns and do not cover other compute services like Fargate and Lambda.
- C. Buy Reserved Instances for High-Usage Member Accounts: This approach is too rigid and focuses only on EC2. It does not account for varying usage patterns and does not provide the broad coverage of a Compute Savings Plan.
- D. Procure an EC2 Instance Savings Plan for Each Member Account: This strategy, although tailored to each account's EC2 usage, misses out on the broader benefits of a Compute Savings Plan. It does not offer savings for other services like Fargate and Lambda and lacks the holistic approach needed for an organization with diverse compute needs.

Question 25

A large organization, utilizing AWS Organizations for managing several AWS accounts, has a specific requirement from its finance team. They need an alert system to monitor daily AWS expenditures across the organization and notify them via email when spending exceeds 80% of the set daily budget. A solution architect is tasked to design a monitoring and notification system that fits these criteria.

The potential solutions for this requirement are:

A. Set Up AWS Budgets with Email Alerts via Amazon SNS in the Management Account: Implement AWS Budgets within the management account to define a daily budget, placing the alert threshold at 80%. Configure Amazon Simple Notification Service (Amazon SNS) to send email notifications to the finance team when this threshold is breached.

B. Enable AWS Trusted Advisor with Organizational View for Cost Alerts: Utilize the organizational view feature in AWS Trusted Advisor within the management account. Create a cost optimization report with an 80% alert threshold and set up the finance team’s email addresses to receive these alerts.

C. Implement Cost Monitoring with AWS Control Tower: Enroll the organization in AWS Control Tower and activate its optional cost control feature. Set a budget threshold at 80% and configure Amazon SNS to alert the finance team when the threshold is reached.

D. Use AWS Cost and Usage Reports with EventBridge and Amazon Athena for Cost Alerts: Arrange for daily AWS Cost and Usage Reports from member accounts to be sent to an S3 bucket in the management account. Schedule an Amazon Athena query via Amazon EventBridge to calculate daily expenses. Configure a CloudWatch alert for when costs exceed 80% of the budget, using Amazon SNS for sending notifications to the finance team.

The best solution is A:

- A. Set Up AWS Budgets with Email Alerts via Amazon SNS in the Management Account: This approach is the most straightforward and effective. AWS Budgets is tailored for tracking and managing AWS spending. By establishing a daily budget with an alert threshold of 80%, coupled with Amazon SNS for email notifications, this solution directly addresses the finance team's needs. It offers a simple, automated system for budget tracking and alerting.

Why the other options are less appropriate:

- B. Enable AWS Trusted Advisor with Organizational View for Cost Alerts: Although Trusted Advisor provides valuable insights, its primary focus isn't on budget tracking and alerts. Setting up cost optimization reports and alerts in Trusted Advisor is more complex and less direct than using AWS Budgets.
- C. Implement Cost Monitoring with AWS Control Tower: Deploying AWS Control Tower solely for this purpose is an overreach. Control Tower is designed for broader governance and compliance, making it a more complex and less direct solution for budget tracking and alerting compared to AWS Budgets.
- D. Use AWS Cost and Usage Reports with EventBridge and Amazon Athena for Cost Alerts: This method is too convoluted and involves multiple AWS services and configurations. It requires more setup and ongoing maintenance than the straightforward and focused functionality of AWS Budgets.

Question 26

An art auction company, serving a client base in North America and Europe, has its application running on Amazon EC2 instances in the AWS us-east-1 Region. The application allows artists to upload high-resolution, large-sized image files of their artwork directly from their mobile phones to a centralized Amazon S3 bucket, also located in the us-east-1 Region. However, European users are experiencing slow upload speeds for their images. A solutions architect is tasked with enhancing the upload performance for these users.

The suggested methods to improve the image upload process are:

A. Implement S3 Multipart Uploads in the Application: Modify the application to utilize S3 multipart uploads for handling large image files.

B. Establish an Amazon CloudFront Distribution with the Application as Custom Origin: Set up a CloudFront distribution and configure it to use the application as its custom origin.

C. Enable S3 Transfer Acceleration on the Buckets: Turn on S3 Transfer Acceleration for the S3 buckets to improve the speed of file transfers over long distances.

D. Create an Auto Scaling Group for EC2 Instances with a Scaling Policy: Implement an Auto Scaling group for the EC2 instances and devise a scaling policy.

The most effective solution is C:

- C. Enable S3 Transfer Acceleration on the Buckets: This option directly addresses the issue of slow uploads over long distances. S3 Transfer Acceleration optimizes the transfer path, using Amazon's globally distributed edge locations to accelerate uploads to the S3 bucket. This is particularly beneficial for users in Europe uploading to a bucket in the us-east-1 Region, as it significantly reduces latency and increases transfer speeds.

Why the other options are less suitable:

- A. Implement S3 Multipart Uploads in the Application: While multipart uploads can improve the efficiency of uploading large files, they do not specifically address the latency issues faced by users at a great geographical distance from the S3 bucket's location.
- B. Establish an Amazon CloudFront Distribution with the Application as Custom Origin: CloudFront is primarily a content delivery network service that accelerates the delivery of content to users. However, it is more effective for content downloading rather than speeding up uploads from users to S3 buckets.
- D. Create an Auto Scaling Group for EC2 Instances with a Scaling Policy: Auto Scaling groups and scaling policies enhance the application's ability to handle varying loads by adjusting the number of EC2 instances. However, they do not directly impact the speed of uploads from remote locations to the S3 bucket.

Question 27

A company plans to migrate its multi-tier web application, comprising web, application, and database layers, from an on-premises setup to AWS. The application needs to be containerized, fault-tolerant, scalable, and should ensure certain frequently used data remains accessible across application servers. Additionally, the frontend web servers require

session persistence and scalability in response to traffic fluctuations. The company is seeking a solution that minimizes operational overhead.

The proposed solutions to achieve these objectives are:

A. Utilize Amazon ECS on AWS Fargate with Amazon EFS and SQS: Deploy the application on Amazon Elastic Container Service (Amazon ECS) using AWS Fargate. Employ Amazon Elastic File System (Amazon EFS) for shared data access between web and application tiers. Use Amazon Simple Queue Service (Amazon SQS) for storing session data of frontend web servers.

B. Deploy on Amazon ECS on Amazon EC2 with ElastiCache and EBS Multi-Attach: Run the application on Amazon ECS hosted on Amazon EC2 instances. Implement Amazon ElastiCache for Redis to manage session data for frontend web servers. Use Amazon Elastic Block Store (Amazon EBS) with Multi-Attach feature on EC2 instances spread across multiple Availability Zones for data storage.

C. Run on Amazon EKS with Managed Node Groups and EFS: Operate the application on Amazon Elastic Kubernetes Service (Amazon EKS), configuring it with managed node groups. Utilize ReplicaSets for web servers and applications. Establish an Amazon Elastic File System (Amazon EFS) and mount it across all EKS pods for storing session data of frontend web servers.

D. Implement Amazon EKS with Managed Node Groups, DynamoDB, and EFS: Set up the application on Amazon Elastic Kubernetes Service (Amazon EKS) using managed node groups. Run the web servers and application as Kubernetes deployments within the EKS cluster. Store session data of frontend web servers in an Amazon DynamoDB table. Create an Amazon Elastic File System (Amazon EFS) volume for all applications to mount at deployment.

The best solution is D:

- D. Implement Amazon EKS with Managed Node Groups, DynamoDB, and EFS: This solution optimally meets the requirements for fault tolerance, scalability, and reduced operational overhead. Using Amazon EKS with managed node groups simplifies container orchestration and scaling. DynamoDB provides a highly available and scalable storage for session data, efficiently handling the session persistence requirement for frontend web servers. EFS integration ensures shared and consistent data access across applications. This combination offers a robust, scalable architecture with minimal maintenance.

Why the other options are less suitable:

- A. Utilize Amazon ECS on AWS Fargate with Amazon EFS and SQS: While this setup offers scalability and shared data access, using SQS for session data storage is unconventional and may not provide the required session persistence effectively compared to a database solution like DynamoDB.
- B. Deploy on Amazon ECS on Amazon EC2 with ElastiCache and EBS Multi-Attach: This approach involves higher operational overhead due to the management of EC2 instances. EBS Multi-Attach has limitations in terms of scalability and cross-instance data sharing.
- C. Run on Amazon EKS with Managed Node Groups and EFS: Although this setup provides a scalable environment and shared data access via EFS, it lacks an optimal solution for session persistence. EFS might not be the most efficient choice for storing session data compared to a database service like DynamoDB.

Question 28

A solutions architect is tasked with transferring critical legacy Microsoft SQL Server databases to AWS, aiming to modernize the data architecture while ensuring minimal downtime during the migration process.

The potential solutions to accomplish this migration are:

A. Migrate Using AWS Application Migration Service and AWS SCT: Initiate the migration by utilizing AWS Application Migration Service and the AWS Schema Conversion Tool (AWS SCT). Conduct an upgrade in the current environment prior to migration. After the migration and cutover, transfer the data to Amazon Aurora Serverless and redirect the application endpoints to the new Aurora database.

B. Migrate via AWS Database Migration Service (AWS DMS) to Amazon S3, then to RDS: Employ AWS DMS to transfer the database directly to Amazon S3, using change data capture (CDC) for ongoing replication. After achieving full synchronization between the source and the S3 target, migrate the data from S3 into an Amazon RDS for Microsoft SQL Server instance.

C. Utilize Native Database High Availability Tools for Replication to Amazon RDS: Leverage the native high availability tools of the database to establish a connection between the existing system and an Amazon RDS for Microsoft SQL Server instance. Set up the necessary replication processes. Upon completion of data replication, transition the workload fully to the Amazon RDS instance.

D. Rehost Database on Amazon EC2 via AWS Application Migration Service, then to RDS: Use AWS Application Migration Service to rehost the existing database server on an Amazon EC2 instance. After completing data replication, detach the database from the EC2 instance

and transfer it to an Amazon RDS for Microsoft SQL Server instance. Then reattach the database and proceed with the network cutover.

The most suitable solution is C:

- **C. Utilize Native Database High Availability Tools for Replication to Amazon RDS**: This approach is the most effective for ensuring minimal downtime. By using native database high availability and replication tools, the database can be seamlessly replicated to an Amazon RDS for Microsoft SQL Server instance. This method provides a direct, controlled migration path, maintaining data integrity and availability, which is crucial for legacy database systems.

Why the other options are less appropriate:

- **A. Migrate Using AWS Application Migration Service and AWS SCT**: This method introduces complexity by involving a schema conversion and an in-place upgrade before migration. Additionally, moving to Amazon Aurora Serverless from a legacy SQL Server requires significant changes and testing, which might not be feasible for critical legacy systems.
- **B. Migrate via AWS DMS to Amazon S3, then to RDS**: Using AWS DMS to move data to S3 and then to RDS introduces additional steps and complexity. This two-stage process can increase the risk of data inconsistency and prolong the migration duration, which contradicts the goal of near-zero downtime.
- **D. Rehost Database on Amazon EC2 via AWS Application Migration Service, then to RDS**: This option, involving an initial rehosting on EC2 followed by a move to RDS, is more complex and time-consuming. It adds additional steps in the migration process, increasing the potential for extended downtime, which is not ideal for critical legacy databases.

Question 29

A company managing a multi-application environment across multiple Availability Zones in a single AWS Region is facing higher than expected data transfer charges. The company oversees two organizations in AWS Organizations following a recent acquisition. In one organization, several service provider applications are developed as AWS PrivateLink-powered VPC endpoint services. In the other, multiple service consumer applications are created. To curtail these escalating costs, the solutions architect is tasked with establishing guidelines for developers that will reduce data transfer charges across the entire environment.

The proposed solutions to minimize data transfer costs are (choose two):

A. Share Subnets Using AWS Resource Access Manager: Utilize AWS Resource Access Manager within the organization to share subnets that host the service provider applications with other accounts.

B. Consolidate Service Applications in the Same Organization: Place both the service provider and consumer applications in AWS accounts within the same AWS Organization.

C. Disable Cross-Zone Load Balancing on Network Load Balancers: Turn off cross-zone load balancing for Network Load Balancers used in all service provider application deployments.

D. Use Availability Zone-Specific Endpoint Services: Ensure that service consumer compute resources utilize the Availability Zone-specific endpoint service by leveraging the endpoint's local DNS name.

E. Adopt a Savings Plan for Inter-Availability Zone Data Transfer: Create a Savings Plan to cover the expected costs of inter-Availability Zone data transfer within the organization.

The most suitable solutions are A and D:

- A. Share Subnets Using AWS Resource Access Manager: This approach can help localize network traffic by sharing subnets within the same organization, potentially reducing cross-Availability Zone data transfer costs.
- D. Use Availability Zone-Specific Endpoint Services: By directing service consumer applications to use the local DNS name for endpoint services, data transfer can be confined to the same Availability Zone, thus reducing costs associated with data transfer across different Zones.

Reasons why the other options are less suitable:

- B. Consolidate Service Applications in the Same Organization: While this might simplify management, it does not inherently impact data transfer costs, especially those related to cross-Availability Zone transfers.
- C. Disable Cross-Zone Load Balancing on Network Load Balancers: This could reduce data transfer costs, but at the expense of potentially impacting application performance and availability.
- E. Adopt a Savings Plan for Inter-Availability Zone Data Transfer: Savings Plans typically target compute and EC2 instance usage rather than data transfer costs, and might not effectively address the specific issue of minimizing cross-Availability Zone data transfers.

Question 30

A business is looking to shift its nightly 200 GB Microsoft SQL Server database backups from an on-premises setup to Amazon S3 for enhanced storage capabilities. They have already established a 10 Gbps AWS Direct Connect link from their local data center to AWS. The objective is to find the most cost-effective method to transfer these nightly backups to Amazon S3.

The available options for transferring these backups are:

A. Implement AWS Storage Gateway File Gateway for S3: Set up a new S3 bucket and deploy an AWS Storage Gateway file gateway within the connected VPC. Create an SMB file share on this file gateway and configure the database to write its nightly exports to this SMB file share.

B. Utilize Amazon FSx for Windows File Server (Single-AZ): Create a new S3 bucket and establish an Amazon FSx for Windows File Server Single-AZ file system in the VPC connected to Direct Connect. Set up an SMB file share and write the nightly database exports to this file share. Also, enable nightly backups for this FSx file system.

C. Use Amazon FSx for Windows File Server (Multi-AZ): Set up a new S3 bucket and deploy an Amazon FSx for Windows File Server Multi-AZ file system within the connected VPC. Create an SMB file share for database exports and enable nightly backups.

D. Deploy AWS Storage Gateway Volume Gateway for S3: Create a new S3 bucket and install an AWS Storage Gateway volume gateway within the connected VPC. Establish an SMB file share on this volume gateway, write nightly exports to it, and automate the copying of this data to the S3 bucket.

The most suitable solution is A:

- A. Implement AWS Storage Gateway File Gateway for S3: This option is the most cost-effective for the company's needs. AWS Storage Gateway's file gateway enables the direct writing of files to S3 using standard storage protocols like SMB. This setup reduces complexity and offers a straightforward, cost-efficient path for storing large database backups on S3, leveraging the established Direct Connect link for high-speed, reliable transfers.

The reasons why the other options are less suitable:

- B. Utilize Amazon FSx for Windows File Server (Single-AZ): While FSx provides a fully managed Windows file system, it is more complex and likely more expensive than necessary for simply transferring backup files to S3. The additional cost for the file system service and backup features does not justify the use case.

- C. Use Amazon FSx for Windows File Server (Multi-AZ): This option adds even more cost and complexity compared to option B. The Multi-AZ deployment is overkill for the requirement of merely transferring backup files to S3.
- D. Deploy AWS Storage Gateway Volume Gateway for S3: The volume gateway provides block storage backed by S3, which is more complex and potentially more costly than a file gateway solution. It's more suited to scenarios where block storage is required for legacy applications, which is not the case here.

Question 31

A company is looking to create a network link from its on-premises data center to AWS, aiming to connect its multiple VPCs spread across various AWS Regions. The solution must support transitive routing across VPC networks, reduce outbound network traffic costs, enhance bandwidth throughput, and offer a consistent network experience for users.

The solutions proposed to meet these requirements include:

A. Set Up an AWS Site-to-Site VPN with a Central VPC and VPC Peering: Establish a Site-to-Site VPN between the on-premises data center and a newly created central VPC. Then, implement VPC peering connections from this central VPC to all other VPCs.

B. Implement AWS Direct Connect with Transit VIF and Direct Connect Gateway: Create an AWS Direct Connect link between the on-premises center and AWS. Provision a transit Virtual Interface (VIF) and connect it to a Direct Connect gateway. Associate the Direct Connect gateway with transit gateways in each region, connecting to all the VPCs.

C. Deploy a Site-to-Site VPN with a Transit Gateway: Develop an AWS Site-to-Site VPN connection from the on-premises center to a new central VPC, using a transit gateway with dynamic routing. Connect this transit gateway to all other VPCs.

D. Combine AWS Direct Connect with Site-to-Site VPN and VPC Peering: Create an AWS Direct Connect link and a Site-to-Site VPN between the on-premises data center and AWS. In each region, establish VPC peering connections from a central VPC to all other VPCs.

The most appropriate solution is B:

- B. Implement AWS Direct Connect with Transit VIF and Direct Connect Gateway: This option is the most efficient and cost-effective for the company's requirements. AWS Direct Connect provides a dedicated network connection that can increase bandwidth throughput and reduce network costs. By using a transit VIF and a Direct Connect gateway associated with transit gateways in each region, the solution enables transitive routing between all VPCs and the on-premises data center, ensuring a consistent network experience.

The reasons why the other options are less suitable:

- A. Set Up an AWS Site-to-Site VPN with a Central VPC and VPC Peering: While this setup provides connectivity, VPC peering does not support transitive routing, which means this solution cannot route traffic between VPCs through the central VPC.
- C. Deploy a Site-to-Site VPN with a Transit Gateway: This approach, while supporting transitive routing, may not provide the same level of bandwidth throughput and cost efficiency as Direct Connect. VPN connections typically have higher latency and lower throughput compared to Direct Connect.
- D. Combine AWS Direct Connect with Site-to-Site VPN and VPC Peering: This solution is unnecessarily complex and also does not support transitive routing due to the use of VPC peering. It combines multiple networking services without providing the required transitive routing capability efficiently.

Question 32

A business is transitioning its development and production environments into a new AWS Organizations structure. This structure includes individual member accounts for both development and production, with billing consolidated under the management account. A key requirement is for a solutions architect in the management account to create an IAM user capable of stopping or terminating resources in both the development and production member accounts.

The potential solutions for granting this access are:

A. Establish Cross-Account Role in Management Account: Create an IAM user and a cross-account role within the management account itself. This role should be configured with minimal necessary privileges for accessing the member accounts.

B. Create IAM Users in Each Member Account and Cross-Account Role in Management Account: Set up individual IAM users in both member accounts. In the management account, create a cross-account role with limited privileges and configure a trust policy to grant these IAM users access to the cross-account role.

C. Create IAM User in Management Account and IAM Groups in Member Accounts: Generate an IAM user in the management account and create IAM groups in the member accounts with restricted access rights. Add the management account's IAM user to these groups in each of the member accounts.

D. Create IAM User in Management Account and Cross-Account Roles in Member Accounts: Formulate an IAM user in the management account and establish cross-account roles in the member accounts. These roles should have narrowly defined access privileges, and the IAM user should be authorized to assume these roles through a trust policy.

The most appropriate solution is D:

- D. Create IAM User in Management Account and Cross-Account Roles in Member Accounts: This method aligns closely with AWS best practices for managing cross-account access. The IAM user in the management account can be given the necessary permissions to assume roles in the member accounts. These roles, created in the development and production accounts, can be tailored with least privilege access to stop or terminate resources as required. The trust policy ensures secure and controlled access management.

Reasons why the other options are less suitable:

- A. Establish Cross-Account Role in Management Account: Creating a cross-account role in the management account does not directly allow for control over resources in the member accounts. The role needs to be in the member accounts to be assumed by the management account user.
- B. Create IAM Users in Each Member Account and Cross-Account Role in Management Account: This approach unnecessarily complicates the setup by requiring the creation of additional IAM users in each member account. The standard practice is to use a single IAM user in the management account to assume roles in other accounts.
- C. Create IAM User in Management Account and IAM Groups in Member Accounts: IAM groups cannot be used across accounts. An IAM user in one account cannot be added to an IAM group in another account, making this option unfeasible.

Question 33

A company aims to establish a disaster recovery plan on AWS for their on-premises application, which operates on hundreds of Windows-based servers, all accessing a shared common drive. The company's disaster recovery objectives include a Recovery Time Objective (RTO) of 15 minutes and a Recovery Point Objective (RPO) of 5 minutes. The solution should facilitate seamless failover and fallback processes and must be cost-effective.

The proposed solutions to achieve these disaster recovery goals are:

A. Utilize AWS Storage Gateway File Gateway for Daily Backups: Implement an AWS Storage Gateway File Gateway for daily backups of the Windows servers, storing the data on Amazon S3. In case of a disaster, recover the on-premises servers using these backups. During failback, run the on-premises servers on Amazon EC2 instances.

B. Deploy Infrastructure via AWS CloudFormation and Data Replication with AWS DataSync: Create AWS CloudFormation templates for infrastructure deployment. Replicate data to Amazon Elastic File System (EFS) using AWS DataSync. In a disaster scenario, use

AWS CodePipeline to deploy the templates and restore the servers. For failback, revert the data transfer using DataSync.

C. **Implement Active-Active Environment with AWS CDK and Data Replication to Amazon S3**: Set up a multi-site active-active environment using AWS Cloud Development Kit (CDK), replicating data to Amazon S3 via the `s3 sync` command. In the event of a disaster, switch DNS endpoints to AWS. Fail back using the `s3 sync` command.

D. **Implement AWS Elastic Disaster Recovery with Data Replication to Amazon FSx for Windows File Server**: Use AWS Elastic Disaster Recovery for replicating the on-premises servers. Replicate data to an Amazon FSx for Windows File Server filesystem using AWS DataSync and mount it to AWS servers. During a disaster, switch the operation from on-premises servers to AWS, and utilize Elastic Disaster Recovery for failback to new or existing servers.

The most appropriate solution is D:

- **D. Implement AWS Elastic Disaster Recovery with Data Replication to Amazon FSx for Windows File Server**: This option best aligns with the company's RTO and RPO requirements. AWS Elastic Disaster Recovery is specifically designed for such scenarios, providing efficient server replication capabilities. Replicating data to Amazon FSx for Windows File Server using DataSync ensures the shared data is readily available on AWS. This solution supports native failover and failback, making it the most comprehensive and fitting choice for the company's needs.

The reasons why the other options are less suitable:

- **A. Utilize AWS Storage Gateway File Gateway for Daily Backups**: This approach might not meet the stringent RTO and RPO requirements due to the daily backup schedule. It's less efficient for rapid disaster recovery scenarios.
- **B. Deploy Infrastructure via AWS CloudFormation and Data Replication with AWS DataSync**: While this method provides infrastructure replication, the process of deploying CloudFormation templates and replicating data may not align with the required quick RTO and RPO.
- **C. Implement Active-Active Environment with AWS CDK and Data Replication to Amazon S3**: Setting up an active-active environment with manual data syncing can be complex and may not guarantee the RTO and RPO targets. This approach also introduces additional complexity in managing DNS switching and data syncing for failover and failback.

Question 34

A company has developed a high-performance computing (HPC) cluster in AWS to handle a workload that necessitates extensive file sharing, using Amazon EFS for storage. Initially, the cluster, comprising 100 Amazon EC2 instances, delivered satisfactory performance. However, on expanding the cluster to 1,000 EC2 instances, the company observed a significant drop in performance. To optimize the HPC cluster's performance with this increased scale, the solutions architect needs to make informed design decisions.

The potential design choices for enhancing the HPC cluster's performance are:

A. Launch Cluster in a Single Availability Zone: Configure the HPC cluster to operate entirely within one Availability Zone.

B. Attach EC2 Instances with Multiple Elastic Network Interfaces: Connect each EC2 instance in the cluster with several elastic network interfaces, specifically in multiples of four.

C. Use EC2 Instances with Elastic Fabric Adapter (EFA): Opt for EC2 instances equipped with Elastic Fabric Adapter (EFA) technology.

D. Deploy Cluster Across Multiple Availability Zones: Set up the HPC cluster to span across various Availability Zones.

E. Switch from Amazon EFS to RAID-configured Amazon EBS Volumes: Replace the existing Amazon EFS storage with a configuration of multiple Amazon EBS volumes arranged in a RAID array.

F. Transition from Amazon EFS to Amazon FSx for Lustre: Replace the Amazon EFS storage solution with Amazon FSx for Lustre, a file system optimized for high-performance workloads.

