Free Questions for Professional-Cloud-DevOps-Engineer by dumpssheet

Shared by Hardin on 15-04-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

You are the Site Reliability Engineer responsible for managing your company's data services and products. You regularly navigate operational challenges, such as unpredictable data volume and high cost, with your company's data ingestion processes. You recently learned that a new data ingestion product will be developed in Google Cloud. You need to collaborate with the product development team to provide operational input on the new product. What should you do?

Options:

A- Deploy the prototype product in a test environment, run a load test, and share the results with the product development team.

B- When the initial product version passes the quality assurance phase and compliance assessments, deploy the product to a staging environment. Share error logs and performance metrics with the product development team.

C- When the new product is used by at least one internal customer in production, share error logs and monitoring metrics with the product development team.

D- Review the design of the product with the product development team to provide feedback early in the design phase.

Answer:

D

Explanation:

The correct answer is D, Review the design of the product with the product development team to provide feedback early in the design phase.

According to the Google Cloud DevOps best practices, a Site Reliability Engineer (SRE) should collaborate with the product development team from the beginning of the product lifecycle, not just after the product is deployed or tested. This way, the SRE can provide operational input on the product design, such as scalability, reliability, security, and cost efficiency. The SRE can also help define service level objectives (SLOs) and service level indicators (SLIs) for the product, as well as monitoring and alerting strategies. By collaborating early and often, the SRE and the product development team can ensure that the product meets the operational requirements and expectations of the customers.

Preparing for Google Cloud Certification: Cloud DevOps Engineer Professional Certificate, Course 1: Site Reliability Engineering and DevOps, Week 1: Introduction to SRE and DevOps.

Question 2

Question Type: MultipleChoice

Your company runs services by using Google Kubernetes Engine (GKE). The GKE clusters in the development environment run applications with verbose logging enabled. Developers view logs by using the kubect1 logs

command and do not use Cloud Logging. Applications do not have a uniform logging structure defined. You need to minimize the costs associated with application logging while still collecting GKE operational logs. What should you do?

Options:

A- Run the gcloud container clusters update --logging---SYSTEM command for the development cluster.

B- Run the gcloud container clusters update logging=WORKLOAD command for the development cluster.

C- Run the gcloud logging sinks update _Defau1t --disabled command in the project associated with the development environment.

D- Add the severity >= DEBUG resource. type 'k83 container' exclusion filter to the Default logging sink in the project associated with the development environment.

Answer:

A

Question 3

Question Type: **MultipleChoice**

Your company runs applications in Google Kubernetes Engine (GKE) that are deployed following a GitOps methodology.

Application developers frequently create cloud resources to support their applications. You want to give developers the ability to manage infrastructure as code, while ensuring that you follow Google-recommended practices. You need to ensure that infrastructure as code reconciles periodically to avoid configuration drift. What should you do?

Options:

A- Install and configure Config Connector in Google Kubernetes Engine (GKE).

B- Configure Cloud Build with a Terraform builder to execute plan and apply commands.

C- Create a Pod resource with a Terraform docker image to execute terraform plan and terraform apply commands.

D- Create a Job resource with a Terraform docker image to execute terraforrm plan and terraform apply commands.

Answer:

A

Explanation:

The best option to give developers the ability to manage infrastructure as code, while ensuring that you follow Google-recommended practices, is to install and configure Config Connector in Google Kubernetes Engine (GKE).

Config Connector is a Kubernetes add-on that allows you to manage Google Cloud resources through Kubernetes. You can use Config Connector to create, update, and delete Google Cloud resources using Kubernetes manifests. Config Connector also reconciles the state of the Google Cloud resources with the desired state defined in the manifests, ensuring that there is no configuration drift1.

Config Connector follows the GitOps methodology, as it allows you to store your infrastructure configuration in a Git repository, and use tools such as Anthos Config Management or Cloud Source Repositories to sync the configuration to your GKE cluster. This way, you can use Git as the source of truth for your infrastructure, and enable reviewable and version-controlled workflows2.

Config Connector can be installed and configured in GKE using either the Google Cloud Console or the gcloud command-line tool. You need to enable the Config Connector add-on for your GKE cluster, and create a Google Cloud service account with the necessary permissions to manage the Google Cloud resources. You also need to create a Kubernetes namespace for each Google Cloud project that you want to manage with Config Connector3.

By using Config Connector in GKE, you can give developers the ability to manage infrastructure as code, while ensuring that you follow Google-recommended practices. You can also benefit from the features and advantages of Kubernetes, such as declarative configuration, observability, and portability4.

