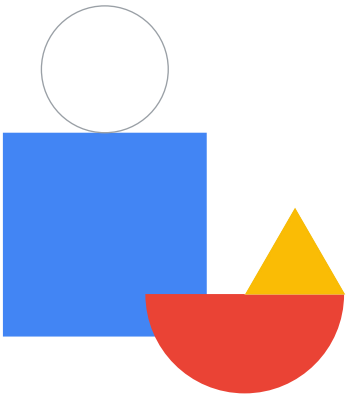
Google Cloud

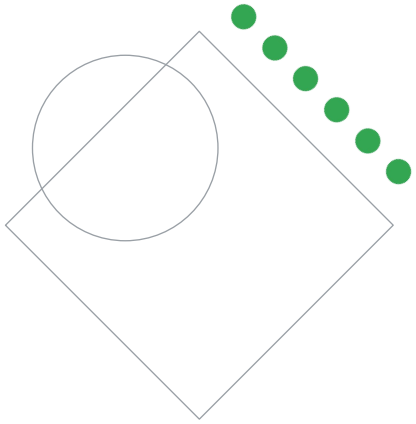


Preparing for Your Professional Cloud Architect Journey

Module 4: Analyzing and Optimizing Technical and Business Processes

Welcome to Module 4: Analyzing and Optimizing Technical and Business Processes.

Review and study planning



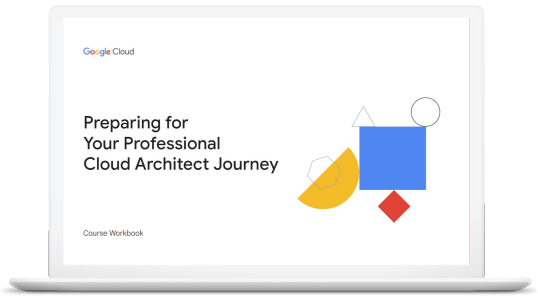
Google Cloud

Now let’s review how to use these diagnostic questions to help you identify what to include in your study plan.

As a reminder - this course isn’t designed to teach you everything you need to know for the exam - and the diagnostic questions don’t cover everything that could be on the exam. Instead, this activity is meant to give you a better sense of the scope of this section and the different skills you’ll want to develop as you prepare for the certification.

Your study plan:

Analyzing and optimizing technical and business processes



4.1	Analyzing and defining technical processes
4.2	Analyzing and defining business processes
4.3	Developing procedures to ensure reliability of solutions in production

Google Cloud

We'll approach this review by looking at the objectives of this exam section and the questions you just answered about each one. We'll introduce an objective, briefly review the answers to the related questions, then talk about where you can find out more in the learning resources and/or in Google Cloud documentation. As we go through each section objective, use the page in your workbook to mark the specific documentation, courses (and modules!), and quests you'll want to emphasize in your study plan.

4.1 Analyzing and defining technical processes

Considerations include:

- Software development life cycle (SDLC)
- Continuous integration / continuous deployment
- Troubleshooting / root cause analysis best practices
- Testing and validation of software and infrastructure
- Service catalog and provisioning
- Business continuity and disaster recovery

Google Cloud

In the architecture design process, a Professional Cloud Architect needs to consider the current technical process and what is desired in the future. For example, if you are in an environment with separate ops and dev teams, you will need a different type of process than an environment that uses SRE and DevOps practices. SRE and DevOps processes may not be what you use currently, but they are at the heart of Google's approach and you should be familiar with these concepts.

Question 1 tested your knowledge of automating infrastructure on Google Cloud with Terraform. Question 2 explored the use of CI/CD pipelines to automate service deployment. Question 3 tested your knowledge of designing services to meet requirements for availability, durability, and scalability. Question 4 asked you to identify steps to use Cloud Source Repositories for source and version control. Question 5 tested your knowledge of using Artifact Registry to manage containers. Question 6 explored your knowledge of recommended design patterns for disaster recovery. Question 7 tested your knowledge of using a circuit breaker and truncated exponential backoff design patterns. Question 8 tested your ability to implement fault-tolerant systems by avoiding single points of failure, correlated failures, and cascading failures. Question 9 asked you to select a deployment pattern for rolling out updates.

