Amazon

(AWS Certified DevOps Engineer - Professional DOP-C02)

AWS Certified DevOps Engineer - Professional DOP-C02

Total: **355 Questions**

Link:

Question: 1

A company has a mobile application that makes HTTP API calls to an Application Load Balancer (ALB). The ALB routes requests to an AWS Lambda function. Many different versions of the application are in use at any given time, including versions that are in testing by a subset of users. The version of the application is defined in the user-agent header that is sent with all requests to the API.
After a series of recent changes to the API, the company has observed issues with the application. The company needs to gather a metric for each API operation by response code for each version of the application that is in use. A DevOps engineer has modified the Lambda function to extract the API operation name, version information from the user-agent header and response code.
Which additional set of actions should the DevOps engineer take to gather the required metrics?

A.Modify the Lambda function to write the API operation name, response code, and version number as a log line to an Amazon CloudWatch Logs log group. Configure a CloudWatch Logs metric filter that increments a metric for each API operation name. Specify response code and application version as dimensions for the metric.

B.Modify the Lambda function to write the API operation name, response code, and version number as a log line to an Amazon CloudWatch Logs log group. Configure a CloudWatch Logs Insights query to populate CloudWatch metrics from the log lines. Specify response code and application version as dimensions for the metric.

C.Configure the ALB access logs to write to an Amazon CloudWatch Logs log group. Modify the Lambda function to respond to the ALB with the API operation name, response code, and version number as response metadata. Configure a CloudWatch Logs metric filter that increments a metric for each API operation name. Specify response code and application version as dimensions for the metric.

D.Configure AWS X-Ray integration on the Lambda function. Modify the Lambda function to create an X-Ray subsegment with the API operation name, response code, and version number. Configure X-Ray insights to extract an aggregated metric for each API operation name and to publish the metric to Amazon CloudWatch. Specify response code and application version as dimensions for the metric.

Answer: A

Explanation:

The correct answer is A. Here's why:

Requirements: The company needs to gather metrics for each API operation by response code and application version.

Why Option A is Correct:

Logging: Writing the API operation name, response code, and version number to CloudWatch Logs captures all necessary data points. This is a common and efficient practice for detailed application monitoring.
CloudWatch Logs Metric Filters: Metric filters allow you to define patterns to search for within your logs and increment CloudWatch metrics based on those patterns. Critically, they also allow specifying dimensions which enable breaking down the metric by different categories. In this case, the dimensions are response code and application version, precisely what the requirements ask for. This aggregates the log data into meaningful metrics.
Scalability: CloudWatch Logs is designed to handle large volumes of log data.

Why Option B is Incorrect:

CloudWatch Logs Insights: While powerful for ad-hoc log analysis, CloudWatch Logs Insights isn't designed for continuous metric population. It's more suited for investigative analysis rather than regularly generating metrics. Additionally, it cannot create dimensions in the way that metric filters can.

Why Option C is Incorrect:

ALB Access Logs: ALB access logs contain information about requests to the ALB, but they might not directly contain the internal API operation name as defined within the Lambda function. The question specifies that the Lambda function extracts the API operation name.

Response Metadata: Lambda cannot directly control response metadata in a manner that is directly usable for metric filters with dimensions in CloudWatch Logs.

Why Option D is Incorrect:

AWS X-Ray: X-Ray is primarily used for tracing requests through a distributed system. While X-Ray can provide insights, it's not the most direct way to gather aggregated metrics broken down by dimensions. **Complexity:** Configuring X-Ray and its insights to extract and publish metrics with dimensions to CloudWatch would be more complex than the straightforward approach offered by option A.

In essence, option A provides the most direct, scalable, and cost-effective solution using the available information, focusing on leveraging CloudWatch Logs metric filters with dimensions to meet the stated requirements.

Supporting Documentation:

CloudWatch Logs Metric Filters:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/MonitoringPolicyExamples.html
CloudWatch Metrics:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/working_with_metrics.html
CloudWatch Logs Insights:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AnalyzingLogData.html
AWS X-Ray:https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html

Question: 2

A company provides an application to customers. The application has an Amazon API Gateway REST API that invokes an AWS Lambda function. On initialization, the Lambda function loads a large amount of data from an Amazon DynamoDB table. The data load process results in long cold-start times of 8-10 seconds. The DynamoDB table has DynamoDB Accelerator (DAX) configured.
Customers report that the application intermittently takes a long time to respond to requests. The application receives thousands of requests throughout the day. In the middle of the day, the application experiences 10 times more requests than at any other time of the day. Near the end of the day, the application's request volume decreases to 10% of its normal total.
A DevOps engineer needs to reduce the latency of the Lambda function at all times of the day.
Which solution will meet these requirements?

A.Configure provisioned concurrency on the Lambda function with a concurrency value of 1. Delete the DAX cluster for the DynamoDB table.

B.Configure reserved concurrency on the Lambda function with a concurrency value of 0.

C.Configure provisioned concurrency on the Lambda function. Configure AWS Application Auto Scaling on the Lambda function with provisioned concurrency values set to a minimum of 1 and a maximum of 100.

D.Configure reserved concurrency on the Lambda function. Configure AWS Application Auto Scaling on the API Gateway API with a reserved concurrency maximum value of 100.

Answer: C

Explanation:

The correct answer is **C. Configure provisioned concurrency on the Lambda function. Configure AWS Application Auto Scaling on the Lambda function with provisioned concurrency values set to a minimum of 1 and a maximum of 100.**

Here's why:

The problem describes a Lambda function with long cold start times due to loading data from DynamoDB. Cold starts occur when a Lambda function is invoked for the first time or after a period of inactivity, requiring the function's code to be loaded and initialized. This initialization includes fetching the large dataset from DynamoDB.

Provisioned concurrency addresses this issue directly. It pre-initializes a specified number of Lambda function instances and keeps them warm, ready to respond to requests. This eliminates the cold start latency for requests routed to these pre-initialized instances.

Option A is incorrect because deleting DAX would likely increase DynamoDB read latency, negatively impacting application performance. DAX is designed to accelerate DynamoDB reads, especially for frequently accessed data, which helps mitigate the cold start problem to some extent before introducing provisioned concurrency. Also, setting provisioned concurrency to 1 is ineffective to handle fluctuating workloads.

Option B is incorrect because Reserved Concurrency is used to limit the maximum number of concurrent executions for a function. Setting it to 0 would effectively prevent the function from running at all. Reserved concurrency doesn't pre-initialize instances like provisioned concurrency does.

Option D is incorrect because reserved concurrency limits overall concurrency, not solving cold starts. Auto Scaling on the API Gateway does not address Lambda function cold starts. The problem resides in Lambda itself, not the number of API requests being served. Auto Scaling API Gateway would only distribute requests effectively, but the latency issue from Lambda would still exist on initial Lambda invocations.

Using Application Auto Scaling to manage provisioned concurrency is crucial because the application experiences significant fluctuations in traffic throughout the day. Scaling the provisioned concurrency between 1 and 100 allows the application to handle the increased traffic during peak hours without experiencing cold starts, while also reducing costs during periods of low traffic. A minimum of 1 ensures that at least one instance is always warm, preventing cold starts during off-peak times. The maximum of 100 handles the sudden tenfold increase in requests during peak times. This dynamic scaling optimizes resource utilization and minimizes latency at all times of the day.

Relevant Documentation:

AWS Lambda Provisioned Concurrency:https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html
AWS Application Auto Scaling:https://docs.aws.amazon.com/autoscaling/application/userguide/what-is-application-auto-scaling.html
Amazon DynamoDB Accelerator (DAX):
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DAX.html

Question: 3

A company is adopting AWS CodeDeploy to automate its application deployments for a Java-Apache Tomcat application with an Apache Webserver. The development team started with a proof of concept, created a deployment group for a developer environment, and performed functional tests within the application. After completion, the team will create additional deployment groups for staging and production.
The current log level is configured within the Apache settings, but the team wants to change this configuration dynamically when the deployment occurs, so that they can set different log level configurations depending on the deployment group without having a different application revision for each group.
How can these requirements be met with the LEAST management overhead and without requiring different script versions for each deployment group?

A.Tag the Amazon EC2 instances depending on the deployment group. Then place a script into the application revision that calls the metadata service and the EC2 API to identify which deployment group the instance is part of. Use this information to configure the log level settings. Reference the script as part of the AfterInstall lifecycle hook in the appspec.yml file.

B.Create a script that uses the CodeDeploy environment variable DEPLOYMENT_GROUP_ NAME to identify which deployment group the instance is part of. Use this information to configure the log level settings. Reference this script as part of the BeforeInstall lifecycle hook in the appspec.yml file.

C.Create a CodeDeploy custom environment variable for each environment. Then place a script into the application revision that checks this environment variable to identify which deployment group the instance is part of. Use this information to configure the log level settings. Reference this script as part of the ValidateService lifecycle hook in the appspec.yml file.

D.Create a script that uses the CodeDeploy environment variable DEPLOYMENT_GROUP_ID to identify which deployment group the instance is part of to configure the log level settings. Reference this script as part of the Install lifecycle hook in the appspec.yml file.

Answer: B

Explanation:

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

The requirement is to dynamically adjust the Apache log level based on the CodeDeploy deployment group without creating different application revisions or script versions for each group. This necessitates a mechanism to identify the deployment group during the deployment process.

Option B leverages the DEPLOYMENT_GROUP_NAME environment variable, which is automatically provided by CodeDeploy during deployments. This variable holds the name of the deployment group to which the instance is being deployed. A script can readily access this variable and use it to determine the appropriate log level configuration. Placing this script in the BeforeInstall lifecycle hook allows it to run before the application is actually installed, ensuring that the log level is configured before the application starts using it.
This approach avoids the overhead of managing EC2 tags or custom CodeDeploy environment variables.

Option A is less efficient. While EC2 tags can identify deployment groups, querying the metadata service and the EC2 API introduces complexity and overhead. Also, the AfterInstall lifecycle hook is too late for configuring the log level effectively, as the application might already be running.

Option C, creating custom CodeDeploy environment variables for each environment, adds unnecessary management overhead. Each environment would need to be configured separately, which defeats the purpose of having a single deployment process. Also, the ValidateService lifecycle hook is designed for validating the service's integrity, not for configuration.

Option D is close but the DEPLOYMENT_GROUP_ID is less human readable and is harder to configure and read.

In summary, Option B provides the least management overhead by leveraging the built-in CodeDeploy environment variable and executing the configuration script during the BeforeInstall lifecycle hook, fulfilling the requirements efficiently and effectively.

Relevant links for further research:

AWS CodeDeploy Lifecycle Events: https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html
AWS CodeDeploy Environment Variables:
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-environment-variables.html

Question: 4

A company requires its developers to tag all Amazon Elastic Block Store (Amazon EBS) volumes in an account to indicate a desired backup frequency. This requirement Includes EBS volumes that do not require backups. The company uses custom tags named Backup_Frequency that have values of none, dally, or weekly that correspond to the desired backup frequency. An audit finds that developers are occasionally not tagging the EBS volumes.
A DevOps engineer needs to ensure that all EBS volumes always have the Backup_Frequency tag so that the company can perform backups at least weekly unless a different value is specified.
Which solution will meet these requirements?

A.Set up AWS Config in the account. Create a custom rule that returns a compliance failure for all Amazon EC2 resources that do not have a Backup Frequency tag applied. Configure a remediation action that uses a custom AWS Systems Manager Automation runbook to apply the Backup_Frequency tag with a value of weekly.

B.Set up AWS Config in the account. Use a managed rule that returns a compliance failure for EC2::Volume resources that do not have a Backup Frequency tag applied. Configure a remediation action that uses a custom AWS Systems Manager Automation runbook to apply the Backup_Frequency tag with a value of weekly.

C.Turn on AWS CloudTrail in the account. Create an Amazon EventBridge rule that reacts to EBS CreateVolume events. Configure a custom AWS Systems Manager Automation runbook to apply the Backup_Frequency tag with a value of weekly. Specify the runbook as the target of the rule.

D.Turn on AWS CloudTrail in the account. Create an Amazon EventBridge rule that reacts to EBS CreateVolume events or EBS ModifyVolume events. Configure a custom AWS Systems Manager Automation runbook to apply the Backup_Frequency tag with a value of weekly. Specify the runbook as the target of the rule.

Answer: B

Explanation:

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

AWS Config continuously monitors and assesses the configuration of your AWS resources. It's ideal for enforcing compliance rules, as required by the scenario. Option B leverages AWS Config with a managed rule, specifically targeting EC2::Volume resources. This ensures that all EBS volumes are checked for the presence of the Backup_Frequency tag. Managed rules are pre-built and readily available within AWS Config,
simplifying the implementation. If a volume lacks the required tag, the rule marks it as non-compliant.

Crucially, Option B also includes a remediation action using a Systems Manager Automation runbook. This runbook automatically applies the missing Backup_Frequency tag with a default value of weekly. This automated remediation addresses the problem of developers occasionally forgetting to tag volumes, ensuring that all volumes are tagged and receive at least weekly backups unless explicitly configured otherwise. Systems Manager Automation provides a secure and auditable way to apply the tag.

Option A is similar but suggests using a custom Config rule, which is unnecessary since a managed rule already exists for checking tags on EBS volumes (EC2::Volume). Managed rules are generally preferred for simplicity and reduced overhead.

Options C and D rely on AWS CloudTrail and EventBridge, triggered by CreateVolume (or ModifyVolume) events. While this approach can work, it has drawbacks. It reacts after the volume has been created, potentially allowing untagged volumes to exist briefly. Also, reacting to ModifyVolume events is unnecessary as the requirement is to ensure tags are present from the start. AWS Config continuously monitors and remediates, offering a more proactive and consistent approach for ensuring ongoing compliance. Config provides historical configuration data and continuous compliance checks, features absent in event-driven, post-creation tagging.

In essence, AWS Config, with its managed rule and automated remediation via Systems Manager, provides a comprehensive and proactive solution for ensuring all EBS volumes are tagged with the desired Backup_Frequency, meeting the company's compliance requirements.

Here are authoritative links for further research:

AWS Config:https://aws.amazon.com/config/
AWS Config Managed Rules:https://docs.aws.amazon.com/config/latest/developerguide/managed-rules-by-aws.html
AWS Systems Manager Automation:https://aws.amazon.com/systems-manager/automation/
Tagging AWS Resources:https://docs.aws.amazon.com/general/latest/gr/aws_tagging.html

Question: 5

A company is using an Amazon Aurora cluster as the data store for its application. The Aurora cluster is configured with a single DB instance. The application performs read and write operations on the database by using the cluster's instance endpoint.
The company has scheduled an update to be applied to the cluster during an upcoming maintenance window. The cluster must remain available with the least possible interruption during the maintenance window.
What should a DevOps engineer do to meet these requirements?

A.Add a reader instance to the Aurora cluster. Update the application to use the Aurora cluster endpoint for write operations. Update the Aurora cluster's reader endpoint for reads.

B.Add a reader instance to the Aurora cluster. Create a custom ANY endpoint for the cluster. Update the application to use the Aurora cluster's custom ANY endpoint for read and write operations.

C.Turn on the Multi-AZ option on the Aurora cluster. Update the application to use the Aurora cluster endpoint for write operations. Update the Aurora cluster's reader endpoint for reads.

D.Turn on the Multi-AZ option on the Aurora cluster. Create a custom ANY endpoint for the cluster. Update the application to use the Aurora cluster's custom ANY endpoint for read and write operations

Answer: A

Explanation:

The best approach to minimize interruption during an Aurora cluster update while maintaining availability is option A. Here's why:

1. **Read Scalability and Availability:** Adding a reader instance provides a read replica. During the maintenance window, the primary instance might be unavailable temporarily. The reader instance continues to serve read requests, improving availability for read operations.

2. **Cluster Endpoint for Writes:** Using the cluster endpoint for writes ensures that write operations are automatically directed to the primary instance in the cluster, even during the maintenance process when the primary may failover. Aurora handles the redirection behind the scenes.

3. **Reader Endpoint for Reads:** The reader endpoint specifically directs read traffic to available reader instances. This is crucial because, as mentioned before, the primary instance may be unavailable briefly during the update. Directing read traffic to the reader instance prevents read operations from failing.

4. **Why other options are less ideal:**

   Option B: Creating a custom ANY endpoint is not a standard practice for Aurora. While custom endpoints can be made, the default cluster and reader endpoints are more appropriate and easily managed for this scenario. Creating a custom endpoint adds unnecessary complexity. Also, an ANY endpoint won't automatically redirect writes to a new primary if a failover happens.

   Option C: Turning on Multi-AZ is a good practice for high availability. However, by itself, it only provides failover capabilities. During the failover itself, there will be a short period of unavailability. It does not address read availability explicitly.

   Option D: Combining Multi-AZ with a custom ANY endpoint has the same drawbacks as options B and C combined. It doesn't give the advantages of the built in reader endpoint.

5. **Least Interruption:** By using the cluster endpoint for writes and the reader endpoint for reads with a reader instance, the application experiences the least interruption during the maintenance window. Aurora manages the failover to a new primary (if needed), and the reader instance continues serving reads. The application logic itself doesn't need to be aware of the failover process.

In summary: Option A leverages Aurora's built-in high availability and read scaling features in the most straightforward and effective way to minimize downtime during a maintenance window. It ensures that reads can continue to be served even if the primary instance is temporarily unavailable, and the cluster endpoint seamlessly handles write redirection.

Relevant Links:

Amazon Aurora Endpoints
High Availability for Amazon Aurora

Question: 6

A company must encrypt all AMIs that the company shares across accounts. A DevOps engineer has access to a source account where an unencrypted custom AMI has been built. The DevOps engineer also has access to a target account where an Amazon EC2 Auto Scaling group will launch EC2 instances from the AMI. The DevOps engineer must share the AMI with the target account.
The company has created an AWS Key Management Service (AWS KMS) key in the source account.
Which additional steps should the DevOps engineer perform to meet the requirements? (Choose three.)

A.In the source account, copy the unencrypted AMI to an encrypted AMI. Specify the KMS key in the copy action.

B.In the source account, copy the unencrypted AMI to an encrypted AMI. Specify the default Amazon Elastic Block Store (Amazon EBS) encryption key in the copy action.

C.In the source account, create a KMS grant that delegates permissions to the Auto Scaling group service-linked role in the target account.

D.In the source account, modify the key policy to give the target account permissions to create a grant. In the target account, create a KMS grant that delegates permissions to the Auto Scaling group service-linked role.

E.In the source account, share the unencrypted AMI with the target account.

F.In the source account, share the encrypted AMI with the target account.

Answer: ADF

Explanation:

Here's a detailed justification for the correct answer (ADF) to the AMI encryption and sharing scenario:

A. In the source account, copy the unencrypted AMI to an encrypted AMI. Specify the KMS key in the copy action. This step is fundamental. You cannot directly encrypt an existing unencrypted AMI. You must create a copy and encrypt it during the copy process. Specifying the KMS key ensures that the AMI is encrypted using the company's designated key, fulfilling the requirement of encrypting all shared AMIs.
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html

D. In the source account, modify the key policy to give the target account permissions to create a grant. In the target account, create a KMS grant that delegates permissions to the Auto Scaling group service-linked role. When sharing encrypted AMIs across accounts, the target account needs permission to use the KMS key to decrypt the AMI and associated EBS volumes. The source account's KMS key policy must grant the target account permission to perform actions like kms:CreateGrant. Then the grant allows the service linked role of autoscaling in target account to use the KMS key.
https://docs.aws.amazon.com/kms/latest/developerguide/grants.html

F. In the source account, share the encrypted AMI with the target account. Only the encrypted AMI should be shared. Sharing the unencrypted AMI (as suggested in option E) would violate the security requirement of encrypting all shared AMIs. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sharingamis-encrypt.html

Why other options are incorrect:

B: Using the default EBS encryption key would not fulfill the requirement to use the company's KMS key. The company wants control and manageability of encryption via their specific KMS key.

C: While grants are important, the target account needs permission to create the grant first. Option C skips the crucial step of modifying the KMS key policy in the source account to allow the target account to create a grant.

E: As mentioned earlier, this directly contradicts the requirement to share only encrypted AMIs.

Question: 7

A company uses AWS CodePipeline pipelines to automate releases of its application A typical pipeline consists of three stages build, test, and deployment. The company has been using a separate AWS CodeBuild project to run scripts for each stage. However, the company now wants to use AWS CodeDeploy to handle the deployment stage of the pipelines.
The company has packaged the application as an RPM package and must deploy the application to a fleet of Amazon EC2 instances. The EC2 instances are in an EC2 Auto Scaling group and are launched from a common AMI.
Which combination of steps should a DevOps engineer perform to meet these requirements? (Choose two.)

A.Create a new version of the common AMI with the CodeDeploy agent installed. Update the IAM role of the EC2 instances to allow access to CodeDeploy.

B.Create a new version of the common AMI with the CodeDeploy agent installed. Create an AppSpec file that contains application deployment scripts and grants access to CodeDeploy.

C.Create an application in CodeDeploy. Configure an in-place deployment type. Specify the Auto Scaling group as the deployment target. Add a step to the CodePipeline pipeline to use EC2 Image Builder to create a new AMI. Configure CodeDeploy to deploy the newly created AMI.

D.Create an application in CodeDeploy. Configure an in-place deployment type. Specify the Auto Scaling group as the deployment target. Update the CodePipeline pipeline to use the CodeDeploy action to deploy the application.

E.Create an application in CodeDeploy. Configure an in-place deployment type. Specify the EC2 instances that are launched from the common AMI as the deployment target. Update the CodePipeline pipeline to use the CodeDeploy action to deploy the application.

Answer: AD

Explanation:

Here's a detailed justification for why options A and D are the correct choices, and why the others are incorrect, for integrating AWS CodeDeploy into an existing CodePipeline to deploy an RPM package to an EC2 Auto Scaling group.

Justification for Option A: Create a new version of the common AMI with the CodeDeploy agent installed. Update the IAM role of the EC2 instances to allow access to CodeDeploy.

The CodeDeploy agent is a prerequisite for CodeDeploy to manage deployments on an EC2 instance. The agent must be installed and running on each instance targeted for deployment. Baking the agent into the common AMI ensures that every EC2 instance launched from that AMI has the necessary software.

CodeDeploy needs permissions to perform actions on the EC2 instances, such as stopping, starting, and copying files. The IAM role attached to the EC2 instances must grant these permissions. Without the correct IAM role, CodeDeploy will fail to deploy the application.https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-create-iam-instance-profile.html

Justification for Option D: Create an application in CodeDeploy. Configure an in-place deployment type. Specify the Auto Scaling group as the deployment target. Update the CodePipeline pipeline to use the CodeDeploy action to deploy the application.

Before using CodeDeploy, an application must be created. This application is a container for deployments. For deploying to a fleet of EC2 instances managed by an Auto Scaling group, the "in-place" deployment type is appropriate. This deployment type updates the application directly on the existing instances.

Specifying the Auto Scaling group as the deployment target ensures that CodeDeploy deploys to all instances within the group. This is crucial for maintaining consistency across the fleet.

To integrate CodeDeploy into the existing CodePipeline, the CodePipeline must be updated to use the CodeDeploy action in the deployment stage. This step will initiate the deployment process through CodeDeploy.https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-create-pipeline.html

Why other options are incorrect:

Option B: While creating a new AMI with the CodeDeploy agent is correct, creating an AppSpec file doesn't inherently grant access to CodeDeploy. The IAM role handles access. The AppSpec file defines how the application is deployed, not who can deploy it.

Option C: While EC2 Image Builder can be used to create AMIs, it is not necessary to use it as a step in every pipeline execution. The company already uses a common AMI. Using EC2 Image Builder in this way would cause AMI proliferation.

Option E: Specifying individual EC2 instances instead of the Auto Scaling group as a deployment target defeats the purpose of using an Auto Scaling group. If new instances are launched as part of scaling, they will not be included in the deployment. Targeting the Auto Scaling group ensures that all instances within the group are deployed.

In summary, the combination of creating an AMI with the CodeDeploy agent, updating the IAM role of EC2 instances, creating a CodeDeploy application, specifying an in-place deployment type with the Auto Scaling group as the target, and integrating CodeDeploy into the CodePipeline will meet the requirements of automating the deployment of an RPM package to an EC2 Auto Scaling group.

Question: 8

A company's security team requires that all external Application Load Balancers (ALBs) and Amazon API Gateway APIs are associated with AWS WAF web ACLs. The company has hundreds of AWS accounts, all of which are included in a single organization in AWS Organizations. The company has configured AWS Config for the organization. During an audit, the company finds some externally facing ALBs that are not associated with AWS WAF web ACLs.
Which combination of steps should a DevOps engineer take to prevent future violations? (Choose two.)

A.Delegate AWS Firewall Manager to a security account.

B.Delegate Amazon GuardDuty to a security account.

C.Create an AWS Firewall Manager policy to attach AWS WAF web ACLs to any newly created ALBs and API Gateway APIs.

D.Create an Amazon GuardDuty policy to attach AWS WAF web ACLs to any newly created ALBs and API Gateway APIs.

E.Configure an AWS Config managed rule to attach AWS WAF web ACLs to any newly created ALBs and API Gateway APIs.

Answer: AC

Explanation:

The correct answer is AC. Here's why:

A. Delegate AWS Firewall Manager to a security account: AWS Firewall Manager is designed to centrally manage and enforce security rules across multiple AWS accounts within an organization. Delegating Firewall Manager to a dedicated security account centralizes the configuration and management of WAF rules, making it easier to enforce the company's security policy consistently. This delegation is a prerequisite for creating organization-wide firewall policies.

C. Create an AWS Firewall Manager policy to attach AWS WAF web ACLs to any newly created ALBs and

API Gateway APIs: This is the core step for preventing future violations. Firewall Manager allows you to create policies that automatically associate WAF web ACLs with newly created resources (ALBs and API Gateway APIs, in this case). This policy enforcement ensures that all externally facing ALBs and API Gateways are protected by WAF, meeting the company's security requirements. Firewall Manager policies can be configured to apply across the entire organization or specific organizational units (OUs).

Why the other options are incorrect:

B. Delegate Amazon GuardDuty to a security account: GuardDuty is a threat detection service, it does not enforce WAF policies. It detects malicious activity but doesn't prevent the creation of unprotected resources. **D. Create an Amazon GuardDuty policy to attach AWS WAF web ACLs to any newly created ALBs and API Gateway APIs:** GuardDuty doesn't have the capability to automatically attach WAF web ACLs. It only reports security threats.

E. Configure an AWS Config managed rule to attach AWS WAF web ACLs to any newly created ALBs and API Gateway APIs: AWS Config's primary function is to assess and audit the configuration of your AWS resources. While Config can detect resources that are non-compliant (i.e., not associated with WAF), it cannot automatically attach WAF web ACLs to newly created resources in the same way as Firewall Manager. Config can trigger remediation actions via Systems Manager Automation documents, Lambda functions, or CloudWatch Events/EventBridge, it would require additional scripting/automation for the WAF attachment, which Firewall Manager simplifies.

In summary: Firewall Manager provides the centralized policy enforcement capabilities required to automatically associate WAF web ACLs with new ALBs and API Gateway APIs across an organization, which addresses the company's need to prevent future violations. Delegating Firewall Manager to a security account provides a central place for management.

Supporting Documentation:

AWS Firewall Manager:https://aws.amazon.com/firewall-manager/
AWS WAF:https://aws.amazon.com/waf/
AWS Organizations:https://aws.amazon.com/organizations/
AWS Config:https://aws.amazon.com/config/
Amazon GuardDuty:https://aws.amazon.com/guardduty/

Question: 9

A company uses AWS Key Management Service (AWS KMS) keys and manual key rotation to meet regulatory compliance requirements. The security team wants to be notified when any keys have not been rotated after 90 days.
Which solution will accomplish this?

A.Configure AWS KMS to publish to an Amazon Simple Notification Service (Amazon SNS) topic when keys are more than 90 days old.

B.Configure an Amazon EventBridge event to launch an AWS Lambda function to call the AWS Trusted Advisor API and publish to an Amazon Simple Notification Service (Amazon SNS) topic.

C.Develop an AWS Config custom rule that publishes to an Amazon Simple Notification Service (Amazon SNS) topic when keys are more than 90 days old.

D.Configure AWS Security Hub to publish to an Amazon Simple Notification Service (Amazon SNS) topic when keys are more than 90 days old.

Answer: C

Explanation:

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

Why Option C is Correct: Develop an AWS Config custom rule that publishes to an Amazon Simple Notification Service (Amazon SNS) topic when keys are more than 90 days old.

AWS Config is the ideal service for tracking configuration changes and compliance status across your AWS resources. It allows you to define custom rules to evaluate whether your resources comply with your internal policies. Specifically for this scenario:

1. **Continuous Monitoring:** AWS Config continuously monitors the configuration of your AWS resources, including KMS keys.
2. **Custom Rule Creation:** You can create a custom AWS Config rule using Lambda to evaluate the rotation status of KMS keys. The Lambda function can check the KeyRotationStatus and CreationDate attributes of the KMS key.
3. **Age Evaluation:** The custom rule can be configured to check if the key's age (calculated from the creation date or last rotation date) exceeds 90 days.
4. **Non-Compliance Reporting:** If a KMS key is found to be older than 90 days, the Config rule will mark it as non-compliant.
5. **SNS Notification:** AWS Config can be configured to publish an Amazon SNS notification when a resource's compliance status changes. In this case, when a KMS key becomes non-compliant due to being older than 90 days, a notification will be sent to the specified SNS topic.
6. **Automation and Auditability:** This approach automates the compliance checking process and provides an auditable record of compliance status over time.

Why Other Options are Incorrect:

Option A: Configure AWS KMS to publish to an Amazon Simple Notification Service (Amazon SNS) topic when keys are more than 90 days old. AWS KMS does not natively offer a mechanism to publish to SNS based on key age. KMS publishes events for key usage and management operations (like key rotation actions, not lack thereof), but not based on a period since the last rotation.

Option B: Configure an Amazon EventBridge event to launch an AWS Lambda function to call the AWS Trusted Advisor API and publish to an Amazon Simple Notification Service (Amazon SNS) topic. While EventBridge and Lambda can be used, using Trusted Advisor for this specific KMS key rotation check isn't ideal. Trusted Advisor provides high-level best practices, cost optimization, security improvements, performance enhancements, and service limits. Tracking key rotation through AWS Config provides a more focused and auditable solution. The Trusted Advisor API doesn't specifically target KMS key rotation in a way that easily facilitates the 90-day check.

Option D: Configure AWS Security Hub to publish to an Amazon Simple Notification Service (Amazon SNS) topic when keys are more than 90 days old. AWS Security Hub aggregates findings from various AWS security services. While Security Hub might surface findings related to KMS keys through integrations with other services, it doesn't directly monitor key age and trigger SNS notifications based on a 90-day threshold for rotation. It relies on other services or integrated third-party tools to provide such detailed configuration analysis. Security Hub findings are often more general security alerts, rather than specific configuration violations like the age of a KMS key.

Supporting Documentation:

AWS Config:https://aws.amazon.com/config/
AWS Config Custom Rules:https://docs.aws.amazon.com/config/latest/developerguide/evaluate-config.html **AWS KMS Key Rotation:**https://docs.aws.amazon.com/kms/latest/developerguide/rotate-keys.html

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

Question: 10

A security review has identified that an AWS CodeBuild project is downloading a database population script from an Amazon S3 bucket using an unauthenticated request. The security team does not allow unauthenticated requests to S3 buckets for this project.
How can this issue be corrected in the MOST secure manner?

A.Add the bucket name to the AllowedBuckets section of the CodeBuild project settings. Update the build spec to use the AWS CLI to download the database population script.

B.Modify the S3 bucket settings to enable HTTPS basic authentication and specify a token. Update the build spec to use cURL to pass the token and download the database population script.

C.Remove unauthenticated access from the S3 bucket with a bucket policy. Modify the service role for the CodeBuild project to include Amazon S3 access. Use the AWS CLI to download the database population script.

D.Remove unauthenticated access from the S3 bucket with a bucket policy. Use the AWS CLI to download the database population script using an IAM access key and a secret access key.

Answer: C

Explanation:

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

Why Option C is the MOST Secure:

Option C enforces the principle of least privilege and uses AWS best practices for secure access management.

1. **Restricting Unauthenticated Access:** Removing unauthenticated access from the S3 bucket is a crucial first step. This prevents anyone without proper credentials from accessing the database population script. A bucket policy is the standard method for controlling access to an S3 bucket.

2. **Leveraging IAM Roles:** IAM roles are the preferred method for granting permissions to AWS services. CodeBuild, like other AWS services, can assume an IAM role that defines what resources it can access. Instead of using potentially exposed access keys, a service role automatically provides temporary credentials for access.

3. **Granting S3 Access via Service Role:** Modifying the CodeBuild's service role to include Amazon S3 access means CodeBuild will be able to access S3 buckets securely, eliminating the need for unauthenticated requests or storing sensitive keys within the CodeBuild project.