1: Overview | Artifact Registry Documentation | Google Cloud

2: Deploy Anthos on GKE with Terraform part 1: GitOps with Config Sync | Google Cloud Blog

3: Installing Config Connector | Config Connector Documentation | Google Cloud

4: Why use Config Connector? | Config Connector Documentation | Google Cloud

Question 4

Question Type: **MultipleChoice**

You recently migrated an ecommerce application to Google Cloud. You now need to prepare the application for the upcoming peak traffic season. You want to follow Google-recommended practices. What should you do first to prepare for the busy season?

Options:

A- Migrate the application to Cloud Run, and use autoscaling.

B- Load test the application to profile its performance for scaling.

C- Create a Terraform configuration for the application's underlying infrastructure to quickly deploy to additional regions.

D- Pre-provision the additional compute power that was used last season, and expect growth.

Answer:

B

Explanation:

The first thing you should do to prepare your ecommerce application for the upcoming peak traffic season is to load test the application to profile its performance for scaling. Load testing is a process of simulating high traffic or user demand on your application and measuring how it responds. Load testing can help you identify any bottlenecks, errors, or performance issues that might affect your application during the busy season1. Load testing can also help you determine the optimal scaling strategy for your application, such as horizontal scaling (adding more instances) or vertical scaling (adding more resources to each instance)2.

There are different tools and methods for load testing your ecommerce application on Google Cloud, depending on the type and complexity of your application. For example, you can use Cloud Load Balancing to distribute traffic across multiple instances of your application, and use Cloud Monitoring to measure the latency, throughput, and error rate of your application3. You can also use Cloud Functions or Cloud Run to create serverless load generators that can simulate user requests and send them to your application4. Alternatively, you can use third-party tools such as Apache JMeter or Locust to create and run load tests on your application.

By load testing your ecommerce application before the peak traffic season, you can ensure that your application is ready to handle the expected load and provide a good user experience. You can also use the results of your load tests to plan and implement other steps to prepare your application for the busy season, such as migrating to a more scalable platform, creating a Terraform configuration for deploying to additional regions, or pre-provisioning additional compute power.

1: Load Testing 101: How To Test Website Performance | BlazeMeter

2: Scaling applications | Google Cloud

3: Load testing using Google Cloud | Solutions | Google Cloud

4: Serverless load testing using Cloud Functions | Solutions | Google Cloud

Question 5

Question Type: MultipleChoice

You are the Operations Lead for an ongoing incident with one of your services. The service usually runs at around 70% capacity. You notice that one node is returning 5xx errors for all requests. There has also been a noticeable increase in support cases from customers. You need to remove the offending node from the load balancer pool so that you can isolate and investigate the node. You want to follow Google-recommended practices to manage the incident and reduce the impact on users. What should you do?

Options:

A- 1. Communicate your intent to the incident team.
2. Perform a load analysis to determine if the remaining nodes can handle the increase in traffic offloaded from the removed node, and scale appropriately.
3. When any new nodes report healthy, drain traffic from the unhealthy node, and remove the unhealthy node from service.

B- 1. Communicate your intent to the incident team.
2. Add a new node to the pool, and wait for the new node to report as healthy.
3. When traffic is being served on the new node, drain traffic from the unhealthy node, and remove the old node from service.

C- 1 . Drain traffic from the unhealthy node and remove the node from service.
2. Monitor traffic to ensure that the error is resolved and that the other nodes in the pool are handling the traffic appropriately.
3. Scale the pool as necessary to handle the new load.
4. Communicate your actions to the incident team.

D- 1 . Drain traffic from the unhealthy node and remove the old node from service.
2. Add a new node to the pool, wait for the new node to report as healthy, and then serve traffic to the new node.
3. Monitor traffic to ensure that the pool is healthy and is handling traffic appropriately.
4. Communicate your actions to the incident team.

Answer:

A

Explanation:

The correct answer is A, Communicate your intent to the incident team. Perform a load analysis to determine if the remaining nodes can handle the increase in traffic offloaded from the removed node, and scale appropriately. When any new nodes report healthy, drain traffic from the unhealthy node, and remove the unhealthy node from service.

This answer follows the Google-recommended practices for incident management, as described in the Chapter 9 - Incident Response, Google SRE Book1. According to this source, some of the best practices are:

Maintain a clear line of command. Designate clearly defined roles. Keep a working record of debugging and mitigation as you go. Declare incidents early and often.

Communicate your intent before taking any action that might affect the service or the incident response. This helps to avoid confusion, duplication of work, or unintended consequences.

Perform a load analysis before removing a node from the load balancer pool, as this might affect the capacity and performance of the service. Scale the pool as necessary to handle the expected load.

