Q1. A Developer is designing a continuous deployment workflow for a new Development team to facilitate the process for source code promotion in AWS. Developers would like to store and promote code for deployment from development to production while maintaining the ability to roll back that deployment if it fails.
Which design will incur the LEAST amount of downtime?

Options

- Create one repository in AWS CodeCommit. Create a development branch to hold merged changes. Use AWS CodeBuild to build and test the code stored in the development branch triggered on a new commit. Merge to the master and deploy to production by using AWS CodeDeploy for a blue/green deployment.
- Create one repository for each Developer in AWS CodeCommit and another repository to hold the production code. Use AWS CodeBuild to merge development and production repositories, and deploy to production by using AWS CodeDeploy for a blue/green deployment.
- Create one repository for development code in AWS CodeCommit and another repository to hold the production code. Use AWS CodeBuild to merge development and production repositories, and deploy to production by using AWS CodeDeploy for a blue/green deployment.
- Create a shared Amazon S3 bucket for the Development team to store their code. Set up an Amazon CloudWatch Events rule to trigger an AWS Lambda function that deploys the code to production by using AWS CodeDeploy for a blue/green deployment.

Q2. A business has an application that consists of five independent AWS Lambda functions.
The Engineer has built a CI/CD pipeline using AWS CodePipeline and AWS CodeBuild that builds, tests, packages, and deploys each Lambda function in sequence. The pipeline uses an Amazon CloudWatch Events rule to ensure the pipeline execution starts as quickly as possible after a change is made to the application source code.
After working with the pipeline for a few months, the Engineer has noticed the pipeline takes too long to complete.
What should the Engineer implement to BEST improve the speed of the pipeline?

Options

- ○ Modify the CodeBuild projects within the pipeline to use a compute type with more available network throughput.
- ○ Create a custom CodeBuild execution environment that includes a symmetric multiprocessing configuration to run the builds in parallel.
- ◉ Modify the CodePipeline configuration to execute actions for each Lambda function in parallel by specifying the same runOrder.
- ○ Modify each CodeBuild project to run within a VPC and use dedicated instances to increase throughput.

Q3. A Engineer must track the health of a stateless RESTful service sitting behind a Classic Load Balancer. The deployment of new application revisions is through a CI/CD pipeline. If the service's latency increases beyond a defined threshold, deployment should be stopped until the service has recovered.
Which of the following methods allow for the QUICKEST detection time?

Options

- Use Amazon CloudWatch metrics provided by Elastic Load Balancing to calculate average latency. Alarm and stop deployment when latency increases beyond the defined threshold.
- Use AWS Lambda and Elastic Load Balancing access logs to detect average latency. Alarm and stop deployment when latency increases beyond the defined threshold.
- Use AWS CodeDeploy's MinimumHealthyHosts setting to define thresholds for rolling back deployments. If these thresholds are breached, roll back the deployment.
- Use Metric Filters to parse application logs in Amazon CloudWatch Logs. Create a filter for latency. Alarm and stop deployment when latency increases beyond the defined threshold.

Q4. A Engineer is building a continuous deployment pipeline for a serverless application using AWS CodePipeline and AWS CodeBuild. The source, build, and test stages have been created with the deploy stage remaining. The company wants to reduce the risk of an unsuccessful deployment by deploying to a specified subset of customers and monitoring prior to a full release to all customers.
How should the deploy stage be configured to meet these requirements?

Options

- (●) Use AWS CloudFormation to publish a new version on every stack update. Then set up a CodePipeline approval action for a Developer to test and approve the new version. Finally, use a CodePipeline invoke action to update an AWS Lambda function to use the production alias
- ( ) Use CodeBuild to use the AWS CLI to update the AWS Lambda function code, then publish a new version of the function and update the production alias to point to the new version of the function.
- ( ) Use AWS CloudFormation to define the serverless application and AWS CodeDeploy to deploy the AWS Lambda functions using DeploymentPreference: . Canary10Percent15Minutes
- ( ) Use AWS CloudFormation to publish a new version on every stack update. Use the RoutingConfig property of the AWS::Lambda::Alias resource to update the traffic routing during the stack update.

Q5. A Development team is currently using AWS CodeDeploy to deploy an application revision to an Auto Scaling group. If the deployment process fails, it must be rolled back automatically and a notification must be sent.
What is the MOST effective configuration that can satisfy all of the requirements?

Options

- Create Amazon CloudWatch Events rules for CodeDeploy operations. Configure a CloudWatch Events rule to send out an Amazon SNS message when the deployment fails. Configure CodeDeploy to automatically roll back when the deployment fails.
- Use available Amazon CloudWatch metrics for CodeDeploy to create CloudWatch alarms. Configure CloudWatch alarms to send out an Amazon SNS message when the deployment fails. Use AWS CLI to redeploy a previously deployed revision.
- Configure a CodeDeploy agent to create a trigger that will send notification to Amazon SNS topics when the deployment fails. Configure CodeDeploy to automatically roll back when the deployment fails.
- Use AWS CloudTrail to monitor API calls made by or on behalf of CodeDeploy in the AWS account. Send an Amazon SNS message when deployment fails. Use AWS CLI to redeploy a previously deployed revision.

Q6. A company is using AWS for an application. The Development team must automate its deployments. The team has set up an AWS CodePipeline to deploy the application to Amazon EC2 instances by using AWS CodeDeploy after it has been built using the AWS CodeBuild service.

The team would like to add automated testing to the pipeline to confirm that the application is healthy before deploying it to the next stage of the pipeline using the same code. The team requires a manual approval action before the application is deployed, even if the test is successful. The testing and approval must be accomplished at the lowest costs, using the simplest management solution.

Which solution will meet these requirements?

Options

- Add a manual approval action after the last deploy action of the pipeline. Use Amazon SNS to inform the team of the stage being triggered. Next, add a test action using CodeBuild to do the required tests. At the end of the pipeline, add a deploy action to deploy the application to the next stage.
- Add a test action after the last deploy action of the pipeline. Configure the action to use CodeBuild to perform the required tests. If these tests are successful, mark the action as successful. Add a manual approval action that uses Amazon SNS to notify the team, and add a deploy action to deploy the application to the next stage.
- Create a new pipeline that uses a source action that gets the code from the same repository as the first pipeline. Add a deploy action to deploy the code to a test environment. Use a test action using AWS Lambda to test the deployment. Add a manual approval action by using Amazon SNS to notify the team, and add a deploy action to deploy the application to the next stage.
- Add a test action after the last deployment action. Use a Jenkins server on Amazon EC2 to do the required tests and mark the action as successful if the tests pass. Create a manual approval action that uses Amazon SQS to notify the team and add a deploy action to deploy the application to the next stage.