

GCP Professional Certificate

Creating a new configuration:

Gcloud config configurations create [Name]

Setting properties for a configuration:

Gcloud config set project [project_id]

Gcloud config set compute/zone [Zone]

Gcloud config set compute/region [Region]

Activating a configuration:

Gcloud config configurations activate [Name]

Step 1

Gcloud auth login - Opens a browser to login

Step 2

Gcloud config set project [Project_ID] #Replace with actual project

Other common commands:

Gcloud init - default settings, new or existing project, set default compute region/zone

Gcloud projects list - Displays all projects

Gcloud services list - shows all APIs

Gcloud projects describe - provides detailed metadata about the specified project

Gcloud compute instances list - List all VMs running in the project

Gcloud config configurations create/activate - create or activate gcloud configuration

Cloud Shell:

Home Directory - ~/tools, ~/projects, ~/bin - system path, /tmp or /var/tmp for system logs

Anything in the home directory will persist between sessions

1. It has 5GB persistent storage. Patches are updated automatically
2. It is useful for DevOps teams
 - a. Pre-installed tools (gcloud, kubectl, gsutil)
 - b. Deploy and troubleshoot app
 - c. Automate tasks and CI/CD for cloud build

Resource Hierarchy:

Project

Lowest level of unit

GCP Resources/Services reside

Distinct entity for billing, access management and APIs

In order to GCP, you must be in project all the times

Organization:

Top-level container in GCP

Entire company, organization or division

Projects can exist without organization

Centralize management of multiple projects
Cloud identity or google workspace
Manage resource, policies and billing within the organization
Roles can be granted at this level

Folders:

Folders sit between project and organization
Intermediate container
Organize into logical groups
Grouping marketing projects together
Inherits policies and permissions from the parent org

My Organization (Entire company)
We have several folders
Folder 1 has project A/B - They have diff APIs
Folder 2 has Project C

Project name vs ID vs number

Project Name	Project ID	Project Number
<ul style="list-style-type: none">• Human-readable• Can be edited anytime• Does not have to be unique	<ul style="list-style-type: none">• Globally unique alphanumeric identifier• Immutable• What you use with gcloud commands and APIs	<ul style="list-style-type: none">• Globally unique numerical identifier• Cannot be chosen or modified• Used internally by GCP

"Analytics Project" "analytics-project-prod-4912" 847193562410

Governance and Compliance: Org Policies

Security compliance
Enforce specific resource configurations
Domain Restriction

All the above are combined with IAM for comprehensive governance

Org Policies

constraints/compute.resourceLocations - Restrict within certain geographic region
constraints/iam.allowedPolicyMemberDomains - Control which domain users must belong to in order to be added to IAM
constraints/compute.vmExternalIpAccess - Constrain or block VMs from having external IPs
constraints/compute.requireOSLogin - Must use IAM to control SSH access to VMs

Allowlist and Denylist

1. Defined what is permitted and everything is blocked - Prevent external IPs across orgs
2. Defined what is denied and permits everything else - Block regions

Org Policies can be set at any level: Organization, Folder, Project

Creating Exceptions: Overridden at lower levels, can't relax policies from higher level

The effective policy at a given node is determined by combining all inherited policies, with the strictest policies taking precedence.

Org Level Roles:

1. Organization Administrator - Full access to all resources within the org
2. Organization Policy Administration - Manages org policies, constraints, conditions
3. Viewer - View-only access, useful for audits and monitoring
4. Browser - Read-only access to view structure and metadata of projects a folder

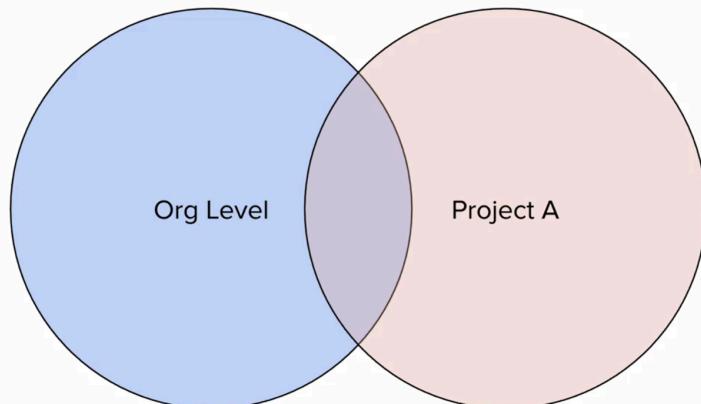
Roles are inherited

Effective IAM Policy:

Effective IAM permissions = Union of policies at a given node

Effective IAM Policy

If someone has roles at Org, Folder, **and** Project levels, what is their **actual access** at a **given node**?



Effective IAM Permissions = union of policies at a given node
(entire Venn diagram)

The effective policy at a given node is determined by combining all inherited policies, with the strictest policy taking precedence. This ensures that security and governance rules are consistently enforced and cannot be bypassed at lower levels.

Policies are not only set at the organization level—they can also be applied at the folder and project levels.

Lower-level policies do not operate in isolation; they inherit and build upon higher-level restrictions.

Precedence is determined by strictness, not by when a policy was applied. Policies are cumulative, ensuring that more restrictive rules always override less restrictive ones.

The effective IAM policies at the project level are the union of both the organization-level and project-level policies, meaning all policies from the entire Venn diagram apply. This is because IAM policies in GCP are inherited from higher levels (such as the organization) and combined with any policies set directly at the project level. As a result, the project level retains all permissions granted by the organization while also adding any additional permissions that are specifically assigned at the project level.

Cloud Billing:

1. Centralized platform for all billing related activities and payments
2. Budget alerts, usage reports, cost monitoring
3. Detailed breakdowns of costs by project, service
4. Setup budgets for different projects or services to monitor and control spending
5. Customize alerts to notify admins when spending exceeds thresholds

Billing Export data -> Big Query -> Visualize Looker

Resource labels:

1. Better cost tracking
2. Attach labels to resources
3. Apply consistent labels across related resources to enable advanced cost tracking and analysis

Billing Account Access/Roles

1. Access can be granted to Managed separately from access to underlying project
2. Access can be granted to a Billing account without granting access to underlying project
3. Project Billing Manager at project level and Billing Administrator at Org level
4. Budget alert - Billing Account Admin

Cloud IAM:

Who has what access (roles/permissions) to which resources

Principals: Identity or member granted access to GCP resources

Principles are what IAM policies applied to

User Accounts - Associated with human, accessing cloud services, authentication via username/pwd

Service Accounts - Associated with an application, process, VM. Used to enable automated/scheduled tasks. Authentication via keys/tokens

Service Accounts:

1. Cloud app with storage, Vm, Pub/sub etc
2. Crucial for secure communication between cloud component
3. Only authorised resources can access each other
4. Essential for scheduled/automated processes

Ex:

App Engine should interact with BQ and Cloud Functions

Data retrieved from and written to BQ and cloud function triggered

In this example, each action would be taken care of in automated fashion by the App Engine service account. We assign service accounts without any manual interaction to talk with each other

Two types of service accounts

Google Managed

Automatically created by google

Used by GCP services

Come with default permissions

Name predetermined

Ex: Compute engine default service account

Project_number-compute@developer.gserviceaccount.com

App Engine service account

project_id@appspot.gserviceaccount.com

User Managed

Permissions customised from the start

Custom name

Usually used for a very specific task

my-app-service-account@projectid.iam.gserviceaccount.com

Service Account Admin

To create and modify user-managed service accounts in GCP, you need the service account admin role

- a. Create b. Assign permissions c. delete or disable

Service Account Type	Identifier Used	Domain (The real giveaway)
User Created	Project ID	iam.gserviceaccount.com
Default (Compute)	Project Number	developer.gserviceaccount.com
Default (App Engine)	Project ID	appspot.gserviceaccount.com
Google Agent	Project Number	cloudservices.gserviceaccount.com

Best practice: Run scripts under service accounts

1. Python → GCP Service (Cloud SQL) -> Authenticate scripts with service account to securely access GCP. That's what service accounts are for.
2. Ensure service account has appropriate permissions (Cloud SQL viewer here)
3. Do not use your own user account

Service Account Key Management:

-
1. Google managed - handled by gcp, google creates, rotates, stores keys securely, attached directly to GCP resource
 2. Manually generated - Managed by user, use with external systems like on-prem servers, 3rd-party apps, key file required

Migration Example:

On-prem system migrated to GCP - Data to Firestore and app to GCE (user-managed key for on-prem system) and Google managed keys for Vm and GCE

IAM Permissions and Roles:

For example Data Engineer read files in cloud storage, write to dataset in BQ, create/cancel dataflow jobs then we need permissions

Permissions are very granular actions authorize principle or service account

Roles - Collections of permissions grouped together are called roles

Basic - Broad permissions at the project

Predefined - Defined by GCP, tailored for common job functions

Custom - Managed by you, can be any combination of permissions

Descriptive Role Name - Data Catalog Entry Viewer - Readable name, used in documentation or conversation. Clearly describes the role's purpose in plain language

Technical Role Name - aataCatalog.entryViewer - Fully qualified name it is gcloud or console. Follows service.roleName

Basic Roles

1. Broad level of access
2. Owner - Full access, manage roles and permissions, project admins and team leads
3. Editor - Read/write access, can't manage roles and permissions - Devs/Operators
4. Viewer - Read only access, can't modify any resources in GCP - Auditors/Clients

Predefined roles

1. Specific to given GCP services/API
2. Correlates to common roles/use cases
3. Intended to balance minimizing overhead and reducing security risk

BigQuery Job User, Storage Object Viewer, Cloud Run Developer, Compute Engine Admin, Cloud Functions Developer, BigQuery Data Editor

Composer User is collection of many specific permissions

Storage Object Creator: objects, folders, multipart and managed folders

Custom Roles

1. Defined and managed by you
2. Combination of permissions

Example:

Data Analyst - View and query BQ datasets, Run dataflow jobs for data processing, Read GCS objects, Access specific GCE instance logs for troubleshooting - WE CAN DEFINE A CUSTOM ROLE

Principle of Least Privilege:

1. Dev has privileged access to dev, QA and prod - if he checks out the code to prod then its gone - security, integrity and confidentiality - over privileged access
2. User is given read and write access to all BQ - If hacker hijacked the account he has access to BQ, Cloud SQL, Dataflow + cloud run + cloud storage and App Engine it is over-privileged access
3. Privilege escalation - if user has Read/Write access for BQ and Cloud IAM - Hacker gives his access to himself and access other accounts

Solution - Allocate minimum necessary permissions for members or service accounts. GIVE EXACT PERMISSIONS

- a. Reduces blast radius to compromised accounts
- b. Mitigates risk of insider threats
- c. Lowers chances of accidents

- d. Simplified compliance and auditing

Separation of environments, Resource hierarchy, principle of least privilege

Org -> Development project, Testing, Staging, Production

Each team has access only to environment it needs to perform its work

Groups:

To manage multiple members with similar or identical access needs

- a. Create google group using google workspace or cloud identity domain
- b. Add each member to that group
- c. Add group to IAM with appropriate roles

Typically named with

- a. Environment
- b. Project/app
- c. Purpose/role

Ex: dev-database-readers@finsecure.com or data-scientists-prod@finsecure.com

IAM Conditions

- 1. Constraints to IAM policy bindings

Time - expiring access, day of week, hour of day

Resource - Type, Service, Name, Tags

Example: Giving 3rd party vendor to access to storage buckets for a week-long project

Principle: group:cendor-team@yourcompany.com; role-storage object viewer: resource-
gs://bucket_name; condition-one week

Federated Authentication:

Allows users GCP resources using their existing credentials from trusted identity provided
(IdP) like Okta or Azure AD

User attempts to access GCP resource -----> GCP

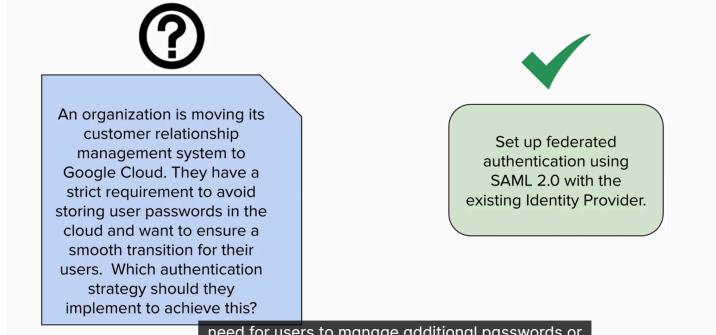
Instead of logging in directly it is redirected to IdPs any SAML 2.0 or ODBC-compliant IdP

It verifies creds and sends secure token SAML -> User is granted access based on IAM policies (roles/permissions)

It centralized identity access system

Exam Scenario:

Example exam scenario



Gcloud commands:

Copy roles between projects: gcloud iam roles copy

Gcloud get-iam-policy to get IAM policies for different resource

Gcloud projects get-iam-policy {project_ID}

```
{  
  "bindings": [  
    {  
      "role": "roles/compute.instanceAdmin", <-- The Role  
      "members": [ <-- The Users  
        "user:userX@gmail.com",  
        "user:userY@gmail.com"  
      ]  
    }, <-- End of Binding 1  
    {  
      "role": "roles/viewer",  
      "members": [  
        "user:audit@company.com"  
      ]  
    } <-- End of Binding 2  
  ] <-- This whole file is the POLICY  
}
```

IAM Policy of collection of role bindings on particular resource or project

Compute Services:

Compute Engine - Less abstraction, overhead, ops, required

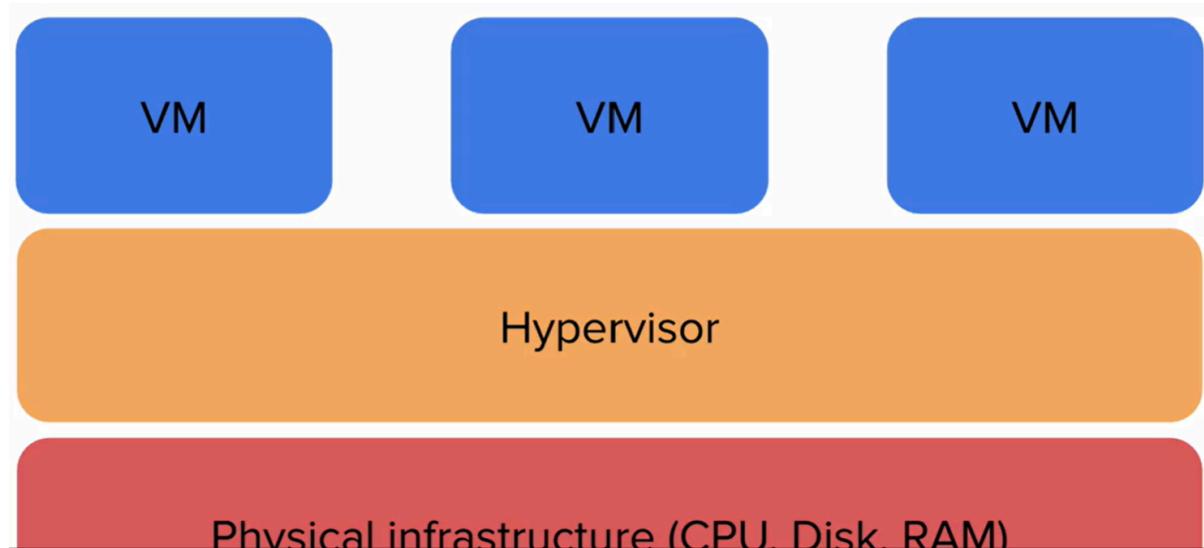
GKE - Orchestrate and Scale

Cloud Run - Stateless containers - Minimal overhead - Scaling and infrastructure

App Engine - Load balancing, health checks, write code

Cloud Functions - Less overhead, Greater abstraction, No-Ops, Serverless, event driven

1. Scalable, high-performance VMs running on Google's infrastructure
2. VM = software-based simulations of physical computers - Scalable, flexible and isolation



3. Region and Zone - It can't be changed; GCP will choose one for us
4. Considerations for region and zone - Ingress and Egress - Same region is cheaper and faster, Failover/Redundancy - Designing systems across multiple zones or regions can mitigate impact of failures/outages, Hardware Availability - not all regions/zones have same options for GPUs
5. Machine Type, Conf and instance size
 - a. Shared Core - Low cost, light weight, workloads with minimal resource requirements - E2-micro, e2-small
 - b. General Purpose - Balanced workloads with moderate CPU and memory requirements- N1, N2, N3
 - c. Compute Optimized - Best for CPU-Bound tasks requires heavy computation ex - C2D, C2
 - d. Memory optimized - Optimized for memory-heavy tasks ex: M2, M3
 - e. Storage Optimized
 - f. GPUs

Last number of instance type refers to the number of vCPUs ex: n1-standard-32 has 32 CPUs

Custom conf - Cores/Memory

6. Network settings - Firewall options - Allow Http, Https, Allow Load Balancer Health Checks. By default all external traffic is blocked.

7. Network Tags - Managing and applying firewall rules
8. Dynamic IP Address and Static IPs - By default, VMs given dynamic IPs. Whenever they are stopped and restarted, the IP address changes
9. Static IPs - can be reserved and stay linked to VM. Consistent communication needs, like DNS mapping or allowlists
10. Preemptibility - Availability Policy has - standard or Spot instances
 - a. Excellent cost saving for specific type of workload
 - b. Stop - Reclaimed by google for other workloads with 30 secs warning
 - c. Rendering and media encoding, data analysis loads - predictions, simulations etc., Hadoop/Spark clusters - Tolerate failures well, so more compute power at lower costs, CI/CD pipelines - Dev teams uses preemptible VMs for dynamic CI/CD builds, rerunning tests if failed
 - d. Not good use cases - Real-time video games, High-frequency trading, live video streaming, Critical Healthcare systems
11. Ops agent - Monitoring and Logging - Telemetry data - collect and export to cloud monitoring and cloud logging
12. Delete protection - Enable delete protection to prevent accidental deletion of an instance - Used for production instances
13. On-Host maintenance - Controls VM behavior when Google performs physical maintenance, set to Migrate VM instance to avoid interruptions or terminate interruptions (like non-critical environments)
14. Boot Disk:
 - a. OS - VM to run (Debian, CentOS, Ubuntu), Microsoft
 - b. Boot Disk Type - Always they are persistent disks - durable and persistent even VM is stopped or deleted
 - i. Balanced - General workloads
 - ii. Extreme - Maximum performance and cost
 - iii. SSD - High performance, I/O
 - iv. Standard - most cost effective
 - c. Size - Default size is 10GB but can be resized. Larger sizes needed for system files, apps or logs. Indicated by disk utilization metric above 70-80%. Size adjustments don't affect disk type. Cost depend on size and type

Additional Storage:

Boot Disk vs Attached Disks:

Boot Disk

- a. Contains OS and boot files
- b. Can be resized but is typically used for the OS and not data storage

Attached Disk

- a. Extra storage for data or applications
- b. Added/removed independently of boot disk
- c. Some types persist even if VM is deleted

Attached Disks: Persistent Disks

Standard - Most cost effective, Backups, infrequent access

Balanced - General workloads, Web Apps, Dev Vms

SSD - High performance, I/O. Analytics, DBs, transactions

Extreme - Max performance/cost. High-end DBs, enterprise jobs

Same types as you can choose for boot disk. Resize between 10GB and 65GB. Resize can be done without detaching

Lblk -> resize2fs

Zonal and Regional PDs:

1. Customize location - Default - Attached disks are in the same region and zone as the VM
2. Zonal Persistent Disk - Disk created in different zone, can be different region from VM
3. Regional Persistent Disk - Disk given replica in another zone. Both in the same region. It can be different region from VM

Toggling Auto-Delete:

By default, boot disks are deleted when VM is deleted

Disable auto-delete in order to retain even when the VM is deleted

Attached PDs - Retained by default, Enable auto-delete to eliminate them when VM is deleted

Attached Disks - Local SSD (Different from Persistent)

1. Physically attached to the host VM
2. For ultra-low latency, high I/O use cases
3. Temporary/ephemeral storage
4. Tied to the lifecycle of the host VM
5. Can't enable/disable auto-delete or make regional/zonal

Exam Question:

Organization needs High availability + Durability + Zero downtime + Failover -> Regional PD
Low-latency, High-throughput, Fast I/O, High efficiency - SSD persistent or Local. Adding or increasing size

Firestore:

1. It is network file system
2. Shared, POSIX-compliant file system (read/write/password protect/file locking)

3. Multiple VMs can mount the same fileshare, enabling them to read and write to same data
4. File appear as though they are part of the local file system, easy to integrate into applications
5. Good for high I/O , high throughput, low latency use cases that require this multishare approach

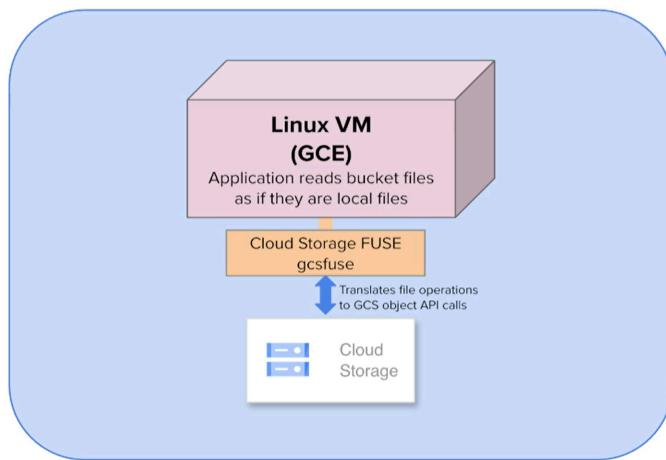
Choosing Filestore service tiers

Performance to scale automatically with storage - Filestore Enterprise

High availability across zones (Up time ect) - Filestore Regional or Enterprise

Do you not need high availability or automatic performance scaling? Filestore Zonal or Basic

Cloud Storage Fuse (gcsfuse):



1. Mounts cloud storage buckets as a file system on Linux Vms
2. Applications access object data through standard file paths
3. Reads happen directly from cloud storage (no copying)

Works well:

1. Sequential read workloads (ML training data, analytics pipelines)
2. Simple file operations without file locking
3. When you want VMs to access cloud storage like a file system

Reasons to sometimes use instead of Filestore/NFS

1. Single source of truth in Cloud Storage
2. Avoids Filestore costs and sync/replication jobs
3. Good for large datasets consumed sequentially

Limitations

1. Not optimized for random I/O or low-latency needs
2. No file locking or complex POSIX operations
3. Not a replacement for high-performance NFS

Service Account Assignment:

1. Assign a service account to the VM during creation
2. By default **Compute Engine default service account** will be automatically selected which does not follow least privilege
3. Best practice, unique service account for each instance following principle of least privilege

Example:

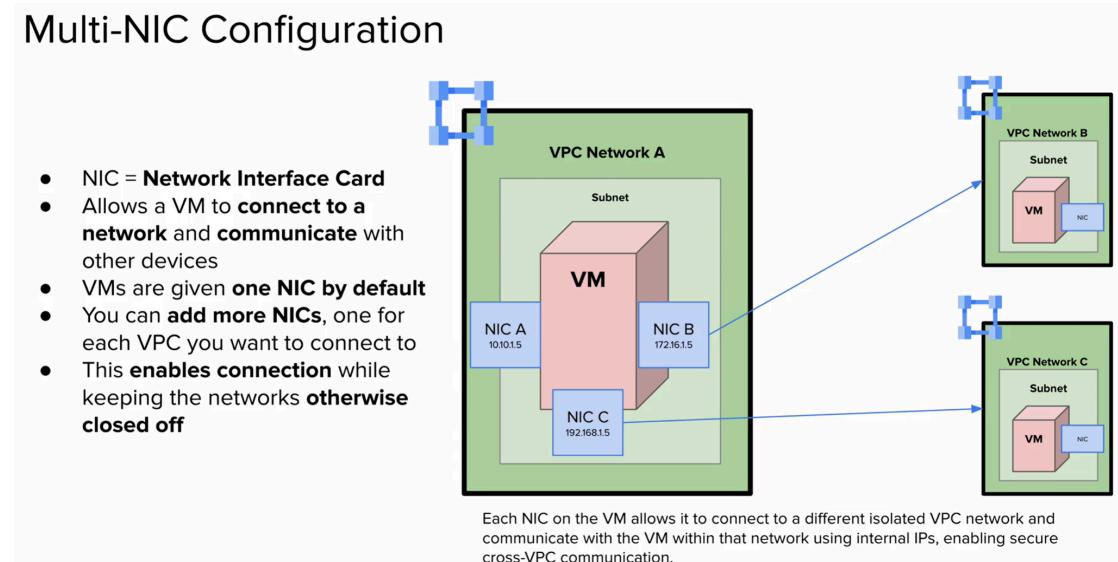
Script you're running a VM instance that needs to access data in cloud storage, but we keep getting permissions denouncing errors? What should you do?

Create service account and give it necessary permissions (read/write access to cloud storage) -> Attach the service account to the VM running the job that needs to access GCS
-> Run the script and access cloud storage as needed

Multi-NIC Configuration

1. NIC = Network Interface Card; connect to network and communicate with other devices
2. By default VMs are given one NIC by default
3. You can add more NICs, one for each VPC you want to connect to
4. Enables connection while keeping the networks otherwise closed off

Multi-NIC Configuration



Backup Methods:

1. Disk snapshots - Vm has boot disk and attached disk - We can create snapshot for boot disk and for additional data and stored in cloud storage - We can restore
2. Disk-specific backups that enables easy recovery and migration of data
3. Use cases - Backup critical data, restore corrupted disks to previous state, migrate or clone a disk to region/zone. Typically stored in cloud storage until needed
4. Custom images - Custom images create copies of Boot disk- System, OS, conf. Can be created from running disk, existing snapshot or imported from outside GCP
5. Use cases - Deploy identical instance quickly, create instance templates

6. Snapshots focus on incremental data backup while custom images are specifically designed as reusable, bootable template
7. Machine Images - Reusable template like a stamp that captures everything, disks + configurations - Boot disk + additional disk + network + metadata + machine type. Store in cloud storage for later. Quickly deploying identical VMs
8. Create a machine image from a snapshot - Boot Disk Snapshot + Machine Config = Machine Image for scaling, DR and Scaling

Connecting to VM - Add Manually generated SSH keys, It should be using OS Login enabled

Identity Aware Proxy

1. When public IPs are prohibited in GCP environment, you can enable IAP for your instance
2. IAP acts as secure proxy, routing your SSH connection through Google's infrastructure
3. Grant `iap.tunnelinstances.accessVialAP` permission to user or service account
`Gcloud compute ssh instance_name --zone Zone --tunnel-through-iap`

OS Login

Users authenticate using their google cloud user accounts and access is granted through IAM roles.

