Free Questions for DOP-C02 by actualtestdumps

Shared by Simmons on 15-04-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

A company runs a web application that extends across multiple Availability Zones. The company uses an Application Load Balancer (ALB) for routing. AWS Fargate (or the application and Amazon Aurora for the application data The company uses AWS CloudFormation templates to deploy the application The company stores all Docker images in an Amazon Elastic Container Registry (Amazon ECR) repository in the same AWS account and AWS Region.

A DevOps engineer needs to establish a disaster recovery (DR) process in another Region. The solution must meet an RPO of 8 hours and an RTO of 2 hours The company sometimes needs more than 2 hours to build the Docker images from the Dockerfile

Which solution will meet the RTO and RPO requirements MOST cost-effectively?

Options:

A- Copy the CloudFormation templates and the Dockerfile to an Amazon S3 bucket in the DR Region Use AWS Backup to configure automated Aurora cross-Region hourly snapshots In case of DR, build the most recent Docker image and upload the Docker image to an ECR repository in the DR Region Use the CloudFormation template that has the most recent Aurora snapshot and the Docker image from the ECR repository to launch a new CloudFormation stack in the DR Region Update the application DNS records to point to the new ALB

B- Copy the CloudFormation templates to an Amazon S3 bucket in the DR Region Configure Aurora automated backup Cross-Region Replication Configure ECR Cross-Region Replication. In case of DR use the CloudFormation template with the most recent Aurora

snapshot and the Docker image from the local ECR repository to launch a new CloudFormation stack in the DR Region Update the application DNS records to point to the new ALB

C- Copy the CloudFormation templates to an Amazon S3 bucket in the DR Region. Use Amazon EventBridge to schedule an AWS Lambda function to take an hourly snapshot of the Aurora database and of the most recent Docker image in the ECR repository. Copy the snapshot and the Docker image to the DR Region in case of DR, use the CloudFormation template with the most recent Aurora snapshot and the Docker image from the local ECR repository to launch a new CloudFormation stack in the DR Region

D- Copy the CloudFormation templates to an Amazon S3 bucket in the DR Region. Deploy a second application CloudFormation stack in the DR Region. Reconfigure Aurora to be a global database Update both CloudFormation stacks when a new application release in the current Region is needed. In case of DR. update, the application DNS records to point to the new ALB.

Answer:

B

Explanation:

The most cost-effective solution to meet the RTO and RPO requirements is option B. This option involves copying the CloudFormation templates to an Amazon S3 bucket in the DR Region, configuring Aurora automated backup Cross-Region Replication, and configuring ECR Cross-Region Replication. In the event of a disaster, the CloudFormation template with the most recent Aurora snapshot and the Docker image from the local ECR repository can be used to launch a new CloudFormation stack in the DR Region. This approach avoids the need to build Docker images from the Dockerfile, which can sometimes take more than 2 hours, thus meeting the RTO requirement. Additionally, the use of automated backups and replication ensures that the RPO of 8 hours is met.

AWS Documentation on Disaster Recovery:Plan for Disaster Recovery (DR) - Reliability Pillar

AWS Blog on Establishing RPO and RTO Targets:Establishing RPO and RTO Targets for Cloud Applications

AWS Documentation on ECR Cross-Region Replication: Amazon ECR Cross-Region Replication

AWS Documentation on Aurora Cross-Region Replication: Replicating Amazon Aurora DB Clusters Across AWS Regions

Question 2

Question Type: **MultipleChoice**

A DevOps engineer wants to find a solution to migrate an application from on premises to AWS The application is running on Linux and needs to run on specific versions of Apache Tomcat HAProxy and Varnish Cache to function properly. The application's operating system-level parameters require tuning The solution must include a way to automate the deployment of new application versions. The infrastructure should be scalable and faulty servers should be replaced automatically.

Which solution should the DevOps engineer use?