The most effective solutions are A, C, and F:

- A. Launch Cluster in a Single Availability Zone: This approach ensures minimal network latency between EC2 instances in the HPC cluster, crucial for tightly coupled workloads. Operating within a single Availability Zone can significantly boost inter-node communication efficiency.
- C. Use EC2 Instances with Elastic Fabric Adapter (EFA): EFAs are designed to offer high-performance networking for HPC workloads. They provide lower latency and higher throughput, ideal for tightly coupled cluster computing environments.
- F. Transition from Amazon EFS to Amazon FSx for Lustre: Amazon FSx for Lustre is specifically tailored for high-performance computing scenarios. It delivers faster performance for workloads that require high-speed file systems, which is more suitable for large-scale HPC clusters compared to EFS.

The reasons why the other options are less suitable:

- B. Attach EC2 Instances with Multiple Elastic Network Interfaces: Simply increasing the number of network interfaces does not inherently address the performance issues in a large-scale HPC environment. It may add complexity without resolving the underlying performance bottlenecks.
- D. Deploy Cluster Across Multiple Availability Zones: Distributing an HPC cluster across multiple Availability Zones can increase network latency, adversely affecting the performance of tightly coupled workloads.
- E. Switch from Amazon EFS to RAID-configured Amazon EBS Volumes: While RAID-configured EBS volumes can offer high IOPS, managing them at the scale of 1,000 instances is complex. Furthermore, EBS does not natively provide a shared file system, which is a key requirement for many HPC workloads.

Question 35

A company is setting up its AWS Organizations structure and seeks to implement a standardized tagging process across the organization. The intention is to mandate specific tag values when creating new resources, with each Organizational Unit (OU) having distinct tag values.

The solutions proposed to enforce this tagging strategy are:

A. Implement SCPs and OU-Specific Tag Policies: Utilize Service Control Policies (SCPs) to prevent the creation of resources without the requisite tags. Establish tag policies defining the unique tag values for each OU and attach these policies directly to the respective OUs.

B. Apply SCPs and Attach Tag Policies to the Management Account: Use SCPs to block resource creation lacking necessary tags. Formulate tag policies with specific OU values and attach these policies to the organization's management account.

C. Create SCPs for Conditional Resource Creation and OU-Based Tag Policies: Develop SCPs that allow resource creation only if the required tags are present. Design tag policies containing OU-specific tag values and attach these policies to the OUs.

D. Deny Resource Creation via SCPs Without Defined Tags and Attach to OUs: Utilize SCPs to deny the creation of resources that do not include predefined tags. Compile a list of required tags and associate this SCP with the various OUs.

The most appropriate solution is A:

- A. Implement SCPs and OU-Specific Tag Policies: This option effectively ensures compliance with the company's tagging requirements. By using SCPs, the company can enforce a blanket rule that disallows resource creation without the specified tags. The tag policies, tailored to each OU's specific tagging needs, further refine this control by

defining what those tags should be for each OU. Attaching these policies directly to the OUs allows for the necessary customization across the organization.

The reasons why the other options are less suitable:

- B. Apply SCPs and Attach Tag Policies to the Management Account: Attaching tag policies to the management account instead of individual OUs limits the ability to enforce OU-specific tag values, which is a key requirement in this scenario.
- C. Create SCPs for Conditional Resource Creation and OU-Based Tag Policies: While the idea of conditional SCPs aligns with the requirement, SCPs do not inherently support conditional statements based on resource tags. SCPs are more about granting or denying permissions rather than enforcing tagging compliance at the point of resource creation.
- D. Deny Resource Creation via SCPs Without Defined Tags and Attach to OUs: This option misses the aspect of OU-specific tag values. Merely defining a list of tags without specifying unique values for each OU does not fulfill the company's requirement for customization across different OUs.

Question 36

A company is dealing with more than 10,000 sensors that transmit data using the MQTT protocol to an in-house Apache Kafka server. This server processes the data and then saves the processed results to an Amazon S3 bucket. The company faced an issue when their Kafka server crashed, leading to data loss during the recovery period. To prevent such occurrences in the future, a solutions architect must devise a new AWS-based solution that is both highly available and scalable.

The proposed solutions for meeting these requirements are:

A. EC2-based Active/Standby Kafka Server with Route 53: Set up two Amazon EC2 instances to run the Kafka server in an active/standby setup across two Availability Zones. Use Amazon Route 53 with a failover policy to manage the domain name, directing sensor data to this domain.

B. Amazon MSK with Network Load Balancer (NLB): Transition the on-premises Kafka server to Amazon Managed Streaming for Apache Kafka (MSK). Establish a Network Load Balancer pointing to the Amazon MSK broker and enable NLB health checks. Configure the sensors to send data to the NLB.

C. AWS IoT Core with Kinesis Data Firehose and Lambda: Implement AWS IoT Core, connecting it to an Amazon Kinesis Data Firehose delivery stream. Use AWS Lambda for data transformation. Direct the sensors to send data to AWS IoT Core.

D. AWS IoT Core with EC2-hosted Kafka Server: Deploy AWS IoT Core and set up an Amazon EC2 instance to host the Kafka server. Configure AWS IoT Core to relay data to this EC2 instance, and route the sensors to send data to AWS IoT Core.

The best solution is C:

- C. AWS IoT Core with Kinesis Data Firehose and Lambda: This option effectively leverages AWS managed services to create a highly available and scalable solution. AWS IoT Core can reliably handle connections from a large number of sensors using MQTT. Kinesis Data Firehose can manage the ingestion and delivery of streaming data, while Lambda can perform the necessary data transformation. This combination eliminates the need for managing a Kafka server and provides a robust, scalable architecture to prevent data loss.

The reasons why the other options are less suitable:

- A. EC2-based Active/Standby Kafka Server with Route 53: While this provides some level of high availability, it still relies on manually managed EC2 instances for Kafka, which might not offer the scalability and fault tolerance required for handling such a large number of sensors.
- B. Amazon MSK with Network Load Balancer (NLB): Although Amazon MSK is a managed service for Apache Kafka, the solution might be more complex and costlier than necessary. It still requires setup and maintenance of an NLB and might not be the most straightforward solution compared to fully managed AWS services like IoT Core and Kinesis.
- D. AWS IoT Core with EC2-hosted Kafka Server: Hosting Kafka on an EC2 instance does not inherently provide high availability or scalability. This setup still requires manual management and scaling of the Kafka server, which does not fully address the company's need for a highly available and scalable solution.

Question 37

A company has recently migrated to the AWS Cloud, utilizing Amazon EC2 instances, Amazon Elastic File System (EFS), and Amazon RDS DB instances. To align with regulatory and business demands, the company needs to implement a backup strategy with specific criteria:

1. Backups must adhere to tailored daily, weekly, and monthly retention schedules.
2. Each backup must be replicated to a secondary AWS Region immediately after being captured.
3. The backup solution should offer centralized monitoring of backup statuses.
4. The system must instantly notify the relevant personnel if any backup operation fails.

The company is seeking a solution that fulfills these requirements while minimizing operational complexity. The possible strategies are:

A. Create an AWS Backup Plan with Specific Backup Rules: Develop an AWS Backup plan that includes distinct backup rules catering to the various daily, weekly, and monthly retention requirements.

B. Configure AWS Backup Plan for Cross-Region Replication: Adjust the AWS Backup plan to ensure that all backups are automatically replicated to another AWS Region following their capture.

C. Develop an AWS Lambda Function for Backup Replication and Failure Notifications: Implement a custom AWS Lambda function tasked with replicating backups to another Region and dispatching notifications in case of any backup failures.

D. Integrate Amazon SNS with the Backup Plan for Failure Notifications: Add an Amazon Simple Notification Service (SNS) topic to the AWS Backup plan configured to send alerts for all completed jobs, excluding those with the status `BACKUP_JOB_COMPLETED`.

E. Implement Amazon DLM Snapshot Lifecycle Policies for Retention Requirements: Create individual snapshot lifecycle policies using Amazon Data Lifecycle Manager (DLM) for each set of retention criteria.

F. Establish RDS Snapshots for Each Database: Set up manual RDS snapshots for each database instance in line with the required backup schedules.

The most suitable combination of steps is A, B, and D:

- A. Create an AWS Backup Plan with Specific Backup Rules: This solution directly addresses the need for varied retention schedules. AWS Backup allows the creation of customized backup plans that can specify different frequencies and retention durations, aligning with the company's requirements.
- B. Configure AWS Backup Plan for Cross-Region Replication: AWS Backup offers built-in functionality for cross-region replication of backups, fulfilling the requirement to replicate backups to another region immediately after they are taken.
- D. Integrate Amazon SNS with the Backup Plan for Failure Notifications: Incorporating Amazon SNS into the AWS Backup plan provides a straightforward method for sending immediate notifications in case of backup failures, ensuring prompt awareness and response.

The reasons why the other options are less suitable:

- C. Develop an AWS Lambda Function for Backup Replication and Failure Notifications: While a Lambda function could technically achieve the desired replication and notification, it introduces unnecessary complexity and operational overhead compared to the built-in capabilities of AWS Backup.
- E. Implement Amazon DLM Snapshot Lifecycle Policies for Retention Requirements: Amazon DLM is primarily useful for managing EBS snapshots. However, it does not provide a unified backup solution across EC2, EFS, and RDS, nor does it inherently handle cross-region replication or integrated failure notifications.
- F. Establish RDS Snapshots for Each Database: Manual RDS snapshots do not offer the comprehensive, automated backup solution required. This approach would also add significant operational overhead and does not provide a unified backup status monitoring or automatic cross-region replication.

Question 38

A company is creating a gene reporting device that will send 8 KB of genomic data per second to a data platform. This platform is required to rapidly analyze this data and supply the findings back to researchers. The platform must deliver near-real-time analytics, handle data in a flexible, parallel, and durable manner, and channel the processed results to a data warehouse.

The potential strategies for setting up this data platform are:

A. Ingest Data with Amazon Kinesis Data Firehose, Analyze with Kinesis Clients, Store in Amazon RDS: Utilize Amazon Kinesis Data Firehose for data collection, perform analysis using Kinesis clients, and store the processed data in an Amazon RDS instance.

B. Collect Data with Amazon Kinesis Data Streams, Analyze with Kinesis Clients, Store in Amazon Redshift via Amazon EMR: Use Amazon Kinesis Data Streams for data ingestion, analyze the data through Kinesis clients, and save the processed results to an Amazon Redshift cluster employing Amazon EMR.

C. Use Amazon S3 for Data Ingestion, Process Data from Amazon SQS with Kinesis, Save to Amazon Redshift: Employ Amazon S3 to gather device data, analyze the data from Amazon SQS using Kinesis, and store the outcomes in an Amazon Redshift cluster.

D. Implement Amazon API Gateway and Amazon SQS for Data Ingestion, Analyze with AWS Lambda, Store in Amazon Redshift via Amazon EMR: Leverage Amazon API Gateway to feed data into an Amazon SQS queue, process the data with AWS Lambda functions, and save the analytical results to an Amazon Redshift cluster using Amazon EMR.

The best solution is B:

- **B. Collect Data with Amazon Kinesis Data Streams, Analyze with Kinesis Clients, Store in Amazon Redshift via Amazon EMR**: This option effectively meets all the specified requirements. Amazon Kinesis Data Streams is well-suited for real-time data ingestion and processing, handling high-throughput, and low-latency data streaming. Analyzing this data with Kinesis clients allows for near-real-time analytics. Using Amazon EMR to move the processed data into Amazon Redshift aligns with the need for a robust data warehouse solution, providing the necessary flexibility, parallel processing, and durability.

The reasons why the other options are less suitable:

- **A. Ingest Data with Amazon Kinesis Data Firehose, Analyze with Kinesis Clients, Store in Amazon RDS**: While Kinesis Data Firehose is suitable for data ingestion, storing the results in Amazon RDS may not be optimal for large-scale genomic data analysis. RDS is not primarily designed as a data warehouse solution like Redshift, which is better suited for such analytical workloads.
- **C. Use Amazon S3 for Data Ingestion, Process Data from Amazon SQS with Kinesis, Save to Amazon Redshift**: This approach introduces unnecessary complexity. Direct streaming of data to S3 and then processing it from SQS is less efficient compared to using Kinesis Data Streams for real-time data processing.
- **D. Implement Amazon API Gateway and Amazon SQS for Data Ingestion, Analyze with AWS Lambda, Store in Amazon Redshift via Amazon EMR**: Using API Gateway and SQS for data ingestion adds complexity and may not provide the near-real-time processing capability that Kinesis Data Streams offer. AWS Lambda can be used for data processing, but it might not be as efficient for continuous high-throughput data streams as Kinesis.

Question 39

A solutions architect is tasked with designing a reference architecture for a three-tier application (web, application, and NoSQL data layers) in AWS, focusing on high availability within an AWS Region and the capability to switch to a secondary AWS Region for disaster recovery within 1 minute. The goal is to ensure an efficient solution with minimal impact on user experience.

The options for constructing this architecture are:

A. **Implement Amazon Route 53 Weighted Routing Policy (100/0) with 1-Hour TTL**: Set up a Route 53 weighted routing policy with a 100/0 distribution across two AWS Regions and a Time to Live (TTL) setting of 1 hour.

B. **Use Amazon Route 53 Failover Routing Policy with 30-Second TTL**: Configure Route 53 with a failover routing policy that automatically switches from the primary Region to the

disaster recovery Region, setting the TTL to 30 seconds.

C. **Deploy Amazon DynamoDB Global Tables for Cross-Region Data Access**: Utilize global tables in Amazon DynamoDB to allow data access in both the primary and secondary AWS Regions.

D. **Backup DynamoDB Data to Amazon S3 with Cross-Region Replication**: Regularly back up data from the primary Region's DynamoDB table to Amazon S3 every 60 minutes, then replicate it to the disaster recovery Region using S3 cross-region replication. In a disaster recovery scenario, use a script to import the data into DynamoDB.

E. **Implement Hot Standby Model with Auto Scaling and Reserved Instances**: Create a hot standby setup using Auto Scaling groups for the web and application layers across multiple Availability Zones in both Regions. Use Reserved Instances for the baseline server count and On-Demand Instances for additional resources.

F. **Use Auto Scaling Groups with Spot Instances**: Configure Auto Scaling groups for the web and application layers across multiple Availability Zones in both Regions, employing Spot Instances for all required resources.

The best combination of steps is B, C, and E:

- **B. Use Amazon Route 53 Failover Routing Policy with 30-Second TTL**: This choice meets the requirement for a quick failover to another Region in case of a disaster. A 30-second TTL ensures that DNS changes propagate rapidly, facilitating a quick response to region-level failures.
- **C. Deploy Amazon DynamoDB Global Tables for Cross-Region Data Access**: DynamoDB Global Tables provide a fully managed, multi-region, and multi-master database solution, ensuring that data is available and synchronized across Regions. This meets the high availability and quick failover requirements.
- **E. Implement Hot Standby Model with Auto Scaling and Reserved Instances**: A hot standby model in multiple Availability Zones provides high availability within a Region. Using Reserved Instances for the minimum necessary resources ensures cost efficiency, while On-Demand Instances can handle additional demand, striking a balance between cost and scalability.

The reasons why the other options are less suitable:

- **A. Implement Amazon Route 53 Weighted Routing Policy (100/0) with 1-Hour TTL**: A weighted routing policy isn't ideal for failover scenarios, as it's designed for distributing traffic, not for handling region-level outages. Additionally, a TTL of 1 hour could delay the failover process, contradicting the requirement for a 1-minute failover.

- D. Backup DynamoDB Data to Amazon S3 with Cross-Region Replication: While backing up data is important, this method doesn't support real-time data replication or quick failover. The 60-minute backup interval could lead to data loss, not meeting the specified RPO.
- F. Use Auto Scaling Groups with Spot Instances: While Spot Instances are cost-effective, they may not provide the required reliability for a disaster recovery setup, as they can be interrupted by AWS with a two-minute notification. This could compromise the high availability and quick failover needs of the application.

Question 40

A manufacturing company specializing in smart vehicles is facing a challenge with their current on-premises storage solution. Their custom application, which collects vehicle data using the MQTT protocol, is not scaling effectively for peak traffic, leading to data loss. As the volume of data from an increasing number of vehicles grows, especially from newer models that generate more data, the company needs a more scalable solution on AWS with minimal operational overhead.

The proposed AWS solutions to address this issue are:

A. Implement AWS IoT Greengrass with Amazon MSK: Use AWS IoT Greengrass to direct vehicle data to Amazon Managed Streaming for Apache Kafka (MSK). Develop an Apache Kafka application to store data in Amazon S3 and utilize a pretrained model in Amazon SageMaker for anomaly detection.

B. Utilize AWS IoT Core with Amazon Kinesis Data Firehose and S3: Receive vehicle data through AWS IoT Core, setting up rules to route this data to an Amazon Kinesis Data Firehose delivery stream, which then stores the data in Amazon S3. Create an Amazon Kinesis Data Analytics application to analyze the data stream for anomaly detection.

C. Employ AWS IoT FleetWise and Amazon Kinesis Streams with AWS Glue: Use AWS IoT FleetWise for data collection, sending it to an Amazon Kinesis data stream. Then, use Amazon Kinesis Data Firehose to store the data in Amazon S3, applying AWS Glue's built-in machine learning transforms for anomaly detection.

D. Use Amazon MQ for RabbitMQ with Kinesis Data Firehose and Amazon Lookout for Metrics: Collect vehicle data using Amazon MQ for RabbitMQ, forward it to an Amazon Kinesis Data Firehose delivery stream for storage in Amazon S3, and apply Amazon Lookout for Metrics for anomaly detection.

The most appropriate solution is B:

- B. Utilize AWS IoT Core with Amazon Kinesis Data Firehose and S3: This option effectively meets the requirements for scalability and minimal operational overhead. AWS IoT Core is well-suited for handling MQTT protocol-based data ingestion. The integration with Amazon Kinesis Data Firehose simplifies the process of storing large volumes of data in Amazon S3. Using Amazon Kinesis Data Analytics for real-time analysis minimizes the need for extensive configuration and maintenance, thereby reducing operational overhead.

The reasons why the other options are less suitable:

- A. Implement AWS IoT Greengrass with Amazon MSK: While this setup can handle large data streams, it involves more complex configuration and management of an Apache Kafka application and Amazon SageMaker. This increases the operational overhead compared to the streamlined approach in option B.
- C. Employ AWS IoT FleetWise and Amazon Kinesis Streams with AWS Glue: AWS IoT FleetWise is specialized for vehicle data, but its combination with Kinesis Streams and AWS Glue for anomaly detection might not provide the most straightforward solution. This approach could introduce additional complexity in the data processing pipeline.
- D. Use Amazon MQ for RabbitMQ with Kinesis Data Firehose and Amazon Lookout for Metrics: Amazon MQ for RabbitMQ offers a message broker service, but it's not as directly tailored for IoT scenarios as AWS IoT Core. Additionally, using Amazon Lookout for Metrics, while useful, might not be as seamless for real-time processing and analysis as Kinesis Data Analytics.

Question 41

A security audit revealed that a development team was inadvertently committing AWS IAM user secret access keys to their AWS CodeCommit repository. To address this security issue, the security team needs an automated solution to identify and rectify instances where these credentials are exposed.

The potential automated solutions to secure these credentials are:

A. Nightly Script Execution via AWS Systems Manager: Implement a nightly script using AWS Systems Manager Run Command to search for exposed credentials on development instances. If credentials are detected, rotate them through AWS Secrets Manager.

B. Scheduled AWS Lambda Function for Code Scanning: Set up a recurring AWS Lambda function to retrieve and examine the application code from CodeCommit. Upon finding credentials, generate new ones and secure them in AWS Key Management Service (KMS).

C. Amazon Macie Scanning with Lambda for Remediation: Utilize Amazon Macie to conduct scans on CodeCommit repositories for exposed credentials. When credentials are identified,

trigger an AWS Lambda function to deactivate the credentials and alert the concerned user.

D. CodeCommit Trigger with Lambda for Real-Time Scanning: Configure a trigger in CodeCommit to activate an AWS Lambda function that scans new code submissions for credentials. In cases where credentials are found, deactivate them in AWS IAM and notify the user.

The most effective solution is D:

- D. CodeCommit Trigger with Lambda for Real-Time Scanning: This solution addresses the problem proactively by scanning for credentials in real-time as new code is committed. By using a Lambda function triggered by CodeCommit, any instance of exposed credentials can be immediately detected, and remedial actions, such as disabling the credentials in AWS IAM and notifying the user, can be undertaken. This approach provides timely and efficient remediation.

The reasons why the other options are less suitable:

- A. Nightly Script Execution via AWS Systems Manager: Running a script nightly introduces a delay in detection and remediation, leaving a window where exposed credentials remain accessible. This approach is reactive and less timely compared to real-time scanning.
- B. Scheduled AWS Lambda Function for Code Scanning: Similar to option A, this method relies on a scheduled, rather than real-time, approach. While it automates the scanning process, it does not provide immediate detection and response.
- C. Amazon Macie Scanning with Lambda for Remediation: Although Amazon Macie is effective for data classification and detecting sensitive information, it is not specifically optimized for real-time scanning of code repositories like CodeCommit. Additionally, Macie's primary use case is for S3 data, not code repositories.

Question 42

A company is utilizing an Amazon S3-based data lake that needs to be securely accessed by a multitude of applications across various AWS accounts. Adhering to their information security policy, the company requires that the S3 bucket not be accessible via the public internet and that each application is granted only the permissions necessary for its operations. The solutions architect is considering the use of S3 access points, each restricted to specific Virtual Private Clouds (VPCs) for individual applications, to fulfill these requirements.

The steps the solutions architect should consider for implementing this solution are:

A. Set Up S3 Access Points for Each Application in the Owning Account: Create an S3 access point for each application within the account that owns the S3 bucket. Configure these access points to be accessible solely from the respective application's VPC and modify the bucket policy to mandate access via an access point.

B. Establish Interface Endpoints for S3 in Application VPCs: Create an interface endpoint for Amazon S3 within each application's VPC. Adjust the endpoint policy to allow access to the S3 access point and set up a VPC gateway attachment for the S3 endpoint.

C. Create Gateway Endpoints for S3 in Application VPCs: Implement a gateway endpoint for Amazon S3 in each application's VPC. Tailor the endpoint policy to permit access to an S3 access point and define the route table used for accessing the access point.

D. Generate S3 Access Points in Each Application's AWS Account: Create an S3 access point for every application in their respective AWS accounts and link these access points to the central S3 bucket. Ensure each access point is only reachable from the application's VPC and revise the bucket policy to necessitate access through an access point.

E. Configure a Gateway Endpoint for S3 in the Data Lake's VPC: Set up a gateway endpoint for Amazon S3 in the data lake's VPC. Attach an endpoint policy that authorizes access to the S3 bucket and specify the route table for accessing the bucket.

The most suitable combination of steps is A and C:

- A. Set Up S3 Access Points for Each Application in the Owning Account: This approach aligns with the need for fine-grained access control and network isolation. By creating individual access points for each application within the owning AWS account, the solution ensures that each application has the minimal necessary permissions. Restricting these access points to specific VPCs also adheres to the policy of not accessing the S3 bucket over the public internet.
- C. Create Gateway Endpoints for S3 in Application VPCs: Gateway endpoints provide a secure, private link between a VPC and AWS services like S3. By setting up these endpoints in each application's VPC and configuring them to allow access to the S3 access points, the solution enforces private access to the data lake while complying with the security requirements.

The reasons why the other options are less suitable:

- B. Establish Interface Endpoints for S3 in Application VPCs: Interface endpoints (also known as PrivateLink) are generally more complex and costly compared to gateway endpoints. They also require additional network routing configurations, increasing operational overhead.

- D. Generate S3 Access Points in Each Application's AWS Account: Creating access points in each application's account might lead to administrative complexity and does not provide the same level of centralized control as creating them in the owning account.
- E. Configure a Gateway Endpoint for S3 in the Data Lake's VPC: Placing a gateway endpoint only in the data lake's VPC does not address the need for controlled access from each application's VPC. This setup would not enforce the application-level restrictions as effectively as having endpoints in each application's VPC.

Question 43

A company with a hybrid cloud setup involving both a data center and AWS is looking to enhance its monitoring capabilities. They use Amazon EC2 instances, which interact with on-premises relational databases, and these interactions are logged in Amazon CloudWatch. The company already utilizes Splunk for on-premises monitoring and now wants to integrate CloudWatch logs, specifically to track EC2 instance connections to their databases in near-real time.

The solutions architect must determine the best method to route network traffic logs to Splunk. The proposed solutions are:

A. Export CloudWatch Logs to S3 for Splunk Retrieval: Activate VPC flow logs, directing them to CloudWatch. Use AWS Lambda to periodically transfer CloudWatch logs to an Amazon S3 bucket. Provide Splunk with AWS credentials to fetch logs from the S3 bucket.