1. Cloud IAM checks if user has necessary roles for SSH access
2. If check passes then user gain access to VM
3. SSH keys are automatically managed applied by GCP
 - a. Compute OS Login - Access to VM without admin privileges
 - b. Compute OS Admin Login - SSH admin privileges, full admin control
 - c. Compute OS Login External User - Same access as compute OS login, but for users outside your org's domain
4. Gcloud compute reset-windows-password to generate a secure password
5. Or, set the windows username/password under "Remote access" in compute engine

Manage Instance Groups:

1. Simplifies the deployment and management of Vms
2. Automated updates, scaling, consistency, high availability., load balancing, operations
3. Instance Templates - Reusable blueprints to specify machine type, disk, start up scripts etc.,
4. Created from custom images
5. MIGs use instance templates to create and manage fleets of identical VMs
6. Can be used to create standard alone VMs
7. Autoscaling uses default metric CPU Utilization, custom metrics from cloud monitoring (request latency, queue length)
8. Consider MIG with autoscaling for use cases that have variable workloads (scaling test environments, gaming and web apps)

Memory-Based Scaling:

1. All relevant states to get accurate picture of memory usage
2. **Used - Memory actively utilized by running apps**
3. Buffered - Memory used to buffer temporary I/O operations
4. Cached - Memory allocated frequently accessed file/data
5. Slab - Kernel-allocated memory for managing OS structures

Preventing over-provisioning

Initial delay on health checks should exceed the amount of time it takes VMs to become fully operational, otherwise could lead to over-provisioning

Availability and Reliability of MIGs

Autohealing - Recreates unhealthy Vms to ensure high availability

Automatic Restart Quickly reboots individual VMs after a crash, to maintain service availability and performance. Can be used on standalone or MIGs

Zonal and Regional MIGs:

Zonal - Single Zone, simple workloads, no need of HA or redundancy. Low cost/non critical

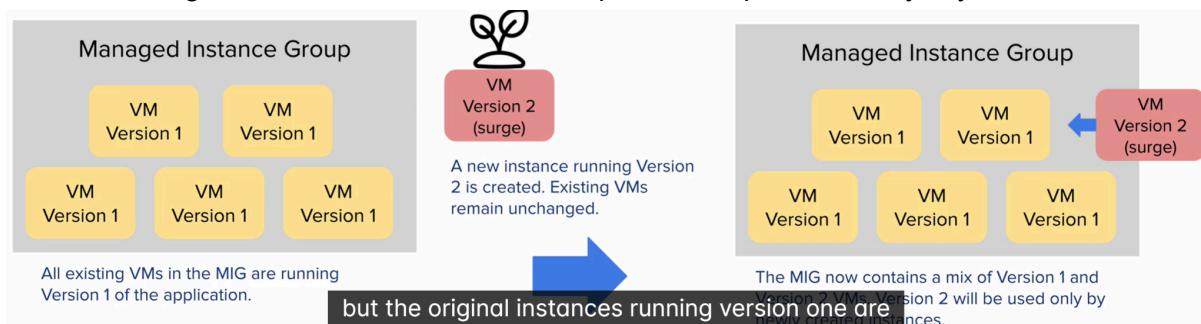
Regional - Instances are distributed across multiple zones within a single region, can specify which zone, HA and zonal redundancy for critical apps, specify a region during setup or the specific zones within the region

Multi-Regional Configurations:

GCE does not directly support multi-regional MIGs. Put a load balancer HTTPS in front of them

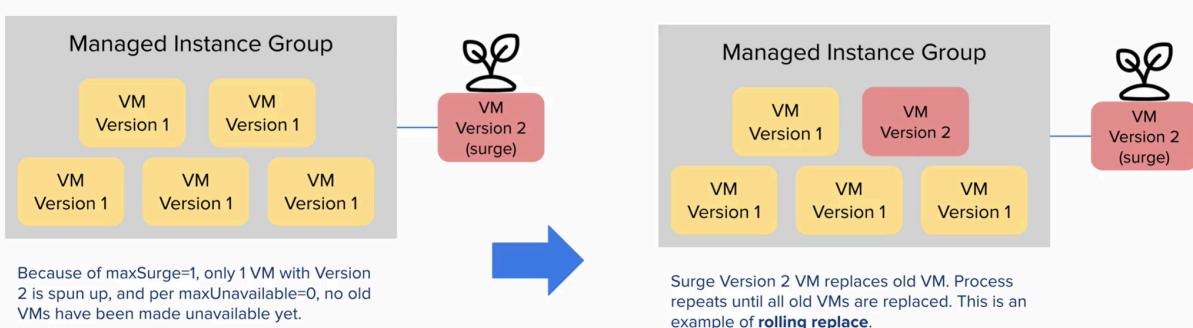
Update Policies with MIG's (Opportunistic Mode):

1. Strategies to minimize risks when updated need to be made to software or configuration of MIG
2. Opportunistic mode ensures that only new instances created in MIG use new instance
3. Existing instances are not restarted, replaced or updated in anyway



Proactive Mode:

1. Update are active, applied to all existing instances, usually one-by-one
2. maxSurge = max number of new instances that can be created beyond current size
3. maxUnavailable = number of instances that can be unavailable at any time during the update
4. Deploy new versions gradually by setting maxSurge=1 and maxUnavailable=0
 - a. Rolling restart - Reuses existing instances and restarts them with new config
 - b. Rolling Replace - Deletes and recreates instances with updated template



Active-Standby Model:

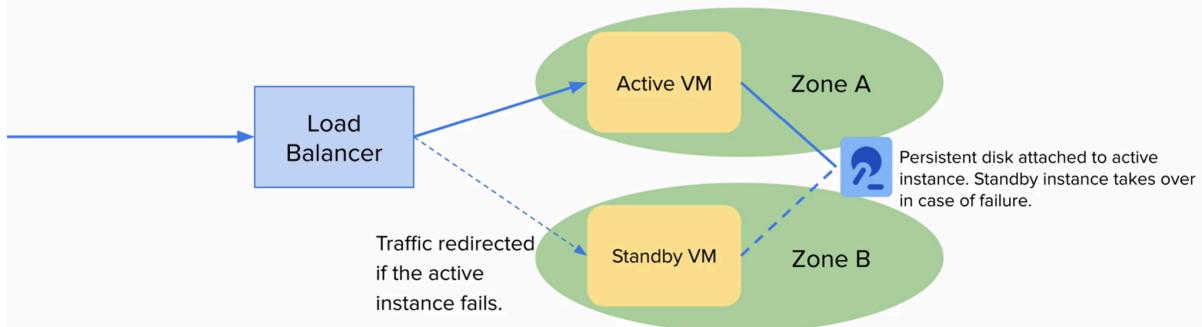
When MIGs are not appropriate

1. When horizontal scaling is not allowed, but you need high availability
2. When concurrent data access is not allowed (i.e. stateful workloads). MIG instances may try to access the same persistent disk
3. When enforcing no traffic/load sharing. If only one active instance is allowed to handle traffic at a time, MIGs are not a good fit

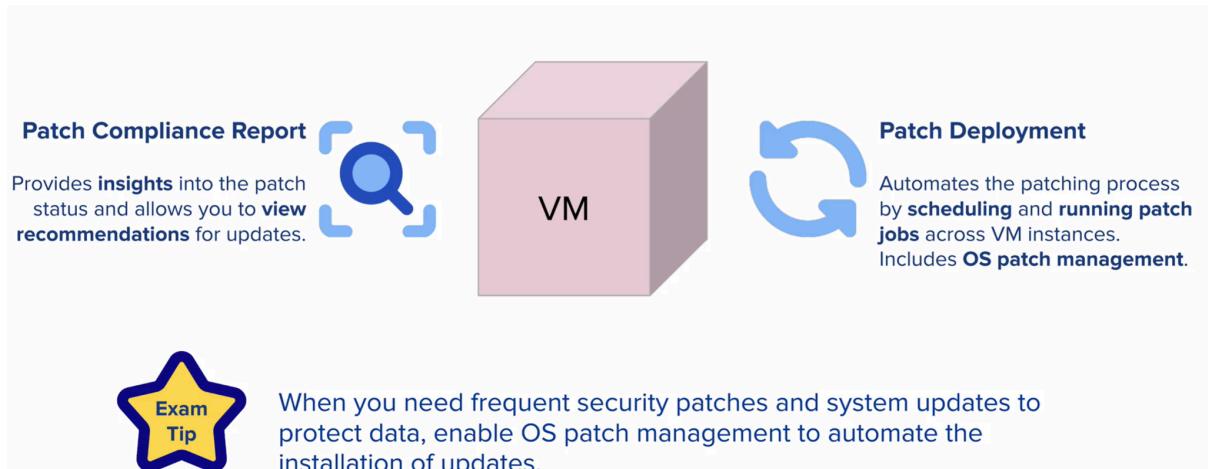
Solution: Active-Standby Model

MIGs are designed to achieve high availability through horizontal scaling and multi-region or multi-zonal configurations.

- If not possible, then instead increase availability by having a **standby instance** ready in a different zone, a **load balancer** configured to redirect traffic there in case of failure, and a **persistent disk** that the standby could attach to (usually regional)



Patch feature helps to automate security and system updates



Migrating VMs:

Service - Migrate To Virtual Machine - Lift and Shift

On-Prem, VMWare, EC2, AzureVm to GCP

1. Automated process
2. Maintains existing Vm confs - Os, apps and data

Migration Runbook

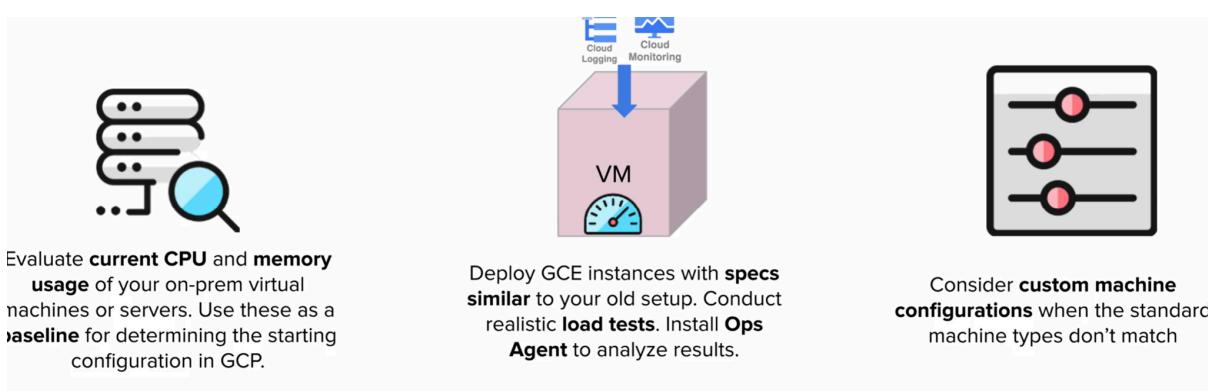
1. Step by step guide for the migration
2. Standardise procedures across multiple migrations
3. Identify dependencies to prevent disruptions
4. Fallback and troubleshooting strategies

Bring Your Own License

1. On-Prem windows server license migrate to compute engine (everything is intact)
2. Both workload and license

Benchmarking

1. Pre-migration practice of evaluating your current system's performance and running controlled tests in GCE to determine the optimal configuration for VMs



Troubleshooting strategies:

1. Cloud logging + cloud monitoring - Foundation of troubleshooting
2. Serial Console - Tool for troubleshooting, low-level, text-based interface for interacting with VMs. Mimics the functionality of a serial port on a traditional server. Direct access to boot process, system messages, recovery tools
3. 503 error and resource quotas - Server is temporarily unavailable and it is commonly caused by overwhelmed resources. Exceeding quotas can prevent the creation of new instances. Max # of replicas configured in autoscaling is also a potential cause (load Balancing)
4. Disable health checks when troubleshooting so VMs are not prematurely terminated + configure access + investigate and resolve + re-enable health checks
5. Re-attach boot disk snapshot to another VM - Snapshot of boot disk and attach to another VM and investigate on the second VM, without disrupting the original VM. Examine a point-in-time snapshot to ensure no critical data or logs are missed

Cloud Recommender:

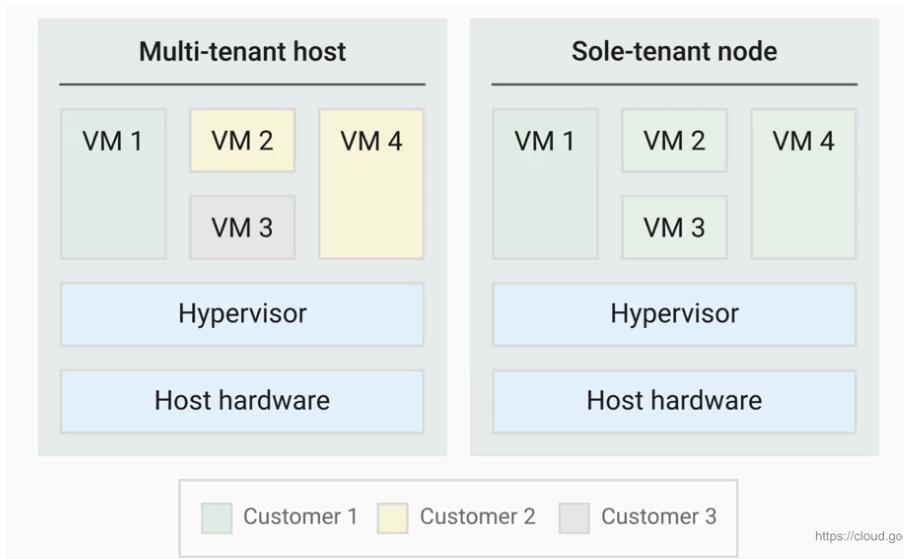
1. AI-powered optimization tool for insufficient resources (not just GCE)
2. Two main use cases in Compute Engine
 - a. Find idle VM instances
Gcloud recommender recommendations list \
--recommender=google.compute.instance.IdleResourceRecommender
 - b. Use Rightsizing Recommendations to fine-tune instance configurations

Shutdown Scripts:

1. Scripts that execute automatically when VM instance is stopped
2. Useful for ensuring that applications can clean up, save data or gracefully close processes
3. Esp useful with preemptible instances, which can be reclaimed by GCP at any time
4. Script time limit: 90s for on-demand termination and 30s for preemptions
 - a. Write a script in any executable file type (Bash, Python)
 - b. Add script to instance metadata (key=shutdown-script)
 - c. VM is stopped, either by you or Google (spot instance)
 - d. Clean-up through shutdown scrip automatically triggered

Sole-Tenant Nodes:

1. Dedicated physical servers in GCP used exclusively by a single customer
2. Ensure that no other workloads share the hardware
3. Provide physical isolation for compliance and security



- Create a node group or individual nodes with Gcloud compute sole-tenancy
- Assign node labels to manage workload/VM placement (business_unit=Finance)
- Create VM instance with node affinity rule, telling GCP which node or node group the VM should be placed on

Compute Engine - Predefined Nodes

Compute storage Admin: Necessary to create VM snapshots

Compute OS Admin Login: SSH admin privileges

Compute Admin: Comprehensive control over all GEC (instances, networks, disks, snapshots etc)

Compute Instance Admin: Specific to managing VM instances, need to modify OS metadata

Compute Viewer: Read-only access to all compute engine resources, including OS metadata

Compute Engine Disaster Recovery Strategies:

Snapshots schedule - Create a snapshot schedule policy to define the frequency and retention of snapshots, managed through the console or gcloud commands

Regional Persistent Disks-Use regional persistent disks to replicate data automatically across two zones in same region, high availability and failover during zone outages

Geographic Redundancy- Deploy instances in multiple zones or regions to protect failures or outages

Vertex AI:

1. GCP's MLOps platform
 2. Formerly AI Platform
 3. Train, host, serve ML models in single environment
- a. Tool Fragmentation - Juggling multiple platforms
 - b. Infrastructure complexity - Setting up and managing infrastructure required DevOps knowledge which takes away from key model development
 - c. Experiment Chaos - Tracking model versions, datasets, hyperparameters across experiments was manual and error prone. Making it impossible to reproduce successful results
 - d. Model Drift and Monitoring - Degraded silently in production

Vertex AI has lot of features which solves above problems

Data -

Managed Datasets - version and organise training data

Feature store - centralized repo, recreating same features

Exploration

Workbench - Dev sandbox, Jupyter notebooks

Colab Enterprise - Realtime collaboration

Model Dev

Custom training - Training process, own code and frameworks

Experiments - Training runs and hyperparameters

AutoML -

Metadata - Track of lineage what data is used etc.,

Model Management

Model Registry - Perfect history - Version control

Deployment

Batch prediction - Large amounts of data

Online prediction (Endpoints) - Automatic scaling

Post-Deployment

Monitoring - For model drift etc.,

Managed Datasets (Generally in order of ML Workflow)

1. Select a data and object
2. Stores and organises training data in the format that vertex AI services can easily consume
3. Supports image, text, tabular, video
4. Import from cloud storage, BigQuery or upload

Benefits:

1. Unified storage (single location)
2. Format standardization
3. Lineage and versioning
4. Integration with downstream services training, AutoML, pipelines
5. Easy data labeling

Preventing data exfiltration from Vertex AI Workflow

Data can't be copied or moved to unauthorised services

Result: VPC Service Controls (Security Boundary)

Cloud Storage (storage.googleapis.com) <-> aiplatform.googleapis.com (Vertex AI)

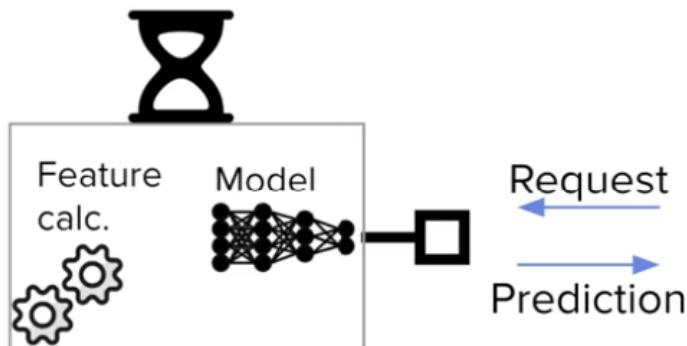
AI Feature Store:

Challenge: Feature duplication across teams

There are 2 teams which did 30_day_purchase_total and second one 30_day_spend

– Teams unknowingly/repeatedly build the same features independently because there is no way to discover and share existing work

Challenge: Production serving bottlenecks



Computing complex features on-demand during predict creates latency issues that make real-time apps impractical

Challenge: Feature Health Blind Spots

Feature quality and monitoring blind spots - Teams lack visibility into feature health, data drift and usage patterns, making it difficult to maintain feature quality overtime.

Challenge: Skew

Same features training pipelines use round(price,2) which prod is using round(price). Input distribution is skewed so performs poorly despite good offline metrics

Vertex AI Feature Store:

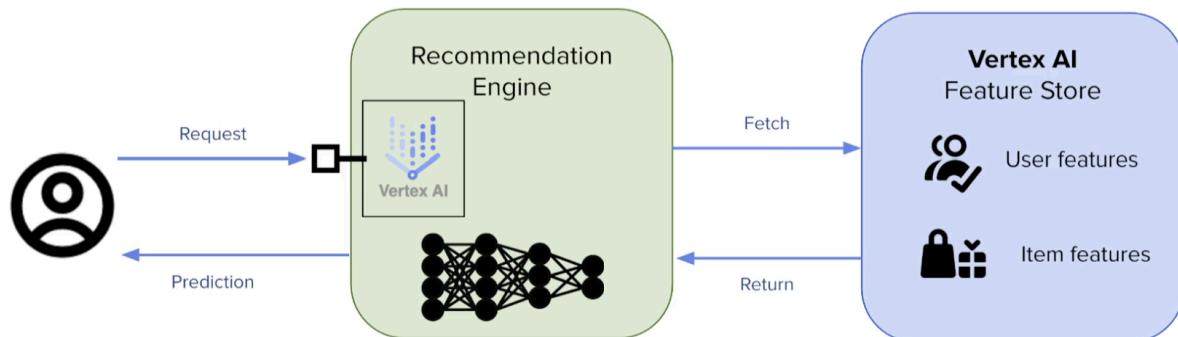
1. Centralized repo for storing, versioning and service pre-computed ML features
2. Features can be reused across multiple models, teams and both in training and prediction
3. Low-latency serving capabilities
4. Helps track feature drift and data quality

Vertex AI Feature Store Workflow

Engineer features from raw data -> Centralise in feature store -> AutoML or custom Training (Use stored features for training models) or Serve same features for predictions

Feature Store: Monitoring End-to-End Response Time:

Best Metric: Exam Scenario



End-to-end response time = Latency Metrics

Metrics: How long does the user wait for recommendations?

Colab Enterprise:

1. Realtime collaboration on Jupyter notebooks
2. Develop ML models, access datasets
3. Fully managed
4. Available through Vertex AI
5. Comments, sharing and versioning

AI Workbench:

1. Managed Jupyter notebook environments
2. JupyterLab run on Compute Engine VM
3. Pre-configured for ML with TF, PyTorch, SkLearn
4. Option to add GPUs
5. Connects easily with GCP Services and Vertex AI components
6. Runs on compute Engine VM
7. Pull in data from BigQuery for feature store
8. storing model artifacts in cloud storage
9. Loading data from BigQuery - BigQuery cell magic, allows to run SQL queries directly in a notebook cell
10. Results are automatically loaded into a pandas dataframe
11. %%bigquery df - it executes the query

Custom Training Jobs:

1. Use own training scripts and model architecture which GCP handles the infrastructure
2. Load data (Managed dataset, storage or BigQuery) -> Training script/app (custom container image or pre-installed) -> once it ready -> select hardware -> Execution (single or distributed VM) -> Pushed to Model Registry -> send to Endpoint

AutoML:

1. No-code solution for the entire model dev process
2. Takes your data and automatically builds, trains, optimizes models through a guided interface
3. Automatically creates features, tests algo/architecture, tunes hyperparameters, provides evaluation and easily deploys model endpoints

Downsides = Cost, limited model and feature customization, data format restrictions

Choose Between Pretrained APIs vs AutoML vs Custom Training:

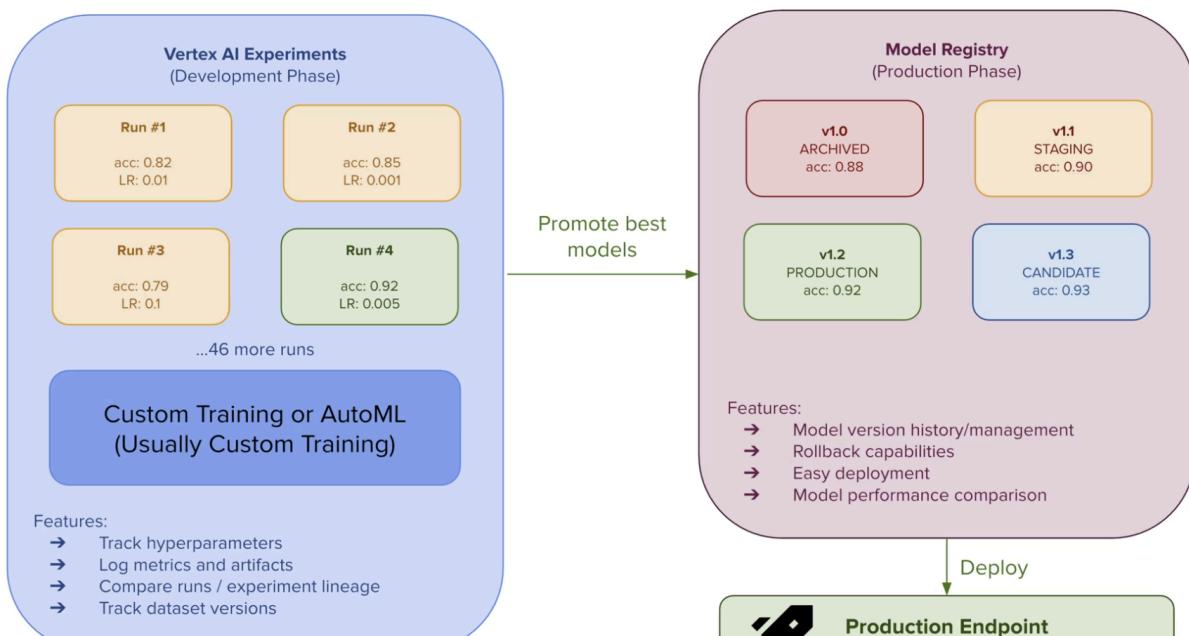
Pre-trained: Custom/standard tasks with general data. Not where domain-specific data/tuning or internal control is needed

AutoML: Goal = Minimize preprocessing and model development time.

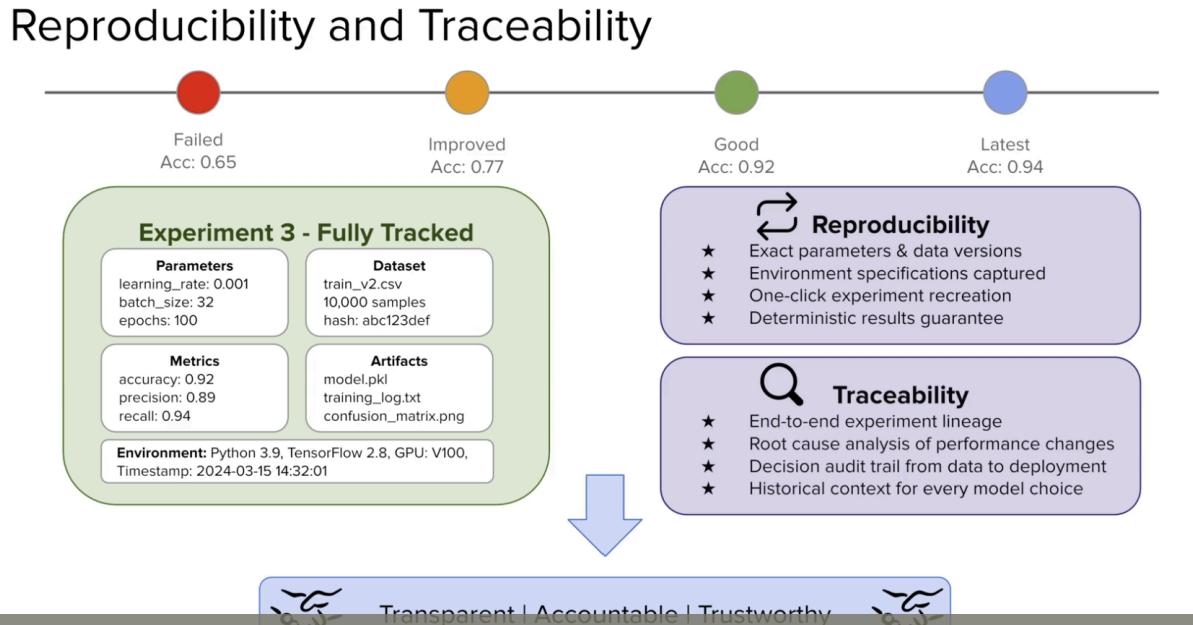
Custom Training: When you need complete control over data, model internals. Have ML expertise

Experiments:

1. ML requires many iterations (Run 1 and adjust and so on...)
2. How to keep track? How to go back?



Reproducibility and traceability



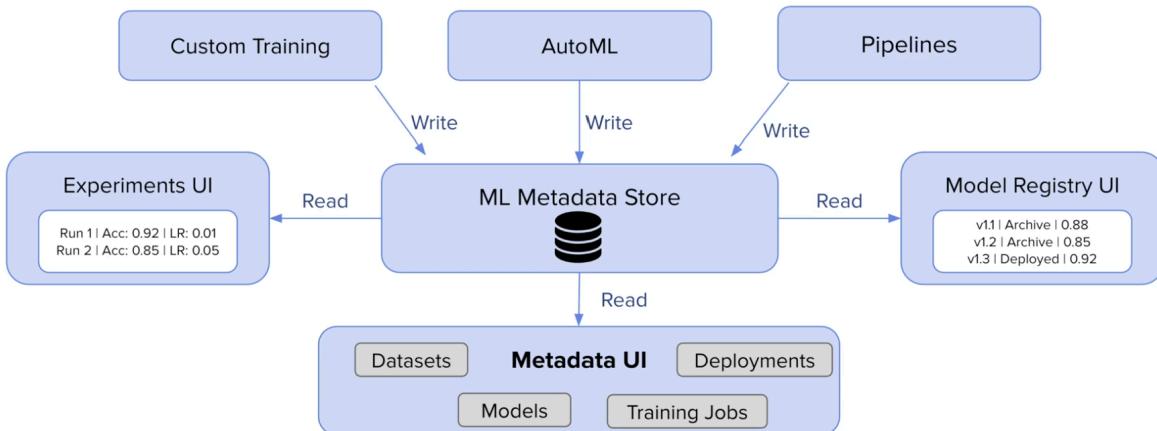
Model Registry:

1. Centralized repository for managing ML model artifacts, versions and associated metadata throughout the model lifecycle
2. Model versioning: Labels like data_science team etc
3. Model Evaluation View - F1 score, Precision, Recall
4. Deploy to endpoint - Easy Model Deployment from Model Registry
5. Post-Deployment Evaluation/Monitoring

ML Metadata:

Single Source of truth

Vertex AI ML Metadata: Single Source of Truth



Vertex AI Deployment Modes:

Batch vs Real-time inference

Batch server -> Update predictions in database

Batch: Scheduled or triggered, Large volume, low frequency, Latency tolerant, Results are typically stored for later use

Ex: Weekly customer loan risk scoring

Real-Time: Millisecond-scale response times, one prediction request after the other.