4. **Using AWS CLI for Authentication:** The AWS CLI automatically uses the credentials provided by the service role to authenticate and authorize requests. Thus, the `aws s3 cp` or `aws s3 sync` commands will download the database population script using the role's permissions.

Why Other Options are Less Secure or Incorrect:

Option A: Adding the bucket to the AllowedBuckets section is a deprecated feature and doesn't inherently provide authentication or authorization. It just tells CodeBuild what buckets it can attempt to access, not how to access them securely. It doesn't address the core issue of unauthenticated access.

Option B: Enabling HTTP basic authentication on an S3 bucket is highly discouraged. It's less secure than using IAM roles because it involves managing and storing secrets (tokens) which are susceptible to leakage. Furthermore, HTTP basic authentication is generally used with HTTPS.

Option D: Storing IAM access keys and secret access keys directly in the CodeBuild environment (or worse, in the build script) is a major security risk. If these keys are compromised, an attacker can use them to access any AWS resources that the keys have permissions to access. This defeats the purpose of using roles.

Supporting Cloud Computing Concepts:

Principle of Least Privilege: Only grant the minimum necessary permissions to a user or service. Option C adheres to this by granting CodeBuild only the S3 access required to download the script.

IAM Roles: IAM roles are a secure way to grant permissions to AWS services without needing to manage long-term credentials.

Authentication vs. Authorization: Authentication verifies the identity of the user or service. Authorization determines what resources the authenticated identity is allowed to access.

Authoritative Links:

IAM Roles:https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
S3 Bucket Policies:https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucket-policies.html **CodeBuild IAM Service Role:**https://docs.aws.amazon.com/codebuild/latest/userguide/security-iam.html

Question: 11

An ecommerce company has chosen AWS to host its new platform. The company's DevOps team has started building an AWS Control Tower landing zone. The DevOps team has set the identity store within AWS IAM Identity Center (AWS Single Sign-On) to external identity provider (IdP) and has configured SAML 2.0.
The DevOps team wants a robust permission model that applies the principle of least privilege. The model must allow the team to build and manage only the team's own resources.
Which combination of steps will meet these requirements? (Choose three.)

A.Create IAM policies that include the required permissions. Include the aws:PrincipalTag condition key.

B.Create permission sets. Attach an inline policy that includes the required permissions and uses the aws:PrincipalTag condition key to scope the permissions.

C.Create a group in the IdP. Place users in the group. Assign the group to accounts and the permission sets in IAM Identity Center.

D.Create a group in the IdP. Place users in the group. Assign the group to OUs and IAM policies.

E.Enable attributes for access control in IAM Identity Center. Apply tags to users. Map the tags as key-value pairs.

F.Enable attributes for access control in IAM Identity Center. Map attributes from the IdP as key-value pairs.

Answer: BCF

Explanation:

The correct answer is BCF. Here's why:

B: Create permission sets. Attach an inline policy that includes the required permissions and uses the aws:PrincipalTag condition key to scope the permissions. Permission sets in IAM Identity Center (successor to AWS Single Sign-On) define the permissions granted to users when they access AWS accounts. By attaching an inline policy with the aws:PrincipalTag condition key, the permissions are scoped based on user attributes, enforcing least privilege. The aws:PrincipalTag condition key allows you to control access to AWS resources based on tags associated with the IAM principal (user or role). This aligns with the requirement to allow teams to manage only their own resources, ensuring resource isolation.https://docs.aws.amazon.com/singlesignon/latest/userguide/permissionsetsconcept.htmlhttps://docs.aws keys.html#condition-keys-principaltag

C: Create a group in the IdP. Place users in the group. Assign the group to accounts and the permission sets in IAM Identity Center. Group management in the external IdP simplifies user assignment to AWS accounts and permission sets. By placing users in groups, you can centrally manage their access to various AWS accounts and the permissions they have within those accounts. IAM Identity Center uses these group assignments to grant access. This central management improves scalability and reduces administrative overhead.https://docs.aws.amazon.com/singlesignon/latest/userguide/manage-group-access.html

F: Enable attributes for access control in IAM Identity Center. Map attributes from the IdP as key-value pairs. Enabling attributes for access control is essential for using the aws:PrincipalTag condition key effectively. By mapping attributes from the external IdP to key-value pairs in IAM Identity Center, these attributes can be used as tags on the IAM principal (user). These tags are then used in the inline policies within permission sets to scope permissions based on user attributes, enabling fine-grained access control based on user attributes defined in the IdP. This integration between the IdP and AWS is crucial for dynamic, attribute-based access control (ABAC).https://docs.aws.amazon.com/singlesignon/latest/userguide/attributemappings.htmlhttps://aws.amazon.co to-centrally-manage-aws-account-access-using-aws-sso-and-attribute-based-access-control/

Why other options are incorrect:

A: Create IAM policies that include the required permissions. Include the aws:PrincipalTag condition key. While you can create IAM policies with the aws:PrincipalTag condition key, this approach is not the ideal solution for AWS Control Tower environments with IAM Identity Center (SSO). IAM policies are generally used for roles assumed by applications or services, whereas permission sets in IAM Identity Center are designed for users accessing AWS accounts through SSO. The question focuses on managing user access, making permission sets more appropriate.

D: Create a group in the IdP. Place users in the group. Assign the group to OUs and IAM policies. Assigning IAM policies directly to OUs (Organizational Units) is typically used for service control policies (SCPs), which establish guardrails and restrict what users can do within the OU. This is too coarse-grained for this scenario. We need a finer level of permission control based on the user.

E: Enable attributes for access control in IAM Identity Center. Apply tags to users. Map the tags as key-value pairs. While enabling attributes is correct (as in option F), applying tags to users directly in IAM Identity Center (SSO) isn't the usual flow when integrating with an external IdP. The intent is to leverage the attributes already existing in the IdP, thus option F mapping these existing attributes is the better approach.

Question: 12

An ecommerce company is receiving reports that its order history page is experiencing delays in reflecting the processing status of orders. The order processing system consists of an AWS Lambda function that uses reserved concurrency. The Lambda function processes order messages from an Amazon Simple Queue Service (Amazon SQS) queue and inserts processed orders into an Amazon DynamoDB table. The DynamoDB table has auto scaling enabled for read and write capacity.
Which actions should a DevOps engineer take to resolve this delay? (Choose two.)

A.Check the ApproximateAgeOfOldestMessage metric for the SQS queue. Increase the Lambda function concurrency limit.

B.Check the ApproximateAgeOfOldestMessage metnc for the SQS queue Configure a redrive policy on the SQS queue.

C.Check the NumberOfMessagesSent metric for the SQS queue. Increase the SQS queue visibility timeout.

D.Check the WriteThrottleEvents metric for the DynamoDB table. Increase the maximum write capacity units (WCUs) for the table's scaling policy.

E.Check the Throttles metric for the Lambda function. Increase the Lambda function timeout.

Answer: AD

Explanation:

The correct answer is AD. Here's why:

A. Check the ApproximateAgeOfOldestMessage metric for the SQS queue. Increase the Lambda function concurrency limit.

The ApproximateAgeOfOldestMessage metric indicates how long messages have been waiting in the SQS queue. If this value is high, it means messages are not being processed quickly enough. A high backlog suggests the Lambda function is not consuming messages from the queue fast enough, which directly contributes to the delay in updating the order history. Increasing the Lambda function's concurrency limit allows more instances of the function to run simultaneously, enabling faster processing of messages from the SQS queue. This helps reduce the backlog and reflects order updates more quickly. Reserved concurrency guarantees that the Lambda function has the resources it needs, but it must be adequately sized.

D. Check the WriteThrottleEvents metric for the DynamoDB table. Increase the maximum write capacity units (WCUs) for the table's scaling policy.

The WriteThrottleEvents metric for DynamoDB signifies that write requests to the table are being throttled because the table's write capacity is insufficient. The Lambda function writes processed order information to DynamoDB. If writes are being throttled, updates will be delayed. DynamoDB auto scaling should ideally adjust to workload changes, but the maximum WCUs limit within the scaling policy might be too low.
Increasing the maximum WCUs limit in the scaling policy allows DynamoDB to scale up its write capacity further, accommodating the increased load and preventing throttling, which in turn reduces the delay in order history updates.

Why other options are incorrect:

B: While a redrive policy is useful for handling failed messages, it doesn't address the root cause of the delay, which is slow processing and write throttling. It only moves problematic messages to a dead-letter queue.
C:NumberOfMessagesSent metric only indicates the number of messages being added to the queue, not the processing status. Increasing the visibility timeout will only delay the messages from being available for other consumers if the initial consumer fails.
E: The Throttles metric for Lambda indicates invocation throttles at the account level due to concurrency limits. While related to concurrency, Throttles is different from issues inside the function itself. Increasing the function timeout won't directly solve the queue backlog or DynamoDB throttling issues.

Supporting Links:

Amazon SQS Metrics: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-available-cloudwatch-metrics.html
AWS Lambda Concurrency:https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html
Amazon DynamoDB Auto Scaling: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.html
DynamoDB Metrics:https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/metrics-dimensions.html

Question: 13

A company has a single AWS account that runs hundreds of Amazon EC2 instances in a single AWS Region. New EC2 instances are launched and terminated each hour in the account. The account also includes existing EC2 instances that have been running for longer than a week.
The company's security policy requires all running EC2 instances to use an EC2 instance profile. If an EC2 instance does not have an instance profile attached, the EC2 instance must use a default instance profile that has no IAM

permissions assigned.
A DevOps engineer reviews the account and discovers EC2 instances that are running without an instance profile.
During the review, the DevOps engineer also observes that new EC2 instances are being launched without an instance profile.
Which solution will ensure that an instance profile is attached to all existing and future EC2 instances in the Region?

A.Configure an Amazon EventBridge rule that reacts to EC2 RunInstances API calls. Configure the rule to invoke an AWS Lambda function to attach the default instance profile to the EC2 instances.

B.Configure the ec2-instance-profile-attached AWS Config managed rule with a trigger type of configuration changes. Configure an automatic remediation action that invokes an AWS Systems Manager Automation runbook to attach the default instance profile to the EC2 instances.

C.Configure an Amazon EventBridge rule that reacts to EC2 StartInstances API calls. Configure the rule to invoke an AWS Systems Manager Automation runbook to attach the default instance profile to the EC2 instances

D.Configure the iam-role-managed-policy-check AWS Config managed rule with a trigger type of configuration changes. Configure an automatic remediation action that invokes an AWS Lambda function to attach the default instance profile to the EC2 instances.

Answer: B

Explanation:

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

The core problem is ensuring all EC2 instances, both existing and future, have an instance profile attached, defaulting to one with no permissions if no other profile is specified. AWS Config is designed for continuous compliance monitoring and remediation. Option B leverages this by using the ec2-instance-profile-attached managed rule. This rule specifically checks if EC2 instances have an associated instance profile.

The configuration changes trigger ensures that whenever an EC2 instance is launched or modified without an instance profile, the rule detects it. The automatic remediation action then invokes an AWS Systems Manager (SSM) Automation runbook. SSM Automation provides a safe and reliable way to automatically attach the default instance profile to the non-compliant EC2 instances. Runbooks allow for pre-defined and tested procedures for incident response and operational tasks.

Option A reacts to RunInstances API calls, which is a good approach for new instances, but it doesn't address existing instances without profiles. It also utilizes Lambda, which is generally acceptable, but SSM Automation offers stronger operational capabilities for this specific remediation task.

Option C focuses on StartInstances API calls, which would only address instances being started, not those already running without a profile or new instances being created but not immediately started.

Option D utilizes the iam-role-managed-policy-check AWS Config managed rule, which is not directly related to ensuring an instance profile is attached. This rule primarily focuses on IAM role policy content, not the existence of an instance profile on an EC2 instance.

In summary, Option B provides a comprehensive solution by leveraging AWS Config for continuous monitoring and SSM Automation for automated remediation, covering both existing and newly launched instances, making it the most suitable approach.

Further reading:

AWS Config Managed Rules: https://docs.aws.amazon.com/config/latest/developerguide/managed-rules-by-aws.html
AWS Systems Manager Automation: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-automation.html
EC2 Instance Profiles: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html

Question: 14

A DevOps engineer is building a continuous deployment pipeline for a serverless application that uses AWS Lambda functions. The company wants to reduce the customer impact of an unsuccessful deployment. The company also wants to monitor for issues.
Which deploy stage configuration will meet these requirements?

A.Use an AWS Serverless Application Model (AWS SAM) template to define the serverless application. Use AWS CodeDeploy to deploy the Lambda functions with the Canary10Percent15Minutes Deployment Preference Type. Use Amazon CloudWatch alarms to monitor the health of the functions.

B.Use AWS CloudFormation to publish a new stack update, and include Amazon CloudWatch alarms on all resources. Set up an AWS CodePipeline approval action for a developer to verify and approve the AWS CloudFormation change set.

C.Use AWS CloudFormation to publish a new version on every stack update, and include Amazon CloudWatch alarms on all resources. Use the RoutingConfig property of the AWS::Lambda::Alias resource to update the traffic routing during the stack update.

D.Use AWS CodeBuild to add sample event payloads for testing to the Lambda functions. Publish a new version of the functions, and include Amazon CloudWatch alarms. Update the production alias to point to the new version. Configure rollbacks to occur when an alarm is in the ALARM state.

Answer: A

Explanation:

The correct answer is **A** because it directly addresses the requirements of minimizing customer impact and monitoring for issues during serverless application deployments.

Here's a detailed justification:

AWS SAM Template: Using AWS SAM (Serverless Application Model) simplifies the definition and deployment of serverless applications on AWS. It is an extension of AWS CloudFormation and makes managing serverless resources easier.

https://aws.amazon.com/serverless/sam/

AWS CodeDeploy with Canary Deployment: Implementing a canary deployment strategy using AWS CodeDeploy (specifically Canary10Percent15Minutes) gradually shifts a small percentage (10%) of traffic to the new Lambda function version over a short period (15 minutes). This limits the impact of any potential issues to a small subset of users. If issues arise, the deployment can be rolled back before affecting a larger user base. Canary deployments are ideal for minimizing the blast radius of deployments.

https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-configurations.html

Amazon CloudWatch Alarms: CloudWatch alarms actively monitor the health and performance of the Lambda functions. Setting up alarms on key metrics (e.g., error rates, latency, invocation counts) enables prompt detection of anomalies or failures after the new version is deployed. When an alarm is triggered, automated actions such as rollbacks can be initiated.

https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html

Why other options are less suitable:

B: Using AWS CloudFormation and manual approval adds a human gate that might slow down the continuous deployment pipeline significantly. While CloudWatch alarms are present, the lack of an automated deployment strategy like canary deployment does not minimize the user impact in case of failure.
C: Although using the RoutingConfig property and CloudWatch alarms are helpful, this option relies heavily on CloudFormation to manage traffic shifting and does not inherently provide a managed canary deployment like CodeDeploy.

D: While adding sample event payloads for testing with CodeBuild and using CloudWatch alarms are good practices, this approach does not employ a progressive deployment strategy. Updating the production alias directly exposes all users to the new version immediately, increasing the risk of a widespread impact if issues exist. Relying solely on alarm-triggered rollbacks is reactive and not as preventative as a canary deployment.

Question: 15

To run an application, a DevOps engineer launches an Amazon EC2 instance with public IP addresses in a public subnet. A user data script obtains the application artifacts and installs them on the instances upon launch. A change to the security classification of the application now requires the instances to run with no access to the internet. While the instances launch successfully and show as healthy, the application does not seem to be installed.
Which of the following should successfully install the application while complying with the new rule?

A.Launch the instances in a public subnet with Elastic IP addresses attached. Once the application is installed and running, run a script to disassociate the Elastic IP addresses afterwards.

B.Set up a NAT gateway. Deploy the EC2 instances to a private subnet. Update the private subnet's route table to use the NAT gateway as the default route.

C.Publish the application artifacts to an Amazon S3 bucket and create a VPC endpoint for S3. Assign an IAM instance profile to the EC2 instances so they can read the application artifacts from the S3 bucket.

D.Create a security group for the application instances and allow only outbound traffic to the artifact repository. Remove the security group rule once the install is complete.

Answer: C

Explanation:

The correct answer is C, which leverages Amazon S3 and VPC endpoints for application artifact retrieval without internet access. Here's why:

Option C is the most secure and compliant solution. It suggests storing the application artifacts in an Amazon S3 bucket. By creating a VPC endpoint for S3, the EC2 instances can access the S3 bucket without traversing the internet. This satisfies the requirement of no internet access for the instances. Furthermore, assigning an IAM instance profile to the EC2 instances ensures that they have the necessary permissions to read the application artifacts from the S3 bucket. This approach aligns with best practices for security and resource management.

Option A is problematic because it temporarily allows internet access via public and Elastic IP addresses, which violates the "no internet access" policy, even if the IPs are disassociated later. This approach creates a window of vulnerability.

Option B, using a NAT Gateway, still requires internet access, albeit indirectly, for the instances in the private subnet. NAT Gateways provide internet access for instances within the private subnet, which contradicts the requirements.

Option D, modifying security group rules, is also problematic. Modifying security group rules after the fact introduces a window of vulnerability. It also requires the EC2 instances to have initial internet access to reach the artifact repository, which is not permitted. Security groups also control inbound and outbound traffic. You would still need to open traffic to the internet temporarily to allow the instances to download the artifacts.
The key with the right answer is utilizing the S3 bucket as storage within the VPC to allow artifact retrieval.

Therefore, Option C is the only option that securely and permanently prevents internet access to the instances while allowing them to retrieve and install the application artifacts.

Supporting links:

VPC Endpoints for S3:https://docs.aws.amazon.com/vpc/latest/privatelink/vpc-endpoints-s3.html **IAM Roles for EC2:**https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_ec2.html

Question: 16

A development team is using AWS CodeCommit to version control application code and AWS CodePipeline to orchestrate software deployments. The team has decided to use a remote main branch as the trigger for the pipeline to integrate code changes. A developer has pushed code changes to the CodeCommit repository, but noticed that the pipeline had no reaction, even after 10 minutes.
Which of the following actions should be taken to troubleshoot this issue?

A.Check that an Amazon EventBridge rule has been created for the main branch to trigger the pipeline.
B.Check that the CodePipeline service role has permission to access the CodeCommit repository.
C.Check that the developer's IAM role has permission to push to the CodeCommit repository.
D.Check to see if the pipeline failed to start because of CodeCommit errors in Amazon CloudWatch Logs.

Answer: A

Explanation:

The correct answer is A: Check that an Amazon EventBridge rule has been created for the main branch to trigger the pipeline.

Here's why:

The problem is that the CodePipeline is not triggering when code changes are pushed to the main branch of the CodeCommit repository. CodePipeline's integration with CodeCommit to trigger on repository events often relies on Amazon EventBridge (formerly CloudWatch Events). EventBridge rules detect events in AWS services and route them to targets, such as starting a CodePipeline execution.

If an EventBridge rule isn't properly configured to listen for CodeCommit repository changes on the main branch, the pipeline won't start automatically when changes are pushed. Therefore, verifying the existence and configuration of this rule is the most relevant first step in troubleshooting. The rule should be configured to detect referenceChanges events for the main branch in the CodeCommit repository and target the CodePipeline to start execution.

Option B is less likely to be the cause because if the CodePipeline service role lacked access to the CodeCommit repository, the pipeline would likely fail during execution rather than failing to trigger in the first place. While permissions are always important, trigger issues typically stem from EventBridge configuration.

Option C is incorrect because the developer's IAM role is relevant for pushing code to CodeCommit, not for triggering the pipeline. The developer was able to successfully push the changes, so their permissions are likely sufficient.

Option D is also less likely to be the primary cause. While CloudWatch Logs can contain information about pipeline failures, checking EventBridge first is more direct because the issue is that the pipeline is not starting at all. If the pipeline were failing during execution, CloudWatch Logs would be a more pertinent troubleshooting step.

Therefore, checking the EventBridge rule is the most direct and appropriate first step for troubleshooting this trigger-related issue.

Further research:

Using CloudWatch Events with CodePipeline
Amazon EventBridge

Question: 17

A company's developers use Amazon EC2 instances as remote workstations. The company is concerned that users can create or modify EC2 security groups to allow unrestricted inbound access.
A DevOps engineer needs to develop a solution to detect when users create unrestricted security group rules. The solution must detect changes to security group rules in near real time, remove unrestricted rules, and send email notifications to the security team. The DevOps engineer has created an AWS Lambda function that checks for security group ID from input, removes rules that grant unrestricted access, and sends notifications through Amazon Simple Notification Service (Amazon SNS).
What should the DevOps engineer do next to meet the requirements?

A.Configure the Lambda function to be invoked by the SNS topic. Create an AWS CloudTrail subscription for the SNS topic. Configure a subscription filter for security group modification events.

B.Create an Amazon EventBridge scheduled rule to invoke the Lambda function. Define a schedule pattern that runs the Lambda function every hour.

C.Create an Amazon EventBridge event rule that has the default event bus as the source. Define the rule’s event pattern to match EC2 security group creation and modification events. Configure the rule to invoke the Lambda function.

D.Create an Amazon EventBridge custom event bus that subscribes to events from all AWS services. Configure the Lambda function to be invoked by the custom event bus.

Answer: C

Explanation:

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

The requirement is to detect security group rule modifications in near real-time and take corrective actions. Amazon EventBridge (formerly CloudWatch Events) is the ideal service for this purpose because it allows you to react to state changes in your AWS environment. EventBridge can route events from AWS services to targets such as Lambda functions.

Option C:

EventBridge event rule with default event bus: The default event bus in EventBridge receives events from all AWS services in your account. This means events related to EC2 security group modifications will automatically flow into this bus.

Event pattern matching security group events: By defining an appropriate event pattern, the EventBridge rule can be configured to specifically target events related to the creation and modification of EC2 security groups. This is crucial to ensure that the Lambda function only triggers when relevant changes occur.

Invoke the Lambda function: Configuring the rule to invoke the Lambda function allows EventBridge to automatically execute the function whenever a matching security group event is detected. This ensures near real-time processing of security group changes.

Why other options are incorrect:

Option A: Using SNS to trigger the Lambda function based on CloudTrail logs is inefficient. CloudTrail logs are written periodically and are not a near real-time event source. Furthermore, CloudTrail sends logs to S3.

You will have to configure SNS subscriptions using S3 Event Notifications, which are not designed for near real-time monitoring. CloudTrail is better used for auditing, not immediate action.

Option B: A scheduled rule is not near real-time. The Lambda function would only run every hour, which does not satisfy the requirement of detecting changes "in near real-time."

Option D: While a custom event bus can be created, it is not necessary for this scenario. The default event bus already receives events from all AWS services, including EC2. Using a custom event bus adds complexity without providing any significant benefit in this case. Moreover, subscribing "events from all AWS Services" is not the standard way to utilize custom event bus, and in many cases, could be overly broad. Custom event buses are typically used for application events, or other isolated workloads.

In summary, EventBridge provides a scalable, event-driven mechanism to achieve near real-time detection and remediation of security group changes, fulfilling the requirements of the question.https://aws.amazon.com/eventbridge/https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-get-started.html

Question: 18

A DevOps engineer is creating an AWS CloudFormation template to deploy a web service. The web service will run on Amazon EC2 instances in a private subnet behind an Application Load Balancer (ALB). The DevOps engineer must ensure that the service can accept requests from clients that have IPv6 addresses.
What should the DevOps engineer do with the CloudFormation template so that IPv6 clients can access the web service?

A.Add an IPv6 CIDR block to the VPC and the private subnet for the EC2 instances. Create route table entries for the IPv6 network, use EC2 instance types that support IPv6, and assign IPv6 addresses to each EC2 instance.

B.Assign each EC2 instance an IPv6 Elastic IP address. Create a target group, and add the EC2 instances as targets. Create a listener on port 443 of the ALB, and associate the target group with the ALB.

C.Replace the ALB with a Network Load Balancer (NLB). Add an IPv6 CIDR block to the VPC and subnets for the NLB, and assign the NLB an IPv6 Elastic IP address.

D.Add an IPv6 CIDR block to the VPC and subnets for the ALB. Create a listener on port 443. and specify the dualstack IP address type on the ALB. Create a target group, and add the EC2 instances as targets. Associate the target group with the ALB.

Answer: D

Explanation:

The correct answer is D because it outlines the necessary steps to enable IPv6 support for an Application Load Balancer (ALB) and the underlying infrastructure within AWS CloudFormation.

Here's a detailed justification:

1. **IPv6 CIDR Block:** To enable IPv6, the VPC and subnets used by the ALB must have associated IPv6 CIDR blocks. This allows for the allocation of IPv6 addresses within those networks.
https://docs.aws.amazon.com/vpc/latest/userguide/get-started-ipv6.html

2. **Dualstack IP Address Type:** ALBs support both IPv4 and IPv6 through the dualstack IP address type. Specifying this on the ALB ensures that it can accept connections from both IPv4 and IPv6 clients.
The ALB will then forward the traffic to the EC2 instances via IPv4 (in this specific architecture).
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/application-load-balancers.html#load-balancer-attributes

3. **Listener on Port 443:** Creating a listener on port 443 (HTTPS) is essential for accepting secure connections from clients. The ALB needs to be configured to listen for incoming traffic on this port.

4. **Target Group and EC2 Instances:** The EC2 instances running the web service are registered as targets within a target group. The ALB then forwards traffic received on port 443 to the registered targets in the target group. This allows the ALB to distribute incoming requests among the available EC2 instances.

5. **Association of Target Group:** Associating the target group with the ALB's listener completes the setup, enabling the ALB to route incoming traffic to the EC2 instances.

Why other options are incorrect:

A: While adding IPv6 to VPC, subnets, and EC2 instances might seem logical, it's not necessary for the scenario where the ALB handles IPv6 termination and forwards traffic via IPv4 to the backend. Also it is not required to explicitly create route table entries for IPv6.

B: Assigning IPv6 Elastic IP addresses to EC2 instances directly and bypassing the ALB defeats the purpose of having a load balancer. The ALB's primary role is to distribute traffic and manage SSL/TLS termination.

C: Replacing the ALB with an NLB is not necessary. ALBs support IPv6 and provide more features for web applications. NLBs are better suited for TCP/UDP traffic and high-performance scenarios. It is also not required to assign an IPv6 Elastic IP to the NLB/ALB.

Question: 19

A company uses AWS Organizations and AWS Control Tower to manage all the company's AWS accounts. The company uses the Enterprise Support plan.
A DevOps engineer is using Account Factory for Terraform (AFT) to provision new accounts. When new accounts are provisioned, the DevOps engineer notices that the support plan for the new accounts is set to the Basic Support plan. The DevOps engineer needs to implement a solution to provision the new accounts with the Enterprise Support plan.
Which solution will meet these requirements?

A.Use an AWS Config conformance pack to deploy the account-part-of-organizations AWS Config rule and to automatically remediate any noncompliant accounts.

B.Create an AWS Lambda function to create a ticket for AWS Support to add the account to the Enterprise Support plan. Grant the Lambda function the support:ResolveCase permission.

C.Add an additional value to the control_tower_parameters input to set the AWSEnterpriseSupport parameter as the organization's management account number.

D.Set the aft_feature_enterprise_support feature flag to True in the AFT deployment input configuration. Redeploy AFT and apply the changes.

Answer: D

Explanation:

The correct solution is D: Set the aft_feature_enterprise_support feature flag to True in the AFT deployment input configuration. Redeploy AFT and apply the changes.

Here's why: Account Factory for Terraform (AFT) provides built-in features to customize account provisioning, and managing the support plan is one of those features. AFT utilizes feature flags to enable or disable certain behaviors during account creation. aft_feature_enterprise_support is a specific flag designed to control the support plan assigned to newly provisioned accounts.

Setting this flag to True instructs AFT to configure new accounts with the Enterprise Support plan. Redeploying AFT after modifying the input configuration applies the changes, ensuring that all subsequent accounts created through AFT will automatically inherit the Enterprise Support plan. This approach is efficient, automated, and directly leverages AFT's capabilities for account customization.

Option A is incorrect because AWS Config conformance packs and rules are primarily for auditing and remediating configuration drift after resources have been created. While it could technically be used to change the support plan after account creation, it's reactive rather than proactive and less efficient than configuring it directly during provisioning. Moreover, this is not the intended use case for AWS Config conformance packs.

Option B is incorrect as it involves creating a Lambda function to raise a support ticket, which is a manual and inefficient process. The support:ResolveCase permission is not appropriate here; creating a support ticket does not automatically grant resolution privileges. Furthermore, relying on manual intervention violates the principles of automation and infrastructure-as-code, which AFT is designed to facilitate.

Option C is incorrect. The control_tower_parameters input in AFT is meant to specify values for parameters that will be passed to the AWS Control Tower account baseline customization stack sets. Setting the AWSEnterpriseSupport parameter to the organization's management account number would not set the support plan and is not a valid use case of this feature.

Therefore, enabling the feature flag aft_feature_enterprise_support is the most direct, automated, and supported method for ensuring new accounts provisioned via AFT receive the Enterprise Support plan.

Refer to the AFT documentation for feature flag configurations: https://aws.amazon.com/blogs/mt/customize-your-aws-control-tower-account-factory-provisioned-accounts-using-account-factory-customizations/ (While this blog may not directly mention the feature flag, it details using AFT and account customizations.)

Question: 20

A company's DevOps engineer uses AWS Systems Manager to perform maintenance tasks during maintenance windows. The company has a few Amazon EC2 instances that require a restart after notifications from AWS Health. The DevOps engineer needs to implement an automated solution to remediate these notifications. The DevOps engineer creates an Amazon EventBridge rule.
How should the DevOps engineer configure the EventBridge rule to meet these requirements?

A.Configure an event source of AWS Health, a service of EC2. and an event type that indicates instance maintenance. Target a Systems Manager document to restart the EC2 instance.

B.Configure an event source of Systems Manager and an event type that indicates a maintenance window. Target a Systems Manager document to restart the EC2 instance.

C.Configure an event source of AWS Health, a service of EC2, and an event type that indicates instance maintenance. Target a newly created AWS Lambda function that registers an automation task to restart the EC2 instance during a maintenance window.

D.Configure an event source of EC2 and an event type that indicates instance maintenance. Target a newly created AWS Lambda function that registers an automation task to restart the EC2 instance during a maintenance window.

Answer: A

Explanation:

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

Requirement: The automation must trigger based on AWS Health events related to EC2 instance maintenance and restart the affected instances. The solution must leverage Systems Manager.

Option A directly addresses this.

It configures EventBridge to listen for AWS Health events specific to EC2 instance maintenance. This ensures that the rule triggers when AWS Health identifies a maintenance event requiring a restart.
It targets a Systems Manager document designed to restart EC2 instances. This fulfills the requirement to use Systems Manager for remediation.

Why other options are incorrect:

Option B: Systems Manager maintenance windows are for scheduled maintenance, not for reacting to AWS Health events. The trigger in this option is thus unrelated to AWS Health events.

Option C: While using Lambda to trigger a task is possible, it adds unnecessary complexity. A Systems Manager document can be directly triggered from EventBridge. Also, the question mentions maintenance tasks being used in windows.

Option D: While EC2 is related, the source of truth for planned maintenance impacting EC2 instances is AWS Health, not EC2 events themselves. AWS Health provides detailed notifications about planned maintenance impacting your AWS resources.

In this option, the registering of automation tasks makes little sense when you can directly target SSM document.

Conclusion: Option A is the most straightforward and efficient way to configure the EventBridge rule to trigger on AWS Health events related to EC2 instance maintenance and use a Systems Manager document to restart the instances, fulfilling all the requirements.

Supporting Links:

AWS Health:https://aws.amazon.com/health/
Amazon EventBridge:https://aws.amazon.com/eventbridge/
AWS Systems Manager:https://aws.amazon.com/systems-manager/
Automating AWS Health Events with EventBridge:https://aws.amazon.com/blogs/mt/automating-aws-health-events-with-amazon-eventbridge/

Question: 21

A company has containerized all of its in-house quality control applications. The company is running Jenkins on Amazon EC2 instances, which require patching and upgrading. The compliance officer has requested a DevOps engineer begin encrypting build artifacts since they contain company intellectual property.
What should the DevOps engineer do to accomplish this in the MOST maintainable manner?

A.Automate patching and upgrading using AWS Systems Manager on EC2 instances and encrypt Amazon EBS volumes by default.

B.Deploy Jenkins to an Amazon ECS cluster and copy build artifacts to an Amazon S3 bucket with default encryption enabled.

C.Leverage AWS CodePipeline with a build action and encrypt the artifacts using AWS Secrets Manager.

D.Use AWS CodeBuild with artifact encryption to replace the Jenkins instance running on EC2 instances.

Answer: D

Explanation:

The most maintainable solution is to migrate the CI/CD pipeline from Jenkins on EC2 to AWS CodeBuild with artifact encryption. Here's why:

Managed Service: CodeBuild is a fully managed build service, eliminating the need for manual patching, upgrading, and infrastructure maintenance of EC2 instances running Jenkins (as required in option A). This reduces operational overhead.