4.1 Diagnostic Question 01 Discussion

You are asked to implement a lift and shift operation for Cymbal Direct's Social Media Highlighting service. You **compose a Terraform configuration file** to build all the necessary Google Cloud resources.

What is the next step in the Terraform workflow for this effort?

A. **Commit the configuration file** to your software repository.
B. Run **terraform plan** to verify the contents of the Terraform configuration file.
C. Run **terraform apply** to deploy the resources described in the configuration file.
D. Run **terraform init** to download the necessary provider modules.

Google Cloud

Feedback:
A. Incorrect. You should run init and run plan on your Terraform workflow before you commit the validated configuration file to your software repository.
B. Incorrect. You should run the init command before you run the plan command.
C. Incorrect. You should run init your providers and test your configuration with a plan before you run apply to allocate or change your resources.
D. Correct! Running init in the directory containing your Terraform configuration file ensures that the correct plugins are loaded for the providers and resources requested.

Where to look:
https://cloud.google.com/docs/terraform

Content mapping:

- Architecting with Google Compute Engine (ILT)
  - M10 Infrastructure Automation

- Elastic Google Cloud Infrastructure: Scaling and Automation (On-demand)
  - M3 Infrastructure Automation

Summary:
Automation tools like Terraform let you implement and manage infrastructure as code. Treating your Google Cloud resources as software lets you deploy, update, and destroy a stack of resources in a repeatable way. Terraform offers a declarative

model used to deploy and manage resources. Imperative methods specify exactly how you want a resource configured. Declarative methods, however, let you specify what you want deployed and aren't as concerned with lower-level details. You must write commands on the command line. Referencing an entire stack of resources you want deployed in a configuration file and calling it once is considered declarative.

A Terraform workflow starts with authoring your TensorFlow configuration file. A Terraform provider specifies the cloud environment used. The available providers for Google Cloud are "Google" or "Google-beta." Resources entries in the configuration file specify what resources you want deployed and let you provide arguments to define their operating characteristics. After you write your configuration file, run the Terraform Init command in the development directory your configuration file is in. This downloads the provider modules required by your configuration file. To verify that there are no mistakes in the config file, you can run a Terraform plan command. You can make any needed changes before deploying your infrastructure. Finally, you run a Terraform apply command to deploy your infrastructure.

4.1 Diagnostic Question 02 Discussion

You have implemented a manual **CI/CD process for the container services** required for the next implementation of the Cymbal Direct's Drone Delivery project. You want to **automate the process**.

What should you do?

A. **Implement and reference a source repository** in your Cloud Build configuration file.

B. **Implement a build trigger** that applies your build configuration when a new software update is committed to Cloud Source Repositories.

C. **Specify the name** of your Container Registry in your Cloud Build configuration.

D. **Configure and push a manifest fil**e into an environment repository in Cloud Source Repositories.

Google Cloud

Feedback:
A. Incorrect. You can reference a source repository in your Cloud Build configuration, but the build won't be automated unless you implement a build trigger.
B. Correct! Configuring a build trigger automates the CI/CD process based on when the software is posted to a repository.
C. Incorrect. The Container Registry specifies where Cloud Build should post the containers it builds.
D. Incorrect. The question asks about automating this process. You would need to configure a build trigger to push the manifest placed in your environment repository to Kubernetes to automate the process.

Where to look:
https://cloud.google.com/source-repositories/docs/features#connected_repositories

Content mapping:

- Architecting with Google Cloud: Design and Process (ILT)
  - M3 DevOps Automation

- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M3 DevOps Automation

Summary:
Continuous integration/continuous delivery (CI/CD) pipelines automate the testing and delivery of code by monitoring a controlled software repository. When new software is

checked in, the pipeline orchestrator will first run unit tests. If the tests are successful, a deployment package is built and saved to a Container Registry, which completes the steps for continuous integration. Continuous delivery deploys your images or artifacts to the operational environment you specify, such as App Engine or GKE.

With Google Cloud, you can use Cloud Source Repositories as your version control repository. Unit testing and container artifacts can be produced through Cloud Build and automated through build triggers. Common places to save and manage your finalized images include Container Registry and Artifact Registry. Continuous Delivery can also be implemented through steps specified in Cloud Build and implementing build triggers to apply build operations when a new manifest is added to a monitored candidate repository.

