DUMPSBOSS.

Google Certified Professional - Cloud Developer

Google Professional-Cloud-Developer

Version Demo

Total Demo Questions: 34

Total Premium Questions: 342

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Designing highly scalable, available, and reliable cloud-native applications	75
Topic 2, Building and testing applications	31
Topic 3, Deploying applications	72
Topic 4, Integrating with Google Cloud services	114
Topic 5, Managing application performance monitoring	49
Topic 6, Mix Questions	1
Total	342

QUESTION NO: 1

You are developing a microservice-based application that will run on Google Kubernetes Engine (GKE). Some of the services need to access different Google Cloud APIs. How should you set up authentication of these services in the cluster following Google-recommended best practices? (Choose two.)

A. Use the service account attached to the GKE node.

B. Enable Workload Identity in the cluster via the gcloud command-line tool.

C. Access the Google service account keys from a secret management service.

D. Store the Google service account keys in a central secret management service.

E. Use gcloud to bind the Kubernetes service account and the Google service account using roles/iam.workloadIdentityUser.

ANSWER: B E

Explanation:

Enable Workload Identity in the cluster via the gcloud command-line tool because Workload Identity Federation for GKE is Google's recommended mechanism for workloads running on GKE to authenticate to Google Cloud APIs. It supplies short-lived credentials to a workload based on its Kubernetes identity, avoiding the operational and security risk of distributing, storing, or rotating downloadable service account keys. After the cluster is configured, each microservice can run under a dedicated Kubernetes service account, supporting least-privilege access separation among services.

Use gcloud to bind the Kubernetes service account and the Google service account using roles/iam.workloadIdentityUser because this IAM binding authorizes the Kubernetes service account to impersonate the corresponding IAM service account. The workload then receives the permissions granted to that IAM service account when it requests Google Cloud resources. In a complete configuration, the Kubernetes service account is also annotated with the IAM service account email address. This design makes application identity explicit, enables granular IAM roles per microservice, and avoids coupling workload permissions to the broad identity of a node. See the GKE Workload Identity Federation documentation and the IAM workload identity federation documentation.

QUESTION NO: 2

For this question, refer to the HipLocal case study.

HipLocal is expanding into new locations. They must capture additional data each time the application is launched in a new European country. This is causing delays in the development process due to constant schema changes and a lack of environments for conducting testing on the application changes. How should they resolve the issue while meeting the business requirements?

A. Create new Cloud SQL instances in Europe and North America for testing and deployment. Provide developers with local MySQL instances to conduct testing on the application changes.

B. Migrate data to Bigtable. Instruct the development teams to use the Cloud SDK to emulate a local Bigtable development environment.

C. Move from Cloud SQL to MySQL hosted on Compute Engine. Replicate hosts across regions in the Americas and Europe. Provide developers with local MySQL instances to conduct testing on the application changes.

D. Migrate data to Firestore in Native mode and set up instances in Europe and North America. Provide developers with a local Firestore emulator to conduct testing on application changes.

ANSWER: D

Explanation:

Migrate data to Firestore in Native mode and set up instances in Europe and North America. Provide developers with a local Firestore emulator to conduct testing on application changes. Firestore in Native mode is a document-oriented database that supports flexible, schemaless data modeling. This lets HipLocal add country-specific fields to documents as new European

launches require them, without performing relational schema migrations for every change. Data can therefore evolve incrementally while the application continues to use a consistent document-based persistence layer.

The Cloud Firestore emulator provides an isolated local development and testing environment. Developers can test reads, writes, security rules, and application behavior without modifying production data or requiring a shared database environment for each change. This shortens the development feedback cycle and enables teams to validate new country-specific data requirements safely before deployment. Selecting appropriate Firestore locations for production supports the organization's geographic deployment needs, while the emulator directly addresses the shortage of test environments. Google documents Firestore's flexible document data model at Cloud Firestore data model and explains local testing with the emulator at Connect your app to the Cloud Firestore Emulator.

QUESTION NO: 3

You are deploying a replicated application on a regional Google Kubernetes Engine (GKE) cluster. The application must remain available during voluntary disruptions, such as node upgrades, and during the loss of a single zone. What should you do? (Choose two.)

A. Create a PodDisruptionBudget for the application.

B. Configure topology spread constraints across zones for the application Pods.

C. Configure all application replicas with node affinity for the same zone.

D. Use a Deployment with a Recreate deployment strategy.

E. Configure a single replica with a high CPU resource limit.

ANSWER: A B

Explanation:

Create a PodDisruptionBudget for the application because a PodDisruptionBudget defines the minimum number or percentage of replicas that must remain available during voluntary disruptions. Kubernetes uses this budget when evicting Pods during actions such as node draining and upgrades, helping prevent all replicas from being removed at the same time.

Configure topology spread constraints across zones because topology spread constraints direct the scheduler to distribute replicas across the available topology domains. Spreading replicas across zones reduces dependence on a single zone and allows replicas in remaining zones to continue serving traffic if one zone fails. See the Kubernetes documentation for PodDisruptionBudgets and topology spread constraints.

QUESTION NO: 4

You are configuring a continuous integration pipeline using Cloud Build to automate the deployment of new container images to Google Kubernetes Engine (GKE). The pipeline builds the application from its source code, runs unit and integration tests in separate steps, and pushes the container to Container Registry. The application runs on a Python web server.

The Dockerfile is as follows:

FROM python:3.7-alpine -

COPY . /app -

WORKDIR /app -

RUN pip install -r requirements.txt

CMD ["gunicorn", "-w 4", "main:app"]

You notice that Cloud Build runs are taking longer than expected to complete. You want to decrease the build time. What should you do? (Choose two.)

A. Select a virtual machine (VM) size with higher CPU for Cloud Build runs.

B. Deploy a Container Registry on a Compute Engine VM in a VPC, and use it to store the final images.

C. Cache the Docker image for subsequent builds using the --cache-from argument in your build config file.

D. Change the base image in the Dockerfile to ubuntu:latest, and install Python 3.7 using a package manager utility.

E. Store application source code on Cloud Storage, and configure the pipeline to use gsutil to download the source code.

ANSWER: A C

Explanation:

Selecting a virtual machine (VM) size with higher CPU for Cloud Build runs can reduce the elapsed time for compute-intensive build operations, including Docker image construction, dependency installation, compilation where applicable, and test execution. Cloud Build supports higher-vCPU machine types through the build configuration. This provides more CPU capacity to steps that can use parallel processing and is an appropriate way to improve build performance without changing the application deployment architecture. Cloud Build notes that nonstandard machine types can have additional startup time, but they can substantially speed up longer-running builds.