Built-in Encryption: CodeBuild natively supports artifact encryption using AWS KMS, fulfilling the compliance requirement without additional configuration complexity or reliance on external secrets management services as suggested in option C.

Scalability and Availability: CodeBuild automatically scales to handle build workloads, ensuring high availability and eliminating single points of failure associated with Jenkins on EC2.

Cost Optimization: Pay-as-you-go pricing model of CodeBuild is more cost-effective than running dedicated EC2 instances for Jenkins, especially during periods of low build activity.

Integration with AWS Ecosystem: CodeBuild integrates seamlessly with other AWS services like S3, CodePipeline, and CloudWatch, creating a streamlined DevOps workflow.

Security Best Practices: CodeBuild allows for secure storage of build artifacts and restricts access based on IAM roles and policies.

Option B introduces ECS, which adds complexity of container orchestration and does not directly address artifact encryption or Jenkins maintenance. Option C leverages AWS CodePipeline but necessitates external secrets management for encryption, which is natively supported in CodeBuild. Option A is incorrect since it still leaves the overhead of managing Jenkins on EC2, even with Systems Manager.

Supporting Links:

AWS CodeBuild: https://aws.amazon.com/codebuild/
AWS KMS: https://aws.amazon.com/kms/

Question: 22

An IT team has built an AWS CloudFormation template so others in the company can quickly and reliably deploy and terminate an application. The template creates an Amazon EC2 instance with a user data script to install the application and an Amazon S3 bucket that the application uses to serve static webpages while it is running.
All resources should be removed when the CloudFormation stack is deleted. However, the team observes that CloudFormation reports an error during stack deletion, and the S3 bucket created by the stack is not deleted. How can the team resolve the error in the MOST efficient manner to ensure that all resources are deleted without errors?

A.Add a DelelionPolicy attribute to the S3 bucket resource, with the value Delete forcing the bucket to be removed when the stack is deleted.

B.Add a custom resource with an AWS Lambda function with the DependsOn attribute specifying the S3 bucket, and an IAM role. Write the Lambda function to delete all objects from the bucket when RequestType is Delete.

C.Identify the resource that was not deleted. Manually empty the S3 bucket and then delete it.

D.Replace the EC2 and S3 bucket resources with a single AWS OpsWorks Stacks resource. Define a custom recipe for the stack to create and delete the EC2 instance and the S3 bucket.

Answer: B

Explanation:

The most efficient and programmatic solution to ensure an S3 bucket created by a CloudFormation stack is deleted along with the stack, even when it contains objects, is option B: Add a custom resource with an AWS Lambda function, DependsOn attribute, and an IAM role.

Here's why:

S3 Bucket Deletion Issues: S3 buckets must be empty before they can be deleted. CloudFormation often fails to delete S3 buckets directly if they contain objects, even if those objects were created by the same stack.

Option B's Solution: This option involves creating a custom resource in CloudFormation. A custom resource leverages a Lambda function to perform custom tasks during stack creation, update, or deletion.

Lambda Function Logic: The Lambda function is specifically designed to empty the S3 bucket during stack deletion. When CloudFormation initiates the stack deletion, it triggers the Lambda function with a RequestType of Delete. The Lambda code then iterates through the S3 bucket, deleting all objects within it. Once the bucket is empty, CloudFormation can successfully delete the bucket itself.

DependsOn Attribute: The DependsOn attribute is critical. It tells CloudFormation to execute the Lambda function before attempting to delete the S3 bucket. This ensures that the bucket is emptied before deletion is attempted.

IAM Role: The Lambda function requires an IAM role with permissions to list and delete objects within the S3 bucket. This ensures the function has the necessary authorization.

Efficiency and Automation: This solution is automated and programmatic. It doesn't require manual intervention to empty the bucket, making it highly efficient for repeated deployments and deletions.

Why other options are less ideal:

A:DeletionPolicy: Delete will only delete the bucket if it is empty. It does not address the core issue of objects preventing deletion.
C: Manually emptying the bucket is not scalable or automated. It is a manual process that introduces potential for error.
D: AWS OpsWorks is unnecessarily complex for this specific problem. Using CloudFormation with a custom resource provides a more streamlined and targeted solution. OpsWorks is designed for application management not basic resource creation and deletion.

Authoritative Links:

AWS CloudFormation Custom Resources:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-custom-resources.html
AWS Lambda:https://aws.amazon.com/lambda/
Amazon S3 Bucket Deletion:https://docs.aws.amazon.com/AmazonS3/latest/userguide/delete-bucket.html

Question: 23

A company has an AWS CodePipeline pipeline that is configured with an Amazon S3 bucket in the eu-west-1 Region. The pipeline deploys an AWS Lambda application to the same Region. The pipeline consists of an AWS CodeBuild project build action and an AWS CloudFormation deploy action.
The CodeBuild project uses the aws cloudformation package AWS CLI command to build an artifact that contains the Lambda function code's .zip file and the CloudFormation template. The CloudFormation deploy action references the CloudFormation template from the output artifact of the CodeBuild project's build action.
The company wants to also deploy the Lambda application to the us-east-1 Region by using the pipeline in eu-west-1. A DevOps engineer has already updated the CodeBuild project to use the aws cloudformation package command to produce an additional output artifact for us-east-1.
Which combination of additional steps should the DevOps engineer take to meet these requirements? (Choose two.)

A.Modify the CloudFormation template to include a parameter for the Lambda function code's zip file location. Create a new CloudFormation deploy action for us-east-1 in the pipeline. Configure the new deploy action to pass in the us-east-1 artifact location as a parameter override.

B.Create a new CloudFormation deploy action for us-east-1 in the pipeline. Configure the new deploy action to use the CloudFormation template from the us-east-1 output artifact.

C.Create an S3 bucket in us-east-1. Configure the S3 bucket policy to allow CodePipeline to have read and write access.

D.Create an S3 bucket in us-east-1. Configure S3 Cross-Region Replication (CRR) from the S3 bucket in eu-west-1 to the S3 bucket in us-east-1.

E.Modify the pipeline to include the S3 bucket for us-east-1 as an artifact store. Create a new CloudFormation deploy action for us-east-1 in the pipeline. Configure the new deploy action to use the CloudFormation template from the us-east-1 output artifact.

Answer: CE

Explanation:

Here's a detailed justification for why options C and E are the correct choices and why the other options are incorrect.

Why C is Correct:

Cross-Region Deployment Requires Artifact Storage in the Target Region: CodePipeline, by default, stores artifacts in a bucket within the same region as the pipeline. When deploying to a different region (us-east-1 in this case), you need a bucket in that region to store the artifacts intended for deployment there.

CodePipeline Access: CodePipeline needs permission to read the CloudFormation template and related artifacts from the destination S3 bucket in the target region. The bucket policy needs to be configured to grant CodePipeline the necessary permissions.

Reference: AWS Documentation - "Cross-region actions in AWS CodePipeline"
https://docs.aws.amazon.com/codepipeline/latest/userguide/actions-create-cross-region.html

Why E is Correct:

Artifact Store for Target Region: Since we are deploying to us-east-1, CodePipeline requires an artifact store (S3 bucket) in that region to stage the artifacts produced for that specific deployment. Adding the us-east-1 S3 bucket to the pipeline as an artifact store makes the artifacts available for the subsequent deployment action.

Dedicated CloudFormation Deploy Action: A new CloudFormation deploy action is needed within the pipeline to handle the deployment in the us-east-1 region.

Targeted Artifact Usage: The new CloudFormation deploy action should specifically utilize the CloudFormation template from the us-east-1 output artifact generated by the CodeBuild project. This ensures that the correct template and related artifacts are used for the deployment in the us-east-1 region.

Reference: AWS Documentation - "AWS CodePipeline Artifacts"
https://docs.aws.amazon.com/codepipeline/latest/userguide/concepts-artifacts.html

Why A is Incorrect:

While parameterization can be helpful in CloudFormation, modifying the template solely to pass the zip file location isn't sufficient for cross-region deployment. CodePipeline still needs an artifact store in the target region.

Why B is Incorrect:

Option B missed the important step about CodePipeline using the S3 bucket as an artifact store, and giving the necessary permissions to do so.

Why D is Incorrect:

While S3 Cross-Region Replication (CRR) copies objects between buckets, it doesn't inherently provide the isolation and control needed for CodePipeline in a multi-region deployment scenario. Relying solely on CRR might lead to unexpected behavior or synchronization issues, and would not be considered a best practice. CodePipeline requires a dedicated artifact store within the target region. It is not a substitute for artifact store in the target region.

Question: 24

A company runs an application on one Amazon EC2 instance. Application metadata is stored in Amazon S3 and must be retrieved if the instance is restarted. The instance must restart or relaunch automatically if the instance becomes unresponsive.
Which solution will meet these requirements?

A.Create an Amazon CloudWatch alarm for the StatusCheckFailed metric. Use the recover action to stop and start the instance. Use an S3 event notification to push the metadata to the instance when the instance is back up and running.

B.Configure AWS OpsWorks, and use the auto healing feature to stop and start the instance. Use a lifecycle event in OpsWorks to pull the metadata from Amazon S3 and update it on the instance.

C.Use EC2 Auto Recovery to automatically stop and start the instance in case of a failure. Use an S3 event notification to push the metadata to the instance when the instance is back up and running.

D.Use AWS CloudFormation to create an EC2 instance that includes the UserData property for the EC2 resource. Add a command in UserData to retrieve the application metadata from Amazon S3.

Answer: B

Explanation:

The correct answer is **B.** Let's break down why:

Requirement 1: Instance Restart/Relaunch on Unresponsiveness: Both EC2 Auto Recovery (option C) and CloudWatch alarms with recovery actions (option A) can handle instance restarts. OpsWorks auto healing (option B) also provides similar functionality. CloudFormation with UserData (option D) only addresses initial instance setup, not automatic recovery from failures.

Requirement 2: Metadata Retrieval from S3: This is where options A and C fall short. While S3 event notifications can trigger actions, they are best suited for reacting to object changes within S3, not reliably pushing data to an instance that has just recovered from a failure. The instance might not be fully ready to receive the notification at that exact moment.

Why OpsWorks Lifecycle Events are Ideal: OpsWorks provides a robust framework for managing application state during instance lifecycle events. Lifecycle events, such as `setup`, `configure`, and `deploy`, are triggered at specific points in the instance's life. By using a lifecycle event (e.g., `setup` after auto-healing restarts the instance), you can reliably pull the metadata from S3 after the instance has fully recovered and is ready to accept connections and execute commands. This guarantees that the metadata is retrieved correctly.
OpsWorks is designed to orchestrate and manage application state in a predictable and reliable way.

Auto Healing in OpsWorks: The auto healing feature in OpsWorks allows you to automatically replace failed instances with new ones. This feature is particularly useful for maintaining high availability and ensuring that your application is always running. https://docs.aws.amazon.com/opsworks/latest/userguide/workingstacks-autohealing.html

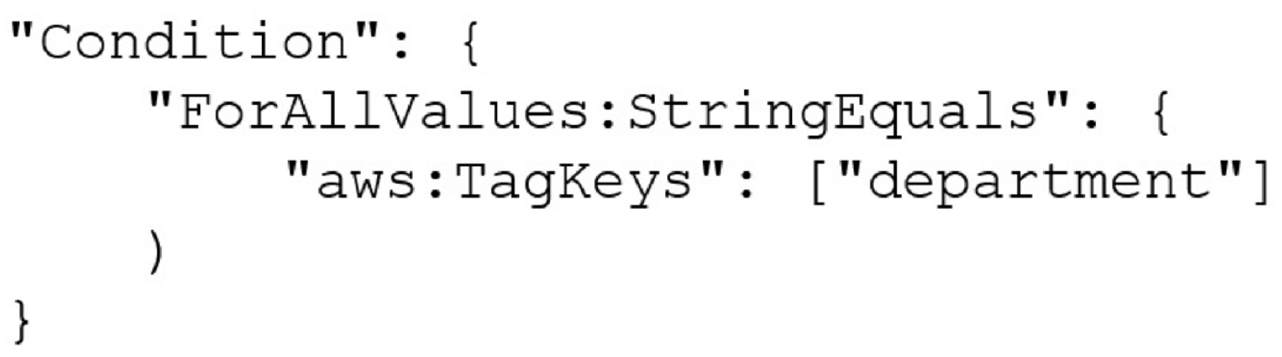
OpsWorks Lifecycle Events Documentation:
https://docs.aws.amazon.com/opsworks/latest/userguide/lifecycle-events.html

In summary, OpsWorks provides both the automatic instance recovery and a reliable mechanism (lifecycle events) to ensure the application metadata is retrieved from S3 after the instance is back online, satisfying both requirements of the question.

Question: 25

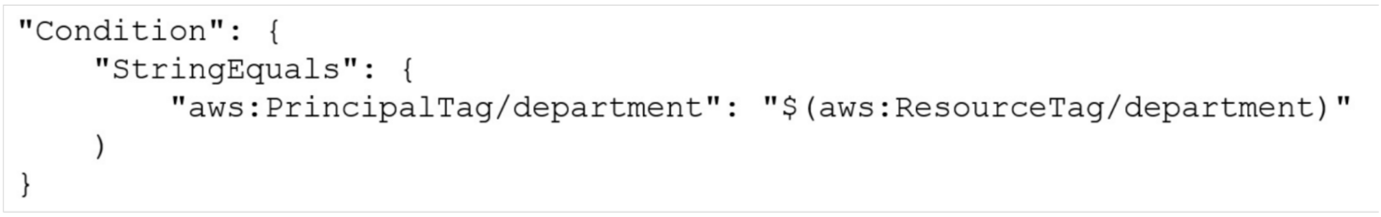
A company has multiple AWS accounts. The company uses AWS IAM Identity Center (AWS Single Sign-On) that is integrated with AWS Toolkit for Microsoft Azure DevOps. The attributes for access control feature is enabled in IAM Identity Center.
The attribute mapping list contains two entries. The department key is mapped to $ path:enterprise.department . The costCenter key is mapped to $ path:enterprise.costCenter .
All existing Amazon EC2 instances have a department tag that corresponds to three company departments (d1, d2, d3). A DevOps engineer must create policies based on the matching attributes. The policies must minimize administrative effort and must grant each Azure AD user access to only the EC2 instances that are tagged with the user's respective department name.
Which condition key should the DevOps engineer include in the custom permissions policies to meet these requirements?

A.



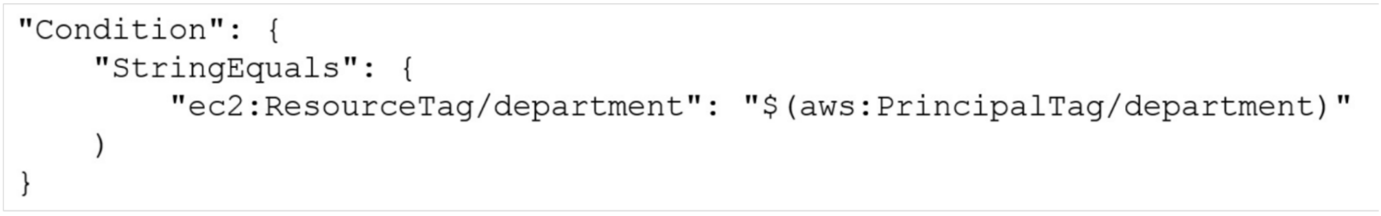
```
"Condition": {
     "ForAllValues:StringEquals": {
          "aws:TagKeys": ["department"]
     )
}
```

B.



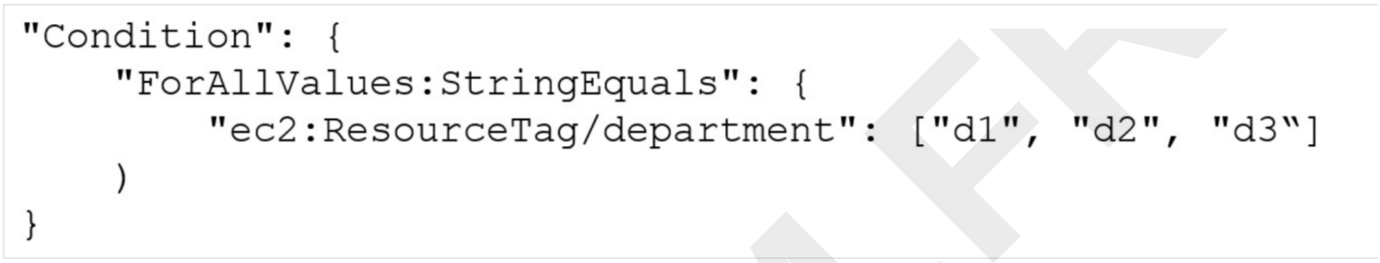
```
"Condition": {
    "StringEquals": {
        "aws:PrincipalTag/department": "$(aws:ResourceTag/department)"
    )
}
```

C.



```
"Condition": {
    "StringEquals": {
        "ec2:ResourceTag/department": "$(aws:PrincipalTag/department)"
    )
}
```

D.



```
"Condition": {
    "ForAllValues:StringEquals": {
        "ec2:ResourceTag/department": ["d1", "d2", "d3"]
    )
}
```

Answer: C

Explanation:

Reference:

https://aws.amazon.com/blogs/aws/new-attributes-based-access-control-with-aws-single-sign-on/

Question: 26

A company hosts a security auditing application in an AWS account. The auditing application uses an IAM role to access other AWS accounts. All the accounts are in the same organization in AWS Organizations.
A recent security audit revealed that users in the audited AWS accounts could modify or delete the auditing application's IAM role. The company needs to prevent any modification to the auditing application's IAM role by any entity other than a trusted administrator IAM role.
Which solution will meet these requirements?

A.Create an SCP that includes a Deny statement for changes to the auditing application's IAM role. Include a condition that allows the trusted administrator IAM role to make changes. Attach the SCP to the root of the organization.

B.Create an SCP that includes an Allow statement for changes to the auditing application's IAM role by the trusted administrator IAM role. Include a Deny statement for changes by all other IAM principals. Attach the SCP to the IAM service in each AWS account where the auditing application has an IAM role.

C.Create an IAM permissions boundary that includes a Deny statement for changes to the auditing application's IAM role. Include a condition that allows the trusted administrator IAM role to make changes. Attach the permissions boundary to the audited AWS accounts.

D.Create an IAM permissions boundary that includes a Deny statement for changes to the auditing application's IAM role. Include a condition that allows the trusted administrator IAM role to make changes. Attach the permissions boundary to the auditing application's IAM role in the AWS accounts.

Answer: A

Explanation:

The correct answer is A. Here's why:

The requirement is to prevent unauthorized modifications to a specific IAM role (used by the auditing application) across multiple AWS accounts within an organization. The chosen solution must allow a trusted administrator to still make necessary changes.

A: SCP with Deny and Allow Condition: Service Control Policies (SCPs) operate at the AWS Organizations level. By attaching an SCP to the root of the organization, we enforce centralized governance across all accounts within that organization. The SCP will include a Deny statement to prevent changes to the auditing application's IAM role for all principals except the trusted administrator role, for which a Condition will be added to allow changes. This fulfills the central control requirement and provides a mechanism for exceptions.
https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scp.html

B: SCP with Allow and Deny: SCPs act as guardrails, limiting permissions. Using an Allow statement in an SCP wouldn't grant permissions; instead, it specifies which actions are not explicitly denied. A Deny statement would be required to explicitly prevent the changes. Moreover, attaching the SCP to the IAM service is not feasible. SCPs are attached to accounts or organizational units (OUs).

C: IAM Permissions Boundary attached to audited accounts: Permissions boundaries set the maximum permissions an IAM role or user can have. However, attaching a permissions boundary to the audited AWS accounts would restrict what users in those accounts could do, not protect the auditing application's IAM role itself. This does not directly protect the role from changes within the account where it resides.
https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html

D: IAM Permissions Boundary attached to the auditing application's IAM role: While this would restrict the permissions of the auditing application's IAM role, it wouldn't prevent users in the AWS account where the role resides from modifying the role's trust policy or other attributes. The permissions boundary limits what the role can do, not who can modify it.

Therefore, SCPs with a Deny statement and a conditional Allow for the trusted administrator applied at the organization root provide the necessary centralized control and exception handling to meet the given requirements.

Question: 27

A company has an on-premises application that is written in Go. A DevOps engineer must move the application to AWS. The company's development team wants to enable blue/green deployments and perform A/B testing. Which solution will meet these requirements?

A.Deploy the application on an Amazon EC2 instance, and create an AMI of the instance. Use the AMI to create an automatic scaling launch configuration that is used in an Auto Scaling group. Use Elastic Load Balancing to distribute traffic. When changes are made to the application, a new AMI will be created, which will initiate an EC2 instance refresh.

B.Use Amazon Lightsail to deploy the application. Store the application in a zipped format in an Amazon S3 bucket. Use this zipped version to deploy new versions of the application to Lightsail. Use Lightsail deployment options to manage the deployment.

C.Use AWS CodeArtifact to store the application code. Use AWS CodeDeploy to deploy the application to a fleet of Amazon EC2 instances. Use Elastic Load Balancing to distribute the traffic to the EC2 instances. When making changes to the application, upload a new version to CodeArtifact and create a new CodeDeploy deployment.

D.Use AWS Elastic Beanstalk to host the application. Store a zipped version of the application in Amazon S3. Use that location to deploy new versions of the application. Use Elastic Beanstalk to manage the deployment options.

Answer: D

Explanation:

The correct answer is D, utilizing AWS Elastic Beanstalk. Elastic Beanstalk simplifies the deployment and management of web applications and services. It automates capacity provisioning, load balancing, auto-scaling, and application health monitoring, aligning perfectly with the DevOps engineer's need to move the Go application to AWS. The ability to store zipped application versions in Amazon S3 and deploy them using Elastic Beanstalk directly addresses the requirement for a streamlined deployment process.

Elastic Beanstalk's built-in support for blue/green deployments and A/B testing makes it an ideal choice. Blue/green deployments can be achieved using Elastic Beanstalk's environment swapping feature, allowing for zero-downtime updates. A/B testing can be implemented by routing a percentage of traffic to the new environment before a full swap. The other options are less ideal because option A requires manual AMI management and instance refreshes, increasing operational overhead. Option B, Amazon Lightsail, is more suited for simpler applications and lacks the advanced deployment management features necessary for blue/green deployments and A/B testing. Option C, using CodeArtifact and CodeDeploy with EC2, involves more manual configuration and management of the underlying infrastructure compared to Elastic Beanstalk, making it a more complex solution.

Elastic Beanstalk abstracts away much of the underlying infrastructure complexity, enabling the development team to focus on writing code and iteratively improving their application. This abstraction is crucial for enabling fast and reliable releases while minimizing the burden on the DevOps team. It facilitates smoother transitions between versions of the application.

Further reading:

AWS Elastic Beanstalk
Performing Blue/Green Deployments with Elastic Beanstalk

Question: 28

A developer is maintaining a fleet of 50 Amazon EC2 Linux servers. The servers are part of an Amazon EC2 Auto Scaling group, and also use Elastic Load Balancing for load balancing.
Occasionally, some application servers are being terminated after failing ELB HTTP health checks. The developer would like to perform a root cause analysis on the issue, but before being able to access application logs, the server is terminated.
How can log collection be automated?

A.Use Auto Scaling lifecycle hooks to put instances in a Pending:Wait state. Create an Amazon CloudWatch alarm for EC2 Instance Terminate Successful and trigger an AWS Lambda function that invokes an SSM Run Command script to collect logs, push them to Amazon S3, and complete the lifecycle action once logs are collected.

B.Use Auto Scaling lifecycle hooks to put instances in a Terminating:Wait state. Create an AWS Config rule for

EC2 Instance-terminate Lifecycle Action and trigger a step function that invokes a script to collect logs, push them to Amazon S3, and complete the lifecycle action once logs are collected.

C.Use Auto Scaling lifecycle hooks to put instances in a Terminating:Wait state. Create an Amazon CloudWatch subscription filter for EC2 Instance Terminate Successful and trigger a CloudWatch agent that invokes a script to collect logs, push them to Amazon S3, and complete the lifecycle action once logs are collected.

D.Use Auto Scaling lifecycle hooks to put instances in a Terminating:Wait state. Create an Amazon EventBridge rule for EC2 Instance-terminate Lifecycle Action and trigger an AWS Lambda function that invokes an SSM Run Command script to collect logs, push them to Amazon S3, and complete the lifecycle action once logs are collected.

Answer: D

Explanation:

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

Justification for Option D (Correct Answer):

Option D provides the most robust and efficient solution for capturing logs from EC2 instances that are being terminated by Auto Scaling. It leverages key AWS services to automate the entire process with minimal impact on the termination process.

1. **Auto Scaling Lifecycle Hooks (Terminating:Wait):** Lifecycle hooks are crucial. The Terminating:Wait state pauses the instance termination process, giving you a window to collect logs before the instance is gone. This is essential for successful log retrieval.
https://docs.aws.amazon.com/autoscaling/ec2/userguide/lifecycle-hooks.html

2. **Amazon EventBridge Rule:** EventBridge rules allow you to react to specific events within your AWS environment. Triggering on the EC2 Instance-terminate Lifecycle Action is ideal. This ensures that the log collection process is initiated immediately when an instance begins its termination.
https://docs.aws.amazon.com/eventbridge/latest/userguide/eventbridge-concepts.html

3. **AWS Lambda Function:** Lambda provides a serverless compute environment to execute your log collection logic. It can be triggered by EventBridge. Its integration with other AWS services (like SSM and S3) makes it easy to orchestrate the task.

4. **SSM Run Command:** SSM Run Command allows you to remotely execute commands on your EC2 instances in a secure manner. Using SSM to run a script that collects the logs is a reliable approach. This eliminates the need for the Lambda function to directly connect to the instance, improving security.
https://docs.aws.amazon.com/systems-manager/latest/userguide/execute-remote-commands.html

5. **Amazon S3:** S3 is a highly scalable and durable object storage service. Storing the collected logs in S3 provides a central and easily accessible repository for analysis.

6. **Lifecycle Hook Completion:** The Lambda function completing the lifecycle action after the logs are collected is critical. This allows the Auto Scaling group to continue the instance termination process and replace the unhealthy instance.

Why other options are incorrect:

Option A: Using Pending:Wait is incorrect. The instance is not yet terminating at that phase, so the lifecycle hook is not activated at the correct instance state. The CloudWatch alarm on EC2 instance termination might be late in detecting that the instance needs log collection, especially during rapid scaling events.

Option B: Using AWS Config rule to trigger is not a direct and efficient way to handle log collection. Config is typically used for compliance and configuration management. Step Functions can be used, but Lambda provides a simpler setup for this specific log collection scenario. Also, Config rules aren't the ideal mechanism to trigger actions on an EC2 terminate lifecycle event with the required immediacy.

Option C: CloudWatch subscription filters are used to process log data ingested into CloudWatch Logs. The scenario describes capturing logs from an instance before it terminates, not from logs that have already been written to CloudWatch Logs. It's also unclear how the CloudWatch agent would be triggered. Cloudwatch agent is to send the log data into cloudwatch not trigger actions.

Question: 29

A company has an organization in AWS Organizations. The organization includes workload accounts that contain enterprise applications. The company centrally manages users from an operations account. No users can be created in the workload accounts. The company recently added an operations team and must provide the operations team members with administrator access to each workload account.
Which combination of actions will provide this access? (Choose three.)

A.Create a SysAdmin role in the operations account. Attach the AdministratorAccess policy to the role. Modify the trust relationship to allow the sts:AssumeRole action from the workload accounts.

B.Create a SysAdmin role in each workload account. Attach the AdministratorAccess policy to the role. Modify the trust relationship to allow the sts:AssumeRole action from the operations account.

C.Create an Amazon Cognito identity pool in the operations account. Attach the SysAdmin role as an authenticated role.

D.In the operations account, create an IAM user for each operations team member.

E.In the operations account, create an IAM user group that is named SysAdmins. Add an IAM policy that allows the sts:AssumeRole action for the SysAdmin role in each workload account. Add all operations team members to the group.

F.Create an Amazon Cognito user pool in the operations account. Create an Amazon Cognito user for each operations team member.

Answer: BDE

Explanation:

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

B: Create a SysAdmin role in each workload account. Attach the AdministratorAccess policy to the role. Modify the trust relationship to allow the sts:AssumeRole action from the operations account. This is necessary because the operations team needs administrative privileges in each workload account. Creating a role with AdministratorAccess directly in each workload account and configuring its trust policy to allow the operations account to assume it is the foundation of providing that access. This approach follows the principle of least privilege by granting permissions only within the specific accounts that require administrative actions.

D: In the operations account, create an IAM user for each operations team member. Since users are centrally managed in the operations account and cannot be created in the workload accounts, creating IAM users in the operations account is a prerequisite. Each operations team member needs an individual identity to authenticate and assume the necessary roles in the workload accounts.

E: In the operations account, create an IAM user group that is named SysAdmins. Add an IAM policy that allows the sts:AssumeRole action for the SysAdmin role in each workload account. Add all operations team members to the group. Creating a group and assigning the necessary IAM policy to that group is a best practice. This policy will give the operations team members (who are part of the group) the ability to assume the SysAdmin role that you created in the workload accounts. The policy attached to the group will specify the necessary permissions, specifically the sts:AssumeRole action for the SysAdmin roles in each of the workload accounts. This follows the principle of least privilege because the users are only granted permissions they need and simplifies management of permissions.

Why other options are incorrect:

A: Creating a SysAdmin role in the operations account doesn't directly grant access to the workload accounts. This role could potentially be used to manage the operations account itself, but not the workloads within other accounts. The trust relationship modification described in option A wouldn't be the correct direction. The trust relationship needs to be modified in the workload accounts.
C: Amazon Cognito is designed for managing user authentication and authorization for web and mobile applications. It's not the appropriate solution for granting administrator access to IAM roles across different AWS accounts within an organization. Cognito is best suited for end-user authentication and authorization, not for internal operational access.
F: Amazon Cognito user pools are designed for managing end-user identities, not internal operations team access. Using Cognito in this context adds unnecessary complexity and doesn't align with best practices for managing IAM access within an AWS organization.

Supporting documentation:

IAM Roles:https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
IAM Policies:https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html
AWS Organizations:https://docs.aws.amazon.com/organizations/latest/userguide/orgs_getting-started_concepts.html
AssumeRole:https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_delegation.html

Question: 30

A company has multiple accounts in an organization in AWS Organizations. The company's SecOps team needs to receive an Amazon Simple Notification Service (Amazon SNS) notification if any account in the organization turns off the Block Public Access feature on an Amazon S3 bucket. A DevOps engineer must implement this change without affecting the operation of any AWS accounts. The implementation must ensure that individual member accounts in the organization cannot turn off the notification.
Which solution will meet these requirements?

A.Designate an account to be the delegated Amazon GuardDuty administrator account. Turn on GuardDuty for all accounts across the organization. In the GuardDuty administrator account, create an SNS topic. Subscribe the SecOps team's email address to the SNS topic. In the same account, create an Amazon EventBridge rule that uses an event pattern for GuardDuty findings and a target of the SNS topic.

B.Create an AWS CloudFormation template that creates an SNS topic and subscribes the SecOps team's email address to the SNS topic. In the template, include an Amazon EventBridge rule that uses an event pattern of CloudTrail activity for s3:PutBucketPublicAccessBlock and a target of the SNS topic. Deploy the stack to every account in the organization by using CloudFormation StackSets.

C.Turn on AWS Config across the organization. In the delegated administrator account, create an SNS topic. Subscribe the SecOps team's email address to the SNS topic. Deploy a conformance pack that uses the s3-bucket-level-public-access-prohibited AWS Config managed rule in each account and uses an AWS Systems Manager document to publish an event to the SNS topic to notify the SecOps team.

D.Turn on Amazon Inspector across the organization. In the Amazon Inspector delegated administrator account, create an SNS topic. Subscribe the SecOps team's email address to the SNS topic. In the same account, create an Amazon EventBridge rule that uses an event pattern for public network exposure of the S3 bucket and publishes an event to the SNS topic to notify the SecOps team.

Answer: C

Explanation:

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

Justification for Option C (Correct):

Option C leverages AWS Config, AWS Organizations, SNS, and SSM to provide a robust, centralized, and auditable solution for monitoring S3 bucket public access settings across multiple accounts.

1. **AWS Config for Continuous Monitoring:** Turning on AWS Config ensures continuous evaluation of resource configurations against desired states. This is crucial for detecting deviations from security policies, such as disabling Block Public Access on S3 buckets. https://aws.amazon.com/config/

2. **Delegated Administrator Account:** Centralizing management in a delegated administrator account (within AWS Organizations) allows for a single point of configuration and reporting, preventing individual member accounts from disabling the monitoring.

3. **SNS for Notification:** Creating an SNS topic in the delegated admin account and subscribing the SecOps team provides a reliable mechanism for real-time notifications when a violation occurs.
https://aws.amazon.com/sns/

4. **Conformance Packs & Managed Rules:** Deploying a conformance pack with the s3-bucket-level-public-access-prohibited AWS Config managed rule automates the compliance checking. This rule specifically evaluates whether the Block Public Access settings are enabled.
https://docs.aws.amazon.com/config/latest/developerguide/conformance-packs.html

5. **SSM Document for Event Publishing:** Using an SSM document allows AWS Config to trigger a remediation action when a violation is detected. In this case, the remediation is to publish an event to the SNS topic, notifying the SecOps team. This ensures that even if a violation is detected, the team is alerted in a timely manner.