Immediate API responses. Low volume, high frequency pattern

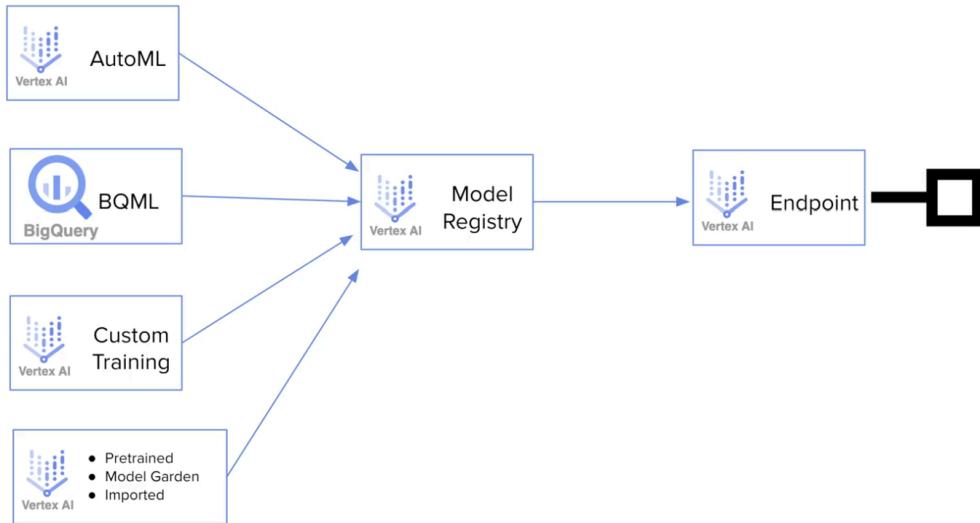
Ex: Credit card fraud detection

Endpoints: (Online Prediction)

1. In order to make ML model to use in Real world
2. Load model in to memory, accept Http request, run model inference, handle scaling and ensure reliability - Endpoints provide this infrastructure

Deployment Sources

Deployment Sources



Traffic Splitting

Model Version 1 (Production)

Model Version 2 (New Model to test)

End point can route requests to make 75% send to Model Version 1 and 25% to 2

Benefits: Risk mitigation, Gradual rollout, A/B Testing and No limit to number of models

Location consideration

Store models and endpoint in same region for lower latency

Callable from other GCP Services

Just like any HTTP endpoint we can use in Cloud Run Functions, Cloud Run, CE etc.,

Deploying from Model Garden with Custom Prediction Routine

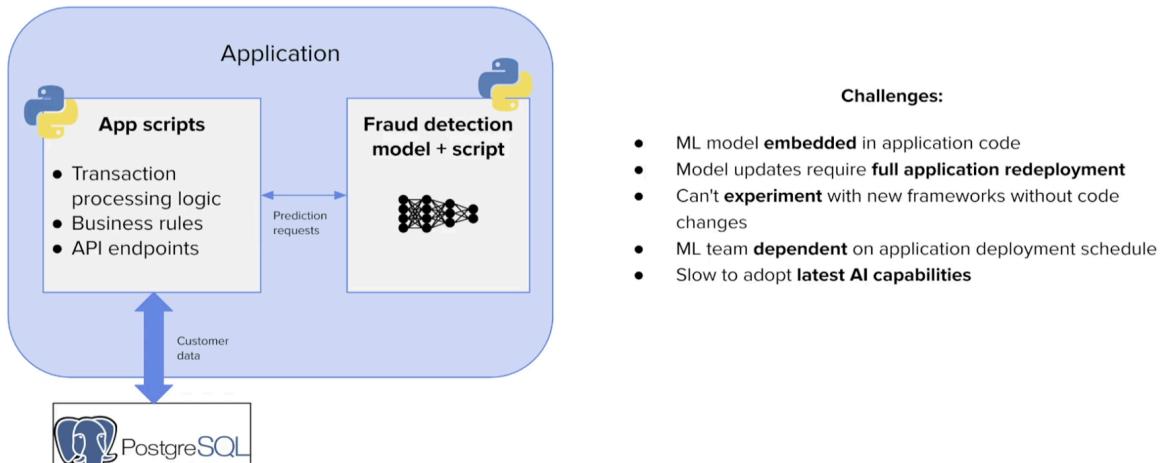
-
1. Many models in model garden can be deployed with one click directly from Model Garden Interface
 2. However, some models require specialised preprocessing that standard Vertex AI containers can't handle
 3. Solution = Package the model and custom code in Docker container
 4. You still get managed serving benefits while using custom processing logic

Model Garden -> Get model artifacts (weights, architecture etc.,) -> Build custom Docker container (model artifacts, custom prediction routine, preprocessing logic) -> (push to) Artifact Register->Deploy to Vertex AI endpoint

Vertex AI Endpoints as a way of modularizing model usage

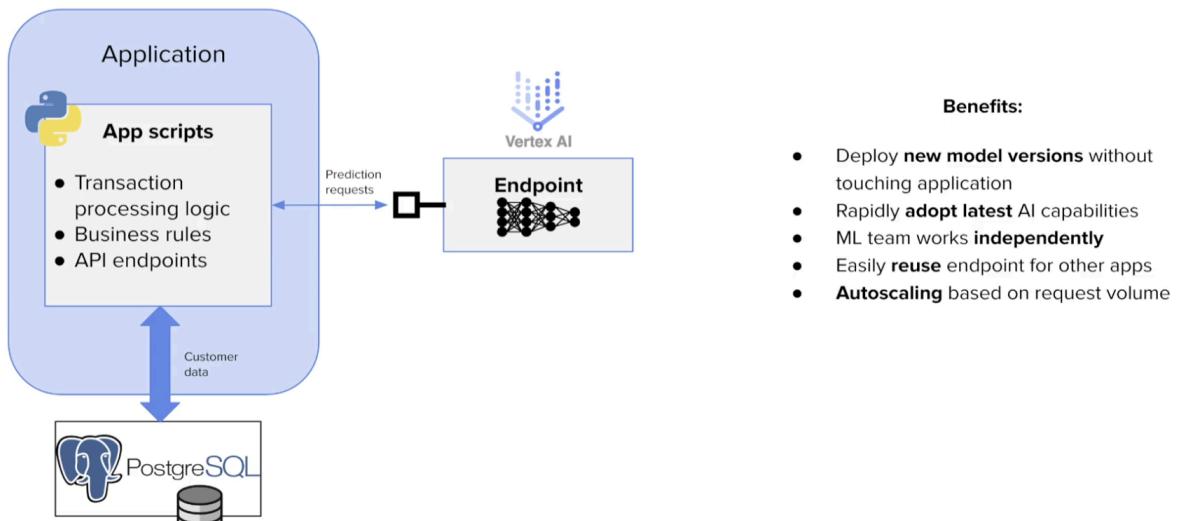
Classic problem: Tightly couple ML Architecture => Model + Application

Classic problem: Tightly Coupled ML Architecture



Decoupling ML from Application

Decoupling ML from Application



Vertex AI Batch or Offline Prediction

1. Processes large datasets in bulk
2. Good for High volume, non-real-time prediction tasks
3. Input (BigQuery, JSONL (cloud storage), JSON) => Batch prediction => Output predictions => BigQuery (Table), JSONL
4. Customer churn prediction (quarterly/monthly)

5. Weekly demand forecast
6. Sales lead scoring

Overview of ML Pipelines

Risks of manual work

- a. Check for new data
 - b. Prepare data
 - c. Trigger training - Mixed up hyperparameters configs
 - d. Monitor training
 - e. Validate performance
 - f. Compare with other models
 - g. Decide whether to deploy
 - h. Deploy manually - Deployed wrong version in prod
 - i. Monitor
1. In practice, manual ML engineering becomes time-consuming and error-prone
 2. Esp when you need to retain models frequently or manage multiple models

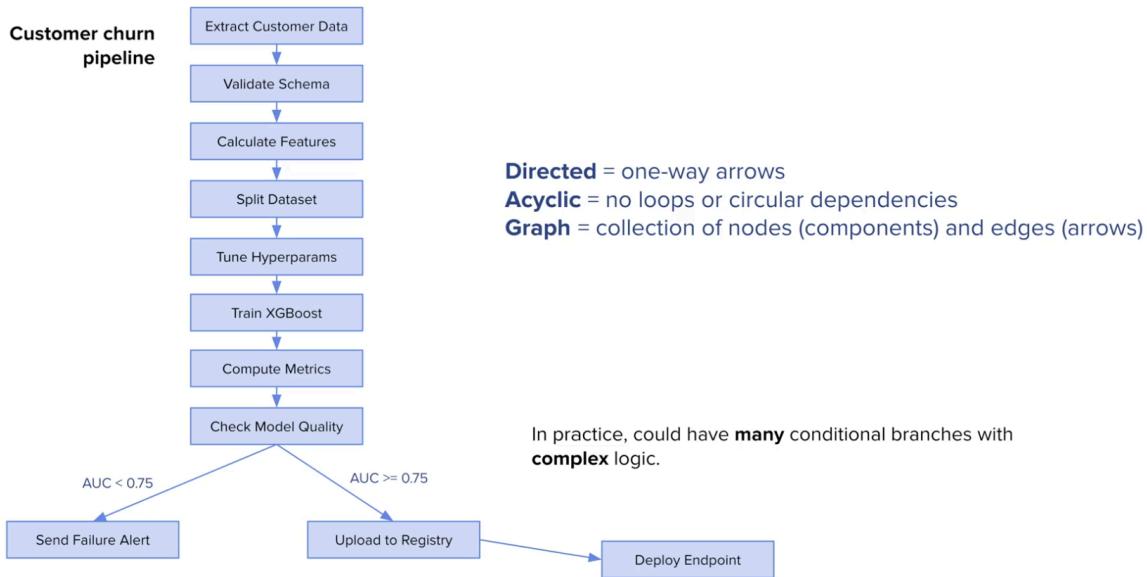
ML Pipelines can do this automatically

- a. Data Ingestion
 - b. Data prep
 - c. Dev and Training
 - d. Model Evaluation
 - e. Model Deployment
 - f. Model Monitoring - Update model (retrain on same data) and Retrain with new data (e.g data flow)
1. Ensures consistency, save time and reduces human error
 2. Can have various trigger criteria (new data, model degradation or scheduled)

Vertex AI - Pipelines

Directed Acyclic Graphs (DAGs)

Customer churn Pipeline



How does retraining happen if cycles aren't allowed?

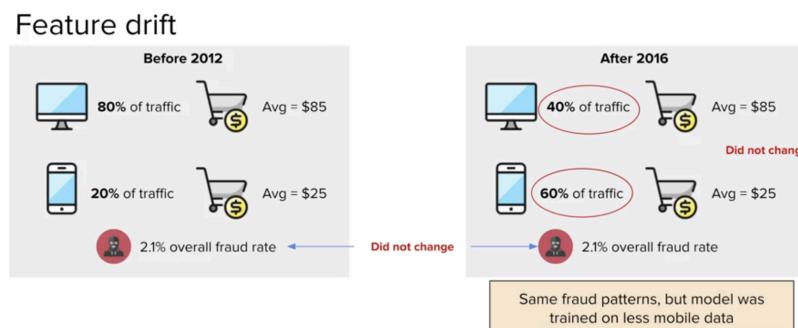
Retraining cycles and triggers happen at the orchestration level (external triggers starting new DAG runs), not within the DAG structure itself. Each individual pipeline execution remains acyclic

Model Monitoring

1. Even with retraining, model degradation/drift is inevitable given that the real world is not static
2. Vertex AI Model Monitoring tracks prediction quality over time on real-world production data
3. Usually for online prediction (Vertex AI Endpoints), but we can use for batch prediction
4. Used to be called Continuous Evaluation when it is AI Platform

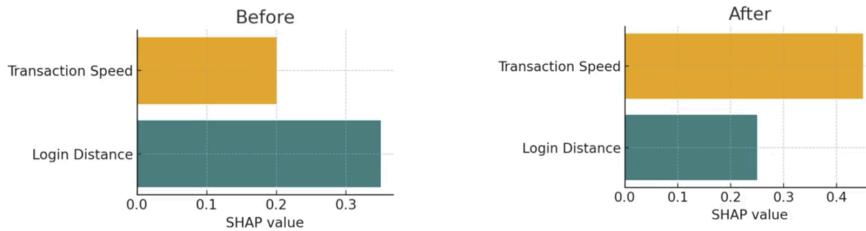
Detection Types in Vertex AI Model Monitoring

Feature Drift



- Feature drift = changes in the statistical distribution of input features
- Output distribution, and relationships between inputs and outputs, can remain the same

Feature Attribution Drift



- This is calculated at the level of the individual prediction, and then aggregated
- Pertains to how much the feature **actually contributes** to predictions
- Different from how much the model **structurally** relies on a feature (feature importance) and how much the **real-world conditions** change (concept drift)

The image **Feature Attribution Drift**. Think of this as checking if "the reasons *why* the AI makes a decision have changed."

Here is the Layman's explanation using a fraud detection example:

The Scenario: Catching Bank Fraud

Imagine you built an AI to flag stolen credit cards.

- **Before (The "Old" Logic):** The AI looked mostly at **Login Distance** (e.g., "Is this user logging in from Russia when they live in Texas?"). If the distance was weird, it blocked the card.
 - *Look at the "Before" chart:* The teal bar (Login Distance) is longer. It was the main driver.
- **After (The "New" Logic):** Suddenly, the AI starts flagging transactions mostly because of **Transaction Speed** (e.g., "Did they swipe the card 5 times in 10 seconds?").
 - *Look at the "After" chart:* The yellow bar (Transaction Speed) is now the longest.

What is "Attribution Drift"?

It means the **drivers of the decision** have swapped.

- **Attribution:** Who gets the credit (or blame) for the decision?
- **Drift:** It moved.

In the image, the AI used to care more about *where you were* (Login Distance). Now, it cares more about *how fast you act* (Transaction Speed).

Why does this matter?

If you didn't change the model, but its "reasons" changed, it might mean the **fraudsters have changed their tactics**.

- Maybe they are using VPNs now (so "Login Distance" looks normal), so the model is forced to rely on "Speed" to catch them.
- It tells you: "Hey, the world changed, and your model is reacting to it differently now."

Drift vs Skew:

1. Feature drift: Statistical distribution of a feature in production changes, compared to itself (training), over time - different time periods
2. Feature Skew: A feature's statistical distribution in production is or becomes different from its statistical distribution in training
3. Feature Attribution drift: The predictive contribution of features in production change compared to themselves overtime
4. Feature Attribution skew: The predictive contribution in production are or become different from their predictive contributions in training

Drift - Changes overtime

Skew - Changes training and production

Skew is also called as “training-serving skew”

Comparison: Drift vs. Skew

Feature	Feature Drift (Data Drift)	Feature Skew (Training-Serving Skew)
Root Cause	External environment: The user behavior or the world changed.	Internal System: Pipeline bugs, logic errors, or data source mismatches.
Timing	Happens gradually or suddenly over time after deployment.	Happens immediately upon deployment.
Example	Incomes increase due to inflation over 5 years.	Training data uses 0 for missing values, but production data uses NULL .
Fix	Retrain the model with new, recent data.	Fix the code in the production pipeline to match training logic.

Summary

- **Drift** is about the **data evolving** (e.g., summer vs. winter clothes).

- **Skew** is about **data inconsistency** (e.g., training on clean data, serving on messy data).

Sampling Rate and Frequency:

1. Sometimes millions of prediction requests to your endpoint everyday
2. Sampling rate = the proportion of prediction requests that are logged and analyzed for model monitoring

RandomSampleConfig (SampleRate: 0.33) -1/3rd will be logged and analysed

Considerations for lowering sampling rate/frequency

- a. Saves cost. Less compute and storage for model monitoring
- b. But it may make monitoring less accurate esp with smaller low volume endpoints

Monitoring Frequency:

1. Determines when the monitoring analysis runs on the collected data
2. Specified in CRON notation (five fields, with each representing a time unit)
3. Keeping the asterix in place means “every” for the corresponding time unit

Minute:Hour:day_of_month:Month:Day_of_the_week

Sampling rate + Frequency

Sampling rate (Which predictions are logged for analysis) + Frequency (How regularly the model monitoring analysis is running) = How quickly you'll detect drift/skew, how reliably you'll detect it. It impacts monitoring costs

Scenarios:

1. Critical financial models, rapid degradation could cost millions > High sampling and High Frequency
2. High-volume recommender with predictable traffic but needing fast alerts patterns > Low sampling, high frequency
3. Diagnostic model with manageable patient volume and changes develop slowly > High sampling, low frequency
4. Large-scale content recommendation system serving millions of users where traffic is massive but content preferences change gradually > low sampling, low frequency

Skew Detection requires a valid training data baseline because by definition “skew” compares training data to serving data

Baseline dataset must match the training dataset used to train the deployed model

Train model -> Deploy -> Upload the same training data -> Enable skew detection. Model monitoring compares incoming data to uploaded baseline

Note: Drift detection doesn't require baseline training data because it compares production data changes over time

Monitoring Multiple Models

Predicting: Next week's revenue

BigQuery (we collected variety data): User profile data and Usage data - We created two separate models

Here we have 2 models and 1 endpoint (app specifies in the request which model to use) - Here we need to set up model monitoring for this?

We use Monitoring config file:

- a. YAML or JSON
- b. Specify using "monitoring-config-from-file" parameter in gcloud command
 - i. Like model_a (user profile data) and model_b

Explainable AI:

1. Provides insights into how a ML model is making its prediction
2. Helps prevent "black box" models by increasing transparency and trust
3. Not a standalone product - embedded within Model Registry and Endpoints

Feature-Based (Structured data) and Example-based approach - Ex. Bird incorrectly classified as plane because it is silhouette

Feature-base methods (Feature Attribution)

Sampled Shapely (SHAP): Good explanations of tabular data with tree-based/ensemble models

Integrated Gradients: Good for Differentiable models. Tabular and image. Uses pixel level for images. Good for Neural Nets

XRAI: Good for explanations of images at a high level, groups into shapes or text. Image classification tasks

Vertex AI endpoints are regional, we must create an endpoint for each region

Your team is developing a custom model using a specialized framework that is not supported by standard pre-built containers. You want to use Vertex AI Hyperparameter Tuning (Vizier) to optimize the model. What is a prerequisite for using a custom container with Hyperparameter Tuning?

The training code inside the container must interpret command-line arguments corresponding to the hyperparameters.

Right answer

The Hyperparameter Tuning service injects trial values as command-line arguments to your container entrypoint, so your code must be written to accept and use them.

** You are implementing a 'Human-in-the-Loop' workflow. If the model's prediction confidence is below 60%, the data should be sent to human reviewers for labeling, and then used to retrain the model. Which Google Cloud service combination best facilitates this?

A. Vertex AI Prediction and Vertex AI Data Labeling Service.

That's right!

Vertex AI allows you to configure prediction requests to sample data (based on confidence) and send it to a Data Labeling task for human review.

Google Kubernetes Engine:

1. K8s is an open-source platform for automating the deployment, scaling and operation of containerized applications
2. Containers provide consistent environment
3. Originally developed by Google to manage its massive workloads
 - a. Needed to automate deployment, scaling and management
4. Widely used

Containers

- a. Portability
- b. Scalability
- c. Reliability - Failures in one container don't impact on other
- d. Before cloud migration, containerize your microservices where possible to modernize your workloads and packages into portable, consistent units

What does Kubernetes solve?

- a. Manual scaling - Horizontal Pod Autoscaler and Cluster Autoscaler
- b. Unreliable deployments - Rolling updates and self-healing mechanism
- c. Lack of Portability - Enabling transportation across different envs

Introduction to GKE

- a. Managed Kubernetes Service
- b. Simplifies like scaling, updates and maintenance
- c. Fully integrated with rest of GCP
- d. Good for complex microservice architectures

Core Kubernetes Components

Cluster = Collection of nodes (Physical servers) that runs containerized apps

Node = Worker machine within a cluster that can run multiple pods

Pod = Smallest deployable unit in Kubernetes, which can contain one or more containers. Where there are multiple identical pods, these are called "replicas"

Configuration:

Kubernetes Manifest = A file usually YAML, can be JSON that defines the desired state of objects (pods, services, deployments, etc) in a Kubernetes cluster

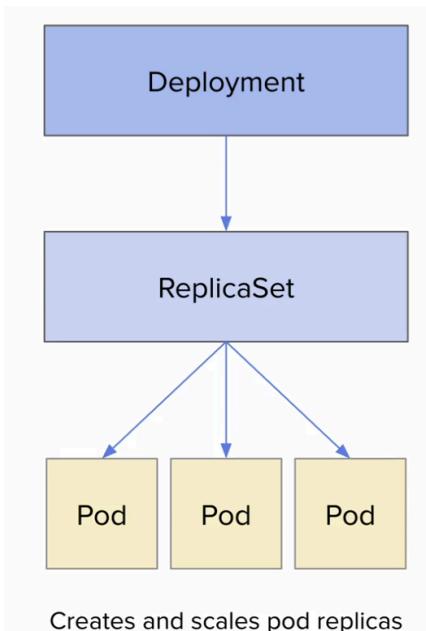
It has appVersion, kind, metadata, specs, template, containers, strategy

Deployments:

1. A “Deployment” is a high-level Kubernetes resource that manages scaling, rolling updates, and high availability of application instances, allowing services to scale independently with seamless updates
2. Ideal for stateless services
3. Use a Deployment for each service

Specify:

- a. No of Pod replicas
- b. Container image
 - i. ImagePullPolicy: IfNotPresent ensures that k8s first checks if the image is already available on the node
 - ii. Ports to use
 - iii. Update Strategy: rolling to minimize downtime



Control Plane - Central Management Layer, coordinates all processes, manage workloads, ensure desired state of cluster is maintained

API Server - Entry point for all kubernetes server and API entry points requests. This is actually how users, kubectl and other kubernetes components

Scheduler - Assigning pods to nodes based on resource availability and constraints and ensures efficient resource utilization

Controller Manager - Monitors the cluster state. Scaling workloads and maintaining replica sets

Node Pools:

1. Within a single cluster, you can create node pools with different machine types for different services/tasks
2. We can have compute-optimized and standard machine types

Commandline Tools:

Use kubectl for:

1. Managing workloads and resources within the GKE cluster
2. Deploying an application, listing services, scaling pods, checking resource status

Use gcloud for:

1. Managing the cluster itself
2. Creating, updating and deleting clusters

Kubectl commands:

kubectl config use-context - We can switch between dev/prod/staging - Switch between contexts

Kubectl config view - detailed info for current config

Kubectl set image deployment/[Deployment_name] image_name - Trigger Rolling Deployment Update

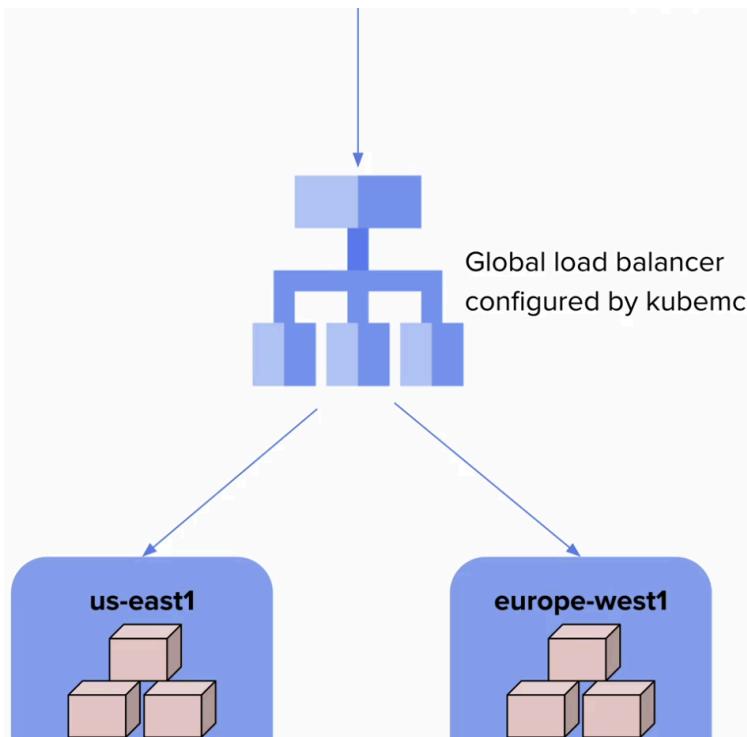
Gcloud commands:

Gcloud container clusters get-credentials [Cluster_name] - Retrieving credentials & context

Gcloud container clusters update [Cluster_name] -enable-autoscaling -min-nodes=[MIN] max_nodes=[MAX]

Kubemci:

1. Multi-cluster ingress for k8s
2. Designed to manage Https traffic across multiple K8s cluster
3. Setup as HTTPs load balancer from cloud load balancing



Helm:

1. It is kubernetes package manager
2. Simplifies app deployment and management by using pre-configured templates called “Helm Charts”

Package everything as Chart

Deployment -> pods

Ingress -> Service

Service -> Pods

GKE Docker Files:

Deploying from Dockerfiles

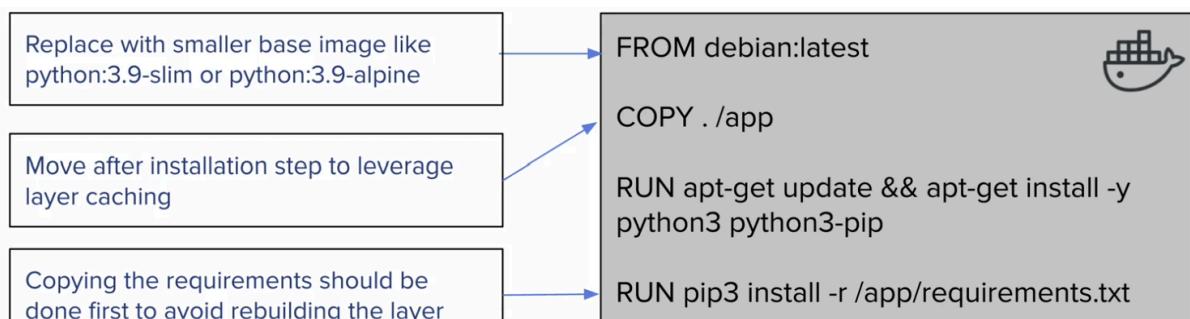
1. Create and manage containers
2. Deployed to pods on K8s cluster
3. Dockerfiles define what goes into a container
4. Docker image = blueprint for the container

Steps:

1. Create or acquire Dockerfile
2. Build Docker image
3. Push Docker image to Artifact Registry - Images are stored
4. Create k8s deployment manifest that references location of image
5. Use kubectl to reference Deployment Manifest file and deploy

Optimizing Dockerfiles:

1. Use lightweight base images - speeds up build process
2. Minimize final image size - Cleaning up unnecessary files
3. Leverage layer caching - Reorder instructions separate frequent changes
4. Avoid unnecessary steps - redundant and poorly steps



GKE: Autoscaling:

Cluster autoscaler: Manages number of nodes in the cluster based on the overall resource demands of the pods. Best practice is also set min and max nodes

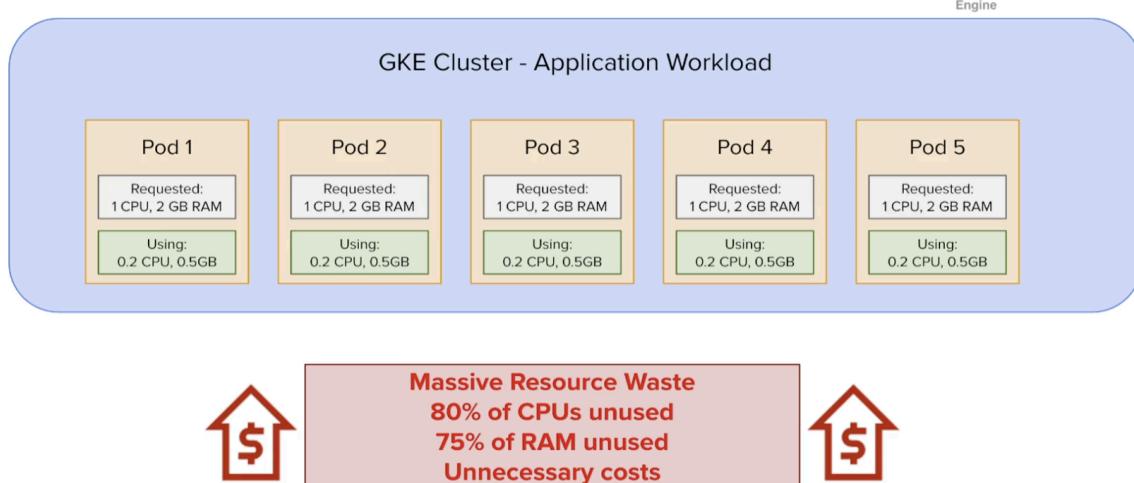
Horizontal pod autoscaler: scales number of pods, doesn't restart pods, adds/removes instead. Can scale based on either CPU/memory or custom metrics. Some stateful workloads incompatible

Vertical Pod Autoscaler: Scales the compute and memory of individual pods based on their usage. May restart pods. Can't be done based on custom metrics

GKE Overprovisioned Pods Use Case:

Scenario

Over-Provisioned Pods in GKE



Solution: Vertical Pod Autoscaler (In Auto Mode)

Note: No. of pods maintained for throughput. This is also why Horizontal Pod Autoscaler is not appropriate

Kubernetes Services to Route Network Traffic:

Internal Traffic:

1. ClusterIP - Default service type, let apps talk to each other inside the cluster using an internal IP address. Includes DNS-based addressing for seamless communication and scaling - Ideal for most applications
2. Headless Service - Direct pod-to-pod communication. Directly exposes the IP addresses of pods (No load balancing, no DNS) for apps that need to know individual pod IPs - Specialized services

External Traffic:

1. NodePort: Opens specific port on all cluster nodes, allowing external access via node IP and port. Often used with Cloud Load Balancer
2. LoadBalancer: Creates external IP address automatically distributes traffic to app
3. ExternalName: Maps application to a DNS alias to look like it's hosted to another domain ex: [test.com](#) -> externalName -> another domain
4. Ingress: Adds sophisticated https gateway/traffic management

Container Security Best Practices:

1. Vulnerability Scanning - Comparing their contents against a database of known vulnerabilities. Generate finding report which has severity, actions etc
2. Attestations - Cryptographic signature that prove a container image meets predefined security or compliance requirements. Typically part of CI/CD pipeline

Build -> Test -> Scan => Attestation created

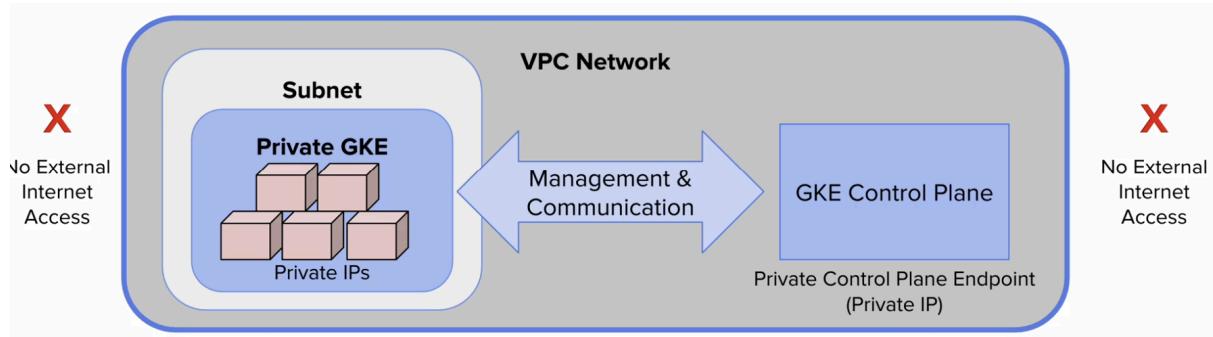
3. Binary Authorization - Managed GCP service, ensuring only trusted and verified container images are deployed. "Binary" refers to the fact it can be applied to any binary executable. Integrates easily with GKE and other GCP Services. Define policies that specify the criteria an image must meet before deployment. If image doesn't meet the criteria, it is blocked from running

Binary Authorization policies - Require attestations, vulnerability scans, allowlists, disallow lists

Cluster Security:

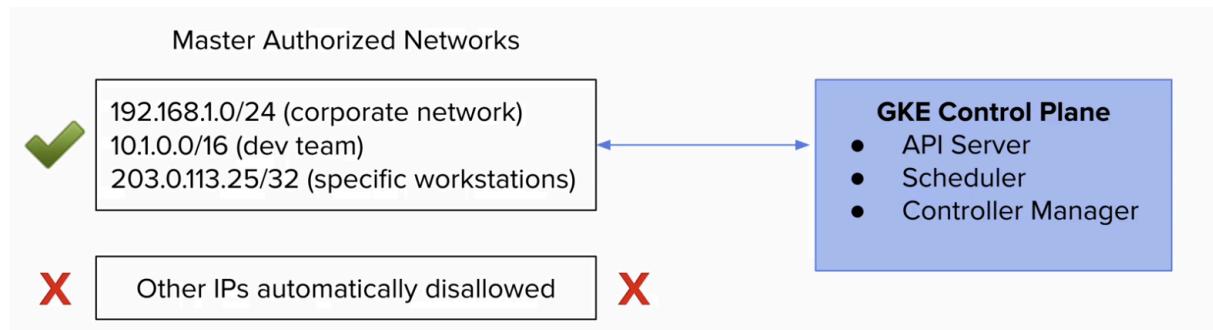
Private clusters and private control plane endpoints

- a. GKE clusters where the nodes have only internal IP addresses
- b. Nodes can communicate only with each other and other resources inside VPC. No internet access
- c. Private control plane endpoint can be set up so that control plane is isolated and only accessible via configured access rules
- d. Significantly reduces attack surface by isolating nodes and control plane from external threats



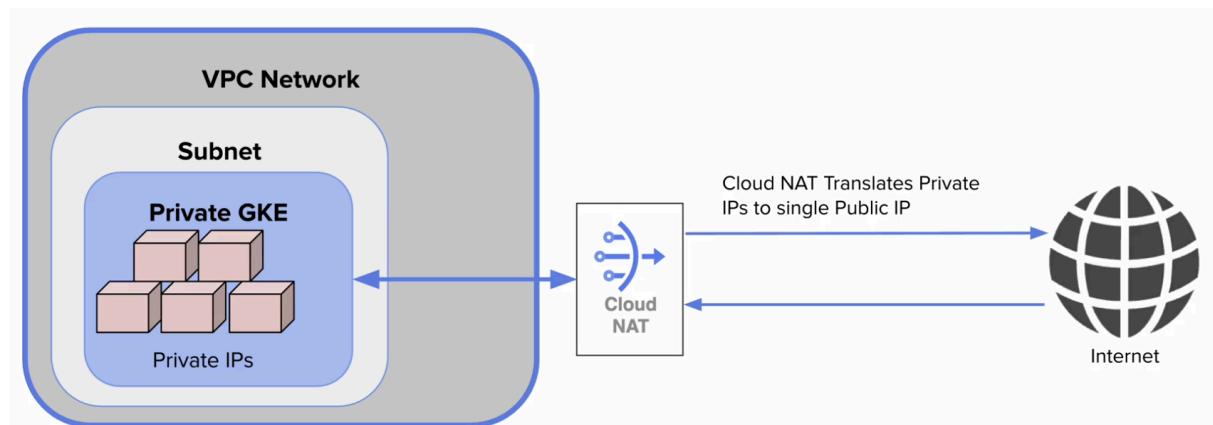
Master Authorized Networks:

1. Allows you to restrict access to the control plane by specifying trusted IP address ranges
2. Ensure that only specific networks or users can access the control plane
3. Can work with public or private clusters



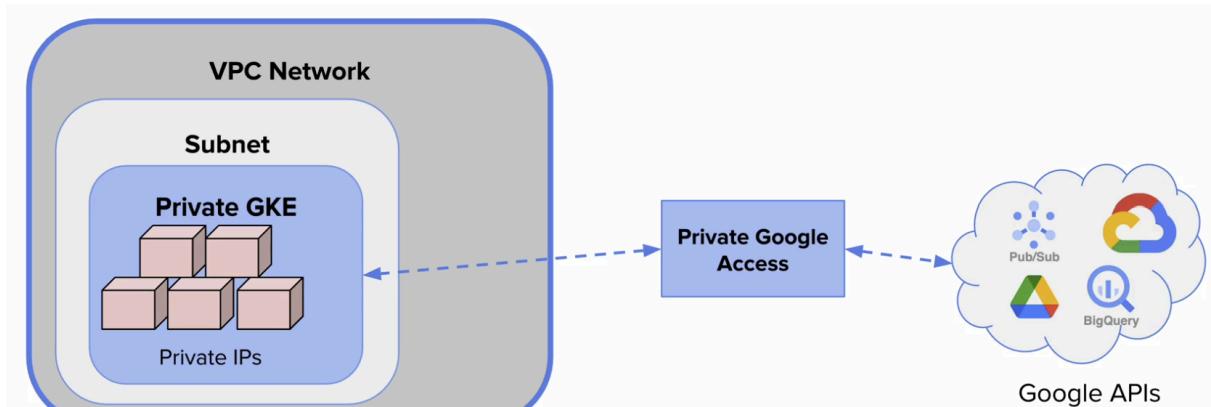
Cloud NAT (Network Address Translation):

1. Managed service that allows private GKE workloads to initiate outbound connections to the internet without being assigned public IPs



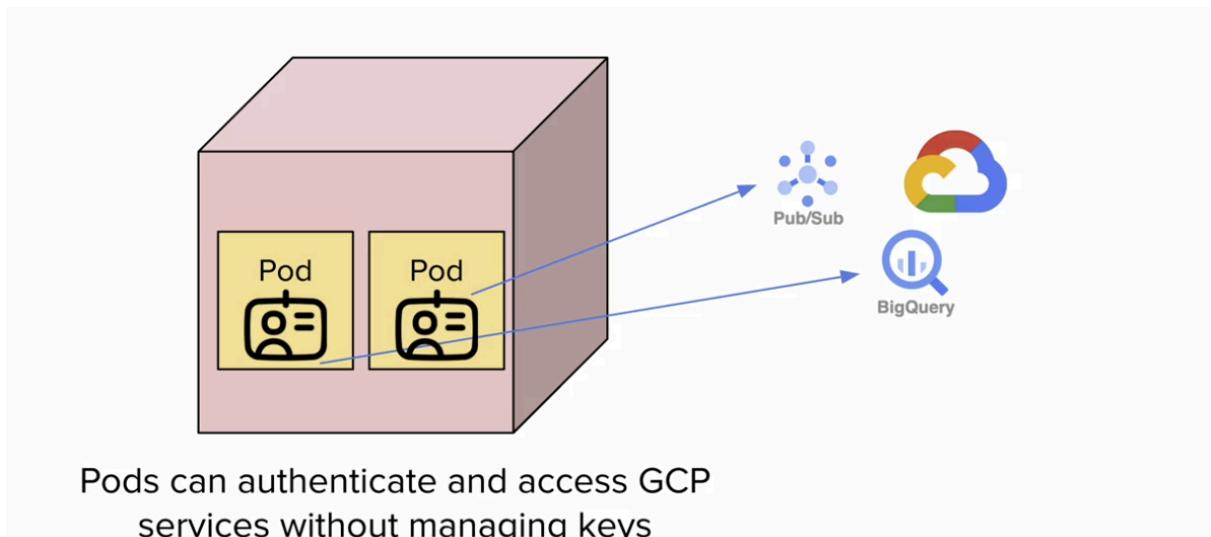
Private Google Access

1. Enable Private Google Access to allow private GKE workloads to access Google APIs and services without requiring public IP address

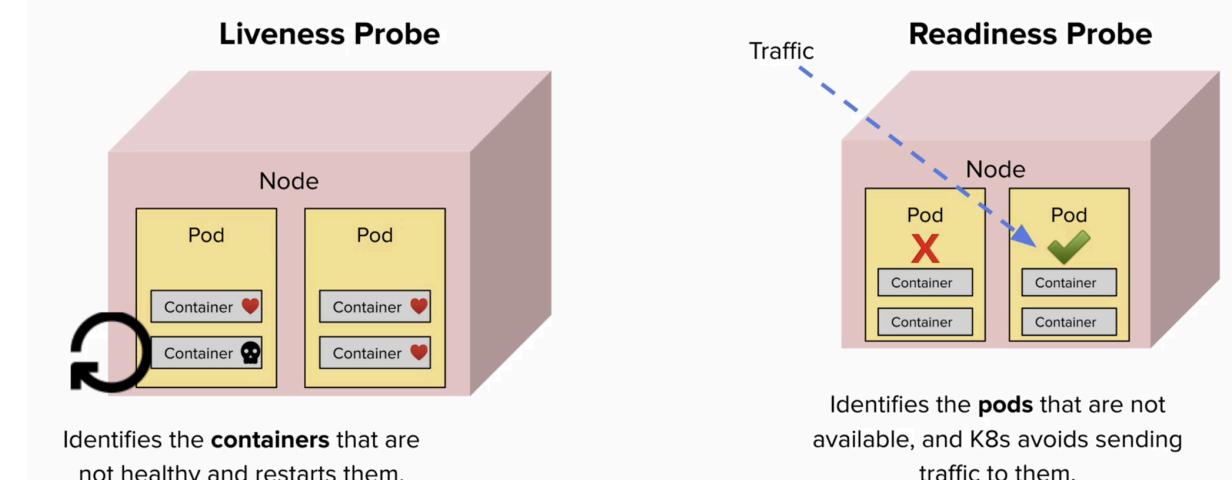


Workload Identity

1. Allows Kubernetes workloads to securely authenticate as service accounts without needing to manage long-lived credentials such as service account keys
2. Otherwise, without require creation and management of service account keys (less secure)

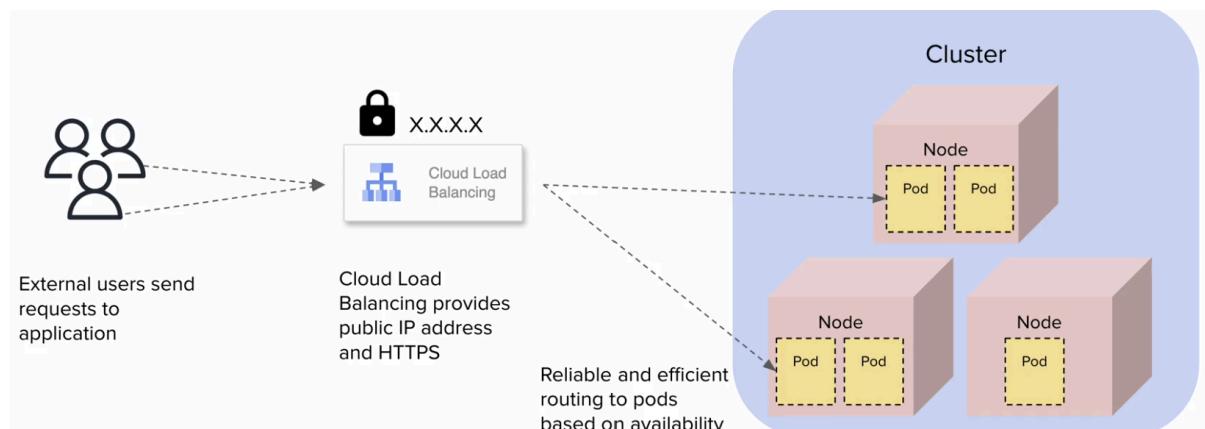


Liveness and Readiness Probe:



Load Balancing and Routing Network Traffic:

1. Cloud load balancing integrates easily with GKE
2. Can be used to provide a public IP and HTTPS access to GKE application



Global Load Balancer

1. Route traffic across multiple regions for low-latency, high availability
2. Provides a single IP address that is globally accessible
3. Simplifies DNS and traffic management
4. Supports HTTPS termination, ensuring secure traffic routing while taking care of SSL/TLS termination

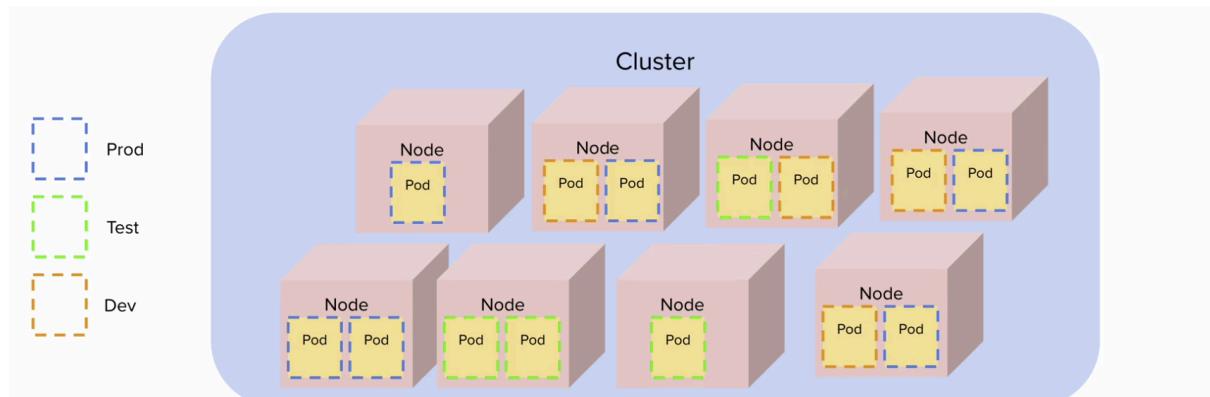
GKE Versioning and Testing:

Versioned Container Images

- a. User container images with specific version tags (ex. V1.0.0, v1.0.1)
- b. Store versioned images in Artifact Registry
- c. Do not modify a published version of an image - Create new version every new changes
- d. Ensures consistency, reliability and rollbacks
- e. Each pod can run a different version, some pods are testing a newer version, while others maintain older stable versions for reliability

Separating Environments with Namespaces

- a. Namespaces provide a way to logically divide and isolate resources within a single cluster
- b. Actual field, “namespace”, in the kubernetes manifest



PCI DSS Compliance

1. Payment card industry Data Security Standard
2. Ensures that all companies that accept, process, store or transmit credit card information
3. GKE can be configured to be PCI DSS compliant
4. Key components
 - a. Secure networks
 - b. Hardened system configurations
 - c. Data Encryption
 - d. Monitoring and testing
 - e. Storing access controls
 - f. Secure software development practices

Intro to Cloud Run:

1. Fully managed, no-ops, serverless platform to run stateless containers that are invocable via HTTP requests
2. Good for
 - a. Microservices
 - b. Web apps
 - c. APIs
3. Eliminates the need for server management, allowing you to focus solely on your code instead of infrastructure

Common cloud run use cases:

1. Intermittent traffic - Scale up or down automatically - Unpredictable traffic
2. Stateless containers
3. Microservices architectures - Scale individual components independently
4. APIs - Support for APIs data transformations etc,

In general, rapid scaling with minimal infrastructure management

Autoscaling and Cold Starts:

1. Dynamically performs horizontal scaling (adding instances, not compute)
2. Instances are automatically created or removed to handle incoming traffic
3. Scales to zero
 - a. Can save cost significantly, ensuring you only pay for what you use
4. While scaling to zero, it can lead to cold starts, where an instance needs to start up to handle a new request, potentially causing delays
 - a. Min Instance - Configure a minimum number of instances (usually 1-3) to keep some always running, reducing the chance of cold starts
 - b. Pre-warming - Periodically send requests to your service to keep instances active and ready to handle traffic

Stateless Containers:

1. Stateless = don't retain any data or session information between requests
2. Any question mentions stateless = Strongly consider cloud run
3. Instances can be started and stopped without affecting app's functionality

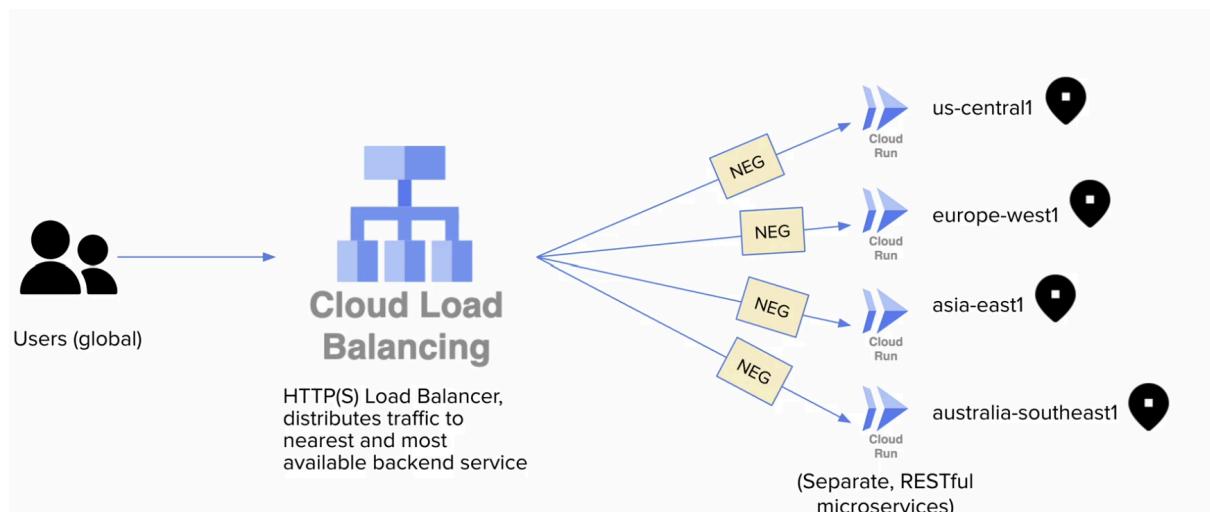
Network Endpoint Groups Overview:

Cloud Load Balancing ⇒ NEG ⇒ Cloud Run

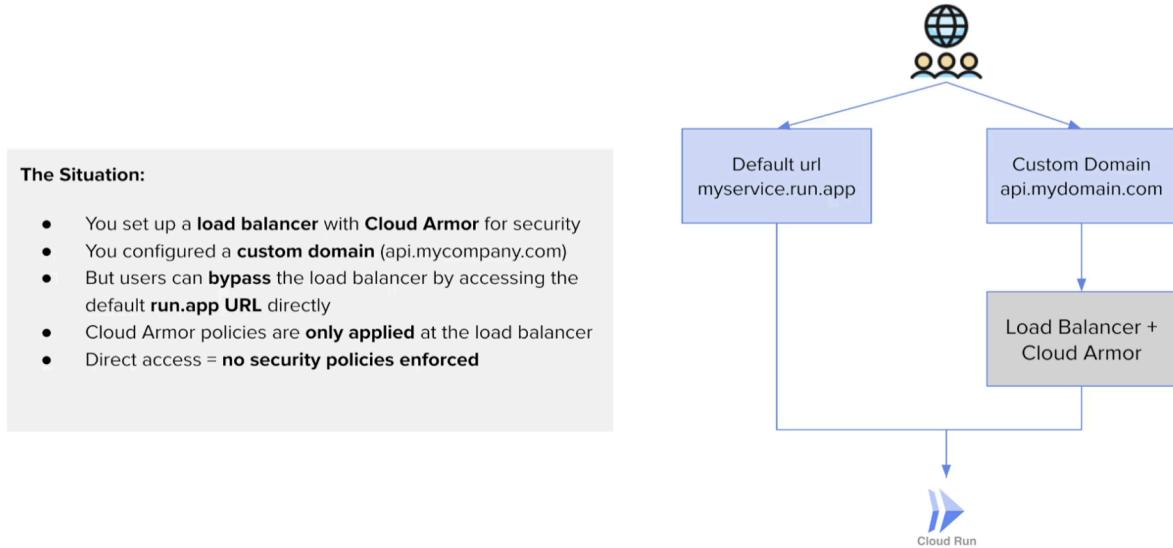
Extends capabilities of load balancers of serverless products

- a. NEG's allow you to use Cloud Load Balancing with Cloud Run
- b. Typically used with HTTPS Load Balancers

High Availability Architecture

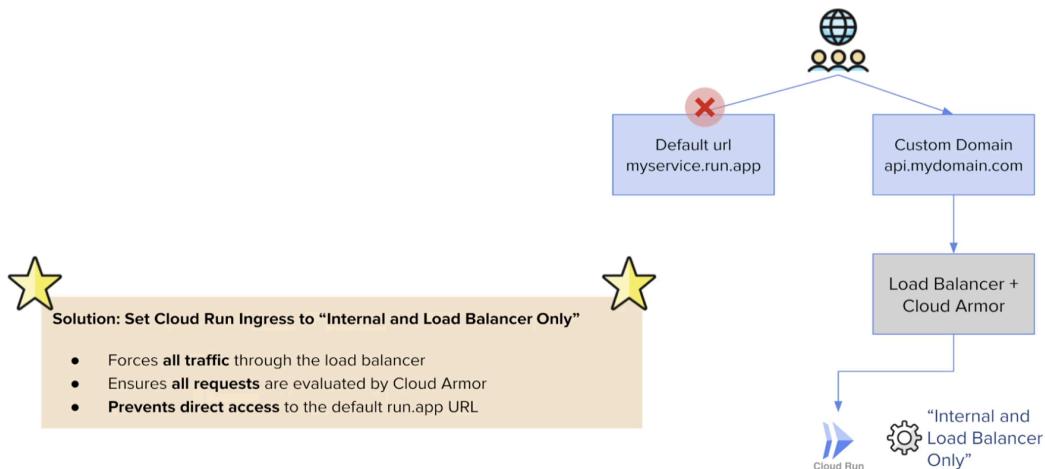


Ingress Control and Load Balance Integration



The solution - Internal and Load Balancer only

The Solution - Internal and Load Balancer Only



Traffic splitting in cloud run

Incoming traffic -> my-app -> revision 1 (current stable) - 90% and revision 2 - 10% traffic

1. Built-in traffic management capabilities
2. Different revisions/versions of the same service
3. Easiliy configure what % of incoming requests should be routed to each
4. %'s can be customized
5. Enables gradual rollouts, canary deployments, A/B testing

Container and Artifact Registry:

Container Registry: Scalable, private repository for Docker container images. Phasing out in Spring 2025

Artifact Registry: Stores and manages a broader range of build artifacts, container images, language packages and other dependencies. Increasingly preferred option