Caching the Docker image for subsequent builds using the `--cache-from` argument in the build config file allows Docker to reuse layers from an image produced by an earlier build. Reusing unchanged layers avoids repeating work such as installing dependencies when the relevant inputs have not changed. The pipeline should pull or otherwise make the prior image available, then specify it as a cache source during the Docker build. This is especially useful for CI pipelines that build the same application repeatedly. Google's guidance for improving build performance covers both choosing higher-CPU machine types and using cached Docker images: Increase vCPUs for builds and Speed up Cloud Build builds.

QUESTION NO: 5

You are designing a resource-sharing policy for applications used by different teams in a Google Kubernetes Engine cluster. You need to ensure that all applications can access the resources needed to run. What should you do? (Choose two.)

A. Specify the resource limits and requests in the object specifications.

B. Create a namespace for each team, and attach resource quotas to each namespace.

C. Create a LimitRange to specify the default compute resource requirements for each namespace.

D. Create a Kubernetes service account (KSA) for each application, and assign each KSA to the namespace.

E. Use the Anthos Policy Controller to enforce label annotations on all namespaces. Use taints and tolerations to allow resource sharing for namespaces.

ANSWER: B C

Explanation:

Creating a namespace for each team and attaching ResourceQuotas to each namespace establishes clear resource boundaries for shared cluster use. A ResourceQuota constrains the aggregate amount of compute resources, storage, object counts, or other quota-scoped resources that workloads in a namespace may consume. This prevents one team's workloads from exhausting shared cluster capacity and helps reserve a predictable portion of resources for each team's applications. Namespace-level quotas are therefore a core multi-tenancy control for fair resource allocation.

Creating a LimitRange to specify default compute resource requirements for each namespace complements quotas by applying default resource requests and limits when workload authors omit them. Requests are especially important because the Kubernetes scheduler uses them to place Pods onto nodes with sufficient available capacity. Defaults ensure that workloads participate consistently in resource accounting and make namespace ResourceQuota enforcement practical without requiring every team to manually specify values in every Pod specification. Together, namespace isolation, ResourceQuota controls, and LimitRange defaults provide a reliable policy foundation so teams can share a Google Kubernetes Engine cluster while retaining access to the capacity allocated to their workloads. See the Kubernetes documentation for ResourceQuotas and LimitRanges.

QUESTION NO: 6

You need to deploy resources from your laptop to Google Cloud using Terraform. Resources in your Google Cloud environment must be created using a service account. Your Cloud Identity has the roles/iam.serviceAccountTokenCreator Identity and Access Management (IAM) role and the necessary permissions to deploy the resources using Terraform. You want to set up your development environment to deploy the desired resources following Google-recommended best practices. What should you do?

A. 2) Set the GOOGLE_APPLICATION_CREDENTIALS environment variable to the path of your downloaded key file.

B. 2) Set the GOOGLE_OAUTH_ACCESS_TOKEN environment variable to the value that is returned by the gcloud auth print-access-token command.

C. 2) In the browser window that opens, authenticate using your personal credentials.

D. 2) Integrate Terraform with Vault to retrieve the key file dynamically, and authenticate to Vault using a short-lived access token.

E. Run gcloud auth application-default login --impersonate-service-account=SERVICE_ACCOUNT_EMAIL.

ANSWER: E

Explanation:

Run `gcloud auth application-default login --impersonate-service-account=SERVICE_ACCOUNT_EMAIL`, replacing `SERVICE_ACCOUNT_EMAIL` with the deployment service account. This configures local Application Default Credentials (ADC) so Terraform, through the Google provider, obtains short-lived credentials by impersonating that service account. The user authenticates interactively to Google Cloud, and the Google Cloud CLI uses the granted Service Account Token Creator permission to generate temporary access tokens for the target service account. Terraform therefore creates resources with the service account's identity and permissions, rather than the developer's personal identity.

This approach follows Google's recommended practice of avoiding user-managed service account keys. It removes the need to download, distribute, store, rotate, or protect a long-lived private key on the laptop or in a separate secrets system. ADC also provides a standard credential-discovery mechanism that Terraform supports without embedding a token in configuration or environment variables. Service account impersonation supports least privilege because the developer needs only permission to impersonate the deployment account, while the deployment account retains the resource-management roles required by Terraform. Google documents local ADC authentication with service account impersonation and recommends avoiding service account keys whenever possible. See Set up ADC for local development and Service account impersonation best practices.

QUESTION NO: 7

You are developing an application that will allow users to read and post comments on news articles. You want to configure your application to store and display user-submitted comments using Firestore.

How should you design the schema to support an unknown number of comments and articles?

A. Store each comment in a subcollection of the article.

B. Add each comment to an array property on the article.

C. Store each comment in a document, and add the comment's key to an array property on the article.

D. Store each comment in a document, and add the comment's key to an array property on the user profile.

ANSWER: A

Explanation:

Store each comment in a subcollection of the article. This is the appropriate Firestore data model because an article can have an unbounded and independently growing set of comments. Each article can be represented by a document in an

articles collection, with its comments stored as separate documents in a comments subcollection, such as `articles/{articleId}/comments/{commentId}`. This keeps each comment independently addressable and lets the application query, paginate, order, and update comments without reading or rewriting the article document. It also avoids concentrating an ever-growing amount of comment data in one document, which is important because Firestore documents have a maximum size of 1 MiB. Subcollections are specifically intended for hierarchical data with potentially expanding lists, and they remain queryable even as the number of parent articles and child comments grows. The comment document can contain fields such as author ID, body text, creation timestamp, and moderation status. Firestore security rules can then enforce that users create comments only as themselves and can read comments according to the article's visibility. See the Firestore data model documentation and Google's guidance for structuring Cloud Firestore data.

QUESTION NO: 8

You recently developed a new application. You want to deploy the application on Cloud Run without a Dockerfile. Your organization requires that all container images are pushed to a centrally managed container repository. How should you build your container using Google Cloud services? (Choose two.)

A. Push your source code to Artifact Registry.

B. Submit a Cloud Build job to push the image.

C. Use the pack build command with pack CLI.

D. Include the --source flag with the gcloud run deploy CLI command.

E. Include the --platform=kubernetes flag with the gcloud run deploy CLI command.

ANSWER: B D

Explanation:

Submitting a Cloud Build job to push the image is appropriate because Cloud Build can build a container directly from source using Cloud Native Buildpacks, without requiring a Dockerfile. For example, the build can use the `--pack` option and name an Artifact Registry image as its destination. Cloud Build then builds the image and pushes it to the designated Artifact Registry repository, allowing the organization to centralize image storage, access controls, vulnerability scanning, and retention policies.