Why Other Options Are Less Suitable:

Option A (GuardDuty): While GuardDuty is a great threat detection service, it primarily focuses on malicious activity and unauthorized behavior. Directly using GuardDuty findings for this specific S3 configuration monitoring is less efficient and precise than using Config. GuardDuty is designed to identify threats, not configuration changes.

Option B (CloudFormation StackSets with EventBridge): While StackSets enable deployment across accounts, relying solely on CloudTrail and EventBridge for monitoring s3:PutBucketPublicAccessBlock events can be complex to manage and potentially miss configuration changes due to timing issues or incomplete CloudTrail logging. It also doesn't ensure continuous compliance like AWS Config. Furthermore, managing CloudTrail events and event patterns across many accounts can be overhead.

Option D (Inspector): Amazon Inspector is primarily designed for vulnerability assessments and security audits, not continuous configuration monitoring. It focuses on finding software vulnerabilities, not detecting changes to S3 bucket public access settings.

Question: 31

A company has migrated its container-based applications to Amazon EKS and want to establish automated email notifications. The notifications sent to each email address are for specific activities related to EKS components. The solution will include Amazon SNS topics and an AWS Lambda function to evaluate incoming log events and publish messages to the correct SNS topic.
Which logging solution will support these requirements?

A.Enable Amazon CloudWatch Logs to log the EKS components. Create a CloudWatch subscription filter for each component with Lambda as the subscription feed destination.

B.Enable Amazon CloudWatch Logs to log the EKS components. Create CloudWatch Logs Insights queries linked to Amazon EventBridge events that invoke Lambda.

C.Enable Amazon S3 logging for the EKS components. Configure an Amazon CloudWatch subscription filter for each component with Lambda as the subscription feed destination.

D.Enable Amazon S3 logging for the EKS components. Configure S3 PUT Object event notifications with AWS Lambda as the destination.

Answer: A

Explanation:

Here's a detailed justification for why option A is the best solution for implementing automated email notifications for EKS components using SNS topics and Lambda, while also explaining why the other options are less suitable:

Why Option A is the Best Choice:

Option A proposes using Amazon CloudWatch Logs to capture logs from EKS components. This is the most logical starting point since CloudWatch Logs is a centralized logging service that readily integrates with AWS services like EKS. By enabling CloudWatch Logs, the logs can be aggregated and made searchable.

The critical part of option A is the use of CloudWatch subscription filters. Subscription filters act like real-time event detectors, scanning incoming log data for specific patterns or keywords related to each EKS component's activity. These filters are the "glue" that connects log events to specific SNS topics.

The Lambda function is configured as the target (or "destination") of the subscription filter. When a log event matches a subscription filter's criteria, CloudWatch Logs automatically invokes the Lambda function, passing the log event data as input.

Inside the Lambda function, logic is implemented to parse the log event data, determine which SNS topic is relevant based on the event's content, and then publish a message to that SNS topic. Subscribers to those SNS topics will then receive the email notifications.

CloudWatch subscription filters provide near real-time event processing, which satisfies the requirement for timely email notifications. The solution leverages the native integration between CloudWatch Logs and Lambda, providing an efficient and cost-effective mechanism.

Why Other Options are Less Suitable:

Option B: While CloudWatch Logs Insights can query logs, linking these queries to Amazon EventBridge is not a practical real-time solution for triggering notifications. CloudWatch Logs Insights is primarily for analyzing historical data, not for reacting to events as they happen. EventBridge is typically used for different purposes, such as inter-service communication, not for continuous log scraping.

Option C: S3 logging is often used for archiving logs, not for real-time event processing. Storing logs in S3 and then using CloudWatch subscription filters is inefficient. CloudWatch Logs is designed for real-time log processing. Using S3 introduces unnecessary complexity and delay.

Option D: S3 PUT Object event notifications can trigger Lambda functions when new log files are written to S3. However, this approach only reacts to the creation of a new log file, not individual log events within the files. To process specific log events, the Lambda function would have to parse the entire log file each time a new one is written, which is highly inefficient and doesn't provide timely, event-driven notifications.
Furthermore, EKS components typically don't directly write logs to S3 in a way that would make S3 event notifications the primary means of consuming logs. They generally write to stdout/stderr, which is then collected by a logging agent and sent to CloudWatch Logs or other logging backends.

In Summary:

Option A offers the most efficient and appropriate solution for implementing real-time, event-driven email notifications for EKS components using SNS topics and Lambda. It leverages the native capabilities of CloudWatch Logs for log aggregation, filtering, and integration with Lambda.

Authoritative Links:

Amazon CloudWatch Logs:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html
CloudWatch Subscription Filters:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html
Amazon SNS:https://aws.amazon.com/sns/
AWS Lambda:https://aws.amazon.com/lambda/

Question: 32

A company is implementing an Amazon Elastic Container Service (Amazon ECS) cluster to run its workload. The company architecture will run multiple ECS services on the cluster. The architecture includes an Application Load Balancer on the front end and uses multiple target groups to route traffic.
A DevOps engineer must collect application and access logs. The DevOps engineer then needs to send the logs to an Amazon S3 bucket for near-real-time analysis.
Which combination of steps must the DevOps engineer take to meet these requirements? (Choose three.)

A.Download the Amazon CloudWatch Logs container instance from AWS. Configure this instance as a task. Update the application service definitions to include the logging task.

B.Install the Amazon CloudWatch Logs agent on the ECS instances. Change the logging driver in the ECS task definition to awslogs.

C.Use Amazon EventBridge to schedule an AWS Lambda function that will run every 60 seconds and will run the Amazon CloudWatch Logs create-export-task command. Then point the output to the logging S3 bucket.

D.Activate access logging on the ALB. Then point the ALB directly to the logging S3 bucket.

E.Activate access logging on the target groups that the ECS services use. Then send the logs directly to the logging S3 bucket.

F.Create an Amazon Kinesis Data Firehose delivery stream that has a destination of the logging S3 bucket. Then create an Amazon CloudWatch Logs subscription filter for Kinesis Data Firehose.

Answer: BDF

Explanation:

The correct solution for collecting application and access logs from an ECS cluster and sending them to an S3 bucket for near-real-time analysis involves several key steps focusing on proper log capture, transportation, and access configuration.

Option B, "Install the Amazon CloudWatch Logs agent on the ECS instances. Change the logging driver in the ECS task definition to awslogs," is crucial for application logs. The awslogs logging driver sends container logs directly to CloudWatch Logs, making them easily accessible. Installing the CloudWatch Logs agent on each instance ensures log ingestion. This is more efficient than using a separate container instance (Option A), which would add unnecessary complexity.
[https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using_awslogs.html]

Option D, "Activate access logging on the ALB. Then point the ALB directly to the logging S3 bucket," correctly captures access logs generated by the Application Load Balancer (ALB). ALBs provide native access logging, which directly records requests made to the load balancer. Configuring the ALB to directly write these logs to the S3 bucket streamlines the process.
[https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-access-logs.html]

Option F, "Create an Amazon Kinesis Data Firehose delivery stream that has a destination of the logging S3 bucket. Then create an Amazon CloudWatch Logs subscription filter for Kinesis Data Firehose," establishes a real-time pipeline for log delivery. The CloudWatch Logs subscription filter directs logs matching a specific pattern from CloudWatch Logs to the Kinesis Data Firehose delivery stream. Kinesis Data Firehose then

efficiently streams these logs to the designated S3 bucket. This offers low latency and ensures logs are available for near-real-time analysis.
[https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_KinesisDataFirehose.html]

Option A is incorrect because managing a separate container instance for logging is overly complex. Option C involves scheduled Lambda functions to export logs, which is less efficient and not near real-time. Option E incorrectly focuses on target group access logs, which are not typically used for analysis compared to ALB access logs.

Question: 33

A company that uses electronic health records is running a fleet of Amazon EC2 instances with an Amazon Linux operating system. As part of patient privacy requirements, the company must ensure continuous compliance for patches for operating system and applications running on the EC2 instances.
How can the deployments of the operating system and application patches be automated using a default and custom repository?

A.Use AWS Systems Manager to create a new patch baseline including the custom repository. Run the AWS-RunPatchBaseline document using the run command to verify and install patches.

B.Use AWS Direct Connect to integrate the corporate repository and deploy the patches using Amazon CloudWatch scheduled events, then use the CloudWatch dashboard to create reports.

C.Use yum-config-manager to add the custom repository under /etc/yum.repos.d and run yum-config-manager-enable to activate the repository.

D.Use AWS Systems Manager to create a new patch baseline including the corporate repository. Run the AWS-AmazonLinuxDefaultPatchBaseline document using the run command to verify and install patches.

Answer: A

Explanation:

The correct answer is A because it leverages AWS Systems Manager Patch Manager, a service designed for automating OS and application patching on EC2 instances. Creating a new patch baseline allows the company to define the specific patches to approve or reject, and importantly, to include custom repositories where their application patches reside. The AWS-RunPatchBaseline document, when executed via Systems Manager Run Command, handles the patching process according to the defined baseline, ensuring continuous compliance.
This approach addresses the core requirement of automating patch deployments using both default and custom repositories.

Option B is incorrect. While AWS Direct Connect provides connectivity to corporate resources, it doesn't directly automate patch deployment. CloudWatch Scheduled Events could be used to trigger a patching process, but the process itself isn't defined, nor does it integrate custom repositories.

Option C is incorrect because while yum-config-manager is a valid tool for managing YUM repositories on Amazon Linux instances, this method does not provide the centralized control, auditability, and automated patching capabilities of Systems Manager Patch Manager. It also doesn't readily scale to a fleet of EC2 instances or provide compliance reporting.

Option D is incorrect because the AWS-AmazonLinuxDefaultPatchBaseline document only leverages the default Amazon Linux repositories. It does not support the inclusion of custom repositories, failing to meet the requirement of integrating application patches alongside OS patches.

Systems Manager Patch Manager offers several advantages, including centralized control, automated patching schedules, compliance reporting, and support for both default and custom patch sources. This aligns perfectly with the company's need for continuous compliance and automation of patch deployments using custom repositories for applications.

Supporting Links:

AWS Systems Manager Patch Manager:https://docs.aws.amazon.com/systems-manager/latest/userguide/patch-manager.html
AWS Systems Manager Run Command:https://docs.aws.amazon.com/systems-manager/latest/userguide/execute-remote-commands.html
Working with Patch Baselines:https://docs.aws.amazon.com/systems-manager/latest/userguide/patch-manager-baselines.html

Question: 34

A company is using AWS CodePipeline to automate its release pipeline. AWS CodeDeploy is being used in the pipeline to deploy an application to Amazon Elastic Container Service (Amazon ECS) using the blue/green deployment model. The company wants to implement scripts to test the green version of the application before shifting traffic. These scripts will complete in 5 minutes or less. If errors are discovered during these tests, the application must be rolled back.
Which strategy will meet these requirements?

A.Add a stage to the CodePipeline pipeline between the source and deploy stages. Use AWS CodeBuild to create a runtime environment and build commands in the buildspec file to invoke test scripts. If errors are found, use the aws deploy stop-deployment command to stop the deployment.

B.Add a stage to the CodePipeline pipeline between the source and deploy stages. Use this stage to invoke an AWS Lambda function that will run the test scripts. If errors are found, use the aws deploy stop-deployment command to stop the deployment.

C.Add a hooks section to the CodeDeploy AppSpec file. Use the AfterAllowTestTraffic lifecycle event to invoke an AWS Lambda function to run the test scripts. If errors are found, exit the Lambda function with an error to initiate rollback.

D.Add a hooks section to the CodeDeploy AppSpec file. Use the AfterAllowTraffic lifecycle event to invoke the test scripts. If errors are found, use the aws deploy stop-deployment CLI command to stop the deployment.

Answer: C

Explanation:

The correct answer is **C** because it leverages CodeDeploy's lifecycle hooks, specifically AfterAllowTestTraffic, to execute tests after traffic has been shifted to the green environment but before fully committing the change. This is the ideal point to validate the new deployment.

Here's a breakdown of why **C** is the best option and why the others are less suitable:

Option A: Adding a CodeBuild stage between the source and deploy stages is too early. The application isn't deployed to the green environment yet, so testing would be based on the built artifacts, not the running application in its target environment. Using aws deploy stop-deployment from CodeBuild to stop the entire deployment may not be ideal and more complex to implement.

Option B: Using a Lambda function in a CodePipeline stage offers more flexibility compared to CodeBuild for certain testing scenarios but shares the same timing problem as Option A. It runs before the application is deployed to the green environment. Also, aws deploy stop-deployment called from the Lambda would affect the entire deployment process.

Option C: Utilizing the AfterAllowTestTraffic hook in the CodeDeploy AppSpec.yml aligns perfectly with the requirement of testing the green environment after a portion of traffic has been shifted. This allows you to test the application with real-world traffic. If the Lambda function encounters errors, simply exiting with an error code will automatically trigger a rollback, as CodeDeploy is designed to handle this. This solution tightly integrates with the blue/green deployment model.

Option D: While using lifecycle hooks is correct, AfterAllowTraffic happens after all traffic has shifted to the green environment. If errors are found at this point, rolling back could cause a service disruption for all users. The AfterAllowTestTraffic event is specifically designed for pre-production validation. Also, you cannot stop the deployment using CLI from within the CodeDeploy lifecycle event.

In summary, option C ensures tests are executed in the correct environment (green), at the correct point in the deployment lifecycle (after test traffic is shifted), and leverages the built-in rollback mechanism of CodeDeploy upon test failure. This strategy minimizes potential disruption and provides a robust mechanism for verifying the new deployment.

Relevant links:

AWS CodeDeploy AppSpec File Reference:
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html Blue/Green Deployments with CodeDeploy:
https://docs.aws.amazon.com/codedeploy/latest/userguide/traffic-management-blue-green.html AWS Lambda: https://aws.amazon.com/lambda/

Question: 35

A company uses AWS Storage Gateway in file gateway mode in front of an Amazon S3 bucket that is used by multiple resources. In the morning when business begins, users do not see the objects processed by a third party the previous evening. When a DevOps engineer looks directly at the S3 bucket, the data is there, but it is missing in Storage Gateway.
Which solution ensures that all the updated third-party files are available in the morning?

A.Configure a nightly Amazon EventBridge event to invoke an AWS Lambda function to run the RefreshCache command for Storage Gateway.

B.Instruct the third party to put data into the S3 bucket using AWS Transfer for SFTP.

C.Modify Storage Gateway to run in volume gateway mode.

D.Use S3 Same-Region Replication to replicate any changes made directly in the S3 bucket to Storage Gateway.

Answer: A

Explanation:

The problem describes a situation where files added to an S3 bucket by a third party overnight are not visible to users accessing the bucket through AWS Storage Gateway's file gateway mode the next morning. The root cause is that Storage Gateway's cache is not automatically updated with the new objects written directly to S3.

Option A is the correct solution because it proactively addresses the cache staleness issue. Using Amazon EventBridge to schedule a nightly event that triggers an AWS Lambda function to execute the RefreshCache command for the Storage Gateway ensures that the gateway's cache is synchronized with the latest state of the S3 bucket every morning. This is a suitable automated solution for keeping the gateway's view of the S3 bucket consistent. This command is designed for this very scenario.

Option B is incorrect because even if the third party uses AWS Transfer for SFTP, the data will still be written directly to the S3 bucket, and Storage Gateway's cache will still need to be refreshed to reflect these changes. Transfer for SFTP does not directly address the cache synchronization problem.

Option C, changing to volume gateway mode, is not the correct approach. Volume gateway mode presents block-based storage to on-premises applications, which is not the required functionality. The file gateway is appropriate since the resources utilize a file system interface. Switching mode would necessitate a significant architectural change unrelated to the cache refresh issue.

Option D, using S3 Same-Region Replication, is also incorrect in this scenario. S3 replication is designed to replicate objects between S3 buckets, but it does not automatically update Storage Gateway's cache. Replication would only copy the data to another bucket; the original problem of the gateway's stale cache would persist.

Therefore, the best solution is to schedule a periodic refresh of the Storage Gateway's cache to ensure consistency between the S3 bucket and the files accessible through the gateway.

Supporting documentation:

AWS Storage Gateway RefreshCache:
https://docs.aws.amazon.com/storagegateway/latest/userguide/HowFileGatewayWorks.html (See section about "How File Gateway Works" and caching behaviour.)
AWS Lambda:https://aws.amazon.com/lambda/
Amazon EventBridge:https://aws.amazon.com/eventbridge/

Question: 36

A DevOps engineer needs to back up sensitive Amazon S3 objects that are stored within an S3 bucket with a private bucket policy using S3 cross-Region replication functionality. The objects need to be copied to a target bucket in a different AWS Region and account.
Which combination of actions should be performed to enable this replication? (Choose three.)

A.Create a replication IAM role in the source account
B.Create a replication I AM role in the target account.
C.Add statements to the source bucket policy allowing the replication IAM role to replicate objects.
D.Add statements to the target bucket policy allowing the replication IAM role to replicate objects.
E.Create a replication rule in the source bucket to enable the replication.
F.Create a replication rule in the target bucket to enable the replication.

Answer: ADE

Explanation:

The correct answer is ADE. Here's why:

A. Create a replication IAM role in the source account: S3 Cross-Region Replication (CRR) needs an IAM role in the source account to perform the replication actions. This role is assumed by S3 on behalf of the source account. This role needs permissions to read objects from the source bucket and write them to the destination bucket. The official documentation confirms the necessity of an IAM role in the source account. (AWS Documentation: https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication-howto-setup.html)

D. Add statements to the source bucket policy allowing the replication IAM role to replicate objects: The source bucket policy needs to grant the replication IAM role permissions to access objects in the source bucket. Specifically, s3:GetObjectVersion permission is required to read objects and their versions. This policy ensures that only the intended replication role can access the data for replication purposes. It also ensures there's explicit permission granted for the replication to occur. (AWS Documentation: https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication-add-bucket-policy.html)

E. Create a replication rule in the source bucket to enable the replication: A replication rule is configured on the source bucket. This rule specifies which objects should be replicated (based on prefixes or tags) and the destination bucket where the objects should be copied. The replication rule also specifies the IAM role that S3 will use to perform the replication. Without a replication rule, S3 won't know which objects to replicate or where to send them. (AWS Documentation: https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication-configure-bucket.html)

Why the other options are incorrect:

B. Create a replication IAM role in the target account: While the target account needs an S3 bucket policy allowing the source account (specifically the replication role in the source account) to write objects, a dedicated IAM role in the target account for replication isn't strictly necessary for basic cross-account replication. The IAM role in the source account performs the replication actions, and the target bucket policy trusts this source account's role. While resource-based policies can exist in the destination bucket, it's not the core component driving the functionality.

C. Add statements to the target bucket policy allowing the replication IAM role to replicate objects: Although the bucket policy on the destination bucket needs to allow the role from the source account to write to it, the statements need to be added to the source bucket policy instead to grant the role permission to read the objects. The target bucket policy is for writing operations by the IAM role from the source account, not for the role itself. Therefore this option is not completely correct.

F. Create a replication rule in the target bucket to enable the replication: Replication rules are only configured on the source bucket, defining the replication behavior. Target buckets do not have replication rules associated with them.

Question: 37

A company has multiple member accounts that are part of an organization in AWS Organizations. The security team needs to review every Amazon EC2 security group and their inbound and outbound rules. The security team wants to programmatically retrieve this information from the member accounts using an AWS Lambda function in the management account of the organization.
Which combination of access changes will meet these requirements? (Choose three.)

A.Create a trust relationship that allows users in the member accounts to assume the management account IAM role.

B.Create a trust relationship that allows users in the management account to assume the IAM roles of the member accounts.

C.Create an IAM role in each member account that has access to the AmazonEC2ReadOnlyAccess managed policy.

D.Create an I AM role in each member account to allow the sts:AssumeRole action against the management account IAM role's ARN.

E.Create an I AM role in the management account that allows the sts:AssumeRole action against the member account IAM role's ARN.

F.Create an IAM role in the management account that has access to the AmazonEC2ReadOnlyAccess managed policy.

Answer: BCE

Explanation:

The correct answer is BCE because it outlines the necessary steps to enable cross-account access for the Lambda function in the management account to retrieve security group information from the member accounts.

B. Create a trust relationship that allows users in the management account to assume the IAM roles of the member accounts: This is crucial for cross-account access. The trust relationship, defined in the member account's IAM role, specifies which principals (in this case, the management account's IAM role) are allowed to assume that role. Without this trust, the management account's Lambda function cannot gain temporary access to the member accounts.

C. Create an IAM role in each member account that has access to the AmazonEC2ReadOnlyAccess managed policy: This ensures that when the management account's Lambda function assumes the role in the member account, it has the necessary permissions to retrieve the security group information. The AmazonEC2ReadOnlyAccess policy grants read-only access to EC2 resources, including security groups and their rules.

E. Create an IAM role in the management account that allows the sts:AssumeRole action against the member account IAM role's ARN: This allows the Lambda function in the management account to actually execute the AssumeRole API call against the member account's IAM role. Without this permission, the Lambda function would be denied access even if the member account's IAM role trusts the management account. This ensures that the management account has permission to assume the IAM role in the member accounts.

Why other options are incorrect:

A. Create a trust relationship that allows users in the member accounts to assume the management account IAM role: This is the inverse of what's needed. The management account needs to access the member accounts, not the other way around.

D. Create an IAM role in each member account to allow the sts:AssumeRole action against the management account IAM role's ARN: The member accounts don't need to assume a role in the management account in this scenario.

F. Create an IAM role in the management account that has access to the AmazonEC2ReadOnlyAccess managed policy: While having some permissions in the management account might be necessary for general operations, it doesn't grant access to resources in the member accounts. Cross-account access is required, and this option doesn't address that.

Supporting Concepts:

IAM Roles: Provide temporary security credentials for users or services to access AWS resources. **Trust Relationships:** Define which principals are allowed to assume a specific IAM role.

AWS Organizations: Enables centralized management and governance across multiple AWS accounts. **Cross-Account Access:** Allows resources in one AWS account to access resources in another account. **AssumeRole:** An AWS STS API action that allows a principal to assume a role in another AWS account.

Authoritative Links:

IAM Roles:https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
AssumeRole:https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html
Cross-Account Access:https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_crossaccount.html **AWS Organizations:**https://aws.amazon.com/organizations/

Question: 38

A space exploration company receives telemetry data from multiple satellites. Small packets of data are received through Amazon API Gateway and are placed directly into an Amazon Simple Queue Service (Amazon SQS) standard queue. A custom application is subscribed to the queue and transforms the data into a standard format.
Because of inconsistencies in the data that the satellites produce, the application is occasionally unable to transform the data. In these cases, the messages remain in the SQS queue. A DevOps engineer must develop a solution that retains the failed messages and makes them available to scientists for review and future processing. Which solution will meet these requirements?

A.Configure AWS Lambda to poll the SQS queue and invoke a Lambda function to check whether the queue messages are valid. If validation fails, send a copy of the data that is not valid to an Amazon S3 bucket so that the scientists can review and correct the data. When the data is corrected, amend the message in the SQS queue by using a replay Lambda function with the corrected data.

B.Convert the SQS standard queue to an SQS FIFO queue. Configure AWS Lambda to poll the SQS queue every 10 minutes by using an Amazon EventBridge schedule. Invoke the Lambda function to identify any messages with a SentTimestamp value that is older than 5 minutes, push the data to the same location as the application's output location, and remove the messages from the queue.

C.Create an SQS dead-letter queue. Modify the existing queue by including a redrive policy that sets the Maximum Receives setting to 1 and sets the dead-letter queue ARN to the ARN of the newly created queue. Instruct the scientists to use the dead-letter queue to review the data that is not valid. Reprocess this data at a later time.

D.Configure API Gateway to send messages to different SQS virtual queues that are named for each of the satellites. Update the application to use a new virtual queue for any data that it cannot transform, and send the message to the new virtual queue. Instruct the scientists to use the virtual queue to review the data that is not valid. Reprocess this data at a later time.

Answer: C

Explanation:

The correct answer is **C** because it directly addresses the requirements of retaining failed messages for review and future processing using a standard and efficient SQS feature: a dead-letter queue (DLQ).

Here's a breakdown:

Problem: Telemetry data from satellites sometimes fails transformation, leaving unusable messages in the SQS queue. We need to preserve these failed messages for scientists to review and reprocess.
Solution C (DLQ):

Creating a DLQ provides a designated place to move messages that have failed processing after a specified number of attempts.

The redrive policy moves messages to the DLQ after a single failed attempt (Maximum Receives = 1), ensuring that problematic messages are quickly isolated for investigation.

Scientists can then access the DLQ to review the problematic data without impacting the main processing flow.

DLQs are a best practice for handling errors and ensuring data durability in asynchronous messaging systems.

https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html

Why other options are not the best fit:

A (Lambda for validation): While Lambda can validate messages, this solution is more complex. Adding validation to Lambda also adds overhead to the processing flow, and re-injecting the data via a replay Lambda function introduces additional complexity. The simplest solution is to leverage SQS DLQ.

B (FIFO and Lambda): Converting to a FIFO queue is unnecessary. FIFO queues are primarily for maintaining message order, which is not stated as a requirement. The suggested Lambda function introduces complexity and polling introduces latencies.

D (API Gateway and Virtual Queues): Using virtual queues adds architectural complexity. It also requires changes to the application code and API Gateway configuration. It makes it harder to manage and monitor the errors compared to a dedicated DLQ that SQS provides.

In conclusion, using an SQS dead-letter queue with a redrive policy is the simplest, most efficient, and best-practice approach to retaining failed messages for review and future processing. This aligns with the DevOps principles of automation, efficiency, and recoverability.

Question: 39

A company wants to use AWS CloudFormation for infrastructure deployment. The company has strict tagging and resource requirements and wants to limit the deployment to two Regions. Developers will need to deploy multiple versions of the same application.
Which solution ensures resources are deployed in accordance with company policy?

A.Create AWS Trusted Advisor checks to find and remediate unapproved CloudFormation StackSets.

B.Create a Cloud Formation drift detection operation to find and remediate unapproved CloudFormation StackSets.

C.Create CloudFormation StackSets with approved CloudFormation templates.

D.Create AWS Service Catalog products with approved CloudFormation templates.

Answer: D

Explanation:

The correct answer is D, creating AWS Service Catalog products with approved CloudFormation templates. Here's why:

AWS Service Catalog allows organizations to create and manage catalogs of IT services that are approved for use. By creating Service Catalog products from approved CloudFormation templates, the company can ensure developers only deploy infrastructure that adheres to corporate policies, including tagging, resource requirements, and regional limitations. Service Catalog provides centralized governance and control over the infrastructure provisioning process. This is crucial for enforcing standardization and compliance. Each product can encapsulate a specific version of the application's infrastructure.

StackSets (Option C) is a valid option for deploying infrastructure across multiple accounts and regions. However, StackSets alone don't inherently enforce the use of approved templates. It's up to the user to select the template, which could lead to policy violations if developers are allowed to use arbitrary templates. While StackSets allow for centralized deployment, they lack the governance features of Service Catalog that would force compliance.

Trusted Advisor (Option A) provides recommendations for cost optimization, security, fault tolerance, service limits, and performance improvement but does not directly restrict infrastructure deployments or enforce the use of approved templates. Trusted Advisor is primarily for auditing and identifying violations after the fact rather than preventing them during deployment. Remediation would be reactive and manual.

CloudFormation Drift Detection (Option B) is also a reactive approach. It identifies differences between the expected state (defined in the template) and the actual state of the deployed resources. While it's useful for detecting unauthorized modifications, it doesn't prevent the initial deployment of non-compliant infrastructure. It only alerts after non-compliant resources have been deployed.

Service Catalog, in contrast, proactively enforces compliance by restricting developers to using only approved CloudFormation templates within the catalog. This approach satisfies all the requirements: strict tagging and resource control, regional limitation, and support for deploying multiple application versions with governance. The other options are either reactive or don't provide the level of centralized control needed to enforce policies consistently.

Authoritative links:

AWS Service Catalog: https://aws.amazon.com/servicecatalog/
AWS CloudFormation: https://aws.amazon.com/cloudformation/
AWS CloudFormation StackSets: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/what-is-stacksets.html

Question: 40

A company requires that its internally facing web application be highly available. The architecture is made up of one Amazon EC2 web server instance and one NAT instance that provides outbound internet access for updates and accessing public data.
Which combination of architecture adjustments should the company implement to achieve high availability?
(Choose two.)

A.Add the NAT instance to an EC2 Auto Scaling group that spans multiple Availability Zones. Update the route tables.

B.Create additional EC2 instances spanning multiple Availability Zones. Add an Application Load Balancer to split the load between them.

C.Configure an Application Load Balancer in front of the EC2 instance. Configure Amazon CloudWatch alarms to recover the EC2 instance upon host failure.

D.Replace the NAT instance with a NAT gateway in each Availability Zone. Update the route tables.

E.Replace the NAT instance with a NAT gateway that spans multiple Availability Zones. Update the route tables.

Answer: BD

Explanation:

Here's a detailed justification for why options B and D are the correct choices for achieving high availability in the described web application architecture:

Option B: Create additional EC2 instances spanning multiple Availability Zones. Add an Application Load Balancer to split the load between them.

This option addresses the single point of failure presented by the single EC2 web server instance. By deploying multiple EC2 instances across different Availability Zones (AZs), the application becomes resilient to failures within a specific AZ. If one AZ experiences an outage, the other instances in the remaining AZs can continue to serve traffic. An Application Load Balancer (ALB) distributes incoming traffic evenly across these healthy instances. The ALB also performs health checks, ensuring that only healthy instances receive traffic, further enhancing availability. This setup ensures that the application remains accessible even if one or more EC2 instances or entire Availability Zones experience issues. Using the ALB offers features like session persistence and health checks, vital for a highly available web application.

Option D: Replace the NAT instance with a NAT gateway in each Availability Zone. Update the route tables.

The single NAT instance is another single point of failure. If the NAT instance fails, the EC2 web server instance loses its ability to access the internet, which, as stated, is needed for updates and accessing public data. Replacing the NAT instance with a NAT gateway in each Availability Zone provides redundancy and eliminates this bottleneck. NAT Gateways are managed by AWS and are designed for high availability and scalability. Each NAT Gateway operates independently within its AZ. By placing a NAT Gateway in each AZ where the EC2 instances reside and updating the route tables to point to the respective NAT Gateway, the application maintains outbound internet access even if one AZ fails. This is because each EC2 instance will use the NAT Gateway within its own AZ.

Why other options are not suitable:

Option A: While placing a NAT instance in an Auto Scaling group improves the NAT instance's resilience, it's less effective than NAT Gateways in terms of availability and management overhead. Furthermore, it doesn't address the primary concern of the single EC2 web server instance.

Option C: Recovering an EC2 instance upon host failure with CloudWatch alarms helps but doesn't provide high availability. There's downtime during the recovery process. Also, this doesn't solve the problem of a single point of failure in the architecture.

Option E: While replacing the NAT instance with a NAT gateway is a step in the right direction, a single NAT gateway spanning multiple AZs isn't a standard or highly recommended AWS architecture. NAT Gateways are designed to be zonal resources. Placing a NAT gateway in each AZ offers superior redundancy.

Authoritative Links:

Application Load Balancer:
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html
NAT Gateway:https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html
High Availability in AWS:https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-28/pillar/AWS-Reliability

Question: 41

A DevOps engineer is building a multistage pipeline with AWS CodePipeline to build, verify, stage, test, and deploy an application. A manual approval stage is required between the test stage and the deploy stage. The development team uses a custom chat tool with webhook support that requires near-real-time notifications.
How should the DevOps engineer configure status updates for pipeline activity and approval requests to post to the chat tool?

A.Create an Amazon CloudWatch Logs subscription that filters on CodePipeline Pipeline Execution State Change. Publish subscription events to an Amazon Simple Notification Service (Amazon SNS) topic. Subscribe the chat webhook URL to the SNS topic, and complete the subscription validation.

B.Create an AWS Lambda function that is invoked by AWS CloudTrail events. When a CodePipeline Pipeline Execution State Change event is detected, send the event details to the chat webhook URL.

C.Create an Amazon EventBridge rule that filters on CodePipeline Pipeline Execution State Change. Publish the events to an Amazon Simple Notification Service (Amazon SNS) topic. Create an AWS Lambda function that sends event details to the chat webhook URL. Subscribe the function to the SNS topic.

D.Modify the pipeline code to send the event details to the chat webhook URL at the end of each stage. Parameterize the URL so that each pipeline can send to a different URL based on the pipeline environment.

Answer: C

Explanation:

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

Event-Driven Architecture: The scenario requires near real-time notifications, making an event-driven architecture the most suitable approach. AWS CodePipeline state changes are events that can trigger actions.