B. Stream Logs to Splunk via Amazon Kinesis Data Firehose: Set up an Amazon Kinesis Data Firehose delivery stream with Splunk as the endpoint. Configure a Lambda function to process log data sent from CloudWatch Logs subscription filters. Enable VPC flow logs to CloudWatch and link them to the Kinesis Data Firehose stream.

C. Database Request Logging with Athena Analysis and Lambda to Splunk: Instruct the company to log each database request, including the EC2 instance IP, in CloudWatch. Export these logs to S3 and use Amazon Athena for query analysis. Route Athena results to another S3 bucket and employ Lambda to send updates to Splunk.

D. Analyze CloudWatch Logs with Kinesis Data Analytics and Stream to Splunk: Direct CloudWatch logs to an Amazon Kinesis data stream and utilize Amazon Kinesis Data Analytics for real-time SQL processing. Set a 1-minute window for event collection and apply a SQL query for anomaly detection. Forward results to a Kinesis Data Firehose delivery stream linked to Splunk.

The most suitable solution is B:

- B. Stream Logs to Splunk via Amazon Kinesis Data Firehose: This option efficiently integrates with Splunk to provide real-time monitoring capabilities. By enabling VPC flow logs to CloudWatch and then using a CloudWatch Logs subscription to send data to a Kinesis Data Firehose delivery stream, the solution ensures continuous and automated log data transfer to Splunk. The pre-processing Lambda function can tailor the data before it reaches Splunk, making this a streamlined and effective approach.

The reasons why the other options are less suitable:

- A. Export CloudWatch Logs to S3 for Splunk Retrieval: This method introduces delays due to the periodic export process, hindering the goal of near-real-time monitoring. It also adds complexity in managing credentials and configuring Splunk to pull data from S3.
- C. Database Request Logging with Athena Analysis and Lambda to Splunk: This approach is overly complex, involving multiple steps and services (Athena, S3, Lambda). It is not as efficient for real-time monitoring due to the processing and exporting steps involved.
- D. Analyze CloudWatch Logs with Kinesis Data Analytics and Stream to Splunk: While this method provides real-time analysis, it focuses more on anomaly detection rather than straightforward log transfer. The additional analytics layer might not be necessary for simply monitoring EC2 connections to databases.

Question 44

A company overseeing five development teams, each with their own AWS accounts (totaling 25 accounts), faces challenges in efficiently tracking spending and ensuring compliance with a policy that restricts resource creation to U.S. AWS Regions only. Currently, the finance team manually collects monthly spending data from each account, and non-compliant resources have been found in non-U.S. Regions. The company needs a solution to streamline expenditure tracking across all accounts and enforce the regional compliance policy.

The potential solutions for meeting these requirements are (choose three):

A. Implement AWS Cost and Usage Reports with S3 for Finance Team: Set up an AWS Cost and Usage Reports system, storing data in an Amazon S3 bucket for the finance team. Use AWS Systems Manager Run Command to periodically export CloudWatch logs to this S3 bucket.

B. Create a Management Account and AWS Organization: Establish a new management account and form an organization using AWS Organizations with all features enabled. Invite all existing accounts to join the organization.

C. Form an OU with U.S.-only Resource Creation SCP: Create an Organizational Unit (OU) encompassing all development teams. Implement an SCP that allows resource creation only in U.S. Regions and apply it to the OU.

D. Use an SCP to Restrict Non-U.S. Resource Creation: Set up an OU for the development teams and create an SCP that explicitly denies resource creation in non-U.S. Regions. Apply this SCP to the OU.

E. IAM Role in Management Account for Finance Team: Create an IAM role in the management account with permissions to view the Billing and Cost Management console. Allow the finance team to assume this role for cost analysis using AWS Cost Explorer and the Billing console.

F. Create IAM Role in Each AWS Account for Finance Team: Establish an IAM role in each of the AWS accounts. Attach a policy that includes permissions to view the Billing and Cost Management console and allow the finance team to assume these roles.

The most effective combination of steps is B, D, and E:

- B. Create a Management Account and AWS Organization: This step centralizes the management of all AWS accounts, allowing for streamlined tracking and consolidated billing under a single organization. It facilitates easier application of policies and monitoring across all accounts.
- D. Use an SCP to Restrict Non-U.S. Resource Creation: Applying an SCP at the organizational level that denies resource creation in non-U.S. Regions ensures compliance with the company's policy. This step effectively prevents the creation of resources in unwanted Regions across all accounts in the organization.
- E. IAM Role in Management Account for Finance Team: Creating an IAM role specifically for the finance team in the management account allows centralized access to billing information across the organization. This approach is more efficient than accessing each account individually and provides a single source for cost tracking.

The reasons why the other options are less suitable:

- A. Implement AWS Cost and Usage Reports with S3 for Finance Team: While AWS Cost and Usage Reports are valuable, this approach does not address the need for centralized management and compliance enforcement. It also adds operational complexity in managing and exporting logs.
- C. Form an OU with U.S.-only Resource Creation SCP: This option, focusing on allowing rather than denying, could potentially leave loopholes for non-compliant resource creation in other Regions. Explicit deny rules (as in option D) are typically more effective for strict compliance.

- F. Create IAM Role in Each AWS Account for Finance Team: This approach is operationally inefficient as it requires managing roles across multiple accounts, complicating the process of consolidated cost tracking and analysis.

Question 45

A company employing AWS Organizations needs a centralized solution to manage multiple AWS accounts across various departments. Additionally, their security team, operating from a dedicated AWS account, requires read-only access to all these accounts. The goal is to configure this access in a way that aligns with AWS best practices and organizational requirements.

The strategies for granting the security team the necessary access are:

A. Modify OrganizationAccountAccessRole IAM Policy in Each Account: Adapt the `OrganizationAccountAccessRole` IAM role in every member account to include a new IAM policy with read-only permissions. Set up a trust relationship between this policy in each member account and the security team's account, enabling the team to access these policies.

B. Create New IAM Role with Read-Only Access in Each Account: Utilize the `OrganizationAccountAccessRole` IAM role to create a new IAM role with read-only permissions in each member account. Form a trust relationship between this new IAM role in each account and the security team's account, allowing the security team to use this role for access.

C. Use AWS STS for Management Account Role Assumption from Security Account: Instruct the security team to use AWS Security Token Service (STS) to call the `AssumeRole` API for the `OrganizationAccountAccessRole` IAM role in the management account. They should then use the temporary credentials generated to access the accounts.

D. Use AWS STS for Member Account Role Assumption from Security Account: Advise the security team to employ AWS STS to invoke the `AssumeRole` API for the `OrganizationAccountAccessRole` IAM role in each member account, using the temporary credentials generated for access.

The most appropriate solution is B:

- B. Create New IAM Role with Read-Only Access in Each Account: This approach is the most effective and aligns with AWS best practices. By creating a new IAM role with read-only permissions in each member account and establishing a trust relationship with the security team's account, the security team can assume these roles to get the necessary

access. This method ensures that the security team has the right level of access across all accounts while maintaining account-level separation and least privilege.

The reasons why the other options are less suitable:

- A. Modify OrganizationAccountAccessRole IAM Policy in Each Account: Modifying the `OrganizationAccountAccessRole` directly is not advisable. This role is automatically created when an account joins an AWS Organization and is intended for use by the organization's management account. Overriding its permissions can lead to security risks.
- C. Use AWS STS for Management Account Role Assumption from Security Account: Assuming the role from the management account does not inherently grant access to all member accounts. This approach would not satisfy the requirement to access each individual member account.
- D. Use AWS STS for Member Account Role Assumption from Security Account: While assuming a role in member accounts using STS is a valid approach, this option suggests using the default `OrganizationAccountAccessRole`, which may not be configured for read-only access and could introduce security concerns. Creating specific roles for this purpose, as in option B, is more secure and controlled.

Question 46

A large company operating across hundreds of AWS accounts has workloads running in VPCs with both public and private subnets spread over multiple Availability Zones. In each VPC, NAT gateways situated in the public subnets facilitate outbound internet traffic from the private subnets. The company's solutions architect is tasked with implementing a hub-and-spoke network architecture, where all internet traffic from the private subnets in the spoke VPCs is directed through a central egress VPC. This egress VPC, located in a central AWS account, already contains a deployed NAT gateway.

The solutions architect needs to determine the next steps to route traffic from the spoke VPCs through the egress VPC. The proposed methods are:

A. Establish VPC Peering Connections: Create peering connections between the egress VPC and each spoke VPC, configuring the necessary routing for internet access.

B. Use a Transit Gateway and Share it Across Accounts: Deploy a transit gateway and share it with all existing AWS accounts. Attach the existing VPCs to this transit gateway and set up the required routing for internet access.

C. Create Transit Gateways in Each Account and Attach NAT Gateways: Implement a transit gateway in every AWS account and connect the NAT gateways to these transit gateways, configuring routing for internet access.

D. Set Up AWS PrivateLink for VPC Connectivity: Create AWS PrivateLink connections between the egress VPC and the spoke VPCs, and configure the necessary routing for internet access.

The most appropriate solution is B:

- B. Use a Transit Gateway and Share it Across Accounts: This approach offers an efficient and scalable way to manage cross-account network connectivity. By setting up a single transit gateway in the central account and sharing it across all AWS accounts, the solutions architect can centralize the network routing through the egress VPC's NAT gateway. This setup simplifies the network architecture and centralizes management, making it an operationally efficient solution for the hub-and-spoke design.

The reasons why the other options are less suitable:

- A. Establish VPC Peering Connections: While VPC peering can connect individual VPCs, it doesn't scale well for hundreds of accounts due to peering limitations and the complexity of managing multiple peer connections. It also doesn't support transitive routing, necessary for a hub-and-spoke model.
- C. Create Transit Gateways in Each Account and Attach NAT Gateways: Creating a transit gateway in every account is operationally inefficient and unnecessary. This approach would complicate the network structure and management, contrary to the goal of centralizing and simplifying the network architecture.
- D. Set Up AWS PrivateLink for VPC Connectivity: AWS PrivateLink is primarily used for secure, private connectivity between AWS services, VPCs, and on-premises applications. It is not suitable for routing all internet traffic through a central egress VPC, as it's not designed for such a use case.

Question 47

An education company operates a web application for college students globally, hosted on Amazon Elastic Container Service (ECS) with an Auto Scaling group and an Application Load Balancer (ALB). The application's authentication service is experiencing a recurring issue with a spike in failed login attempts every week. These attempts, originating from approximately 500 different IP addresses that change weekly, are overwhelming the service. The solutions architect needs to devise a strategy to effectively mitigate these attacks without adding significant operational overhead.

The proposed solutions for addressing this issue are:

A. Implement AWS Firewall Manager for Security Group Management: Set up AWS Firewall Manager to create and manage a security group policy that denies access from the identified IP addresses.

B. Use AWS WAF with Rate-Based Rule on ALB: Configure an AWS WAF web access control list (ACL) with a rate-based rule, setting the rule action to block excessive requests. Associate this web ACL with the application's ALB.

C. Manage Access with AWS Firewall Manager and CIDR Restrictions: Utilize AWS Firewall Manager to create a security group policy that restricts access exclusively to certain CIDR ranges.

D. Deploy AWS WAF with IP Set Match Rule on ALB: Establish an AWS WAF web ACL with an IP set match rule to block traffic from the specified IP addresses, and link this web ACL to the ALB.

The best solution is B:

- B. Use AWS WAF with Rate-Based Rule on ALB: This approach is operationally efficient and directly addresses the challenge of overwhelming login attempts. By implementing a rate-based rule in AWS WAF, the solution can automatically block IPs that exceed a defined threshold of requests, which is effective against the varying IPs involved in the failed login attempts. Associating this with the ALB ensures that the blocking is applied before the traffic reaches the ECS application, thereby reducing the load on the authentication service.

The reasons why the other options are less suitable:

- A. Implement AWS Firewall Manager for Security Group Management: While AWS Firewall Manager is useful for central management of security groups across multiple AWS accounts, it's less effective for dynamic threat scenarios like this, where the offending IP addresses change frequently.
- C. Manage Access with AWS Firewall Manager and CIDR Restrictions: Restricting access to specific CIDR ranges might be too restrictive for a global application like this, potentially blocking legitimate users and not being agile enough to adapt to the changing IP addresses of the attackers.
- D. Deploy AWS WAF with IP Set Match Rule on ALB: Using an IP set match rule in AWS WAF would require constant updates to the IP list, which is not operationally efficient given the weekly change in offending IPs. This approach would also not adapt quickly to new threats.

Question 48

A company currently running an on-premises Software-as-a-Service (SaaS) solution, which processes multiple files daily, is planning to migrate to AWS. The current setup involves providing multiple public SFTP endpoints to customers, who then add these endpoint IP addresses to their firewalls for outbound traffic. It is crucial that the IP addresses of these

SFTP endpoints remain unchanged. The company aims to reduce the operational burden associated with managing this file transfer service as part of their migration to AWS.

The options for migrating this service to AWS while adhering to the specified requirements are:

A. Use AWS Transfer for SFTP with Elastic IP Addresses: Register the customer-owned IP block with the company's AWS account. Create Elastic IP addresses from this pool and assign them to an AWS Transfer for SFTP endpoint. Configure AWS Transfer to store the files in Amazon S3.

B. Deploy EC2 Instances with Elastic IPs Behind ALB for FTP Services: Incorporate a subnet with the customer-owned IP block into a VPC. Create Elastic IP addresses from this block and assign them to an Application Load Balancer (ALB). Use EC2 instances running FTP services behind the ALB, with file storage on Amazon Elastic Block Store (EBS) volumes.

C. Integrate Customer IPs with Route 53 and NLB for FTP Services: Register the customer-owned IP block with Amazon Route 53. Create alias records in Route 53 pointing to a Network Load Balancer (NLB). Set up EC2 instances hosting FTP services in an Auto Scaling group behind the NLB, storing files in Amazon S3.

D. Assign Elastic IPs from Customer Block to S3 VPC Endpoint: Register the customer-owned IP block in the company’s AWS account. Create Elastic IP addresses from this pool and assign them to an Amazon S3 VPC endpoint, enabling SFTP support on the S3 bucket.

The most suitable solution is A:

- A. Use AWS Transfer for SFTP with Elastic IP Addresses: This approach aligns well with the requirement to maintain the same IP addresses for SFTP endpoints. By registering the customer-owned IP block in AWS and using Elastic IP addresses for the AWS Transfer for SFTP service, the company can ensure IP address consistency. AWS Transfer for SFTP provides a managed service that reduces operational overhead and seamlessly integrates with Amazon S3 for file storage.

The reasons why the other options are less suitable:

- B. Deploy EC2 Instances with Elastic IPs Behind ALB for FTP Services: This setup involves significant operational overhead in managing EC2 instances and the ALB for FTP services, which contradicts the goal of decreasing operational burden. Additionally, this approach does not leverage AWS's managed SFTP service.
- C. Integrate Customer IPs with Route 53 and NLB for FTP Services: While this method can route traffic using the customer's IP addresses, it requires managing EC2 instances

and an NLB, increasing operational complexity. It also does not utilize AWS's managed SFTP service.

- D. Assign Elastic IPs from Customer Block to S3 VPC Endpoint: Assigning Elastic IP addresses to an S3 VPC endpoint does not provide an SFTP interface for file transfers. This option does not address the fundamental requirement of maintaining SFTP endpoints.

Question 49

A company has developed a REST API hosted on Amazon API Gateway to facilitate data exchange with its six U.S.-based partners. The API, intended for daily data submission of sales figures by these partners, is accessed through a Regional endpoint on API Gateway. Post-deployment, the company noticed an unusually high volume of traffic: 1,000 requests per second from 500 distinct global IP addresses, suggesting a botnet attack. The company now seeks a cost-effective method to secure its API against this unauthorized access.

The options for securing the API are:

A. Implement CloudFront with AWS WAF and Origin Access Identity (OAI): Set up an Amazon CloudFront distribution with the API as the origin. Use AWS WAF with a web ACL to block clients exceeding five requests per day. Associate this web ACL with the CloudFront distribution. Configure CloudFront with an OAI and restrict API Gateway to accept POST requests only from the OAI.

B. Use CloudFront with AWS WAF and API Key in Custom Header: Create a CloudFront distribution for the API and apply AWS WAF with a web ACL to limit clients to five requests per day. Link this web ACL to CloudFront. Add a custom header in CloudFront with an API key and configure the API to require this API key for POST requests.

C. AWS WAF with Allowlist IPs and API Gateway Resource Policy: Create an AWS WAF web ACL with an allowlist for the partners' IP addresses and associate it with the API. Implement a resource policy on API Gateway with a request limit and require an API key for POST requests.

D. AWS WAF with Allowlist IPs, Usage Plan, and API Key: Establish an AWS WAF web ACL that permits access only from the partners' IP addresses and link it to the API. Create a usage plan with a request limit in API Gateway, generate an API key, and add it to the usage plan.

The most suitable solution is D:

- D. AWS WAF with Allowlist IPs, Usage Plan, and API Key: This approach effectively secures the API by combining several layers of protection. AWS WAF is used to restrict access to known partner IPs, significantly reducing the risk of botnet attacks. The usage plan in API Gateway allows for the management of request limits, ensuring that each

partner can only submit their data once per day as intended. The API key adds an additional layer of security, ensuring that only authorized users can access the API.

The reasons why the other options are less suitable:

- A. Implement CloudFront with AWS WAF and OAI: While using CloudFront and AWS WAF provides a level of security, the implementation of OAI to restrict POST methods specifically to CloudFront complicates the setup and may not be necessary. This option also doesn't directly address the requirement to limit access to the six partners.
- B. Use CloudFront with AWS WAF and API Key in Custom Header: This option, like option A, introduces unnecessary complexity with CloudFront and doesn't focus on restricting access to the specific IP addresses of the partners.
- C. AWS WAF with Allowlist IPs and API Gateway Resource Policy: While the allowlist in AWS WAF is effective, using a resource policy with a request limit on API Gateway is less straightforward and efficient compared to using a usage plan with an API key.

Question 50

A company is deploying a new application that will be hosted on five Amazon EC2 instances within a single AWS Region. The primary requirement for this application is to ensure high-throughput and low-latency networking among the EC2 instances. Fault tolerance is not a concern for this particular application.

The options for setting up the EC2 instances to meet these network performance requirements are:

A. Use Cluster Placement Group for EC2 Instances: Deploy the five EC2 instances within a cluster placement group, ensuring that the selected EC2 instance types support enhanced networking.

B. Set Up EC2 Instances in Auto Scaling Group with Additional Network Interfaces: Launch the five EC2 instances in an Auto Scaling group within the same Availability Zone, adding an extra elastic network interface to each instance.

C. Deploy EC2 Instances in Partition Placement Group: Launch the five EC2 instances in a partition placement group, selecting EC2 instance types that are compatible with enhanced networking.

D. Place EC2 Instances in Spread Placement Group with Additional Network Interfaces: Set up the five EC2 instances in a spread placement group, equipping each instance with an extra elastic network interface.

The best solution is A:

- A. Use Cluster Placement Group for EC2 Instances: Cluster placement groups are specifically designed for applications that require low-latency, high-throughput networking between instances. Placing instances in a cluster placement group ensures that they are physically located close to each other within the same Availability Zone, optimizing for network performance. Selecting instance types that support enhanced networking further enhances this capability, making it the ideal choice for the application's requirements.

The reasons why the other options are less suitable:

- B. Set Up EC2 Instances in Auto Scaling Group with Additional Network Interfaces: While placing instances in the same Availability Zone can reduce latency, an Auto Scaling group does not inherently optimize for network performance like a cluster placement group. Additionally, adding extra network interfaces does not directly contribute to low-latency, high-throughput networking.
- C. Deploy EC2 Instances in Partition Placement Group: Partition placement groups are used to spread instances across different partitions within an Availability Zone, providing isolation and fault tolerance. This approach does not specifically focus on optimizing network performance, which is the primary requirement in this scenario.
- D. Place EC2 Instances in Spread Placement Group with Additional Network Interfaces: Spread placement groups are intended to reduce correlated failures by spreading instances across different hardware. This strategy does not specifically address the high-performance network requirement and is more suited for fault tolerance purposes.

Question 51

A company utilizing an Amazon Aurora PostgreSQL DB cluster for their applications within a single AWS Region needs to implement a solution to monitor all data activities across its databases.

The options for setting up this monitoring are:

A. Use AWS DMS with CDC to Amazon Kinesis Data Firehose and Amazon OpenSearch Service: Configure an AWS Database Migration Service (DMS) change data capture (CDC) task with the Aurora DB cluster as the source and Amazon Kinesis Data Firehose as the target. Direct Kinesis Data Firehose to feed the captured data into an Amazon OpenSearch Service cluster for analysis.

B. Capture Activity Stream to Amazon EventBridge and Process with AWS Lambda: Activate a database activity stream on the Aurora DB cluster to capture the activity stream in Amazon EventBridge. Set up an AWS Lambda function as the target for EventBridge, which

will decrypt EventBridge messages and publish all database activities to Amazon S3 for analysis.

C. Stream Database Activity to Amazon Kinesis and Deliver to Amazon S3 via Kinesis Data Firehose: Initiate a database activity stream on the Aurora DB cluster to send the activity stream to an Amazon Kinesis data stream. Configure Amazon Kinesis Data Firehose to ingest this data stream and subsequently deliver the data to Amazon S3 for analysis.

D. Set Up AWS DMS with CDC to Amazon Kinesis Data Firehose and Amazon Redshift: Establish an AWS DMS change data capture (CDC) task with the Aurora DB cluster as the source and Kinesis Data Firehose as the target. Arrange for Kinesis Data Firehose to upload the data to an Amazon Redshift cluster, where queries can be executed to analyze database activities on Aurora.

The most appropriate solution is C:

- C. Stream Database Activity to Amazon Kinesis and Deliver to Amazon S3 via Kinesis Data Firehose: This option effectively captures all database activities from the Aurora DB cluster using a database activity stream and streams this data to an Amazon Kinesis data stream. Leveraging Amazon Kinesis Data Firehose to then move the data to Amazon S3 for further analysis is operationally efficient and aligns well with the requirement to monitor all data activity on the databases. This setup provides a robust and scalable solution for comprehensive database activity monitoring.

The reasons why the other options are less suitable:

- A. Use AWS DMS with CDC to Amazon Kinesis Data Firehose and Amazon OpenSearch Service: While AWS DMS and CDC can capture changes, this approach is more suited for data migration and replication tasks rather than for real-time activity monitoring and analysis.
- B. Capture Activity Stream to Amazon EventBridge and Process with AWS Lambda: Capturing the activity stream in EventBridge and processing it with Lambda introduces additional complexity and may not provide the streamlined, real-time monitoring and analysis that the database team requires.
- D. Set Up AWS DMS with CDC to Amazon Kinesis Data Firehose and Amazon Redshift: Similar to option A, using AWS DMS with CDC is more aligned with data migration and replication. Additionally, directing data to Amazon Redshift for query analysis adds operational complexity and may not be the most efficient way to monitor database activities.

Question 52

An entertainment company launched a new game and initially set up 12 r6g.16xlarge (memory-optimized) Amazon EC2 instances behind a Network Load Balancer to manage anticipated traffic. However, CloudWatch metrics indicated that only about a quarter of the expected CPU and memory resources were used during the launch. As the game's initial demand decreased and became more variable, the company plans to implement an Auto Scaling group to dynamically adjust the number of instances based on CPU and memory usage, aiming for cost-effectiveness.

The options for configuring the Auto Scaling group are:

A. **Deploy c6g.4xlarge (Compute Optimized) Instances**: Set up the Auto Scaling group with c6g.4xlarge instances, maintaining a minimum and desired capacity of 3 instances and a maximum of 12.

B. **Use m6g.4xlarge (General Purpose) Instances**: Configure the Auto Scaling group with m6g.4xlarge instances, setting the minimum and desired capacity at 3 and the maximum at 12.

C. **Opt for r6g.4xlarge (Memory Optimized) Instances**: Arrange the Auto Scaling group to use r6g.4xlarge instances, with a minimum and desired capacity of 3, and a maximum capacity of 12.

D. **Choose r6g.8xlarge (Memory Optimized) Instances**: Configure the Auto Scaling group with r6g.8xlarge instances, and set a minimum and desired capacity of 2, and a maximum capacity of 6.

The most suitable solution is C:

- **C. Opt for r6g.4xlarge (Memory Optimized) Instances**: This choice is appropriate because the original setup used memory-optimized instances, indicating that memory optimization is likely important for the application. By selecting a smaller size of the same instance type (r6g.4xlarge), the company can maintain the memory-optimized characteristics while scaling down to match the lower resource utilization observed. The specified minimum, desired, and maximum capacities allow for both scaling up and down as needed, enhancing cost efficiency.