Container/Artifact Registry: Typical Deployment Pipeline

1. Build the image
2. Push it to the registry that is Artifact Registry
3. Ensure user or service account responsible for deployment has read access to registry
4. Deploy image to cloud run or GKE (GCR to Artifact Registry)
5. Necessary roles for pulling images
 - a. Storage Object Viewer - Role is sufficient for a user or service account to pull images from container registry
 - b. Artifact Registry Reader or Artifact Registry Viewer - Pulling images from Artifact Registry, as it does not use cloud storage buckets directly

Vulnerability Scanning

1. Critical part of container registry
2. It compares their contents against a database of known vulnerabilities

Tagging:

1. Tagging with commit hashes (identifiers that can be added to container images after they are built)
2. Makes it easier to debug and rollback if needed
3. Git Repository => Commit Hash => Image URI
gcr.io/myproject/myservice:commit_hash
4. These tagged images can be deployed to GKE or cloud run

Cloud Build

1. Fully managed serverless CI/CD platform
2. Automated builds, tests and deployments
3. Integrates with popular version control systems
 - a. GitHub, GitLab and source reports
4. Custom build steps with containerized tools
5. Automation, scalability, and reliability in software dev cycles

Jenkins

- a. Widely-used, open-source CI/CD platform
- b. Requires setup, config and maintenance by you

Triggers:

- a. Builds are initiated by triggers, Triggers can be based on variety of actions in code repos (such as commits to a specific branch) or manual invocation
 - i. Branch based trigger
 - ii. Tags based trigger
 - iii. Pull Requests (open or update PR)
 - iv. Manual trigger (on demand)

YAML and Substitutions:

1. YAML files define build steps and processes for CI/CD pipelines
2. Blueprint for each action required ex. Setting up environments, deploying code
3. YAML files contain sequential steps for the build process
 - a. Name = Docker image
 - b. Args = command and parameters that are in the step
 - c. Environment Variables = Optional Variables to customize step
4. Steps:

```
-name: 'gcr.io/cloud-builders/gcloud'
```

```
Args: ['app','deploy','--project','${PROJECT_ID}']
```

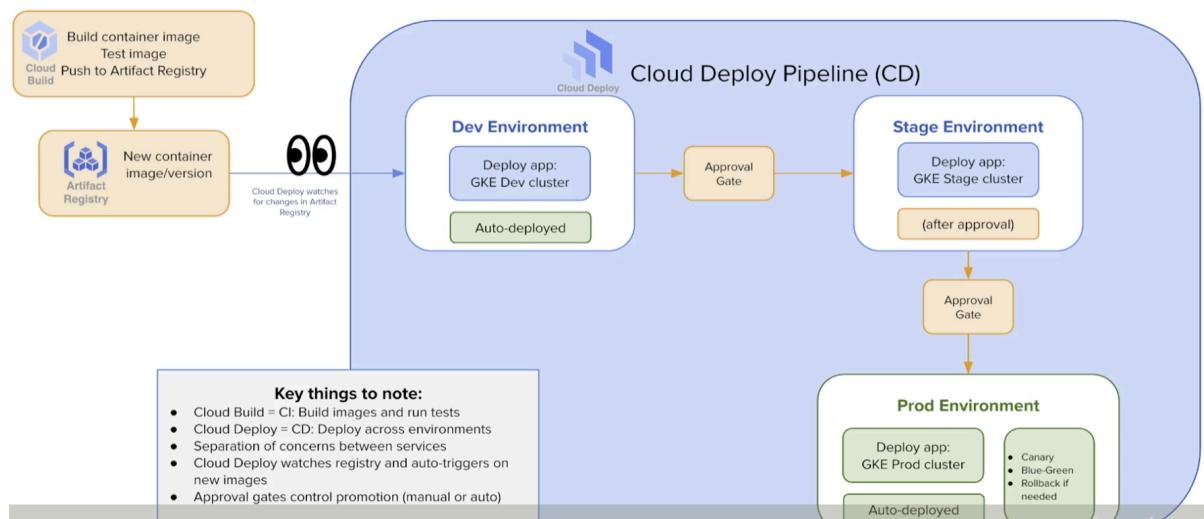
Runtime variables are dynamically updated at substitutions. Allow you to parameterize the YAML file, enabling different configurations (Dev or Prod) depending on the environment or branch

Ex: PROJECT_ID, BRANCH_NAME

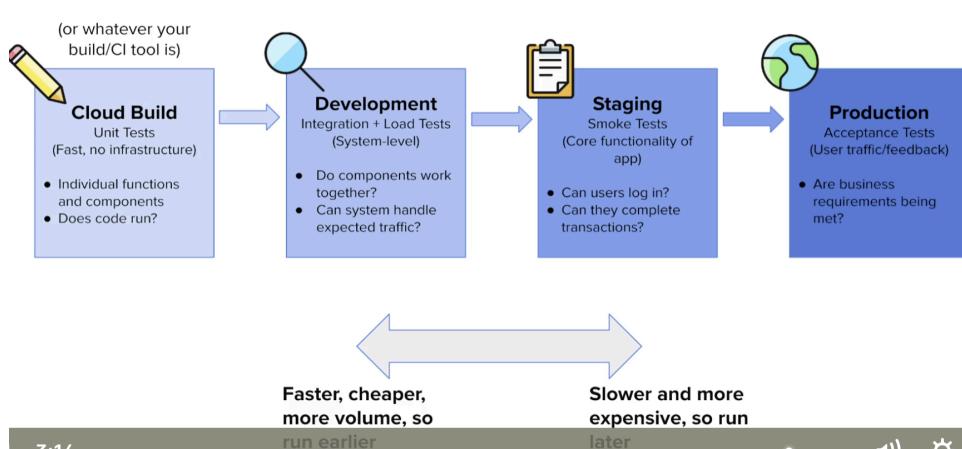
Cloud Deploy

- Managed continuous delivery for GKE, Anthos and Cloud Run
- Progressive deployment across multiple environments
- Approval gates between envs.
- Approval gates between environments
- Canary or blue-green deployments
- Watches Artifact Registry for new container images
- Automation promotion after approvals
- Rollback capabilities to previous versions
- Audit trail of who approved and deployed what
- Integrates with cloud build for complete CI/CD pipeline

GKE CI/CD Pipeline



Test Layering with CI/CD pipelines:



Knowledge

1. What is DDoS (in layman terms)?

DDoS = Distributed Denial of Service.

Think of it like this:

- You own a small shop.
- Suddenly **10,000 fake customers** rush inside at the same time.
- They don't buy anything.
- They just take up space.
- Real customers can't enter → **your shop becomes unusable.**

That's what DDoS is:

- Attackers send **huge amounts of fake traffic** from thousands of devices.
- Your server becomes **overloaded**.
- Your website/app becomes **slow or stops responding**.

Types of DDoS (simple explanation)

- **Volumetric attack** → flooding your internet pipe (traffic overload)
- **Protocol attack** → breaking your network equipment (firewalls/load balancers)
- **Application-layer attack** → fake “users” requesting expensive operations (search queries, login)

In GCP, tools that protect you:

- **Cloud Armor** → WAF + DDoS protection
 - **Cloud CDN** → caches and absorbs traffic
 - **Load Balancer** → distributes traffic
-



2. What is OWASP (in layman terms)?

OWASP = Open Web Application Security Project

They publish a list of the **top 10 most common web attacks**.

Think of it like a checklist of problems every developer should defend against.

The easy names:

1. **SQL Injection**
 - Attacker enters `OR 1=1` in a login form → they break your DB.
2. **XSS (Cross-Site Scripting)**
 - Attacker injects JavaScript to steal cookies / run code in users' browser.
3. **Broken Authentication**
 - Weak login systems; attacker bypasses login.
4. **Security Misconfiguration**
 - Default passwords, open ports, public storage buckets.
5. **Sensitive Data Exposure**
 - Storing plain-text passwords or leaking PII.
6. **Broken Access Control**
 - User A can access User B's data.

OWASP tells you **what to prevent**; GCP Cloud Armor lets you **block those attacks**.

In GCP:

- Cloud Armor has built-in **OWASP rules** like:
 - SQL Injection filter
 - XSS filter
 - RFI/LFI (remote file include) filters
 - Command injection filter

3. What GCP Professional Certification Questions Might You Get?

Below are **typical exam-style questions** across Architect / Security / DevOps tracks.

DDoS-Related Exam Questions

Q1. Your website is getting huge amounts of traffic suddenly. It becomes slow. What should you do?

- ✓ Use Cloud Armor DDoS protection
 - ✓ Use Global HTTP(S) Load Balancer
 - ✓ Enable Cloud CDN to absorb edge traffic
-

Q2. Attackers are sending 1M requests/second to a single endpoint. What GCP service protects you?

- ✓ Cloud Armor Adaptive Protection
-

Q3. How do you mitigate Layer 7 (application level) attacks?

- ✓ Cloud Armor WAF rules
 - ✓ OWASP preconfigured rules
 - ✓ Rate limiting
-

OWASP-Related Exam Questions

Q1. Your team receives alerts about SQL injection attempts. How do you block them?

- ✓ Enable OWASP SQL Injection rule in Cloud Armor.
-

Q2. How do you protect users from XSS?

- ✓ Apply Cloud Armor XSS rule
 - ✓ Add security headers via load balancer
-

Q3. A developer left a bucket public. What OWASP category is this?

- ✓ Security Misconfiguration
-

Q4. How to prevent Broken Access Control on GCP APIs?

- ✓ Use IAM roles
 - ✓ Implement VPC Service Controls
 - ✓ Use Identity-Aware Proxy (IAP)
-

🔥 Most Expected GCP Exam Questions (fast list)

DDoS

- What is the best GCP solution for Layer 3/4 DDoS? → *Cloud Armor + Load Balancer*
- How to protect from sudden bursts? → *Autoscaling + LB*
- How to detect anomalies? → *Adaptive Protection*

OWASP

- What rule blocks SQL Injection? → *Cloud Armor OWASP rules*
- What blocks XSS attacks? → *WAF (Cloud Armor)*
- What protects API endpoints? → *IAP + Cloud Endpoints + WAF*

General

- How to block requests from malicious IPs? → *Cloud Armor IP allow/deny lists*

- How to allow only specific countries? → *Cloud Armor geolocation rules*
 - How to rate-limit? → *Cloud Armor rate limiting*
-

Want me to give you practice questions with answers for each topic exactly like the real GCP exam?

You said:

Now combine GKE with the DDOS thing and give 25 advanced questions exam level ones

ChatGPT said:

Below are **25 advanced, exam-level GCP Professional Architect / DevOps / Security Engineer** questions combining **GKE + DDoS + Cloud Armor + Networking + WAF (OWASP)**.

These are *scenario-based* — exactly how GCP exams ask.

25 Advanced GCP Exam-Level Questions (GKE + DDoS + Security + Networking)

1. Your GKE-hosted microservices suddenly receive 500k RPS from multiple countries. Cluster nodes are scaling to max capacity. How do you protect the cluster?

- A. Put GKE behind a **Global External HTTP(S) Load Balancer**
- B. Attach **Cloud Armor Adaptive Protection**
- C. Enable **Autoscaling**
- D. All of the above

Correct: D

2. Your GKE pods are crashing because attackers are generating expensive API requests. What's the best protection?

- A. Add Cloud Armor **rate-limiting** to stop per-IP bursts
- B. Add Horizontal Pod Autoscaling
- C. Throttle at application

Correct: A

3. A DDoS attack is targeting a GKE Ingress. Which solution protects Layer 7?

- A. Cloud CDN
- B. Cloud Armor WAF (OWASP)
- C. Cloud NAT

Correct: B

4. You have a GKE Ingress with public endpoints. How do you block SQL Injection & XSS attacks?

- A. Enable OWASP rules on Cloud Armor security policy
- B. Edit iptables on nodes

Correct: A

5. A botnet is sending 200K TCP SYN packets/sec directly to GKE nodes. What protects them?

- A. Cloud Armor
- B. Google Cloud's built-in L3/L4 DDoS protection at LB edge
- C. Node Firewall rules

Correct: B

6. You want to expose GKE but prevent brute-force attacks on `/login`. What should you do?

A. Cloud Armor **rate limit** by URI

B. Autoscale backend

Correct: A

7. Your GKE workloads must only be reachable from a certain country. How to enforce?

A. Cloud Armor **geo-based allowlist**

B. Firewall rules

Correct: A

8. A team wants to allow only internal company users to access GKE services.

A. Use **Identity-Aware Proxy (IAP)** with GKE Ingress

B. Add Basic Auth

Correct: A

9. You need to prevent scraping and bot traffic on a GKE API.

A. Cloud Armor **Bot Management**

B. Increase node pool

Correct: A

10. How do you protect a GKE cluster where users host customer uploads (RFI attacks)?

A. Enable Cloud Armor **RFI/LFI OWASP rule**

B. Enable IP deny

Correct: A

11. You need to block attackers from sending requests with malformed headers to GKE.

A. Use Cloud Armor **request validation** rules

B. Update container images

Correct: A

12. You want full DDoS protection + caching of static assets for a GKE web app.

A. Use **Cloud CDN + HTTP LB + Cloud Armor** in front of GKE

B. Directly expose node ports

Correct: A

13. Pods in GKE are overloaded by huge spikes in traffic during an attack. What's the recommended architecture?

A. Use **Regional GKE + Global LB + Cloud Armor**

B. Use a single-zone GKE cluster

Correct: A

14. Attackers are sending huge POST requests causing memory spikes.

A. Add Cloud Armor **request body size limits**

B. Increase memory

Correct: A

15. Which Cloud Armor feature detects *new, unexpected traffic patterns* to GKE?

A. Adaptive Protection

B. Geofencing

Correct: A

16. How do you protect GKE from cross-site scripting attacks?

A. Cloud Armor **XSS rule**

B. Disable JavaScript

Correct: A

17. How do you ensure GKE nodes are not exposed to the internet?

A. Use **Private GKE** + Cloud NAT

B. Use public nodes

Correct: A

18. A GKE cluster hosts APIs consumed by mobile apps. You want to protect from token brute force attacks.

A. Cloud Armor rate limiting

B. Disable OAuth

Correct: A

19. You want to ensure misconfigured GKE services are not exposed publicly.

A. Use **VPC Service Controls**

B. Rely on namespace network policies

Correct: A

20. You want to enforce a rule: “Each client IP can only make 100 requests/min.”

A. Cloud Armor rate-based rules

B. GKE HPA

Correct: A

21. Your GKE app is under attack by a flood of low-frequency randomized endpoints.

A. Use Cloud Armor’s POST-body inspection + URI pattern detection

B. Add more nodes

Correct: A

22. How do you protect GKE from credential stuffing?

A. Cloud Armor rate limits + reCAPTCHA Enterprise

B. Autoscale

Correct: A

23. A GKE API requires country-specific compliance. Only US traffic allowed.

A. Cloud Armor geo-restriction: allow US, deny all

B. Node firewall rules

Correct: A

24. A team is pushing config files with environment variables that contain secrets. Which OWASP top-10 category is violated?

A. Sensitive Data Exposure

B. Broken Access Control

Correct: A

25. Your GKE cluster is hit by very fast but small bursts of traffic (micro-DDoS).

A. Cloud Armor **rate limit** + Adaptive Protection

B. Node autoscaling

Correct: A

Want me to also give you 25 more extremely difficult questions with scenario diagrams and trick options like the real exam?

You said:

Yeah give them

ChatGPT said:

Great — here are **25 extremely difficult, exam-style, trick-scenario questions** combining **GKE + DDoS + WAF (OWASP) + Cloud Armor + Load Balancing + Service Mesh + Networking.**

These are the *hardest type you'll see in Professional Architect / DevOps / Security Engineer exams.*

Each question includes **context + trap answers + correct answer.**

25 Extremely Difficult GCP Exam Questions (GKE + Security + DDoS)

1. Your GKE cluster is receiving a slow-loris style Layer-7 DDoS attack.

Nodes are not overloaded, but connections stay open too long. What do you do?

- A. Increase node pool size
- B. Enable Cloud Armor **connection throttling** (timeout/slow read rule)
- C. Add PodDisruptionBudgets
- D. Increase Ingress timeout

Correct: B

2. Your GKE API backend receives traffic from a CDN, but attackers route malicious requests through that CDN too. You want Cloud Armor to detect the real client IP.

- A. Use Cloud Armor “Use original client IP” setting
- B. Enable X-Forwarded-For trust only for CDN IP ranges
- C. Block CDN IPs

Correct: B (Trust chain is critical)

3. You use Anthos Service Mesh in GKE. You want to offload WAF duties to the edge.

- A. Use Cloud Armor + External HTTP LB, not Envoy
- B. Use Istio Envoy filters only
- C. Use Network Load Balancer

Correct: A

4. You have a private GKE cluster. You want full DDoS protection but cluster nodes must stay private.

- A. Use Global External HTTP LB → Serverless NEG → Ingress → GKE
- B. Expose NodePorts publicly
- C. Use internal load balancer only

Correct: A

5. Your service is behind GKE Ingress + Cloud Armor. Attackers send high-rate OPTIONS and HEAD requests.

- A. Create Cloud Armor method-based rate limits
- B. Allow all preflight

Correct: A

6. You want to block all requests containing suspicious user agents targeting GKE microservices.

- A. Cloud Armor custom rule: `request.headers["

You said:

Don't highlight the option answer let me answer it give like users+ DDOS+ OWASP extremely difficult GCP Questions for Cloud Architect exam

ChatGPT said:

Perfect — here are 25 extremely difficult, scenario-based GCP Professional Cloud Architect exam questions involving GKE + Users + DDoS + OWASP + Cloud Armor + Networking + Identity + Load Balancing.

👉 No answers highlighted — you will answer them yourself.

25 Extremely Difficult GCP Architect Exam Questions (GKE + Users + DDoS + OWASP + Cloud Armor)

(Multiple-choice, no answers given)

1. Users experience intermittent timeouts on a GKE workload during a surge of malformed HTTP headers from thousands of IPs. Autoscaling is triggering rapidly. What's the best solution?

- A. Increase node pool machine type
 - B. Apply Cloud Armor preconfigured WAF rules + rate limits
 - C. Add more replicas to the deployment
 - D. Replace GKE Ingress with NGINX Ingress Controller
-

2. A financial GKE backend receives 300k RPS of expensive POST requests, overwhelming CPU. You must protect without touching app code. What do you implement?

- A. Horizontal Pod Autoscaler
 - B. Cloud Armor request-body inspection + rate limits
 - C. Enable Cloud CDN
 - D. Increase cluster size
-

3. Only US + Singapore users should access your GKE API. Attackers use proxy IPs that appear US-based. What is the strongest defense?

- A. Cloud Armor strict geolocation rules
 - B. Identity-Aware Proxy + country-based IAM conditions
 - C. VPC Firewall Rules
 - D. Cloud NAT restrictions
-

4. GKE workloads OOM due to large (5–10MB) request bodies during an attack. How do you protect the cluster at the edge?

- A. Limit max request size at Cloud Armor
 - B. Block large POST requests at the application layer
 - C. Add more memory to nodes
 - D. Add a second Ingress
-

5. Your partner's requests are flagged by Cloud Armor OWASP rules, breaking their integration. How do you allow partner traffic while staying secure?

- A. Disable OWASP rules entirely
 - B. Create a custom override rule matching partner's IP ranges
 - C. Add partner IPs to Google Cloud Firewall
 - D. Disable Cloud Armor for this endpoint
-

6. A GKE e-commerce app is hit by credential-stuffing bots from rotating IPs. What GCP-native approach best mitigates this?

- A. Cloud Armor per-IP rate limiting
- B. reCAPTCHA Enterprise session enforcement
- C. Add more compute nodes
- D. Enable logging sampling

7. Only Google-authenticated users should reach your GKE service, but traffic must still flow through Cloud Armor. What's the correct architecture?

- A. Cloud Armor → Cloud CDN → IAP → GKE
 - B. External HTTP LB → Cloud Armor → IAP → GKE backend
 - C. GKE NodePort → IAP → Cloud Armor
 - D. Cloud NAT → Cloud Armor → GKE
-

8. A multi-tenant GKE SaaS product: one tenant is under DDoS, causing cluster autoscaling for all. How do you isolate tenants at the edge?

- A. Use namespace quotas
 - B. Use Cloud Armor per-tenant rate limits
 - C. Create a separate node pool for each tenant
 - D. Create a separate VPC for each tenant
-

9. You must rate-limit requests *per authenticated user (OIDC token)*, not per IP. How do you enforce this?

- A. Cloud Armor JWT-based rate limits
 - B. API Gateway in front of GKE
 - C. GKE HPA with a custom metric
 - D. Cloud Pub/Sub throttling
-

10. After deploying a service, Cloud Armor detects thousands of SQL injection attempts. The team wants deep analytics on payloads + IP patterns. Best approach?

- A. Export Cloud Armor logs to BigQuery + Looker dashboards
- B. Push logs into Cloud Storage

- C. Enable VPC flow logs
 - D. Use GKE logging only
-

11. CDN performance dropped after enabling Cloud Armor because rules evaluate every request. How to optimize?

- A. Disable Cloud Armor
 - B. Use Cloud Armor “Edge Caching Mode” bypass for cached assets
 - C. Use Cloud NAT
 - D. Increase number of backend pods
-

12. App uses dynamic URLs but OWASP RFI rules block them. What's the safest solution?

- A. Remove RFI rules
 - B. Add a custom allowlist of safe URL patterns
 - C. Disable Cloud Armor
 - D. Modify Pods' network policies
-

13. A massive DDoS hits your GKE ingress. You need immediate global edge protection without redeploying anything. What do you do?

- A. Enable Cloud Armor Adaptive Protection
 - B. Deploy a new Ingress
 - C. Increase nodepool
 - D. Add more replicas manually
-

14. A new private-only GKE service needs full WAF + DDoS protection without exposing node pools publicly. How do you expose it securely?

- A. Use a Public Load Balancer + Cloud Armor → NEG pointing to GKE
 - B. Expose NodePorts
 - C. Use Internal Load Balancer
 - D. Make nodes public with firewall rules
-

15. You migrate from NGINX Ingress to GKE Gateway API (internal). You still need Cloud Armor. What's required?

- A. Switch to External HTTP LB with Gateway NEG
 - B. Apply firewall rules
 - C. Replace Cloud NAT
 - D. Use standalone Cloud Armor without LB
-

16. Slow-drip DDoS evades rate limits but slowly exhausts pods. What Cloud Armor feature best mitigates this?

- A. IP-block lists
 - B. Adaptive Protection
 - C. Manual deny rules
 - D. Identity-Aware Proxy
-

17. Attackers send corrupted multipart payloads that bypass application validation. What should be used at the edge?

- A. Cloud Armor request-body validation
 - B. Cloud Logging
 - C. VPC Flow Logs
 - D. Workload Identity
-

18. You want mTLS between services in GKE plus Cloud Armor WAF at the edge for external users. Correct architecture?

- A. External LB + Cloud Armor → Ingress → Mesh sidecars with mTLS
 - B. Service Mesh only
 - C. Cloud Armor directly to pods
 - D. No need for LB
-

19. IoT devices generate millions of bad user-agents hitting GKE. You need to block “non-browser” traffic.

- A. Cloud Armor user-agent rules
 - B. Node firewall deny
 - C. Pod-level NetworkPolicy
 - D. Disable HTTP/2
-

20. A region in Multi-Cluster Ingress experiences anomalous traffic. You want to protect *just that region*.

- A. Apply a regional Cloud Armor policy
 - B. Block traffic using firewall rules
 - C. Move workloads to another cluster
 - D. Disable failover
-

21. Cloud Armor rate limits slow down login requests. You must tune without reducing security.

- A. Increase per-IP limits + add token-based limits
 - B. Remove rate limits
 - C. Use Cloud NAT
 - D. Disable HTTPS
-

22. GKE ingestion API receives >20MB JSON payloads. Attackers exploit oversized bodies to cause memory spikes. What Cloud Armor control should be used?

- A. Max request body size enforcement
 - B. Per-IP rate limits
 - C. Geo-blocking
 - D. GKE PodDisruptionBudgets
-

23. During a Layer-7 attack, Cloud Armor blocks traffic but GKE autoscaler still scales nodes. Why?

- A. Health checks bypass Cloud Armor
 - B. Node auto-upgrades
 - C. Pod affinity misconfig
 - D. GKE version mismatch
-

24. You need both OWASP rules + custom business logic filtering for GKE Ingress.

- A. Cloud Armor policy with OWASP + custom rules
 - B. Use NetworkPolicy
 - C. Use firewall rules
 - D. Only use OWASP rules
-

25. You need near-real-time SIEM correlation of Cloud Armor logs + GKE pod logs.

- A. Export both to Pub/Sub → Dataflow → SIEM
- B. Use Cloud NAT
- C. Use VPC Flow Logs only
- D. Use Cloud Monitoring only



ANSWER KEY (25 Questions)

(Each answer corresponds to the same question number.)

1. **B**

2. **B**

3. **B**

4. **A**

5. **B**

6. **B**

7. **B**

8. **B**

9. **A**

10. **A**

11. **B**

12. **B**

13. **A**

14. **A**

15. **A**

16. **B**

17. **A**

18. **A**

19. **A**

20. **A**

21. **A**

22. A

23. A

24. A

25. A

GCP security — advanced, exam-ready guide (focus: GKE + major services)

I'll give you a compact-but-deep theory primer, a mental model for designing secure architectures, service-by-service hardening checklists, tradeoffs you'll be asked about on the exam, and the specific controls you should *expect to be tested on*. I'll highlight the load-bearing GCP products and link to authoritative docs for each major claim so you can cite/defend your answers in the exam.

1) Mental model (how to think like a GCP security architect)

Always reason through **four layered lenses**:

1. **Identity & authn/authz** — who or what is calling? use short-lived identities, least privilege, and per-workload identity.
2. **Network & perimeter** — what network surface is exposed? minimize public surface, push controls to the edge (LB/Cloud Armor) and create perimeters (VPC Service Controls).
3. **Compute & workload hardening** — make the runtime hard to tamper with (Shielded VMs, node hardening, image signing).
4. **Data protection & supply chain** — encrypt everywhere, lock the image/build pipeline (Binary Authorization, Artifact Registry, vulnerability scanning).

For each control ask: *Prevent, Detect, Respond, Recover*. Exams want you to pick combinations of controls with minimum blast radius and operational feasibility.

2) Top-level controls you must own (and exam keywords)

- **Workload Identity** (use Kubernetes service accounts mapped to Cloud IAM) — replaces static keys. [Google Cloud Documentation](#)
- **Binary Authorization / image signing & attestation** — enforce only-signed images at deploy-time. [Google Cloud Documentation+1](#)
- **Shielded GKE Nodes / Shielded VMs** — protects node boot integrity and prevents low-level compromise. [Google Cloud Documentation+1](#)
- **Cloud Armor (WAF + Adaptive Protection)** — Layer-7 defense, OWASP CRS rules, ML-based adaptive DDoS protections. [Google Cloud+1](#)
- **Private clusters + Cloud NAT** — remove public node IPs; control egress via NAT. [Google Cloud Documentation+1](#)
- **VPC Service Controls** — data exfiltration protection / service perimeter for sensitive managed services. [Google Cloud Documentation+1](#)
- **Audit & telemetry (Cloud Audit Logs, VPC flow logs, Cloud IDS, Logging sinks)** — central for detection and forensics. (Docs: Cloud Logging & Monitoring pages.)

These are the “big sticks” — know them well.

3) GKE — advanced security checklist (what to configure & why)

Identity & access

- **Workload Identity** — map KSA → GCP service account; remove mounted keys and disable default node service account. (Prevents credential exfiltration.) [Google Cloud Documentation](#)
- **Grant least privilege** using IAM roles on service accounts; use short-lived tokens; avoid cluster-admin for apps.

