Free Questions for DOP-C02

Shared by Delaney on 04-10-2024

Question 1

Question Type: MultipleChoice

A company releases a new application in a new AWS account. The application includes an AWS Lambda function that processes messages from an Amazon Simple Queue Service (Amazon SOS) standard queue. The Lambda function stores the results in an Amazon S3 bucket for further downstream processing. The Lambda function needs to process the messages within a specific period of time after the messages are published. The Lambda function has a batch size of 10 messages and takes a few seconds to process a batch of messages.

As load increases on the application's first day of service, messages in the queue accumulate at a greater rate than the Lambda function can process the messages. Some messages miss the required processing timelines. The logs show that many messages in the queue have data that is not valid. The company needs to meet the timeline requirements for messages that have valid data.

Which solution will meet these requirements?

Options:

A- Increase the Lambda function's batch size. Change the SOS standard queue to an SOS FIFO queue. Request a Lambda concurrency increase in the AWS Region.

B- Reduce the Lambda function's batch size. Increase the SOS message throughput quota. Request a Lambda concurrency increase in the AWS Region.

C- Increase the Lambda function's batch size. Configure S3 Transfer Acceleration on the S3 bucket. Configure an SOS dead-letter queue.

D- Keep the Lambda function's batch size the same. Configure the Lambda function to report failed batch items. Configure an SOS dead-letter queue.
Step 1: Handling Invalid Data with Failed Batch Items
The Lambda function is processing batches of messages, and some messages contain invalid data, causing processing delays. Lambda provides the capability to report failed batch items, which allows valid messages to be processed while skipping invalid ones. This functionality ensures that the valid messages are processed within the required timeline.
Action: Keep the Lambda function's batch size the same and configure it to report failed batch items.
Why: By reporting failed batch items, the Lambda function can skip invalid messages and continue processing valid ones, ensuring that they meet the processing timeline.

Answer:

D

Explanation:

Step 2: Using an SQS Dead-Letter Queue (DLQ) Configuring a dead-letter queue (DLQ) for SQS will ensure that messages with invalid data, or those that cannot be processed successfully, are moved to the DLQ. This prevents such messages from clogging the queue and allows the system to focus on processing valid messages.

Action: Configure an SQS dead-letter queue for the main queue.

Why: A DLQ helps isolate problematic messages, preventing them from continuously reappearing in the queue and causing processing delays for valid messages.

Step 3: Maintaining the Lambda Function's Batch Size Keeping the current batch size allows the Lambda function to continue processing multiple messages at once. By addressing the failed items separately, there's no need to increase or reduce the batch size.

Action: Maintain the Lambda function's current batch size.

Why: Changing the batch size is unnecessary if the invalid messages are properly handled by reporting failed items and using a DLQ.

This corresponds to Option D: Keep the Lambda function's batch size the same. Configure the Lambda function to report failed batch items. Configure an SQS dead-letter queue.

Question 2

Question Type: MultipleChoice

A company has an organization in AWS Organizations for its multi-account environment. A DevOps engineer is developing an AWS CodeArtifact based strategy for application package management across the organization. Each application team at the company has its own account in the organization. Each application team also has limited access to a centralized shared services account.

Each application team needs full access to download, publish, and grant access to its own packages. Some common library packages that the application teams use must also be shared with the entire organization.

Which combination of steps will meet these requirements with the LEAST administrative overhead? (Select THREE.)

Options:

A- Create a domain in each application team's account. Grant each application team's account lull read access and write access to the application team's domain

B- Create a domain in the shared services account Grant the organization read access and CreateRepository access.

C- Create a repository in each application team's account. Grant each application team's account lull read access and write access to its own repository.

D- Create a repository in the shared services account. Grant the organization read access to the repository in the shared services account. Set the repository as the upstream repository in each application team's repository.

E- For teams that require shared packages, create resource-based policies that allow read access to the repository from other application teams' accounts.

F- Set the other application teams' repositories as upstream repositories.
* Step 1: Creating a Centralized Domain in the Shared Services Account
To manage application package dependencies across multiple accounts, the most efficient solution is to create a centralized domain in the shared services account. This allows all application teams to access and manage package repositories within the same domain, ensuring consistency and centralization.
Action: Create a domain in the shared services account.
Why: A single, centralized domain reduces the need for redundant management in each application team's account.

Answer:

B, D, E

Explanation:

This corresponds to Option B: Create a domain in the shared services account. Grant the organization read access and CreateRepository access.