The reasons why the other options are less suitable:

- **A. Deploy c6g.4xlarge (Compute Optimized) Instances**: While this option provides compute optimization, it may not be ideal if the application's primary requirement is memory optimization. Changing the instance type to compute-optimized could impact performance if the game is more memory-intensive.

- B. Use m6g.4xlarge (General Purpose) Instances: General-purpose instances offer a balance of compute, memory, and networking resources. However, if the original deployment specifically chose memory-optimized instances, this shift might not align well with the application's requirements.
- D. Choose r6g.8xlarge (Memory Optimized) Instances: This option still utilizes memory-optimized instances but does not scale down as effectively as option C. The larger instance size (r6g.8xlarge) may result in underutilization of resources, similar to the initial deployment.

Question 53

A financial services company has stored millions of historical stock trade records in an Amazon DynamoDB table configured with on-demand capacity mode. The table experiences a daily influx of a few million new records at midnight and sees bursty read activity throughout the day, focusing on a limited set of frequently accessed keys. The company is looking for strategies to lower their DynamoDB-related expenses.

The proposed solutions for cost reduction are:

A. Implement Amazon ElastiCache in Front of DynamoDB: Introduce an Amazon ElastiCache cluster to cache frequently accessed data from the DynamoDB table.

B. Utilize DynamoDB Accelerator (DAX) and Auto Scaling with Savings Plans: Set up DynamoDB Accelerator (DAX) for caching, enable DynamoDB auto-scaling, and invest in Savings Plans through AWS Cost Explorer.

C. Switch to Provisioned Capacity Mode with Savings Plans: Transition the DynamoDB table to provisioned capacity mode and purchase Savings Plans in AWS Cost Explorer.

D. Combine DAX, Provisioned Capacity Mode, and Auto Scaling: Implement DynamoDB Accelerator (DAX), switch to provisioned capacity mode for the table, and configure DynamoDB auto-scaling.

The most suitable solution is D:

- D. Combine DAX, Provisioned Capacity Mode, and Auto Scaling: This option offers a comprehensive approach to reducing costs. Deploying DAX provides an in-memory cache to handle the frequent read requests for a limited set of keys, significantly reducing read throughput demand on the DynamoDB table. Switching to provisioned capacity mode and enabling auto-scaling allows the company to manage capacity efficiently based on actual usage, potentially lowering costs compared to on-demand capacity mode. This tailored approach aligns well with the usage pattern of bursty reads and daily bulk writes.

The reasons why the other options are less suitable:

- A. Implement Amazon ElastiCache in Front of DynamoDB: While ElastiCache can cache frequent read requests, it introduces additional complexity and cost. DAX is a more integrated solution for DynamoDB, offering caching benefits with less overhead.
- B. Utilize DynamoDB Accelerator (DAX) and Auto Scaling with Savings Plans: This option includes DAX and Savings Plans but retains the on-demand capacity mode. Without switching to provisioned capacity, the company may miss out on potential cost savings that can be achieved by effectively managing capacity based on predictable workload patterns.
- C. Switch to Provisioned Capacity Mode with Savings Plans: Although switching to provisioned capacity mode and purchasing Savings Plans can reduce costs, this option does not address the high read demand efficiency. Without a caching solution like DAX, the company may still incur higher costs due to frequent read operations.

Question 54

A company is developing a centralized logging system on Amazon EC2, designed to gather and analyze logs from numerous AWS accounts. To facilitate secure connectivity between client services in these accounts and the centralized logging service, AWS PrivateLink is utilized. Each client AWS account has established an interface endpoint for the logging service. The logging service itself is hosted on EC2 instances, distributed across various subnets, and connected to a Network Load Balancer (NLB). Despite this setup, clients are currently unable to send logs to the centralized service via the VPC endpoint.

To address this connectivity issue, the solutions architect should consider the following steps:

A. Verify Network ACLs (NACLs) for Logging Service and NLB Subnets: Ensure the NACL associated with the logging service's subnets permits communication to and from the NLB subnets. Similarly, check that the NACL for the NLB subnets allows traffic to and from the subnets where the logging service's EC2 instances are running.

C. Inspect Logging Service EC2 Instances' Security Group: Confirm that the security group assigned to the EC2 instances running the logging service is configured to allow inbound traffic from the NLB's subnets.

The most effective combination of steps is A and C:

- A. Verify Network ACLs (NACLs) for Logging Service and NLB Subnets: Ensuring proper NACL configuration is crucial for facilitating traffic flow between the logging service's EC2 instances and the NLB. NACLs act as a firewall for controlling traffic in and

out of subnets, so it's important to verify that they are configured to allow the necessary communications for the logging service to function correctly.

- C. Inspect Logging Service EC2 Instances' Security Group: Security groups act as virtual firewalls for EC2 instances. It's important that the security group associated with the logging service's EC2 instances allows ingress from the NLB's subnets. This ensures that the logging data forwarded by the NLB can reach the EC2 instances.

The reasons why the other options are less suitable:

- B. Check NACLs for Logging Service and Interface Endpoint Subnets: While checking NACLs is important, the focus should be on the interaction between the logging service subnets and the NLB subnets, rather than the interface endpoint subnets, to resolve the connectivity issue.
- D. Check Security Group for EC2 Instances for Ingress from Clients: Modifying the security group to allow direct ingress from clients is not necessary in this context, as the communication is managed through the NLB and AWS PrivateLink.
- E. Check Security Group for NLB for Ingress from Interface Endpoint Subnets: NLBs do not have associated security groups. Traffic rules for NLBs are managed through the security groups attached to the target EC2 instances, making this step irrelevant.

Question 55

A company has a substantial number of objects stored in an Amazon S3 bucket using the S3 Standard storage class. These objects are frequently accessed by a rapidly growing number of users and applications. Currently, the objects are encrypted using server-side encryption with AWS Key Management Service (KMS) keys (SSE-KMS). The company's recent AWS invoices show a significant increase in AWS KMS costs attributed to the high volume of requests from Amazon S3. The solutions architect is tasked with optimizing these costs while ensuring minimal application changes.

The proposed solutions to address this are:

A. Migrate to SSE-C Encryption: Create a new S3 bucket with server-side encryption using customer-provided keys (SSE-C). Copy the existing objects into this bucket, specifying SSE-C for encryption.

B. Switch to SSE-S3 Encryption: Establish a new S3 bucket with server-side encryption using Amazon S3 managed keys (SSE-S3). Utilize S3 Batch Operations to copy the current objects to the new bucket, applying SSE-S3 encryption.

C. Adopt AWS CloudHSM for Key Management: Implement AWS CloudHSM for key storage. Create a new S3 bucket and use S3 Batch Operations to transfer existing objects to this bucket, encrypting them with keys from CloudHSM.

D. Use S3 Intelligent-Tiering with Glacier Deep Archive: Convert the current S3 bucket to use the S3 Intelligent-Tiering storage class. Set up an Intelligent-Tiering archive configuration to transition objects not accessed for 90 days to S3 Glacier Deep Archive.

The most effective solution is B:

- B. Switch to SSE-S3 Encryption: This approach significantly reduces costs associated with AWS KMS by using Amazon S3 managed keys (SSE-S3) for server-side encryption. This change eliminates the KMS request charges while maintaining the same level of security for stored objects. The use of S3 Batch Operations facilitates an efficient and streamlined migration process from the existing bucket to the new one.

The reasons why the other options are less suitable:

- A. Migrate to SSE-C Encryption: While using SSE-C removes the dependency on AWS KMS, it introduces operational overhead in managing and providing encryption keys for each request. This change would require modifications to applications accessing the S3 bucket, which goes against the requirement for minimal application changes.
- C. Adopt AWS CloudHSM for Key Management: Transitioning to AWS CloudHSM for key management may reduce KMS costs, but it introduces complexity and operational overhead in managing CloudHSM. This approach also entails significant changes to the application, which is not desired.
- D. Use S3 Intelligent-Tiering with Glacier Deep Archive: Changing to S3 Intelligent-Tiering and using Glacier Deep Archive addresses storage costs but does not impact the costs associated with KMS for encryption. This solution doesn't address the main concern of reducing KMS-related expenses.

Question 56

A media storage application, which enables users to upload photos to Amazon S3 for processing by AWS Lambda, is encountering issues. Users have reported that some photos are not processed correctly. Investigations by the application developers reveal that problems arise when a large number of users upload photos simultaneously. These challenges are attributed to Lambda's concurrency limitations and the performance constraints of Amazon DynamoDB during data write operations.

To enhance the performance and reliability of the application, the following actions are proposed:

A. Adjust Read Capacity Units (RCUs) of DynamoDB Tables: Review and modify the DynamoDB tables' RCUs to improve the read performance.

B. Adjust Write Capacity Units (WCUs) of DynamoDB Tables: Evaluate and increase the DynamoDB tables' WCUs to enhance write performance.

C. Implement Amazon ElastiCache for Lambda Performance: Add an Amazon ElastiCache layer to boost the performance of Lambda functions.

D. Integrate Amazon SQS for Queuing and Reprocessing: Insert an Amazon Simple Queue Service (Amazon SQS) queue with reprocessing logic between Amazon S3 and the Lambda functions.

E. Utilize S3 Transfer Acceleration for Faster Uploads: Implement S3 Transfer Acceleration to reduce upload latency for users.

The most appropriate combination of actions is B and D:

- B. Adjust Write Capacity Units (WCUs) of DynamoDB Tables: This step directly addresses the issue with DynamoDB's performance during peak loads. By increasing the WCUs, the DynamoDB tables can handle a higher volume of write operations, which is crucial when the application experiences a surge in photo uploads and corresponding data writes.
- D. Integrate Amazon SQS for Queuing and Reprocessing: Implementing an SQS queue between S3 and Lambda provides a buffer that helps manage the load. This setup allows for the decoupling of photo uploads from processing, enabling better handling of Lambda concurrency limits. It also facilitates reprocessing logic for photos that failed to be processed initially, improving the application's reliability.

The reasons why the other options are less suitable:

- A. Adjust Read Capacity Units (RCUs) of DynamoDB Tables: Adjusting RCUs would improve read performance but does not address the issue at hand, which is related to write performance and Lambda concurrency.
- C. Implement Amazon ElastiCache for Lambda Performance: While ElastiCache can enhance performance, it does not specifically address the identified issues of Lambda concurrency limits and DynamoDB's write performance under heavy loads.
- E. Utilize S3 Transfer Acceleration for Faster Uploads: S3 Transfer Acceleration optimizes data transfer to S3, which is not the bottleneck in this scenario. The issue lies in processing the uploads, not in the upload speed itself.

Question 57

A company operates an on-premises application that allows users to upload media files. This application, hosted on an application server with an attached file server, is experiencing issues with overutilization leading to occasional upload failures. Additionally, the company

regularly needs to expand the file server's storage capacity. The application is accessed by users in the United States and Canada, and it requires user authentication for file uploads. The company is open to refactoring the application and aims to enhance the development process while migrating to AWS, seeking a solution that minimizes operational overhead.

The proposed solutions for migrating and optimizing the application on AWS are:

A. **Migrate Application Server to EC2 with Auto Scaling and Use Amazon S3 for Storage**: Utilize AWS Application Migration Service to move the application server to Amazon EC2 instances. Implement an Auto Scaling group for these EC2 instances and distribute requests using an Application Load Balancer. Refactor the application to store files on Amazon S3 and manage user authentication with Amazon Cognito.

B. **Migrate to EC2, Implement Auto Scaling, and Use IAM Identity Center for Authentication**: Migrate the application server to Amazon EC2 using AWS Application Migration Service. Create an Auto Scaling group for the EC2 instances and use an Application Load Balancer for request distribution. Set up AWS IAM Identity Center (AWS Single Sign-On) for user sign-in and modify the application to use Amazon S3 for file storage.

C. **Develop a Static Website with AWS AppSync, Lambda, and Amazon S3**: Build a static website for file uploads, hosting static assets in Amazon S3. Create an API using AWS AppSync and utilize AWS Lambda resolvers for uploading media files to S3. Employ Amazon Cognito for user authentication.

D. **Use AWS Amplify for Static Website and CloudFront Distribution, Store Files in S3**: Develop a static website for media file uploads using AWS Amplify. Leverage Amplify Hosting to serve the website through Amazon CloudFront. Use Amazon S3 for storing uploaded media files and Amazon Cognito for user authentication.

The most suitable solution is D:

- **D. Use AWS Amplify for Static Website and CloudFront Distribution, Store Files in S3**: This option provides a comprehensive and low-overhead approach. AWS Amplify streamlines the development and deployment of a static website, integrating seamlessly with Amazon CloudFront for content delivery and Amazon S3 for media file storage. Amplify's easy-to-use framework accelerates application development and deployment, while Amazon Cognito integrates smoothly for user authentication. This solution effectively addresses both the operational and development acceleration goals with minimal overhead.

The reasons why the other options are less suitable:

- A. Migrate Application Server to EC2 with Auto Scaling and Use Amazon S3 for Storage: While this approach addresses scalability and storage, it involves migrating and managing EC2 instances, which adds operational complexity compared to a serverless or static website approach.
- B. Migrate to EC2, Implement Auto Scaling, and Use IAM Identity Center for Authentication: Similar to option A, this method includes managing EC2 instances and introduces AWS IAM Identity Center, adding complexity in both migration and ongoing management.
- C. Develop a Static Website with AWS AppSync, Lambda, and Amazon S3: This option, although leveraging serverless technologies, involves a more complex setup with AppSync and Lambda compared to the streamlined process offered by AWS Amplify.

Question 58

A company operates an application deployed on Amazon EC2 instances, which are orchestrated by an Auto Scaling group and routed through an Application Load Balancer (ALB). The application experiences variable workloads, leading to frequent scaling in and out of EC2 instances. The development team faces a challenge: they lose access to valuable application logs when instances are terminated during scale-in events. They seek a solution that allows them to access these logs even after the instances are no longer active.

The proposed solutions to preserve and access application logs post scale-in are:

A. Store ALB Access Logs in Amazon S3: Enable and configure the ALB to save access logs to an Amazon S3 bucket.

B. Publish Logs to Amazon CloudWatch Logs via CloudWatch Agent: Set up the EC2 instances to send logs to Amazon CloudWatch Logs using the CloudWatch unified agent.

C. Implement Step Scaling Policy for the Auto Scaling Group: Adjust the Auto Scaling group to use a step scaling policy.

D. Integrate Application with AWS X-Ray for Tracing: Apply AWS X-Ray tracing within the application for performance analysis.

The most suitable solution is B:

- B. Publish Logs to Amazon CloudWatch Logs via CloudWatch Agent: Configuring the EC2 instances to forward logs to CloudWatch Logs using the CloudWatch agent effectively addresses the issue. This approach ensures that application logs are continuously uploaded to CloudWatch Logs, making them accessible for analysis even after the originating EC2 instances are terminated. It provides a centralized, durable storage solution for logs, independent of the EC2 instance lifecycle.

The reasons why the other options are less suitable:

- A. Store ALB Access Logs in Amazon S3: While ALB access logs can provide valuable insights, they primarily capture information about HTTP requests and responses through the ALB, not the detailed application logs that the development team requires for performance analysis.
- C. Implement Step Scaling Policy for the Auto Scaling Group: Modifying the scaling policy of the Auto Scaling group does not address the fundamental issue of retaining logs after instances are terminated. This solution focuses on scaling behavior rather than log preservation.
- D. Integrate Application with AWS X-Ray for Tracing: AWS X-Ray is a powerful tool for tracing and analyzing requests made to the application, but it is not specifically designed for log storage or retrieval. X-Ray provides insights into application performance and request tracing but does not replace the need for direct access to application logs.

Question 59

A company hosts an unauthenticated static website, which includes a user registration form, using Amazon S3 and Amazon CloudFront. The website is configured with AWS WAF for security. When users submit the registration form, the website calls an Amazon API Gateway endpoint, triggering an AWS Lambda function. This function processes the form data and makes an external API call. However, during testing, a cross-origin resource sharing (CORS) error arises. The solutions architect verifies that the CloudFront distribution's origin is set with the `Access-Control-Allow-Origin` header for `www.example.com`. The task is to resolve this CORS issue.

The potential solutions to fix the CORS error are:

A. Modify S3 Bucket CORS Configuration: Update the CORS settings in the S3 bucket to include rules in the `AllowedOrigin` element for `www.example.com`.

B. Configure CORS in AWS WAF: Enable CORS in AWS WAF and create a web ACL rule to set the `Access-Control-Allow-Origin` header for `www.example.com`.

C. Adjust CORS Settings on API Gateway Endpoint: Enable CORS on the API Gateway API endpoint and configure it to return responses with the `Access-Control-Allow-Origin` header set to `www.example.com`.

D. Set CORS Headers in Lambda Function Response: Enable CORS in the Lambda function and ensure the function's response includes the `Access-Control-Allow-Origin` header set to `www.example.com`.

The best solution is C:

- C. Adjust CORS Settings on API Gateway Endpoint: CORS errors typically occur when a browser makes a request to a different domain than the one serving the web page. In this case, the error is likely due to the API Gateway endpoint not including the correct CORS headers in its responses. Enabling CORS on the API Gateway and ensuring it includes the `Access-Control-Allow-Origin` header in all responses will allow the browser to accept responses from this different origin, resolving the CORS issue.

The reasons why the other options are less suitable:

- A. Modify S3 Bucket CORS Configuration: Adjusting the CORS settings on the S3 bucket hosting the static website is not directly related to the issue. The CORS error is occurring due to requests to the API Gateway, not requests to S3.
- B. Configure CORS in AWS WAF: AWS WAF does not handle CORS settings. CORS is managed at the server level (in this case, the API Gateway), not through web application firewall rules.
- D. Set CORS Headers in Lambda Function Response: While the Lambda function could be modified to include CORS headers in its response, it's more efficient and cleaner to handle CORS at the API Gateway level. This centralizes CORS management for all potential Lambda functions and API methods.

Question 60

A company currently operates multiple AWS accounts for various departments without centralized billing or management. They use Microsoft Azure Active Directory and are looking to consolidate their AWS account management and billing. Additionally, the company wants to shift from manual user management to identity federation, favoring the use of temporary credentials over long-lived access keys.

To achieve these goals, the following actions are proposed:

A. Establish a Central Management Account and Form an AWS Organization: Create a new AWS account to act as the central management account. Set up an organization in AWS Organizations and invite each existing AWS account to join. Ensure all accounts accept the invitation.

B. Standardize AWS Account Email Addresses: Configure each AWS account's email address to follow a uniform format like aws+@example.com for centralized management of account-related emails and invoices.

C. Implement AWS IAM Identity Center with Azure AD Integration: Deploy AWS IAM Identity Center (AWS Single Sign-On) in the central management account. Connect it to Microsoft Azure Active Directory and set up automatic synchronization of users and groups.

D. Deploy AWS Managed Microsoft AD and Share via AWS RAM: Set up an AWS Managed Microsoft AD directory in the central management account and share it across all AWS accounts using AWS Resource Access Manager (AWS RAM).

E. Create and Assign IAM Identity Center Permission Sets: Develop AWS IAM Identity Center permission sets and attach these to the appropriate IAM Identity Center groups and AWS accounts for access management.

F. Configure IAM in Each Account for AWS Managed Microsoft AD Authentication: In every AWS account, set up AWS Identity and Access Management (IAM) to use AWS Managed Microsoft AD for authentication and authorization.

The most suitable combination of actions is A, C, and E:

- A. Establish a Central Management Account and Form an AWS Organization: This is a fundamental step for achieving centralized billing and account management. By creating a management account and forming an AWS Organization, the company can effectively manage all AWS accounts and consolidate billing.
- C. Implement AWS IAM Identity Center with Azure AD Integration: This action addresses the need for identity federation and the use of temporary credentials. By integrating AWS IAM Identity Center with Azure AD, the company can leverage their existing identity management system, allowing for streamlined and secure access control.
- E. Create and Assign IAM Identity Center Permission Sets: This step complements the identity federation setup by allowing the company to define specific access permissions for different users and groups across the AWS accounts, ensuring that access is appropriately managed in line with company policies.

The reasons why the other options are less suitable:

- B. Standardize AWS Account Email Addresses: While having a standard email format might help with organizing communications, it does not contribute to the primary goals of centralized account management, billing, or identity federation.
- D. Deploy AWS Managed Microsoft AD and Share via AWS RAM: This approach adds unnecessary complexity and does not leverage the existing Azure AD setup. It's less efficient compared to integrating directly with Azure AD through AWS IAM Identity Center.
- F. Configure IAM in Each Account for AWS Managed Microsoft AD Authentication: This method would require significant effort to implement and maintain across multiple accounts and does not take advantage of the existing Azure AD infrastructure.

Question 61

A company seeks to migrate 20 business-critical but infrequently used applications to AWS to manage costs. These applications, a mix of Java and Node.js, are currently running across various instance clusters. The goal is to minimize costs while adopting a unified deployment approach. These applications are mainly used for month-end processing, with occasional use at other times. Memory usage varies, with an average below 1 GB, but can peak at 2.5 GB for some applications. A key application within this group is a Java-based billing report that accesses multiple data sources and has lengthy execution times.

The proposed cost-effective solutions for migrating these applications to AWS are:

A. Individual AWS Lambda Functions for Each Application: Use AWS Lambda to deploy each application separately. Monitor job completion with AWS CloudTrail logs and Amazon CloudWatch alarms.

B. Amazon ECS Containers on EC2 with Auto Scaling for Memory Utilization: Use Amazon ECS with containers on Amazon EC2 instances. Configure Auto Scaling based on 75% memory utilization, with an ECS task for each application and task scaling. Monitor services and hosts with Amazon CloudWatch.

C. AWS Elastic Beanstalk with Auto Scaling for Each Application: Deploy each application using AWS Elastic Beanstalk, with Auto Scaling to ensure adequate resources. Use CloudWatch alarms for monitoring each deployment.

D. EC2 Instance Cluster with Auto Scaling and Reserved Instances: Create a new EC2 instance cluster to host all applications, utilizing EC2 Auto Scaling and Application Load Balancers. Implement scaling based on custom memory utilization metrics. Purchase 3-year Reserved Instance reservations to match the Auto Scaling group's `GroupMaxSize` parameter.

The most suitable solution is B:

- B. Amazon ECS Containers on EC2 with Auto Scaling for Memory Utilization: This approach offers a balance between cost-efficiency and flexibility. By using Amazon ECS with containers, the company can standardize deployment across different applications regardless of their runtime environment. Auto Scaling based on memory utilization ensures that resources are efficiently used, scaling up only when necessary. This is particularly beneficial for infrequently used applications with variable memory requirements. Monitoring with CloudWatch ensures visibility into the performance and health of the applications.

The reasons why the other options are less suitable:

- A. Individual AWS Lambda Functions for Each Application: Lambda is suitable for short-duration, event-driven workloads but may not be optimal for long-running

processes like the billing report application. Also, memory constraints and execution time limits of Lambda might not accommodate some of the applications' requirements.

- C. AWS Elastic Beanstalk with Auto Scaling for Each Application: While Elastic Beanstalk simplifies deployment, using it for 20 separate applications could become cumbersome and might not offer the same level of resource optimization as containerized solutions like ECS.
- D. EC2 Instance Cluster with Auto Scaling and Reserved Instances: This approach could lead to over-provisioning, especially for infrequently used applications. Reserved Instances require upfront commitment and may not align well with the variable and sporadic usage patterns of these applications.

Question 62

solutions architect is tasked with optimizing the compute costs of an Amazon EMR cluster that uses the EMR File System (EMRFS). The cluster, critical for business operations, currently runs on Amazon EC2 On-Demand Instances for all its task, primary, and core nodes. The EMR tasks are scheduled to run every morning at 1:00 AM and take approximately 6 hours to complete. The timing of the task completion is not critical, as the processed data is not needed until later in the day. The goal is to redesign the architecture to reduce compute costs.

The proposed solutions for cost optimization are:

A. Use Spot Instances for All Nodes and Terminate Cluster Post-Processing: Configure the EMR cluster to use Spot Instances for task, primary, and core nodes in an instance fleet. Terminate the entire cluster, including all instances, after the processing tasks are finished.

B. On-Demand Instances for Primary/Core Nodes, Spot for Task Nodes, and Terminate Cluster Post-Processing with Compute Savings Plans: Use On-Demand Instances for the primary and core nodes, and Spot Instances for task nodes in an instance fleet. Terminate the entire cluster after processing, and purchase Compute Savings Plans for the On-Demand Instance usage.

C. Maintain On-Demand Instances and Terminate Cluster Post-Processing with Compute Savings Plans: Continue using On-Demand Instances for all nodes. Terminate the cluster after processing is complete and purchase Compute Savings Plans to cover the On-Demand Instance usage.