Amazon EventBridge: EventBridge is designed for routing events between AWS services and custom applications. It provides the necessary filtering capabilities to specifically target CodePipeline state change events.
https://aws.amazon.com/eventbridge/

Filtering: EventBridge rules can filter events based on specific criteria, such as pipeline name, stage name, and state. This ensures that only relevant events (e.g., pipeline execution state changes, approval requests) trigger notifications.

Amazon SNS: SNS is a messaging service that enables decoupled communication between services. By publishing events to an SNS topic, multiple subscribers can receive notifications.
https://aws.amazon.com/sns/

AWS Lambda for Transformation & Delivery: Directly publishing EventBridge events to a chat webhook might not be ideal due to formatting requirements. A Lambda function can subscribe to the SNS topic, receive the event details, transform the data into the format expected by the chat tool's webhook, and then send the formatted message to the webhook URL. This provides flexibility in adapting the event data to the specific API of the chat tool. https://aws.amazon.com/lambda/

Near Real-Time: EventBridge, SNS, and Lambda work together to provide near real-time notifications as soon as a relevant pipeline event occurs.

Why other options are incorrect:

A: CloudWatch Logs are primarily for log data, not general event routing. While you can use CloudWatch Logs subscriptions to react to events, EventBridge is the dedicated service for event routing.

B: CloudTrail logs API calls and changes to resources. While it can be used for pipeline state changes, it's less efficient and more resource-intensive than EventBridge for this specific purpose. CloudTrail is also not designed for near-real time.

D: Modifying the pipeline code directly is not a scalable or maintainable solution. It tightly couples the pipeline logic with the notification mechanism, making it harder to manage and update notifications for multiple pipelines or different chat tools. It also introduces redundancy by requiring modification of each pipeline.

Question: 42

A company's application development team uses Linux-based Amazon EC2 instances as bastion hosts. Inbound SSH access to the bastion hosts is restricted to specific IP addresses, as defined in the associated security groups. The company's security team wants to receive a notification if the security group rules are modified to allow SSH access from any IP address.
What should a DevOps engineer do to meet this requirement?

A.Create an Amazon EventBridge rule with a source of aws.cloudtrail and the event name AuthorizeSecurityGroupIngress. Define an Amazon Simple Notification Service (Amazon SNS) topic as the target.

B.Enable Amazon GuardDuty and check the findings for security groups in AWS Security Hub. Configure an Amazon EventBridge rule with a custom pattern that matches GuardDuty events with an output of NON_COMPLIANT. Define an Amazon Simple Notification Service (Amazon SNS) topic as the target.

C.Create an AWS Config rule by using the restricted-ssh managed rule to check whether security groups disallow unrestricted incoming SSH traffic. Configure automatic remediation to publish a message to an Amazon Simple Notification Service (Amazon SNS) topic.

D.Enable Amazon Inspector. Include the Common Vulnerabilities and Exposures-1.1 rules package to check the security groups that are associated with the bastion hosts. Configure Amazon Inspector to publish a message to an Amazon Simple Notification Service (Amazon SNS) topic.

Answer: A

Explanation:

The correct answer is A. Here's why:

A. Create an Amazon EventBridge rule with a source of aws.cloudtrail and the event name AuthorizeSecurityGroupIngress. Define an Amazon Simple Notification Service (Amazon SNS) topic as the target.

This solution directly addresses the requirement of detecting modifications to security group rules that allow SSH access from any IP address.

CloudTrail: AWS CloudTrail logs API calls made within your AWS account. Modifying security group rules triggers AuthorizeSecurityGroupIngress API calls.

EventBridge: EventBridge allows you to create rules that trigger actions based on specific events. In this case, the rule listens for the AuthorizeSecurityGroupIngress event from CloudTrail.

Filtering: The EventBridge rule can be configured to filter specifically for AuthorizeSecurityGroupIngress events that modify security group rules to allow SSH (port 22) from any IP address (0.0.0.0/0). This level of filtering is crucial to avoid unnecessary notifications for other security group changes.

SNS: Amazon SNS is a messaging service used to send notifications. The EventBridge rule is configured to send a notification to an SNS topic whenever a matching security group modification event occurs. Subscribers to this SNS topic (e.g., the security team) will receive the notification.

Why other options are not as suitable:

B. Enable Amazon GuardDuty and check the findings for security groups in AWS Security Hub. Configure an Amazon EventBridge rule with a custom pattern that matches GuardDuty events with an output of NON_COMPLIANT. Define an Amazon Simple Notification Service (Amazon SNS) topic as the target.

GuardDuty is primarily designed for threat detection and continuous security monitoring, rather than configuration change tracking. While GuardDuty might eventually detect this, the delay is unacceptable compared to immediately triggering off the event. Security Hub is a central console for security findings, not a system for direct notification.

C. Create an AWS Config rule by using the restricted-ssh managed rule to check whether security groups disallow unrestricted incoming SSH traffic. Configure automatic remediation to publish a message to an Amazon Simple Notification Service (Amazon SNS) topic. AWS Config is valuable for compliance and configuration management. While it can detect deviations from desired configurations, the standard restricted-ssh rule is not triggered immediately upon a change; rather it runs periodic checks, which may delay the alert and not meet the requirements. Automatic remediation is overkill and may be undesirable; the requirement only specifies notification.

D. Enable Amazon Inspector. Include the Common Vulnerabilities and Exposures-1.1 rules package to check the security groups that are associated with the bastion hosts. Configure Amazon Inspector to publish a message to an Amazon Simple Notification Service (Amazon SNS) topic. Amazon Inspector focuses on vulnerability assessments of EC2 instances and container images, rather than configuration changes. It is less suitable for tracking security group modifications.

Supporting Links:

AWS CloudTrail:https://aws.amazon.com/cloudtrail/
Amazon EventBridge:https://aws.amazon.com/eventbridge/
Amazon SNS:https://aws.amazon.com/sns/
CloudTrail Events for Amazon EC2 Security Groups:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-control-access-to-instances.html#security-groups-commands

Question: 43

A DevOps team manages an API running on-premises that serves as a backend for an Amazon API Gateway endpoint. Customers have been complaining about high response latencies, which the development team has verified using the API Gateway latency metrics in Amazon CloudWatch. To identify the cause, the team needs to collect relevant data without introducing additional latency.
Which actions should be taken to accomplish this? (Choose two.)

A.Install the CloudWatch agent server side and configure the agent to upload relevant logs to CloudWatch.

B.Enable AWS X-Ray tracing in API Gateway, modify the application to capture request segments, and upload those segments to X-Ray during each request.

C.Enable AWS X-Ray tracing in API Gateway, modify the application to capture request segments, and use the X-Ray daemon to upload segments to X-Ray.

D.Modify the on-premises application to send log information back to API Gateway with each request.

E.Modify the on-premises application to calculate and upload statistical data relevant to the API service requests to CloudWatch metrics.

Answer: AC

Explanation:

The correct answer is AC. Here's why:

A. Install the CloudWatch agent server side and configure the agent to upload relevant logs to CloudWatch. Installing the CloudWatch agent on the on-premises server allows you to capture logs directly from the API server without adding latency to the API's response time. The agent can asynchronously upload these logs to CloudWatch. Analyzing these logs in CloudWatch will provide insights into server-side errors, slow queries, or other performance bottlenecks that contribute to the API latency. This is crucial because the API is on-premises, meaning direct access to the server logs is essential for troubleshooting.

C. Enable AWS X-Ray tracing in API Gateway, modify the application to capture request segments, and use the X-Ray daemon to upload segments to X-Ray. X-Ray allows you to trace requests from API Gateway through your backend application. Enabling X-Ray in API Gateway automatically instruments the API calls received by API Gateway. By modifying your application to generate and send request segments to X-Ray via the X-Ray daemon, you can gain visibility into how long the request spends in the on-premises application. The X-Ray daemon is optimized to efficiently handle the upload of trace data, minimizing the impact on the application's performance. Crucially, X-Ray provides latency distribution, enabling efficient analysis to trace the root cause of API's latency.

Why other options are not as suitable:

B. Enable AWS X-Ray tracing in API Gateway, modify the application to capture request segments, and upload those segments to X-Ray during each request. While X-Ray tracing is beneficial, sending segments synchronously with each request directly from the application will significantly increase response latency, which directly contradicts the problem statement's constraint. The X-Ray daemon is designed to handle the asynchronous upload of trace data to avoid this performance overhead.

D. Modify the on-premises application to send log information back to API Gateway with each request. Sending log data to API Gateway during each request will significantly increase the API latency. API Gateway is designed to route API requests and not designed for log aggregation.

E. Modify the on-premises application to calculate and upload statistical data relevant to the API service requests to CloudWatch metrics. While uploading metrics to CloudWatch is generally useful for monitoring, simply calculating and uploading aggregate statistics might not provide the detailed, granular insights required to pinpoint the cause of the latency. Moreover, calculating and uploading during each request cycle will add latency, which the problem statement wants to avoid. While helpful in the long run, it doesn't directly address the initial problem of identifying the cause without introducing latency.

Supporting Documentation:

AWS X-Ray:https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html
Amazon CloudWatch Agent:https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html
Amazon API Gateway CloudWatch Metrics:
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-metrics-and-dimensions.html

Question: 44

A company has an application that is using a MySQL-compatible Amazon Aurora Multi-AZ DB cluster as the database. A cross-Region read replica has been created for disaster recovery purposes. A DevOps engineer wants to automate the promotion of the replica so it becomes the primary database instance in the event of a failure. Which solution will accomplish this?

A.Configure a latency-based Amazon Route 53 CNAME with health checks so it points to both the primary and replica endpoints. Subscribe an Amazon SNS topic to Amazon RDS failure notifications from AWS CloudTrail and use that topic to invoke an AWS Lambda function that will promote the replica instance as the primary.

B.Create an Aurora custom endpoint to point to the primary database instance. Configure the application to use this endpoint. Configure AWS CloudTrail to run an AWS Lambda function to promote the replica instance and modify the custom endpoint to point to the newly promoted instance.

C.Create an AWS Lambda function to modify the application's AWS CloudFormation template to promote the replica, apply the template to update the stack, and point the application to the newly promoted instance.
Create an Amazon CloudWatch alarm to invoke this Lambda function after the failure event occurs.

D.Store the Aurora endpoint in AWS Systems Manager Parameter Store. Create an Amazon EventBridge event that detects the database failure and runs an AWS Lambda function to promote the replica instance and update the endpoint URL stored in AWS Systems Manager Parameter Store. Code the application to reload the endpoint from Parameter Store if a database connection fails.

Answer: D

Explanation:

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

Why Option D is Correct:

Option D provides a comprehensive and resilient solution for automating the promotion of an Aurora cross-Region read replica in the event of a primary database failure. It leverages several AWS services in a coordinated manner to achieve high availability and disaster recovery.

1. **AWS Systems Manager Parameter Store:** Storing the Aurora endpoint in Parameter Store provides a centralized and secure location to manage the database connection string. Applications can retrieve the endpoint dynamically, eliminating the need for hardcoded values and simplifying updates. 2. **Amazon EventBridge:** EventBridge allows for real-time event detection. By configuring an event rule to detect database failure events, the automation process is triggered immediately upon failure detection. This provides fast and reliable detection compared to polling mechanisms.
3. **AWS Lambda:** Lambda provides the serverless compute required to orchestrate the promotion process. The Lambda function would perform the following actions:

   Promote the cross-Region read replica to become the new primary instance.

   Update the Aurora endpoint URL in AWS Systems Manager Parameter Store with the endpoint of the promoted instance.
4. **Application Logic:** The application code is designed to handle database connection failures. Upon failure, the application would retrieve the updated endpoint from Parameter Store and attempt to reconnect, seamlessly switching to the newly promoted primary instance.

This solution is robust because it:

Automates the Failover Process: Reduces manual intervention and minimizes downtime.
Dynamically Updates the Connection String: Ensures applications always connect to the correct database instance.
Leverages Event-Driven Architecture: Enables immediate reaction to database failures.

Why Other Options Are Incorrect:

Option A: Route 53 latency-based routing is primarily designed for traffic distribution based on latency, not for automatic failover. While health checks can detect failure, the DNS propagation time might be significant, leading to prolonged downtime. CloudTrail, while providing auditing, is not the ideal trigger for immediate failover actions. Relying solely on SNS notifications from CloudTrail might introduce delays and unnecessary complexity.

Option B: Custom endpoints are generally useful for directing specific types of traffic (e.g., read-only traffic), but they are not inherently designed for failover scenarios. Furthermore, modifying CloudFormation to promote the replica and update custom endpoint directly introduces potential risks and complexity.

Option C: Modifying CloudFormation templates during a failover event introduces significant risks and potential delays. CloudFormation updates can be time-consuming and prone to errors. Relying on CloudWatch alarm and Lambda function to modify CloudFormation templates would be slow and complex for a database failover scenario.

Supporting Links:

AWS Systems Manager Parameter Store:https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html
Amazon EventBridge:https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html
AWS Lambda:https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
Amazon Aurora Global Database:https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-global-database.html (Relevant for cross-region replication)

Question: 45

A company hosts its staging website using an Amazon EC2 instance backed with Amazon EBS storage. The company wants to recover quickly with minimal data losses in the event of network connectivity issues or power failures on the EC2 instance.
Which solution will meet these requirements?

A.Add the instance to an EC2 Auto Scaling group with the minimum, maximum, and desired capacity set to 1.

B.Add the instance to an EC2 Auto Scaling group with a lifecycle hook to detach the EBS volume when the EC2 instance shuts down or terminates.

C.Create an Amazon CloudWatch alarm for the StatusCheckFailed System metric and select the EC2 action to recover the instance.

D.Create an Amazon CloudWatch alarm for the StatusCheckFailed Instance metric and select the EC2 action to reboot the instance.

Answer: C

Explanation:

The correct answer is **C: Create an Amazon CloudWatch alarm for the StatusCheckFailed System metric and select the EC2 action to recover the instance.**

Here's a detailed justification:

The key requirement is to recover quickly with minimal data losses from network or power failures on the EC2 instance.

System Status Checks monitor the underlying EC2 host. Failures indicate problems with the hardware hosting your instance (network connectivity, power outage, hardware issues). The StatusCheckFailed_System metric specifically captures these system-level issues.

EC2 Recover action: When a StatusCheckFailed_System alarm triggers, the "Recover" action attempts to move the affected instance to new healthy hardware within the same Availability Zone. This is the fastest way to restore service because the underlying EBS volume remains attached, preserving data. It avoids the potentially longer time required to launch a new instance and restore from a snapshot.

Minimal Data Loss: Because the EBS volume isn't detached during the recovery process, there is minimal data loss. Only the data in memory and in-transit that hasn't yet been written to disk would be lost.

Why other options are incorrect:

A: Placing the instance in an Auto Scaling group (ASG) with a capacity of 1 would replace the instance if it becomes unhealthy, but this is slower than recovery. The ASG needs to detect the failure, launch a new instance, and potentially configure it. Furthermore, restoring the data to the new instance requires a full boot process and copying data (from a snapshot or similar), leading to more downtime.

B: Adding a lifecycle hook to detach the EBS volume before shutdown is counterproductive. Detaching the volume and then reattaching it adds considerable time and complexity to the recovery process. It doesn't minimize downtime.

D: Rebooting an instance (using StatusCheckFailed_Instance alarm) only addresses problems within the guest OS, not underlying hardware issues. It will not solve network connectivity issues or power failures affecting the host. StatusCheckFailed_Instance measures the health of the instance itself, not the underlying system.

In summary, StatusCheckFailed_System + EC2 Recovery offers the fastest path to recovery from underlying hardware problems (network, power) while keeping the EBS volume attached, minimizing data loss.

Authoritative Links:

EC2 Instance Recovery:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-recover.html
CloudWatch Metrics for EC2:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-cloudwatch-metrics.html
EC2 Status Checks:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-system-instance-status-check.html

Question: 46

A company wants to use AWS development tools to replace its current bash deployment scripts. The company currently deploys a LAMP application to a group of Amazon EC2 instances behind an Application Load Balancer (ALB). During the deployments, the company unit tests the committed application, stops and starts services, unregisters and re-registers instances with the load balancer, and updates file permissions. The company wants to maintain the same deployment functionality through the shift to using AWS services.
Which solution will meet these requirements?

A.Use AWS CodeBuild to test the application. Use bash scripts invoked by AWS CodeDeploy's appspec.yml file to restart services, and deregister and register instances with the ALB. Use the appspec.yml file to update file permissions without a custom script.

B.Use AWS CodePipeline to move the application from the AWS CodeCommit repository to AWS CodeDeploy. Use CodeDeploy's deployment group to test the application, unregister and re-register instances with the ALand restart services. Use the appspec.yml file to update file permissions without a custom script.

C.Use AWS CodePipeline to move the application source code from the AWS CodeCommit repository to AWS CodeDeploy. Use CodeDeploy to test the application. Use CodeDeploy's appspec.yml file to restart services and update permissions without a custom script. Use AWS CodeBuild to unregister and re-register instances with the ALB.

D.Use AWS CodePipeline to trigger AWS CodeBuild to test the application. Use bash scripts invoked by AWS CodeDeploy's appspec.yml file to restart services. Unregister and re-register the instances in the AWS CodeDeploy deployment group with the ALB. Update the appspec.yml file to update file permissions without a custom script.

Answer: D

Explanation:

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

The scenario requires a solution that integrates testing, deployment orchestration, service management, and load balancer interaction, all while transitioning from bash scripts to AWS development tools.

CodePipeline for Orchestration: AWS CodePipeline is the core service for continuous integration and continuous delivery (CI/CD). It automates the build, test, and deploy phases of the software release process.
So, it will be the most appropriate for this process. https://aws.amazon.com/codepipeline/

CodeBuild for Testing: AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. Using CodeBuild for unit testing aligns with the company's requirements for testing the committed application. https://aws.amazon.com/codebuild/

CodeDeploy for Deployment & Service Management: AWS CodeDeploy automates code deployments to various compute services such as EC2 instances. The appspec.yml file defines deployment actions and is crucial for automating the tasks that the company previously performed through bash scripts.
https://aws.amazon.com/codedeploy/

appspec.yml for Tasks: The appspec.yml file within CodeDeploy deployments allows for the execution of custom scripts (bash in this case) at various lifecycle events, such as ApplicationStop, BeforeInstall, AfterInstall, ApplicationStart, and ValidateService. This enables restarting services using bash scripts invoked by appspec.yml. Additionally, it allows to update file permissions.

ALB Integration: CodeDeploy natively integrates with Application Load Balancers. During deployment lifecycle events, instances within the CodeDeploy deployment group can be automatically deregistered from the ALB before deployment and re-registered after the deployment is successful. This ensures minimal downtime and a seamless deployment process.

Option A is incorrect because it doesn't use CodePipeline, and it is more robust approach to rely on CodePipeline for a CI/CD pipeline. Option B and C are incorrect as they suggest CodeDeploy itself can perform testing, which is not one of its core functions; CodeBuild is better suited. Option C also incorrectly suggests AWS CodeBuild can directly unregister/re-register instances with the ALB; this is a function handled by CodeDeploy.

In summary, option D provides the most comprehensive solution that addresses all the company's requirements by utilizing the strengths of CodePipeline for orchestration, CodeBuild for testing, and CodeDeploy for deployment with the help of appspec.yml for service management and ALB integration.

Question: 47

A company runs an application with an Amazon EC2 and on-premises configuration. A DevOps engineer needs to standardize patching across both environments. Company policy dictates that patching only happens during non-business hours.
Which combination of actions will meet these requirements? (Choose three.)

A.Add the physical machines into AWS Systems Manager using Systems Manager Hybrid Activations.
B.Attach an IAM role to the EC2 instances, allowing them to be managed by AWS Systems Manager.
C.Create IAM access keys for the on-premises machines to interact with AWS Systems Manager.
D.Run an AWS Systems Manager Automation document to patch the systems every hour
E.Use Amazon EventBridge scheduled events to schedule a patch window.
F.Use AWS Systems Manager Maintenance Windows to schedule a patch window.

Answer: ABF

Explanation:

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

A. Add the physical machines into AWS Systems Manager using Systems Manager Hybrid Activations:

AWS Systems Manager (SSM) is a management service that allows you to manage your EC2 instances and on-premises servers and virtual machines. To manage on-premises servers with SSM, you need to onboard them as managed instances. Systems Manager Hybrid Activations enable you to register your on-premises servers and VMs with SSM, making them manageable through the same interface as your EC2 instances. This standardization is crucial for centralized patching. https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-managed-instances.html

B. Attach an IAM role to the EC2 instances, allowing them to be managed by AWS Systems Manager: EC2 instances need permissions to communicate with SSM. This is achieved by attaching an IAM role to the EC2 instances. This IAM role grants the necessary permissions for SSM Agent, running on the instances, to perform actions on behalf of SSM, such as receiving commands, reporting status, and downloading patches.

Without this role, the EC2 instances cannot be managed by SSM. https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-iam-roles.html

F. Use AWS Systems Manager Maintenance Windows to schedule a patch window: Maintenance Windows in SSM provides a defined period to perform potentially disruptive actions, such as patching, on your instances. It allows you to schedule these operations to occur during non-business hours, fulfilling the company's policy requirement. You can target specific instances or instance groups for patching during the maintenance window. This is a critical element to automate patching within the defined timeframe.

https://docs.aws.amazon.com/systems-manager/latest/userguide/maintenance-windows.html

Why other options are incorrect:

C. Create IAM access keys for the on-premises machines to interact with AWS Systems Manager: While IAM access keys can be used for programmatic access to AWS services, they are generally discouraged for use on instances (EC2 or on-premise) due to security concerns (key rotation, accidental exposure). Roles are the preferred method for granting permissions to instances. Hybrid Activations is the method by which on-premise machines get added.

D. Run an AWS Systems Manager Automation document to patch the systems every hour: Running a patch automation every hour would likely conflict with the requirement to only patch during non-business hours. Maintenance Windows are designed for scheduled operations.

E. Use Amazon EventBridge scheduled events to schedule a patch window: While EventBridge could trigger an SSM Automation document, Maintenance Windows provide a more direct and purpose-built feature for scheduling patching operations, including built-in retry mechanisms and concurrency control. Maintenance Windows are more appropriate for this use case.

Question: 48

A company has chosen AWS to host a new application. The company needs to implement a multi-account strategy. A DevOps engineer creates a new AWS account and an organization in AWS Organizations. The DevOps engineer also creates the OU structure for the organization and sets up a landing zone by using AWS Control Tower. The DevOps engineer must implement a solution that automatically deploys resources for new accounts that users create through AWS Control Tower Account Factory. When a user creates a new account, the solution must apply AWS CloudFormation templates and SCPs that are customized for the OU or the account to automatically deploy all the resources that are attached to the account. All the OUs are enrolled in AWS Control Tower.
Which solution will meet these requirements in the MOST automated way?

A.Use AWS Service Catalog with AWS Control Tower. Create portfolios and products in AWS Service Catalog. Grant granular permissions to provision these resources. Deploy SCPs by using the AWS CLI and JSON documents.

B.Deploy CloudFormation stack sets by using the required templates. Enable automatic deployment. Deploy stack instances to the required accounts. Deploy a CloudFormation stack set to the organization's management account to deploy SCPs.

C.Create an Amazon EventBridge rule to detect the CreateManagedAccount event. Configure AWS Service Catalog as the target to deploy resources to any new accounts. Deploy SCPs by using the AWS CLI and JSON documents.

D.Deploy the Customizations for AWS Control Tower (CfCT) solution. Use an AWS CodeCommit repository as the source. In the repository, create a custom package that includes the CloudFormation templates and the SCP JSON documents.

Answer: D

Explanation:

The correct answer is D because the Customizations for AWS Control Tower (CfCT) solution is specifically designed to automate the deployment of resources and SCPs to new accounts created through AWS Control Tower Account Factory. It uses a CodeCommit repository as a central source for customizations. This allows you to manage CloudFormation templates and SCPs as code, version them, and apply them consistently across accounts.

Option A is less ideal because while AWS Service Catalog can provision resources, it doesn't directly integrate with AWS Control Tower's account creation process for automatic deployment in the same way as CfCT. Moreover, deploying SCPs via the CLI is less automated.

Option B's CloudFormation StackSets, though powerful, require more manual configuration to target newly created accounts dynamically. While StackSets can be configured for automatic deployment, integrating them with the Control Tower Account Factory requires extra scripting or event handling, which increases complexity. Deploying SCPs via another StackSet to the management account might not achieve the desired granular control and automation at the OU/account level.

Option C, while utilizing EventBridge, also relies on AWS Service Catalog. Though event-driven, the Service Catalog approach lacks the native integration and streamlined deployment mechanisms of CfCT. Also, SCP deployment is performed manually.

CfCT's main advantage lies in its ability to automate the entire process from account creation to resource and SCP deployment using a centralized configuration repository, providing the most automated solution as requested by the problem statement. It's purpose-built for extending Control Tower's governance capabilities.

Supporting References:

AWS Control Tower Customizations for AWS Control Tower (CfCT): https://aws.amazon.com/solutions/implementations/customizations-for-aws-control-tower/ AWS Organizations SCPs: https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scps.html AWS CloudFormation: https://aws.amazon.com/cloudformation/

Question: 49

An online retail company based in the United States plans to expand its operations to Europe and Asia in the next six months. Its product currently runs on Amazon EC2 instances behind an Application Load Balancer. The instances run in an Amazon EC2 Auto Scaling group across multiple Availability Zones. All data is stored in an Amazon Aurora database instance.
When the product is deployed in multiple regions, the company wants a single product catalog across all regions, but for compliance purposes, its customer information and purchases must be kept in each region.
How should the company meet these requirements with the LEAST amount of application changes?

A.Use Amazon Redshift for the product catalog and Amazon DynamoDB tables for the customer information and purchases.

B.Use Amazon DynamoDB global tables for the product catalog and regional tables for the customer information and purchases.

C.Use Aurora with read replicas for the product catalog and additional local Aurora instances in each region for the customer information and purchases.

D.Use Aurora for the product catalog and Amazon DynamoDB global tables for the customer information and purchases.

Answer: C

Explanation:

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

Option C, using Aurora with read replicas for the product catalog and additional local Aurora instances in each region for customer information and purchases, offers the most efficient and compliant solution with the least application changes.

The core requirement is a single, globally available product catalog while maintaining regional separation of customer data for compliance. Aurora is a highly scalable and reliable relational database suitable for a product catalog. Leveraging Aurora Read Replicas allows read-only copies of the catalog to be distributed to the new regions (Europe and Asia). This provides low-latency access to product information for customers in each region without requiring complex data synchronization mechanisms. Because read replicas are asynchronous, performance impact on the primary Aurora instance in the US is minimized.

For customer information and purchases, which need to be region-specific due to compliance, creating separate Aurora instances within each region ensures data isolation. This satisfies the regulatory requirement of keeping customer data within the region where it's collected. Using Aurora consistently across both the product catalog and customer data simplifies database administration and leverages the company's existing expertise with Aurora.

Compared to other options:

Option A (Redshift & DynamoDB tables): Introducing Redshift adds unnecessary complexity. Redshift is primarily designed for analytics, not transactional data like a product catalog. The catalog doesn't require the analytical capabilities of Redshift.

Option B (DynamoDB Global tables for product catalog): Migrating the entire product catalog to DynamoDB would involve significant application changes. The existing system is designed for Aurora, and a complete migration to NoSQL for the product catalog requires considerable effort.

Option D (Aurora and DynamoDB global tables for customer data): While DynamoDB global tables are suitable for multi-region replication, using them for customer data still necessitates application modifications to integrate with a new database technology for sensitive information, which is undesirable.

Therefore, option C minimizes application changes by leveraging existing Aurora infrastructure and introducing only the relatively straightforward addition of read replicas. The region-specific Aurora instances for customer data cleanly segregate sensitive information in compliance with regulations.

Supporting concepts:

Read Replicas: Provide read-only copies of a database, improving read performance and availability without impacting the primary database.

Database Replication: Copying data across multiple servers or locations for redundancy, availability, and disaster recovery.

Data Sovereignty/Regional Compliance: Regulations requiring data to be stored and processed within specific geographic regions.

Authoritative links:

Amazon Aurora Read Replicas:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.ReadReplicas.html **AWS Global Infrastructure:**https://aws.amazon.com/about-aws/global-infrastructure/

Question: 50

A company is implementing a well-architected design for its globally accessible API stack. The design needs to ensure both high reliability and fast response times for users located in North America and Europe.
The API stack contains the following three tiers:

Amazon API Gateway -

AWS Lambda -

Amazon DynamoDB -
Which solution will meet the requirements?

A.Configure Amazon Route 53 to point to API Gateway APIs in North America and Europe using health checks. Configure the APIs to forward requests to a Lambda function in that Region. Configure the Lambda functions to retrieve and update the data in a DynamoDB table in the same Region as the Lambda function.

B.Configure Amazon Route 53 to point to API Gateway APIs in North America and Europe using latency-based routing and health checks. Configure the APIs to forward requests to a Lambda function in that Region. Configure the Lambda functions to retrieve and update the data in a DynamoDB global table.

C.Configure Amazon Route 53 to point to API Gateway in North America, create a disaster recovery API in Europe, and configure both APIs to forward requests to the Lambda functions in that Region. Retrieve the data from a DynamoDB global table. Deploy a Lambda function to check the North America API health every 5 minutes. In the event of a failure, update Route 53 to point to the disaster recovery API.

D.Configure Amazon Route 53 to point to API Gateway API in North America using latency-based routing. Configure the API to forward requests to the Lambda function in the Region nearest to the user. Configure the Lambda function to retrieve and update the data in a DynamoDB table.

Answer: B

Explanation:

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

High Availability & Low Latency: The requirement is to ensure high reliability and fast response times for users in North America and Europe. The key here is providing services geographically closer to the users, minimizing latency.

Route 53 Latency-Based Routing: Option B utilizes Route 53 latency-based routing, which directs users to the API Gateway endpoint that provides the lowest latency. This ensures users are routed to the closest API endpoint. Health checks ensure that only healthy endpoints are used.
[https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html#routing-policy-latency]

Regional API Gateways & Lambda Functions: The API Gateway and Lambda functions are deployed in both North America and Europe. This means requests are processed closer to the user's location, further minimizing latency. Keeping Lambda functions and DynamoDB tables in the same Region optimizes network communication.

DynamoDB Global Tables: DynamoDB global tables provide automatic, multi-Region replication. This means that data written in one Region is automatically replicated to other Regions. Using global tables ensures that data is available in both North America and Europe, regardless of where it was initially written.
[https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/globaltables.V2.html] This approach also allows for read and write operations to be handled in the user's local region whenever possible.

Why other options are not optimal:

Option A: While it uses Route 53 with health checks, it does not use latency-based routing. This might not direct users to the closest endpoint, potentially leading to higher latency. Configuring DynamoDB tables in only one Region would introduce cross-region read/write operations, increasing latency.
Option C: This introduces a disaster recovery (DR) setup. While DR is important, this design focuses on

immediate low-latency performance. The failover approach in this option is slower to respond than latency-based routing. Furthermore, relying on a single API and a DR API increases complexity.
Option D: This uses latency-based routing for Route53 but has a single API Gateway, which is not sufficient for redundancy. Also, accessing a single DynamoDB table from all regions would lead to latency issues.

Question: 51

A rapidly growing company wants to scale for developer demand for AWS development environments.
Development environments are created manually in the AWS Management Console. The networking team uses AWS CloudFormation to manage the networking infrastructure, exporting stack output values for the Amazon VPC and all subnets.
The development environments have common standards, such as Application Load Balancers, Amazon EC2 Auto Scaling groups, security groups, and Amazon DynamoDB tables.
To keep up with demand, the DevOps engineer wants to automate the creation of development environments. Because the infrastructure required to support the application is expected to grow, there must be a way to easily update the deployed infrastructure. CloudFormation will be used to create a template for the development environments.
Which approach will meet these requirements and quickly provide consistent AWS environments for developers?

A.Use Fn::ImportValue intrinsic functions in the Resources section of the template to retrieve Virtual Private Cloud (VPC) and subnet values. Use CloudFormation StackSets for the development environments, using the Count input parameter to indicate the number of environments needed. Use the UpdateStackSet command to update existing development environments.

B.Use nested stacks to define common infrastructure components. To access the exported values, use TemplateURL to reference the networking team's template. To retrieve Virtual Private Cloud (VPC) and subnet values, use Fn::ImportValue intrinsic functions in the Parameters section of the root template. Use the CreateChangeSet and ExecuteChangeSet commands to update existing development environments.

C.Use nested stacks to define common infrastructure components. Use Fn::ImportValue intrinsic functions with the resources of the nested stack to retrieve Virtual Private Cloud (VPC) and subnet values. Use the CreateChangeSet and ExecuteChangeSet commands to update existing development environments.

D.Use Fn::ImportValue intrinsic functions in the Parameters section of the root template to retrieve Virtual Private Cloud (VPC) and subnet values. Define the development resources in the order they need to be created in the CloudFormation nested stacks. Use the CreateChangeSet. and ExecuteChangeSet commands to update existing development environments.