* Step 2: Sharing Repositories Across Teams with Upstream Configurations To share common library packages across the organization, each application team's repository can point to the shared services repository as an upstream repository. This enables teams to access shared packages without managing them individually in each team's account.

Action: Create a repository in the shared services account and set it as the upstream repository for each application team.

Why: Upstream repositories allow package sharing while maintaining individual team repositories for managing their own packages.

This corresponds to Option D: Create a repository in the shared services account. Grant the organization read access to the repository in the shared services account. Set the repository as the upstream repository in each application team's repository.

* Step 3: Using Resource-Based Policies for Cross-Account Access For teams that need to share their packages with other application teams, resource-based policies can be applied to grant the necessary permissions. These policies allow cross-account access without having to manage permissions at the individual account level.

Action: Create resource-based policies that allow read access to the repositories across application teams.

Why: This simplifies management by centralizing permissions in the shared services account while allowing cross-team collaboration.

This corresponds to Option E: For teams that require shared packages, create resource-based policies that allow read access to the repository from other application teams' accounts.

Question 3

Question Type: **MultipleChoice**

A company uses an AWS CodeCommit repository to store its source code and corresponding unit tests. The company has configured an AWS CodePipeline pipeline that includes an AWS CodeBuild project that runs when code is merged to the main branch of the repository.

The company wants the CodeBuild project to run the unit tests. If the unit tests pass, the CodeBuild project must tag the most recent commit.

How should the company configure the CodeBuild project to meet these requirements?

Options:

A- Configure the CodeBuild project to use native Git to clone the CodeCommit repository. Configure the project to run the unit tests. Configure the project to use native Git to create a tag and to push the Git tag to the repository if the code passes the unit tests.

B- Configure the CodeBuild project to use native Git to clone the CodeCommit repository. Configure the project to run the unit tests. Configure the project to use AWS CLI commands to create a new repository tag in the repository if the code passes the unit tests.

C- Configure the CodeBuild project to use AWS CLI commands to copy the code from the CodeCommit repository. Configure the project lo run the unit tests. Configure the project to use AWS CLI commands to create a new Git tag in the repository if the code passes the unit tests.

D- Configure the CodeBuild project to use AWS CLI commands to copy the code from the CodeCommit repository. Configure the project to run the unit tests. Configure the project to use AWS CLI commands to create a new repository tag in the repository if the code passes the unit tests.
Step 1: Using Native Git in CodeBuild
To meet the requirement of running unit tests and tagging the most recent commit if the tests pass, the CodeBuild project should be configured to use native Git to clone the CodeCommit repository. Native Git support allows full functionality for managing the repository, including the ability to create and push tags.
Action: Configure the CodeBuild project to use native Git to clone the repository and run the tests.
Why: Using native Git provides flexibility for managing tags and other repository operations after the tests are successfully executed.
Step 2: Tagging the Most Recent Commit
Once the unit tests pass, the CodeBuild project can use native Git to create a tag for the most recent commit and push that tag to the repository. This ensures that the tagged commit is linked to the test results.
Action: Configure the project to use native Git to create and push a tag to the repository if the tests pass.
Why: This ensures the correct commit is tagged automatically, streamlining the workflow.

Answer:

A

Explanation:

This corresponds to Option A: Configure the CodeBuild project to use native Git to clone the CodeCommit repository. Configure the project to run the unit tests. Configure the project to use native Git to create a tag and to push the Git tag to the repository if the code passes the unit tests.

Question 4

Question Type: MultipleChoice

A company has developed a static website hosted on an Amazon S3 bucket. The website is deployed using AWS CloudFormation. The CloudFormation template defines an S3 bucket and a custom resource that copies content into the bucket from a source location.

The company has decided that it needs to move the website to a new location, so the existing CloudFormation stack must be deleted and re-created. However, CloudFormation reports that the stack could not be deleted cleanly.

What is the MOST likely cause and how can the DevOps engineer mitigate this problem for this and future versions of the website?

Options:

A- Deletion has failed because the S3 bucket has an active website configuration. Modify the Cloud Formation template to remove the WebsiteConfiguration properly from the S3 bucket resource.

B- Deletion has failed because the S3 bucket is not empty. Modify the custom resource's AWS Lambda function code to recursively empty the bucket when RequestType is Delete.