Cloud Build handles building, testing, and deploying your application logic through a build configuration. This build configuration is executed by cloud builders, which are container instances with a common set of tools loaded on them. Provided builders include curl, docker, gcloud CLI, gsutil, git, and gke-deploy. You can also implement your own cloud builder.

4.1 Diagnostic Question 03 Discussion

You have an application implemented on **Compute Engine.** You want to **increase the durability** of your application.

What should you do?

A. Implement a **scheduled snapshot** on your Compute Engine instances.
B. Implement a **regional managed instance group.**
C. Monitor your application's usage metrics and implement **autoscaling**.
D. Perform **health checks** on your Compute Engine instances.

Google Cloud

Feedback:
A. Correct! Durability ensures that your data is protected and available. Snapshots are a viable way of backing up your data in Compute Engine.
B. Incorrect. Managed instance groups improve availability, not durability.
C. Incorrect. Autoscaling is a method for implementing scalable applications, not durability.
D. Incorrect. Health checks provide information about availability, not durability.

Where to look:
https://cloud.google.com/architecture/framework/reliability

Content mapping:
- Architecting with Google Cloud: Design and Process (ILT)
  - M7 Designing Reliable Systems
- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M7 Designing Reliable Systems

Summary:
Availability, durability, and scalability are design characteristics you need to consider when developing a cloud application. Availability is a measurement of uptime. Availability can be achieved through fault tolerance. You can implement backup systems in different zones or regions and use health checks to notify you when to failover to alternate resources. In Compute Engine, you can use regional or

multi-regional instance groups with health checks as an option for increasing availability.

Durability is a measurement related to data protection. In your disaster recovery planning, durability directly relates to your recovery point objective, which is how much data you can afford to lose when a system failure happens. Important methods to improve durability include implementing a backup strategy and replicating data to multiple zones or regions depending on your needs. Implementing snapshots is a way to increase the durability of persistent disks of a Compute Engine instance.

Scalability defines an application's ability to handle increased load without failing. Making an application scalable requires usage monitoring and autoscaling to respond to changes in load. Managed instance groups in Compute Engine provide autoscaling capabilities. Autoscaling in a managed instance group is based on a policy you specify. An autoscaling policy defines how the autoscaler reacts to operational needs based on instance group metrics such as average CPU, load-balancing requests per second, or specified Cloud Monitoring metrics. The autoscaler will scale in (remove instances) or scale out (add instances) based on these metrics.

4.1 Diagnostic Question 04 Discussion

Developers on your team frequently write new versions of the code for one of your applications. You want to **automate the build process when updates are pushed to Cloud Source Repositories.**

What should you do?

A. Implement a **Cloud Build configuration file** with build steps.
B. Implement a **build trigger** that references your repository and branch.
C. Set proper **permissions** for Cloud Build to access deployment resources.
D. Upload **application updates and Cloud Build configuration files** to Cloud Source Repositories.

Google Cloud

Feedback:
A. Incorrect. Cloud Build configuration files specify the arguments for the containerized build requirements for your application. They do not automate the process.
B. Correct! Cloud Build triggers automate the build process when new files are placed into the name and branch of the repository that you specify.
C. Incorrect. Permissions provide Cloud Build with the required access to deployment resources. Having the correct permissions does not automate the process.
D. Incorrect. Unless you have implemented a build trigger, uploading new files will not automatically start the build process.

Where to look:
https://cloud.google.com/source-repositories/docs/concepts

Content mapping:
- Architecting with Google Cloud: Design and Process (ILT)
  - M3 DevOps Automation
- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M3 DevOps Automation

Summary:
Cloud Source Repositories are private git repositories hosted on Google Cloud. You first create a repository with the create command. You can then clone it to your local

environment and add or modify code in that local directory. When you are done coding, add the files you worked on to a new commit. You then push the local changes to your remote Cloud Source Repository.

When the app is ready for deployment, you can run the appropriate gcloud CLI command from your development environment. If you want to automate this process, ensure that your Cloud Build service account has the proper permissions to deploy to the service of your choice and provide Cloud Build configuration info in a YAML file. Cloud Build consists of simple containers that run commands in their build environment without your having to configure and manage hardware. When your build configuration is documented, you can save that in your repository also.

Cloud Build triggers will monitor your Cloud Source Repositories when new push events happen. Your trigger will require the name of the repository and branch you want to create the event from. You then specify the Cloud Build configuration you want to reference based on that event. Now when you update your app and push it to your repository, it will fire the trigger and automatically start the build operation.