Including the `--source` flag with the `gcloud run deploy` CLI command is also appropriate for a source-based Cloud Run deployment. Cloud Run uses Cloud Build and buildpacks to turn the supplied source into a container image when no Dockerfile is provided. The resulting image is stored in Artifact Registry; an image location can be specified to ensure the output is placed in the organization's managed repository. This approach provides a managed build-and-deploy workflow while retaining Artifact Registry as the container-image registry. See the Cloud Run source deployment documentation and Cloud Build buildpacks documentation.

QUESTION NO: 9

You are developing an application that will store and access sensitive unstructured data objects in a Cloud Storage bucket. To comply with regulatory requirements, you need to ensure that all data objects are available for at least 7 years after their initial creation. Objects created more than 3 years ago are accessed very infrequently (less than once a year). You need to configure object storage while ensuring that storage cost is optimized. What should you do? (Choose two.)

A. Set a retention policy on the bucket with a period of 7 years.

B. Use IAM Conditions to provide access to objects 7 years after the object creation date.

C. Enable Object Versioning to prevent objects from being accidentally deleted for 7 years after object creation.

D. Create an object lifecycle policy on the bucket that moves objects from Standard Storage to Archive Storage after 3 years.

E. Implement a Cloud Function that checks the age of each object in the bucket and moves the objects older than 3 years to a second bucket with the Archive Storage class. Use Cloud Scheduler to trigger the Cloud Function on a daily schedule.

ANSWER: A D

Explanation:

Set a retention policy on the bucket with a period of 7 years. A Cloud Storage bucket retention policy establishes a retention period for every object in that bucket. During that period, objects are retained and cannot be deleted or replaced, which supports the requirement to preserve each object for at least seven years from its creation. For regulatory controls that require the policy to be irreversible, the retention policy can additionally be locked after it has been configured and validated. Cloud Storage retention policies apply independently of an object's storage class, so changing an object's class later does not reduce its required retention period.

Create an object lifecycle policy on the bucket that moves objects from Standard Storage to Archive Storage after 3 years. Lifecycle management can automatically change an object's storage class when it reaches a specified age, eliminating the need for custom scheduled code. Archive Storage is intended for data accessed less than once per year and has the lowest storage price among Cloud Storage classes. Thus, retaining objects in Standard Storage for the more active initial three-year period and transitioning them to Archive Storage thereafter aligns storage cost with the stated access pattern while preserving online access. See Cloud Storage retention policies and Object Lifecycle Management.

QUESTION NO: 10

Your App Engine standard configuration is as follows:

service: production instance_class: B1

You want to limit the application to 5 instances.

Which code snippet should you include in your configuration?

A. manual_scaling: instances: 5
min_pending_latency: 30ms

B. manual_scaling: max_instances: 5 idle_timeout: 10m

C. basic_scaling: instances: 5
min_pending_latency: 30ms

D. basic_scaling: max_instances: 5
idle_timeout: 10m

ANSWER: D

Explanation:

The correct configuration is `basic_scaling: max_instances: 5 idle_timeout: 10m`. App Engine standard applications using basic scaling create instances when requests arrive and can shut them down after they have been idle for the configured duration. The `max_instances` setting is the basic-scaling parameter that establishes the upper limit on the number of instances that App Engine may run for that service, so setting it to `5` meets the stated requirement. The `idle_timeout` setting is also part of a valid basic-scaling configuration and specifies how long an idle instance can remain running before App Engine shuts it down. An instance class of `B1` supports basic scaling, making this a valid scaling configuration for the service. Basic scaling is appropriate when the service should scale up in response to incoming work but should not exceed a defined instance cap. Google's App Engine configuration reference documents `max_instances` and `idle_timeout` as the relevant basic-scaling fields: App Engine app.yaml basic_scaling reference. For the behavior and lifecycle of basic-scaled instances, see App Engine scaling types documentation.

QUESTION NO: 11

You are deploying a microservices application to Google Kubernetes Engine (GKE) that will broadcast livestreams. You expect unpredictable traffic patterns and large variations in the number of concurrent users. Your application must meet the following requirements:

• Scales automatically during popular events and maintains high availability

• Is resilient in the event of hardware failures

How should you configure the deployment parameters? (Choose two.)

A. Distribute your workload evenly using a multi-zonal node pool.

B. Distribute your workload evenly using multiple zonal node pools.

C. Use cluster autoscaler to resize the number of nodes in the node pool, and use a Horizontal Pod Autoscaler to scale the workload.

D. Create a managed instance group for Compute Engine with the cluster nodes. Configure autoscaling rules for the managed instance group.

E. Create alerting policies in Cloud Monitoring based on GKE CPU and memory utilization. Ask an on-duty engineer to scale the workload by executing a script when CPU and memory usage exceed predefined thresholds.

ANSWER: A C

Explanation:

Distribute your workload evenly using a multi-zonal node pool. Spreading nodes and Pods across multiple zones provides fault-domain separation: if hardware or infrastructure in one zone becomes unavailable, workloads scheduled in the remaining zones can continue serving viewers. For a highly available GKE deployment, this placement strategy reduces dependence on any single zonal failure domain. Pod topology-spread constraints or appropriate anti-affinity rules can further help ensure replicas are balanced across the available nodes and zones.

Use cluster autoscaler to resize the number of nodes in the node pool, and use a Horizontal Pod Autoscaler to scale the workload. The Horizontal Pod Autoscaler adjusts the number of Pod replicas in response to observed metrics, such as CPU or memory utilization, so the livestream services can react automatically to demand spikes. When the increased Pod count cannot be scheduled because of insufficient node capacity, Cluster Autoscaler adds nodes to an eligible node pool. Conversely, both mechanisms can reduce capacity after demand declines. This coordinated Pod- and node-level autoscaling is the managed GKE approach for unpredictable traffic while retaining the capacity required for available replicas. See GKE Cluster Autoscaler overview and GKE Horizontal Pod Autoscaling.

QUESTION NO: 12

Your company has a new security initiative that requires all data stored in Google Cloud to be encrypted by customer-managed encryption keys. You plan to use Cloud Key Management Service (KMS) to configure access to the keys. You need to follow the "separation of duties" principle and Google-recommended best practices. What should you do? (Choose two.)

A. Provision Cloud KMS in its own project.

B. Do not assign an owner to the Cloud KMS project.

C. Provision Cloud KMS in the project where the keys are being used.

D. Grant the roles/cloudkms.admin role to the owner of the project where the keys from Cloud KMS are being used.

E. Grant an owner role for the Cloud KMS project to a different user than the owner of the project where the keys from Cloud KMS are being used.

ANSWER: A B

Explanation:

“Provision Cloud KMS in its own project.” supports separation of duties by separating the project that administers cryptographic keys from the projects that run workloads and consume those keys. This project boundary enables independent IAM policies: key administrators can manage key rings, keys, rotation, and key IAM without receiving administrative access to the application resources encrypted by those keys. Conversely, workload teams can receive only

the narrowly scoped permissions needed for their services to use a key, such as encryption or decryption, without being able to administer that key.