Options:

A- Upload the application as a Docker image that contains all the necessary software to Amazon ECR Create an Amazon ECS cluster using an AWS Fargate launch type and an Auto Scaling group. Create an AWS CodePipeline pipeline that uses Amazon ECR as a

source and Amazon ECS as a deployment provider

B- Upload the application code to an AWS CodeCommit repository with a saved configuration file to configure and install the software Create an AWS Elastic Beanstalk web server tier and a load balanced-type environment that uses the Tomcat solution stack Create an AWS CodePipeline pipeline that uses CodeCommit as a source and Elastic Beanstalk as a deployment provider

C- Upload the application code to an AWS CodeCommit repository with a set of ebextensions files to configure and install the software. Create an AWS Elastic Beanstalk worker tier environment that uses the Tomcat solution stack Create an AWS CodePipeline pipeline that uses CodeCommit as a source and Elastic Beanstalk as a deployment provider

D- Upload the application code to an AWS CodeCommit repository with an appspec.yml file to configure and install the necessary software. Create an AWS CodeDeploy deployment group associated with an Amazon EC2 Auto Scaling group Create an AWS CodePipeline pipeline that uses CodeCommit as a source and CodeDeploy as a deployment provider

Answer:

D

Explanation:

The correct answer is D. The scenario requires a solution that can migrate an application from on premises to AWS, run on specific versions of Apache Tomcat, HAProxy, and Varnish Cache, tune the operating system-level parameters, automate the deployment of new application versions, and scale and replace faulty servers automatically. Option D meets all these requirements by using AWS CodeCommit, AWS CodeDeploy, AWS CodePipeline, and Amazon EC2 Auto Scaling. AWS CodeCommit is a fully managed source control service that hosts Git repositories and works with Git-based tools. AWS CodeDeploy is a fully managed deployment service that automates software deployments to a variety of compute services, including Amazon EC2, AWS Fargate, AWS Lambda, and on-

premises servers. AWS CodePipeline is a fully managed continuous delivery service that helps automate the release pipelines for fast and reliable application updates. Amazon EC2 Auto Scaling helps maintain application availability and allows scaling of Amazon EC2 capacity up or down automatically according to the defined conditions. By using these services together, the DevOps engineer can migrate the application code to AWS, configure and install the necessary software using the appspec.yml file, automate the deployment process using the pipeline, and scale and replace the servers using the Auto Scaling group.

Option A is incorrect because AWS Fargate is a serverless compute engine for containers that works with Amazon ECS and Amazon EKS. Fargate removes the need to provision and manage servers, but it also limits the ability to tune the operating system-level parameters, which is a requirement in the scenario. Moreover, Fargate does not support HAProxy and Varnish Cache as sidecar containers, which are needed to run the application properly.

Option B is incorrect because AWS Elastic Beanstalk is a fully managed service that automates the deployment and scaling of web applications and services using familiar servers such as Apache, Nginx, Passenger, and IIS. However, Elastic Beanstalk does not support HAProxy and Varnish Cache as part of the Tomcat solution stack, which are needed to run the application properly. Moreover, Elastic Beanstalk web server tier environments are designed to serve HTTP requests, not to process background tasks, which is the purpose of worker tier environments.

Option C is incorrect because AWS Elastic Beanstalk worker tier environments are designed to process background tasks using a daemon process that runs on each Amazon EC2 instance in the environment. Worker tier environments are not suitable for running web applications that serve HTTP requests, which is the case in the scenario. Moreover, Elastic Beanstalk does not support HAProxy and Varnish Cache as part of the Tomcat solution stack, which are needed to run the application properly.