D. On-Demand for Primary/Core Nodes, Spot for Task Nodes, Terminate Task Nodes Post-Processing with Compute Savings Plans: Deploy primary and core nodes on On-Demand Instances, and task nodes on Spot Instances in an instance fleet. Only terminate the task

node instances after processing, and purchase Compute Savings Plans for the On-Demand Instance usage.

The most appropriate solution is B:

- **B. On-Demand Instances for Primary/Core Nodes, Spot for Task Nodes, and Terminate Cluster Post-Processing with Compute Savings Plans**: This approach balances cost and reliability. Using On-Demand Instances for primary and core nodes ensures stability and uninterrupted operation of the cluster's critical components. Task nodes, which can tolerate interruptions, are run on Spot Instances, providing significant cost savings. Terminating the entire cluster after morning processing aligns with the usage pattern and further reduces costs. Compute Savings Plans for the On-Demand usage offer additional savings, making this a comprehensive cost-optimization strategy.

The reasons why the other options are less suitable:

- **A. Use Spot Instances for All Nodes and Terminate Cluster Post-Processing**: While using Spot Instances for all nodes maximizes cost savings, it risks stability, especially for primary and core nodes, which are critical for cluster operation. Spot Instances can be interrupted, which might affect critical tasks.
- **C. Maintain On-Demand Instances and Terminate Cluster Post-Processing with Compute Savings Plans**: This option, although reliable, does not leverage the cost-saving potential of Spot Instances for task nodes, missing an opportunity to reduce expenses.
- **D. On-Demand for Primary/Core Nodes, Spot for Task Nodes, Terminate Task Nodes Post-Processing with Compute Savings Plans**: Only terminating task nodes and keeping primary/core nodes running incurs unnecessary costs, especially when the data isn't needed immediately after processing.

Question 63

A company has moved a legacy application to AWS, where it runs on three Amazon EC2 instances distributed across three Availability Zones, each in its own private subnet. These instances are targets for an Application Load Balancer (ALB) connected to three public subnets. The application requires communication with on-premises systems, and the company's security policy allows access to these systems only from specific company IP addresses. The company has designated a single IP address from its internal range for cloud usage, added this IP to its firewall's allow list, and created an Elastic IP address corresponding to this internal IP.

The solutions architect is tasked with enabling the application's communication with on-premises systems while ensuring automatic failure mitigation. The following solutions are

proposed:

A. Deploy NAT Gateways with Elastic IP and Health Checks: Set up three NAT gateways, one in each public subnet, and assign the Elastic IP address to these NAT gateways. Implement health checks, and if a NAT gateway fails, recreate it and reassign the Elastic IP address.

B. Use NLB with Elastic IP and Health Checks: Switch from ALB to a Network Load Balancer (NLB), assign the Elastic IP address to the NLB, and enable health checks. In case of a health check failure, redeploy the NLB in different subnets.

C. Single NAT Gateway with CloudWatch Monitoring and Lambda: Deploy a single NAT gateway in a public subnet and assign the Elastic IP address to it. Monitor the NAT gateway with Amazon CloudWatch and a custom metric. Use an AWS Lambda function to create a new NAT gateway in a different subnet and assign the Elastic IP address to it if the original NAT gateway becomes unhealthy.

D. Assign Elastic IP to ALB with Route 53 Health Checks: Assign the Elastic IP address to the ALB and create an Amazon Route 53 simple record pointing to this Elastic IP. Set up Route 53 health checks, and in case of failure, recreate the ALB in different subnets.

The most suitable solution is C:

- C. Single NAT Gateway with CloudWatch Monitoring and Lambda: This approach is effective and efficient. A single NAT gateway simplifies the network setup while still providing the necessary communication path to on-premises systems using the designated Elastic IP address. Utilizing CloudWatch for monitoring and a Lambda function for automated redeployment ensures high availability and automatic mitigation in case the NAT gateway becomes unhealthy. This setup provides a balance between simplicity, cost, and reliability.

The reasons why the other options are less suitable:

- A. Deploy NAT Gateways with Elastic IP and Health Checks: Deploying multiple NAT gateways adds complexity and cost without providing additional benefits, as the traffic only needs to come from a single company IP address.
- B. Use NLB with Elastic IP and Health Checks: Replacing the ALB with an NLB and assigning an Elastic IP to it doesn't address the requirement for the application instances in private subnets to initiate communication with on-premises systems. NLBs are primarily used for incoming traffic, not for outbound communication scenarios like this one.
- D. Assign Elastic IP to ALB with Route 53 Health Checks: Assigning an Elastic IP to the ALB does not facilitate outbound communication from the EC2 instances to the on-

premises systems. Like option B, this setup focuses on inbound traffic management, which is not the primary requirement in this scenario.

Question 64

A company managing over a thousand AWS accounts through AWS Organizations has established a new organization specifically for developers. It needs to transfer 540 existing developer accounts, each capable of independent operation, from the current organization to this new developer-focused organization.

To accomplish this account migration, several steps are proposed:

A. Utilize MoveAccount Operation in Old Organization: Execute the `MoveAccount` operation within the Organizations API from the old organization's management account to shift the developer accounts to the new developer organization.

B. Remove Accounts from Old Organization via Management Account: From the management account of the old organization, employ the `RemoveAccountFromOrganization` operation in the Organizations API to disassociate each developer account from the old organization.

C. Developer Accounts Self-remove from Old Organization: Instruct owners of each developer account to independently use the `RemoveAccountFromOrganization` operation in the Organizations API to exit the old organization.

D. Create Placeholder Account in New Organization for Migration: Log into the new developer organization's management account and establish a placeholder member account to serve as a migration target for the developer accounts.

E. Invite Developer Accounts from New Organization's Management Account: From the management account of the new developer organization, utilize the `InviteAccountToOrganization` operation in the Organizations API to extend invitations to the developer accounts.

F. Developers Confirm Joining New Organization: Require each developer to log into their individual AWS account and affirmatively accept the invitation to become part of the new developer organization.

The most effective combination of steps is B, E, and F:

- B. Remove Accounts from Old Organization via Management Account: This step is essential for initiating the migration process. An AWS account must be removed from its current organization before it can join another. Doing this from the management account of the old organization streamlines the process.

- E. Invite Developer Accounts from New Organization's Management Account: Once the accounts are removed from the old organization, the new developer organization's management account can formally invite these accounts. This step is a standard procedure in AWS for adding accounts to an organization.
- F. Developers Confirm Joining New Organization: Finally, each developer must manually accept the invitation to join the new organization. This ensures that the transfer of each account is authorized and controlled by the account holder.

The reasons why the other options are less suitable:

- A. Utilize MoveAccount Operation in Old Organization: AWS Organizations does not currently support a direct `MoveAccount` operation for transferring accounts between organizations. The process requires removing accounts from one organization and then inviting them to another.
- C. Developer Accounts Self-remove from Old Organization: While technically possible, this approach is operationally intensive and could lead to coordination challenges and inconsistencies.
- D. Create Placeholder Account in New Organization for Migration: Creating a placeholder account is unnecessary for the migration process. The focus should be on transitioning existing accounts from the old organization to the new one effectively.

Question 65

A company operates a web application that serves images from an Amazon S3 bucket via an Amazon CloudFront distribution. Occasionally, corrupted images are uploaded into the S3 bucket by third-party tools, negatively impacting the user experience. The company has developed Python code capable of identifying these corrupt images. The requirement is to seamlessly integrate this detection logic into their workflow, ensuring that corrupt images are identified swiftly after being ingested, thereby minimizing any impact on the user experience.

The proposed solutions for integrating the detection logic are:

A. Implement a Lambda@Edge Function Triggered by Viewer-Response Event: Use a Lambda@Edge function, triggered during a viewer-response event in CloudFront, to execute the image detection logic.

B. Implement a Lambda@Edge Function Triggered by Origin-Response Event: Deploy a Lambda@Edge function that is invoked during an origin-response event in CloudFront to carry out the image detection process.

C. Trigger an AWS Lambda Function via S3 Event Notification: Configure an event notification in the S3 bucket to invoke an AWS Lambda function whenever new images are uploaded, allowing the function to execute the image corruption detection logic.

D. Activate AWS Step Functions via S3 Event Notification: Set up an S3 event notification to trigger an AWS Step Functions state machine when new images are added to the bucket.

The best solution is C:

- C. Trigger an AWS Lambda Function via S3 Event Notification: This approach is the most direct and efficient. By setting up an S3 event notification to trigger a Lambda function upon the upload of new images, the detection logic can be executed immediately after image ingestion. This setup minimizes the latency between image upload and corruption detection, ensuring corrupt images are identified before they are served to users.

The reasons why the other options are less suitable:

- A. Implement a Lambda@Edge Function Triggered by Viewer-Response Event: Lambda@Edge functions triggered by viewer-response events act after CloudFront serves the content to the user. This timing would be too late for detecting and filtering out corrupted images, as the user would have already received the image.
- B. Implement a Lambda@Edge Function Triggered by Origin-Response Event: While Lambda@Edge functions triggered on origin-response events act before the content is cached by CloudFront, they are invoked after the content is fetched from the origin (S3 in this case). This means corrupted images could still be served to the first user requesting them and then cached.
- D. Activate AWS Step Functions via S3 Event Notification: While AWS Step Functions could orchestrate a more complex workflow, it adds unnecessary complexity for this use case. A direct invocation of a Lambda function is more straightforward and efficient for immediate image corruption detection.

Question 66

A company's application, hosted on Amazon EC2 instances within an EC2 Auto Scaling group, undergoes frequent scaling events, resulting in a constantly changing set of instances. The company utilizes AWS CodePipeline for deploying application updates. Currently, they manually install the AWS CodeDeploy agent on new EC2 instances and associate these instances with a CodeDeploy deployment group each time there's a scaling event. The application is scheduled to be updated and go live in the next 24 hours. The company seeks a recommendation from a solutions architect on how to automate this deployment process to minimize operational overhead.

The solutions presented for automating the application deployment are:

A. Use EventBridge and Lambda for Instance Association: Configure Amazon EventBridge to trigger an AWS Lambda function when new EC2 instances launch in the Auto Scaling

group. The Lambda function should automatically associate these new instances with the CodeDeploy deployment group.

B. Script to Suspend Auto Scaling and Update AMI: Write a script to temporarily suspend EC2 Auto Scaling activities during the new code deployment. Once the deployment is complete, create a new AMI with the updated code, configure the Auto Scaling group's launch template to use this new AMI, and then resume Auto Scaling operations.

C. Use CodeBuild for AMI Creation and Auto Scaling Update: Set up a new AWS CodeBuild project to build a new AMI containing the updated application code. Update the Auto Scaling group's launch template to this new AMI and execute an EC2 Auto Scaling instance refresh.

D. Pre-install CodeDeploy Agent on AMI and Associate Deployment Group with Auto Scaling Group: Build a new AMI that already includes the CodeDeploy agent. Update the Auto Scaling group's launch template to use this new AMI. Modify the setup so that the CodeDeploy deployment group is associated with the Auto Scaling group rather than individual EC2 instances.

The best solution is D:

- D. Pre-install CodeDeploy Agent on AMI and Associate Deployment Group with Auto Scaling Group: This approach streamlines the deployment process significantly. By incorporating the CodeDeploy agent into the AMI used by the Auto Scaling group, there's no need to install the agent each time a new instance is launched. Associating the CodeDeploy deployment group directly with the Auto Scaling group, instead of individual instances, ensures that all current and future instances in the group are automatically included in the deployment process. This setup greatly reduces manual intervention and operational overhead.

The reasons why the other options are less suitable:

- A. Use EventBridge and Lambda for Instance Association: While this method automates the association of new instances with the CodeDeploy deployment group, it introduces additional complexity with the need to maintain a Lambda function. It also doesn't address the need to install the CodeDeploy agent on new instances.
- B. Script to Suspend Auto Scaling and Update AMI: This method disrupts the Auto Scaling process and requires manual intervention to create and update AMIs. It's not a seamless or efficient solution, especially for an environment with frequent scaling events.
- C. Use CodeBuild for AMI Creation and Auto Scaling Update: Although this option automates the creation of a new AMI and its incorporation into the Auto Scaling group, it requires the setup and maintenance of a CodeBuild project. Like option B, it could be less efficient due to the frequent scaling events.

Question 67

A company operates a website hosted on four Amazon EC2 instances, all routed through an Application Load Balancer (ALB). Currently, when an EC2 instance becomes unavailable and triggers a CloudWatch alarm, a manual process is followed by the operations team to replace the failed instance. The company is looking for a solution architect to devise a method that automates the replacement of EC2 instances, ensuring high availability and minimizing downtime during the transition to this new automated process.

The potential solutions for automating instance replacement are:

A. Replace ALB and Integrate Auto Scaling Group: Remove the current ALB, then set up an Auto Scaling group designed for the web application's traffic. Implement a new launch template for the Auto Scaling group, create a new ALB, link the Auto Scaling group to this new ALB, and finally add the existing EC2 instances to the Auto Scaling group.

B. Integrate Auto Scaling Group with Existing ALB: Establish an Auto Scaling group tailored to the web application's traffic demands. Connect a new launch template to this Auto Scaling group, and then attach it to the existing ALB. Add the current EC2 instances to this Auto Scaling group.

C. Rebuild ALB and EC2 Instances with Auto Scaling Group: Eliminate the existing ALB and EC2 instances. Create a new Auto Scaling group for the web application, attach a new launch template to it, and then set up a new ALB. Link the Auto Scaling group to this new ALB and wait for it to launch the necessary minimum number of EC2 instances.

D. Create Auto Scaling Group and Wait for ALB Registration: Formulate an Auto Scaling group to manage the website's traffic. Add a new launch template to the Auto Scaling group and connect it to the existing ALB. Wait for the current ALB to automatically register the existing EC2 instances with the Auto Scaling group.

The best solution is B:

- B. Integrate Auto Scaling Group with Existing ALB: This approach is the most efficient and involves minimal disruption. By creating an Auto Scaling group and linking it to the existing ALB, the solution ensures continuous operation of the website. The new Auto Scaling group, equipped with a launch template, automates the process of replacing any unavailable EC2 instances, thus maintaining high availability. The integration with the existing ALB avoids unnecessary downtime that would occur with setting up a new ALB.

The reasons why the other options are less suitable:

- A. Replace ALB and Integrate Auto Scaling Group: This option introduces unnecessary downtime by replacing the existing ALB. The creation of a new ALB is not required for

automating instance replacement and would lead to additional complexity and potential service disruption.

- C. Rebuild ALB and EC2 Instances with Auto Scaling Group: Completely rebuilding the ALB and EC2 instances would cause significant downtime and disruption to the service. This approach is more complex and risky compared to simply integrating an Auto Scaling group with the existing infrastructure.
- D. Create Auto Scaling Group and Wait for ALB Registration: This option misunderstands the functionality of ALBs and Auto Scaling groups. The existing ALB does not automatically register existing EC2 instances with a new Auto Scaling group. Manual intervention would be required, which does not meet the requirement for automation.

Question 68

A company is seeking ways to reduce data-transfer and compute costs incurred due to developer activities within its AWS Organizations structure. Developers are currently launching Amazon EC2 instances in a single AWS Region and retrieving about 1 TB of data daily from Amazon S3. This activity has led to substantial monthly charges for data transfer and NAT gateway processing between the EC2 instances and S3 buckets, in addition to high compute costs. The company aims to implement cost-effective solutions that enforce approved architectural patterns for EC2 instances and VPC infrastructure deployed by developers. This enforcement should not hinder the developers' efficiency in performing their tasks.

The proposed solutions for optimizing costs and enforcing architectural patterns are:

A. Use SCPs and Provide CloudFormation Templates: Implement Service Control Policies (SCPs) to restrict launching of unapproved EC2 instance types. Supply developers with an AWS CloudFormation template for deploying a standard VPC configuration with S3 interface endpoints. Restrict developers to launching VPC resources only through CloudFormation.

B. Monitor with AWS Budgets and Implement Budget Actions: Establish a forecasted budget in AWS Budgets to monitor EC2 and S3 data-transfer costs. Send alerts to developers when costs reach 75% of the budgeted amount, and take action to terminate EC2 instances and VPC infrastructure if costs hit 100% of the budget.

C. Deploy AWS Service Catalog Portfolio: Create an AWS Service Catalog portfolio containing approved VPC configurations with S3 gateway endpoints and EC2 instances. Share this portfolio with developer accounts, and use a launch constraint with an approved IAM role. Limit developers' IAM permissions to only accessing AWS Service Catalog.

D. Monitor Compliance with AWS Config and Remediate: Utilize AWS Config to monitor compliance of EC2 and VPC resources in developer accounts. Set rules to identify unapproved EC2 instances or VPCs lacking S3 gateway endpoints, and automate the termination of non-compliant resources.

The most cost-effective solution is C:

- C. Deploy AWS Service Catalog Portfolio: This solution effectively balances cost optimization with operational efficiency. By providing a Service Catalog portfolio of approved resources, the company ensures that developers deploy only compliant VPC and EC2 configurations, including S3 gateway endpoints which reduce data-transfer costs. This approach not only enforces architectural standards but also allows developers to continue their work with minimal disruption. The limitation of IAM permissions to the Service Catalog prevents deviations from approved resources, maintaining cost control.

The reasons why the other options are less suitable:

- A. Use SCPs and Provide CloudFormation Templates: While SCPs effectively restrict certain actions, relying solely on CloudFormation templates for resource deployment could limit flexibility in certain scenarios. This method might not fully address the specific cost concerns related to S3 data transfers.
- B. Monitor with AWS Budgets and Implement Budget Actions: This approach is reactive rather than proactive. While it helps in monitoring and controlling costs, it does not proactively guide developers to use cost-effective architectures. Terminating resources based on budget thresholds could disrupt ongoing development work.
- D. Monitor Compliance with AWS Config and Remediate: Using AWS Config for compliance monitoring is a good practice, but the automatic termination of non-compliant resources might be too disruptive for ongoing development activities. This approach could potentially slow down the development process.

Question 69

A growing company is restructuring its AWS resources by dividing them into hundreds of separate AWS accounts across multiple regions. To ensure compliance and control, the company requires a solution that restricts operations exclusively to certain specified AWS Regions, preventing any activities outside these approved regions.

The proposed solutions for regional restriction are:

A. Implement Region-Specific IAM Roles: Set up individual IAM roles in each account with IAM policies that conditionally allow permissions only in the approved regions.

B. **Use AWS Organizations with Region-Restricted IAM Users**: Establish an organization in AWS Organizations, create IAM users for each account, and attach policies to these users that block access to unapproved regions.

C. **Deploy AWS Control Tower with SCPs in OUs**: Implement an AWS Control Tower landing zone, organize accounts into Organizational Units (OUs), and apply Service Control Policies (SCPs) that deny the ability to run services outside of designated regions.

D. **Activate AWS Security Hub with Regional Controls**: Enable AWS Security Hub in each account and configure controls that specify the regions where accounts are permitted to deploy infrastructure.

The best solution is C:

- **C. Deploy AWS Control Tower with SCPs in OUs**: This approach effectively meets the requirement of regional restriction at scale. AWS Control Tower provides a robust and centralized way to manage multiple accounts and regions. By using SCPs within OUs, the company can precisely control which services and actions are allowed or denied in specific regions for all accounts under the OU. This setup not only enforces regional compliance but also simplifies management and oversight of the accounts.

The reasons why the other options are less suitable:

- **A. Implement Region-Specific IAM Roles**: While IAM roles with conditional policies can restrict region-specific access, managing these roles and policies across hundreds of accounts would be operationally complex and time-consuming. This approach lacks the centralized control and scalability offered by AWS Control Tower.
- **B. Use AWS Organizations with Region-Restricted IAM Users**: Attaching region-specific policies to IAM users in each account is less efficient and more prone to management challenges, especially as the number of accounts and users grows. It does not offer the same level of governance and ease of administration as using SCPs within AWS Control Tower.
- **D. Activate AWS Security Hub with Regional Controls**: Security Hub is primarily a security and compliance monitoring service and does not have the capability to enforce regional restrictions on operations like deploying infrastructure. It is not designed to function as a governance tool for regional access control.

Question 70

A company is looking to modernize its retail ordering web application, which is currently based on a load-balanced Amazon EC2 instance fleet handling web hosting, database API services, and business logic. The aim is to transition to a decoupled and scalable architecture

that also includes a reliable system for capturing and retaining failed orders, all while keeping operational costs low.

The proposed solutions for this architectural refactoring are:

A. **S3 for Web Hosting, API Gateway, SQS, and ECS with SQS Long Polling**: Utilize Amazon S3 for hosting the web application, Amazon API Gateway for database API services, and Amazon Simple Queue Service (SQS) for managing order queues. Implement business logic using Amazon Elastic Container Service (ECS) and use SQS long polling to handle failed orders.

B. **Elastic Beanstalk for Web Hosting, API Gateway, Amazon MQ, and Step Functions with S3 Glacier Deep Archive**: Deploy web hosting on AWS Elastic Beanstalk, use Amazon API Gateway for database APIs, and Amazon MQ for order queuing. Employ AWS Step Functions for business logic and store failed orders in Amazon S3 Glacier Deep Archive.

C. **S3 for Web Hosting, AppSync for Database APIs, SQS, and Lambda with SQS Dead-Letter Queue**: Host the web application on Amazon S3 and use AWS AppSync for database API services. Leverage Amazon SQS for order queuing and AWS Lambda for business logic. Utilize an Amazon SQS dead-letter queue for retaining failed orders.

D. **Lightsail for Web Hosting, AppSync for Database APIs, SES for Order Queuing, EKS, and OpenSearch Service**: Host the web application on Amazon Lightsail, with AWS AppSync for database API services. Use Amazon Simple Email Service (SES) for order queuing and Amazon Elastic Kubernetes Service (EKS) for business logic. Retain failed orders using Amazon OpenSearch Service.

The best solution is C:

- **C. S3 for Web Hosting, AppSync for Database APIs, SQS, and Lambda with SQS Dead-Letter Queue**: This architecture offers a highly scalable and cost-effective solution. Amazon S3 is an efficient platform for static web hosting. AWS AppSync provides a flexible and scalable approach to handle database API interactions. Amazon SQS effectively manages order queuing, and AWS Lambda allows for serverless execution of business logic, reducing operational overhead. The use of an SQS dead-letter queue for capturing failed orders is an efficient way to handle errors and exceptions.

The reasons why the other options are less suitable:

- **A. S3 for Web Hosting, API Gateway, SQS, and ECS with SQS Long Polling**: While this is a viable solution, it introduces more complexity and potential cost with ECS compared

to the serverless approach of Lambda in option C. ECS requires more management and configuration, which can increase operational overhead.

- B. Elastic Beanstalk for Web Hosting, API Gateway, Amazon MQ, and Step Functions with S3 Glacier Deep Archive: This setup is less efficient for a decoupled architecture. Elastic Beanstalk and Step Functions introduce additional layers of management, and storing failed orders in S3 Glacier Deep Archive is not optimal for quick retrieval and reprocessing.
- D. Lightsail for Web Hosting, AppSync for Database APIs, SES for Order Queuing, EKS, and OpenSearch Service: Using Lightsail and EKS adds complexity and potentially higher costs. Amazon SES is not designed for order queuing like SQS. Also, using Amazon OpenSearch Service for retaining failed orders is an over-engineered solution for this requirement.

Question 71

A company's web application is hosted on AWS in the us-east-1 Region, utilizing three Availability Zones for its application servers behind an Application Load Balancer. The backend database currently operates on a MySQL database hosted on an Amazon EC2 instance. As part of enhancing disaster recovery capabilities, the company plans to extend its operations to the us-west-2 Region. They aim to achieve a Recovery Time Objective (RTO) of less than 5 minutes and a Recovery Point Objective (RPO) of less than 1 minute. The company has already started deploying application servers in the us-west-2 Region and set up Amazon Route 53 for health checks and DNS failover to this new region. The solutions architect is now tasked with determining the best approach to ensure the database meets the desired RTO and RPO in a cross-Region context.

The proposed solutions for the database cross-Region data recovery are:

A. Use Amazon RDS for MySQL with Cross-Region Read Replica in us-west-2: Transition the database to Amazon RDS for MySQL and establish a cross-Region read replica in the us-west-2 Region.

B. Implement Amazon Aurora Global Database with Primary in us-east-1 and Secondary in us-west-2: Convert the database to an Amazon Aurora global database, maintaining the primary cluster in us-east-1 and setting up a secondary cluster in us-west-2.

C. Transition to Amazon RDS for MySQL with Multi-AZ Deployment: Migrate the database to an Amazon RDS for MySQL instance configured for Multi-AZ deployment.

D. Set Up a Standby MySQL Database on EC2 in us-west-2: Create a standby MySQL database on an EC2 instance in the us-west-2 Region.