“Do not assign an owner to the Cloud KMS project.” is also required by the recommended model because the primitive Owner role has broad authority, including the ability to modify IAM policy. A principal with Owner access could grant itself Cloud KMS administrative or key-use permissions, undermining the intended administrative boundary. Instead, assign predefined Cloud KMS roles and project-management permissions deliberately, with different responsible groups for key administration and workload administration. This preserves effective control over who can administer keys and who can use them, while still allowing customer-managed encryption keys to protect data across projects. See Google’s Cloud KMS separation of duties guidance and its Cloud KMS IAM roles documentation.

QUESTION NO: 13

HipLocal’s data science team wants to analyze user reviews.

How should they prepare the data?

A. Use the Cloud Data Loss Prevention API for redaction of the review dataset.

B. Use the Cloud Data Loss Prevention API for de-identification of the review dataset.

C. Use the Cloud Natural Language Processing API for redaction of the review dataset.

D. Use the Cloud Natural Language Processing API for de-identification of the review dataset.

ANSWER: B

Explanation:

Use the Cloud Data Loss Prevention API for de-identification of the review dataset. User reviews can contain personally identifiable information and other sensitive data in free-form text, such as names, email addresses, phone numbers, physical addresses, or financial identifiers. Sensitive Data Protection (the current name for Cloud Data Loss Prevention) can inspect unstructured text using built-in infoType detectors and then de-identify detected findings before the data science team processes the reviews. Its transformations can mask, replace, tokenize, generalize, or otherwise transform sensitive values, enabling analysis while reducing the risk that individual users can be identified. This is appropriate preparation for a review corpus because the resulting dataset retains useful non-sensitive context and can retain transformed representations where analysis requires them. The service supports de-identification for text and structured data, and Google recommends selecting a transformation strategy according to the intended use of the de-identified data and the required privacy protection. For example, replacement or masking can remove direct identifiers from narrative review text before sentiment analysis, categorization, or model training. See the Sensitive Data Protection de-identification documentation and the transformation reference.

QUESTION NO: 14

You are deploying your applications on Compute Engine. One of your Compute Engine instances failed to launch. What should you do? (Choose two.)

A. Determine whether your file system is corrupted.

B. Access Compute Engine as a different SSH user.

C. Troubleshoot firewall rules or routes on an instance.

D. Check whether your instance boot disk is completely full.

E. Check whether network traffic to or from your instance is being dropped.

ANSWER: A D

Explanation:

Determine whether your file system is corrupted and check whether your instance boot disk is completely full. Both conditions can prevent a Compute Engine VM from completing its operating-system boot sequence. A corrupted filesystem can cause disk checks, mount operations, or essential system services to fail during startup. Google recommends using the VM's serial console output and recovery procedures to identify boot-time filesystem errors and, where appropriate, attach the boot disk to a helper VM for repair. A full boot disk can also stop the guest OS from starting normally because startup processes need free space for temporary files, logs, runtime state, and service initialization. Reviewing the serial console is especially useful because it provides boot messages even when normal guest access is unavailable. After identifying insufficient space, free disk capacity through an appropriate recovery workflow; after identifying filesystem damage, repair the filesystem before retrying the boot. These checks address underlying guest OS and boot-disk conditions that directly affect whether the instance can launch successfully. See Google Cloud's VM startup troubleshooting guide and troubleshooting for full disks for supported diagnostic and recovery steps.

QUESTION NO: 15

You are building a mobile application that will store hierarchical data structures in a database. The application will enable users working offline to sync changes when they are back online. A backend service will enrich the data in the database using a service account. The application is expected to be very popular and needs to scale seamlessly and securely. Which database and IAM role should you use?

A. Use Cloud SQL, and assign the roles/cloudsql.editor role to the service account.

B. Use Bigtable, and assign the roles/bigtable.viewer role to the service account.

C. Use Firestore in Native mode and assign the roles/datastore.user role to the service account.

D. Use Firestore in Datastore mode and assign the roles/datastore.viewer role to the service account.

ANSWER: C

Explanation:

Use Firestore in Native mode and assign the roles/datastore.user role to the service account. Cloud Firestore in Native mode is designed for mobile and web applications, using documents and subcollections to represent hierarchical application data naturally. Its client SDKs provide offline persistence: data used by the app is cached locally, reads and writes can continue while disconnected, and locally queued changes synchronize automatically after connectivity returns. This directly supports a mobile-first application in which users may work offline.

Firestore is a fully managed, serverless document database that automatically scales to support growing application demand. Native mode also integrates with Firebase and Cloud Firestore mobile client libraries, which provide the intended synchronization behavior. The backend enrichment service should authenticate with its service account rather than end-user credentials. The Cloud Datastore User role, `roles/datastore.user`, grants the service account permissions to read and write Firestore data, enabling it to enrich documents while applying least privilege relative to broad administrative permissions. Application access should additionally be protected through Firestore Security Rules, while IAM governs the trusted backend service account. See the Firestore offline persistence documentation and the Cloud Datastore and Firestore IAM roles reference.

QUESTION NO: 16

Your security team is auditing all deployed applications running in Google Kubernetes Engine. After completing the audit, your team discovers that some of the applications send traffic within the cluster in clear text. You need to ensure that all application traffic is encrypted as quickly as possible while minimizing changes to your applications and maintaining support from Google. What should you do?

A. Use Network Policies to block traffic between applications.

B. Install Anthos Service Mesh, enable proxy injection on your application namespace, and then enable mTLS.

C. Define Trusted Network ranges within the application, and configure the applications to allow traffic only from those networks.

D. Use an automated process to request SSL Certificates for your applications from Letâ€™s Encrypt and add them to your applications.

ANSWER: B

Explanation:

Install Anthos Service Mesh, enable proxy injection on your application namespace, and then enable mTLS. Anthos Service Mesh provides a Google-supported service mesh based on Istio that transparently adds Envoy sidecar proxies to workloads. Once sidecar injection is enabled, the mesh can establish mutual TLS between participating services, encrypting service-to-service traffic in transit inside the cluster and authenticating workload identities. This approach is especially appropriate when rapid adoption and minimal application modification are required: the proxies handle certificate provisioning, rotation, encryption, and traffic policy outside the application code. A mesh-wide strict mTLS policy can be applied after workloads participate in the mesh, ensuring that services communicate over encrypted connections rather than plaintext. Google documents Anthos Service Mesh security capabilities, including automatic mutual TLS for workloads in the mesh, at Anthos Service Mesh security overview. The deployment process and supported sidecar injection configuration are described in the Anthos Service Mesh installation documentation. This provides encryption at the application-traffic layer while preserving the existing application protocols and avoiding the need to add and maintain TLS implementation logic in every service.