Drain traffic from the unhealthy node before removing it from service, as this helps to avoid dropping requests or causing errors for users.

Answer A follows these best practices by communicating the intent to the incident team, performing a load analysis and scaling the pool, and draining traffic from the unhealthy node before removing it.

Answer B does not follow the best practice of performing a load analysis before adding or removing nodes, as this might cause overloading or underutilization of resources.

Answer C does not follow the best practice of communicating the intent before taking any action, as this might cause confusion or conflict with other responders.

Answer D does not follow the best practice of draining traffic from the unhealthy node before removing it, as this might cause errors for users.

1: Chapter 9 - Incident Response, Google SRE Book

Question 6

Question Type: **MultipleChoice**

Your Cloud Run application writes unstructured logs as text strings to Cloud Logging. You want to convert the unstructured logs to JSON-based structured logs. What should you do?

Options:

A- A Install a Fluent Bit sidecar container, and use a JSON parser.

B- Install the log agent in the Cloud Run container image, and use the log agent to forward logs to Cloud Logging.

C- Configure the log agent to convert log text payload to JSON payload.

D- Modify the application to use Cloud Logging software development kit (SDK), and send log entries with a jsonPay10ad field.

Answer:

D

Explanation:

The correct answer is D, Modify the application to use Cloud Logging software development kit (SDK), and send log entries with a jsonPayload field.

Cloud Logging SDKs are libraries that allow you to write structured logs from your Cloud Run application. You can use the SDKs to create log entries with a jsonPayload field, which contains a JSON object with the properties of your log entry. The jsonPayload field allows you to use advanced features of Cloud Logging, such as filtering, querying, and exporting logs based on the properties of your log entry1.

To use Cloud Logging SDKs, you need to install the SDK for your programming language, and then use the SDK methods to create and send log entries to Cloud Logging. For example, if you are using Node.js, you can use the following code to write a structured log entry with a jsonPayload field2:

```
// Imports the Google Cloud client library

const {Logging} = require('@google-cloud/logging');
```

```
// Creates a client

const logging = new Logging();

// Selects the log to write to

const log = logging.log('my-log');

// The data to write to the log

const text = 'Hello, world!';

const metadata = {

// Set the Cloud Run service name and revision as labels

labels: {

service_name: process.env.K_SERVICE || 'unknown',

revision_name: process.env.K_REVISION || 'unknown',

},

// Set the log entry payload type and value

jsonPayload: {

message: text,
```

```
timestamp: new Date(),

},

};

// Prepares a log entry

const entry = log.entry(metadata);

// Writes the log entry

await log.write(entry);

console.log(`Logged: ${text}`);
```

Using Cloud Logging SDKs is the best way to convert unstructured logs to structured logs, as it provides more flexibility and control over the format and content of your log entries.

Using a Fluent Bit sidecar container is not a good option, as it adds complexity and overhead to your Cloud Run application. Fluent Bit is a lightweight log processor and forwarder that can be used to collect and parse logs from various sources and send them to different destinations3. However, Cloud Run does not support sidecar containers, so you would need to run Fluent Bit as part of your main container image. This would require modifying your Dockerfile and configuring Fluent Bit to read logs from supported locations and parse them as JSON. This is more cumbersome and less reliable than using Cloud Logging SDKs.

Using the log agent in the Cloud Run container image is not possible, as the log agent is not supported on Cloud Run. The log agent is a service that runs on Compute Engine or Google Kubernetes Engine instances and collects logs from various applications and system components. However, Cloud Run does not allow you to install or run any agents on its underlying infrastructure, as it is a fully managed

service that abstracts away the details of the underlying platform.

Storing the password directly in the code is not a good practice, as it exposes sensitive information and makes it hard to change or rotate the password. It also requires rebuilding and redeploying the application each time the password changes, which adds unnecessary work and downtime.

1: Writing structured logs | Cloud Run Documentation | Google Cloud

2: Write structured logs | Cloud Run Documentation | Google Cloud

3: Fluent Bit - Fast and Lightweight Log Processor & Forwarder

: Logging Best Practices for Serverless Applications - Google Codelabs

: About the logging agent | Cloud Logging Documentation | Google Cloud

: Cloud Run FAQ | Google Cloud

Question 7

Question Type: **MultipleChoice**

You are designing a system with three different environments: development, quality assurance (QA), and production.

Each environment will be deployed with Terraform and has a Google Kubemetes Engine (GKE) cluster created so that application teams can deploy their applications. Anthos Config Management will be used and templated to deploy

infrastructure level resources in each GKE cluster. All users (for example, infrastructure operators and application owners) will use GitOps. How should you structure your source control repositories for both Infrastructure as Code (laC) and application code?