4.1 Diagnostic Question 05 Discussion

Your development team used **Cloud Source Repositories**, **Cloud Build**, and **Artifact Registry** to successfully implement the build portion of an application's CI/CD process.. However, the deployment process is erroring out. Initial troubleshooting shows that the **runtime environment does not have access to the build images.** You need to advise the team on how to resolve the issue.

What could cause this problem?

A. The r**untime environment does not have permissions to the Artifact Registry** in your current project.

B. The **runtime environment does not have permissions** to **Cloud Source Repositories** in your current project.

C. The Artifact Registry might be in a **different project.**

D. You need to specify the Artifact Registry **image by name.**

Google Cloud

Feedback:
A. Incorrect. Runtime environments have read access permissions to Artifact Registry in the same project.
B. Incorrect. Runtime environments do not need access to Cloud Source Repositories as part of the deployment process.
C. Correct! Permissions must be configured to give the runtime service account permissions to the Artifact Registry in another project.
D. Incorrect. In Artifact Registry, you need to identify images by tag or digest.

Where to look:
https://cloud.google.com/source-repositories/docs/concepts

Content mapping:

- Architecting with Google Cloud: Design and Process (ILT)
  - M3 DevOps Automation

- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M3 DevOps Automation

Summary:
Artifact Registry provides managed storage and sharing of container images. It can be integrated into your CI/CD process to deploy new logic to staging and production environments. Cloud Build and Cloud Source Repositories are two other Google Cloud products that can be used with Artifact Registry to provide a managed delivery

pipeline. In Cloud Build, a build config file defines the registry where a container image is saved. Runtimes pull container images from your repository during the continuous delivery step of your CI/CD process. Supported runtimes include Cloud Run and Google Kubernetes Engine. These services have access to a registry within the same project. Permissions can also be configured for access to a registry in another project.

4.1 Diagnostic Question 06 Discussion

You are implementing a **disaster recovery plan** for the cloud version of your drone solution. **Sending videos to the pilots** is crucial from an operational perspective.

What design pattern should you choose for this part of your architecture?

A. **Hot** with a **low** recovery time objective (RTO)
B. **Warm** with a **high** recovery time objective (RTO)
C. **Cold** with a **low** recovery time objective (RTO)
D. **Hot** with a **high** recovery time objective (RTO)

Google Cloud

Feedback:
A. Correct! Safety and compliance require your application to have a low RTO, so you need a hot design pattern with minimal downtime.
B. Incorrect. A warm design pattern would consist of a standby system that you would fail over to if something went wrong. The RTO would be higher than with a hot design pattern.
C. Incorrect. By definition, cold design pattern has the highest RTO of the design patterns. Depending on the backup/snapshot pattern you have, you might be able to decrease the RPO.
D. Incorrect. The system requires a low RTO, so we ensure that the pilots get the video they need to navigate the drones where they need to go and avoid obstacles.

Where to look:
https://cloud.google.com/blog/products/storage-data-transfer/designing-and-implementing-your-disaster-recovery-plan-using-gcp
https://cloud.google.com/architecture/dr-scenarios-planning-guide

Content mapping:

- Architecting with Google Cloud: Design and Process (ILT)
  - M7 Designing Reliable Systems

- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M7 Designing Reliable Systems

Summary:
Two key metrics associated with disaster recovery planning include a recovery time objective (RTO) and a recovery point objective (RPO). The RTO is the maximum acceptable length of time that your application can be offline. The RPO describes how much data you will lose while a system is down. When these two metrics are smaller, your application will cost more to run.

Google Cloud has features that can help you with DR planning, such as global network design, built-in redundancy, scalability, security, and compliance.

DR patterns can be cold, warm, or hot. These patterns determine how quickly a system can recover if something goes wrong. Cold means your system has no failover or standby strategy. Your system will be down until you manually configure a replacement or the system comes back up. A good example of this is a system that accesses and queries historical data. A warm pattern could be a cold standby, where you implement resources but must configure your application to point to them and possibly start them if something goes wrong with the primary. A hot pattern is an active-active architecture, where you transfer data synchronously to your secondary system and load balance across the systems, so if one goes down, the other one will pick up the slack.