QUESTION NO: 17

Your company just experienced a Google Kubernetes Engine (GKE) API outage due to a zone failure. You want to deploy a highly available GKE architecture that minimizes service interruption to users in the event of a future zone failure. What should you do?

A. Deploy Zonal clusters

B. Deploy Regional clusters

C. Deploy Multi-Zone clusters

D. Deploy GKE on-premises clusters

ANSWER: B

Explanation:

Deploy Regional clusters. A regional GKE cluster replicates the control plane across multiple zones in a single Google Cloud region, rather than locating it in only one zone. This design maintains control-plane availability if a zone fails, helping ensure that the Kubernetes API and cluster-management operations remain available during a zonal outage. Regional clusters are therefore the appropriate GKE architecture when the primary concern is avoiding a repeat of an API outage caused by loss of one zone.

For workload availability, GKE can also distribute node pools across multiple zones in the regional cluster's region. GKE uses three zones by default for node pools in a regional cluster, allowing application replicas to be scheduled across independent failure domains. Applications should be configured with multiple replicas and suitable workload spreading rules so that healthy pods remain available outside the failed zone. This combination of a replicated regional control plane and multi-zone worker nodes minimizes user-facing interruption from an individual zone failure. Google documents regional clusters as the highly available configuration because the control plane has multiple replicas distributed across zones. See GKE cluster types: Regional clusters and GKE regional clusters overview.

QUESTION NO: 18

You are using the Cloud Client Library to upload an image in your application to Cloud Storage. Users of the application report that occasionally the upload does not complete and the client library reports an HTTP 504 Gateway Timeout error. You want to make the application more resilient to errors. What changes to the application should you make?

A. Write an exponential backoff process around the client library call.

B. Write a one-second wait time backoff process around the client library call.

C. Design a retry button in the application and ask users to click if the error occurs.

D. Create a queue for the object and inform the users that the application will try again in 10 minutes.

ANSWER: A

Explanation:

Write an exponential backoff process around the client library call. An HTTP 504 Gateway Timeout is generally a transient failure: the request may not have completed because a gateway or upstream service did not respond in time. Retrying transient Cloud Storage failures with exponential backoff makes the application resilient while avoiding a burst of immediate repeat requests that can worsen temporary service or network congestion. Each successive retry should wait longer, typically with randomized jitter, and the process should have a bounded number of attempts or overall deadline.

Cloud Storage documents exponential backoff as its recommended retry strategy, and identifies HTTP 504 among the retryable response codes. Where an application explicitly controls upload retries, it should also use the relevant object-generation precondition when appropriate, so an ambiguous timeout can be retried safely without unintentionally overwriting an object. Google Cloud client libraries include retry behavior for supported operations, but an application-level retry policy can be used when custom resilience behavior is required. See the Cloud Storage retry strategy documentation and the Cloud Storage request preconditions documentation.

QUESTION NO: 19

You are planning to deploy your application in a Google Kubernetes Engine (GKE) cluster. Your application can scale horizontally, and each instance of your application needs to have a stable network identity and its own persistent disk.

Which GKE object should you use?

A. Deployment

B. StatefulSet

C. ReplicaSet

D. ReplicaController

ANSWER: B

Explanation:

StatefulSet is designed for stateful workloads whose individual replicas require a persistent, distinct identity. It gives each Pod a stable ordinal name and network identity, such as a predictable hostname, which remains associated with that replica through rescheduling. This allows application members to be addressed reliably and supports workloads that need to recognize or communicate with a specific peer.

A StatefulSet can also use *volumeClaimTemplates* to create a separate PersistentVolumeClaim for every Pod replica. Each replica therefore receives its own persistent disk, and the claim stays associated with that replica even if its Pod is deleted and recreated. When the workload is scaled horizontally, Kubernetes provisions the additional replicas in an ordered, predictable way and creates the corresponding storage claims. These characteristics make StatefulSet the appropriate GKE workload object for databases, clustered services, and similar applications requiring stable per-instance networking and storage. See the Kubernetes documentation for StatefulSets and Google Cloud guidance on deploying stateful applications on GKE.

QUESTION NO: 20

You have decided to migrate your Compute Engine application to Google Kubernetes Engine. You need to build a container image and push it to Artifact Registry using Cloud Build. What should you do?

(Choose two.)

A. Run gcloud builds submit in the directory that contains the application source code.

B. Run gcloud run deploy app-name --image gcr.io/$PROJECT_ID/app-name in the directory that contains the application source code.

C. Run gcloud container images add-tag gcr.io/$PROJECT_ID/app-name gcr.io/$PROJECT_ID/app-name:latest in the directory that contains the application source code.

D. In the application source directory, create a file named cloudbuild.yaml that contains the following contents:

E. In the application source directory, create a file named cloudbuild.yaml that contains the following contents:

ANSWER: A D

Explanation:

To build and publish the application container through Cloud Build, define the build in `cloudbuild.yaml` and submit the source directory to Cloud Build with `gcloud builds submit`. The Cloud Build configuration supplies the Docker build step, including the target Artifact Registry image name, and declares that image for publishing. When the build succeeds, Cloud Build pushes the tagged image to the specified Artifact Registry Docker repository, assuming the repository exists and the Cloud Build service account has the required Artifact Registry write permissions.

Running `gcloud builds submit` from the directory containing the application source uploads that directory as the build context and automatically uses a `cloudbuild.yaml` file in that directory unless another configuration file is specified. This makes the Dockerfile and source files available to the Docker builder step and executes the image build remotely. Artifact Registry image references use the `LOCATION-docker.pkg.dev/PROJECT_ID/REPOSITORY/IMAGE` format, which should be present in the build configuration. See the Cloud Build guide for building and pushing Docker images and the Artifact Registry Docker image documentation.

QUESTION NO: 21

You work at a rapidly growing financial technology startup. You manage the payment processing application written in Go and hosted on Cloud Run in the Singapore region (asia-southeast1). The payment processing application processes data stored in a Cloud Storage bucket that is also located in the Singapore region.

The startup plans to expand further into the Asia Pacific region. You plan to deploy the Payment Gateway in Jakarta, Hong Kong, and Taiwan over the next six months. Each location has data residency requirements that require customer data to reside in the country where the transaction was made. You want to minimize the cost of these deployments. What should you do?

A. Create a Cloud Storage bucket in each region, and create a Cloud Run service of the payment processing application in each region.

B. Create a Cloud Storage bucket in each region, and create three Cloud Run services of the payment processing application in the Singapore region.

C. Create three Cloud Storage buckets in the Asia multi-region, and create three Cloud Run services of the payment processing application in the Singapore region.