C- Deletion has failed because the custom resource does not define a deletion policy. Add a DeletionPolicy property to the custom resource definition with a value of RemoveOnDeletion.

D- Deletion has failed because the S3 bucket is not empty. Modify the S3 bucket resource in the CloudFormation template to add a

DeletionPolicy property with a value of Empty.
Step 1: Understanding the Deletion Failure
The most likely reason why the CloudFormation stack failed to delete is that the S3 bucket was not empty. AWS CloudFormation cannot delete an S3 bucket that contains objects, so if the website files are still in the bucket, the deletion will fail.
Issue: The S3 bucket is not empty during deletion, preventing the stack from being deleted.
Step 2: Modifying the Custom Resource to Handle Deletion
To mitigate this issue, you can modify the Lambda function associated with the custom resource to automatically empty the S3 bucket when the stack is being deleted. By adding logic to handle the RequestType: Delete event, the function can recursively delete all objects in the bucket before allowing the stack to be deleted.
Action: Modify the Lambda function to recursively delete the objects in the S3 bucket when RequestType is set to Delete.
Why: This ensures that the S3 bucket is empty before CloudFormation tries to delete it, preventing the stack deletion failure.

Answer:

B

Explanation:

This corresponds to Option B: Deletion has failed because the S3 bucket is not empty. Modify the custom resource's AWS Lambda function code to recursively empty the bucket when RequestType is Delete.

Question 5

Question Type: **MultipleChoice**

A company recently migrated its application to an Amazon Elastic Kubernetes Service (Amazon EKS) cluster that uses Amazon EC2 instances. The company configured the application to automatically scale based on CPU utilization.

The application produces memory errors when it experiences heavy loads. The application also does not scale out enough to handle the increased load. The company needs to collect and analyze memory metrics for the application over time.

Which combination of steps will meet these requirements? (Select THREE.)

Options:

A- Attach the Cloud WatchAgentServer Pol icy managed 1AM policy to the 1AM instance profile that the cluster uses.

B- Attach the Cloud WatchAgentServer Pol icy managed 1AM policy to a service account role for the cluster.

C- Collect performance metrics by deploying the unified Amazon CloudWatch agent to the existing EC2 instances in the cluster. Add the agent to the AMI for any new EC2 instances that are added to the cluster.

D- Collect performance logs by deploying the AWS Distro for OpenTelemetry collector as a DaemonSet.

E- Analyze the pod_memory_utilization Amazon CloudWatch metric in the ContainerInsights namespace by using the Service dimension.

F- Analyze the node_memory_utilization Amazon CloudWatch metric in the ContainerInsights namespace by using the ClusterName dimension.

* Step 1: Attaching the CloudWatchAgentServerPolicy to the IAM Role
The CloudWatch agent needs permissions to collect and send metrics, including memory metrics, to Amazon CloudWatch. You can attach the CloudWatchAgentServerPolicy managed IAM policy to the IAM instance profile or service account role to grant these

permissions.
Action: Attach the CloudWatchAgentServerPolicy managed IAM policy to the IAM instance profile that the EKS cluster uses.
Why: This ensures the CloudWatch agent has the necessary permissions to collect memory metrics.

Answer:

A, C, E

Explanation:

This corresponds to Option A: Attach the CloudWatchAgentServerPolicy managed IAM policy to the IAM instance profile that the cluster uses.

* Step 2: Deploying the CloudWatch Agent to EC2 Instances To collect memory metrics from the EC2 instances running in the EKS cluster, the CloudWatch agent needs to be deployed on these instances. The agent collects system-level metrics, including memory usage.

Action: Deploy the unified Amazon CloudWatch agent to the existing EC2 instances in the EKS cluster. Update the Amazon Machine Image (AMI) for future instances to include the CloudWatch agent.

Why: The CloudWatch agent allows you to collect detailed memory metrics from the EC2 instances, which is not enabled by default.

This corresponds to Option C: Collect performance metrics by deploying the unified Amazon CloudWatch agent to the existing EC2 instances in the cluster. Add the agent to the AMI for any new EC2 instances that are added to the cluster.

* Step 3: Analyzing Memory Metrics Using Container Insights After collecting the memory metrics, you can analyze them using the pod_memory_utilization metric in Amazon CloudWatch Container Insights. This metric provides visibility into the memory usage of the containers (pods) in the EKS cluster.

Action: Analyze the pod_memory_utilization CloudWatch metric in the Container Insights namespace by using the Service dimension.