Options:

A- Cloud Infrastructure (Terraform) repository is shared: different directories are different environments
GKE Infrastructure (Anthos Config Management Kustomize manifests) repository is shared: different overlay directories are different environments
Application (app source code) repositories are separated: different branches are different features

B- Cloud Infrastructure (Terraform) repository is shared: different directories are different environments
GKE Infrastructure (Anthos Config Management Kustomize manifests) repositories are separated: different branches are different environments
Application (app source code) repositories are separated: different branches are different features

C- Cloud Infrastructure (Terraform) repository is shared: different branches are different environments
GKE Infrastructure (Anthos Config Management Kustomize manifests) repository is shared: different overlay directories are different environments
Application (app source code) repository is shared: different directories are different features

D- Cloud Infrastructure (Terraform) repositories are separated: different branches are different environments
GKE Infrastructure (Anthos Config Management Kustomize manifests) repositories are separated: different overlay directories are different environments

Application (app source code) repositories are separated: different branches are different features

Answer:

B

Explanation:

The correct answer is B, Cloud Infrastructure (Terraform) repository is shared: different directories are different environments. GKE Infrastructure (Anthos Config Management Kustomize manifests) repositories are separated: different branches are different environments. Application (app source code) repositories are separated: different branches are different features.

This answer follows the best practices for using Terraform and Anthos Config Management with GitOps, as described in the following sources:

For Terraform, it is recommended to use a single repository for all environments, and use directories to separate them. This way, you can reuse the same Terraform modules and configurations across environments, and avoid code duplication and drift. You can also use Terraform workspaces to isolate the state files for each environment12.

For Anthos Config Management, it is recommended to use separate repositories for each environment, and use branches to separate the clusters within each environment. This way, you can enforce different policies and configurations for each environment, and use pull requests to promote changes across environments. You can also use Kustomize to create overlays for each cluster that apply specific patches or customizations34.

For application code, it is recommended to use separate repositories for each application, and use branches to separate the features or bug fixes for each application. This way, you can isolate the development and testing of each application, and use pull requests to merge

changes into the main branch. You can also use tags or labels to trigger deployments to different environments5 .

1: Best practices for using Terraform | Google Cloud

2: Terraform Recommended Practices - Part 1 | Terraform - HashiCorp Learn

3: Deploy Anthos on GKE with Terraform part 1: GitOps with Config Sync | Google Cloud Blog

4: Using Kustomize with Anthos Config Management | Anthos Config Management Documentation | Google Cloud

5: Deploy Anthos on GKE with Terraform part 3: Continuous Delivery with Cloud Build | Google Cloud Blog

: GitOps-style continuous delivery with Cloud Build | Cloud Build Documentation | Google Cloud

Question 8

Question Type: **MultipleChoice**

You are deploying an application to Cloud Run. The application requires a password to start. Your organization requires that all passwords are rotated every 24 hours, and your application must have the latest password. You need to deploy the application with no downtime. What should you do?

Options:

A- Store the password in Secret Manager and send the secret to the application by using environment variables.

B- Store the password in Secret Manager and mount the secret as a volume within the application.

C- Use Cloud Build to add your password into the application container at build time. Ensure that Artifact Registry is secured from public access.

D- Store the password directly in the code. Use Cloud Build to rebuild and deploy the application each time the password changes.

Answer:

B

Explanation:

The correct answer is B, Store the password in Secret Manager and mount the secret as a volume within the application.

Secret Manager is a service that allows you to securely store and manage sensitive data such as passwords, API keys, certificates, and tokens. You can use Secret Manager to rotate your secrets automatically or manually, and access them from your Cloud Run applications1.

There are two ways to use secrets from Secret Manager in Cloud Run:

As environment variables: You can set environment variables that point to secrets in Secret Manager. Cloud Run will resolve the secrets at runtime and inject them into the environment of your application. However, this method has some limitations, such as:

The environment variables are cached for up to 10 minutes, so you may not get the latest version of the secret immediately.

The environment variables are visible in plain text in the Cloud Console and the Cloud SDK, which may expose sensitive information.

The environment variables are limited to 4 KB of data, which may not be enough for some secrets.2

As file system volumes: You can mount secrets from Secret Manager as files in a volume within your application. Cloud Run will create a tmpfs volume and write the secrets as files in it. This method has some advantages, such as:

The files are updated every 30 seconds, so you can get the latest version of the secret faster.

The files are not visible in the Cloud Console or the Cloud SDK, which provides better security.