The best solution is B:

- B. Implement Amazon Aurora Global Database with Primary in us-east-1 and Secondary in us-west-2: Aurora Global Databases are designed for exactly this kind of cross-Region data recovery scenario. They offer fast replication (typically under a second) across Regions, which aligns with the company's RPO of less than 1 minute. In the event of a regional failure, the secondary cluster in us-west-2 can be promoted to take over, meeting the RTO requirement of less than 5 minutes. This solution provides a seamless, high-performance cross-Region database failover mechanism.

The reasons why the other options are less suitable:

- A. Use Amazon RDS for MySQL with Cross-Region Read Replica in us-west-2: While RDS cross-Region read replicas provide some level of cross-Region data recovery, the replication lag might not consistently meet the RPO of less than 1 minute. Also, the failover process may exceed the RTO of 5 minutes.
- C. Transition to Amazon RDS for MySQL with Multi-AZ Deployment: Multi-AZ deployments provide high availability within a single Region but do not address cross-Region disaster recovery requirements. This option wouldn't provide the necessary geographic redundancy.
- D. Set Up a Standby MySQL Database on EC2 in us-west-2: Managing a standby database on EC2 requires significant manual effort for replication, failover, and maintenance. This approach is likely to be less reliable and more time-consuming compared to the managed service offered by Aurora Global Databases.

Question 72

A company managing multiple accounts through AWS Organizations has a regulatory requirement to limit certain member accounts to operate only within specific AWS Regions. Additionally, there's a need to enforce standardized resource tagging across these accounts. The company seeks a centralized, minimally intensive configuration approach to enforce these requirements.

The potential solutions for regional restriction and tagging enforcement are:

A. Implement AWS Config Rules and Tag Policy in Member Accounts: Set up AWS Config rules within the specific member accounts to restrict the use of Regions and implement a tag policy in these accounts.

B. Use AWS Billing and Cost Management to Disable Regions and Apply Tag Policy: In the management account, use the AWS Billing and Cost Management console to disable Regions for the specific member accounts and apply a tag policy at the root level.

C. Apply Tag Policy and SCP at Root Level: Link the specific member accounts with the root of the AWS Organization, and then apply a tag policy and a Service Control Policy (SCP) with

conditions to limit Regions at the root level.

D. Create a New Organizational Unit (OU) for Member Accounts with SCP and Tag Policy: Place the specific member accounts in a new OU and apply a tag policy along with an SCP that includes conditions to restrict Regions.

The best solution is D:

- D. Create a New Organizational Unit (OU) for Member Accounts with SCP and Tag Policy: This approach effectively centralizes and streamlines the management of both regional restrictions and tag policies. By organizing the specific member accounts into a new OU, the company can uniformly apply an SCP to limit Regions and a tag policy to enforce tagging standards across all accounts within the OU. This method provides a consolidated and efficient way to manage policies across multiple accounts.

The reasons why the other options are less suitable:

- A. Implement AWS Config Rules and Tag Policy in Member Accounts: While AWS Config can help enforce compliance, setting up rules and tag policies in each member account individually is operationally intensive and does not leverage the centralized management capabilities of AWS Organizations.
- B. Use AWS Billing and Cost Management to Disable Regions and Apply Tag Policy: AWS Billing and Cost Management does not provide the capability to disable Regions for specific accounts. Additionally, applying tag policies at the root level without an OU structure may not provide the targeted control needed for specific member accounts.
- C. Apply Tag Policy and SCP at Root Level: Applying these policies directly at the root level affects all accounts in the organization, not just the specific member accounts needing regional restrictions. This lacks the granularity that creating a separate OU offers.

Question 73

A company's application creates reports and saves them in an Amazon S3 bucket. To provide access to these reports, the application generates signed URLs for users. However, the company's security team has identified a significant security issue: the report files are publicly accessible without any authentication. Consequently, the company has halted the production of new reports until this issue is resolved. The goal is to quickly fix this security vulnerability without disrupting the application's standard operation of generating and providing access to reports.

The proposed solutions for addressing this security concern are:

A. Implement Lambda Function for Deny Policy: Develop an AWS Lambda function to enforce a policy that denies access to all unauthenticated users, and set up a scheduled event to trigger this Lambda function.

B. Follow AWS Trusted Advisor Recommendations: Consult the AWS Trusted Advisor bucket permissions check and enact the suggested measures.

C. Execute a Script for Private ACLs on Objects: Execute a script to change the Access Control List (ACL) settings of all objects in the S3 bucket to private.

D. Activate Block Public Access in Amazon S3: Use the Block Public Access feature in Amazon S3 to enable the 'IgnorePublicAcls' setting, setting it to TRUE for the bucket.

The best solution is D:

- D. Activate Block Public Access in Amazon S3: This option provides an immediate and effective solution to the problem. By enabling the Block Public Access feature and setting 'IgnorePublicAcls' to TRUE, the S3 bucket will automatically block any public access granted through ACLs. This action ensures that the files can no longer be accessed publicly, aligning with the security requirements, and it does so without affecting the application's functionality to generate signed URLs for authenticated access.

The reasons why the other options are less suitable:

- A. Implement Lambda Function for Deny Policy: While a Lambda function could potentially enforce access restrictions, it introduces additional complexity and may not provide an immediate solution. It also requires ongoing management and does not leverage built-in S3 features designed for this purpose.
- B. Follow AWS Trusted Advisor Recommendations: While Trusted Advisor can provide valuable insights into security settings, it is more of a diagnostic tool and does not offer an immediate remediation action like the direct modification of bucket settings.
- C. Execute a Script for Private ACLs on Objects: Manually updating ACLs on each object in the bucket could be time-consuming and error-prone, especially if the bucket contains a large number of files. It also doesn't prevent future objects from being uploaded with public access.

Question 74

A company intends to transition their Amazon RDS for Oracle database to an RDS for PostgreSQL DB instance hosted in a different AWS account. The migration strategy must ensure zero downtime and minimize the time taken to complete the migration. It's crucial that the new PostgreSQL database replicates all existing data from the Oracle database, as well as any new data that gets added during the migration process. Post-migration, the target

PostgreSQL database should be an exact replica of the original Oracle database. The applications currently interact with the Oracle database using an Amazon Route 53 CNAME record pointing to the database's endpoint. The Oracle DB instance is located within a private subnet. The objective is to determine a set of steps to fulfill these migration requirements effectively.

The proposed steps for this migration are (choose three):

A. **Create New RDS for PostgreSQL and Migrate Schema with AWS SCT**: Launch a new RDS for PostgreSQL DB instance in the target AWS account and use the AWS Schema Conversion Tool (AWS SCT) to migrate the database schema from the source Oracle database to the target PostgreSQL database.

B. **Use AWS SCT for RDS PostgreSQL Instance Creation and Initial Data Migration**: Utilize the AWS SCT to create a new RDS for PostgreSQL DB instance in the target account, including the schema and initial data transfer from the source Oracle database.

C. **Set Up VPC Peering and Configure Security Groups for DB Connectivity**: Establish VPC peering between the VPCs in both AWS accounts, ensuring connectivity to both the Oracle and PostgreSQL DB instances from the target account. Modify the security groups of each DB instance to permit traffic on the database port from the VPC in the target account.

D. **Make Source DB Publicly Accessible for Target Account Connectivity**: Temporarily configure the source Oracle DB instance for public accessibility to enable connectivity from the VPC in the target account. Adjust the security groups of each DB instance to allow database port traffic from the target account's VPC.

E. **Perform Full Load and CDC Migration with AWS DMS and Update CNAME Record**: Use AWS Database Migration Service (AWS DMS) in the target account for a full load migration combined with change data capture (CDC) from the Oracle database to the PostgreSQL database. Once migration completes, update the Route 53 CNAME record to direct to the endpoint of the target PostgreSQL DB instance.

F. **Conduct CDC Migration with AWS DMS and Update CNAME Record**: Implement AWS DMS in the target account for a CDC-only migration from the Oracle database to the PostgreSQL database. After the migration is finished, update the Route 53 CNAME record to the target DB instance's endpoint.

The most suitable combination of steps is A, C, and E:

- **A. Create New RDS for PostgreSQL and Migrate Schema with AWS SCT**: This step effectively initiates the migration by setting up the target environment (PostgreSQL DB) and migrating the schema, which is a critical first step in database migration.

- C. Set Up VPC Peering and Configure Security Groups for DB Connectivity: Establishing VPC peering and configuring security groups is a secure way to ensure connectivity between the source and target databases across different AWS accounts without exposing the database to the public internet.
- E. Perform Full Load and CDC Migration with AWS DMS and Update CNAME Record: Using AWS DMS for both full data load and CDC migration ensures that all data, including ongoing changes, are replicated to the target database. Updating the CNAME record post-migration allows for seamless redirection of application traffic to the new database.

The reasons why the other options are less suitable:

- B. Use AWS SCT for RDS PostgreSQL Instance Creation and Initial Data Migration: AWS SCT doesn't create RDS instances or perform initial data migration; it's primarily used for schema conversion.
- D. Make Source DB Publicly Accessible for Target Account Connectivity: This approach poses a security risk by making the database publicly accessible, even temporarily.
- F. Conduct CDC Migration with AWS DMS and Update CNAME Record: A CDC-only migration does not cover the initial full load of existing data, which is essential for ensuring that the target database is a complete replica of the source.

Question 75

A company's event-driven ordering system, which utilizes an Amazon Simple Queue Service (Amazon SQS) standard queue, encountered an issue during testing where the processing of orders halted. Investigation revealed that a specific order message in the SQS queue was causing an error in the backend, subsequently blocking the processing of all following order messages. The queue's visibility timeout is currently set at 30 seconds, while the backend processing has a timeout limit of 10 seconds. The solutions architect is tasked with devising a strategy to handle such faulty order messages effectively, ensuring uninterrupted processing of subsequent messages in the system.

The proposed solutions for managing the faulty messages are:

A. Align Backend Processing Timeout with Visibility Timeout: Increase the backend processing timeout to 30 seconds to match the SQS queue's visibility timeout.

B. Reduce Queue Visibility Timeout: Decrease the visibility timeout of the SQS queue to automatically remove the problematic message.

C. Use SQS FIFO Queue as a Dead-Letter Queue: Set up a new SQS FIFO (First-In-First-Out) queue as a dead-letter queue to segregate the faulty messages.

D. Implement SQS Standard Queue as a Dead-Letter Queue: Establish a new SQS standard queue to function as a dead-letter queue, isolating the erroneous messages.

The most appropriate solution is D:

- D. Implement SQS Standard Queue as a Dead-Letter Queue: Configuring a dead-letter queue (DLQ) is the best practice to handle messages that repeatedly cause errors. By setting up an SQS standard queue as the DLQ, the system can automatically redirect messages that fail to process multiple times (like the faulty order message) to this separate queue. This approach ensures that problematic messages are isolated and do not impede the processing of other messages in the main queue.

The reasons why the other options are less suitable:

- A. Align Backend Processing Timeout with Visibility Timeout: Simply aligning the backend processing timeout with the visibility timeout doesn't address the root issue of handling the faulty message. It may lead to longer processing times without resolving message blockage.
- B. Reduce Queue Visibility Timeout: Reducing the visibility timeout won't effectively solve the problem of a message that causes errors; it might just lead to the faulty message being retried more frequently, potentially causing more disruptions.
- C. Use SQS FIFO Queue as a Dead-Letter Queue: Using an SQS FIFO queue as a DLQ is not recommended for a standard queue because the ordering of messages is not preserved in standard queues. A standard queue as a DLQ is more appropriate for compatibility with the source standard queue.

Question 76

A company's automated nightly process for retraining machine learning models, utilizing AWS Step Functions and AWS Lambda, has encountered an issue. The workflow, comprising multiple steps, has failed on several occasions, unbeknownst to the company due to a lack of failure notifications. The task at hand for the solutions architect is to enhance the workflow to ensure that any kind of failure in the retraining process triggers a notification.

The proposed steps to enable failure notifications are:

A. Set Up an Amazon SNS Topic with Email Subscription: Create an Amazon Simple Notification Service (Amazon SNS) topic and subscribe the team's email list to this topic via an "Email" subscription type.

B. Create an "Email" Task for SNS Notification: Implement a new task within the workflow, named "Email", which sends the input arguments from the failed step to the SNS topic.

C. Integrate Catch Field for All State Types with SNS Task: Add a "Catch" field to every "Task", "Map", and "Parallel" state in the Step Functions workflow, specifying "ErrorEquals": ["States.ALL"] and setting "Next": "Email" to redirect all errors to the "Email" task.

D. Add and Verify Email in Amazon SES: Register a new email address with Amazon Simple Email Service (Amazon SES) and verify the email address.

E. Create an "Email" Task for SES Notification: Establish a new task named "Email" in the workflow that sends the input arguments from failed steps to the specified SES email address.

F. Include Catch Field for Runtime Errors with Email Notification: Incorporate a "Catch" field in all "Task", "Map", and "Parallel" states with the condition "ErrorEquals": ["States.Runtime"] and "Next": "Email" for handling runtime errors.

The most effective combination of steps is A, B, and C:

- A. Set Up an Amazon SNS Topic with Email Subscription: Creating an SNS topic with an email subscription is a straightforward and efficient way to set up notifications. It allows for the easy distribution of alerts to a team's mailing list whenever the specified event (workflow failure) occurs.
- B. Create an "Email" Task for SNS Notification: Implementing an "Email" task within the Step Functions workflow specifically for forwarding failure notifications to the SNS topic ensures that notifications are sent promptly upon any failure in the workflow.
- C. Integrate Catch Field for All State Types with SNS Task: Adding a "Catch" field to all relevant state types with a directive to proceed to the "Email" task upon encountering any error ('States.ALL') ensures comprehensive error handling. This step guarantees that any failure at any point in the workflow triggers a notification.

The reasons why the other options are less suitable:

- D. Add and Verify Email in Amazon SES: While SES is a valid service for sending emails, it introduces unnecessary complexity for this scenario. SNS is more straightforward for setting up notifications, especially when integrated with Step Functions.
- E. Create an "Email" Task for SES Notification: Similar to option D, using SES for notifications is more complex compared to using SNS, particularly in the context of a Step Functions workflow.
- F. Include Catch Field for Runtime Errors with Email Notification: Focusing only on 'States.Runtime' errors may not capture all types of failures that can occur in the workflow. It's more comprehensive to use 'States.ALL' to ensure no error types are missed.

Question 77

A company is setting up a new private intranet service on Amazon EC2 instances within a VPC. This service needs to be integrated with the company's existing on-premises services, which are accessible via hostnames within the "company.example" DNS zone. This DNS zone is hosted exclusively on the company's private network and is not publicly accessible. The VPC is connected to the company's on-premises network via an AWS Site-to-Site VPN. A crucial requirement is for the new intranet service on EC2 to resolve these on-premises hostnames in order to communicate with the existing services. The solutions architect is tasked with implementing a solution that enables this DNS resolution.

The proposed solutions for enabling the new service to resolve on-premises hostnames are:

A. Create Private Zone in Route 53 with Additional NS Record: Establish an empty private DNS zone in Amazon Route 53 for "company.example" and add an NS record in the on-premises DNS zone pointing to the Route 53 zone's authoritative name servers.

B. Enable DNS Hostnames and Configure Route 53 Resolver Outbound Endpoint: Activate DNS hostnames for the VPC and set up a new outbound endpoint using Amazon Route 53 Resolver. Create a Resolver rule to forward DNS requests for "company.example" to the on-premises DNS servers.

C. Activate DNS Hostnames and Set Up Route 53 Resolver Inbound Endpoint: Enable DNS hostnames in the VPC and configure a new inbound resolver endpoint with Amazon Route 53 Resolver. Adjust the on-premises DNS server to forward "company.example" requests to this new resolver.

D. Install Hosts File via AWS Systems Manager and EventBridge Rule: Utilize AWS Systems Manager to create a run document that installs a hosts file containing necessary hostnames. Set an Amazon EventBridge rule to execute this document when an EC2 instance enters the running state.

The most suitable solution is B:

- B. Enable DNS Hostnames and Configure Route 53 Resolver Outbound Endpoint: This option effectively meets the requirement by enabling DNS resolution for the EC2 instances in the VPC to query the on-premises DNS servers for "company.example" hostnames. By setting up an outbound endpoint with Route 53 Resolver and a forwarding rule, DNS queries for the specified domain are directed to the on-premises DNS servers, ensuring seamless integration with existing services.

The reasons why the other options are less suitable:

- A. Create Private Zone in Route 53 with Additional NS Record: This approach would not work as expected because the "company.example" DNS zone is not hosted in Route 53, and an empty private zone would not have the necessary records for resolution.
- C. Activate DNS Hostnames and Set Up Route 53 Resolver Inbound Endpoint: Configuring an inbound resolver is not relevant in this scenario, as the requirement is to resolve DNS queries originating from the VPC (outbound), not from the on-premises network.
- D. Install Hosts File via AWS Systems Manager and EventBridge Rule: While this might provide a workaround for static DNS entries, it is not scalable or practical for a dynamic environment, and it requires continuous updates and maintenance.

Question 78

A company employs AWS CloudFormation for deploying applications across several VPCs, all of which are connected to a transit gateway. Traffic destined for the public internet from any VPC is routed through a shared services VPC. In each VPC, subnets use the default VPC route table to route their traffic to the transit gateway, and the transit gateway itself utilizes its default route table for VPC attachments. A recent security audit uncovered that an Amazon EC2 instance in any of the company's VPCs can communicate with EC2 instances in other VPCs. To enhance security, the company now requires that each VPC should only be able to communicate with a specific set of authorized VPCs.

The solutions to limit inter-VPC traffic to only authorized VPCs are:

A. Modify Network ACLs for Subnet Traffic Restriction: Adjust the network ACL of each subnet within a VPC to permit outbound traffic exclusively to the authorized VPCs, eliminating all deny rules except the default.

B. Revise Security Groups to Restrict Outbound Traffic: Update all security groups within a VPC to block outbound traffic to the security groups associated with unauthorized VPCs.

C. Implement Dedicated Transit Gateway Route Tables: Create separate transit gateway route tables for each VPC attachment, configuring them to route traffic solely to the authorized VPCs.

D. Alter Main Route Table for Authorized VPC Traffic: Modify the main route table of each VPC to direct traffic only to the authorized VPCs via the transit gateway.

The most effective solution is C:

- C. Implement Dedicated Transit Gateway Route Tables: This approach is the most direct and effective way to control traffic flow between VPCs connected via a transit gateway. By creating a dedicated route table for each VPC attachment and configuring

routes to only include authorized VPCs, the company can precisely manage which VPCs can communicate with each other. This solution offers granular control and aligns with the requirement to restrict communication to a predefined set of VPCs.

The reasons why the other options are less suitable:

- A. Modify Network ACLs for Subnet Traffic Restriction: While network ACLs can control traffic at the subnet level, they are stateless and can be complex to manage in this context. This approach would require extensive and detailed ACL configuration and wouldn't be as efficient or easy to manage as using dedicated transit gateway route tables.
- B. Revise Security Groups to Restrict Outbound Traffic: Security groups are stateful and primarily used for instance-level traffic control. Managing inter-VPC communication through security groups would be cumbersome and not as straightforward or effective as route table manipulation.
- D. Alter Main Route Table for Authorized VPC Traffic: Updating the main route table of each VPC would not be practical in this scenario, as the main route table is used for local VPC traffic and not for controlling traffic through the transit gateway.

Question 79

A company with a Windows-based desktop application needs to integrate employees from a newly acquired company who use Linux systems. To facilitate this, the company plans to migrate and rehost the application on AWS, ensuring all employees, regardless of their operating system, can access it. Authentication for all employees is a must, and the company currently uses an on-premises Active Directory system. The goal is to find a solution that rehosts the application on AWS with minimal development effort and effectively manages access for all employees.

The proposed solutions for rehosting and managing access to the application are:

A. Amazon WorkSpaces with Amazon Cognito Identity Pools: Deploy Amazon WorkSpaces virtual desktops for each employee and use Amazon Cognito identity pools for authentication. Employees would access the application via their WorkSpaces desktops.

B. Auto Scaling Group of Windows EC2 Instances with Active Directory: Set up an Auto Scaling group of Windows-based Amazon EC2 instances, joining them to the on-premises Active Directory domain. Employees would access the application using Windows Remote Desktop.

C. Amazon AppStream 2.0 with AppStream 2.0 User Pools: Utilize Amazon AppStream 2.0 to create an image containing the application. Establish an AppStream 2.0 On-Demand fleet with dynamic scaling policies for the image, and manage authentication with AppStream 2.0

user pools. Employees would access the application via browser-based AppStream 2.0 streaming sessions.

D. Refactor for Amazon ECS and AWS Fargate with Amazon Cognito User Pools: Refactor and containerize the application for web-based access, running it on Amazon ECS with AWS Fargate. Implement step scaling policies and manage authentication using Amazon Cognito user pools. Employees would access the application through their web browsers.

The most effective solution is C:

- C. Amazon AppStream 2.0 with AppStream 2.0 User Pools: This solution is ideal for providing access to a Windows-based desktop application across different operating systems with minimal development effort. AppStream 2.0 enables the application to be streamed from AWS to any device with a web browser, eliminating the need for complex refactoring or desktop virtualization setups. AppStream 2.0 user pools simplify authentication management, allowing seamless integration with the existing Active Directory.

The reasons why the other options are less suitable:

- A. Amazon WorkSpaces with Amazon Cognito Identity Pools: While WorkSpaces can provide virtual desktops compatible with different operating systems, it would require more management overhead and might not be the most efficient way to provide access to a single application.
- B. Auto Scaling Group of Windows EC2 Instances with Active Directory: This approach involves significant management overhead and complexity, as it requires maintaining a fleet of EC2 instances and handling remote desktop sessions, which might not be optimal for Linux users.
- D. Refactor for Amazon ECS and AWS Fargate with Amazon Cognito User Pools: Refactoring the application for web-based deployment is a time-consuming process that involves significant development effort. This approach would not meet the requirement of minimal development effort.

Question 80

A company has been accumulating substantial data from its IoT device network. This data is currently saved as Optimized Row Columnar (ORC) files within the Hadoop Distributed File System (HDFS) on a continuously running Amazon EMR cluster. The data analytics team uses Apache Presto on the same EMR cluster for SQL-based querying. These queries, which scan vast amounts of data, are consistently short (under 15 minutes) and are executed only during a specific time window each day (from 5 PM to 10 PM). The company is looking to reduce the

costs associated with this setup while maintaining the ability to perform SQL queries on the data.

The suggested solutions for a more cost-effective approach to data storage and querying are:

A. Use Amazon S3 with Amazon Redshift Spectrum: Transition data storage to Amazon S3 and employ Amazon Redshift Spectrum for querying.

B. Migrate to Amazon S3 with AWS Glue Data Catalog and Amazon Athena: Shift data storage to Amazon S3, utilize AWS Glue Data Catalog for data organization, and leverage Amazon Athena for SQL querying.

C. Retain on EMR File System (EMRFS) and Query with Presto in Amazon EMR: Keep data in EMRFS and continue using Presto on Amazon EMR for querying.

D. Transfer to Amazon Redshift and Query with Redshift: Move data to Amazon Redshift and use its capabilities for data querying.

The most cost-effective solution is B:

- B. Migrate to Amazon S3 with AWS Glue Data Catalog and Amazon Athena: This option presents a highly cost-effective and efficient solution. Storing data in Amazon S3 is typically more cost-effective than persistent HDFS on EMR. Using AWS Glue Data Catalog for data cataloging and Amazon Athena for SQL querying aligns well with the company's needs. Athena's serverless architecture means the company only pays for the queries it runs, which is ideal given the limited daily time window for query execution.

The reasons why the other options are less suitable:

- A. Use Amazon S3 with Amazon Redshift Spectrum: While Redshift Spectrum allows querying data in S3, it requires an active Redshift cluster, which can be more expensive, especially if the querying needs are sporadic and concentrated within a short daily timeframe.
- C. Retain on EMR File System (EMRFS) and Query with Presto in Amazon EMR: Continuing with the current EMR-based solution would not address the concern of high costs, as maintaining a persistent EMR cluster is typically more expensive than serverless options, especially when the querying activity is limited to a few hours daily.
- D. Transfer to Amazon Redshift and Query with Redshift: Migrating to Amazon Redshift involves maintaining a data warehouse, which may not be the most cost-effective solution given the limited and short-duration querying needs. Redshift is typically more suited for continuous, heavy-duty querying and data warehousing needs.

Question 81

A major corporation has encountered a surge in costs related to Amazon RDS and Amazon DynamoDB. To enhance their understanding and management of AWS Billing and Cost Management, the company needs a more detailed insight, especially considering its diverse accounts under AWS Organizations. These include numerous development and production accounts. While there's no uniform tagging strategy across the organization, there is an existing policy mandating the use of AWS CloudFormation for infrastructure deployment, ensuring consistent tagging. The company's management now requires that all existing and new DynamoDB tables and RDS instances be associated with specific cost center numbers and project ID numbers.