Answer: C

Explanation:

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

The scenario requires automating and standardizing the creation of AWS development environments using CloudFormation, while also allowing for easy updates to the deployed infrastructure.

Why Option C is Correct:

Nested Stacks: Nested stacks are a CloudFormation feature that allows you to break down complex infrastructure into smaller, more manageable, and reusable components. This is essential for managing common infrastructure components, as specified in the requirement.
Fn::ImportValue: This intrinsic function enables cross-stack referencing. It allows retrieving exported values from other CloudFormation stacks (in this case, the networking team's stack, which exports VPC and subnet information). Using it within the resources of the nested stacks ensures that the shared networking information is correctly utilized by each environment's infrastructure resources.
Change Sets: Change sets provide a preview of the changes that CloudFormation will make to your infrastructure before actually applying them. This significantly reduces the risk of unexpected changes and outages. Using CreateChangeSet and ExecuteChangeSet ensures controlled and safe updates to the environments.

Why Other Options are Incorrect:

Option A (StackSets with Count): While StackSets are useful for deploying across multiple accounts or regions, they are not the most appropriate for creating independent development environments within a single account. The Count parameter isn't suitable here because the goal isn't to deploy identical stacks repeatedly, but to create distinct environments.
Option B: Using TemplateURL to reference the entire networking team's template is not correct. Rather, the required approach is to import individual VPC and subnet IDs. Also, Fn::ImportValue is not appropriate in the Parameters section of the root template in this context, it needs to be used within resources of nested stacks, which require the VPC and subnet IDs to function.
Option D: Placing Fn::ImportValue in the Parameters section is less appropriate than using it directly within the resources of the nested stacks that need the VPC and subnet IDs. Furthermore, explicitly defining the creation order of resources becomes less manageable as the infrastructure grows. CloudFormation generally handles dependencies, and unnecessary ordering can complicate the template.

Supporting Links:

CloudFormation Nested Stacks:https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-nested-stacks.html
Fn::ImportValue:https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-importvalue.html
CloudFormation Change Sets:https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html

In summary, option C provides the most scalable, maintainable, and controlled approach for automating the creation and updating of development environments using CloudFormation.

Question: 52

A company uses AWS Organizations to manage multiple accounts. Information security policies require that all unencrypted Amazon EBS volumes be marked as non-compliant. A DevOps engineer needs to automatically deploy the solution and ensure that this compliance check is always present.
Which solution will accomplish this?

A.Create an AWS CloudFormation template that defines an AWS Inspector rule to check whether EBS encryption is enabled. Save the template to an Amazon S3 bucket that has been shared with all accounts within the company. Update the account creation script pointing to the CloudFormation template in Amazon S3.

B.Create an AWS Config organizational rule to check whether EBS encryption is enabled and deploy the rule using the AWS CLI. Create and apply an SCP to prohibit stopping and deleting AWS Config across the organization.

C.Create an SCP in Organizations. Set the policy to prevent the launch of Amazon EC2 instances without encryption on the EBS volumes using a conditional expression. Apply the SCP to all AWS accounts. Use Amazon Athena to analyze the AWS CloudTrail output, looking for events that deny an ec2:RunInstances action.

D.Deploy an IAM role to all accounts from a single trusted account. Build a pipeline with AWS CodePipeline with a stage in AWS Lambda to assume the IAM role, and list all EBS volumes in the account. Publish a report to Amazon S3.

Answer: B

Explanation:

The correct answer is B. Here's why:

AWS Config organizational rules provide a centralized and automated way to evaluate the configuration of AWS resources across an entire organization, checking for compliance with desired policies. In this case, the Config rule can be specifically configured to evaluate whether EBS encryption is enabled. This addresses the requirement of automatically checking for unencrypted EBS volumes across all accounts.

Using the AWS CLI to deploy the organizational rule ensures a programmatic and repeatable deployment process, fitting the DevOps engineer's need for automation.

Implementing a Service Control Policy (SCP) to prohibit stopping or deleting AWS Config ensures that the compliance checks are consistently enforced and cannot be bypassed by individual accounts. This guarantees the persistence of the compliance assessment. SCPs operate at the organization level and govern the permissions available to member accounts.

Option A is less suitable because AWS Inspector is more focused on security assessments during runtime and might not provide continuous compliance monitoring like AWS Config. Sharing CloudFormation templates requires management across multiple accounts.

Option C, using SCPs to prevent EC2 instance launches without encryption, prevents non-compliant resources, but it doesn't monitor existing unencrypted volumes. Also, Athena and CloudTrail for detecting denied actions are reactive rather than proactive.

Option D requires a complex pipeline involving IAM roles and Lambda functions which adds complexity for the given requirement. Further, it lists and reports rather than automatically enforcing compliance.

Justification with Cloud Computing Concepts:

AWS Organizations: Enables centralized management and governance over multiple AWS accounts.

AWS Config: Provides continuous compliance monitoring by evaluating resource configurations against desired rules.

Organizational Rules: Extend Config rules to the organization level, ensuring consistent compliance checks across all accounts.

Service Control Policies (SCPs): Allow centralized control over the maximum available permissions in member accounts, crucial for preventing actions that could undermine compliance efforts.

Automation: The use of the AWS CLI ensures that the rule deployment is automated and can be easily repeated.

Authoritative Links:

AWS Config Organizational Rules: https://docs.aws.amazon.com/config/latest/developerguide/config-rule-managing-rules.html
AWS Organizations Service Control Policies (SCPs):
https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scp.html

Question: 53

A company is performing vulnerability scanning for all Amazon EC2 instances across many accounts. The accounts are in an organization in AWS Organizations. Each account's VPCs are attached to a shared transit gateway. The VPCs send traffic to the internet through a central egress VPC. The company has enabled Amazon Inspector in a delegated administrator account and has enabled scanning for all member accounts.
A DevOps engineer discovers that some EC2 instances are listed in the "not scanning" tab in Amazon Inspector.
Which combination of actions should the DevOps engineer take to resolve this issue? (Choose three.)

A.Verify that AWS Systems Manager Agent is installed and is running on the EC2 instances that Amazon Inspector is not scanning.

B.Associate the target EC2 instances with security groups that allow outbound communication on port 443 to the AWS Systems Manager service endpoint.

C.Grant inspector:StartAssessmentRun permissions to the IAM role that the DevOps engineer is using.

D.Configure EC2 Instance Connect for the EC2 instances that Amazon Inspector is not scanning.

E.Associate the target EC2 instances with instance profiles that grant permissions to communicate with AWS

Systems Manager.

F.Create a managed-instance activation. Use the Activation Code and the Activation ID to register the EC2 instances.

Answer: ABE

Explanation:

The correct answer is ABE. Here's why:

A. Verify that AWS Systems Manager Agent is installed and is running on the EC2 instances that Amazon Inspector is not scanning. Amazon Inspector relies on the AWS Systems Manager (SSM) Agent to be installed and running on EC2 instances to perform vulnerability assessments. If the agent isn't present or is not running, Inspector cannot scan the instance, and it will appear in the "not scanning" tab. This is a fundamental requirement for Inspector to function.
https://docs.aws.amazon.com/inspector/latest/userguide/inspector_systems-manager-prerequisites.html

B. Associate the target EC2 instances with security groups that allow outbound communication on port 443 to the AWS Systems Manager service endpoint. The SSM Agent needs to communicate with the AWS SSM service endpoint to receive instructions and send back assessment data. Port 443 (HTTPS) is the standard port for secure communication. If the security groups associated with the EC2 instances block outbound traffic on port 443 to the SSM endpoint, the agent won't be able to communicate, and the instances won't be scanned.
https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-security-best-practices.html

E. Associate the target EC2 instances with instance profiles that grant permissions to communicate with AWS Systems Manager. The EC2 instances need appropriate IAM permissions to allow the SSM Agent to communicate with AWS Systems Manager. This is typically done by attaching an IAM role to the EC2 instance (instance profile). This role should include the AmazonSSMManagedInstanceCore policy or equivalent permissions allowing SSM access. Without these permissions, the agent cannot properly communicate with the SSM service, preventing Inspector from scanning the instances. https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-security-iam-id-based-policy-examples.html

Why the other options are incorrect:

C. Grant inspector:StartAssessmentRun permissions to the IAM role that the DevOps engineer is using. While inspector:StartAssessmentRun is needed to initiate an assessment run, it does not address the reason why the EC2 instances are listed as "not scanning". The instances must first be properly configured to be scanned, which requires SSM connectivity.

D. Configure EC2 Instance Connect for the EC2 instances that Amazon Inspector is not scanning. EC2 Instance Connect allows connecting to instances using SSH, but it doesn't play a role in enabling Amazon Inspector to scan the instances. Instance Connect is a convenience for connecting to instances and does not relate to SSM or Inspector functionality.

F. Create a managed-instance activation. Use the Activation Code and the Activation ID to register the EC2 instances. Managed-instance activation is primarily used for registering on-premises servers or VMs with AWS Systems Manager. EC2 instances, which are already part of AWS, don't typically require this activation process, especially in the context described with delegated administrator. Using an activation code and ID is a more complex setup that isn't necessary if the issue is just missing SSM connectivity.
https://docs.aws.amazon.com/systems-manager/latest/userguide/managed-instances-onpremise.html

Question: 54

A development team uses AWS CodeCommit for version control for applications. The development team uses AWS CodePipeline, AWS CodeBuild. and AWS CodeDeploy for CI/CD infrastructure. In CodeCommit, the development team recently merged pull requests that did not pass long-running tests in the code base. The development team needed to perform rollbacks to branches in the codebase, resulting in lost time and wasted effort.
A DevOps engineer must automate testing of pull requests in CodeCommit to ensure that reviewers more easily see the results of automated tests as part of the pull request review.
What should the DevOps engineer do to meet this requirement?

A.Create an Amazon EventBridge rule that reacts to the pullRequestStatusChanged event. Create an AWS Lambda function that invokes a CodePipeline pipeline with a CodeBuild action that runs the tests for the application. Program the Lambda function to post the CodeBuild badge as a comment on the pull request so that developers will see the badge in their code review.

B.Create an Amazon EventBridge rule that reacts to the pullRequestCreated event. Create an AWS Lambda function that invokes a CodePipeline pipeline with a CodeBuild action that runs the tests for the application. Program the Lambda function to post the CodeBuild test results as a comment on the pull request when the test results are complete.

C.Create an Amazon EventBridge rule that reacts to pullRequestCreated and pullRequestSourceBranchUpdated events. Create an AWS Lambda function that invokes a CodePipeline pipeline with a CodeBuild action that runs the tests for the application. Program the Lambda function to post the CodeBuild badge as a comment on the pull request so that developers will see the badge in their code review.

D.Create an Amazon EventBridge rule that reacts to the pullRequestStatusChanged event. Create an AWS Lambda function that invokes a CodePipeline pipeline with a CodeBuild action that runs the tests for the application. Program the Lambda function to post the CodeBuild test results as a comment on the pull request when the test results are complete.

Answer: C

Explanation:

The correct answer is C. Here's why:

The requirement is to automate testing of pull requests in CodeCommit so reviewers can easily see the test results. This means triggering tests whenever a new pull request is created or when the source branch of an existing pull request is updated.

EventBridge Rule: Option C correctly identifies the need to trigger the testing process on two events: pullRequestCreated and pullRequestSourceBranchUpdated. A new pull request needs to be tested, and any changes pushed to the source branch of an existing pull request should also trigger a new test run to ensure continued validity. Option A and D only trigger on pullRequestStatusChanged, which is not appropriate for initial pull request validation or updates to existing requests. B only addresses creation, ignoring updates.
Lambda Function: The Lambda function acts as the orchestrator. It's triggered by the EventBridge rule and then starts a CodePipeline.
CodePipeline and CodeBuild: The CodePipeline contains a CodeBuild action. CodeBuild is ideal for running the actual tests. It compiles the code, executes the tests, and produces the results.
Posting Badge: The Lambda function then posts the CodeBuild badge as a comment on the pull request. A badge visually indicates the status of the build (success or failure) directly within the pull request, making it very easy for reviewers to quickly assess the test results. While Option D posts the test results as a comment, a badge offers a more immediate visual indicator.

Option C handles both the initial pull request creation and subsequent updates, triggers testing, and provides a clear, visual indication of the test results within the pull request, fulfilling all requirements.

Authoritative Links:

Amazon EventBridge:https://aws.amazon.com/eventbridge/
AWS Lambda:https://aws.amazon.com/lambda/
AWS CodePipeline:https://aws.amazon.com/codepipeline/
AWS CodeBuild:https://aws.amazon.com/codebuild/

AWS CodeCommit:https://aws.amazon.com/codecommit/

Question: 55

A company has deployed an application in a production VPC in a single AWS account. The application is popular and is experiencing heavy usage. The company's security team wants to add additional security, such as AWS WAF, to the application deployment. However, the application's product manager is concerned about cost and does not want to approve the change unless the security team can prove that additional security is necessary.
The security team believes that some of the application's demand might come from users that have IP addresses that are on a deny list. The security team provides the deny list to a DevOps engineer. If any of the IP addresses on the deny list access the application, the security team wants to receive automated notification in near real time so that the security team can document that the application needs additional security. The DevOps engineer creates a VPC flow log for the production VPC.
Which set of additional steps should the DevOps engineer take to meet these requirements MOST cost-effectively?

A.Create a log group in Amazon CloudWatch Logs. Configure the VPC flow log to capture accepted traffic and to send the data to the log group. Create an Amazon CloudWatch metric filter for IP addresses on the deny list. Create a CloudWatch alarm with the metric filter as input. Set the period to 5 minutes and the datapoints to alarm to 1. Use an Amazon Simple Notification Service (Amazon SNS) topic to send alarm notices to the security team.

B.Create an Amazon S3 bucket for log files. Configure the VPC flow log to capture all traffic and to send the data to the S3 bucket. Configure Amazon Athena to return all log files in the S3 bucket for IP addresses on the deny list. Configure Amazon QuickSight to accept data from Athena and to publish the data as a dashboard that the security team can access. Create a threshold alert of 1 for successful access. Configure the alert to automatically notify the security team as frequently as possible when the alert threshold is met.

C.Create an Amazon S3 bucket for log files. Configure the VPC flow log to capture accepted traffic and to send the data to the S3 bucket. Configure an Amazon OpenSearch Service cluster and domain for the log files. Create an AWS Lambda function to retrieve the logs from the S3 bucket, format the logs, and load the logs into the OpenSearch Service cluster. Schedule the Lambda function to run every 5 minutes. Configure an alert and condition in OpenSearch Service to send alerts to the security team through an Amazon Simple Notification Service (Amazon SNS) topic when access from the IP addresses on the deny list is detected.

D.Create a log group in Amazon CloudWatch Logs. Create an Amazon S3 bucket to hold query results. Configure the VPC flow log to capture all traffic and to send the data to the log group. Deploy an Amazon Athena CloudWatch connector in AWS Lambda. Connect the connector to the log group. Configure Athena to periodically query for all accepted traffic from the IP addresses on the deny list and to store the results in the S3 bucket. Configure an S3 event notification to automatically notify the security team through an Amazon Simple Notification Service (Amazon SNS) topic when new objects are added to the S3 bucket.

Answer: A

Explanation:

The most cost-effective solution is option A, which leverages CloudWatch Logs, Metric Filters, Alarms, and SNS. This approach avoids the overhead of managing storage, query engines, and complex data pipelines found in the other options.

Here's why option A is superior:

Cost Efficiency: CloudWatch Logs and Metric Filters are designed for log analysis and metric extraction. They are more cost-effective for this specific use case compared to S3, Athena, OpenSearch, or QuickSight, which are geared toward larger-scale data analytics and visualization.
Near Real-Time Notification: CloudWatch Alarms can be configured with a 5-minute period, providing timely notification when traffic from the deny list is detected. This satisfies the "near real-time" requirement.
Specific Targeting: Filtering the VPC flow logs for accepted traffic reduces noise and focuses on actual attempts to access the application from the denied IPs.
Automation: The entire process, from log capture to notification, is automated, minimizing manual intervention.
Simplicity: The solution avoids complex data transformation and loading processes required by S3, Lambda,

and OpenSearch.

Here's why the other options are less suitable:

Option B (S3, Athena, QuickSight): This is unnecessarily complex and costly for simple IP address monitoring. Athena queries can be slow, and QuickSight adds visualization overhead that's not needed for a simple notification system.

Option C (S3, OpenSearch, Lambda): OpenSearch Service is a powerful search and analytics engine, but it is overkill for this simple use case. The Lambda function adds operational complexity and cost.

Option D (CloudWatch Logs, S3, Athena, Lambda): Similar to option B, this option introduces unnecessary complexity with Athena and S3. The Athena connector through Lambda is also not the most efficient solution.

CloudWatch documentation:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.htmlVPC Flow Logs:
https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html

Question: 56

A DevOps engineer has automated a web service deployment by using AWS CodePipeline with the following steps: 1. An AWS CodeBuild project compiles the deployment artifact and runs unit tests.
2. An AWS CodeDeploy deployment group deploys the web service to Amazon EC2 instances in the staging environment.
3. A CodeDeploy deployment group deploys the web service to EC2 instances in the production environment. The quality assurance (QA) team requests permission to inspect the build artifact before the deployment to the production environment occurs. The QA team wants to run an internal penetration testing tool to conduct manual tests. The tool will be invoked by a REST API call.
Which combination of actions should the DevOps engineer take to fulfill this request? (Choose two.)

A.Insert a manual approval action between the test actions and deployment actions of the pipeline.

B.Modify the buildspec.yml file for the compilation stage to require manual approval before completion.

C.Update the CodeDeploy deployment groups so that they require manual approval to proceed.

D.Update the pipeline to directly call the REST API for the penetration testing tool.

E.Update the pipeline to invoke an AWS Lambda function that calls the REST API for the penetration testing tool.

Answer: AE

Explanation:

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

A. Insert a manual approval action between the test actions and deployment actions of the pipeline. This is the most appropriate way to introduce a gate for QA approval. By adding a manual approval action after the unit tests (done in the CodeBuild stage) and before the production deployment (done in the CodeDeploy stage), the pipeline will pause and wait for an authorized user (QA team member) to explicitly approve the artifact. This allows the QA team to inspect the artifact and decide whether to proceed with the deployment. This approach aligns with the goal of manual inspection and human-in-the-loop verification before promoting to production.

E. Update the pipeline to invoke an AWS Lambda function that calls the REST API for the penetration testing tool. Directly calling the REST API from within the pipeline is generally not a best practice and might expose credentials. Instead, using a Lambda function provides a secure and manageable way to interact with the external tool. The Lambda function encapsulates the logic for calling the REST API, handling
authentication, and potentially transforming the data. The pipeline invokes the Lambda function, which then interacts with the penetration testing tool, providing a decoupled and secure architecture. The Lambda

function can be granted specific IAM permissions to access the necessary resources and the REST API, limiting the blast radius of any potential security breach.

Here's why the other options are incorrect:

B. Modify the buildspec.yml file for the compilation stage to require manual approval before completion. The buildspec.yml file is primarily for defining build commands and doesn't have a built-in mechanism for manual approval. Trying to implement a manual approval within the build stage would be cumbersome and non-standard.

C. Update the CodeDeploy deployment groups so that they require manual approval to proceed. CodeDeploy deployment groups already provide options for manual approval during the deployment process. However, in this case, manual approval is required to inspect the build artifact before it is deployed to production instances.

D. Update the pipeline to directly call the REST API for the penetration testing tool. While technically feasible, this is less secure and less maintainable than using a Lambda function. Storing API keys and sensitive information directly in the pipeline definition is a security risk.

Supporting Links:

AWS CodePipeline Manual Approval Actions:
https://docs.aws.amazon.com/codepipeline/latest/userguide/approvals.html
AWS Lambda:https://aws.amazon.com/lambda/
AWS CodePipeline:https://aws.amazon.com/codepipeline/
AWS CodeBuild:https://aws.amazon.com/codebuild/
AWS CodeDeploy:https://aws.amazon.com/codedeploy/

Question: 57

A company is hosting a web application in an AWS Region. For disaster recovery purposes, a second region is being used as a standby. Disaster recovery requirements state that session data must be replicated between regions in near-real time and 1% of requests should route to the secondary region to continuously verify system functionality.
Additionally, if there is a disruption in service in the main region, traffic should be automatically routed to the secondary region, and the secondary region must be able to scale up to handle all traffic.
How should a DevOps engineer meet these requirements?

A.In both regions, deploy the application on AWS Elastic Beanstalk and use Amazon DynamoDB global tables for session data. Use an Amazon Route 53 weighted routing policy with health checks to distribute the traffic across the regions.

B.In both regions, launch the application in Auto Scaling groups and use DynamoDB for session data. Use a Route 53 failover routing policy with health checks to distribute the traffic across the regions.

C.In both regions, deploy the application in AWS Lambda, exposed by Amazon API Gateway, and use Amazon RDS for PostgreSQL with cross-region replication for session data. Deploy the web application with client-side logic to call the API Gateway directly.

D.In both regions, launch the application in Auto Scaling groups and use DynamoDB global tables for session data. Enable an Amazon CloudFront weighted distribution across regions. Point the Amazon Route 53 DNS record at the CloudFront distribution.

Answer: A

Explanation:

Here's a detailed justification for why option A is the best solution, incorporating relevant concepts and links:

Justification:

Option A provides the most complete solution for meeting all requirements. It leverages several key AWS services designed for high availability, disaster recovery, and global distribution.

1. **Elastic Beanstalk and Auto Scaling:** Elastic Beanstalk simplifies the deployment and management of web applications. Auto Scaling within Elastic Beanstalk configurations (or explicitly with Auto Scaling groups as in other options) ensures that the application can automatically scale resources based on demand, meeting the requirement for the secondary region to handle all traffic during failover.

2. **DynamoDB Global Tables:** DynamoDB Global Tables provide near real-time replication of data across AWS Regions. This addresses the requirement for session data replication between regions for a consistent user experience during failover. (https://aws.amazon.com/dynamodb/global-tables/)

3. **Route 53 Weighted Routing Policy with Health Checks:** Route 53 weighted routing is ideal for distributing traffic across multiple regions. Assigning a weight of 99 to the primary region and 1 to the secondary region satisfies the 1% traffic routing requirement to continuously verify system functionality in the secondary region. Health checks monitor the health of the application in each region. If the primary region becomes unhealthy, Route 53 will automatically shift traffic to the healthy secondary region, achieving automatic failover.

   (https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html#routing-policy-weighted)

Why other options are less ideal:

Option B: While Route 53 failover routing provides automatic failover, it doesn't support the weighted routing requirement for directing a small percentage of traffic to the secondary region for continuous verification.

Option C: Using Lambda and API Gateway for the entire web application, while possible, introduces unnecessary complexity for a typical web app scenario. RDS for PostgreSQL with cross-region replication is more complex to manage than DynamoDB Global Tables, especially for session data, and it won't scale as easily. Client-side logic calling API Gateway directly introduces security risks.

Option D: While DynamoDB Global Tables are suitable, CloudFront is primarily a content delivery network (CDN) and is best suited for caching static content closer to users. Although CloudFront supports origin failover, it is not the ideal mechanism for directing a small percentage of requests for continuous verification or for managing application logic failover. Using it this way is also more expensive than simply using Route53 with healthchecks. Route 53 provides more granular control over routing policies and health checks for application availability.

In summary, option A strikes the right balance between simplicity, cost-effectiveness, and functionality by effectively utilizing Elastic Beanstalk, DynamoDB Global Tables, and Route 53 weighted routing with health checks to meet all the stated requirements.

Question: 58

A company runs an application on Amazon EC2 instances. The company uses a series of AWS CloudFormation stacks to define the application resources. A developer performs updates by building and testing the application on a laptop and then uploading the build output and CloudFormation stack templates to Amazon S3. The developer's peers review the changes before the developer performs the CloudFormation stack update and installs a new version of the application onto the EC2 instances.
The deployment process is prone to errors and is time-consuming when the developer updates each EC2 instance with the new application. The company wants to automate as much of the application deployment process as possible while retaining a final manual approval step before the modification of the application or resources. The company already has moved the source code for the application and the CloudFormation templates to AWS CodeCommit. The company also has created an AWS CodeBuild project to build and test the application.
Which combination of steps will meet the company’s requirements? (Choose two.)

A.Create an application group and a deployment group in AWS CodeDeploy. Install the CodeDeploy agent on the EC2 instances.

B.Create an application revision and a deployment group in AWS CodeDeploy. Create an environment in CodeDeploy. Register the EC2 instances to the CodeDeploy environment.

C.Use AWS CodePipeline to invoke the CodeBuild job, run the CloudFormation update, and pause for a manual approval step. After approval, start the AWS CodeDeploy deployment.

D.Use AWS CodePipeline to invoke the CodeBuild job, create CloudFormation change sets for each of the application stacks, and pause for a manual approval step. After approval, run the CloudFormation change sets and start the AWS CodeDeploy deployment.

E.Use AWS CodePipeline to invoke the CodeBuild job, create CloudFormation change sets for each of the application stacks, and pause for a manual approval step. After approval, start the AWS CodeDeploy deployment.

Answer: AD

Explanation:

The correct answer is **AD.** Let's break down why:

A: Create an application group and a deployment group in AWS CodeDeploy. Install the CodeDeploy agent on the EC2 instances.

This step is crucial for automating application deployments to the EC2 instances. AWS CodeDeploy automates code deployments to various compute services like EC2. To use CodeDeploy, you need to:

Install the CodeDeploy agent: This agent runs on your EC2 instances and is responsible for pulling the application revision (the new code) from S3 and deploying it according to your deployment configuration.

https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent.html
Create an application: This is a container for organizing deployments.

Create a deployment group: This defines the set of EC2 instances that will be part of the deployment. You can use tags, Auto Scaling groups, or EC2 instance names to identify these instances.
https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-create-deployment-group.html

D: Use AWS CodePipeline to invoke the CodeBuild job, create CloudFormation change sets for each of the application stacks, and pause for a manual approval step. After approval, run the CloudFormation change sets and start the AWS CodeDeploy deployment.

This step orchestrates the entire deployment process and incorporates the required manual approval. AWS CodePipeline is a continuous delivery service that enables you to automate your release pipelines. This makes it perfect for this scenario. https://aws.amazon.com/codepipeline/

Invoke CodeBuild: The pipeline starts by triggering the CodeBuild project, which builds and tests the application.

Create CloudFormation Change Sets: Instead of directly updating the CloudFormation stacks, the pipeline creates change sets. Change sets allow you to preview the changes that CloudFormation will make to your resources before applying them. This is a crucial safety measure, especially in production environments.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html
Manual Approval: The pipeline then pauses for a manual approval. Someone can review the change sets and decide whether to proceed. This fulfills the requirement of a final manual approval step.

Execute Change Sets: After approval, the pipeline executes the change sets, updating the CloudFormation stacks.

Start CodeDeploy: Finally, the pipeline triggers a CodeDeploy deployment to update the application on the EC2 instances.

Why other options are incorrect:

B: While application revisions and deployment groups are part of CodeDeploy, "creating an environment" is not part of the CodeDeploy steps. It adds unnecessary complexity, and the registration of the EC2 instances is already covered within deployment groups.

C & E: These options perform the CloudFormation update before the CodeDeploy deployment. This is not necessarily wrong, but less optimal. It's usually best to update the infrastructure first (CloudFormation change set), then update the application running on that infrastructure (CodeDeploy). Option E includes starting the CodeDeploy before executing the CloudFormation changes which would cause inconsistencies.

Question: 59

A DevOps engineer manages a web application that runs on Amazon EC2 instances behind an Application Load Balancer (ALB). The instances run in an EC2 Auto Scaling group across multiple Availability Zones. The engineer needs to implement a deployment strategy that:
Launches a second fleet of instances with the same capacity as the original fleet.
Maintains the original fleet unchanged while the second fleet is launched.
Transitions traffic to the second fleet when the second fleet is fully deployed.
Terminates the original fleet automatically 1 hour after transition.
Which solution will satisfy these requirements?

A.Use an AWS CloudFormation template with a retention policy for the ALB set to 1 hour. Update the Amazon Route 53 record to reflect the new ALB.

B.Use two AWS Elastic Beanstalk environments to perform a blue/green deployment from the original environment to the new one. Create an application version lifecycle policy to terminate the original environment in 1 hour.

C.Use AWS CodeDeploy with a deployment group configured with a blue/green deployment configuration Select the option Terminate the original instances in the deployment group with a waiting period of 1 hour.

D.Use AWS Elastic Beanstalk with the configuration set to Immutable. Create an .ebextension using the Resources key that sets the deletion policy of the ALB to 1 hour, and deploy the application.

Answer: C

Explanation:

The correct answer is C because it directly addresses all requirements of the deployment strategy. AWS CodeDeploy's blue/green deployment capability is designed for exactly this type of scenario: launching a new fleet, transitioning traffic, and terminating the old fleet.

Here's a breakdown:

Launches a second fleet of instances: CodeDeploy's blue/green deployment creates a new environment (the "green" environment) which includes launching a new fleet of EC2 instances with the desired capacity.

Maintains the original fleet unchanged: The original environment (the "blue" environment) remains untouched while the green environment is being provisioned and tested.

Transitions traffic to the second fleet when fully deployed: CodeDeploy manages the traffic shifting. It waits until the new fleet is healthy and then gradually shifts traffic to the new environment using the configured deployment settings.

Terminates the original fleet automatically 1 hour after transition: CodeDeploy offers a configuration setting called "Terminate the original instances in the deployment group with a waiting period" which perfectly fulfills this requirement. Setting the waiting period to 1 hour will ensure the old instances are terminated 1 hour after the traffic is shifted.

Option A is incorrect because while CloudFormation can be used to manage the infrastructure, its retention policy is not a suitable mechanism for traffic shifting and automated termination in the context of blue/green deployment. Route 53 record updates need to be carefully coordinated with health checks of the new environment, something CodeDeploy automates.

Option B using Elastic Beanstalk comes close but lacks the granular control over termination timing compared to CodeDeploy's direct setting. While Beanstalk supports blue/green deployments, using application version lifecycle policies isn't the most direct or reliable method to achieve a precise 1-hour delay before terminating the old environment.

Option D using .ebextensions for deletion policy manipulation on the ALB is risky. It couples infrastructure lifecycle management with application deployment logic in a potentially unstable manner. It is also harder to control the exact timing of the deletion. The ALB deletion might take longer or fail, leaving the old environment running.

Authoritative links:

AWS CodeDeploy Blue/Green Deployments:
https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-create-blue-green.html CodeDeploy Deployment Group Settings:
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-deployment-groups.html

Question: 60

A video-sharing company stores its videos in Amazon S3. The company has observed a sudden increase in video access requests, but the company does not know which videos are most popular. The company needs to identify the general access pattern for the video files. This pattern includes the number of users who access a certain file on a given day, as well as the number of pull requests for certain files.
How can the company meet these requirements with the LEAST amount of effort?

A.Activate S3 server access logging. Import the access logs into an Amazon Aurora database. Use an Aurora SQL query to analyze the access patterns.

B.Activate S3 server access logging. Use Amazon Athena to create an external table with the log files. Use Athena to create a SQL query to analyze the access patterns.

C.Invoke an AWS Lambda function for every S3 object access event. Configure the Lambda function to write the file access information, such as user. S3 bucket, and file key, to an Amazon Aurora database. Use an Aurora SQL query to analyze the access patterns.

D.Record an Amazon CloudWatch Logs log message for every S3 object access event. Configure a CloudWatch Logs log stream to write the file access information, such as user, S3 bucket, and file key, to an Amazon Kinesis Data Analytics for SQL application. Perform a sliding window analysis.

Answer: B

Explanation:

The correct answer is B because it provides the most efficient and cost-effective solution for analyzing S3 access patterns with minimal effort. S3 server access logging is a built-in feature that automatically captures detailed information about every request made to the S3 buckets.
[https://docs.aws.amazon.com/AmazonS3/latest/userguide/logging-with-S3.html]

Athena, a serverless interactive query service, allows direct querying of data stored in S3 using standard SQL. Creating an external table in Athena pointing to the S3 access logs eliminates the need for data ingestion and transformation. Athena then lets you quickly run SQL queries to analyze the access patterns, determining the number of users accessing specific files and the frequency of requests. This approach avoids the overhead and complexity of managing infrastructure like databases or Lambda functions.

Option A, while also using server access logs, involves importing the logs into Aurora, a managed relational database. This requires setting up and managing an Aurora cluster, data loading processes, and ongoing maintenance, which is more complex than using Athena. Aurora is better suited for structured, transactional data, rather than log analysis.

Option C involves invoking a Lambda function for every S3 object access. This serverless compute approach adds significant overhead and cost, as each request triggers a function execution. It also increases complexity through function deployment, error handling, and scalability management, making it less efficient for large-scale access pattern analysis than server access logging. Furthermore, writing directly to Aurora from Lambda creates a tight coupling and scalability challenges.