4.1 Diagnostic Question 07 Discussion

The number of requests received by your application is **nearing the maximum specified in your design**. You want to **limit the number of incoming requests** until the system can handle the workload.

What design pattern does this situation describe?

A. Applying a **circuit breaker**
B. Applying **exponential backoff**
C. Increasing **jitter**
D. Applying **graceful degradation**

Google Cloud

Feedback:
A. Correct! A circuit breaker limits requests based on a threshold that you specify.
B. Incorrect. Exponential backoff increases the amount of time between retry requests. It does not limit them. Applying exponential backoff is a client-side solution; we need a server-side solution to address this situation.
C. Incorrect. Jitter adds randomness to the exponential backoff to better spread the retries received by the system. Increasing jitter is a client-side solution; we need a server-side solution to address this situation.
D. Incorrect! Graceful degradation limits the results provided by the system when certain thresholds are met. It does not limit or respond with errors based on new or additional requests.

Where to look:
https://cloud.google.com/architecture/scalable-and-resilient-apps#patterns_and_practices
https://cloud.google.com/traffic-director/docs/configure-advanced-traffic-management
https://sre.google/sre-book/addressing-cascading-failures/

Content mapping:

- Architecting with Google Cloud: Design and Process (ILT)
  - M7 Designing Reliable Systems

- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M7 Designing Reliable Systems

Summary:
Traffic management lets you reduce traffic to overloaded systems or services. Two techniques that can help you in this situation are circuit breaker patterns and exponential backoffs. These techniques can help avoid cascading failures, where an issue with one service causes other services to fail too.

With circuit breakers, failure thresholds are set to prevent client requests from overloading your backends. When the threshold is met, no new connections or additional requests are allowed. Circuit breaking sends an error when requests are refused. Example settings include:

- Maximum requests per connection
- Maximum number of connections
- Maximum pending requests
- Maximum requests
- Maximum retries

Another issue when systems start being overwhelmed with requests is that the number of retries increases. Capped exponential backoff means that clients multiply their backoff by a constant after each attempt, up to some maximum value. Jitter introduces randomness to the exponential backoff, so spikes occur less frequently and at a more constant rate.

4.1 Diagnostic Question 08 Discussion

The pilot subsystem in your Delivery by Drone service is critical to your service. You want to ensure that **connections to the pilots can survive a VM outage** without affecting connectivity.

What should you do?

A. Configure proper **startup scripts** for your VMs.
B. Deploy a **load balancer** to distribute traffic across multiple machines.
C. Create **persistent disk snapshots**.
D. Implement a **managed instance group** and **load balancer**.

Google Cloud

Feedback:
A. Incorrect. Startup scripts ensure that your machines are properly configured and ready to run your app. They do not help with outages.
B. Incorrect. Cloud Load Balancing helps distribute traffic across machines in multiple instance groups. It does not heal or scale VMs.
C. Incorrect. Persistent disk snapshots prevent the loss of data in an outage. They do not help with healing or connectivity.
D. Correct! Managed instance groups with a load balancer offer scaling and autohealing that automatically replaces the instances that are not responding.

Where to look:
https://cloud.google.com/compute/docs/tutorials/robustsystems
https://cloud.google.com/architecture/scalable-and-resilient-apps#patterns_and_practices

Content mapping:

- Architecting with Google Cloud: Design and Process (ILT)
  - M7 Designing Reliable Systems

- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M7 Designing Reliable Systems

Summary:
Single points of failure occur when not enough backup resources are allocated. They

can be avoided through data replication and by implementing multiple compute instances. It is better to distribute load across multiple small, distributed units than to have fewer, larger ones. If failures do occur, the backup systems must have enough capacity to handle the extra load.

A correlated failure is when related items fail simultaneously. The impact of correlated failures can be reduced by containerizing your application and implementing microservices that can run on multiple platforms. If it makes sense in your architecture, you can do this across multiple zones or regions.

Cascading failures are closely related to correlated failures.This happens when one component of the system is overwhelmed and stops working. Other parts of the system try to pick up the load. They are also overwhelmed and start failing. The failures flow, or cascade, across the system.

Possible ways to reduce the occurrence of cascading failures include monitoring the deployment of your application and ensuring that it is in the proper state before it accepts requests, and ensuring that it has the proper resources provisioned. Server overload can be minimized by serving degraded results, load shedding, or graceful degradation.