AWS CodeCommit

AWS CodeDeploy

AWS CodePipeline

Amazon EC2 Auto Scaling

AWS Fargate

AWS Elastic Beanstalk

Question 3

Question Type: **MultipleChoice**

A company recently deployed its web application on AWS. The company is preparing for a large-scale sales event and must ensure that the web application can scale to meet the demand

The application's frontend infrastructure includes an Amazon CloudFront distribution that has an Amazon S3 bucket as an origin. The backend infrastructure includes an Amazon API Gateway API. several AWS Lambda functions, and an Amazon Aurora DB cluster

The company's DevOps engineer conducts a load test and identifies that the Lambda functions can fulfill the peak number of requests However, the DevOps engineer notices request latency during the initial burst of requests Most of the requests to the Lambda functions produce queries to the database A large portion of the invocation time is used to establish database connections

Which combination of steps will provide the application with the required scalability? (Select TWO)

Options:

A- Configure a higher reserved concurrency for the Lambda functions.

B- Configure a higher provisioned concurrency for the Lambda functions

C- Convert the DB cluster to an Aurora global database Add additional Aurora Replicas in AWS Regions based on the locations of the company's customers.

D- Refactor the Lambda Functions Move the code blocks that initialize database connections into the function handlers.

E- Use Amazon RDS Proxy to create a proxy for the Aurora database Update the Lambda functions to use the proxy endpoints for database connections.

Answer:

B, E

Explanation:

The correct answer is B and E. Configuring a higher provisioned concurrency for the Lambda functions will ensure that the functions are ready to respond to the initial burst of requests without any cold start latency. Using Amazon RDS Proxy to create a proxy for the Aurora database will enable the Lambda functions to reuse existing database connections and reduce the overhead of establishing new ones. This will also improve the scalability and availability of the database by managing the connection pool size and handling failovers. Option A is incorrect because reserved concurrency only limits the number of concurrent executions for a function, not pre-warms them. Option C is incorrect because converting the DB cluster to an Aurora global database will not address the issue of database connection latency, and may introduce additional costs and complexity. Option D is incorrect because moving the code blocks that initialize database

connections into the function handlers will not improve the performance or scalability of the Lambda functions, and may actually worsen the cold start latency.Reference:

AWS Lambda Provisioned Concurrency

Using Amazon RDS Proxy with AWS Lambda

Certified DevOps Engineer - Professional (DOP-C02) Study Guide(page 173)

Question 4

Question Type: **MultipleChoice**

A DevOps engineer is using AWS CodeDeploy across a fleet of Amazon EC2 instances in an EC2 Auto Scaling group. The associated CodeDeploy deployment group, which is integrated with EC2 Auto Scaling, is configured to perform in-place deployments with codeDeployDefault.oneAtATime During an ongoing new deployment, the engineer discovers that, although the overall deployment finished successfully, two out of five instances have the previous application revision deployed. The other three instances have the newest application revision

What is likely causing this issue?

Options:

A- The two affected instances failed to fetch the new deployment.

B- A failed Afterinstall lifecycle event hook caused the CodeDeploy agent to roll back to the previous version on the affected instances

C- The CodeDeploy agent was not installed in two affected instances.

D- EC2 Auto Scaling launched two new instances while the new deployment had not yet finished, causing the previous version to be deployed on the affected instances.

Answer:

B

Explanation:

When AWS CodeDeploy performs an in-place deployment, it updates the instances with the new application revision one at a time, as specified by the deployment configurationcodeDeployDefault.oneAtATime. If a lifecycle event hook, such asAfterInstall, fails during the deployment, CodeDeploy will attempt to roll back to the previous version on the affected instances. This is likely what happened with the two instances that still have the previous application revision deployed. The failure of theAfterInstalllifecycle event hook triggered the rollback mechanism, resulting in those instances reverting to the previous application revision.

AWS CodeDeploy documentation on redeployment and rollback procedures1.

Stack Overflow discussions on re-deploying older revisions with AWS CodeDeploy2.

AWS CLI reference guide for deploying a revision2.