The files can store up to 64 KB of data, which allows for larger secrets.3

Therefore, for your use case, it is better to use the second method and mount the secret as a file system volume within your application. This way, you can ensure that your application has the latest password, and you can deploy it with no downtime.

To mount a secret as a file system volume in Cloud Run, you can use the following command:

gcloud beta run deploy SERVICE --image IMAGE_URL --update-secrets=/path/to/file=secretName:version

where:

SERVICE is the name of your Cloud Run service.

IMAGE_URL is the URL of your container image.

/path/to/file is the path where you want to mount the secret file in your application.

secretName is the name of your secret in Secret Manager.

You can also use the Cloud Console to mount secrets as file system volumes. For more details, see Mounting secrets from Secret Manager.

1: Overview | Secret Manager Documentation | Google Cloud

2: Using secrets as environment variables | Cloud Run Documentation | Google Cloud

3: Mounting secrets from Secret Manager | Cloud Run Documentation | Google Cloud

Question 9

Question Type: MultipleChoice

You are developing reusable infrastructure as code modules. Each module contains integration tests that launch the module in a test project. You are using GitHub for source control. You need to Continuously test your feature branch and ensure that all code is tested before changes are accepted. You need to implement a solution to automate the integration tests. What should you do?

Options:

A- Use a Jenkins server for Cl/CD pipelines. Periodically run all tests in the feature branch.

B- Use Cloud Build to run the tests. Trigger all tests to run after a pull request is merged.

C- Ask the pull request reviewers to run the integration tests before approving the code.

D- Use Cloud Build to run tests in a specific folder. Trigger Cloud Build for every GitHub pull request.

Answer:

D

Explanation:

Cloud Build is a service that executes your builds on Google Cloud Platform infrastructure.Cloud Build can import source code from Google Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives1.Cloud Build can also run integration tests as part of your build steps2.

You can use Cloud Build to run tests in a specific folder by specifying the path to the folder in thedirfield of your build step3. For example, if you have a folder namedteststhat contains your integration tests, you can use the following build step to run them:

steps:

- name: 'gcr.io/cloud-builders/go'

args: ['test', '-v']

dir: 'tests'

Copy

You can use Cloud Build to trigger builds for every GitHub pull request by using the Cloud Build GitHub app.The app allows you to automatically build on Git pushes and pull requests and view your build results on GitHub and Google Cloud console4.You can configure the app to run builds on specific branches, tags, or paths5. For example, if you want to run builds on pull requests that target themasterbranch, you can use the following trigger configuration:

includedFiles:

- '**'

name: 'pull-request-trigger'

github:

name: 'my-repo'

owner: 'my-org'

pullRequest:

branch: '^master$'

Using Cloud Build to run tests in a specific folder and trigger builds for every GitHub pull request is a good way to continuously test your feature branch and ensure that all code is tested before changes are accepted. This way, you can catch any errors or bugs early and prevent them from affecting the main branch.

Using a Jenkins server for CI/CD pipelines is not a bad option, but it would require more setup and maintenance than using Cloud Build, which is fully managed by Google Cloud. Periodically running all tests in the feature branch is not as efficient as running tests for every pull request, as it may delay the feedback loop and increase the risk of conflicts or failures.

Using Cloud Build to run the tests after a pull request is merged is not a good practice, as it may introduce errors or bugs into the main branch that could have been prevented by testing before merging.

Asking the pull request reviewers to run the integration tests before approving the code is not a reliable way of ensuring code quality, as it depends on human intervention and may be prone to errors or oversights.

1:Overview | Cloud Build Documentation | Google Cloud

2:Running integration tests | Cloud Build Documentation | Google Cloud

3: Build configuration overview | Cloud Build Documentation | Google Cloud

4:Building repositories from GitHub | Cloud Build Documentation | Google Cloud

5: Creating GitHub app triggers | Cloud Build Documentation | Google Cloud

Question 10

Question Type: MultipleChoice

Your organization is starting to containerize with Google Cloud. You need a fully managed storage solution for container images and Helm charts. You need to identify a storage solution that has native integration into existing Google Cloud services, including Google Kubernetes Engine (GKE), Cloud Run, VPC Service Controls, and Identity and Access Management (IAM). What should you do?

Options:

A- Use Docker to configure a Cloud Storage driver pointed at the bucket owned by your organization.

B- Configure Container Registry as an OCI-based container registry for container images.

C- Configure Artifact Registry as an OCI-based container registry for both Helm charts and container images.

D- Configure an open source container registry server to run in GKE with a restrictive role-based access control (RBAC) configuration.

Answer:

C