The strategies to achieve this detailed cost tracking are:

A. Tag Existing Resources Using Tag Editor and Create Cost Allocation Tags: Utilize AWS Tag Editor to tag current resources. Establish cost allocation tags for cost center and project ID, allowing 24 hours for the tags to be applied to existing resources.

B. AWS Config Rule for Untagged Resource Alerts and Centralized Lambda Solution: Implement an AWS Config rule to notify the finance team about untagged resources. Develop a centralized Lambda solution that tags untagged RDS and DynamoDB resources hourly via a cross-account role.

C. Tag Existing Resources, Create Cost Allocation Tags, and Use SCPs for Enforcement: Apply tags to current resources using Tag Editor. Create cost allocation tags for cost center and project ID. Implement Service Control Policies (SCPs) in AWS Organizations to enforce the inclusion of these tags on all resource creations.

D. Create Cost Allocation Tags and Update Federated Roles for Tag Enforcement: Establish cost allocation tags for cost center and project ID, allowing 24 hours for tags to apply to existing resources. Modify existing federated roles to limit provisioning permissions to only those resources that include the necessary cost center and project ID.

The most appropriate solution is C:

- C. Tag Existing Resources, Create Cost Allocation Tags, and Use SCPs for Enforcement: This strategy offers a comprehensive solution. First, it involves tagging existing resources with the required cost center and project ID information. Secondly, it involves creating cost allocation tags for ongoing and future resource tracking. Finally, and most importantly, it utilizes SCPs to enforce tagging policies across the organization. SCPs ensure compliance by restricting resource creation that doesn't adhere to the tagging guidelines, thus providing a proactive approach to cost management and visibility.

The reasons why the other options are less suitable:

- A. Tag Existing Resources Using Tag Editor and Create Cost Allocation Tags: While this approach addresses the tagging of existing resources, it lacks a mechanism to enforce the tagging policy for new resources, which is crucial for ongoing cost management.
- B. AWS Config Rule for Untagged Resource Alerts and Centralized Lambda Solution: This solution reacts to untagged resources rather than preventing their creation. It is more reactive and may not be as effective in ensuring compliance with the tagging policy.
- D. Create Cost Allocation Tags and Update Federated Roles for Tag Enforcement: This option also addresses the tagging of resources, but updating federated roles to enforce tagging can be complex and may not be as effective or encompassing as using SCPs within AWS Organizations.

Question 82

A company needs a secure way to transfer data from its on-premises systems to Amazon S3 buckets located in three different AWS accounts. The data transfer must be private, avoiding any transit over the public internet, and the company currently lacks dedicated AWS connectivity infrastructure.

To achieve this, the following steps are considered:

A. Set Up AWS Direct Connect with Private VIF in a Dedicated Networking Account: Establish a dedicated networking account in AWS, create a private Virtual Private Cloud (VPC) within it, and then set up an AWS Direct Connect connection with a private Virtual Interface (VIF) linking the on-premises environment to this private VPC.

B. Establish AWS Direct Connect with Public VIF in a Dedicated Networking Account: Create a networking account in AWS, build a private VPC within it, and install an AWS Direct Connect connection with a public VIF between the on-premises setup and the private VPC.

C. Create an Amazon S3 Interface Endpoint in the Networking Account: In the networking account, deploy an Amazon S3 interface endpoint within the private VPC.

D. Implement an Amazon S3 Gateway Endpoint in the Networking Account: Set up an Amazon S3 gateway endpoint in the networking account's private VPC.

E. VPC Peering Between S3 Hosting Accounts and Networking Account: Create a private VPC in a dedicated networking account and establish VPC peering connections with the VPCs in the accounts that contain the S3 buckets.

The most appropriate combination of steps is A and C:

- A. Set Up AWS Direct Connect with Private VIF in a Dedicated Networking Account: This step provides a private, dedicated network connection from the company's on-premises environment to AWS, ensuring that data does not traverse the public internet. Using a private VIF allows for secure, private access to AWS services, including Amazon S3, within the AWS network.
- C. Create an Amazon S3 Interface Endpoint in the Networking Account: By setting up an S3 interface endpoint (AWS PrivateLink) in the networking account's VPC, the company ensures that data transferred to S3 buckets in different accounts can route securely over the AWS network, maintaining privacy and avoiding internet exposure.

The reasons why the other options are less suitable:

- B. Establish AWS Direct Connect with Public VIF in a Dedicated Networking Account: Using a public VIF would not meet the requirement of keeping data off the public internet, as it routes traffic over the AWS public network.
- D. Implement an Amazon S3 Gateway Endpoint in the Networking Account: S3 gateway endpoints are used to route traffic within the same VPC to S3. In this scenario, it would not facilitate cross-account access to S3 buckets.
- E. VPC Peering Between S3 Hosting Accounts and Networking Account: VPC peering is not necessary in this scenario, especially since Direct Connect and PrivateLink provide the needed connectivity without the need for peering VPCs across accounts.

Question 83

A quick-service restaurant company experiences high sales volume for 4 hours daily, with lower traffic at other times. Their point of sale and management system, hosted on AWS, relies on an Amazon DynamoDB backend. The DynamoDB table is currently configured with provisioned throughput, set at 100,000 Read Capacity Units (RCUs) and 80,000 Write Capacity Units (WCUs), aligned with their peak resource demands. The company is looking for a more cost-effective solution for their DynamoDB usage that also minimizes the operational burden on their IT staff.

The proposed solutions for optimizing DynamoDB costs are:

A. Manually Reduce Provisioned RCUs and WCUs: Decrease the number of provisioned RCUs and WCUs on the DynamoDB table.

B. Switch to DynamoDB On-Demand Capacity: Convert the DynamoDB table to use the on-demand capacity mode.

C. Implement DynamoDB Auto Scaling: Activate auto scaling for the DynamoDB table to automatically adjust RCUs and WCUs based on actual usage.

D. Purchase 1-Year Reserved Capacity for Peak Load: Buy reserved capacity for DynamoDB to cover the peak load for 4 hours each day for a year.

The most cost-effective solution is C:

- C. Implement DynamoDB Auto Scaling: Auto scaling dynamically adjusts the provisioned throughput (RCUs and WCUs) in response to actual usage patterns, scaling up during peak hours and scaling down during off-peak times. This approach effectively manages resource utilization to align with demand, potentially reducing costs significantly, especially when traffic is variable. It also reduces operational overhead as it automates the scaling process.

The reasons why the other options are less suitable:

- A. Manually Reduce Provisioned RCUs and WCUs: Manually adjusting RCUs and WCUs can be less efficient and risk throttling during unexpected surges in demand. It also increases operational overhead as it requires constant monitoring and adjustment.
- B. Switch to DynamoDB On-Demand Capacity: While on-demand capacity is simple and scales automatically, it is generally more expensive than provisioned capacity for predictable workloads. It might not be the most cost-effective choice for a business with consistent peak periods.
- D. Purchase 1-Year Reserved Capacity for Peak Load: Buying reserved capacity for the peak load might not be the most efficient option, as it does not account for the lower demand during off-peak hours. This could lead to paying for unused capacity.

Question 84

A company operates a blog post application on AWS, utilizing Amazon API Gateway, Amazon DynamoDB, and AWS Lambda. The application, which currently doesn't implement API keys for request authorization, includes features like fetching post details, user details, and comments through specific API endpoints (GET /posts/{postId}, GET /users/{userId}, GET /comments/{commentId}). Noticing increased user activity in the comments section, the company aims to enhance user engagement by enabling real-time display of comments, thereby reducing latency and improving the user experience.

The potential solutions for achieving real-time comments display are:

A. Implement Edge-Optimized API with Amazon CloudFront Caching: Use an edge-optimized API along with Amazon CloudFront to cache API responses.

B. **Periodic Polling for Comments**: Update the blog application to make requests to GET /comments/{commentId} every 10 seconds to fetch new comments.

C. **Integrate AWS AppSync with WebSockets**: Employ AWS AppSync and utilize WebSockets to deliver comments in real-time.

D. **Adjust Lambda Function Concurrency Limit**: Modify the concurrency limit of the Lambda functions to decrease API response times.

The most suitable solution is C:

- **C. Integrate AWS AppSync with WebSockets**: AWS AppSync with WebSockets is an excellent choice for real-time data delivery, such as displaying comments as they are posted. AppSync enables efficient, real-time synchronization and updates, providing a seamless user experience without the need for frequent API calls or manual refreshes.

The reasons why the other options are less suitable:

- **A. Implement Edge-Optimized API with Amazon CloudFront Caching**: While this could improve the performance of API responses, caching is not suitable for real-time data like comments, as it would not reflect the most recent data instantly.
- **B. Periodic Polling for Comments**: Regular polling every 10 seconds increases the load on the server and can still lead to a delay in displaying new comments. This approach is less efficient compared to real-time updates through WebSockets.
- **D. Adjust Lambda Function Concurrency Limit**: Changing the concurrency limit of Lambda functions may improve response times but does not address the need for real-time updates. It also doesn't guarantee reduced latency for end-users in the context of fetching new comments.

Question 85

A company overseeing numerous AWS accounts within AWS Organizations has recently permitted its product teams to independently create and manage S3 access points in their respective accounts. However, the company has mandated that these S3 access points must be accessible exclusively within Virtual Private Clouds (VPCs) and not over the internet. The challenge lies in implementing this policy efficiently across all the accounts in the organization.

The proposed methods for ensuring that S3 access points are only accessible within VPCs are:

A. **Modify S3 Access Point Resource Policy for VPC-Only Access**: Adjust the resource policy on each S3 access point to deny the `s3:CreateAccessPoint` action unless the

`s3:AccessPointNetworkOrigin` condition key is set to "VPC".

B. Implement an SCP at the Organization's Root Level for VPC-Only Access: Create a Service Control Policy (SCP) at the root level of the organization that denies the `s3:CreateAccessPoint` action unless the `s3:AccessPointNetworkOrigin` condition key is set to "VPC".

C. Use AWS CloudFormation StackSets for IAM Policy Creation in Each Account: Employ AWS CloudFormation StackSets to generate a new IAM policy in every AWS account, which allows the `s3:CreateAccessPoint` action only if the `s3:AccessPointNetworkOrigin` condition key is set to "VPC".

D. Adjust S3 Bucket Policy for VPC-Only Access Point Creation: Modify the policy of the S3 buckets to deny the `s3:CreateAccessPoint` action unless the `s3:AccessPointNetworkOrigin` condition key indicates "VPC".

The most operationally efficient solution is B:

- B. Implement an SCP at the Organization's Root Level for VPC-Only Access: Implementing an SCP at the organization's root level is the most efficient way to enforce this requirement across all accounts. SCPs allow central management and enforcement of policies across the entire organization. By setting this policy at the root level, it ensures that all current and future accounts within the organization will comply with the requirement that S3 access points must be VPC-only, without the need for individual account-level configurations.

The reasons why the other options are less suitable:

- A. Modify S3 Access Point Resource Policy for VPC-Only Access: This approach requires changes to be made individually on each S3 access point across all accounts, which is operationally cumbersome and inefficient for a large number of accounts.
- C. Use AWS CloudFormation StackSets for IAM Policy Creation in Each Account: While StackSets can automate the policy creation across multiple accounts, this method is more complex and less direct than using an SCP. It also requires maintenance of the StackSets and does not provide a centralized control mechanism like SCPs.
- D. Adjust S3 Bucket Policy for VPC-Only Access Point Creation: Modifying the S3 bucket policy for each bucket in every account is operationally inefficient and prone to errors, especially when dealing with a large number of buckets across multiple accounts.

Question 86

A solutions architect is tasked with updating an application hosted on AWS Elastic Beanstalk using a blue/green deployment strategy. To do this, the architect has set up a new

environment within Elastic Beanstalk that mirrors the current (blue) environment and has successfully deployed the updated application to this new (green) environment. The next step is to transition from the old to the new environment.

The potential next steps in this process are:

A. Implement Traffic Redirection using Amazon Route 53: Use Amazon Route 53 to redirect traffic from the old environment to the new one.

B. Utilize the Swap Environment URLs Feature: Employ Elastic Beanstalk's built-in 'Swap Environment URLs' functionality.

C. Update the Auto Scaling Launch Configuration: Change the Auto Scaling group's launch configuration to direct new instances to the new environment.

D. Manually Update DNS Records to the Green Environment: Modify the DNS records to redirect traffic to the new (green) environment.

The correct next step is B:

- B. Utilize the Swap Environment URLs Feature: Elastic Beanstalk provides a seamless way to switch traffic between two environments with its 'Swap Environment URLs' feature. This action swaps the CNAMEs (Canonical Names) of the two environments, redirecting traffic from the old (blue) environment to the new (green) one without the need for DNS changes. This method ensures a quick and smooth transition with minimal downtime.

The reasons why the other options are less suitable:

- A. Implement Traffic Redirection using Amazon Route 53: While Route 53 can be used to redirect traffic, it's not the most efficient method in this context. Using Elastic Beanstalk's built-in URL swapping feature is a simpler and more integrated solution for blue/green deployments within Elastic Beanstalk.
- C. Update the Auto Scaling Launch Configuration: Changing the Auto Scaling launch configuration does not directly affect traffic routing between the two environments. It's more relevant to scaling strategies rather than to blue/green deployments.
- D. Manually Update DNS Records to the Green Environment: Manually updating DNS records is less efficient and can lead to longer propagation delays. The 'Swap Environment URLs' feature in Elastic Beanstalk provides a quicker and more reliable method for switching traffic between environments.

Question 87

A company is developing a web-based image service where up to 10,000 users globally can upload photos. These photos will have text overlaid by the application and then be displayed on the company's website. The company needs a scalable and efficient architecture for storing, processing, and serving these images.

The potential architectural designs to consider are:

A. Use Amazon EFS for Image Storage and CloudWatch Logs for Processing: Images are stored in Amazon Elastic File System (Amazon EFS). Image upload events are logged to Amazon CloudWatch Logs. A group of Amazon EC2 instances processes the images based on CloudWatch Logs and stores processed images in a different EFS directory. Amazon CloudFront is enabled with an EC2 instance as the origin.

B. S3 Bucket with SNS Notifications and EC2 Fleet Behind ALB: Images are uploaded to an Amazon S3 bucket, triggering S3 event notifications to Amazon Simple Notification Service (Amazon SNS). A fleet of EC2 instances behind an Application Load Balancer (ALB) processes the images based on SNS messages and stores them in Amazon EFS. CloudWatch metrics for SNS are used to scale the EC2 instances. Amazon CloudFront is enabled with the ALB as the origin.

C. S3 Bucket with SQS Queue and EC2 Fleet for Image Processing: Images are stored in an Amazon S3 bucket, with S3 event notifications sent to an Amazon Simple Queue Service (Amazon SQS) queue. A fleet of EC2 instances processes the images based on messages from the SQS queue and stores processed images in another S3 bucket. Amazon CloudWatch metrics for the SQS queue depth are used to scale the EC2 instances. Amazon CloudFront is configured with the S3 bucket containing processed images as the origin.

D. Shared EBS Volume with EC2 Spot Instances and DynamoDB for Tracking: Images are stored on a shared Amazon Elastic Block Store (Amazon EBS) volume mounted to EC2 Spot instances. An Amazon DynamoDB table tracks each image's upload and processing status. Amazon EventBridge rules scale the EC2 instances. Amazon CloudFront is enabled with an Elastic Load Balancer (ELB) as the origin.

The most suitable solution is C:

- C. S3 Bucket with SQS Queue and EC2 Fleet for Image Processing: This design effectively leverages S3 for scalable storage and SQS for managing the processing queue. The use of SQS ensures reliable delivery and processing of image upload events, while CloudWatch metrics allow for dynamic scaling of EC2 instances based on the queue depth. Storing processed images in another S3 bucket simplifies management and integrates seamlessly with CloudFront for content delivery, offering a scalable and efficient solution.

The reasons why the other options are less suitable:

- A. Use Amazon EFS for Image Storage and CloudWatch Logs for Processing: This approach is less scalable and efficient compared to using S3 and SQS. EFS is not as optimized for object storage and high-throughput scenarios as S3, and using CloudWatch Logs for image processing triggers is unconventional and less efficient.
- B. S3 Bucket with SNS Notifications and EC2 Fleet Behind ALB: While this approach uses S3 for initial storage, the use of SNS for processing triggers is less efficient than SQS for queue management. Additionally, storing processed images in EFS instead of S3 is not optimal for scalability and integration with CloudFront.
- D. Shared EBS Volume with EC2 Spot Instances and DynamoDB for Tracking: This design is complex and less scalable. Using a shared EBS volume for image storage and DynamoDB for tracking is operationally cumbersome and does not leverage the benefits of S3 and SQS for storage and queue management.

Question 88

A company has its database hosted on an Amazon RDS for MySQL instance in the us-east-1 Region and needs to provide access to this data for customers in Europe without latency issues or outdated data. Both European and US customers require the ability to write to the database, with the necessity of real-time updates visible to all users.

The proposed solutions to achieve this are:

A. Migrate to Amazon Aurora with Cross-Region Deployment: Transition from RDS for MySQL to an Amazon Aurora MySQL replica. Convert this replica into a standalone DB cluster. Enable write forwarding and add eu-west-1 as a secondary region to the Aurora cluster. Deploy the application in eu-west-1 and point it to the eu-west-1 Aurora endpoint.

B. Create Cross-Region RDS Replica with Bidirectional Replication: Establish a cross-region replica of the RDS for MySQL DB instance in eu-west-1 and set up bidirectional replication. Deploy the application in eu-west-1, using the eu-west-1 RDS endpoint.

C. Snapshot and Replicate RDS Instance to eu-west-1: Take a snapshot of the RDS for MySQL DB instance, copy it to eu-west-1, and create a new RDS instance from this snapshot. Set up logical replication between the two regions and enable write forwarding. Deploy the application in eu-west-1, using the new RDS endpoint.

D. Convert to Amazon Aurora and Enable Cross-Region Deployment: Convert the existing RDS for MySQL instance to an Amazon Aurora MySQL DB cluster. Add eu-west-1 as a secondary region to the Aurora cluster and enable write forwarding. Deploy the application in eu-west-1 and configure it to use the Aurora endpoint in this region.

The most suitable solution is D:

- D. Convert to Amazon Aurora and Enable Cross-Region Deployment: Converting the existing RDS for MySQL to an Amazon Aurora MySQL DB cluster and adding a secondary region (eu-west-1) is the best approach. Aurora supports cross-region deployments and write forwarding, which allows real-time data availability and writing capabilities in multiple regions. This setup ensures low latency and up-to-date data access for users in both the US and Europe.

The reasons why the other options are less suitable:

- A. Migrate to Amazon Aurora with Cross-Region Deployment: While this solution involves moving to Aurora, the process of promoting the Aurora Replica to a standalone cluster and then adding a secondary region is more complex and might lead to data synchronization issues.
- B. Create Cross-Region RDS Replica with Bidirectional Replication: RDS for MySQL doesn't natively support bidirectional replication across regions. This would require a complex and custom replication setup, which is not recommended for production environments due to potential data conflicts.
- C. Snapshot and Replicate RDS Instance to eu-west-1: This approach involves manual steps for replication and does not provide a seamless, real-time replication solution. Additionally, managing logical replication and write forwarding across regions can be operationally challenging.

Question 89

A company is currently offering file services to its customers through an SFTP server hosted on an Amazon EC2 instance with an Elastic IP address. The server is accessible over the internet using SSH for authentication, and its security group permits access from all customer IP addresses. The company seeks a solution to enhance the availability and simplify infrastructure management without disrupting the current access method for its customers.

The possible solutions for this requirement are:

A. Migrate to AWS Transfer Family with Public Endpoint: Transition from the EC2 instance to an AWS Transfer Family server with a publicly accessible endpoint. Associate the SFTP server's Elastic IP address with the new Transfer Family endpoint and direct it to an Amazon S3 bucket for file hosting. Transfer all files from the existing SFTP server to the S3 bucket.

B. Migrate to AWS Transfer Family with VPC-Hosted Endpoint: Move to an AWS Transfer Family server with a VPC-hosted, internet-facing endpoint. Disassociate the Elastic IP address from the EC2 instance and associate it with the new Transfer Family endpoint. Use the same security group that includes customer IP addresses for the new endpoint. Point the

Transfer Family server to an Amazon S3 bucket and synchronize files from the EC2-based SFTP server to the S3 bucket.

C. Use AWS Fargate with EFS and Network Load Balancer: Replace the EC2 instance with an AWS Fargate task definition running an SFTP server, using Amazon Elastic File System (EFS) for file storage. Set up a Network Load Balancer (NLB) in front of the Fargate service, re-associate the Elastic IP with the NLB, and use the existing security group. Transfer files from the SFTP server to EFS.

D. Implement Multi-Attach EBS with Auto Scaling and NLB: Disassociate the Elastic IP from the EC2 instance and create a multi-attach Amazon EBS volume for file hosting. Set up a Network Load Balancer (NLB) with the Elastic IP, create an Auto Scaling group of EC2 instances running an SFTP server using the multi-attach EBS volume, and configure the Auto Scaling group to use the existing security group. Synchronize files from the EC2-based SFTP server to the EBS volume.

The best solution is B:

- B. Migrate to AWS Transfer Family with VPC-Hosted Endpoint: This approach effectively uses AWS Transfer Family, which is a managed service specifically designed for SFTP operations and eliminates the need for managing underlying infrastructure. The VPC-hosted, internet-facing endpoint provides secure access to customers, while associating the existing Elastic IP maintains the current method of connection. Using Amazon S3 for file storage offers scalability and high availability, and the existing security group ensures continued access control.

The reasons why the other options are less suitable:

- A. Migrate to AWS Transfer Family with Public Endpoint: While this uses AWS Transfer Family, a publicly accessible endpoint does not leverage the VPC's security features, potentially exposing the service to broader security risks compared to a VPC-hosted endpoint.
- C. Use AWS Fargate with EFS and Network Load Balancer: This solution introduces unnecessary complexity by deploying a containerized SFTP server with Fargate and EFS, which is more complex than using a managed service like AWS Transfer Family.
- D. Implement Multi-Attach EBS with Auto Scaling and NLB: This option is operationally complex and less reliable than using a managed SFTP service. Managing a cluster of SFTP servers with multi-attach EBS and ensuring synchronization across instances is challenging and not as scalable or reliable as AWS Transfer Family.

Question 90

A company is managing streaming market data and needs a cost-effective solution for processing this data. The data flow is steady, and a nightly batch process computes aggregate statistics over four hours. The statistical analysis, although not business-critical, should ideally continue seamlessly if a run fails. They currently use a set of Amazon EC2 Reserved Instances for continuous data ingestion and storage, with nightly batch processing done on EC2 On-Demand Instances. As the Reserved Instance reservations are about to expire, the company is evaluating whether to renew these reservations or adopt a new architectural approach.

Here are the possible solutions:

A. Shift to Kinesis Data Firehose and EC2 On-Demand Instances for Nightly Processing: Revise the data ingestion process to utilize Amazon Kinesis Data Firehose for directing data to Amazon S3. Implement a script to launch EC2 On-Demand Instances nightly for batch processing of the data in S3, terminating these instances post-processing.

B. Use Kinesis Data Firehose for Ingestion and AWS Batch with Spot Instances for Processing: Transition the data ingestion to Amazon Kinesis Data Firehose, storing data in Amazon S3. Employ AWS Batch for nightly processing, utilizing EC2 Spot Instances with a maximum bidding price set at 50% of the On-Demand rate.

C. Retain EC2 Reserved Instances for Ingestion and Use AWS Batch with Spot Instances for Processing: Maintain data ingestion using a new fleet of EC2 Reserved Instances (with 3-year reservations) behind a Network Load Balancer. Utilize AWS Batch with EC2 Spot Instances for nightly processing, setting the maximum Spot price at 50% of the On-Demand rate.

D. Ingest Data via Kinesis Data Firehose to Amazon Redshift and Use Lambda for Nightly Statistics: Alter the ingestion process to funnel data through Amazon Kinesis Data Firehose into Amazon Redshift. Schedule an AWS Lambda function using Amazon EventBridge to run nightly queries on Redshift for generating daily statistics.

The most cost-effective solution is B:

- B. Use Kinesis Data Firehose for Ingestion and AWS Batch with Spot Instances for Processing: This option provides a significant cost advantage by replacing EC2 Reserved Instances with a serverless ingestion solution (Kinesis Data Firehose) and using Amazon S3 for storage. AWS Batch with Spot Instances for nightly processing offers flexibility and cost savings, as Spot Instances are often available at lower prices than On-Demand Instances. The 50% cap on the Spot price prevents cost overruns.