Question 5

Question Type: MultipleChoice

A company is building a web and mobile application that uses a serverless architecture powered by AWS Lambda and Amazon API Gateway The company wants to fully automate the backend Lambda deployment based on code that is pushed to the appropriate environment branch in an AWS CodeCommit repository

The deployment must have the following:

* Separate environment pipelines for testing and production

* Automatic deployment that occurs for test environments only

Which steps should be taken to meet these requirements'?

Options:

A- Configure a new AWS CodePipelme service Create a CodeCommit repository for each environment Set up CodePipeline to retrieve the source code from the appropriate repository Set up the deployment step to deploy the Lambda functions with AWS CloudFormation.

B- Create two AWS CodePipeline configurations for test and production environments Configure the production pipeline to have a

manual approval step Create a
CodeCommit repository for each environment Set up each CodePipeline to retrieve the source code from the appropriate repository Set up the deployment step to deploy the Lambda functions with AWS CloudFormation.

C- Create two AWS CodePipeline configurations for test and production environments Configure the production pipeline to have a manual approval step. Create one CodeCommit repository with a branch for each environment Set up each CodePipeline to retrieve the source code from the appropriate branch in the repository. Set up the deployment step to deploy the Lambda functions with AWS CloudFormation

D- Create an AWS CodeBuild configuration for test and production environments Configure the production pipeline to have a manual approval step. Create one CodeCommit repository with a branch for each environment Push the Lambda function code to an Amazon S3 bucket Set up the deployment step to deploy the Lambda functions from the S3 bucket.

Answer:

C

Explanation:

The correct approach to meet the requirements for separate environment pipelines and automatic deployment for test environments is to create two AWS CodePipeline configurations, one for each environment. The production pipeline should have a manual approval step to ensure that changes are reviewed before being deployed to production. A single AWS CodeCommit repository with separate branches for each environment allows for organized and efficient code management. Each CodePipeline retrieves the source code from the appropriate branch in the repository. The deployment step utilizes AWS CloudFormation to deploy the Lambda functions, ensuring that the infrastructure as code is maintained and version-controlled.

AWS Lambda with Amazon API Gateway:Using AWS Lambda with Amazon API Gateway

Tutorial on using Lambda with API Gateway:Tutorial: Using Lambda with API Gateway

AWS CodePipeline automatic deployment:Set Up a Continuous Deployment Pipeline Using AWS CodePipeline

Building a pipeline for test and production stacks:Walkthrough: Building a pipeline for test and production stacks

Question 6

Question Type: **MultipleChoice**

A DevOps engineer manages a company's Amazon Elastic Container Service (Amazon ECS) cluster. The cluster runs on several Amazon EC2 instances that are in an Auto Scaling group. The DevOps

engineer must implement a solution that logs and reviews all stopped tasks for errors.

Which solution will meet these requirements?

Options:

A- Create an Amazon EventBridge rule to capture task state changes. Send the event to Amazon CloudWatch Logs. Use CloudWatch Logs Insights to investigate stopped tasks.

B- Configure tasks to write log data in the embedded metric format. Store the logs in Amazon CloudWatch Logs. Monitor the ContainerInstanceCount metric for changes.

C- Configure the EC2 instances to store logs in Amazon CloudWatch Logs. Create a CloudWatch Contributor Insights rule that uses the EC2 instance log data. Use the Contributor Insights rule to investigate stopped tasks.

D- Configure an EC2 Auto Scaling lifecycle hook for the EC2_INSTANCE_TERMINATING scale-in event. Write the SystemEventLog file to Amazon S3. Use Amazon Athena to query the log file for errors.

Answer:

A

Explanation:

The best solution to log and review all stopped tasks for errors is to use Amazon EventBridge and Amazon CloudWatch Logs. Amazon EventBridge allows the DevOps engineer to create a rule that matches task state change events from Amazon ECS. The rule can then send the event data to Amazon CloudWatch Logs as the target. Amazon CloudWatch Logs can store and monitor the log data, and also provide CloudWatch Logs Insights, a feature that enables the DevOps engineer to interactively search and analyze the log data. Using CloudWatch Logs Insights, the DevOps engineer can filter and aggregate the log data based on various fields, such as cluster, task, container, and reason. This way, the DevOps engineer can easily identify and investigate the stopped tasks and their errors.