Option D uses CloudWatch Logs and Kinesis Data Analytics. While CloudWatch logs can capture S3 events and Kinesis can perform real time analysis, this solution adds unnecesary complexity and cost. It's more efficient to use the built in S3 access logging features and Athena for direct querying.

In summary, the combination of S3 server access logging and Athena offers a simple, scalable, and cost-effective solution for analyzing S3 access patterns without requiring complex infrastructure or custom coding.

Question: 61

A development team wants to use AWS CloudFormation stacks to deploy an application. However, the developer IAM role does not have the required permissions to provision the resources that are specified in the AWS CloudFormation template. A DevOps engineer needs to implement a solution that allows the developers to deploy the stacks. The solution must follow the principle of least privilege.
Which solution will meet these requirements?

A.Create an IAM policy that allows the developers to provision the required resources. Attach the policy to the developer IAM role.

B.Create an IAM policy that allows full access to AWS CloudFormation. Attach the policy to the developer IAM role.

C.Create an AWS CloudFormation service role that has the required permissions. Grant the developer IAM role a cloudformation:* action. Use the new service role during stack deployments.

D.Create an AWS CloudFormation service role that has the required permissions. Grant the developer IAM role the iam:PassRole permission. Use the new service role during stack deployments.

Answer: D

Explanation:

The correct solution is D because it adheres to the principle of least privilege and leverages AWS CloudFormation's service roles to grant necessary permissions without directly assigning broad permissions to the developer IAM role.

Option D utilizes a CloudFormation service role. A service role is an IAM role that a service (in this case, CloudFormation) assumes to perform actions on your behalf. This role has the permissions required to create, update, or delete resources defined in the CloudFormation template. Crucially, developers are not given permissions to directly create those resources.

To allow developers to use the service role, they are granted iam:PassRole permission. iam:PassRole allows a user to pass an IAM role to an AWS service, effectively authorizing that service to act on their behalf with the permissions associated with that role. It does not allow them to assume the role themselves, thus maintaining separation of concerns and security. The developer specifies the service role to be used when deploying the stack, and CloudFormation assumes that role to create the resources.

Option A is incorrect because granting developers direct permissions to provision resources, even with a restricted policy, goes against the principle of least privilege. If the template changes and requires new resource types, the policy attached to the developer role would also need to be updated.

Option B is incorrect due to its excessive permissions. Granting full access to CloudFormation is a massive security risk. Developers could then create, update, and delete any stack, potentially impacting other applications or infrastructure.

Option C is incorrect because granting developers the cloudformation:* action provides overly broad permissions, violating the principle of least privilege. It does not restrict the developer to only deploying stacks with the specified service role. The missing piece is iam:PassRole, without which the developer cannot specify the service role for CloudFormation to assume.

In summary, option D is the most secure and appropriate solution because it uses a service role with specific permissions and grants the developer role only the necessary iam:PassRole permission to use the service role, adhering to the principle of least privilege.

Supporting Documentation:

AWS CloudFormation Service Role: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-iam-service-role.html IAM PassRole permission: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_passrole.html Principle of Least Privilege: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html

Question: 62

A production account has a requirement that any Amazon EC2 instance that has been logged in to manually must be terminated within 24 hours. All applications in the production account are using Auto Scaling groups with the Amazon CloudWatch Logs agent configured.
How can this process be automated?

A.Create a CloudWatch Logs subscription to an AWS Step Functions application. Configure an AWS Lambda function to add a tag to the EC2 instance that produced the login event and mark the instance to be decommissioned. Create an Amazon EventBridge rule to invoke a second Lambda function once a day that will terminate all instances with this tag.

B.Create an Amazon CloudWatch alarm that will be invoked by the login event. Send the notification to an Amazon Simple Notification Service (Amazon SNS) topic that the operations team is subscribed to, and have them terminate the EC2 instance within 24 hours.

C.Create an Amazon CloudWatch alarm that will be invoked by the login event. Configure the alarm to send to an Amazon Simple Queue Service (Amazon SQS) queue. Use a group of worker instances to process messages from the queue, which then schedules an Amazon EventBridge rule to be invoked.

D.Create a CloudWatch Logs subscription to an AWS Lambda function. Configure the function to add a tag to the EC2 instance that produced the login event and mark the instance to be decommissioned. Create an Amazon EventBridge rule to invoke a daily Lambda function that terminates all instances with this tag.

Answer: D

Explanation:

Here's a detailed justification for why option D is the best solution to automatically terminate EC2 instances manually logged into within 24 hours, focusing on its suitability and comparing it against the other options:

Option D provides an automated solution that leverages several AWS services: CloudWatch Logs, Lambda, EventBridge, and EC2 tags. First, a CloudWatch Logs subscription streams the EC2 login events to a Lambda function. The Lambda function then tags the EC2 instance indicating it needs to be decommissioned. This tagging is crucial as it provides a mechanism for identification and action later. Finally, an EventBridge rule triggers a daily Lambda function that identifies all instances with the "decommission" tag and terminates them. This approach ensures that the EC2 instances, once manually logged into, are terminated within the stipulated 24-hour window in an automated fashion, satisfying the given requirement. This ensures security compliance and avoids manual intervention.

Option A is similar but more complex, introducing AWS Step Functions which are not necessary for this relatively straightforward task. Step Functions are suitable for orchestrating complex workflows, but here the requirement can be met without it.

Option B relies on human intervention, which contradicts the automation requirement. Sending an SNS notification to the operations team requires manual termination, rendering it unsuitable.

Option C involves CloudWatch alarms, SQS, worker instances, and scheduling EventBridge rules. This introduces unnecessary complexity and more potential points of failure compared to the simpler, more direct solution in Option D. Managing worker instances and an SQS queue adds operational overhead. Also scheduling an EventBridge rule within the worker further complicates the flow.

In summary, option D provides a simple, serverless, and automated solution. It leverages core AWS services effectively to meet the requirement of terminating EC2 instances that have been manually accessed within 24 hours. Tagging the EC2 instance with the identification of the event combined with a scheduled EventBridge rule ensures that a cleanup process terminates instances automatically.

Here are some links for further research:

CloudWatch Logs Subscriptions:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html
AWS Lambda:https://aws.amazon.com/lambda/
Amazon EventBridge:https://aws.amazon.com/eventbridge/
EC2 Instance Tagging:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html

Question: 63

A company has enabled all features for its organization in AWS Organizations. The organization contains 10 AWS accounts. The company has turned on AWS CloudTrail in all the accounts. The company expects the number of AWS accounts in the organization to increase to 500 during the next year. The company plans to use multiple OUs for these accounts.
The company has enabled AWS Config in each existing AWS account in the organization. A DevOps engineer must implement a solution that enables AWS Config automatically for all future AWS accounts that are created in the organization.
Which solution will meet this requirement?

A.In the organization's management account, create an Amazon EventBridge rule that reacts to a CreateAccount API call. Configure the rule to invoke an AWS Lambda function that enables trusted access to AWS Config for the organization.

B.In the organization's management account, create an AWS CloudFormation stack set to enable AWS Config. Configure the stack set to deploy automatically when an account is created through Organizations.

C.In the organization's management account, create an SCP that allows the appropriate AWS Config API calls to enable AWS Config. Apply the SCP to the root-level OU.

D.In the organization's management account, create an Amazon EventBridge rule that reacts to a CreateAccount API call. Configure the rule to invoke an AWS Systems Manager Automation runbook to enable AWS Config for the account.

Answer: B

Explanation:

Here's a detailed justification for why option B is the most suitable solution for automatically enabling AWS Config in new AWS accounts created within an AWS Organization:

The primary goal is to automate the AWS Config enablement process for future accounts in the organization. AWS CloudFormation StackSets are specifically designed to deploy resources across multiple AWS accounts and regions from a single point of control. This makes StackSets ideal for organization-wide configuration tasks like enabling AWS Config.

Option B leverages the "automatic deployment" capability of StackSets with AWS Organizations. By configuring the StackSet to deploy when an account is created within the organization, any newly created account will automatically have AWS Config enabled. The CloudFormation template within the StackSet will define the necessary AWS Config resources (configuration recorder, delivery channel, and any desired configuration rules). This simplifies the management and deployment process.

Conversely, Option A, using EventBridge and Lambda, could work but would be more complex. It requires writing and maintaining custom Lambda code to handle the account creation event and enable Config. Furthermore, enabling trusted access for Config is a one-time organizational action, making it unsuitable for a triggered Lambda function. EventBridge and Lambda are better suited for more reactive, dynamic, and complex use cases than simple Config enablement.

Option C, SCPs (Service Control Policies), control what actions can be performed within an account. While you could use an SCP to allow Config to be enabled, it doesn't actually enable it. SCPs primarily act as guardrails, not enablers. They provide permissions boundaries. You would still need a mechanism like CloudFormation or Lambda to actually create the AWS Config resources in each account.

Option D, using EventBridge and Systems Manager Automation runbook, is also more complex than necessary. While SSM Automation can perform Config enablement, combining it with EventBridge adds unnecessary complexity compared to CloudFormation StackSets. It introduces additional components to manage and troubleshoot.

Therefore, utilizing CloudFormation StackSets with automatic deployment offers the most straightforward, scalable, and manageable solution for automatically enabling AWS Config in all future AWS accounts created within the organization.

Relevant resources:

AWS CloudFormation StackSets:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-concepts.html **AWS Config:**https://aws.amazon.com/config/
AWS Organizations:https://aws.amazon.com/organizations/

Question: 64

A company has many applications. Different teams in the company developed the applications by using multiple languages and frameworks. The applications run on premises and on different servers with different operating systems. Each team has its own release protocol and process. The company wants to reduce the complexity of the release and maintenance of these applications.
The company is migrating its technology stacks, including these applications, to AWS. The company wants centralized control of source code, a consistent and automatic delivery pipeline, and as few maintenance tasks as possible on the underlying infrastructure.
What should a DevOps engineer do to meet these requirements?

A.Create one AWS CodeCommit repository for all applications. Put each application's code in a different branch. Merge the branches, and use AWS CodeBuild to build the applications. Use AWS CodeDeploy to deploy the applications to one centralized application server.

B.Create one AWS CodeCommit repository for each of the applications. Use AWS CodeBuild to build the applications one at a time. Use AWS CodeDeploy to deploy the applications to one centralized application server.

C.Create one AWS CodeCommit repository for each of the applications. Use AWS CodeBuild to build the applications one at a time and to create one AMI for each server. Use AWS CloudFormation StackSets to automatically provision and decommission Amazon EC2 fleets by using these AMIs.

D.Create one AWS CodeCommit repository for each of the applications. Use AWS CodeBuild to build one Docker image for each application in Amazon Elastic Container Registry (Amazon ECR). Use AWS CodeDeploy to deploy the applications to Amazon Elastic Container Service (Amazon ECS) on infrastructure that AWS Fargate manages.

Answer: D

Explanation:

Here's a breakdown of why option D is the best solution and why the others aren't, followed by supporting documentation.

Justification for Option D:

Option D provides the most comprehensive solution to meet all requirements. It addresses source code control, consistent delivery pipelines, and minimizes infrastructure maintenance through containerization.

Creating a separate CodeCommit repository for each application aligns with best practices for code organization, separation of concerns, and independent development lifecycles. Using CodeBuild to create Docker images ensures consistent build artifacts, regardless of the underlying language or framework used by each application. These images are then stored in Amazon ECR, providing a secure and centralized container registry. Deploying to Amazon ECS using Fargate abstracts away the underlying server
infrastructure management. Fargate handles provisioning, scaling, and patching, minimizing operational overhead and addressing the "as few maintenance tasks as possible" requirement. CodeDeploy integrates well with ECS and facilitates automated deployments and rollbacks. This approach also promotes portability, consistency across environments, and scalability.

Why Other Options are Suboptimal:

Option A: Placing all applications into a single CodeCommit repository using branches creates a monolithic repository. This can lead to merge conflicts, difficulties in code ownership, and scaling challenges. Deploying all applications to a single centralized application server introduces a single point of failure and limits independent scaling and updates.

Option B: While using separate CodeCommit repositories is good, deploying all applications to a single centralized application server suffers from the same drawbacks as Option A: a single point of failure and limited independent scalability.

Option C: Using AMIs would create a tight coupling with the OS and would increase the amount of work required to maintain infrastructure. Using CloudFormation StackSets for EC2 is an option, but it doesn't offer the same level of abstraction and reduced management overhead as Fargate with containers. It will still require manual configuration and maintenance of the EC2 instances.

In summary, Option D is preferred because it leverages containerization with Docker and serverless container management with Fargate to achieve centralized control, consistent pipelines, and minimal infrastructure maintenance. The other options fall short on these aspects.

Supporting Documentation:

AWS CodeCommit:https://aws.amazon.com/codecommit/
AWS CodeBuild:https://aws.amazon.com/codebuild/
AWS CodeDeploy:https://aws.amazon.com/codedeploy/
Amazon ECR:https://aws.amazon.com/ecr/
Amazon ECS:https://aws.amazon.com/ecs/
AWS Fargate:https://aws.amazon.com/fargate/
Docker:https://www.docker.com/

Question: 65

A company's application is currently deployed to a single AWS Region. Recently, the company opened a new office on a different continent. The users in the new office are experiencing high latency. The company's application runs on Amazon EC2 instances behind an Application Load Balancer (ALB) and uses Amazon DynamoDB as the database layer. The instances run in an EC2 Auto Scaling group across multiple Availability Zones. A DevOps engineer is tasked with minimizing application response times and improving availability for users in both Regions.
Which combination of actions should be taken to address the latency issues? (Choose three.)

A.Create a new DynamoDB table in the new Region with cross-Region replication enabled.

B.Create new ALB and Auto Scaling group global resources and configure the new ALB to direct traffic to the new Auto Scaling group.

C.Create new ALB and Auto Scaling group resources in the new Region and configure the new ALB to direct traffic to the new Auto Scaling group.

D.Create Amazon Route 53 records, health checks, and latency-based routing policies to route to the ALB.

E.Create Amazon Route 53 aliases, health checks, and failover routing policies to route to the ALB.

F.Convert the DynamoDB table to a global table.

Answer: CDF

Explanation:

Here's a detailed justification for choosing options C, D, and F as the solution to minimize application response times and improve availability for a globally distributed application with high latency in a new region:

C. Create new ALB and Auto Scaling group resources in the new Region and configure the new ALB to direct traffic to the new Auto Scaling group: Deploying the application stack (ALB and EC2 instances in an Auto Scaling group) in the new region directly addresses the latency issue for users in that region. By hosting the application closer to these users, network latency is significantly reduced. This approach implements a multi-region deployment strategy.

D. Create Amazon Route 53 records, health checks, and latency-based routing policies to route to the ALB: Route 53's latency-based routing is crucial for directing users to the closest available application deployment (either the original region or the new region). Health checks ensure that only healthy instances behind the ALBs receive traffic, improving application availability. Latency-based routing minimizes the network distance traffic must travel.
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html#routing-policy-latency

F. Convert the DynamoDB table to a global table: DynamoDB global tables provide fully managed, multi-region, multi-master replication. Converting the existing DynamoDB table to a global table replicates data across regions, ensuring low-latency data access for users in both regions. This eliminates the need for users in the new region to access data in the original region, further reducing latency and improving application performance.
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/globaltables.html

Why other options are not the best choices:

A.Create a new DynamoDB table in the new Region with cross-Region replication enabled: This is a more complex and potentially less efficient solution compared to using DynamoDB Global Tables. Global Tables offer automatic and managed replication, while this approach requires manual management or custom replication logic. This increases operational overhead.

B. Create new ALB and Auto Scaling group global resources and configure the new ALB to direct traffic to the new Auto Scaling group: Global resources (like the AWS Global Accelerator) are not suitable for ALB and Auto Scaling groups. ALB and Auto Scaling groups exist within a specific region. While AWS Global Accelerator could be considered for static content, its use case is different than regional ALBs managing dynamic application traffic.

E. Create Amazon Route 53 aliases, health checks, and failover routing policies to route to the ALB: Failover routing is good for high availability, but it isn't the most effective method to reduce latency in this situation.

While it offers redundancy, failover routing doesn't actively direct users to the lowest latency endpoint in a typical configuration (it's primarily for directing traffic if one region fails). Latency-based routing is far more appropriate for the described scenario.

Question: 66

A DevOps engineer needs to apply a core set of security controls to an existing set of AWS accounts. The accounts are in an organization in AWS Organizations. Individual teams will administer individual accounts by using the AdministratorAccess AWS managed policy. For all accounts. AWS CloudTrail and AWS Config must be turned on in all available AWS Regions. Individual account administrators must not be able to edit or delete any of the baseline resources. However, individual account administrators must be able to edit or delete their own CloudTrail trails and AWS Config rules.
Which solution will meet these requirements in the MOST operationally efficient way?

A.Create an AWS CloudFormation template that defines the standard account resources. Deploy the template to all accounts from the organization's management account by using CloudFormation StackSets. Set the stack policy to deny Update:Delete actions.

B.Enable AWS Control Tower. Enroll the existing accounts in AWS Control Tower. Grant the individual account administrators access to CloudTrail and AWS Config.

C.Designate an AWS Config management account. Create AWS Config recorders in all accounts by using AWS CloudFormation StackSets. Deploy AWS Config rules to the organization by using the AWS Config management account. Create a CloudTrail organization trail in the organization’s management account. Deny modification or deletion of the AWS Config recorders by using an SCP.

D.Create an AWS CloudFormation template that defines the standard account resources. Deploy the template to all accounts from the organization's management account by using Cloud Formation StackSets Create an SCP that prevents updates or deletions to CloudTrail resources or AWS Config resources unless the principal is an administrator of the organization's management account.

Answer: C

Explanation:

Here's a breakdown of why option C is the most suitable solution, along with supporting details:

Why Option C is Correct:

Option C offers the most operationally efficient way to enforce the required security controls (CloudTrail and Config enabled in all regions, immutability of baseline resources, and delegated control) while minimizing administrative overhead. Here's a detailed explanation:

Centralized AWS Config Management: Designating a central AWS Config management account lets you deploy AWS Config recorders and rules across the entire organization from a single point. This simplifies configuration and ensures consistency. https://docs.aws.amazon.com/config/latest/developerguide/config-multiple-accounts.html
AWS CloudFormation StackSets for Recorders: StackSets provide a mechanism to consistently deploy AWS Config recorders across all accounts in the organization and all available regions. This ensures that all accounts and regions are monitored from the start. Using a stackset ensures that any newly added accounts also get the Config recorders created.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/what-is-cloudformation.html
CloudTrail Organization Trail: Creating a CloudTrail organization trail from the management account ensures that all activity across the organization is logged centrally. This simplifies auditing and security analysis. This meets the requirement of CloudTrail being enabled in all accounts and regions.

https://docs.aws.amazon.com/awscloudtrail/latest/userguide/creating-an-organizational-trail.html **SCP for Immutability of AWS Config Recorders:** Using a Service Control Policy (SCP) to deny modification or deletion of the AWS Config recorders prevents individual account administrators from tampering with the core monitoring infrastructure. This ensures that baseline security controls remain in place.

https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scp.html
Delegated control of CloudTrail & Config rules: Individual teams get administrative access to edit their trails and rules but cannot edit or delete baseline resources due to the application of SCPs.

Why Other Options Are Less Suitable:

Option A: While StackSets can deploy resources, it doesn't provide a centralized way to manage AWS Config rules across the organization like a Config management account does. Furthermore, setting the stack policy would affect all resources created by the stack, not only AWS Config resources. This limits the individual administrators' ability to customize the deployed stack, impacting agility. It also does not consider a way to manage CloudTrail.

Option B: AWS Control Tower is a more comprehensive solution that sets up an entire landing zone. While it includes features for governance and compliance, it might be overkill for just enabling CloudTrail and Config with restricted access. It would require more overhead to set up and manage compared to Config
Management. The problem statement states the accounts already exist.

Option D: This option uses stacksets and an SCP. It is not as effective as Option C. The CloudFormation stack is more complicated to manage compared to the method described in Option C.

In summary, option C balances centralized control with delegated administration, offering the most efficient approach to meet the given requirements.

Question: 67

A company has its AWS accounts in an organization in AWS Organizations. AWS Config is manually configured in each AWS account. The company needs to implement a solution to centrally configure AWS Config for all accounts in the organization The solution also must record resource changes to a central account.
Which combination of actions should a DevOps engineer perform to meet these requirements? (Choose two.)

A.Configure a delegated administrator account for AWS Config. Enable trusted access for AWS Config in the organization.

B.Configure a delegated administrator account for AWS Config. Create a service-linked role for AWS Config in the organization's management account.

C.Create an AWS CloudFormation template to create an AWS Config aggregator. Configure a CloudFormation stack set to deploy the template to all accounts in the organization.

D.Create an AWS Config organization aggregator in the organization's management account. Configure data collection from all AWS accounts in the organization and from all AWS Regions.

E.Create an AWS Config organization aggregator in the delegated administrator account. Configure data collection from all AWS accounts in the organization and from all AWS Regions.

Answer: AE

Explanation:

The requirement is to centrally manage AWS Config across an AWS Organization and aggregate resource change records into a central account.

Option A is correct because delegating administrative privileges for AWS Config to a dedicated account simplifies management and follows AWS best practices for separation of duties. Enabling trusted access for AWS Config within the organization is essential for AWS Config to operate seamlessly across all member accounts. Trusted access establishes a service-linked role in each member account, allowing Config to collect configuration data without explicit IAM configurations in each account.

Option E is correct because, with a delegated administrator configured, creating an AWS Config organization aggregator in that delegated administrator account allows for the centralized aggregation of configuration data. Specifying data collection from all AWS accounts and all regions ensures complete coverage across the organization.

Option B is incorrect. While a delegated administrator account is necessary, creating a service-linked role in the organization's management account is not required when using trusted access, as trusted access implicitly handles the necessary service-linked roles in member accounts.

Option C is incorrect because using CloudFormation stack sets to create an AWS Config aggregator is a valid approach, but it is more complex than leveraging the delegated administrator approach and managing via the AWS Organizations service linked role, which makes option D and E the more efficient answer.

Option D is incorrect because the AWS Config organization aggregator should be created in the delegated administrator account, not the organization's management account, to maintain separation of duties and follow the delegated administrator model.

Therefore, the correct answer is AE.

Relevant Links:

AWS Config delegated administrator
AWS Config trusted access
AWS Organizations

Question: 68

A company wants to migrate its content sharing web application hosted on Amazon EC2 to a serverless architecture. The company currently deploys changes to its application by creating a new Auto Scaling group of EC2 instances and a new Elastic Load Balancer, and then shifting the traffic away using an Amazon Route 53 weighted routing policy.
For its new serverless application, the company is planning to use Amazon API Gateway and AWS Lambda. The company will need to update its deployment processes to work with the new application. It will also need to retain the ability to test new features on a small number of users before rolling the features out to the entire user base. Which deployment strategy will meet these requirements?

A.Use AWS CDK to deploy API Gateway and Lambda functions. When code needs to be changed, update the AWS CloudFormation stack and deploy the new version of the APIs and Lambda functions. Use a Route 53 failover routing policy for the canary release strategy.

B.Use AWS CloudFormation to deploy API Gateway and Lambda functions using Lambda function versions. When code needs to be changed, update the CloudFormation stack with the new Lambda code and update the API versions using a canary release strategy. Promote the new version when testing is complete.

C.Use AWS Elastic Beanstalk to deploy API Gateway and Lambda functions. When code needs to be changed, deploy a new version of the API and Lambda functions. Shift traffic gradually using an Elastic Beanstalk blue/green deployment.

D.Use AWS OpsWorks to deploy API Gateway in the service layer and Lambda functions in a custom layer. When code needs to be changed, use OpsWorks to perform a blue/green deployment and shift traffic gradually.

Answer: B

Explanation:

Here's a detailed justification for why option B is the correct answer, along with explanations of why the other options are less suitable:

Option B: Use AWS CloudFormation to deploy API Gateway and Lambda functions using Lambda function versions. When code needs to be changed, update the CloudFormation stack with the new Lambda code and update the API versions using a canary release strategy. Promote the new version when testing is complete.

This strategy directly addresses the requirements of a serverless deployment with canary testing. Here's why:

Infrastructure as Code (IaC): CloudFormation is a mature IaC service that allows you to define and provision your entire infrastructure (API Gateway, Lambda functions, permissions, etc.) in a declarative template. This ensures consistency and reproducibility.

Lambda Function Versions: Lambda allows you to create multiple versions of your functions. Each version represents a snapshot of your code at a specific point in time. This is crucial for managing deployments and rollback.

API Gateway Integration: API Gateway can route traffic to specific Lambda function versions or aliases. This is the key to implementing canary deployments.

Canary Releases with Weighted Routing: You can configure API Gateway to send a small percentage of traffic (e.g., 10%) to the new Lambda function version (the "canary"). If no errors are detected, you can gradually increase the traffic percentage to the new version until it receives 100% of the traffic.
Promotion: Once the canary version has been tested and validated, you can promote it to the primary version.

Why other options are incorrect:

Option A: Route 53 failover routing policy is not suitable for canary release Failover routing is designed for high availability by directing traffic to a healthy resource if another fails. It doesn't gradually shift traffic for testing purposes.

Option C: AWS Elastic Beanstalk is not suitable for deploying API Gateway and Lambda functions. Elastic Beanstalk is designed for deploying web applications to EC2 instances. It doesn't natively support deploying serverless applications using API Gateway and Lambda. While you could potentially use Elastic Beanstalk to manage some aspects of the application, it's not the right tool for this serverless architecture.

Option D: AWS OpsWorks is not suitable for deploying serverless applications using API Gateway and Lambda OpsWorks is primarily designed for managing and automating the configuration of EC2-based infrastructure, typically using Chef or Puppet. It's not optimized for serverless deployments with API Gateway and Lambda. While you could potentially make it work, CloudFormation is a more direct and appropriate tool for managing the infrastructure in this scenario.

Authoritative Links:

AWS CloudFormation:https://aws.amazon.com/cloudformation/
AWS Lambda Versions:https://docs.aws.amazon.com/lambda/latest/dg/configuration-versions.html **Amazon API Gateway Deployment Stages:**
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-deployment-stages.html **AWS API Gateway Canary Deployments:**https://aws.amazon.com/blogs/compute/performing-canary-deployments-for-your-apis-with-amazon-api-gateway/

In summary, option B provides the most appropriate and direct solution for deploying a serverless application with canary testing using AWS services. It leverages IaC, Lambda versions, and API Gateway integration to achieve the desired functionality.

Question: 69

A development team uses AWS CodeCommit, AWS CodePipeline, and AWS CodeBuild to develop and deploy an application. Changes to the code are submitted by pull requests. The development team reviews and merges the pull requests, and then the pipeline builds and tests the application.
Over time, the number of pull requests has increased. The pipeline is frequently blocked because of failing tests. To prevent this blockage, the development team wants to run the unit and integration tests on each pull request before it is merged.
Which solution will meet these requirements?

A.Create a CodeBuild project to run the unit and integration tests. Create a CodeCommit approval rule template. Configure the template to require the successful invocation of the CodeBuild project. Attach the approval rule to the project's CodeCommit repository.

B.Create an Amazon EventBridge rule to match pullRequestCreated events from CodeCommit Create a CodeBuild project to run the unit and integration tests. Configure the CodeBuild project as a target of the EventBridge rule that includes a custom event payload with the CodeCommit repository and branch information from the event.

C.Create an Amazon EventBridge rule to match pullRequestCreated events from CodeCommit. Modify the existing CodePipeline pipeline to not run the deploy steps if the build is started from a pull request. Configure the EventBridge rule to run the pipeline with a custom payload that contains the CodeCommit repository and branch information from the event.

D.Create a CodeBuild project to run the unit and integration tests. Create a CodeCommit notification rule that matches when a pull request is created or updated. Configure the notification rule to invoke the CodeBuild project.

Answer: B

Explanation:

The correct answer is B. Here's why:

Option B Rationale:

This solution effectively addresses the requirement of running tests on pull requests before they are merged, and it avoids blocking the main pipeline. Here's a breakdown:

1. **Event Trigger:** Amazon EventBridge is used to detect pullRequestCreated events in CodeCommit. This ensures that a test build is automatically triggered whenever a new pull request is created.
2. **Dedicated Test Build:** A dedicated CodeBuild project is created specifically for running unit and integration tests on pull requests. This isolates the testing process from the main deployment pipeline.
3. **Payload Transfer:** The EventBridge rule is configured to pass relevant information (repository name, branch name) to the CodeBuild project as a custom payload. This allows CodeBuild to know which code to test.
4. **Independent Testing:** This setup ensures that the CodeBuild project runs the tests independently on each pull request. Developers get immediate feedback on the test results, enabling them to address issues before merging the code.
5. **Non-Blocking Pipeline:** The main CodePipeline remains unblocked since the testing of pull requests occurs separately. Failed tests in a pull request will not affect the deployment pipeline.

Why other options are not the best choice:

Option A: Approval rules in CodeCommit can enforce specific conditions before a pull request is merged. It's useful for requiring approvals or code reviews. But directly requiring the success of a CodeBuild project using approval rules requires more manual steps for each PR. EventBridge triggers the tests automatically making it superior.

Option C: Modifying the existing pipeline to skip deploy steps based on pull requests can work, but it might complicate the pipeline logic and potentially lead to errors. EventBridge provides a cleaner way to trigger a dedicated testing process.

Option D: CodeCommit notification rules, while useful for general notifications, are not designed to directly trigger a build process with custom payload in the most efficient manner. EventBridge is better suited for this event-driven scenario.

In Summary: Option B offers the most scalable and automated solution. It triggers testing immediately upon pull request creation, providing developers with quick feedback and preventing the main pipeline from being blocked by failing tests.

Authoritative Links:

Amazon EventBridge:https://aws.amazon.com/eventbridge/

AWS CodeBuild:https://aws.amazon.com/codebuild/
AWS CodeCommit:https://aws.amazon.com/codecommit/

Question: 70

A company has an application that runs on a fleet of Amazon EC2 instances. The application requires frequent restarts. The application logs contain error messages when a restart is required. The application logs are published to a log group in Amazon CloudWatch Logs.
An Amazon CloudWatch alarm notifies an application engineer through an Amazon Simple Notification Service (Amazon SNS) topic when the logs contain a large number of restart-related error messages. The application engineer manually restarts the application on the instances after the application engineer receives a notification from the SNS topic.
A DevOps engineer needs to implement a solution to automate the application restart on the instances without restarting the instances.
Which solution will meet these requirements in the MOST operationally efficient manner?

A.Configure an AWS Systems Manager Automation runbook that runs a script to restart the application on the instances. Configure the SNS topic to invoke the runbook.

B.Create an AWS Lambda function that restarts the application on the instances. Configure the Lambda function as an event destination of the SNS topic.

C.Configure an AWS Systems Manager Automation runbook that runs a script to restart the application on the instances. Create an AWS Lambda function to invoke the runbook. Configure the Lambda function as an event destination of the SNS topic.

D.Configure an AWS Systems Manager Automation runbook that runs a script to restart the application on the instances. Configure an Amazon EventBridge rule that reacts when the CloudWatch alarm enters ALARM state. Specify the runbook as a target of the rule.

Answer: D

Explanation:

The most operationally efficient solution is **D. Configure an AWS Systems Manager Automation runbook that runs a script to restart the application on the instances. Configure an Amazon EventBridge rule that reacts when the CloudWatch alarm enters ALARM state. Specify the runbook as a target of the rule.**

Here's a detailed justification:

Systems Manager Automation Runbook: This provides a structured and repeatable way to restart the application on the EC2 instances. It allows you to define the steps necessary for the restart process in a documented and auditable manner.
Amazon EventBridge: EventBridge enables you to react to events in your AWS environment. In this case, you can configure an EventBridge rule to trigger when the CloudWatch alarm transitions to the ALARM state, indicating a high number of restart-related error messages.
Direct Integration: EventBridge can directly invoke a Systems Manager Automation runbook as a target. This eliminates the need for an intermediary Lambda function, simplifying the architecture and reducing operational overhead. The tight integration reduces complexity and potential points of failure.
Operational Efficiency: By using EventBridge, you avoid the need to manage and maintain a Lambda function specifically for triggering the runbook. This reduces the amount of code you need to write, deploy, and monitor. This is the key aspect of 'operational efficiency'.
Scalability and Reliability: EventBridge is a highly scalable and reliable event bus, ensuring that the restart process is triggered promptly and consistently. It offers built-in retry mechanisms and error handling. **No Polling:** This solution is event-driven, meaning that it reacts to the CloudWatch alarm state change in real-time without polling or scheduled tasks.
Security: Systems Manager Automation leverages IAM roles for secure access to EC2 instances, ensuring that only authorized processes can restart the application.

Why other options are not ideal:

Option A: Using SNS to directly invoke the runbook is not optimal. SNS is primarily designed for human notifications or simple message distribution, not for complex workflow orchestration like triggering a Systems Manager Automation runbook. The SNS message structure might not directly map to the runbook's input parameters, making the integration more complex.