D. Create three Cloud Storage buckets in the Asia multi-region, and create three Cloud Run revisions of the payment processing application in the Singapore region.

ANSWER: A

Explanation:

Create a Cloud Storage bucket in each region, and create a Cloud Run service of the payment processing application in each region. This design keeps each transaction's customer data in the Google Cloud region corresponding to the applicable country: Jakarta for Indonesia, Hong Kong, and Taiwan. A regional Cloud Storage bucket stores its data redundantly within the selected region, supporting a deployment model in which data is retained locally to satisfy the stated residency requirement.

Deploying the Cloud Run service alongside its regional bucket also minimizes operational and network cost. The payment service reads and processes data in the same region rather than regularly accessing a bucket across regional boundaries.

This avoids unnecessary cross-region data-transfer patterns and improves latency for payment processing. Cloud Run supports the relevant Asia Pacific regions, including Jakarta, Hong Kong, and Taiwan, so independent regional services can use the same Go application image and configuration pattern while handling data locally. The existing Singapore deployment can remain aligned with its Singapore bucket under the same architecture.

Google documents the available Cloud Run regional endpoints at Cloud Run locations and describes regional Cloud Storage bucket placement at Cloud Storage bucket locations.

QUESTION NO: 22

You recently developed an application. You need to call the Cloud Storage API from a Compute Engine instance that doesn't have a public IP address. What should you do?

A. Use Carrier Peering

B. Use VPC Network Peering

C. Use Shared VPC networks

D. Use Private Google Access

ANSWER: D

Explanation:

Use Private Google Access. This feature enables VM instances that have only internal IP addresses to reach Google APIs and services, including the Cloud Storage API, without assigning an external IP address to the instance. Private Google Access is configured on the subnet used by the Compute Engine VM. Once enabled, eligible traffic to Google APIs can be sent from the instance through Google's network rather than requiring direct public internet connectivity.

The application should authenticate using the Compute Engine instance's attached service account, and that service account must be granted the required Cloud Storage IAM permissions, such as permissions provided through an appropriate Storage role. Private Google Access addresses network reachability; IAM and service-account credentials continue to control authorization for the API request. This design avoids exposing the VM through a public IPv4 address while still allowing it to consume supported Google APIs. For workloads that need a constrained egress path, Google also supports private Google access options using restricted or private API virtual IP ranges, depending on the required API access model and organization security controls.

See Private Google Access documentation and Cloud Storage IAM documentation.

QUESTION NO: 23

Your application consists of services running on Google Kubernetes Engine and Cloud Run. You need to identify latency introduced by individual service calls for a request that crosses both platforms. What should you do? (Choose two.)

A. Instrument the services by using OpenTelemetry to create and export traces.

B. Propagate trace context with each service-to-service request.

C. Create a log sink that exports all logs to a Cloud Storage bucket.

D. Enable VPC Flow Logs for every subnet used by the services.

E. Configure Cloud Profiler to capture heap dumps for each request.

ANSWER: A B

Explanation:

Instrument the services by using OpenTelemetry to create and export traces. OpenTelemetry provides vendor-neutral APIs and SDKs for generating trace spans around request handling and dependency calls. Exporting spans to Cloud Trace makes the distributed request path and latency for individual operations available for analysis.

Propagate trace context with each service-to-service request. Trace context links spans created by separate services into one distributed trace. Without propagating this context, Cloud Trace can show individual spans but cannot reliably represent the complete request path across the GKE and Cloud Run services. See the Cloud Trace instrumentation documentation and the OpenTelemetry observability documentation.

QUESTION NO: 24

You want to use the Stackdriver Logging Agent to send an application's log file to Stackdriver from a Compute Engine virtual machine instance.

After installing the Stackdriver Logging Agent, what should you do first?

A. Enable the Error Reporting API on the project.

B. Grant the instance full access to all Cloud APIs.

C. Configure the application log file as a custom source.

D. Create a Stackdriver Logs Export Sink with a filter that matches the application's log entries.

ANSWER: C

Explanation:

After the Stackdriver Logging Agent is installed, configure the application log file as a custom source. Installation alone configures the agent software and its standard inputs, but the agent must be told the path and format of an application-specific log file before it can collect and send those entries to Cloud Logging. For the legacy Logging agent, this is done by adding or editing a Fluentd configuration file in the agent configuration directory, typically under `/etc/google-fluentd/config.d/`, with a source that tails the target file. The agent is then restarted so it loads the new configuration and begins forwarding newly written log records.

Custom log configuration should identify the exact log path and, where needed, define parsing behavior so entries are represented meaningfully in Cloud Logging. The VM also needs appropriate logging permissions, but that is an access prerequisite rather than the first configuration action that makes the specified application file a collected source. Google now recommends the Ops Agent for new deployments, but the custom-log principle remains the same: explicitly configure the files the agent should ingest. See the legacy Logging agent custom-log configuration documentation and the Ops Agent configuration guide.

QUESTION NO: 25

Your organization deploys containerized applications to Google Kubernetes Engine (GKE). You need to identify known vulnerabilities in images and prevent deployments of images that have not passed the organization’s required security verification. What should you do? (Choose two.)

A. Store images in Artifact Registry, and enable vulnerability scanning.

B. Configure Binary Authorization to require attestations before images are deployed to GKE.

C. Configure Cloud Logging to delete image-push logs when a vulnerability is detected.

D. Grant the GKE node service account Artifact Registry Administrator on the project.

E. Use a Kubernetes CronJob to scan each image after it has been deployed.

ANSWER: A B

Explanation:

Store images in Artifact Registry and enable vulnerability scanning. Artifact Analysis scans supported container images stored in Artifact Registry and produces vulnerability findings for known operating-system package vulnerabilities. Development and security teams can review these findings as part of their build and release process.

Configure Binary Authorization to enforce a deployment policy for GKE. Binary Authorization can require an image to have a valid attestation before it is admitted to a GKE cluster. An attestor can represent the organization's required verification process, such as an approved vulnerability review or successful security gate in the CI/CD pipeline. Together, image scanning provides vulnerability information and Binary Authorization enforces that only verified images are deployed. See Artifact Analysis for Artifact Registry and the Binary Authorization overview.

QUESTION NO: 26

You are building a telemetry-processing service with Pub/Sub. Events for the same device must be processed in the order in which they were generated, but events for different devices should be processed in parallel. What should you do? (Choose two.)

A. Set the device identifier as the ordering key when publishing each message.

B. Enable message ordering on the Pub/Sub subscription.

C. Create one Pub/Sub topic for every device.

D. Use a push subscription so Pub/Sub can preserve global message order.

E. Set the message retention duration to the maximum supported value.

ANSWER: A B

Explanation:

Set the device identifier as the ordering key when publishing each message. Pub/Sub ordering is scoped to an ordering key, so messages published with the same device identifier are delivered in publication order for that key. Using a different key for each device allows messages from separate devices to proceed independently.