The other options are not as effective or efficient as the solution in option A. Option B is not suitable because the embedded metric format is designed for custom metrics, not for logging task state changes. Option C is not feasible because the EC2 instances do not store the task state change events in their logs. Option D is not relevant because the EC2_INSTANCE_TERMINATING lifecycle hook is triggered when an EC2 instance is terminated by the Auto Scaling group, not when a task is stopped by Amazon ECS.

: Creating a CloudWatch Events Rule That Triggers on an Event - Amazon Elastic Container Service

: Sending and Receiving Events Between AWS Accounts - Amazon EventBridge

: Working with Log Data - Amazon CloudWatch Logs

: Analyzing Log Data with CloudWatch Logs Insights - Amazon CloudWatch Logs

: Embedded Metric Format - Amazon CloudWatch

: Amazon EC2 Auto Scaling Lifecycle Hooks - Amazon EC2 Auto Scaling

Question 7

Question Type: **MultipleChoice**

A company uses AWS Directory Service for Microsoft Active Directory as its identity provider (IdP). The company requires all infrastructure to be

defined and deployed by AWS CloudFormation.

A DevOps engineer needs to create a fleet of Windows-based Amazon EC2 instances to host an application. The DevOps engineer has created a

CloudFormation template that contains an EC2 launch template, IAM role, EC2 security group, and EC2 Auto Scaling group. The DevOps engineer must implement a solution that joins all EC2 instances to the domain of the AWS Managed Microsoft AD directory.

Which solution will meet these requirements with the MOST operational efficiency?

Options:

A- In the CloudFormation template, create an AWS::SSM::Document resource that joins the EC2 instance to the AWS Managed Microsoft AD domain by using the parameters for the existing directory. Update the launch template to include the SSMAssociation property to use the new SSM document. Attach the AmazonSSMManagedInstanceCore and AmazonSSMDirectoryServiceAccess AWS managed policies to the IAM role that the EC2 instances use.

B- In the CloudFormation template, update the launch template to include specific tags that propagate on launch. Create an AWS::SSM::Association resource to associate the AWS-JoinDirectoryServiceDomain Automation runbook with the EC2 instances that have the specified tags. Define the required parameters to join the AWS Managed Microsoft AD directory. Attach the AmazonSSMManagedInstanceCore and AmazonSSMDirectoryServiceAccess AWS managed policies to the IAM role that the EC2 instances use.

C- Store the existing AWS Managed Microsoft AD domain connection details in AWS Secrets Manager. In the CloudFormation template, create an AWS::SSM::Association resource to associate the AWS-CreateManagedWindowsInstanceWithApproval Automation runbook with the EC2 Auto Scaling group. Pass the ARNs for the parameters from Secrets Manager to join the domain. Attach the AmazonSSMDirectoryServiceAccess and SecretsManagerReadWrite AWS managed policies to the IAM role that the EC2 instances use.

D- Store the existing AWS Managed Microsoft AD domain administrator credentials in AWS Secrets Manager. In the CloudFormation template, update the EC2 launch template to include user data. Configure the user data to pull the administrator credentials from Secrets Manager and to join the AWS Managed Microsoft AD domain. Attach the AmazonSSMManagedInstanceCore and

SecretsManagerReadWrite AWS managed policies to the IAM role that the EC2 instances use.