Why: This provides detailed insights into memory usage at the container level, which helps diagnose memory-related issues.

This corresponds to Option E: Analyze the pod_memory_utilization Amazon CloudWatch metric in the Container Insights namespace by using the Service dimension.

Question 6

Question Type: **MultipleChoice**

A company uses AWS Organizations to manage its AWS accounts. A DevOps engineer must ensure that all users who access the AWS Management Console are authenticated through the company's corporate identity provider (IdP).

Which combination of steps will meet these requirements? (Select TWO.)

Options:

A- Use Amazon GuardDuty with a delegated administrator account. Use GuardDuty to enforce denial of 1AM user logins

B- Use AWS 1AM Identity Center to configure identity federation with SAML 2.0.

C- Create a permissions boundary in AWS 1AM Identity Center to deny password logins for 1AM users.

D- Create 1AM groups in the Organizations management account to apply consistent permissions for all 1AM users.

E- Create an SCP in Organizations to deny password creation for 1AM users.
* Step 1: Using AWS IAM Identity Center for SAML-based Identity Federation
To ensure that all users accessing the AWS Management Console are authenticated via the corporate identity provider (IdP), the best approach is to set up identity federation with AWS IAM Identity Center (formerly AWS SSO) using SAML 2.0.
Action: Use AWS IAM Identity Center to configure identity federation with the corporate IdP that supports SAML 2.0.
Why: SAML 2.0 integration enables single sign-on (SSO) for users, allowing them to authenticate through the corporate IdP and gain access to AWS resources.

Answer:

B, E

Explanation:

This corresponds to Option B: Use AWS IAM Identity Center to configure identity federation with SAML 2.0.

* Step 2: Creating an SCP to Deny Password Logins for IAM Users To enforce that IAM users do not create passwords or access the Management Console directly without going through the corporate IdP, you can create a Service Control Policy (SCP) in AWS

Organizations that denies password creation for IAM users.

Action: Create an SCP that denies password creation for IAM users.

Why: This ensures that users cannot set passwords for their IAM user accounts, forcing them to use federated access through the corporate IdP for console login.

This corresponds to Option E: Create an SCP in Organizations to deny password creation for IAM users.

Question 7

Question Type: MultipleChoice

A company uses AWS WAF to protect its cloud infrastructure. A DevOps engineer needs to give an operations team the ability to analyze log messages from AWS WAR. The operations team needs to be able to create alarms for specific patterns in the log output.

Which solution will meet these requirements with the LEAST operational overhead?

Options:

A- Create an Amazon CloudWatch Logs log group. Configure the appropriate AWS WAF web ACL to send log messages to the log group. Instruct the operations team to create CloudWatch metric filters.

B- Create an Amazon OpenSearch Service cluster and appropriate indexes. Configure an Amazon Kinesis Data Firehose delivery stream to stream log data to the indexes. Use OpenSearch Dashboards to create filters and widgets.

C- Create an Amazon S3 bucket for the log output. Configure AWS WAF to send log outputs to the S3 bucket. Instruct the operations team to create AWS Lambda functions that detect each desired log message pattern. Configure the Lambda functions to publish to an Amazon Simple Notification Service (Amazon SNS) topic.

D- Create an Amazon S3 bucket for the log output. Configure AWS WAF to send log outputs to the S3 bucket. Use Amazon Athena to create an external table definition that fits the log message pattern. Instruct the operations team to write SOL queries and to create Amazon CloudWatch metric filters for the Athena queries.
Step 1: Sending AWS WAF Logs to CloudWatch Logs
AWS WAF allows you to log requests that are evaluated against your web ACLs. These logs can be sent directly to CloudWatch Logs, which enables real-time monitoring and analysis.
Action: Configure the AWS WAF web ACL to send log messages to a CloudWatch Logs log group.
Why: This allows the operations team to view the logs in real time and analyze patterns using CloudWatch metric filters.

Answer:

A

Explanation:

Step 2: Creating CloudWatch Metric Filters CloudWatch metric filters can be used to search for specific patterns in log data. The operations team can create filters for certain log patterns and set up alarms based on these filters.

Action: Instruct the operations team to create CloudWatch metric filters to detect patterns in the WAF log output.

Why: Metric filters allow the team to trigger alarms based on specific patterns without needing to manually search through logs.