Enable message ordering on the subscription used by the processing service. Subscription-level ordering must be enabled for ordered delivery to subscribers. The subscriber should acknowledge each message successfully before processing the next message for the same ordering key. Ordering does not provide exactly-once business processing, so the consumer should also be designed to handle redelivery safely. See Pub/Sub message ordering.

QUESTION NO: 27

You are developing a marquee stateless web application that will run on Google Cloud. The rate of the incoming user traffic is expected to be unpredictable, with no traffic on some days and large spikes on other days. You need the application to automatically scale up and down, and you need to minimize the cost associated with running the application. What should you do?

A. Build the application in Python with Firestore as the database. Deploy the application to Cloud Run.

B. Build the application in C# with Firestore as the database. Deploy the application to App Engine flexible environment.

C. Build the application in Python with CloudSQL as the database. Deploy the application to App Engine standard environment.

D. Build the application in Python with Firestore as the database. Deploy the application to a Compute Engine managed instance group with autoscaling.

ANSWER: A

Explanation:

Build the application in Python with Firestore as the database. Deploy the application to Cloud Run. Cloud Run is designed for stateless HTTP applications and automatically adjusts the number of container instances according to incoming request volume. It can rapidly add instances during traffic spikes and, with the default minimum-instance setting of zero, can scale down to zero instances when there is no traffic. This directly addresses an unpredictable workload while avoiding charges for continuously provisioned application instances during idle periods. Python is supported as a Cloud Run runtime through a container image or Google buildpacks, so it is suitable for this deployment model.

Firestore complements this design because it is a fully managed, serverless document database that scales automatically with demand. Its pricing is primarily based on operations, stored data, and network usage rather than maintaining a continuously running database server. Combining request-driven Cloud Run scaling with Firestore therefore provides an operationally simple architecture that can accommodate both zero-traffic periods and sudden spikes while aligning infrastructure cost with actual use. See the Cloud Run instance autoscaling documentation and the Firestore pricing documentation.

QUESTION NO: 28

Your team is writing a backend application to implement the business logic for an interactive voice response (IVR) system that will support a payroll application. The IVR system has the following technical characteristics:

• Each customer phone call is associated with a unique IVR session.

• The IVR system creates a separate persistent gRPC connection to the backend for each session.

• If the connection is interrupted, the IVR system establishes a new connection, causing a slight latency for that call.

You need to determine which compute environment should be used to deploy the backend application. Using current call data, you determine that:

• Call duration ranges from 1 to 30 minutes.

• Calls are typically made during business hours.

• There are significant spikes of calls around certain known dates (e.g., pay days), or when large payroll changes occur.

You want to minimize cost, effort, and operational overhead. Where should you deploy the backend application?

A. Compute Engine

B. Google Kubernetes Engine cluster in Standard mode

C. Cloud Functions

D. Cloud Run

ANSWER: D

Explanation:

Cloud Run is the appropriate deployment environment because it combines managed, request-driven scaling with support for gRPC, including streaming RPCs. Each IVR call can hold an active gRPC request for the duration of its session, while Cloud Run automatically adds instances as concurrent calls increase during payroll deadlines or other predictable surges. Conversely, when call volume declines outside business hours, the service can scale down, avoiding the cost and operational work of continuously managing idle virtual machines or Kubernetes nodes.

The service request timeout must be configured to accommodate the longest expected call. Cloud Run supports request timeouts up to 60 minutes, so a 30-minute IVR session fits within that limit. The IVR client should also be designed to reconnect, as described in the scenario, because Cloud Run instances can be shut down or replaced and a client connection may consequently be interrupted. Cloud Run is therefore a strong fit where brief reconnection latency is acceptable and the application benefits from fully managed infrastructure, autoscaling, and pay-per-use behavior. Google documents gRPC and streaming support in Using gRPC with Cloud Run, and documents the configurable request-duration limit in Setting request timeout.

QUESTION NO: 29

Your company's development teams want to use Cloud Build in their projects to build and push Docker images to Container Registry. The operations team requires all Docker images to be published to a centralized, securely managed Docker registry that the operations team manages.

What should you do?

A. Use Container Registry to create a registry in each development team's project. Configure the Cloud Build build to push the Docker image to the project's registry. Grant the operations team access to each development team's registry.

B. Create a separate project for the operations team that has Container Registry configured. Assign appropriate permissions to the Cloud Build service account in each developer team's project to allow access to the operation team's registry.

C. Create a separate project for the operations team that has Container Registry configured. Create a Service Account for each development team and assign the appropriate permissions to allow it access to the operations team's registry. Store the service account key file in the source code repository and use it to authenticate against the operations team's registry.

D. Create a separate project for the operations team that has the open source Docker Registry deployed on a Compute Engine virtual machine instance. Create a username and password for each development team. Store the username and password in the source code repository and use it to authenticate against the operations team's Docker registry.

ANSWER: B

Explanation:

Create a separate project for the operations team that has Container Registry configured. Assign appropriate permissions to the Cloud Build service account in each developer team's project to allow access to the operation team's registry. This establishes a central ownership boundary: the operations project owns the registry and can consistently manage repository access, image retention, auditing, and organization-wide controls, while development projects retain responsibility for their own builds. Cloud Build runs using a service account, so granting that identity the required write permissions on the storage resources backing the centralized Container Registry enables builds in each development project to push images across project boundaries without distributing credentials. This follows the Google Cloud IAM model of granting least-privilege access directly to workload identities rather than relying on manually managed secrets. For this legacy Container Registry scenario, access was controlled through IAM permissions on the underlying Cloud Storage bucket; Google documents the relevant Container Registry access-control model at Container Registry access control. Container Registry has since been deprecated and shut down for writes, so a current implementation should use Artifact Registry with the same centralized-project and cross-project service-account access pattern. Google's migration guidance is available at Transition from Container Registry.

QUESTION NO: 30

Your company wants to expand their users outside the United States for their popular application. The company wants to ensure 99.999% availability of the database for their application and also wants to minimize the read latency for their users across the globe.

Which two actions should they take? (Choose two.)

A. Create a multi-regional Cloud Spanner instance with "nam-eur-asia1" configuration.

B. Create a multi-regional Cloud Spanner instance with "nam3" configuration.

C. Create a cluster with at least 3 Spanner nodes.

D. Create a cluster with at least 1 Spanner node.

E. Create a minimum of two Cloud Spanner instances in separate regions with at least one node.

F. Create a Cloud Dataflow pipeline to replicate data across different databases.

ANSWER: A C

Explanation:

A multi-regional Cloud Spanner instance using the `nam-eur-asia1` configuration is designed for applications serving users across North America, Europe, and Asia. This configuration places voting and read-only replicas across geographically distributed locations, allowing strongly consistent reads to be served near users where possible and reducing global read latency. Multi-region configurations are also engineered for high availability: Cloud Spanner provides a 99.999% availability service-level objective for multi-region instances, making this deployment model appropriate for the stated availability target.