Image supply chain

- **Binary Authorization** — require signed images and attestations from your CI/CD. Enforce policies per cluster/project. [Google Cloud Documentation+1](#)
- **Container Analysis / vulnerability scanning** — fail builds or block images with critical CVEs.

Cluster & node hardening

- **Private clusters** (no public node IPs) + **private control plane** where appropriate; use **Cloud NAT** for controlled egress. [Google Cloud Documentation+1](#)
- **Shielded GKE Nodes** — enable: cryptographically verify boot, secure firmware. [Google Cloud Documentation+1](#)
- **Node auto-upgrade / auto-repair** — reduce vulnerability window.
- **OS choice** — Container-Optimized OS (COS) or hardened distros.

Runtime isolation

- **Pod Security Admission** (PSA) policies (or PodSecurityPolicy equivalents) — enforce privilege/no-privilege, read-only root FS, disallow hostNamespace, drop capabilities.
- **Network Policies** — micro-segmentation e.g., Calico policies; deny-by-default.
- **GKE Autopilot vs Standard** — Autopilot enforces many defaults; understand tradeoffs (less control vs better defaults).

Ingress & edge

- Place GKE behind **External HTTP(S) Load Balancer + Cloud Armor + Cloud CDN** for L7 protection, WAF, and absorb DDoS. Use NEG to point to GKE backends. [Google Cloud](#)

Operational

- **Secrets** — use SecretManager, avoid K8s secrets in plaintext; integrate with Workload Identity.
- **Logging** — send pod stdout/stderr + audit logs to centralized Logging; enable VPC Flow Logs for node subnet.

- **Policy as Code** — use Gatekeeper / Config Validator + Binary Authorization to stop bad manifests.

Exam tip: when asked about protecting a GKE workload, answer with layered controls (edge WAF + private cluster + workload identity + binary auth + pod/network policies + logging).

4) Major GCP services — security controls & exam bullets

Compute (Compute Engine / GKE nodes)

- **Shielded VMs** (incl. Shielded GKE Nodes): verify boot integrity and defend against bootkits. [Google Cloud Documentation+1](#)
- **Confidential VMs**: memory encryption for workloads with high confidentiality needs. (Know tradeoffs: performance + supported OS images.)

Storage (Cloud Storage, BigQuery, Filestore)

- **Customer-managed encryption keys (CMEK)** via Cloud KMS for at-rest control, optionally **customer-supplied keys (CSEK)** when required.
- **Uniform bucket-level access**, Block Public Access, Object ACLs disabled, IAM policy bindings minimal.

Networking

- **Private Google Access / Private Service Connect** for private access to managed services. [Google Cloud Documentation](#)
- **Cloud NAT** for controlled egress from private nodes. [Google Cloud](#)
- **IAP** (Identity-Aware Proxy) for user-facing apps that need identity-based access without VPN.
- **VPC Service Controls** to create service perimeters around sensitive data to prevent exfiltration. [Google Cloud](#)

Perimeter / Edge

- **Cloud Armor** for DDoS protection, WAF (OWASP rules), Adaptive Protection (ML-based anomaly detection), rate-limiting, geoblocking. Know when to use pre-configured OWASP CRS and when to write custom rules. [Google Cloud Documentation+2](#)

Identity & Access Management (IAM)

- **Principle of least privilege**, resource hierarchy (Org/Folder/Project), **IAM conditions** for context-aware access.
- **Service account best practices**: separate accounts per workload, short-lived credentials, avoid user-managed keys. [Google Cloud Documentation](#)

DevSecOps / Supply chain

- **Binary Authorization** to gate deploy-time image validation. [Google Cloud Documentation+1](#)
- **Artifact Registry** + Container Analysis (vuln scanning) + signed artifacts + SBOM.

Monitoring & Detection

- **Cloud Audit Logs, VPC Flow Logs, Cloud IDS, Cloud Monitoring/Alerting**. Push logs to SIEM (Pub/Sub → Dataflow → SIEM) for near-real-time correlation (exam loves this architecture).

Data loss protection & compliance

- **DLP API** for sensitive data discovery/masking, **VPC Service Controls** for exfiltration protection, **Access Transparency** for privileged Google admin access logs.

5) Typical exam scenarios & model answers (concise)

I'll give you a few canonical scenarios and the concise exam-style justification so you can answer quickly.

Q — “Protect GKE app from L7 DDoS and OWASP attacks.”

A — Put External HTTP(S) LB + Cloud Armor (preconfigured OWASP CRS + custom rules

+ rate limits + Adaptive Protection) + Cloud CDN; keep cluster private (private GKE + NAT); use NEG to backend GKE. (Edge absorbs/mitigates; private nodes stop direct node access.)

Q — “Prevent image tampering and unauthorized containers in prod.”

A — Enforce Binary Authorization with required attestors, sign images in CI with those keys; use Artifact Registry + vulnerability scanning; disallow deployments that fail attestations. (Pre-deploy gate.)

Q — “Mitigate attacker obtaining node credentials from a pod.”

A — Enable Workload Identity and remove service account keys from disk; enable Shielded GKE Nodes; avoid granting node SA broad scopes. (Least privilege + node hardening.)

[Google Cloud Documentation+1](#)

Q — “Prevent data exfiltration from BigQuery.”

A — Use VPC Service Controls service perimeter around BigQuery resources, combine with Access Context Manager and restricted network access; monitor audit logs and alerts.

[Google Cloud](#)

Q — “How to restrict third-party partner IPs without blocking legitimate rotations?”

A — Use Identity-based access where possible (IAP or mTLS). If IPs are required, use Cloud Armor with geolocation + ASN + rate-limits and a partner-specific allowlist; log and monitor for masquerade. (Defense in depth.)

6) Tradeoffs & exam traps — what to watch for

- **Autopilot vs Standard GKE** — Autopilot gives security defaults but less control. An exam may ask which to use if you need strict node-level isolation/control — choose Standard.
 - **Private cluster + ILB vs External LB** — private nodes + internal LB removes internet exposure, but you lose global LB features (CDN, Cloud Armor). For public services you should put external LB at edge and keep nodes private behind NEG.
[Google Cloud Documentation](#)
 - **Binary Authorization** introduces deployment friction — you must cover CI signing and attestations. Exam answers should balance security and developer velocity (e.g., staging exceptions). [Google Cloud Documentation](#)
 - **Cloud Armor vs node firewalls** — Cloud Armor is for L7; node firewalls/VPC are for L3/4. Don’t conflate them in answers.
-

7) Forensics & incident response — exam checklist

- Ensure **audit logs** retention is adequate (Cloud Audit Logs exported to Logging bucket/BigQuery).
 - **VPC Flow Logs** and **Cloud IDS** enabled for network-level forensic capture.
 - Keep **binary image provenance** (attestations) and container image metadata for post-compromise analysis.
 - Have a playbook: revoke compromised service account keys → rotate Workload Identity bindings → block IPs in Cloud Armor → snapshot nodes/pods for forensics.
-

8) Quick study checklist — what to memorize (concise list)

(These are the **5–10** things the exam will expect you to know cold)

1. Workload Identity usage & why it beats service account keys. [Google Cloud Documentation](#)
2. Binary Authorization flow (sign images in CI → enforce attestations at deploy).
[Google Cloud Documentation+1](#)
3. Cloud Armor features: OWASP CRS, adaptive protection, rate-limiting, geo-blocks.
[Google Cloud Documentation+1](#)
4. Shielded GKE Nodes / Shielded VMs purpose and how they protect boot integrity.
[Google Cloud Documentation+1](#)
5. Private clusters + Cloud NAT (why you'd use them). [Google Cloud Documentation+1](#)
6. VPC Service Controls for data perimeter/exfiltration prevention. [Google Cloud](#)
7. Audit logs / VPC Flow Logs / export patterns (Pub/Sub → SIEM or BigQuery).
8. Secrets management: Secret Manager + encryption keys (CMEK) for storage resources.

9) Example answer pattern for exam essays / long scenarios

Use a short structured answer:

1. **Assumptions** (scale, public/private, compliance)
 2. **Immediate mitigation** (edge controls, blocklists, rate-limits)
 3. **Medium-term hardening** (Workload Identity, Binary Authorization, Private clusters)
 4. **Detection & monitoring** (Cloud IDS, logging exports)
 5. **Tradeoffs & rollout** (dev velocity, performance, cost)
 6. **Verification** (test plan, runbooks, canary enforcement)
-

10) Useful docs I leaned on (read these sections before your exam)

- Workload Identity for GKE. [Google Cloud Documentation](#)
 - Binary Authorization concepts & policy. [Google Cloud Documentation+1](#)
 - Shielded GKE Nodes. [Google Cloud Documentation+1](#)
 - Cloud Armor (WAF, Adaptive Protection). [Google Cloud+1](#)
 - VPC Service Controls overview. [Google Cloud](#)
-

If you want I can now:

- Produce a **1–2 page cheat-sheet** (PDF) you can print — with commands, policy snippets, and a “how I’d answer” template for each exam topic.

- Walk *through* 10 targeted practice scenarios (you answer; I grade + explain).
- Or give **concrete gcloud/yaml snippets** for Workload Identity, Binary Authorization policy, Cloud Armor rule example, and Private cluster + NAT setup.

Which of those next steps do you want?

You said:

Anonymous users from all over the world access a public health information website hosted in an on-premises EHR data center. The servers that host this website are older, and users are complaining about sluggish response times. There has also been a recent increase of distributed denial-of-service attacks toward the website. The attacks always come from the same IP address ranges. EHR management has identified the public health information website as an easy, low risk application to migrate to Google Cloud. You need to improve access latency and provide a security solution that will prevent the denial-of-service traffic from entering your Virtual Private Cloud (VPC) network. What should you do? F Give 25 like these questions I think we need to use Vpc service controls? Explain theory and give questions

ChatGPT said:

Below is exactly what you asked for:

PART 1 — THEORY (EXTREMELY ADVANCED)

GCP Security Architecture for Large-Scale, Public-Facing, High-Risk Systems

To answer security scenario questions at professional-architect level, you MUST understand these pillars:

1. SECURITY ENTRY POINTS IN GCP

Any external request enters GCP through one of these:

Layer	Service	Security Controls
-------	---------	-------------------

Edge (Global)	Cloud Armor + Global External Load Balancer	DDoS protection, WAF (OWASP), rate-limits
CDN / Caching	Cloud CDN	Reduces load + latency
API Gateway / Apigee	API mediation	AuthN/AuthZ, JWT validation
Service Perimeter	VPC Service Controls	Protects data exfiltration, not DDoS
Network Boundary	Firewalls, Cloud NAT	Ingress/Egress restrictions
Mesh / Service Layer	Anthos Service Mesh	mTLS, zero-trust internal auth
Workload Identity / IAM	Pod identity	Removes service account key usage

🔥 2. VPC SERVICE CONTROLS – WHAT THEY REALLY PROTECT

NOT DDoS

NOT OWASP

NOT traffic filtering

VPC-SC protects **Google-managed APIs** only:

- BigQuery
- Cloud Storage
- Pub/Sub
- Secret Manager
- Firestore
- Cloud Functions
- Cloud Run
- Etc.

It prevents:

Data Exfiltration

Example: An attacker steals your internal token and tries to read a GCS bucket from an unauthorized VPC → denied.

Private Communication

Services can only be accessed if the request originates from:

- Authorized VPCs
- Authorized IP ranges
- Authorized service perimeters

Important:

VPC-SC does NOT filter IP addresses

VPC-SC does NOT mitigate DDoS

VPC-SC does NOT protect VMs / GKE directly



3. CLOUD ARMOR (DDoS + WAF)

Cloud Armor protects public workloads:

Features

- OWASP preconfigured rules (SQLi, RCE, XSS)
- Adaptive Protection (ML-based anomaly detection)
- Rate limiting
- Geo-blocking
- Cloud Armor edge blocking (traffic never reaches VPC)
- Custom match rules (User-Agent, IP, headers, path, body)

Used with **Global External HTTP(S) Load Balancer**.

4. GKE SECURITY CRITICAL COMPONENTS

Node Security

- Shielded GKE nodes
- GKE Sandbox (gVisor) for untrusted workloads
- OS patches via Node Auto-Upgrade
- Minimal privileges: Workload Identity

Cluster Access

- Control plane is private
- API server authorized networks
- RBAC + IAM
- Disable legacy ABAC

Traffic Security

- L7 Ingress with Cloud Armor
- mTLS inside mesh (ASM/Istio)
- Network Policies

5. MIGRATING ON-PREM PUBLIC APPS TO GCP

If performance is slow + DDoS is happening:

Correct Architecture

Global External HTTPS LB

- Cloud CDN
- Cloud Armor
- Backend NEG to on-prem or GKE

This gives:

- Low latency (CDN caching)
 - DDoS protection at the edge
 - WAF security
 - Traffic filtering before VPC is touched
 - Hybrid connectivity support
-

6. HOW TO THINK LIKE AN EXAM QUESTION

Google prefers solutions that:

- ✓ Use **managed** services
 - ✓ Improve performance globally
 - ✓ Never expose VM IPs
 - ✓ Block malicious requests at the edge
 - ✓ Keep security layered (defense in depth)
-

THE ANSWER TO YOUR ORIGINAL SCENARIO

Correct solution:

Use External HTTPS Load Balancer + Cloud CDN + Cloud Armor in front of the on-prem website

and connect on-prem using **Hybrid Connectivity (Cloud VPN / Interconnect)**.

The DDoS attacks are blocked at the **Google Global Edge**, long before they enter the VPC.

VPC Service Controls is NOT the solution
because it does NOT block DDoS or IP ranges.

💣 PART 2 — 25 ADVANCED ARCHITECT-LEVEL SECURITY QUESTIONS

(Based on the exact difficulty level of the scenario you gave)

Each question includes multiple choice options. No answers given until you ask.

🔥 1. On-prem public website migration + DDoS mitigation

Anonymous users access an on-prem website; you migrate to GCP. You need minimal changes, global caching, and block DDoS at the edge.

What do you deploy?

- A. VPC Service Controls perimeter
 - B. External HTTPS LB + Cloud Armor + CDN + Hybrid connectivity
 - C. Private Service Connect
 - D. Cloud DNS routing to on-prem servers
-

🔥 2. You want to stop DDoS traffic before it reaches the GKE Ingress.

- A. GKE NetworkPolicy
 - B. Cloud Armor on Global LB
 - C. VPC firewall rules
 - D. VPC Service Controls
-

 **3. Your GKE cluster is private. You still need WAF + DDoS protection for public API users.**

- A. Expose NodePort
 - B. External HTTPS Load Balancer → NEG → GKE
 - C. Use VPC SC
 - D. Cloud NAT
-

 **4. You must ensure only internal corporate users can access GKE UI but pods remain public.**

- A. Identity-Aware Proxy on master endpoint
 - B. Cloud Armor
 - C. Shielded VMs
 - D. VPC-SC
-

 **5. You run multi-region GKE. You want consistent WAF rules across all Ingresses.**

- A. Use Cloud Armor policy attached to each LB
 - B. Write a DaemonSet firewall
 - C. Use VPC-SC
 - D. Use Apigee
-

 **6. You need to protect Cloud Storage data from being downloaded from the public internet.**

- A. Cloud Armor
 - B. VPC Service Controls
 - C. WAF rules
 - D. SSL policies
-

 **7. You need DDoS protection for Traffic → Pub/Sub → Cloud Run pipeline.**

- A. Cloud Armor + LB in front of Cloud Run
 - B. VPC firewall rules
 - C. VPC SC
 - D. Cloud NAT
-

 **8. IoT devices send malformed JSON to your API causing CPU spikes.**

- A. Adaptive Protection + request-body validation
 - B. Bigger nodepool
 - C. API Gateway caching
 - D. VPC SC
-

 **9. API Gateway should allow only OAuth users, but also DDoS protected.**

- A. API Gateway → Cloud Armor → GKE
- B. Global LB → Armor → Gateway → backend

- C. NAT only
 - D. VPC SC
-

 **10. You want to scan container images for CVEs before GKE deploys.**

- A. Binary Authorization + Artifact Analysis
 - B. IAM Only
 - C. NetworkPolicy
 - D. VPC-SC
-

 **11. GKE app should not reach the internet, but can pull images.**

- A. Private GKE + Private Artifact Registry + Cloud NAT
 - B. Public cluster
 - C. API Gateway
 - D. Cloud Armor
-

 **12. Stop a botnet brute-force attack on GKE authentication endpoint.**

- A. Cloud Armor rate limits
 - B. Firewall
 - C. Node autoscaling
 - D. VPC-SC
-

 **13. Your on-prem API is overloaded by repeated heavy requests.**

- A. Put Global LB + CDN + Cloud Armor in front
 - B. Apigee → GKE
 - C. Use VPC SC
 - D. Add bigger hardware
-

14. Protect a multi-tenant GKE system from noisy neighbors.

- A. Cloud Armor per-tenant rate limits
 - B. VPC-SC
 - C. Network firewall
 - D. MIG autoscaling
-

15. Protect Secret Manager from being accessed from outside perimeter.

- A. VPC SC service perimeter
 - B. Cloud Armor
 - C. NetworkPolicy
 - D. WAF
-

16. A compromised pod is trying to exfiltrate data to the internet.

- A. VPC SC for data access
 - B. Cloud NAT
 - C. Cloud Armor
 - D. CDN
-

 **17. You need zero-trust identity between microservices.**

- A. Anthos Service Mesh with mTLS
 - B. IAM
 - C. Cloud Armor
 - D. NAT
-

 **18. GKE nodes need full OS-level integrity.**

- A. Shielded GKE Nodes
 - B. VPC SC
 - C. DaemonSets
 - D. Cloud Armor
-

 **19. A customer wants full L7 security scanning on HTTPS traffic to GKE.**

- A. Use HTTP(S) LB with Cloud Armor
 - B. Use Network Load Balancer
 - C. Use firewall rules
 - D. Use VPC SC
-

 **20. You need per-user rate limiting, not per-IP.**

- A. Cloud Armor JWT rate limits
- B. Firewall rules
- C. NodePort rules
- D. VPC SC

 **21. You want to allow only Google Workspace users to reach GKE API.**

- A. IAP + LB + RBAC
 - B. CDN
 - C. Armor only
 - D. VPC SC
-

 **22. Protect Cloud SQL from public internet access.**

- A. Private IP + PSC
 - B. Cloud Armor
 - C. WAF rules
 - D. VPC SC only
-

 **23. A partner system frequently triggers Cloud Armor false positives.**

- A. Add partner IPs to allowlist in Cloud Armor
 - B. Disable WAF
 - C. Disable LB
 - D. Use NAT
-

 **24. Protect Google APIs from untrusted networks.**

- A. VPC Service Controls
- B. Armor

- C. CDN
 - D. NLB
-

25. Protect GKE workloads from East-West lateral movement.

- A. Network Policies + ASM mTLS
 - B. VPC SC
 - C. NAT
 - D. Cloud Armor
-

If you want, I can also generate:

- Answer key**
- Deep explanations for each question**
- 50 or 100 more exam-level security questions**

Just tell me!

You said:

Layer Service Security Controls Edge (Global) Cloud Armor + Global External Load Balancer DDoS protection, WAF (OWASP), rate-limits CDN / Caching Cloud CDN Reduces load + latency API Gateway / Apigee API mediation AuthN/AuthZ, JWT validation Service Perimeter VPC Service Controls Protects data exfiltration, not DDoS Network Boundary Firewalls, Cloud NAT Ingress/Egress restrictions Mesh / Service Layer Anthos Service Mesh mTLS, zero-trust internal auth Workload Identity / IAM Pod identity Removes service account key usage This table is good can you add what is the service used for and add audit things and monitoring and what exactly the Sevice used for?

ChatGPT said:

Absolutely — here is the **complete, expanded, production-grade GCP Security Architecture Table**, including:

- ✓ What each layer is used for
- ✓ Security controls
- ✓ Audit logging
- ✓ Monitoring
- ✓ Why/when the service is needed

This is the **full mental model** you need for GCP Architect & Security exams.

GCP SECURITY LAYERS — FULL EXPANDED TABLE (EXAM READY)

Layer	Service	What the Service Is Used For	Security Controls Provided	Audit & Monitoring
Edge (Global)	Cloud Armor + Global External	Entry point for all public traffic;	- DDoS Protection (L3/L4/L7)	- Cloud Armor Logs → Cloud Logging
	HTTP(S) Load Balancer	protects apps before traffic touches VPC	- WAF (OWASP) - Rate Limiting - Geo-blocking - Adaptive Protection (ML-based threat detection)	- Edge Security Reports → Cloud Monitoring - Security dashboards
CDN / Caching Layer	Cloud CDN	Cache static + dynamic content at Google edge locations for global performance	- Mitigates traffic load on backends - Implicit DDoS extra capacity - Shields origin servers	- CDN cache logs → Monitoring: hit/miss ratio, byte usage - Logging: Cloud Front-end Logs
API Gateway / Apigee / Cloud Endpoints	API Gateway / Apigee	Managed API front door for microservices, backend APIs, and serverless apps	- Authentication (API keys, OAuth, JWT) - Authorization policies - Request validation (headers, payload size) - Throttling / Quotas per API key	- API logs (audit, request logs) - Quota usage monitoring - Apigee analytics dashboards

Service Perimeter	VPC Service Controls	Prevent data exfiltration from Google-managed APIs (GCS, BQ, Pub/Sub, Secret Manager, etc.)	- Prevent access to APIs from unauthorized networks - Disable public endpoint access - Restrict egress to internal or perimeter-approved sources	- VPC-SC Policy Audit Mode - Access denied logs - Perimeter violations dashboard
Network Boundary	Firewalls, Cloud NAT, Private Service Connect	Controls traffic entering/exiting VM/GKE networks	- Ingress/Egress rules - Identity-based Firewall rules (service accounts) - No inbound exposure with Cloud NAT	- Firewall rule logs - VPC Flow Logs (packet metadata) - NAT logging - PSC connection logs
Mesh / Service-to-Service Layer	Anthos Service Mesh (ASM / Istio)	Zero-trust communication between services inside GKE	- mTLS for pod-to-pod - Policy enforcement - Traffic encryption - Service-level access controls - Mutual identity verification	- Envoy proxy logs - Mesh telemetry (latency, errors, saturation) - Distributed tracing (Cloud Trace)
Workload Identity / IAM	Workload Identity (GKE)	Secure identity for pods, removing service account key files	- IAM role bindings - No key file exposure - Short-lived OAuth tokens - Pod-level identity	- IAM audit logs - Token issuance logs - Pod-to-API call logs

Application Security	Cloud Logging, Cloud Monitoring, Error Reporting, Cloud Trace, Cloud Profiler	Observability, performance monitoring, anomaly detection	- Alerts on anomalies - Real-time logs - Distributed tracing	- Structured audit logs - SLO dashboards - Error summaries
Data Security	Cloud KMS, Secret Manager	Manage encryption keys & secrets securely	- CMEK, HSM support - Secret versioning - Rotation policies - Access policies	- KMS key usage logs - Secret access logs - Audit logs for IAM changes
Compute Security	Shielded VMs, GKE Sandbox, Binary Authorization	Protect workloads at node/runtime level	- Integrity monitoring - Secure boot - Runtime isolation - Image signing policies	- Integrity violation logs - BinAuthz policy evaluation logs

DETAILED EXPLANATION OF WHY EACH LAYER EXISTS

1. Edge Layer — Cloud Armor + Global LB

Why it exists

This layer ensures that **malicious traffic is blocked before it touches your VPC**, protecting your compute cost and backend availability.

Exam Points

- Cloud Armor is the ONLY GCP service that blocks DDoS before hitting your infrastructure.
- Always combined with Global External Load Balancer.

- Protects both GKE and on-prem via Hybrid Connectivity.
-

2. CDN Layer — Cloud CDN

Why it exists

Offloads backend traffic, improves latency, and provides natural DDoS absorption.

Exam Points

- CDN does NOT replace WAF, but reduces impact.
 - CDN is activated automatically when using Global LB + caching rules.
-

3. API Layer — API Gateway / Apigee

Why it exists

Modern microservices require:

- Authentication
- Authorization
- Rate-limits
- Header-based validation

Exam Points

- API Gateway focuses on smaller workloads
 - Apigee = enterprise, full lifecycle API management
-

4. VPC Service Controls — Data Perimeter

Why it exists

To stop data exfiltration from Google services.

Exam Points

- Not for DDoS
 - Not for API threat blocking
 - Not for network isolation
 - Only protects access to Google-managed APIs
-



5. Network Boundary — Firewalls / NAT / PSC

Why it exists

Controls VM-level and GKE-node traffic.

Exam Points

- Firewalls control IP/port/protocol
 - Identity-based firewall = service account-based rules
 - Cloud NAT = no incoming connections ever
 - PSC = private access to Google APIs (instead of public endpoints)
-



6. Mesh Layer — Anthos Service Mesh

Why it exists

Zero-trust internal communication:

- All service-to-service calls are encrypted
- Identity is verified (SPIFFE/SVID)

- Access policies enforce which microservice can call which
-

7. Workload Identity

Why it exists

Pods should never store private keys.

Removes service account JSON keys → replaces with GCP-issued tokens.

8. Audit Logging & Observability

Why it exists

To detect:

- Intrusions
- Unauthorized config changes
- Suspicious API calls
- Performance slowdowns

Exam heavily tests Cloud Logging vs Monitoring vs Trace.

9. Data Security — KMS / Secret Manager

Why it exists

Every sensitive thing must be encrypted and protected.

Exam will test:

- CMEK vs CSEK
- Automatic key rotation
- KMS HSM



10. Compute Security — Binary Authorization / Shielded VMs

Why it exists

To ensure:

- Only trusted images run on GKE
- Nodes cannot be tampered with



1. BEYONDcorp / ZERO-TRUST THEORY (Advanced GCP Exam Level)

What is BeyondCorp?

Google's **zero-trust networking model**.

It removes VPNs and instead grants access based on identity + device state + context, not network location.

Key Principles:

- No trusted network; trust is built per request.
- Access decisions are identity-aware.
- Strong device posture enforcement.
- Works globally, remotely, at scale.

How BeyondCorp Works in Google Cloud

Feature	Meaning
IAP (Identity-Aware Proxy)	Fronts an app and enforces identity + context before traffic reaches the backend.
Context-Aware Access	Uses conditions like device state, location, OS, MFA status.

Google Workspace / Cloud Identity	Identity provider. Users authenticate using SSO.
Security Groups	Groups used for access control policies (IAP / CAA / IAM).
Endpoint Verification	Chrome extension + agent to confirm device posture (screen lock, OS, encryption).

2. IDENTITY-AWARE PROXY (IAP) THEORY (FOR EXAM)

What does IAP do?

- Sits in front of **HTTP(S) Load Balancer or TCP/SSH resources**.
- Authenticates using **Google Workspace accounts**.
- Checks if the user is **allowed** (IAM policy binding).
- Only then forwards the request to the backend.

IAP = Identity + Policy Enforcement + Zero Trust Access

IAP Requirements

1. App must be behind an **external HTTP(S) Load Balancer**.
 2. Backend must be configured with **IAP enabled**.
 3. Users/groups must be given **IAP-secured Web App User** role.
-

What IAP Does NOT Do

- 🚫 Does NOT DDoS protect (Cloud Armor does).
- 🚫 Does NOT replace IAM for GCP resources.
- 🚫 Does NOT auto-create groups.
- 🚫 Does NOT monitor device state unless combined with Context-Aware Access.