Option B: While Lambda can restart the application, managing credentials and logic within Lambda, along with managing EC2 instance interaction, is more complex than using Systems Manager. Lambda is introducing an unnecessary layer of code for performing a well-defined operational task that can be natively handled by Systems Manager.

Option C: While this utilizes an automation runbook (good), introducing a Lambda function to trigger the runbook is redundant. EventBridge provides native integration to trigger systems manager runbooks, so adding lambda is just increasing the attack surface and operational overhead.

Supporting Links:

AWS Systems Manager Automation:https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-automation.html
Amazon EventBridge:https://aws.amazon.com/eventbridge/
CloudWatch Alarms:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html

Question: 71

A DevOps engineer at a company is supporting an AWS environment in which all users use AWS IAM Identity Center (AWS Single Sign-On). The company wants to immediately disable credentials of any new IAM user and wants the security team to receive a notification.
Which combination of steps should the DevOps engineer take to meet these requirements? (Choose three.)

A.Create an Amazon EventBridge rule that reacts to an IAM CreateUser API call in AWS CloudTrail.

B.Create an Amazon EventBridge rule that reacts to an IAM GetLoginProfile API call in AWS CloudTrail.

C.Create an AWS Lambda function that is a target of the EventBridge rule. Configure the Lambda function to disable any access keys and delete the login profiles that are associated with the IAM user.

D.Create an AWS Lambda function that is a target of the EventBridge rule. Configure the Lambda function to delete the login profiles that are associated with the IAM user.

E.Create an Amazon Simple Notification Service (Amazon SNS) topic that is a target of the EventBridge rule. Subscribe the security team's group email address to the topic.

F.Create an Amazon Simple Queue Service (Amazon SQS) queue that is a target of the Lambda function. Subscribe the security team's group email address to the queue.

Answer: ACE

Explanation:

The requirement is to disable new IAM user credentials and notify the security team whenever a new IAM user is created while using IAM Identity Center (SSO).

A: Create an Amazon EventBridge rule that reacts to an IAM CreateUser API call in AWS CloudTrail. This is necessary to trigger the automation when a new IAM user is created. CloudTrail logs API calls, and EventBridge can react to these logs based on defined rules. Specifically, we need to monitor CreateUser API calls to detect the creation of a new user.
https://docs.aws.amazon.com/eventbridge/latest/userguide/cloudtrail-event-source.html

C: Create an AWS Lambda function that is a target of the EventBridge rule. Configure the Lambda function to disable any access keys and delete the login profiles that are associated with the IAM user. The Lambda

function is essential to automate the disabling of credentials. Upon detecting a CreateUser event, EventBridge will invoke the Lambda function. The Lambda function will then disable any access keys and remove login profiles for the newly created IAM user. Disabling access keys and deleting login profiles effectively prevents the new user from directly accessing AWS resources using IAM. It's crucial to note that deleting login profiles only applies to IAM users directly created in IAM and not federated users created through Identity Center.
https://docs.aws.amazon.com/lambda/latest/dg/services-eventbridge.html

E: Create an Amazon Simple Notification Service (Amazon SNS) topic that is a target of the EventBridge rule. Subscribe the security team's group email address to the topic. SNS is used for sending notifications.
By setting up an SNS topic and subscribing the security team's email address, the team will be notified whenever a new IAM user is created. This can be done directly from EventBridge to SNS when IAM user gets created. EventBridge can be configured to send a notification to the SNS topic upon detecting a CreateUser event, which will then forward the notification to the subscribed email address.
https://docs.aws.amazon.com/sns/latest/dg/welcome.html

Why other options are incorrect:

B: Create an Amazon EventBridge rule that reacts to an IAM GetLoginProfile API call in AWS CloudTrail. GetLoginProfile would only indicate that someone is retrieving the login profile, not that a new user has been created, so this is not helpful.

D: Create an AWS Lambda function that is a target of the EventBridge rule. Configure the Lambda function to delete the login profiles that are associated with the IAM user. This only deletes the login profile, but does not disable any access keys. Access keys also need to be disabled.

F: Create an Amazon Simple Queue Service (Amazon SQS) queue that is a target of the Lambda function. Subscribe the security team's group email address to the queue. SQS is a queuing service, not a notification service. While you could poll the queue and then send an email, this is less direct and less efficient than using SNS directly from EventBridge or the Lambda function (after disabling the keys). Using SNS directly from EventBridge or the Lambda function provides a simpler and more direct notification mechanism.

Question: 72

A company wants to set up a continuous delivery pipeline. The company stores application code in a private GitHub repository. The company needs to deploy the application components to Amazon Elastic Container Service (Amazon ECS). Amazon EC2, and AWS Lambda. The pipeline must support manual approval actions.
Which solution will meet these requirements?

A.Use AWS CodePipeline with Amazon ECS. Amazon EC2, and Lambda as deploy providers.
B.Use AWS CodePipeline with AWS CodeDeploy as the deploy provider.
C.Use AWS CodePipeline with AWS Elastic Beanstalk as the deploy provider.
D.Use AWS CodeDeploy with GitHub integration to deploy the application.

Answer: B

Explanation:

The correct answer is B: Use AWS CodePipeline with AWS CodeDeploy as the deploy provider. Here's why:

CodePipeline's Role: AWS CodePipeline is a fully managed continuous delivery service that automates the release process. It orchestrates the different stages of your pipeline, from source code retrieval to deployment. (Reference: https://aws.amazon.com/codepipeline/)
CodeDeploy's Role: AWS CodeDeploy automates application deployments to a variety of compute services, including Amazon EC2, AWS Lambda, and Amazon ECS. It eliminates the need for manual deployment processes and reduces errors. (Reference: https://aws.amazon.com/codedeploy/)
Supporting Multiple Deployment Targets: CodeDeploy offers deployment strategies customized for each

service mentioned in the question. Specifically, it allows ECS blue/green deployments, EC2 deployments using in-place or blue/green strategies, and Lambda deployments via traffic shifting.

Manual Approvals: CodePipeline natively supports manual approval actions. This means you can insert a stage in your pipeline that requires a person to approve the deployment before it proceeds to the next stage. **GitHub Integration:** CodePipeline directly integrates with GitHub as a source code provider. This allows your pipeline to automatically trigger when changes are pushed to your private GitHub repository.

Why other options are incorrect:

A: While CodePipeline can directly integrate with ECS, EC2 and Lambda to some extent, CodeDeploy offers more comprehensive deployment strategies and finer-grained control, especially for complex deployment patterns such as blue/green. Direct ECS, EC2, Lambda integrations in CodePipeline are typically simpler deployments.

C: AWS Elastic Beanstalk is primarily a platform-as-a-service (PaaS) and while it can support some deployments, it does not offer as much flexibility and control as CodeDeploy, especially for complex continuous delivery pipelines and deployments across multiple services.

D: CodeDeploy is a deployment service, not a pipeline orchestration service. While CodeDeploy can deploy from GitHub, it doesn't inherently provide the overall pipeline structure, stages, and manual approval capabilities that CodePipeline offers. It also wouldn't orchestrate deployments across ECS, EC2, and Lambda in a unified pipeline.

In summary, using CodePipeline to orchestrate the pipeline and CodeDeploy as the deployment provider is the most complete solution for a continuous delivery pipeline that supports multiple deployment targets (ECS, EC2, Lambda), manual approval actions, and integrates with a private GitHub repository.

Question: 73

A company has an application that runs on Amazon EC2 instances that are in an Auto Scaling group. When the application starts up. the application needs to process data from an Amazon S3 bucket before the application can start to serve requests. The size of the data that is stored in the S3 bucket is growing. When the Auto Scaling group adds new instances, the application now takes several minutes to download and process the data before the application can serve requests. The company must reduce the time that elapses before new EC2 instances are ready to serve requests. Which solution is the MOST cost-effective way to reduce the application startup time?

A.Configure a warm pool for the Auto Scaling group with warmed EC2 instances in the Stopped state. Configure an autoscaling:EC2_INSTANCE_LAUNCHING lifecycle hook on the Auto Scaling group. Modify the application to complete the lifecycle hook when the application is ready to serve requests.

B.Increase the maximum instance count of the Auto Scaling group. Configure an autoscaling:EC2_INSTANCE_LAUNCHING lifecycle hook on the Auto Scaling group. Modify the application to complete the lifecycle hook when the application is ready to serve requests.

C.Configure a warm pool for the Auto Scaling group with warmed EC2 instances in the Running state. Configure an autoscaling:EC2_INSTANCE_LAUNCHING lifecycle hook on the Auto Scaling group. Modify the application to complete the lifecycle hook when the application is ready to serve requests.

D.Increase the maximum instance count of the Auto Scaling group. Configure an autoscaling:EC2_INSTANCE_LAUNCHING lifecycle hook on the Auto Scaling group. Modify the application to complete the lifecycle hook and to place the new instance in the Standby state when the application is ready to serve requests.

Answer: A

Explanation:

The most cost-effective solution to reduce application startup time in this scenario is option A: configuring a warm pool with stopped instances and using a lifecycle hook.

Here's why:

Warm Pools: Warm pools pre-initialize EC2 instances, drastically reducing the time needed to start serving requests. The core problem is the slow data download from S3 on each new instance launch. Warm pools eliminate this delay for a portion of scaling events by keeping pre-configured instances ready.
https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-warm-pools.html

Stopped vs. Running State: Keeping instances in the Stopped state within the warm pool is crucial for cost-effectiveness. While Running instances (option C) are even faster to deploy, they incur EC2 instance hour charges even when idle. Stopped instances only incur minimal EBS storage costs.

Lifecycle Hooks: The autoscaling:EC2_INSTANCE_LAUNCHING lifecycle hook allows the application to perform necessary setup tasks (in this case, signaling readiness after processing S3 data). The hook pauses the instance launch process until the application signals completion.
https://docs.aws.amazon.com/autoscaling/ec2/userguide/lifecycle-hooks.html

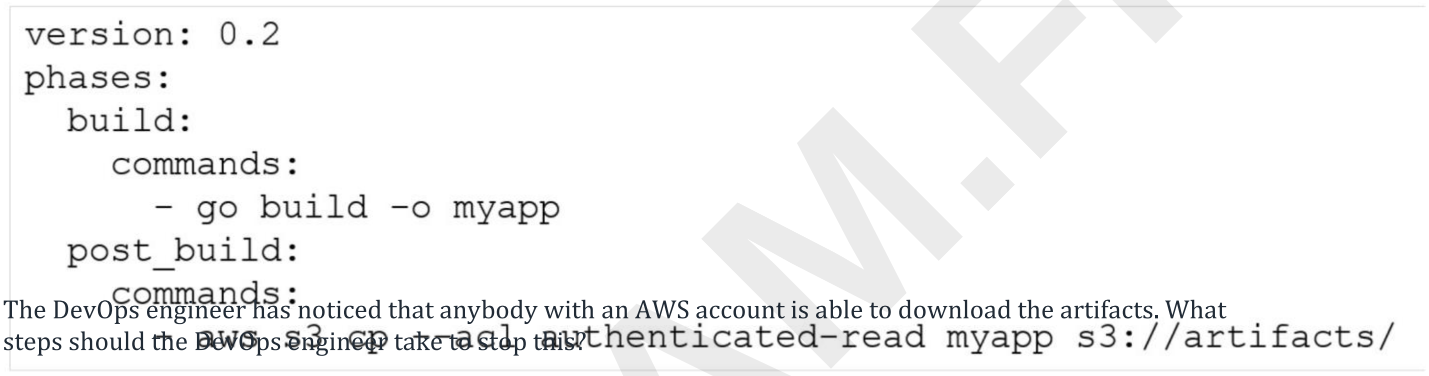
Completing the Lifecycle Hook: By modifying the application to complete the lifecycle hook once it's ready to serve requests, Auto Scaling knows the instance is prepared and can place it into service, routing traffic to it.

Options B and D are less cost-effective because they merely increase the maximum instance count without addressing the root cause of the slow startup. Increasing instance count increases ongoing EC2 costs without solving the S3 data processing delay. Option D also places the instance in Standby state, taking it out of service which defeats the purpose of scaling up.

Therefore, option A offers the best balance of reduced startup time and minimized operational costs.

Question: 74

A company is using an AWS CodeBuild project to build and package an application. The packages are copied to a shared Amazon S3 bucket before being deployed across multiple AWS accounts.
The buildspec.yml file contains the following:



```
version: 0.2
phases:
  build:
    commands:
      - go build -o myapp
  post_build:
    commands:
      - aws s3 cp --acl authenticated-read myapp s3://artifacts/
```

The DevOps engineer has noticed that anybody with an AWS account is able to download the artifacts. What steps should the DevOps engineer take to stop this?

A.Modify the post_build command to use --acl public-read and configure a bucket policy that grants read access to the relevant AWS accounts only.

B.Configure a default ACL for the S3 bucket that defines the set of authenticated users as the relevant AWS accounts only and grants read-only access.

C.Create an S3 bucket policy that grants read access to the relevant AWS accounts and denies read access to the principal "*".

D.Modify the post_build command to remove --acl authenticated-read and configure a bucket policy that allows read access to the relevant AWS accounts only.

Answer: D

Explanation:

Modify the post_ build command to remove --acl authenticated-read and configure a bucket policy that allows read access to the relevant AWS accounts only.

Question: 75

A company has developed a serverless web application that is hosted on AWS. The application consists of Amazon S3. Amazon API Gateway, several AWS Lambda functions, and an Amazon RDS for MySQL database. The company is using AWS CodeCommit to store the source code. The source code is a combination of AWS Serverless Application Model (AWS SAM) templates and Python code.
A security audit and penetration test reveal that user names and passwords for authentication to the database are hardcoded within CodeCommit repositories. A DevOps engineer must implement a solution to automatically detect and prevent hardcoded secrets.
What is the MOST secure solution that meets these requirements?

A.Enable Amazon CodeGuru Profiler. Decorate the handler function with @with_lambda_profiler(). Manually review the recommendation report. Write the secret to AWS Systems Manager Parameter Store as a secure string. Update the SAM templates and the Python code to pull the secret from Parameter Store.

B.Associate the CodeCommit repository with Amazon CodeGuru Reviewer. Manually check the code review for any recommendations. Choose the option to protect the secret. Update the SAM templates and the Python code to pull the secret from AWS Secrets Manager.

C.Enable Amazon CodeGuru Profiler. Decorate the handler function with @with_lambda_profiler(). Manually review the recommendation report. Choose the option to protect the secret. Update the SAM templates and the Python code to pull the secret from AWS Secrets Manager.

D.Associate the CodeCommit repository with Amazon CodeGuru Reviewer. Manually check the code review for any recommendations. Write the secret to AWS Systems Manager Parameter Store as a string. Update the SAM templates and the Python code to pull the secret from Parameter Store.

Answer: B

Explanation:

The correct answer is B. Here's why:

Security Best Practices: Hardcoding secrets in source code is a severe security vulnerability. It exposes sensitive credentials to anyone with access to the repository, potentially leading to unauthorized access and data breaches. Therefore, automatically detecting and preventing hardcoded secrets is crucial.

CodeGuru Reviewer for Secret Detection: Amazon CodeGuru Reviewer is designed to automatically detect potential defects and hard-coded secrets within the codebase during code reviews. Associating the CodeCommit repository with CodeGuru Reviewer provides a continuous, automated check for such vulnerabilities.

Secrets Manager for Secure Storage: AWS Secrets Manager is the recommended service for securely storing and managing secrets like database credentials. It provides features such as encryption, access control, and automatic rotation to enhance security.

Parameter Store vs. Secrets Manager: While AWS Systems Manager Parameter Store can store secrets, Secrets Manager is specifically designed for this purpose and offers features like automatic rotation which Parameter Store lacks. Secrets Manager also integrates better with other AWS services for accessing secrets. Using "secure string" in Parameter Store provides encryption, but it doesn't offer the same level of management and integration capabilities as Secrets Manager.

CodeGuru Profiler is not relevant: Amazon CodeGuru Profiler helps optimize application performance by identifying bottlenecks and inefficiencies. While it's valuable for performance improvement, it does not have capabilities for detecting hardcoded secrets.

Automated vs. Manual Review: The question asks for the most secure solution to automatically detect and prevent hardcoded secrets. CodeGuru Reviewer provides automated checks, whereas relying solely on manual review (after CodeGuru Profiler's recommendation report) is less reliable and scalable.

Therefore, associating CodeCommit with CodeGuru Reviewer, using AWS Secrets Manager, and updating the code to pull secrets from Secrets Manager is the most secure and automated solution to meet the requirements.

Supporting Documentation:

AWS Secrets Manager:https://aws.amazon.com/secrets-manager/
Amazon CodeGuru Reviewer:https://aws.amazon.com/codeguru/
Best practices for storing secrets in AWS:https://aws.amazon.com/blogs/security/secrets-management-on-aws/

Question: 76

A company is using Amazon S3 buckets to store important documents. The company discovers that some S3 buckets are not encrypted. Currently, the company's IAM users can create new S3 buckets without encryption. The company is implementing a new requirement that all S3 buckets must be encrypted.

A DevOps engineer must implement a solution to ensure that server-side encryption is enabled on all existing S3 buckets and all new S3 buckets. The encryption must be enabled on new S3 buckets as soon as the S3 buckets are created. The default encryption type must be 256-bit Advanced Encryption Standard (AES-256).

Which solution will meet these requirements?

A.Create an AWS Lambda function that is invoked periodically by an Amazon EventBridge scheduled rule. Program the Lambda function to scan all current S3 buckets for encryption status and to set AES-256 as the default encryption for any S3 bucket that does not have an encryption configuration.

B.Set up and activate the s3-bucket-server-side-encryption-enabled AWS Config managed rule. Configure the rule to use the AWS-EnableS3BucketEncryption AWS Systems Manager Automation runbook as the remediation action. Manually run the re-evaluation process to ensure that existing S3 buckets are compliant.

C.Create an AWS Lambda function that is invoked by an Amazon EventBridge event rule. Define the rule with an event pattern that matches the creation of new S3 buckets. Program the Lambda function to parse the EventBridge event, check the configuration of the S3 buckets from the event, and set AES-256 as the default encryption.

D.Configure an IAM policy that denies the s3:CreateBucket action if the s3:x-amz-server-side-encryption condition key has a value that is not AES-256. Create an IAM group for all the company's IAM users. Associate the IAM policy with the IAM group.

Answer: B

Explanation:

The correct answer is **B** because it offers a comprehensive and automated solution for both existing and new S3 buckets.

Here's a breakdown of why:

AWS Config Managed Rule: The s3-bucket-server-side-encryption-enabled AWS Config managed rule continuously evaluates whether S3 buckets have server-side encryption enabled. This provides ongoing compliance monitoring. AWS Config is designed for this type of auditing and enforcement. https://docs.aws.amazon.com/config/latest/developerguide/managed-rules-by-aws.html

Remediation Action (Systems Manager Automation Runbook): The AWS-EnableS3BucketEncryption Systems Manager Automation runbook automatically remediates non-compliant buckets by enabling server-side encryption with the specified AES-256 algorithm. This automation streamlines the process of fixing non-compliant resources. https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-aws-enable-s3-bucket-encryption.html

Manual Re-evaluation: Manually re-evaluating the rule ensures that existing S3 buckets are assessed and remediated immediately upon implementing the solution.

Comprehensive Coverage: Config covers both existing and future buckets. The rule runs continuously and is triggered by bucket creation events.

Why other options are not ideal:

A: While a Lambda function can scan and update buckets, it requires custom coding, scheduling, and maintenance. It is not a native monitoring and remediation tool like AWS Config, making it a less robust solution. The Lambda function only triggers periodically, and buckets created between the checks would be unencrypted.

C: This option only addresses new S3 buckets. Existing buckets would remain unencrypted. It also requires custom coding and EventBridge event pattern configuration, which is more complex than using a managed rule.

D: This option prevents users from creating unencrypted buckets, but does not enforce AES-256 encryption, only that some encryption is specified. It doesn't address the existing buckets at all. It also relies on IAM users correctly specifying encryption during bucket creation, which is prone to human error or misconfiguration of the encryption settings.

Question: 77

A DevOps engineer is architecting a continuous development strategy for a company's software as a service (SaaS) web application running on AWS. For application and security reasons, users subscribing to this application are distributed across multiple Application Load Balancers (ALBs), each of which has a dedicated Auto Scaling group and fleet of Amazon EC2 instances. The application does not require a build stage, and when it is committed to AWS CodeCommit, the application must trigger a simultaneous deployment to all ALBs, Auto Scaling groups, and EC2 fleets.

Which architecture will meet these requirements with the LEAST amount of configuration?

A.Create a single AWS CodePipeline pipeline that deploys the application in parallel using unique AWS CodeDeploy applications and deployment groups created for each ALB-Auto Scaling group pair.

B.Create a single AWS CodePipeline pipeline that deploys the application using a single AWS CodeDeploy application and single deployment group.

C.Create a single AWS CodePipeline pipeline that deploys the application in parallel using a single AWS CodeDeploy application and unique deployment group for each ALB-Auto Scaling group pair.

D.Create an AWS CodePipeline pipeline for each ALB-Auto Scaling group pair that deploys the application using an AWS CodeDeploy application and deployment group created for the same ALB-Auto Scaling group pair.

Answer: C

Explanation:

The correct answer is C. Here's why:

Option C offers the most efficient and manageable solution by leveraging the parallel execution capabilities of AWS CodePipeline in conjunction with CodeDeploy deployment groups. The core requirement is a simultaneous deployment across all ALB-Auto Scaling group pairs with minimal configuration effort.

CodePipeline: A single CodePipeline is the most straightforward way to orchestrate the entire deployment process.

CodeDeploy Application: Using a single CodeDeploy application simplifies management. A CodeDeploy application is a container for deployment configurations.

CodeDeploy Deployment Groups: The key to parallel deployments lies in the use of unique deployment groups, one for each ALB-Auto Scaling group pair. Each deployment group specifies the target EC2 instances (those behind the specific ALB). CodePipeline can then execute these deployment groups in parallel, achieving the desired simultaneous deployment.

Option A is less desirable as it involves creating a CodeDeploy application for each ALB-Auto Scaling group, resulting in more administrative overhead without significant benefit.

Option B would not work as it would deploy to all Auto Scaling Groups sequentially using a single deployment group rather than simultaneously.

Option D is also inefficient because it requires creating multiple CodePipeline pipelines (one for each ALB-Auto Scaling group), dramatically increasing configuration and management complexity. The goal is to minimize configuration, which this option fails to meet.

In summary, option C optimally balances efficiency, maintainability, and scalability by using a single CodePipeline and CodeDeploy application with multiple deployment groups for parallel deployments to the distinct environments behind each ALB.

Supporting Concepts:

AWS CodePipeline: A fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. https://aws.amazon.com/codepipeline/ **AWS CodeDeploy:** A fully managed deployment service that automates software deployments to a variety of compute services such as Amazon EC2, AWS Fargate, AWS Lambda, and your on-premises servers.

https://aws.amazon.com/codedeploy/
Deployment Groups (CodeDeploy): A set of instances that are targeted for a deployment. A deployment group contains individually tagged instances, Amazon EC2 instances in Auto Scaling groups, or instances in an Amazon ECS cluster. https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-deployments.html

Question: 78

A company is hosting a static website from an Amazon S3 bucket. The website is available to customers at example.com. The company uses an Amazon Route 53 weighted routing policy with a TTL of 1 day. The company has decided to replace the existing static website with a dynamic web application. The dynamic web application uses an Application Load Balancer (ALB) in front of a fleet of Amazon EC2 instances.

On the day of production launch to customers, the company creates an additional Route 53 weighted DNS record entry that points to the ALB with a weight of 255 and a TTL of 1 hour. Two days later, a DevOps engineer notices that the previous static website is displayed sometimes when customers navigate to example.com.

How can the DevOps engineer ensure that the company serves only dynamic content for example.com?

A.Delete all objects, including previous versions, from the S3 bucket that contains the static website content.

B.Update the weighted DNS record entry that points to the S3 bucket. Apply a weight of 0. Specify the domain reset option to propagate changes immediately.

C.Configure webpage redirect requests on the S3 bucket with a hostname that redirects to the ALB.

D.Remove the weighted DNS record entry that points to the S3 bucket from the example.com hosted zone. Wait for DNS propagation to become complete.

Answer: D

Explanation:

The issue is that some users are still seeing the old static website content even after the dynamic application is launched with a Route 53 weighted record pointing to the ALB. This implies that the Route 53 DNS record pointing to the S3 bucket containing the old static website is still active.

Option A is incorrect because deleting the S3 bucket contents doesn't prevent DNS servers from caching and serving the old DNS record.

Option B is incorrect because while setting the weight to 0 would eventually direct all traffic to the ALB, the TTL of the old record would still cause some users to receive the cached S3 IP address for up to 1 day. The "domain reset option" is not a valid Route 53 feature to immediately propagate changes. DNS propagation depends on the TTL and resolver behavior.

Option C is incorrect. Configuring redirects on the S3 bucket would only redirect new requests to the S3 endpoint. Clients that already have the old DNS entry cached will still hit the S3 endpoint directly before a redirect could occur. The fundamental problem is the presence of an active DNS record pointing at the old content.

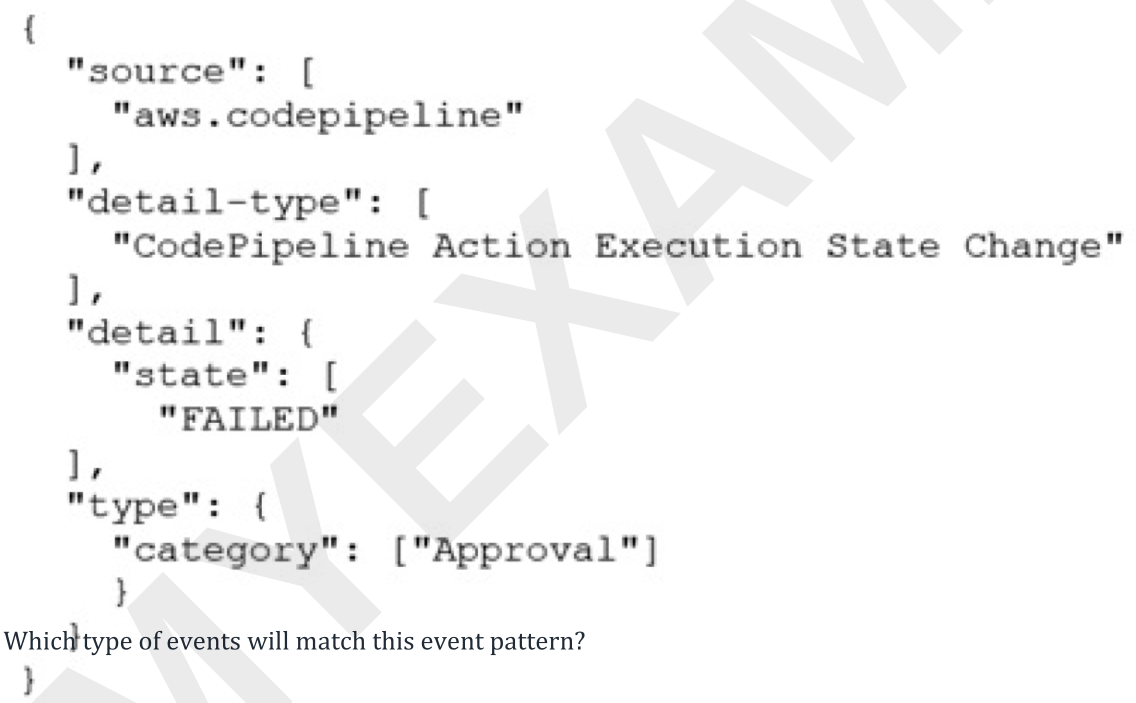
Option D is the correct solution. The most direct way to ensure that only the dynamic content is served is to remove the DNS record pointing to the S3 bucket. By deleting this record, you ensure that no further DNS requests for example.com will resolve to the S3 bucket's IP address. After removing the record, you should wait for the TTL of the previous record to expire to ensure complete propagation of the new DNS configuration. It's the most effective way to eliminate the old website from being served.

Relevant AWS documentation:

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-editing.html
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html

Question: 79

A company is implementing AWS CodePipeline to automate its testing process. The company wants to be notified when the execution state fails and used the following custom event pattern in Amazon EventBridge:



```
{
  "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
  ],
  "type": {
    "category": ["Approval"]
    }
  }
}
```

Which type of events will match this event pattern?

A.Failed deploy and build actions across all the pipelines

B.All rejected or failed approval actions across all the pipelines

C.All the events across all pipelines

D.Approval actions across all the pipelines

Answer: B

Explanation:

Reference:

https://docs.aws.amazon.com/ja_jp/codepipeline/latest/userguide/detect-state-changes-cloudwatch-events.html

Question: 80

An application running on a set of Amazon EC2 instances in an Auto Scaling group requires a configuration file to operate. The instances are created and maintained with AWS CloudFormation. A DevOps engineer wants the instances to have the latest configuration file when launched, and wants changes to the configuration file to be reflected on all the instances with a minimal delay when the CloudFormation template is updated. Company policy requires that application configuration files be maintained along with AWS infrastructure configuration files in source control.

Which solution will accomplish this?

A.In the CloudFormation template, add an AWS Config rule. Place the configuration file content in the rule's InputParameters property, and set the Scope property to the EC2 Auto Scaling group. Add an AWS Systems Manager Resource Data Sync resource to the template to poll for updates to the configuration.

B.In the CloudFormation template, add an EC2 launch template resource. Place the configuration file content in the launch template. Configure the cfn-init script to run when the instance is launched, and configure the cfn-hup script to poll for updates to the configuration.

C.In the CloudFormation template, add an EC2 launch template resource. Place the configuration file content in the launch template. Add an AWS Systems Manager Resource Data Sync resource to the template to poll for updates to the configuration.

D.In the CloudFormation template, add CloudFormation init metadata. Place the configuration file content in the metadata. Configure the cfn-init script to run when the instance is launched, and configure the cfn-hup script to poll for updates to the configuration.

Answer: D

Explanation:

Here's a detailed justification for why option D is the correct answer and why the other options are less suitable, along with supporting concepts and links.

Justification for Option D: CloudFormation init Metadata with cfn-init and cfn-hup

Option D leverages CloudFormation's built-in capabilities to manage instance configuration. CloudFormation init metadata allows you to specify configuration actions within your CloudFormation template, directly alongside your infrastructure definition. This aligns with the company policy of keeping application and infrastructure configurations together in source control. The `cfn-init` script, when run during instance launch, processes this metadata to perform tasks like creating files, installing packages, and starting services. Crucially, `cfn-hup` monitors for changes to the CloudFormation stack metadata. When an update is detected (i.e., when the configuration file content changes in the template), `cfn-hup` triggers a helper script (often a simple shell script) that can re-run `cfn-init` or apply specific configuration updates. This ensures near real-time reflection of configuration changes across all instances with minimal delay upon template updates. This approach supports idempotency, ensuring desired state.

Why other options are incorrect:

Option A (AWS Config Rule): AWS Config is primarily for evaluating resource configurations against desired rules. While you could technically store a configuration file within an AWS Config rule's InputParameters, it's an inefficient and less direct approach for deploying configuration files. AWS Config's main function is compliance checking, not application configuration distribution. Also, using Resource Data Sync to poll for updates would be an unusual application of this service and add unnecessary complexity. The primary use case for Resource Data Sync is centralizing data from multiple AWS accounts or Regions for Systems Manager.

Option B (EC2 Launch Template with cfn-init/cfn-hup): Launch templates are definitely a valid approach for specifying instance configuration, and using cfn-init and cfn-hup alongside them is conceptually sound. The problem is the prompt states the desired solution must allow maintaining application configuration files along with AWS infrastructure configuration files in source control. Launch Templates, while versioned, don't as easily integrate and enable version control from within a CloudFormation template.

Option C (EC2 Launch Template with Resource Data Sync): This option suffers from similar issues as Option B, with the additional misapplication of Resource Data Sync (as described in A). Launch Templates, while valid, don't as easily enable version control from within a CloudFormation template. Resource Data Sync doesn't fit for updating EC2 instance files from CloudFormation.

Supporting Concepts and Links:

AWS CloudFormation Init:https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-init.html - This documentation provides a comprehensive overview of CloudFormation init metadata and the helper scripts (cfn-init, cfn-hup, cfn-signal).

AWS CloudFormation Helper Scripts:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-helper-scripts-reference.html -Details on how to use the CloudFormation helper scripts, including configuration and usage examples.

AWS Systems Manager Resource Data Sync:https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-inventory-datasync.html - Explains the intended purpose and use cases for Resource Data Sync, which is primarily for inventory management and data aggregation.

Launch Templates:https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/launch-templates.html - Details on Launch Templates.

AWS Config Rules:https://docs.aws.amazon.com/config/latest/developerguide/evaluate-config.html - Details on Config Rules.

In summary, option D directly addresses the requirements by integrating configuration management into the CloudFormation template, ensuring version control and efficient deployment/updates using cfn-init and cfn-hup.