This corresponds to Option A: Create an Amazon CloudWatch Logs log group. Configure the appropriate AWS WAF web ACL to send log messages to the log group. Instruct the operations team to create CloudWatch metric filters.

Question 8

Question Type: MultipleChoice

A company uses an Amazon Aurora PostgreSQL global database that has two secondary AWS Regions. A DevOps engineer has configured the database parameter group to guarantee an RPO of 60 seconds. Write operations on the primary cluster are occasionally blocked because of the RPO setting.

The DevOps engineer needs to reduce the frequency of blocked write operations.

Which solution will meet these requirements?

Options:

A- Add an additional secondary cluster to the global database.

B- Enable write forwarding for the global database.

C- Remove one of the secondary clusters from the global database.

D- Configure synchronous replication for the global database.
Step 1: Reducing Replication Lag in Aurora Global Databases
In Amazon Aurora global databases, write operations on the primary cluster can be delayed due to the time it takes to replicate to secondary clusters, especially when there are multiple secondary regions involved.
Issue: The write operations are occasionally blocked due to the RPO setting, which guarantees replication within 60 seconds.
Action: Remove one of the secondary clusters from the global database.
Why: Fewer secondary clusters will reduce the overall replication lag, improving write performance and reducing the frequency of blocked writes.

Answer:

C

Explanation:

This corresponds to Option C: Remove one of the secondary clusters from the global database.

Question 9

Question Type: MultipleChoice

A DevOps engineer uses AWS CodeBuild to frequently produce software packages. The CodeBuild project builds large Docker images that the DevOps engineer can use across multiple builds. The DevOps engineer wants to improve build performance and minimize costs. Which solution will meet these requirements?

Options:

A- Store the Docker images in an Amazon Elastic Container Registry (Amazon ECR) repository. Implement a local Docker layer cache for CodeBuild.

B- Cache the Docker images in an Amazon S3 bucket that is available across multiple build hosts. Expire the cache by using an S3 Lifecycle policy.

C- Store the Docker images in an Amazon Elastic Container Registry (Amazon ECR) repository. Modify the CodeBuild project runtime configuration to always use the most recent image version.

D- Create custom AMIs that contain the cached Docker images. In the CodeBuild build, launch Amazon EC2 instances from the custom AMIs.

Step 1: Storing Docker Images in Amazon ECR
Docker images can be large, and storing them in a centralized, scalable location can greatly reduce build times. Amazon Elastic Container Registry (ECR) is a fully managed container registry that stores, manages, and deploys Docker container images.
Action: Store the Docker images in an ECR repository.
Why: Storing Docker images in ECR ensures that Docker images can be reused across multiple builds, improving build performance by avoiding the need to rebuild the images from scratch.

Answer:

A

Explanation:

Step 2: Implementing Docker Layer Caching in CodeBuild Docker layer caching is essential for improving performance in continuous integration pipelines. CodeBuild supports local caching of Docker layers, which speeds up builds that reuse Docker images across multiple runs.

Action: Implement Docker layer caching within the CodeBuild project.

Why: This improves performance by allowing frequently used Docker layers to be cached locally, avoiding the need to pull or build the layers every time.

This corresponds to Option A: Store the Docker images in an Amazon Elastic Container Registry (Amazon ECR) repository. Implement a local Docker layer cache for CodeBuild.

Question 10

Question Type: MultipleChoice

A company's organization in AWS Organizations has a single OU. The company runs Amazon EC2 instances in the OU accounts. The company needs to limit the use of each EC2 instance's credentials to the specific EC2 instance that the credential is assigned to. A DevOps engineer must configure security for the EC2 instances.

Which solution will meet these requirements?

Options:

A- Create an SCP that specifies the VPC CIDR block. Configure the SCP to check whether the value of the aws:VpcSourceIp condition key is in the specified block. In the same SCP check, check whether the values of the aws:EC2InstanceSourcePrivateIPv4 and aws:SourceVpc condition keys are the same. Deny access if either condition is false. Apply the SCP to the OU.

B- Create an SCP that checks whether the values of the aws:EC2InstanceSourceVPC and aws:SourceVpc condition keys are the same. Deny access if the values are not the same. In the same SCP check, check whether the values of the aws:EC2InstanceSourcePrivateIPv4 and awsVpcSourceIp condition keys are the same. Deny access if the values are not the same. Apply the SCP to the OU.