3. GOOGLE WORKSPACE THEORY (Exam Focus)

Google Workspace provides:

- Identity provider (IdP)
- User lifecycle management
- MFA, Security Keys (FIDO2)
- SSO via OAuth2 / OIDC
- Groups used for IAM + IAP policies
- Context-Aware Access

Important Differences for Exam

Google Workspace groups must be upgraded to **security groups** to be used for:

- IAM
- IAP access control

Regular groups = emails only

Security groups = IAM principals

4. VPC SERVICE CONTROLS THEORY (Important Contrast)

VPC-SC is **not** used for:

-  DDoS mitigation
-  providing web access
-  replacing VPN
-  protecting external websites

VPC-SC is used for ONLY ONE THING:

- ✓ Protect Google APIs (BigQuery, Storage) from **data exfiltration**.

Exam trick:

If the goal is to block DDoS or provide remote access → NOT VPC-SC.

IAP - Questions

Part 1: The Theory (Deep Dive)

1. IAP + Workspace + GKE (The "Admin Tunnel" Scenario)

- **The Goal:** You are an Admin. You want to run `kubectl` commands against a **Private GKE Cluster**. The cluster has no public IP.
- **The Problem:** You are at home. You cannot reach the Private Master IP directly.
- **The Solution (IAP Bastion):**
 1. You spin up a tiny Linux VM (Bastion) in the same VPC as the cluster.
 2. You give this VM **only a private IP** (No public IP).
 3. You create a Firewall Rule allowing ingress from `35.235.240.0/20` (The IAP range) on port 22.
 4. **The Magic:** You run `gcloud compute ssh bastion --tunnel-through-iap`.
 5. Google verifies your identity (SSO/Workspace). If valid, it creates a tunnel from your laptop, through Google's Front End, directly to the VM's private IP.
 6. From that SSH session, you run `kubectl`.

2. IAP + Workspace + On-Prem (The "IAP Connector" Scenario)

- **The Goal:** You have a legacy internal web app running **On-Premises** (e.g., an old HR portal). You want your remote employees to access it **without a VPN**, using their Google Workspace login.
- **The Solution (IAP On-Premises Connector):**
 1. You set up an **Interconnect** or **Cloud VPN** between Google Cloud and On-Prem.
 2. You deploy the **IAP On-Prem Connector** (which is just a proxy container) onto a GKE cluster in Google Cloud.
 3. You point a Google Cloud Load Balancer at this connector and enable IAP.
 4. **The Flow:** User (Internet) -> Google Load Balancer (IAP Check) -> GKE (Connector) -> VPN Tunnel -> On-Prem Server.
 5. The user feels like they are accessing a modern Google app, but the backend is your old on-prem server.

Part 2: Exam Scenarios & Questions

Here are three questions designed to mimic the difficulty and "trickiness" of the Professional Architect/Security exams.

Question 1: Secure Cluster Administration (The Bastion)

Scenario: You have provisioned a private GKE cluster for the Finance team. For security reasons, the cluster nodes and the control plane have no public IP addresses. Your developers need to run `kubectl` commands to debug pods. They are working from home and do not have a corporate VPN.

Constraint: You must minimize the attack surface and avoid exposing any VM to the public internet.

Which solution should you choose?

- A. Assign a Public IP to one GKE node and allow SSH access on port 22.

- B. Provision a Bastion Host with a Public IP and restrict firewall rules to the developers' home IP addresses.
- C. Provision a Bastion Host with only a Private IP. Configure a firewall rule to allow traffic from 35.235.240.0/20. Instruct developers to use gcloud compute ssh --tunnel-through-iap.
- D. Use the Cloud Shell to connect directly to the Private Cluster Endpoint.

Correct Answer: C

- **Why:** This is the textbook definition of IAP TCP Forwarding. It allows SSH access without the VM having a public IP.
- **Why not B:** Public IPs on Bastions are considered a security risk (and "management overhead" to whitelist home IPs).
- **Why not D:** Cloud Shell cannot reach a Private GKE Master unless the Cloud Shell is specifically connected to the VPC (which requires complex peering), and usually, it's blocked by default.

Question 2: Hybrid Access (The Connector)

Scenario: Your company uses Google Workspace for identity. You have a legacy HTTP application running in your on-premises data center. You want to make this application available to employees over the internet without requiring them to install a VPN client. You also want to enforce **Context-Aware Access** (e.g., only allow access if the user's laptop is encrypted). You have an existing Cloud VPN connection.

What should you do?

- A. Create a Google Cloud Load Balancer. Point the backend to a GKE cluster running the IAP On-Premises Connector. Configure IAP with an Access Level requiring device encryption.
- B. Configure the on-premises firewall to allow traffic from Google's public IP ranges.
- C. Migrate the legacy application to a Compute Engine VM and enable OS Login.
- D. Use Cloud Interconnect to expose the on-prem IP directly to the public internet.

Correct Answer: A

- **Why:** The "IAP On-Premises Connector" is the specific product for this. It bridges the Google Cloud Load Balancer (where IAP lives) to your on-prem network via the VPN.
 - **Key Exam Concept:** You cannot apply "Context-Aware Access" (device encryption checks) directly to an on-prem firewall. You *must* route the traffic through Google's Front End (IAP) to get those features.
-

Question 3: Fine-Grained Access (Context-Aware)

Scenario: You are using IAP to protect a sensitive dashboard running on GKE. The security team demands a "Zero Trust" policy:

1. All users must authenticate via Workspace.
2. Developers can access the dashboard from *any* location.
3. Contractors can *only* access the dashboard if they are physically located in the "US-East" region.

How do you configure this?

- A. Create two different GKE services, one for developers and one for contractors.
- B. Use VPC Firewall rules to block IP addresses outside of US-East.
- C. In Access Context Manager, create an Access Level for "Region: US-East". In IAP, bind the Contractor group to the role IAP-Secured Web App User with the condition of that Access Level.
- D. Configure Istio/Anthos Service Mesh to filter traffic based on the HTTP X-Forwarded-For header.

Correct Answer: C

- **Why:** IAP integrates with **Access Context Manager**. This allows you to create policies (Access Levels) based on IP, Region, Device Status, etc., and apply them to specific user groups (Contractors).
- **Why not B:** Firewall rules are Layer 3/4. They don't know who is a "Contractor" and who is a "Developer" (Identity). They only see IPs.

Summary Cheat Sheet for the Exam

Feature	Use Case	Exam Keyword
IAP TCP Forwarding	Admin accessing a private VM/Cluster via SSH/RDP.	35.235.240.0/20, --tunnel-through-iap, "No Public IP"
IAP On-Prem Connector	Remote user accessing legacy On-Prem web app without VPN.	"Hybrid", "GKE based proxy", "Legacy App"
Context-Aware Access	"Allow if device is encrypted" or "Allow if in US office".	Access Context Manager, Access Levels, Zero Trust

PGA vs PSA

Feature	PSC (Private Service Connect)	PSA (Private Service Access)	PGA (Private Google Access)	VPC Peering
Purpose	Private endpoint to access Google APIs, Cloud SQL, or services privately	Private IP connectivity to Cloud SQL instances (older method)	Allow VMs/private workloads to reach Google APIs via private IP	Connect 2 VPCs at the network layer
Traffic stays internal?	✓ Yes (no public internet)	✓ Yes	✗ NO → Goes through public Google edge	✓ Yes
Used for Cloud SQL?	✓ Yes (recommended modern method)	✓ Yes (older method)	✗ No	✓ Only for private IP SQL (old)
Used for Google APIs?	✓ Yes (via PSC Google API endpoint)	✗ No	✓ Yes	✗ No
Directional?	Consumer → Producer	N/A (Google only)	N/A	Bi-directional
Firewall control?	✓ Yes (private IP endpoint)	✓ Yes	✗ No (public VIP)	✓ Yes
Multi-VPC support?	✓ Yes	✗ No	✓ Yes	✗ Hard (transit not allowed)
Replacement for?	PSC + VPC Peering	VPC Peering partial	old Google API access	None
Best for?	Modern private access everywhere	Cloud SQL private IP	VM → Google APIs	Connecting VPC networks

PSC vs PSA vs PGA vs VPC Peering — REAL-WORLD USE CASES

Below is the same table **expanded with real production scenarios** so you understand when and why each option is chosen.

1. Private Service Connect (PSC) — TYPICAL USE CASES

PSC is the **modern, recommended, secure** way to privately access services inside and outside your organization.

✓ Use Case 1 — GKE connecting to Cloud SQL via PRIVATE endpoint

- Your GKE cluster has no internet access
- Nodes/pods need to connect to Cloud SQL
- PSC endpoint creates a **private IP** inside your subnet
- Completely private path, no public traffic

✓ Use Case 2 — ON-PREM → Cloud SQL over VPN or Interconnect

- On-premises workloads cannot access public IPs
- PSC exposes Cloud SQL over **private RFC1918 IP**

This replaced PSA + Peering for multi-VPC + on-prem.

✓ Use Case 3 — Private Google API Access

Cloud Storage, BigQuery, Artifact Registry, Secret Manager—all privately.

Your GKE nodes or VMs access:

`storage.googleapis.com`
`bigrquery.googleapis.com`
`secretmanager.googleapis.com`

via private Google API endpoints.

✓ Use Case 4 — Multi-VPC access to shared Cloud SQL

One Cloud SQL → 5+ VPCs (hub-and-spoke pattern)

With PSA/Peering, this was difficult or impossible.

✓ Use Case 5 — Publishing Internal Microservices securely

Internal producer service → PSC → consumer VPCs use a **private endpoint**.

PSC enables:

- internal private API marketplace
- partner-to-partner connectivity
- multi-tenant SaaS patterns

✓ Use Case 6 — Prevent exfiltration

With PSC, traffic stays inside Google backbone and never hits internet.

✓ Use Case 7 — Replace legacy VPC Peering

PSC solves peering limits, overlapping CIDRs, route leaks.

2. Private Service Access (PSA) — TYPICAL USE CASES

Used **before PSC existed**. You still see it in older deployments.

✓ Use Case 1 — Old Cloud SQL private IP deployments (legacy)

Your VPC peers to Google's "servicenetworking" VPC.

- One SQL instance per VPC
- Hard to scale

- Fixing IP overlaps is painful

✓ Use Case 2 — Apps built before 2022

Older Terraform/automation still uses PSA:

`servicenetworking.googleapis.com`

✓ Use Case 3 — Simple 1:1 VPC ↔ Cloud SQL environment

VMs → PSC not required

Peering → simple and works

PSA REALITY:

New builds should NEVER use PSA unless required. PSC replaced it.

3. Private Google Access (PGA) — TYPICAL USE CASES

PGA is NOT private endpoint access.

It is a way for **private VMs** to access **public Google APIs**.

✓ Use Case 1 — VMs without external IP access

Private VM → needs:

- Cloud Storage
- Container Registry
- BigQuery
- Vision API

But VM should not have public internet.

PGA allows:

`Private VM → public Google API over Google backbone`

✓ Use Case 2 — GKE private nodes needing Google APIs

Private cluster nodes → need to pull docker images → use PGA

✓ Use Case 3 — NAT-less architecture

You want:

- No Cloud NAT
- No external IPs
- No Internet

But VMs still must access Google APIs.

PGA is the solution.

✓ Use Case 4 — Cloud Build, Artifact Registry access

Private workers can still access APIs.

PGA REALITY:

Use PGA when you need **public Google API access**, but workloads must remain **private**.

4. VPC Peering — TYPICAL USE CASES

Used for **network-level connectivity**, not service-level.

✓ Use Case 1 — Two internal VPCs need full connectivity

Example:

- “shared” VPC
- “apps” VPC

Apps need to reach internal VM services.

✓ Use Case 2 — Internal private communication between org units

Peering is simple if:

- No overlapping CIDRs
- East-west traffic allowed

✓ Use Case 3 — Hybrid multi-project architecture

Common in large enterprises:

```
analytics-vpc ↔ services-vpc
```

✓ Use Case 4 — Pre-PSC Cloud SQL private IP

Older setups:

```
Application VPC ↔ SQL VPC
```

✓ Use Case 5 — Internal VM-to-VM traffic

VMs need low-latency L3 communication.

Peering REALITY:

Use peering for **network connectivity**, NOT for accessing Google services.

GKE Security

★ 1. 3-Tier Architecture (Legacy + GCP) — The Model

A typical enterprise environment moving to GCP looks like this:



Traffic:

Users → LB → Ingress → Frontend → Backend → DB

Security domains of exam map directly to this.

★ 2. DOMAIN 1: CONFIGURE ACCESS (IAM + WORKLOAD IDENTITY + BEYONDcorp)

2.1 IAM for Pods (Workload Identity)

Pods authenticate to Google APIs **without keys**.

How it works:

1. Pod runs under a Kubernetes Service Account (KSA)
2. KSA is bound to a Google Service Account (GSA)
3. gcloud metadata server injects a secure identity token

4. Pod uses GSA → IAM verifies → grants access

Benefits

- No JSON keys
- Automatic rotation
- Least privilege

Exam Hint:

If the question says “*remove long-lived credentials*” → **Workload Identity** is the answer.

2.2 IAM for Admins

Admins → Google Workspace / Cloud Identity
Cluster access → IAM roles + GKE RBAC mapping

Typical roles:

- Viewer
- Security Reviewer
- GKE Admin
- Logging Viewer
- Monitoring Viewer

Use **gcloud auth login**, **kubectl** uses **gcloud/OAuth tokens**.

2.3 Zero Trust Access with IAP

For:

- kubectl access (IAP TCP tunnel)
- Internal dashboards

- Legacy apps migrated to GCP

IAP does:

- Identity verification
- Context-aware access
- MFA
- No VPN needed

Exam Hint:

If you see "no VPN", "remote workers", or "BeyondCorp", answer = **IAP**.

★ 3. DOMAIN 2: SECURE COMMUNICATIONS + BOUNDARY PROTECTION

3.1 External Security

✓ External HTTPS LB (Global)

TLS termination
Google Front End (GFE)
Protected edges

✓ Cloud Armor (WAF)

- OWASP rules
- Rate limits
- Geo-blocking
- L7 filtering

✓ Cloud CDN

Reduce load + latency
Mitigate traffic spikes

3.2 Internal Boundary Protection

✓ Private GKE Cluster

Nodes have **no public IPs**

Control plane accessible only via:

- VPN
- Interconnect
- Bastion + IAP
- Authorized Networks

✓ VPC Firewall Rules

Default architecture:

- Deny all ingress
- Allow internal East-West
- Restrict egress except required destinations

✓ NAT Gateway (How Pods Reach Internet)

IMPORTANT:

Pods **do not** get public IPs.

Pods reach the internet via:

Pod → Node → Cloud Router → Cloud NAT → Internet

Benefits:

- Outbound-only
- No inbound from internet

- Hides private node IPs
 - Centralized logging + control
-

3.3 Anthos Service Mesh → Zero Trust Inside Cluster

Mesh gives you:

- Mutual TLS between all services
- Service-level authorization
- Fine-grained policies

Example policy:

```
Frontend can talk → Backend  
Backend can talk → DB  
No other service can talk → DB
```

This is a major exam focus.

★ 4. DOMAIN 3: ENSURE DATA PROTECTION

4.1 Database Security

Use:

- Cloud SQL or AlloyDB with **Private IP**
- Disable public DB access
- Only allow GKE node subnet
- Workload Identity for DB auth (IAM DB Auth)

- CMEK (Customer-managed encryption keys)

Exam Hint:

"If organization requires BYOK / HSM" → use **Cloud HSM + CMEK**.

4.2 Secrets Protection

Wrong:

- Kubernetes Secrets (base64 only)
- Storing creds in ConfigMaps
- Storing keys in images

Correct:

- Secret Manager + GKE CSI driver
 - Rotation policies
 - IAM permissions
 - CMEK
-

4.3 Encryption

- GCP encrypts all data at rest AES-256 by default
 - For compliance → enable CMEK or CSEK
 - For sovereignty → use **Sovereign Controls** (if exam mentions EU regions / locality)
-

★ 5. DOMAIN 4: MANAGE OPERATIONS (MONITORING + THREAT DETECTION)

5.1 Logging

Enable:

- GKE Audit Logs
- Data Access logs (for sensitive datasets)
- Firewall logs
- NAT logs
- LB logs
- VPC Flow Logs
- Cloud SQL logs
- Cloud Armor logs

Security Workflows:

- Log sinks → BigQuery → SIEM (Chronicle / Splunk)
 - Trigger Cloud Functions / Cloud Run for automation
-

5.2 Threat Detection

Enable:

- Security Command Center (SCC) Premium
- Container Threat Detection (CTD)
- Event Threat Detection
- VM Threat Detection

- Web Security Scanner

Exam often asks:

“What product identifies risky container images?”

Answer: **Container Threat Detection or Container Analysis (scanner)**.

★ 6. DOMAIN 5: SUPPORT COMPLIANCE (POLICIES + ORG CONTROLS)

6.1 Organization Policies

Use Constraints to enforce:

- No public IPs
- Shielded VMs only
- CMEK required
- Restricted services
- Domain restricted sharing
- Allowed regions
- Allowed IAM roles

Exam loves questions like:

“Prevent accidental exposure of Cloud Storage buckets”—

Answer: **VPC-SC or Org Policies or Both.**

6.2 VPC Service Controls

Used ONLY for:

- Preventing data exfiltration from Google-managed services

- Adding a perimeter around:
 - BigQuery
 - Cloud Storage
 - Pub/Sub
 - Secret Manager
 - AI APIs

Not for:

- DDoS
- Identity
- GKE networking
- Ingress protection

Exam trick:

If question ends with “**prevent exfiltration**” → VPC-SC.

6.3 Supply Chain Security

Use:

- Artifact Registry (no DockerHub pulling)
- Binary Authorization
- SBOM (Artifact Analysis)
- Signed images (SLSA)
- CI/CD policies

Exam Hint:

"If an org wants only approved images deployed to GKE" → **Binary Authorization**.

⭐ FULL 3-TIER EXAMPLE SECURITY FLOW

Here is the secure flow in 3 tiers:

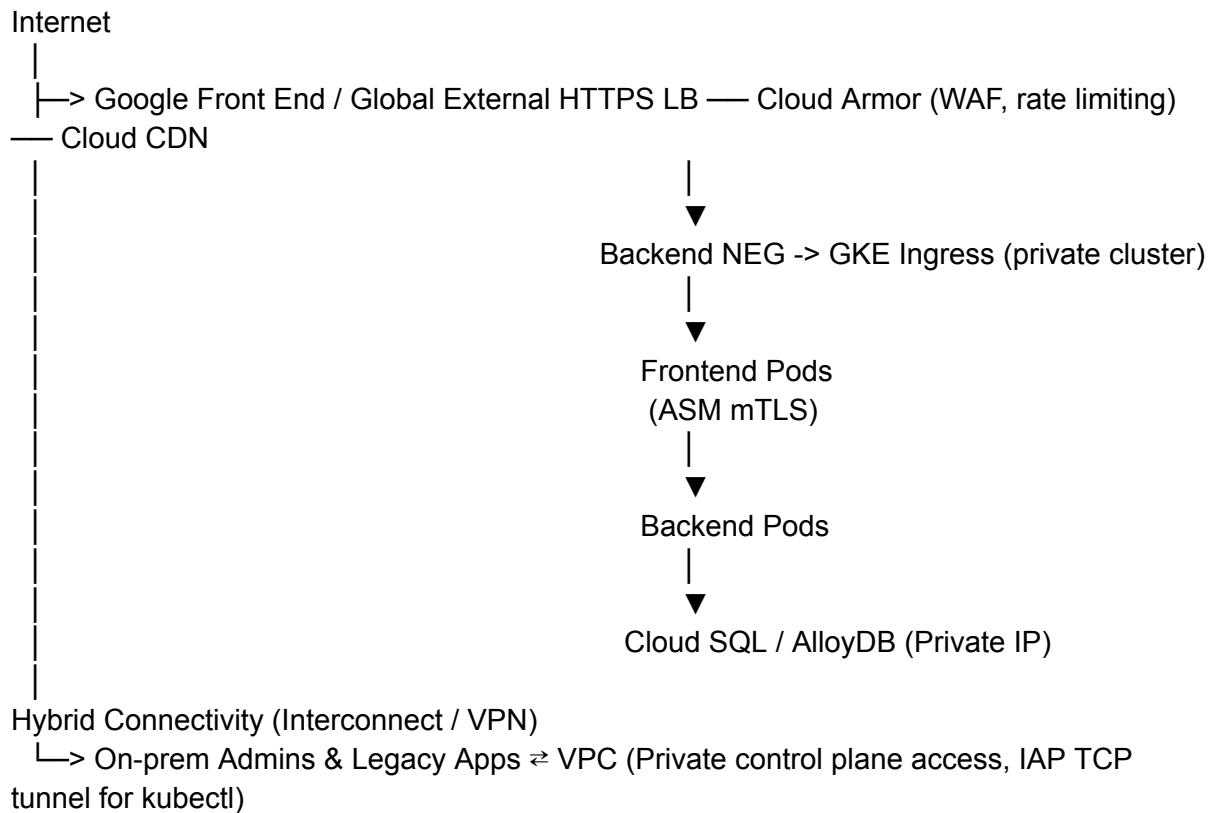
```
Internet
| (DDoS filtered)
V
Cloud Armor → HTTPS LB → Ingress → Frontend Pods (mTLS) → Backend
Pods (mTLS) → DB Private IP
```

Behind the scenes security:

```
Pods no public IP
↓
Outbound-only via Cloud NAT
↓
Private Cluster, RBAC, IAM, Workload Identity
↓
mTLS between services (ASM)
↓
Data encrypted CMEK
↓
CI/CD signed images (Binary Auth)
↓
Compliance (Org Policies + VPC-SC)
```

On-prem workflow:

```
On-Prem User → IAP → OAuth2 Login → RBAC Authorization → GKE API
```



Key datapaths:

- Edge filtering (Cloud Armor) blocks DDoS and OWASP attacks.
- Ingress -> NEG targets GKE pods via NodePort/Service or serverless NEG.
- Egress: Pods → Nodes → Cloud NAT → Internet (outbound only)
- On-prem access: either via Interconnect/VPN to private control plane or via IAP TCP tunnel (zero-trust).

GKE Final

Layer 1: The Front Door (User Access to GKE)

How do human admins and developers securely access the cluster dashboard and API?

Feature	Function	Exam Key Concept
Cloud Console (GKE UI)	The web interface for viewing workloads.	Requires Kubernetes RBAC permissions to view resources, not just IAM permissions.
Connect Gateway	Allows users to connect to private clusters via the Google Cloud Console without a VPN.	" Identity-Aware Proxy for Clusters. " It proxies your request through Google's infrastructure securely.
IAP + Bastion	"Break-glass" command line access (kubectl).	Use for Private Clusters where Connect Gateway isn't used. Zero Trust Network Access.
Cloud IAM	"Can I log in?" (Authentication).	Grants broad project-level roles (e.g., Viewer , Container Admin).
Kubernetes RBAC	"Can I delete this pod?" (Authorization).	Granular, namespace-level permissions (RoleBinding). Least Privilege.



Monitoring at this Layer:

- **Cloud Audit Logs (Admin Activity):** "Who created this cluster?" "Who modified the firewall?"
- **Cloud Audit Logs (Data Access):** "Who ran [kubectl exec](#) to get inside a container?" (Must be enabled explicitly).

Layer 2: The Network Perimeter (VPC)

How do we stop unauthorized network traffic?

Feature	Function	Exam Key Concept
Private Cluster	Nodes and Control Plane have private IPs.	The Foundation. No public internet access by default.
Master Authorized Networks	Whitelist specific IPs that can talk to the Control Plane.	The Doorman. Blocks all other IPs from running <code>kubectl</code> commands.
VPC Service Controls	Prevents data from moving across project boundaries.	Data Exfiltration Defense. Stops copying data to personal projects.
Network Policies	Controls Pod-to-Pod traffic inside the cluster.	"Internal Firewalls." By default, all pods can talk. Use this to block DB access from Frontend.



Monitoring at this Layer:

- **VPC Flow Logs:** "Is traffic trying to hit port 22 from China?" (Packet headers only).
 - **Firewall Rule Logging:** "Which specific rule blocked the traffic?"
-

Layer 2.5: The Service Mesh (East-West Security)

Replaces/Upgrades standard Kubernetes Network Policies.

Feature	Function	Exam Key Concept
mTLS (Mutual TLS)	Encrypts traffic <i>between</i> pods automatically.	"Zero Trust in Transit." Even if a hacker is on the node sniffing traffic, they see only encrypted garbage.
AuthorizationPolicy	Strict rules on <i>who</i> can call <i>whom</i> .	"Fine-Grained Access." " <i>Frontend can call Backend GET /items, but CANNOT call POST /admin.</i> " (Much stronger than Layer 2 Network Policies).
Ingress Gateway	The unified entry point for external traffic.	"The Edge." Handles SSL termination and traffic routing before it enters the mesh.

Layer 3: The Supply Chain (Code to Container)

How do we ensure only trusted code runs?

Feature	Function	Exam Key Concept
Artifact Registry	Stores container images securely.	Uses Private Google Access for internal pulling (No NAT needed).
Artifact Analysis	Scans images for vulnerabilities (CVEs).	" Shift Left. " Catch bugs before deploy.
Binary Authorization	Policy engine that blocks deployment.	" The Signature Check. " Only images signed by the "Attestor" (CI/CD) are allowed to boot.

Layer 4: The Workload (Pod Identity & Secrets)

How do we secure the running application?

Feature	Function	Exam Key Concept
Workload Identity	Maps KSA to GSA.	" No Keys. " Pods authenticate to GCP APIs via short-lived tokens.
GKE Sandbox	Isolates the container kernel (gVisor).	" Untrusted Code. " Use this if you are running multi-tenant code (e.g., SaaS).
Secret Manager + CSI	Mounts secrets as files in RAM.	" Vault. " Replaces insecure Kubernetes Secrets (etcd).
Sidecars	Helper containers (Proxy, Logs).	Cloud SQL Auth Proxy lives here.

Layer 5: Observability & Threat Detection (The "Eyes")

How do we know if something is wrong?

This is where your **Dashboard** question fits in.

A. The Operational View: GKE Security Posture Dashboard

- **Who uses it?** The DevOps / Platform Engineer.
- **Where is it?** Inside the **GKE Console** tab.
- **What it shows:**
 - "Which pods have critical vulnerabilities?"
 - "Which workloads are missing Network Policies?"
 - "Which images are running with **privileged** flags?"
- **Exam Value:** It gives you a **workload-centric** view of misconfigurations without needing to leave the Kubernetes context.

B. The Security HQ: Security Command Center (SCC) Premium

- **Who uses it?** The CISO / Security Team.
- **Where is it?** The global Security dashboard.
- **What it shows:**
 - **Container Threat Detection (CTD):** "A Reverse Shell was just opened in Cluster A." (Runtime Threat).
 - **Event Threat Detection (ETD):** "A Service Account is downloading 1TB of data." (API Threat).
- **Exam Value:** It aggregates threats from GKE, Compute, Storage, and IAM into one "Single Pane of Glass."

Layer 5 Upgrade: Deep Observability (Layer 7)

Upgrades VPC Flow Logs.

ASM adds a massive new data source to your "Eyes."

Feature	Function	Exam Key Concept
Service Telemetry	Logs every HTTP request (Code 200, 404, 500).	"Golden Signals." Latency, Traffic, Errors, Saturation.

Distributed Tracing	Tracks a request ID across microservices.	"The Needle in the Haystack." Find exactly which microservice caused the 5-second delay.
Access Logging	Logs exactly <i>who</i> called <i>whom</i> .	"Audit Trail." "Service A tried to call Service B but was rejected by policy."

How it changes your Roadmap:

- VPC Flow Logs (Layer 2): "Packet went from IP A to IP B." (Dumb).
- ASM Logs (Layer 5): "Frontend (User=Alice) called Checkout (POST /buy) and got a 500 Error." (Smart).