The reasons why the other options are less suitable:

- A. Shift to Kinesis Data Firehose and EC2 On-Demand Instances for Nightly Processing: While using Kinesis Data Firehose and Amazon S3 improves the ingestion process, relying on EC2 On-Demand Instances for nightly processing may not be as cost-effective as using Spot Instances.
- C. Retain EC2 Reserved Instances for Ingestion and Use AWS Batch with Spot Instances for Processing: Continuing with EC2 Reserved Instances for data ingestion is less efficient compared to serverless options like Kinesis Data Firehose. The upfront commitment to Reserved Instances may also be unnecessary given the evolving cloud landscape.
- D. Ingest Data via Kinesis Data Firehose to Amazon Redshift and Use Lambda for Nightly Statistics: Using Amazon Redshift for storage and Lambda for processing could be an overkill for the use case, potentially leading to higher costs compared to the simpler S3 and AWS Batch combination.

Question 91

A company is migrating its on-premises SFTP site, which runs on a Linux VM and shares uploaded files via an NFS share, to AWS. The migration requires high availability and the ability to provide external vendors with static public IP addresses. The company already has an AWS Direct Connect connection between its on-premises data center and its VPC. The goal is to find a solution that minimizes operational complexity.

Here are the rephrased solution options:

A. Implement AWS Transfer Family Server with VPC Endpoint and Elastic IP: Set up an AWS Transfer Family server, using an internet-facing VPC endpoint configuration. Assign specific Elastic IP addresses for each subnet. Direct the Transfer Family server to store files in an Amazon Elastic File System (EFS) that spans multiple Availability Zones. Update the downstream applications to access files via the EFS mount point, replacing the existing NFS share.

B. Set Up AWS Transfer Family Server with Public Endpoint and EFS Storage: Establish an AWS Transfer Family server with a publicly accessible endpoint. Configure the server to store files in an Amazon EFS file system spread across multiple Availability Zones. Update the downstream applications to mount the EFS endpoint instead of the current NFS share.

C. Migrate Linux VM to EC2 and Use EFS for Storage: Utilize AWS Application Migration Service to move the existing Linux VM to an Amazon EC2 instance. Assign an Elastic IP address to this EC2 instance. Attach an Amazon EFS file system to the EC2 instance and configure the SFTP server to save files to EFS. Adapt the downstream applications to use the EFS mount point instead of the current NFS share.

D. Migrate Linux VM to AWS Transfer Family Server with FSx for Lustre: Migrate the existing Linux VM to an AWS Transfer Family server using AWS Application Migration Service. Set up a publicly accessible endpoint for the Transfer Family server. Direct the server to store files in an Amazon FSx for Lustre file system that spans multiple Availability Zones. Alter the downstream applications to mount the FSx for Lustre endpoint, replacing the current NFS share.

The most suitable solution is A:

- A. Implement AWS Transfer Family Server with VPC Endpoint and Elastic IP: This option efficiently achieves high availability and complies with the need for static public IP addresses via Elastic IP addresses. Utilizing AWS Transfer Family simplifies the operational overhead, and Amazon EFS ensures high availability and scalability. The transition for downstream applications from NFS to EFS is straightforward, minimizing disruption.

Reasons why the other options are less appropriate:

- B. Set Up AWS Transfer Family Server with Public Endpoint and EFS Storage: While this option also uses AWS Transfer Family and EFS, it doesn't specify the use of Elastic IPs, which is a requirement for providing static public IP addresses to external vendors.
- C. Migrate Linux VM to EC2 and Use EFS for Storage: This approach involves more operational complexity by continuing to manage an EC2 instance. Also, it doesn't address the requirement for static public IP addresses.
- D. Migrate Linux VM to AWS Transfer Family Server with FSx for Lustre: Migrating to FSx for Lustre adds unnecessary complexity and may not be the most cost-effective or simplest solution for SFTP file storage compared to EFS. This option also lacks the specification of Elastic IPs.

Question 92

A solutions architect is working with an operational workload that involves Amazon EC2 instances in an Auto Scaling group, spread across two Availability Zones (AZs) within a VPC. The VPC is connected to an on-premises network, and maintaining this connectivity is crucial. The Auto Scaling group can have up to 20 instances running simultaneously. The VPC and subnets have the following IPv4 address configurations:

- VPC CIDR: 10.0.0.0/23
- AZ1 subnet CIDR: 10.0.0.0/24
- AZ2 subnet CIDR: 10.0.1.0/24

With the availability of a third AZ in the region, the architect aims to include this new AZ without altering the existing IPv4 address space or causing service downtime. Here are the rephrased solution options:

A. Sequential Subnet Reconfiguration and Expansion: Modify the Auto Scaling group to only target the AZ2 subnet initially. Redefine the AZ1 subnet with half its original address range and reintroduce it to the Auto Scaling group. After confirming instance health, switch the Auto Scaling group exclusively to the new AZ1 subnet. Resize the AZ2 subnet to use the remaining addresses from the original AZ1 range. Create a new AZ3 subnet using half the addresses from the original AZ2 range, and finally update the Auto Scaling group to target all three newly configured subnets.

B. Subnet Replacement and Sequential Integration: Terminate EC2 instances in AZ1, then recreate the AZ1 subnet with a reduced address range. Update the Auto Scaling group to use the new AZ1 subnet. Repeat these steps for AZ2. Create a new subnet in AZ3 and update the Auto Scaling group to include all three new subnets.

C. Create a New VPC with Original Address Space: Establish a new VPC with the same IPv4 address range and set up three subnets, one in each AZ. Update the existing Auto Scaling group to target the new subnets in the new VPC.

D. In-Place Subnet Update and Redistribution: Adjust the Auto Scaling group to only use the AZ2 subnet. Resize the AZ1 subnet to utilize half of its original address space and reintroduce it to the Auto Scaling group. After ensuring instance health, switch the Auto Scaling group back to the newly resized AZ1 subnet. Update the AZ2 subnet with the remaining addresses from the original AZ1 subnet. Create a new AZ3 subnet using half of the original AZ2 range, and update the Auto Scaling group to include all three updated subnets.

The most suitable solution is A:

- A. Sequential Subnet Reconfiguration and Expansion: This approach enables the expansion into the new AZ without service interruption or changes to the existing IPv4 space. By sequentially adjusting and redistributing the subnet address spaces, it ensures that the existing VPC setup remains intact while accommodating the additional AZ.

Reasons why the other options are less appropriate:

- B. Subnet Replacement and Sequential Integration: This method involves terminating EC2 instances, which could lead to service interruptions, contrary to the requirement of maintaining continuous service.
- C. Create a New VPC with Original Address Space: Establishing a new VPC introduces significant changes and potential complexities, which might affect the existing on-premises connectivity.

- D. In-Place Subnet Update and Redistribution: While this option attempts to redistribute the subnet address spaces, it lacks the sequential approach of option A that ensures continuous service availability during the process.

Question 93

A company, utilizing AWS Organizations for managing its AWS accounts, deploys its infrastructure using AWS CloudFormation. The finance team, aiming to develop a chargeback model, requested that resources be tagged with specific project values. However, inconsistencies in project tag values were observed when the team analyzed the AWS Cost and Usage Report through AWS Cost Explorer.

The company seeks an efficient method to enforce the application of project tags on newly created resources. The potential solutions include:

A. Organization-Wide Tag Policy with Service Control Policy (SCP) Enforcement: Implement a tag policy in the organization's management account specifying allowed project tag values. Establish an SCP that restricts the `cloudformation:CreateStack` API call unless a project tag is present, and apply this SCP across all Organizational Units (OUs).

B. Individual OU Tag Policies with SCP Enforcement: Create a tag policy for each OU, detailing permitted project tag values. Introduce an SCP that blocks the `cloudformation:CreateStack` API call if a project tag is missing, and attach this SCP to each respective OU.

C. IAM Policy Assignment to Users: Develop a tag policy in the AWS management account with approved project tag values. Create an IAM policy that denies the `cloudformation:CreateStack` API operation in the absence of a project tag, and assign this policy to every user in the organization.

D. AWS Service Catalog Management with TagOptions Library: Utilize AWS Service Catalog to administer CloudFormation stacks as catalog products. Employ a TagOptions library for managing project tag values and share this portfolio with all OUs within the organization.

The most effective solution is A:

- A. Organization-Wide Tag Policy with SCP Enforcement: This approach is streamlined and minimizes effort by establishing a uniform tag policy at the management account level, applicable organization-wide. The SCP ensures compliance by conditioning resource creation on the presence of required project tags, reinforcing the tag policy across all OUs.

Reasons why the other options are less suitable:

- B. Individual OU Tag Policies with SCP Enforcement: While this option is similar to A, it requires redundant effort to set up and manage individual tag policies for each OU, increasing operational complexity.
- C. IAM Policy Assignment to Users: This method is less efficient as it involves assigning policies to individual users, which is more labor-intensive and less centralized compared to an organization-wide approach.
- D. AWS Service Catalog Management with TagOptions Library: Managing CloudFormation stacks through the Service Catalog adds unnecessary complexity for the sole purpose of enforcing tagging, making it less efficient compared to a direct policy enforcement strategy.

Question 94

An application operates on Amazon EC2 instances within an Auto Scaling group. However, these instances are not being fully utilized in terms of CPU and memory. To optimize costs and improve utilization, a solutions architect needs to find a solution that minimizes the need for future configuration changes.

The potential solutions include:

A. Utilize Multiple Instance Types: Identify instance types with characteristics similar to the current ones and update the Auto Scaling group's launch template to include a variety of these instance types.

B. Select a More Suitable Instance Type: Analyze CPU and memory usage data to choose a more fitting instance type. Update the Auto Scaling group by adding this new type and removing the old one.

C. Specify CPU and Memory Requirements in Launch Template: Based on CPU and memory usage, define the necessary CPU and memory requirements in a new version of the Auto Scaling group's launch template and eliminate the current instance type from the configuration.

D. Automated Instance Type Selection via Scripting: Develop a script to choose appropriate instance types from the AWS Price List Bulk API and use these selections to create a new version of the Auto Scaling group's launch template.

The most effective solution is C:

- C. Specify CPU and Memory Requirements in Launch Template: This approach directly addresses underutilization by allowing the Auto Scaling group to automatically select the most appropriate instance types based on specified CPU and memory requirements. This

reduces costs and increases efficiency without necessitating frequent manual adjustments or specific instance type selection.

Reasons why the other options are less suitable:

- A. Utilize Multiple Instance Types: While diversifying instance types can provide flexibility, it doesn't specifically address the underutilization issue and might require future adjustments as application needs evolve.
- B. Select a More Suitable Instance Type: Manually selecting a new instance type might optimize current utilization, but it does not offer the same level of future adaptability as specifying requirements in the launch template.
- D. Automated Instance Type Selection via Scripting: This method introduces unnecessary complexity by relying on external scripts and frequent updates, which is more labor-intensive compared to specifying requirements in the launch template.

Question 95

A company has deployed a containerized application using Amazon Elastic Container Service (Amazon ECS) and Amazon API Gateway. The application's data is stored in Amazon Aurora and Amazon DynamoDB databases. Infrastructure provisioning is automated with AWS CloudFormation, and application deployment is managed through AWS CodePipeline. The company is looking to establish a disaster recovery (DR) strategy with a Recovery Point Objective (RPO) of 2 hours and a Recovery Time Objective (RTO) of 4 hours, seeking the most cost-effective solution.

The possible solutions include:

A. Global Database Replication with CloudFront Failover: Implement Aurora global databases and DynamoDB global tables for cross-region replication. Use API Gateway with Regional endpoints in both primary and secondary regions, and employ Amazon CloudFront with origin failover for traffic routing during a DR event.

B. AWS DMS and Lambda Replication with Route 53 Failover: Utilize AWS Database Migration Service (AWS DMS), Amazon EventBridge, and AWS Lambda for replicating Aurora databases, and DynamoDB Streams with Lambda for DynamoDB replication to a secondary region. Configure API Gateway with Regional endpoints in both regions and use Amazon Route 53 failover routing to redirect traffic.

C. AWS Backup with Route 53 Failover: Employ AWS Backup for backing up Aurora and DynamoDB databases to a secondary region. Configure API Gateway with Regional endpoints in both primary and secondary regions, and use Amazon Route 53 failover routing for traffic redirection.

D. Global Database Replication with Route 53 Failover: Set up Aurora global databases and DynamoDB global tables for replication to a secondary region. Configure API Gateway with Regional endpoints in both regions, and use Amazon Route 53 failover routing for traffic redirection.

The most cost-effective solution is C:

- C. AWS Backup with Route 53 Failover: This approach aligns with the required RPO and RTO by using AWS Backup to periodically back up the databases to a secondary region, ensuring data recovery within the specified 2-hour RPO. The use of Amazon Route 53 failover routing for traffic redirection allows for quick recovery within the 4-hour RTO, providing a cost-effective and efficient DR strategy.

Reasons why the other options are less suitable:

- A & D. Global Database Replication: While these options provide real-time replication, they are likely more expensive than necessary for meeting the specified RPO and RTO. Continuous replication may exceed the company's cost and complexity requirements for a 2-hour RPO.
- B. AWS DMS and Lambda Replication: This method introduces additional complexity and potential for higher operational costs. The use of AWS DMS and Lambda for replication is more elaborate and might not be as cost-effective compared to the simplicity of AWS Backup.

Question 96

A company utilizes Amazon CloudFront for its web application to enhance global scalability and performance. However, users have been experiencing a slowdown in the application. The operations team has noticed a decreasing cache hit ratio in CloudFront. They have identified that the query strings in URLs are inconsistently ordered and vary in case (sometimes mixed-case, sometimes lowercase), affecting the cache efficiency.

To address this issue and improve the cache hit ratio, the following actions are proposed:

A. Lambda@Edge for Query String Normalization: Implement a Lambda@Edge function that normalizes query string parameters by sorting them by name and converting them to lowercase. This function should be triggered on CloudFront viewer requests.

B. Disable Caching Based on Query Strings: Modify the CloudFront distribution settings to stop caching based on query string parameters.

C. Deploy a Reverse Proxy for URL Processing: Set up a reverse proxy after the load balancer to post-process URLs in the application, ensuring that all URL strings are converted

to lowercase.

D. Case-Insensitive Query String Processing in CloudFront: Update the CloudFront distribution to handle query strings in a case-insensitive manner.

The most suitable solution is A:

- A. Lambda@Edge for Query String Normalization: This approach directly addresses the identified issue by standardizing the format of query strings, thus improving cache hits. Lambda@Edge allows for the quick processing and normalization of query strings at the edge locations, which increases the cache hit ratio without requiring changes to the application itself.

Reasons why the other options are less suitable:

- B. Disable Caching Based on Query Strings: While this would eliminate the issue with varying query strings, it could significantly reduce the overall effectiveness of caching, leading to decreased performance and potentially increased load on the origin server.
- C. Deploy a Reverse Proxy for URL Processing: This solution involves additional infrastructure and complexity. It would require maintenance and could introduce latency, as it processes URLs after they pass through the load balancer but before reaching CloudFront.
- D. Case-Insensitive Query String Processing in CloudFront: Currently, CloudFront does not provide a built-in feature to process query strings in a case-insensitive manner. Therefore, this option is not feasible without such a feature.

Question 97

An ecommerce company operates its application in a single AWS Region, utilizing a five-node Amazon Aurora MySQL DB cluster for customer and order data. The database is subject to high volumes of write transactions daily. To fulfill disaster recovery obligations with a Recovery Point Objective (RPO) of one hour, the company seeks to replicate the Aurora database data to a different Region in a cost-effective manner.

The proposed solutions to achieve this are:

A. Transform to Aurora Global Database: Convert the existing Aurora database into an Aurora global database and establish a secondary Aurora database in a different Region.

B. Enable Aurora Backtrack and Daily Lambda Snapshot Copy: Activate the Aurora Backtrack feature and use an AWS Lambda function to copy daily snapshots of the database to a backup Region.

C. Utilize AWS DMS for Continuous Replication to S3: Deploy AWS Database Migration Service (AWS DMS) with a change data capture (CDC) task to continuously replicate changes from the Aurora database to an Amazon S3 bucket in another Region.

D. Configure Manual Aurora Backups with Hourly Frequency: Disable automatic Aurora backups, set up manual backups every hour, and choose a different Region as the backup destination, targeting the Aurora database.

The most suitable solution is C:

- C. Utilize AWS DMS for Continuous Replication to S3: This option effectively meets the RPO requirement by continuously replicating data changes to an S3 bucket in another Region. AWS DMS provides a reliable and low-cost method for data replication, ensuring that the database is replicated with the necessary frequency for the specified RPO.

Reasons why the other options are less appropriate:

- A. Transform to Aurora Global Database: While this option provides real-time replication, it may incur higher costs compared to AWS DMS, especially for high-write environments. It might be more than necessary for the specified RPO of 1 hour.
- B. Enable Aurora Backtrack and Daily Lambda Snapshot Copy: Backtrack is primarily for point-in-time recovery within the same database and does not facilitate cross-Region replication. Daily Lambda snapshot copies do not meet the 1-hour RPO requirement.
- D. Configure Manual Aurora Backups with Hourly Frequency: Turning off automated backups and manually configuring hourly backups could be labor-intensive and riskier in maintaining the required RPO. It also doesn't provide the continuous replication needed for an RPO of 1 hour.

Question 98

A company is reassessing an AWS workload that includes a stateless application tier running on an outdated, large EC2 instance, and a MySQL database on another EC2 instance. The application server often maxes out its CPU usage, leading to unresponsiveness, and manual patching of instances has caused downtime. The company is looking for a way to enhance the application's availability with minimal developmental changes.

The potential solutions are:

A. Shift to AWS Lambda with Application Load Balancer (ALB): Transition the application tier to AWS Lambda functions within the existing VPC, using an ALB for traffic distribution. Implement Amazon GuardDuty for Lambda function scanning, and migrate the database to Amazon DocumentDB (MongoDB compatible).

B. Opt for a Smaller Graviton EC2 Instance and DynamoDB: Swap the current EC2 instance for a smaller Graviton-powered type, using a launch template for Auto Scaling and an ALB for traffic distribution. Configure scaling based on CPU usage and migrate the database to Amazon DynamoDB.

C. Utilize Docker Containers on Amazon ECS: Containerize the application using Docker and deploy on Amazon ECS with EC2 instances. Employ an ALB for traffic distribution and configure ECS scaling based on CPU usage. Transition the database to Amazon Neptune.

D. Use a New AMI with AWS SSM Agent and Aurora MySQL: Create a new AMI equipped with the AWS Systems Manager Agent. Use this AMI for an Auto Scaling group with smaller instances, an ALB for traffic distribution, and scale based on CPU utilization. Migrate the database to Amazon Aurora MySQL.

The most appropriate solution is D:

- D. Use a New AMI with AWS SSM Agent and Aurora MySQL: This approach ensures high availability with the least developmental effort. It involves updating the AMI and utilizing smaller instances in an Auto Scaling group, providing scalability and efficiency. The ALB ensures effective traffic distribution, and the migration to Amazon Aurora MySQL enhances database reliability and scalability. The inclusion of the AWS Systems Manager Agent facilitates patch management and reduces downtime.

Reasons why the other options are less appropriate:

- A. Shift to AWS Lambda with ALB: While Lambda provides scalability, it necessitates a significant alteration of the application to fit the serverless model. Migrating to DocumentDB also requires significant changes to the application's database interactions.
- B. Opt for a Smaller Graviton EC2 Instance and DynamoDB: Changing the EC2 instance type and migrating to DynamoDB involves significant development changes, particularly in adapting the application for DynamoDB's NoSQL model.
- C. Utilize Docker Containers on Amazon ECS: Containerizing the application and migrating the database to Amazon Neptune requires considerable development effort, especially in adapting the application for Neptune's graph database model.

Question 99

A company is planning to migrate various applications to AWS but lacks comprehensive knowledge about its application portfolio, which includes both physical machines and virtual machines. One of the applications to be migrated has several latency-sensitive dependencies, communicating via a custom IP-based protocol on port 1000. The company aims to identify and migrate these dependencies alongside the application to AWS to

maintain low-latency communication. The company has been gathering data using the AWS Application Discovery Agent for several months.

The potential solutions for identifying the dependencies are:

A. Leverage AWS Migration Hub and Amazon Athena for Analysis: Utilize AWS Migration Hub to visualize the network connections of the application servers. Activate data exploration in Amazon Athena to analyze data transferred between servers and identify those communicating on port 1000. Based on Athena's findings, create a move group in Migration Hub.

B. Use AWS Application Migration Service for Testing and Grouping: Select the application servers in AWS Application Migration Service and visualize network interactions. Set up test instances for servers that interact with the application, conducting acceptance tests. Form a move group based on the servers that pass these tests.

C. Integrate Network Access Analyzer with Migration Hub: In AWS Migration Hub, choose the application servers and turn on data exploration with Network Access Analyzer. Utilize Network Access Analyzer to pinpoint servers communicating on port 1000 and then create a move group in Migration Hub based on these findings.

D. Deploy CloudWatch Agent and Analyze with Athena: In AWS Migration Hub, select the application servers and deploy Amazon CloudWatch agents using AWS Application Discovery Agent. Export CloudWatch logs to Amazon S3 and query them using Amazon Athena to find servers communicating on port 1000. Create a move group in Migration Hub based on Athena's findings.

The most appropriate solution is A:

- A. Leverage AWS Migration Hub and Amazon Athena for Analysis: This method effectively utilizes the data collected by the AWS Application Discovery Agent. AWS Migration Hub provides a visual network graph to identify servers interacting with the application. Amazon Athena then allows for detailed querying of data transfers to pinpoint servers using port 1000. This approach is comprehensive and ensures that all relevant dependencies are included in the move group for migration.

Reasons why the other options are less appropriate:

- B. Use AWS Application Migration Service for Testing and Grouping: While this option includes testing, it doesn't specifically target the analysis of communication on port 1000, which is crucial for identifying latency-sensitive dependencies.
- C. Integrate Network Access Analyzer with Migration Hub: Network Access Analyzer is not specifically designed for the detailed analysis of application dependencies, especially

for custom protocols on specific ports like port 1000.

- D. Deploy CloudWatch Agent and Analyze with Athena: This approach is more complex and may not provide the necessary level of detail to identify all servers communicating on port 1000, as it relies on CloudWatch logs which might not capture all relevant network traffic data.

Question 100

A company is developing an application that will be accessed by numerous customers and hosted on an AWS Lambda function. To manage customer usage, the company seeks to implement a system where each customer has a specified number of allowable requests within a certain timeframe. Additionally, the company requires the flexibility to offer different quotas to different customers, tailored to their specific usage patterns, with some customers needing a higher quota over a shorter duration.

The options for implementing these requirements are:

A. Use Amazon API Gateway REST API with Proxy Integration and Usage Plans: Implement a REST API in Amazon API Gateway with a proxy setup to trigger the Lambda function. For each customer, establish a distinct usage plan in API Gateway, assigning a specific request quota. Generate an API key for each user within a customer's account based on their respective usage plan.

B. Deploy Amazon API Gateway HTTP API with Proxy Integration and Throttling: Set up an HTTP API in Amazon API Gateway with proxy integration to the Lambda function. For each customer, create a usage plan with a designated request quota and apply route-level throttling to each plan. Provide each user an API key from their customer's usage plan.

C. Create Lambda Function Aliases with Concurrency Limits: Generate a unique Lambda function alias for every customer, including a concurrency limit that acts as a request quota. Create a Lambda function URL for each alias and distribute the specific URL to each corresponding customer.

D. Configure Application Load Balancer (ALB) with AWS WAF Web ACL: Establish an ALB to direct traffic to the Lambda function. Set up an AWS WAF web ACL on the ALB and for each customer, configure a rule-based request quota within the web ACL.

The most suitable solution is A:

- A. Use Amazon API Gateway REST API with Proxy Integration and Usage Plans: This solution effectively manages individual customer quotas through API Gateway's usage plans. It allows for precise control over the number of requests each customer can make,

aligning with their specific usage patterns. API keys ensure secure and distinct access for each customer or user.

Reasons why the other options are less suitable:

- **B. Deploy Amazon API Gateway HTTP API with Proxy Integration and Throttling**: While this option also utilizes API Gateway, it focuses on route-level throttling rather than setting distinct quotas for each customer. It's less flexible in terms of customizing quotas based on individual customer needs.
- **C. Create Lambda Function Aliases with Concurrency Limits**: This approach does not directly address the requirement for specific request quotas over a time period. Concurrency limits control simultaneous execution rather than total requests over time.
- **D. Configure Application Load Balancer (ALB) with AWS WAF Web ACL**: Using an ALB with AWS WAF for quota management is more complex and less direct compared to using API Gateway's built-in quota management features.