4.1 Diagnostic Question 09 Discussion

Cymbal Direct wants to improve its drone pilot interface. You want to **collect feedback on proposed changes** from the community of pilots **before rolling out updates systemwide**.

What type of deployment pattern should you implement?

A. You should implement **canary testing.**
B. You should implement **A/B testing.**
C. You should implement a **blue/green deployment.**
D. You should implement an **in-place release.**

Google Cloud

Feedback:
A. Incorrect. Canary testing uses a subset of real traffic to test the production performance of a new version.
B. Correct! A/B testing is a pattern that lets you evaluate new proposed functionality.
C. Incorrect. Blue/green does not let you test your new features. It instantiates a new version and then moves traffic to it when its resources are stable.
D. Incorrect. An in-place release will not let you test or evaluate your new features. It will replace the version when it is deployed to the existing resources.

Where to look:
https://cloud.google.com/architecture/implementing-deployment-and-testing-strategies-on-gke
https://sre.google/workbook/canarying-releases/

Content mapping:
- Getting Started with Google Kubernetes Engine (ILT and On-demand)
  - M4 Introduction to Kubernetes Workloads

Summary:
Deployment options for new versions of your application include replacing the old version with the new release in-place. In this pattern, your new version starts accepting production traffic as soon as it is deployed. Another option is to scale down the current version before you scale up the new one, but this option does incur downtime. In rolling updates, a subset of application instances, instead of all of them,

is upgraded at the same time. This method requires no downtime.

In a blue/green deployment pattern (also called red/black), you deploy the new version next to the current one, but only one version is live at a time. The green version is deployed and tested, and then traffic is routed to it when it is stable. The blue version (your original version) can be kept up for possible rollback and eventually be decommissioned or used for a subsequent update. There is no downtime associated with blue/green deployments.

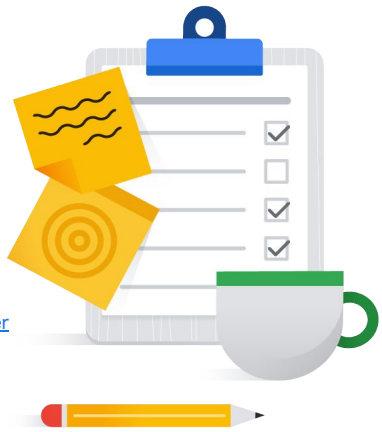
With a canary test, you deploy your new version next to your current one. You specify a subset of your production traffic to be routed to the canary version to evaluate its performance. The benefit of this pattern is that you are testing against production traffic.

A/B testing is closely related to canary testing. As opposed to being version-oriented, A/B testing is implemented to measure the effectiveness of proposed changes. Canary testing is concerned with production performance, while A/B testing is concerned about effectiveness of new features.

4.1 Analyzing and defining technical processes

Resources to start your journey

- Securing the software development lifecycle with Cloud Build and SLSA
- CI/CD with Google Cloud
- Site Reliability Engineering
- DevOps tech: Continuous testing | Google Cloud
- Application deployment and testing strategies | Cloud Architecture Center
- Chapter 17 - Testing for Reliability
- Service Catalog documentation | Google Cloud
- What is Disaster Recovery? | Google Cloud
- API design guide



Google Cloud

You just reviewed a number of diagnostic questions that asked you to analyze and define technical processes. These are some links to learn more about the concepts in these questions.

https://cloud.google.com/blog/products/application-development/google-introduces-slsa-framework
https://cloud.google.com/docs/ci-cd
https://sre.google/
https://cloud.google.com/architecture/devops/devops-tech-test-automation
https://cloud.google.com/architecture/application-deployment-and-testing-strategies
https://sre.google/sre-book/testing-reliability/
https://cloud.google.com/service-catalog/docs
https://cloud.google.com/learn/what-is-disaster-recovery
https://cloud.google.com/apis/design/

4.2 Analyzing and defining business processes

Considerations include:

- Stakeholder management (e.g. influencing and facilitation)
- Change management
- Team assessment / skills readiness
- Decision-making processes
- Customer success management
- Cost optimization / resource optimization (capex / opex)

Google Cloud

Although your diagnostic questions did not focus specifically on this objective, analyzing and defining business processes is a key part of a Professional Cloud Architect's job. A cloud architect is not simply an engineer (as if "just" being an engineer was simple). An engineer typically focuses on how to build the best *technical* solution. An architect also considers which solution best meets the business or organization's needs and how it fits into the current processes.