Answer:

B

Explanation:

To meet the requirements, the DevOps engineer needs to create a solution that joins all EC2 instances to the domain of the AWS Managed Microsoft AD directory with the most operational efficiency. The DevOps engineer can use AWS Systems Manager Automation to automate the domain join process using an existing runbook called AWS-JoinDirectoryServiceDomain. This runbook can join Windows instances to an AWS Managed Microsoft AD or Simple AD directory by using PowerShell commands. The DevOps engineer can create an AWS::SSM::Association resource in the CloudFormation template to associate the runbook with the EC2 instances that have specific tags. The tags can be defined in the launch template and propagated on launch to the EC2 instances. The DevOps engineer can also define the required parameters for the runbook, such as the directory ID, directory name, and organizational unit. The DevOps engineer can attach the AmazonSSMManagedInstanceCore and AmazonSSMDirectoryServiceAccess AWS managed policies to the IAM role that the EC2 instances use. These policies grant the necessary permissions for Systems Manager and Directory Service operations.

Question 8

Question Type: MultipleChoice

A company needs a strategy for failover and disaster recovery of its data and application. The application uses a MySQL database and Amazon EC2 instances. The company requires a maximum RPO of 2 hours and a maximum RTO of 10 minutes for its data and application at all times.

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

Options:

A- Create an Amazon Aurora Single-AZ cluster in multiple AWS Regions as the data store. Use Aurora's automatic recovery capabilities in the event of a disaster.

B- Create an Amazon Aurora global database in two AWS Regions as the data store. In the event of a failure, promote the secondary Region to the primary for the application. Update the application to use the Aurora cluster endpoint in the secondary Region.

C- Create an Amazon Aurora cluster in multiple AWS Regions as the data store. Use a Network Load Balancer to balance the database traffic in different Regions.

D- Set up the application in two AWS Regions. Use Amazon Route 53 failover routing that points to Application Load Balancers in both Regions. Use health checks and Auto Scaling groups in each Region.

E- Set up the application in two AWS Regions. Configure AWS Global Accelerator to point to Application Load Balancers (ALBs) in both Regions. Add both ALBs to a single endpoint group. Use health checks and Auto Scaling groups in each Region.

Answer:

B, E

Explanation:

Verified Answer: B and E

Short To meet the requirements of failover and disaster recovery, the company should use the following deployment strategies:

Create an Amazon Aurora global database in two AWS Regions as the data store. In the event of a failure, promote the secondary Region to the primary for the application. Update the application to use the Aurora cluster endpoint in the secondary Region. This strategy can provide a low RPO and RTO for the data, as Aurora global database replicates data with minimal latency across Regions and allows fast and easy failover12. The company can use the Amazon Aurora cluster endpoint to connect to the current primary DB cluster without needing to change any application code1.

Set up the application in two AWS Regions. Configure AWS Global Accelerator to point to Application Load Balancers (ALBs) in both Regions. Add both ALBs to a single endpoint group. Use health checks and Auto Scaling groups in each Region. This strategy can provide high availability and performance for the application, as AWS Global Accelerator uses the AWS global network to route traffic to the closest healthy endpoint3. The company can also use static IP addresses that are assigned by Global Accelerator as a fixed entry point for their application1. By using health checks and Auto Scaling groups, the company can ensure that their application can scale up or down based on demand and handle any instance failures4.

The other options are incorrect because:

Creating an Amazon Aurora Single-AZ cluster in multiple AWS Regions as the data store would not provide a fast failover or disaster recovery solution, as the company would need to manually restore data from backups or snapshots in another Region in case of a failure.

Creating an Amazon Aurora cluster in multiple AWS Regions as the data store and using a Network Load Balancer to balance the database traffic in different Regions would not work, as Network Load Balancers do not support cross-Region routing. Moreover, this

strategy would not provide a consistent view of the data across Regions, as Aurora clusters do not replicate data automatically between Regions unless they are part of a global database.

Setting up the application in two AWS Regions and using Amazon Route 53 failover routing that points to Application Load Balancers in both Regions would not provide a low RTO, as Route 53 failover routing relies on DNS resolution, which can take time to propagate changes across different DNS servers and clients. Moreover, this strategy would not provide deterministic routing, as Route 53 failover routing depends on DNS caching behavior, which can vary depending on different factors.