C- Create an SCP that includes a list of acceptable VPC values and checks whether the value of the aws:SourceVpc condition key is in the list. In the same SCP check, define a list of acceptable IP address values and check whether the value of the aws:VpcSourceIp condition key is in the list. Deny access if either condition is false. Apply the SCP to each account in the organization.

D- Create an SCP that checks whether the values of the aws:EC2InstanceSourceVPC and aws:VpcSourceIp condition keys are the same. Deny access if the values are not the same. In the same SCP check, check whether the values of the aws:EC2InstanceSourcePrivatoIPv4 and aws:SourceVpc condition keys are the same. Deny access if the values are not the same. Apply the SCP to each account in the organization.
Step 1: Using Service Control Policies (SCPs) for EC2 Security

To limit the use of EC2 instance credentials to the specific EC2 instance they are assigned to, you can create a Service Control Policy (SCP) that verifies specific conditions, such as whether the EC2 instance's source VPC and private IP match expected values.
Action: Create an SCP that checks whether the values of the aws:EC2InstanceSourceVPC and aws:SourceVpc condition keys are the same. Deny access if they are not.
Why: This ensures that credentials cannot be used outside the designated EC2 instance or VPC.
Step 2: Further Validation with Private IPs
The SCP should also verify that the EC2 instance's private IP matches the IP range specified for the VPC. If the instance's private IP does not match, access should be denied.
Action: In the same SCP, check whether the values of the aws:EC2InstanceSourcePrivateIP and aws:VpcSourceIP condition keys are the same. Deny access if they are not.
Why: This ensures that the credentials are only used within the specific EC2 instance and its associated VPC.

Answer:

B

Explanation:

This corresponds to Option B: Create an SCP that checks whether the values of the aws:EC2InstanceSourceVPC and aws:SourceVpc condition keys are the same. Deny access if the values are not the same. In the same SCP check, check whether the values of the aws:EC2InstanceSourcePrivateIP and aws:VpcSourceIP condition keys are the same. Deny access if the values are not the same. Apply the SCP to the OU.

Question 11

Question Type: **MultipleChoice**

A company has an AWS Cloud Format ion slack that is deployed in a single AWS account. The company has configured the stack to send event notifications to an Amazon Simple Notification Service (Amazon SNS) topic.

A DevOps engineer must implement an automated solution that applies a tag to the specific Cloud Formation stack instance only after a successful stack update occurs. The DevOps engineer has created an AWS Lambda function that applies and updates this tag (or the specific slack instance.

Which solution will meet these requirements?

Options:

A- Run the AWS-UpdateCloudfomationStack AWS Systems Manager Automation runbook when Systems Manager detects an UPDATE_COMPLETE event for the instance status of the Cloud Formation stack. Configure the runbook to invoke the Lambda function.

B- Create a custom AWS Config rule that produces a compliance change event if the CloudFormation stack has an UPDATE_COMPLETE instance status. Configure AWS Config to directly invoke the Lambda function to automatically remediate the change event.

C- Create an Amazon EventBridge rule that matches the UPDATE COMPLETE event pattern for the instance status of the CloudFormation stack. Configure the rule to invoke the Lambda function.

D- Adjust the configuration of the CloudFormation stack to send notifications for only an UPDATE COMPLETE instance status event to

the SNS topic. Subscribe the Lambda function to the SNS topic.
Step 1: Reacting to CloudFormation Stack Events with EventBridge
AWS CloudFormation emits events during the lifecycle of a stack, including the UPDATE_COMPLETE event after a successful stack update. You can use Amazon EventBridge to detect this event and trigger a specific action (such as invoking a Lambda function).
Action: Create an EventBridge rule that listens for the UPDATE_COMPLETE event for the CloudFormation stack.
Why: EventBridge allows you to automatically detect CloudFormation stack lifecycle events and take action based on them.
Step 2: Invoking the Lambda Function
After the EventBridge rule detects the UPDATE_COMPLETE event, it can invoke the pre-configured Lambda function to apply or update the tag on the specific CloudFormation stack instance.
Action: Configure the EventBridge rule to invoke the Lambda function when the UPDATE_COMPLETE event occurs.
Why: This automation ensures that the tag is applied immediately after a successful stack update without any manual intervention.

Answer:

C

Explanation:

This corresponds to Option C: Create an Amazon EventBridge rule that matches the UPDATE_COMPLETE event pattern for the instance status of the CloudFormation stack. Configure the rule to invoke the Lambda function.