4.2 Analyzing and defining business processes

Resources to start your journey

What is Digital Transformation?
Cloud Cost Optimization: Principles for Lasting Success
Cost Optimization on Google Cloud for Developers and Operators
Certification solutions for Team Readiness

Google Cloud

There is not a diagnostic question for exam sub-section 4.2 in this course. These are some links to help you learn more about how to analyze and define business processes in your role as a Professional Cloud Architect and as you prepare to answer questions on these concepts in your exam.

https://cloud.google.com/learn/what-is-digital-transformationcation-development/google-introduces-slsa-framework
https://cloud.google.com/blog/topics/cost-management/principles-of-cloud-cost-optimization
https://cloud.google.com/architecture/cost-efficiency-on-google-cloudplication-development/google-introduces-slsa-framework
https://cloud.google.com/blog/topics/training-certifications/google-cloud-introduces-new-certifications-and-training-to-address-cloud-skills-crisists/application-development/google-introduces-slsa-framework

4.3 Developing procedures to ensure reliability of solutions in production

- Chaos engineering
- Penetration testing

Google Cloud

In addition to designing reliable solutions, part of your role as a Professional Cloud Architect involves developing procedures to ensure the reliability of solutions after they are in production. The exam guide lists chaos engineering and penetration as examples of practices you need to understand.

Question 10 tested your knowledge of procedures that are used to test the resilience of an application.

4.3 Diagnostic Question 10 Discussion

You want to establish **procedures for testing the resilience** of the delivery-by-drone solution.

How would you simulate a scalability issue?

A. **Block access** to **storage assets** in one of your zones.
B. Inject a **bad health check f**or one or more of your resources.
C. **Load test your application** to see how it responds.
D. **Block access** to all **resources** in a zone.

Google Cloud

Feedback:
A. Incorrect. Ensuring that your data remains available in an outage is part of durability.
B. Incorrect. Health checks help address availability needs.
C. Correct! Designing for increased customer demand is one way to ensure scalability.
D. Incorrect. Responding to outages of zonal resources is a key capability in addressing availability.

Where to look:
https://cloud.google.com/architecture/scalable-and-resilient-apps
https://cloud.google.com/blog/topics/inside-google-cloud/rethinking-business-resilience-with-google-cloud
https://cloud.google.com/architecture/scalable-and-resilient-apps#test_your_resilience

Content mapping:
- Architecting with Google Cloud: Design and Process (ILT)
  - M7 Designing Reliable Systems
- Reliable Google Cloud Infrastructure: Design and Process (On-demand)
  - M7 Designing Reliable Systems

Summary:
Resilience allows an application to continue functioning when different types of

failures occur. You must design for resilience. Methods to help you build highly available and resilient apps include having services available in multiple regions and zones. Your compute resources, whether virtual machines (Compute Engine instance groups) or microservice container architectures (Google Kubernetes Engine clusters), can be distributed across zones of a region. From a storage perspective, regional persistent disks are replicated across zones synchronously. Load balancing allows for low latency routing of your application traffic. Google’s serverless offerings often have built-in redundancy.

4.3

Developing procedures to ensure reliability of solutions in production

Resources to start your journey

Site Reliability Engineering
Site Reliability Engineering (SRE) | Google Cloud
Patterns for scalable and resilient apps | Cloud Architecture Center
How to achieve a resilient IT strategy with Google Cloud
Patterns for scalable and resilient apps | Cloud Architecture Center
Disaster recovery planning guide | Cloud Architecture Center

Google Cloud

The diagnostic question that you just reviewed tested your knowledge of one aspect of developing procedures to ensure reliability of solutions in production. These are some links to learn more.

https://sre.google/
https://cloud.google.com/sre#section-6
https://cloud.google.com/architecture/scalable-and-resilient-apps
https://cloud.google.com/blog/topics/inside-google-cloud/rethinking-business-resilience-with-google-cloud
https://cloud.google.com/architecture/scalable-and-resilient-apps#test_your_resilience
https://cloud.google.com/architecture/dr-scenarios-planning-guide