Provisioning at least three Spanner nodes is required for a multi-region instance because the configuration must have sufficient compute capacity to run its replicated database topology. In the node-based provisioning model used by this question, three nodes is the minimum for a multi-region configuration. These nodes provide the capacity needed for the database to process requests while maintaining the quorum-based replication architecture that supports regional fault tolerance and the multi-region availability objective. Capacity can then be increased as workload demand grows without changing the geographic database design. See the Cloud Spanner instance configuration documentation and Cloud Spanner SLA.

QUESTION NO: 31

You use Cloud Deploy to release an application from a staging Google Kubernetes Engine cluster to a production cluster. All releases must first be deployed to staging. A designated approver must explicitly authorize each promotion to production. What should you do? (Choose two.)

A. Create a Cloud Deploy delivery pipeline with staging and production targets.

B. Require approval on the production target.

C. Configure the production target as the first target in the delivery pipeline.

D. Grant all developers the Kubernetes Engine Admin role on the production cluster.

E. Use Cloud Scheduler to run kubectl apply against the production cluster after every build.

ANSWER: A B

Explanation:

Create a Cloud Deploy delivery pipeline with staging and production targets. A delivery pipeline defines the promotion sequence for releases, and targets identify the deployment environments. Configuring staging before production ensures that releases follow the required progression.

Require approval on the production target. Cloud Deploy approvals pause a rollout before deployment to a target until an authorized principal approves or rejects it. This allows the designated approver to control production promotion while retaining an automated and auditable delivery process. See the Cloud Deploy delivery pipelines documentation and Cloud Deploy approvals and promotions documentation.

QUESTION NO: 32

You are deploying a web application to Google Kubernetes Engine (GKE). The application requires several seconds to initialize and can occasionally enter a state in which it continues running but cannot process requests. You need to ensure that traffic is sent only to ready Pods and that unhealthy Pods are restarted automatically. What should you do? (Choose two.)

A. Configure a readiness probe for the application container.

B. Configure a liveness probe for the application container.

C. Configure a Horizontal Pod Autoscaler based on CPU utilization.

D. Configure a PodDisruptionBudget with a minimum available value of one.

E. Configure a startup script on each GKE node to verify the application endpoint.

ANSWER: A B

Explanation:

Configure a readiness probe to verify that the application is ready to serve requests. Kubernetes removes a Pod from Service endpoints when its readiness probe fails. This prevents a newly started Pod from receiving traffic before initialization completes and stops traffic from reaching a Pod that is temporarily unable to serve requests.

Configure a liveness probe to verify that the application is still functioning. When a liveness probe fails repeatedly, the kubelet restarts the container. This is appropriate for a process that remains running but is deadlocked or otherwise unable to recover on its own. Readiness and liveness probes serve different purposes and should be configured independently. See the Kubernetes documentation for liveness, readiness, and startup probes.

QUESTION NO: 33

You are developing a microservice-based application that will run on Google Kubernetes Engine (GKE). Some of the services need to access different Google Cloud APIs. How should you set up authentication of these services in the cluster following Google-recommended best practices? (Choose two.)

A. Use the service account attached to the GKE node.

B. Enable Workload Identity in the cluster via the gcloud command-line tool.

C. Access the Google service account keys from a secret management service.

D. Store the Google service account keys in a central secret management service.

E. Use gcloud to grant the Kubernetes service account `roles/iam.workloadIdentityUser` on the Google service account and annotate the Kubernetes service account with that Google service account.

ANSWER: B E

Explanation:

Enable Workload Identity in the cluster via the gcloud command-line tool. Workload Identity Federation for GKE is Google's recommended approach for workloads that need to call Google Cloud APIs because it lets Pods obtain short-lived credentials through their Kubernetes service account identity, rather than distributing, storing, or rotating downloadable service account keys. For a microservice architecture, create distinct Kubernetes service accounts for services with different access requirements and map each one to an appropriately privileged Google service account. This supports least-privilege access and limits the impact of a compromised workload.

Use gcloud to grant the Kubernetes service account the `roles/iam.workloadIdentityUser` role on the Google service account, and annotate the Kubernetes service account with that Google service account's email address. The IAM role binding authorizes the Kubernetes principal to impersonate the Google service account; the annotation establishes the identity mapping used by Pods. Grant the Google service account only the roles required for the APIs and resources used by that particular microservice. For configuration steps and commands, see Use Workload Identity Federation for GKE and Workload Identity Federation with Kubernetes.

QUESTION NO: 34

Your development team has been tasked with maintaining a .NET legacy application. The application incurs occasional changes and was recently updated. Your goal is to ensure that the application provides consistent results while moving through the CI/CD pipeline from environment to environment. You want to minimize the cost of deployment while making sure that external factors and dependencies between hosting environments are not problematic. Containers are not yet approved in your organization. What should you do?

A. Rewrite the application using .NET Core, and deploy to Cloud Run. Use revisions to separate the environments.

B. Use Cloud Build to deploy the application as a new Compute Engine image for each build. Use this image in each environment.

C. Deploy the application using MS Web Deploy, and make sure to always use the latest, patched MS Windows Server base image in Compute Engine.

D. Use Cloud Build to package the application, and deploy to a Google Kubernetes Engine cluster. Use namespaces to separate the environments.

ANSWER: B

Explanation:

Use Cloud Build to deploy the application as a new Compute Engine image for each build. Use this image in each environment. This approach applies an immutable-infrastructure deployment model that is well suited to a legacy .NET application when containers are not available. The CI process produces a versioned machine image containing the application, its required runtime, configuration baseline, and operating-system dependencies. That exact tested artifact can then be promoted through development, test, staging, and production without rebuilding or manually modifying the application on each target environment.

Using the same image across environments minimizes configuration drift and removes environmental discrepancies caused by inconsistent installed software, patches, or deployment steps. It also supports dependable rollback: a deployment can return to a previously validated image version. Compute Engine machine images are reusable resources for creating VM instances, while Cloud Build can orchestrate the repeatable build and deployment workflow. This is a pragmatic rehost-oriented modernization step for a legacy application: it improves deployment consistency and operational reliability without requiring an immediate rewrite or a container platform adoption. Google's .NET modernization guidance discusses rehosting existing .NET workloads on Compute Engine, and Compute Engine documents machine images as a way to capture and reuse VM configuration and disks. See Modernization path for .NET applications and Compute Engine machine images.