Summary of Logs (End-to-End)

If a hacker tries to attack your GKE cluster, here is where you see them at each step:

1. **Hacker tries to login:** -> **Cloud IAM Audit Logs**.
2. **Hacker tries to hit the API:** -> **K8s Audit Logs**.
3. **Hacker scans the network:** -> **VPC Flow Logs**.

The Scenario: Imagine a hacker has compromised one of your non-critical containers (e.g., a "Blog" pod). Now they want to find your valuable "Database." Since they don't know the IP address of the database, they start "knocking on every door." This is a **Network Scan**.

What the Hacker does: They run a script (like `nmap`) that tries to connect to:

`10.0.0.1:5432` (Is the DB here?)

`10.0.0.2:5432` (Is it here?)

`10.0.0.3:5432` (Is it here?)

4. **Hacker exploits a vuln in the app:** -> **Application Logs** (Cloud Logging).

If a hacker finds a bug *inside* your Python or Java code (like the infamous Log4j vulnerability or a SQL Injection), the network firewall sees it as "valid traffic" because it's allowed on port 443. The only place this attack leaves a footprint is in the **Application Logs**.

5. **Hacker runs malware:** -> **Container Threat Detection (SCC)**.
6. **Hacker steals data:** -> **Data Access Logs** (Storage).

Example:

The Setup:

You have a microservices-based E-commerce app running on a Private GKE Cluster with Anthos Service Mesh (ASM).

- **Frontend Service:** Public facing.
- **Catalog Service:** Reads product data.
- **Payment Service:** Handles credit cards (High Security).
- **Email Service:** Sends confirmations.
- **Cloud SQL:** Stores orders (Private IP).

Visualizing the Architecture

The Step-by-Step Security Flow

Step 1: I open the ecommerce website

- **The Action:** You type `www.shop.com` in your browser.
- **Layer 7 Protection (Cloud Armor):** Before the request hits your cluster, **Cloud Armor** inspects the traffic.

- *Check:* "Is this user coming from a blacklisted IP?" "Is this a DDoS attack?" "Is there a SQL Injection in the URL?"
- *Verdict:* Safe.
- **Ingress:** The traffic hits the **Global Load Balancer**, which routes it to the **ASM Ingress Gateway**.
- **TLS Termination:** The Gateway decrypts the public HTTPS and re-encrypts it for the internal mesh.

Step 2: I view a product

- **The Action:** You click "Blue Widget".
- **The Mesh (ASM):** The **Frontend Service** calls the **Catalog Service**.
- **Zero Trust (mTLS):**
 - Frontend Envoy Proxy says: "I am the Frontend. Here is my ID certificate."
 - Catalog Envoy Proxy says: "I verify you are the Frontend. I am the Catalog. Encryption tunnel established."
- **Authorization:** ASM checks the **AuthorizationPolicy**: "Is **Frontend** allowed to **GET /products**?" -> **YES**.

Step 3 & 4: Selected product & Added to Cart

- **The Action:** You click "Add to Cart".
- **Identity (Workload Identity):** The **Cart Service** needs to save this temporary data to a **Redis** cache or a **Firebase** database.
 - The Cart Pod does *not* use a password.
 - It swaps its **Kubernetes Service Account** for a **Google Access Token** (Workload Identity).
 - It writes to Firebase using that token.

Step 5 & 6: Proceed with Payment (Giving Credit Card)

- **The Action:** You enter your Credit Card Number (CCN) and click "Pay". (**CRITICAL STEP**)
- **DLP (Sensitive Data Protection):**
 - Before the application logs anything, the application calls the **DLP API** (or uses a sidecar filter).
 - *Action:* Detects the CCN.
 - *Result:* If the app tries to write a log, it writes **[CREDIT_CARD_REDACTED]** instead of the actual number.
- **Secrets (Secret Manager):**
 - The **Payment Service** needs to talk to the external bank (e.g., Stripe).
 - It needs the **Stripe API Key**.
 - The **Secrets Store CSI Driver** mounts the API Key from **Secret Manager** into the Payment Pod's RAM (**tmpfs**). The Pod reads it securely.

Step 7: Payment is done (Writing to Database)

- **The Action:** The order is confirmed. We must save it to **Cloud SQL**.
- **Connectivity (Sidecar Pattern):**
 - The Payment Pod talks to **localhost:5432**.
 - The **Cloud SQL Auth Proxy (Sidecar)** intercepts the traffic.

- It tunnels through **Private Service Access** (VPC Peering) to the **Cloud SQL Private IP**.
- Data is written to the DB encrypted.

Step 8: Received an email (Egress)

- **The Action:** The system sends a "Thank You" email.
- **The Egress (Cloud NAT):**
 - The **Email Service** needs to call an external API like SendGrid or Mailgun.
 - The Pod has no Public IP.
 - Traffic goes: Pod -> Node -> **Cloud NAT** -> Internet -> SendGrid.
- *Alternative (Strict Security):* If you used an **ASM Egress Gateway**, the mesh would verify "Is the Email Service allowed to talk to api.sendgrid.com?" before letting it out.

Monitoring: How do you see all this?

While this is happening, your **Observability Stack** is recording everything:

1. **"Why is the site slow?"**
 - **Cloud Trace:** You see a "Waterfall" graph. "Frontend took 10ms, but Catalog took 2000ms." You know exactly where the lag is.
2. **"Did the payment work?"**
 - **ASM Dashboard:** You see a green line connecting **Frontend** -> **Payment**. You see "HTTP 200 OK".
3. **"Is a hacker watching?"**
 - **Security Command Center:**
 - **CTD:** Checks if a hacker spawned a shell inside the Payment Pod.
 - **VPC Flow Logs:** Checks if any data was sent to a suspicious IP in North Korea.

Summary of the "Packet Walk"

Step	User Action	Primary Security Control
1	Open Site	Cloud Armor (WAF) & Ingress Gateway
2	View Product	ASM mTLS (Encryption in Transit)
3	Add to Cart	Workload Identity (Authentication)
4	Enter CC	DLP (Redaction) & Secret Manager (API Keys)

5	Save Order	Cloud SQL Auth Proxy (Sidecar)
6	Send Email	Cloud NAT (Secure Internet Egress)

The Problem with IPs (Layer 3): In Kubernetes, Pod IPs change all the time.

- Today, the "Frontend" is at IP **10.2.0.5**.
- Tomorrow, the "Frontend" restarts and gets IP **10.2.0.99**.
- **Security Problem:** You cannot easily write a firewall rule saying "Only allow 10.2.0.5 to talk to Payment," because that IP might belong to a hacker tomorrow.

The Solution: SPIFFE ID (The "Passport") Anthos Service Mesh (ASM) assigns a cryptographic identity card to every pod. This is called a **SPIFFE ID**. It looks like a URL: `spiffe://my-cluster.local/ns/default/sa/frontend`

How ASM Identity Blocking works: When "Frontend" tries to talk to "Payment":

1. **The Handshake (mTLS):**
 - **Frontend** says: "Hello Payment. Here is my ID card: `.../sa/frontend`."
 - **Payment** checks the signature: "This ID is valid and signed by our Mesh Authority."
2. **The Policy Check (AuthorizationPolicy):**
 - **Payment Envoy Proxy** looks at its security rule:
 - **Rule:** "Allow access ONLY IF `source.principal == .../sa/frontend`."
 - **Decision:** MATCH. Access Granted.

Why is this "Zero Trust"? Imagine a Hacker creates a rogue pod named "Evil-Pod" at IP **10.2.0.5** (pretending to be the old Frontend IP).

- **Firewall (Layer 3):** Might allow it because the IP looks correct. (**FAIL**)
- **ASM (Layer 2.5):**
 - **Evil-Pod** says: "Hello Payment."
 - **Payment** says: "Show me your ID."
 - **Evil-Pod** shows ID: `.../sa/evil-service-account` (or has no ID).
 - **Payment** checks rule: "Does `.../sa/evil-service-account` match `.../sa/frontend`?"
 - **Decision:** NO MATCH. **Access Denied.**

GKE has TWO audit log systems:

1. Cloud Audit Logs (GCP API calls): - cluster create/delete - node pool modifications - IAM changes
2. Kubernetes Audit Logs (K8s API calls): - kubectl exec (command execution in pod) - kubectl delete pod - Secret access

API Gateway

Layer 1: The Front Door (Network Edge)

How do we stop attacks before they even reach the API logic?

Feature	Function	Exam Key Concept
Cloud Load Balancer	Global Entry Point (Anycast IP).	" Single IP. " Terminates TLS (HTTPS) here so your backend doesn't have to handle crypto.
Cloud Armor	Web Application Firewall (WAF).	" The Bodyguard. " Blocks SQL Injection, XSS, and DDoS attacks at the edge.
Apigee Advanced API Security	AI-driven bot detection.	" Bot Defense. " Detects "scraper" bots stealing your data. (Apigee only).

- **Exam Tip:** If the question asks about "DDoS protection for APIs" -> **Cloud Armor**.

Layer 2: Identity & Access (The "Bouncer")

Who is calling? Are they allowed in?

Feature	Function	Exam Key Concept
API Keys	Simple alphanumeric string.	" Identification. " Identifies the <i>App</i> (e.g., "Mobile App v1"), not the <i>User</i> . Good for quotas, bad for security (easily stolen).
OAuth 2.0 / OIDC	Token-based access.	" Authentication. " Identifies the <i>Human User</i> . Returns a JWT (JSON Web Token).
Service Accounts	Machine-to-Machine identity.	" Backend Auth. " The Gateway uses a Service Account to talk to the backend (GKE/Cloud Run).

- **Exam Tip:**

- **Question:** "How do we identify which mobile app is calling us?" -> **API Key**.
- **Question:** "How do we ensure a user is an employee?" -> **OAuth / OIDC**.

Layer 3: Traffic Control (The "Traffic Cop")

How do we stop one user from crashing the system?

Feature	Function	Exam Key Concept
Rate Limiting	"You can call 100 times per minute."	"Spike Arrest." Protects against "Noisy Neighbors" or brute-force attacks.
Quotas	"You can call 10,000 times per month."	"Business Logic." Used for monetization (Bronze/Silver/Gold tiers).
Caching	Stores responses at the edge.	"Latency Reduction." Returns data from the Gateway memory instead of hitting the backend database.

-
- **Exam Tip:** Apigee has "Spike Arrest" (smooths traffic) and "Quota" (hard limit). API Gateway has basic Rate Limiting

Layer 4: Backend Security (The "Tunnel")

How does the Gateway talk to the Backend safely?

This is the most critical architecture point. Your API Gateway is public, but your GKE Cluster is private. How do they talk?

Feature	Function	Exam Key Concept
Serverless VPC Access	Bridge for API Gateway/Cloud Functions.	"The Connector." Allows a serverless Gateway to reach a private IP in your VPC.
Private Service Connect (PSC)	Private endpoint for Apigee.	"The Enterprise Link." Securely exposes services to Apigee without VPC Peering.
Mutual TLS (mTLS)	Encrypted handshake between Gateway and Backend.	"Zero Trust." Ensures the backend <i>only</i> accepts requests from the Gateway (not from a random VM).

Layer 5: Observability & Monetization (The "Eyes")

How do we know who is buying what?

Feature	Function	Exam Key Concept
Apigee Analytics	Deep business insights.	"Monetization." "Developer X used the API 500 times, charge them \$5." "Latency is high in Europe."
Cloud Trace	Distributed request tracking.	"Debugging." Follow the request ID from Mobile App -> Gateway -> GKE -> Database.
Cloud Logging	Raw logs.	"Audit." "Who deleted this API key?"

Exam Showdown: API Gateway vs. Apigee

The exam will give you a scenario, and you must pick the right tool.

Feature	API Gateway (The "Lite" Version)	Apigee (The "Enterprise" Version)
Target User	Developers, Serverless users.	Enterprise Architects, Product Managers.
Best Use Case	Simple front-end for Cloud Functions/Cloud Run.	Managing APIs as a Product, Monetization, Hybrid Cloud (On-prem + Cloud).
Features	Auth + Rate Limits.	Analytics, Monetization, Developer Portal, SOAP-to-REST conversion, Advanced Threat Protection.
Cost	Pay-per-call (Cheap).	Subscription / Pay-as-you-go (Expensive).
Exam Keyword	"Serverless", "Simple", "GCP Native".	"Monetization", "Analytics", "Hybrid", "Developer Portal".

The Packet Walk: "The Secure API Call"

Let's trace a user buying a product via your Mobile App.

1. **User Action:** User taps "Buy" on iPhone.
2. **Layer 1 (Edge):** Traffic hits **Cloud Load Balancer**. **Cloud Armor** checks for SQL Injection. (Pass)
3. **Layer 2 (The Gateway):** Request hits **Apigee**.
 - **Check:** "Is the API Key valid?" (Pass)
 - **Check:** "Is the OAuth Token valid?" (Pass - User is Alice).
4. **Layer 3 (Traffic):**
 - **Check:** "Has Alice exceeded her 100 calls/minute limit?" (Pass).
5. **Layer 4 (The Bridge):**
 - Apigee transforms the request (JSON -> gRPC).
 - Apigee calls the Backend (GKE) via **Private Service Connect**.
 - Apigee adds a **Service Account Token** header so GKE knows it's trusted.
6. **Backend Processing:** GKE processes the order.

7. **Layer 5 (The Log):** Apigee records: "Alice bought Item #5. Latency: 50ms. Charge: \$0.10."

Mock Exam Question: API Security

Scenario: You are a large bank migrating a legacy SOAP XML backend to Google Cloud. You want to expose this backend to modern mobile apps as a REST API (JSON). You require strict **XML threat protection, traffic quotas** for partners, and a **developer portal** for partners to sign up.

Which solution should you choose?

- A. Deploy API Gateway in front of the legacy service.
- B. Deploy Cloud Endpoints on a GKE sidecar.
- C. Use Apigee X. Configure a "SOAP to REST" policy and enable the Developer Portal.
- D. Use a Global Load Balancer with Cloud Armor to transform the XML to JSON.

GCE

GCE is the "Raw Infrastructure." Unlike GKE, Google manages less of it for you, so **you** are responsible for more security layers (like patching the OS).

Here is your **5-Layer Roadmap for Compute Engine Security**.

Layer 1: The Network Perimeter (The Moat)

How do we control traffic in and out of the VM?

Feature	Function	Exam Key Concept
VPC Firewall Rules	Filter traffic by IP/Port/Tag.	" The Basics. " Stateful blocking. Use Service Accounts as targets instead of IP addresses for dynamic rules.
Identity-Aware Proxy (IAP)	SSH without public IPs.	" Zero Trust SSH. " Stops you from opening Port 22 to the world.
Cloud Armor	WAF (If using a Load Balancer).	" Edge Defense. " Protects the VM from DDoS if it's sitting behind an HTTP Load Balancer.
Private Google Access	VM accesses Google APIs internally.	" No Public IP Egress. " Private VMs can reach Cloud Storage/BigQuery without a NAT Gateway.



Exam Tip:

- **Wrong Answer:** "Open Port 22 to 0.0.0.0/0 for admins."
 - **Right Answer:** "Configure IAP TCP Forwarding."
-

Layer 2: Identity & Access (The Keys)

Who can log in to the OS?

Feature	Function	Exam Key Concept
OS Login	Maps IAM Users to Linux Users.	"No More SSH Keys." Manages SSH access via Google Identity. Revocation is instant.
Service Accounts	The VM's Identity.	"Machine Identity." The VM uses this account to call Google APIs. Constraint: Use distinct Service Accounts per VM (Least Privilege).
IAM Conditionals	"Time-bound access."	"JIT Access." Grant admin access only from 9 AM to 5 PM.

Layer 3: The Hardware & Host (The Vault)

Is the machine itself trustworthy?

Feature	Function	Exam Key Concept
Shielded VMs	Boot integrity.	"Rootkit Defense." Secure Boot + vTPM + Integrity Monitoring.
Confidential VMs	Memory Encryption.	"Data-in-Use." Encrypts RAM so Google admins/Hypervisors cannot peek.
Sole Tenant Nodes	Dedicated physical server.	"Compliance." Physical isolation from other customers (PCI/HIPAA).

Layer 4: Operations & Maintenance (The Health)

How do we keep the OS patched and secure?

Feature	Function	Exam Key Concept
VM Manager (OS Config)	Automated Patch Management.	" Patch Tuesday. " Automatically applies Windows/Linux updates across a fleet of 1000 VMs.
Guest Policies	Software installation compliance.	" Drift Detection. " "Ensure the Cloud Logging Agent is installed on ALL VMs."
Golden Images	Pre-hardened Boot Disks.	" Immutable Infra. " Don't patch live servers; bake a new secure image and replace the VM.

Layer 5: Observability (The Camera)

What is happening on the server?

Feature	Function	Exam Key Concept
Cloud Audit Logs	Admin Activity.	" Who stopped the VM? " Tracks API calls (Stop, Start, Delete).
Auditd / Syslog	OS-level logs.	" Who ran sudo rm -rf? " You must install the Ops Agent to send these logs to Cloud Logging.
VPC Flow Logs	Network metadata.	" Traffic Analysis. " "Why is this VM sending 1GB of data to Russia?"

The Packet Walk: "The Secure Admin & The Web App"

Let's trace two flows: **(A) An Admin fixing a bug** and **(B) A User accessing the App.**

Scenario: A Private VM (No Public IP) running a Web App. It needs to read data from a Storage Bucket.

Flow A: The Admin (SSH Access)

1. **Admin Action:** Runs `gcloud compute ssh my-vm`.

2. **Layer 2 (Auth):** Google checks IAM. "Does user have `roles/compute.osLogin?`" -> **YES**.
3. **Layer 1 (Network):**
 - o `gcloud` starts an **IAP Tunnel**.
 - o Firewall Rule checks: "Allow Ingress from `35.235.240.0/20` (IAP Range)?" -> **YES**.
4. **Layer 3 (Host):**
 - o **Shielded VM** checks: "Is the Bootloader valid?" -> **YES**.
 - o **OS Login** creates a temporary Linux user for the Admin.
5. **Result:** Admin gets a shell.

Flow B: The Web App (User Request)

1. **User Action:** Hits the Load Balancer IP.
2. **Layer 1 (Edge):** **Cloud Armor** scans for XSS attacks. -> **Pass**.
3. **Layer 1 (VPC):** Load Balancer forwards traffic to the VM's private IP.
 - o Firewall Rule: "Allow Ingress from LB Subnets?" -> **YES**.
4. **Layer 3 (Processing):**
 - o The App processes the data in RAM.
 - o **Confidential VM** ensures RAM is encrypted (Hardware Key).
5. **Layer 2 (Backend Access):**
 - o The App needs to read an image from Cloud Storage.
 - o VM Metadata Server provides a **Service Account Token**.
 - o App calls `storage.objects.get`.
6. **Layer 1 (Egress):**
 - o Traffic hits **Private Google Access** (because VM has no Public IP).
 - o Routes internally to `storage.googleapis.com`.
7. **Result:** Data returned to user.

IAM

1. Core IAM Principles (Memorize These!)

The IAM Formula

"Who can do What on Which resource"

- **Who** = Principal (user, group, SA, domain)
- **What** = Role (collection of permissions)
- **Which** = Resource (project, bucket, VM, etc.)

Critical Rules

1. **Policies are ADDITIVE ONLY** - Can never remove parent permissions
2. **Most permissive wins** - Viewer at folder + Editor at project = Editor
3. **Deny doesn't exist in IAM** - Only in Organization Policies
4. **Effective permissions = Union of all ancestor policies**

Policy Evaluation Order

1. Org Policy DENY? → Access Denied (stops here)
 2. VPC-SC BLOCKS? → Access Denied (stops here)
 3. IAM policies from org→folder→project→resource
 4. Union all ALLOWS
 5. Required permission granted? → Access Granted : Denied
-

2. Identity Types Quick Reference

Type	Format	Use Case	Key Point
User	user:alice@example.com	Humans	Never grant individually! Use groups
Service Account	serviceAccount:sa@project.iam	Applications	One per app/env
Group	group:team@example.com	Collections of users	PRIMARY way to manage access
Domain	domain@example.com	All domain users	Use sparingly, very broad
allAuthenticatedUsers	Special identifier	Any Google account	Effectively public

allUsers	Special identifier	Anyone (anonymous)	Completely public
-----------------	--------------------	-----------------------	-------------------

Service Account Types

1. **User-managed:** `sa-name@project.iam` (you create)
 2. **Default:** `PROJECT_NUMBER-compute@developer.gserviceaccount.com` (auto-created, has Editor - DANGER!)
 3. **Google-managed:** `service-PROJECT_NUMBER@...` (Google services use)
-

3. Roles - The Hierarchy

Primitive Roles (AVOID IN PRODUCTION!)

- **Viewer:** Read-only everything
- **Editor:** Viewer + modify resources (not IAM/billing)
- **Owner:** Editor + IAM + billing + delete project

Exam Tip: If question says "production" or "compliance" → NEVER choose primitive roles

Role Levels

Service Level: `roles/compute.admin` (entire service)



Resource Level: `roles/compute.instanceAdmin.v1` (specific resource type)



Function Level: `roles/compute.osLogin` (specific function)

Most Tested Predefined Roles

Compute Engine:

- `compute.admin` - Full control
- `compute.instanceAdmin.v1` - Manage VMs (not networks)
- `compute.networkAdmin` - Manage networks (not VMs)
- `compute.osLogin` - SSH via OS Login
- `compute.viewer` - Read-only

Storage:

- `storage.admin` - Full control (buckets + objects + IAM)
- `storage.objectAdmin` - Objects only (not bucket creation/IAM)
- `storage.objectCreator` - Upload only (no read!)

- `storage.objectViewer` - Download/read only

BigQuery:

- `bigrquery.admin` - Full control
- `bigrquery.dataEditor` - Read/write data
- `bigrquery.dataViewer` - Read data only
- `bigrquery.user` - Run queries, create datasets
- `bigrquery.jobUser` - Run jobs only

IAM:

- `iam.serviceAccountUser` - Attach SA to resources, impersonate
- `iam.serviceAccountTokenCreator` - Generate tokens (impersonation)
- `iam.serviceAccountAdmin` - Manage SAs (not keys!)
- `iam.serviceAccountKeyAdmin` - Manage keys (dangerous!)
- `iam.workloadIdentityUser` - For GKE/external workload access
- `iam.securityAdmin` - Manage IAM policies (god-mode)

Cloud SQL:

- `cloudsql.admin` - Full control
- `cloudsql.client` - Connect to instances (most common for apps)

Cloud Run/Functions:

- `run.admin / cloudfunctions.admin` - Full control
 - `run.developer / cloudfunctions.developer` - Deploy/manage
 - `run.invoker / cloudfunctions.invoker` - Call/invoke (for apps)
-

4. Service Accounts - Critical Exam Topic

SA Authentication Methods

Method	Security	Use Case
Metadata Server (GCE/GKE/Functions)	✓ Best	Default for GCP resources
Workload Identity (GKE)	✓ Best	Pods accessing GCP
Workload Identity Federation	✓ Best	External (AWS/Azure/GitHub)
Service Account Keys	✗ Avoid	Legacy only, compliance risk

SA Key Facts (High Frequency Exam Topic!)

- **User-managed keys:** Valid up to 10 years, can be stolen, NO automatic rotation
- **Google-managed keys:** Auto-rotated every 2 weeks, cannot be downloaded
- **Exam answer:** Almost ALWAYS "avoid keys, use Workload Identity/Federation"

SA Impersonation

Required roles:

- iam.serviceAccountUser: Attach SA to resources (VM, Cloud Run)
- iam.serviceAccountTokenCreator: Generate tokens for SA

Exam Pattern:

- "Need to deploy as SA" → `serviceAccountUser`
- "Need to generate token for SA" → `serviceAccountTokenCreator`

SA Best Practices (Exam Favorites)

- ✓ One SA per application per environment ✓ Never use default SAs in production ✓
Separate dev/staging/prod SAs ✓ Disable unused SAs (don't delete immediately) ✓ Name
descriptively: `api-backend-prod@project.iam`
- ✗ One SA for everything ✗ Using default SA with Editor ✗ Sharing SAs across
environments ✗ Downloading keys when alternatives exist
-

5. IAM Conditions - Key Patterns

Syntax

```
condition = {  
    title = "Human readable"  
    description = "Details"  
    expression = "CEL expression"  
}
```

Common Exam Patterns

Time-based (Contractor):

```
request.time < timestamp('2025-12-31T23:59:59Z')
```

Business Hours:

```
request.time.getHours() >= 9 && request.time.getHours() < 17
```

Weekdays Only:

```
request.time.getDayOfWeek() > 0 && request.time.getDayOfWeek() < 6
```

IP Restriction:

```
origin.ip in ['203.0.113.0/24', '198.51.100.0/24']
```

Resource Pattern:

```
resource.name.startsWith('projects/_/buckets/prod-')
```

Tag-based:

```
resource.matchTag('123456/environment', 'production')
```

Combined:

```
request.time.getHours() >= 9 &&  
request.time.getHours() < 17 &&  
origin.ip in ['203.0.113.0/24']
```

Exam Tip: Policy version MUST be 3 for conditions!

6. Organization Policies vs IAM

Critical Difference

Aspect	IAM	Org Policies
Controls	WHO can access	WHAT can be configured
Example	Alice can create VMs	VMs cannot have external IPs
Inheritance	Additive (child adds more)	Restrictive (child cannot relax)
Deny	No deny	Can deny configurations

Most Tested Org Policies

Boolean (On/Off):

```
compute.requireOsLogin - Force OS Login  
compute.disableSerialPortAccess - Block serial port  
compute.skipDefaultNetworkCreation - No default VPC  
sql.restrictPublicIp - No public Cloud SQL IPs
```

storage.uniformBucketLevelAccess - Force uniform access
iam.disableServiceAccountKeyCreation - Block key creation ★

List (Allow/Deny specific values):

gcp.resourceLocations - Restrict regions ★★★

Example: "in:eu-locations" (EU only)

compute.vmExternalIpAccess - Control external IPs

compute.trustedImageProjects - Limit VM image sources

iam.allowedPolicyMemberDomains - Restrict which domains can be granted access ★

Exam Pattern:

- "Ensure resources ONLY in EU" → `gcp.resourceLocations` with `in:eu-locations`
 - "Prevent SA keys" → `iam.disableServiceAccountKeyCreation`
 - "Prevent external users" → `iam.allowedPolicyMemberDomains`
 - "Child overrides parent" → IMPOSSIBLE (org policies cannot be relaxed)
-

7. VPC Service Controls - High Value Topic

What It Does

Prevents data exfiltration even with valid credentials

Core Components

1. Access Levels (WHO can access)

Conditions:

- IP ranges
- Device policy (screen lock, corp owned, OS version)
- Geographic location
- Members

Combining: AND or OR

2. Service Perimeters (WHAT is protected)

Contains:

- Projects (resources to protect)
- Restricted Services (APIs to protect)
- Access Levels (who can access)

- Ingress/Egress Policies
- VPC Accessible Services

3. Ingress Policies (Data coming IN)

From: Source (access levels, identities)

To: Destination (resources, operations)

4. Egress Policies (Data going OUT)

From: Source identities inside perimeter

To: External resources/operations

5. VPC Accessible Services (API restrictions)

Limit which Google APIs can be called from INSIDE perimeter

Example: Only allow storage, bigquery, sqladmin

Block: compute (prevents creating external VMs)

Exam Scenarios

Scenario: Prevent prod data copy to dev

Solution: VPC-SC perimeter around prod

- No egress policies
- Ingress only from corporate network
- Restricted services: storage, bigquery, sqladmin

Result: Even with credentials, cannot copy data out

Scenario: External partner needs specific BigQuery dataset

Better: Authorized views (not egress policy)

- Create view in shared dataset (outside perimeter)
- View queries protected dataset
- Partner accesses view only

Scenario: On-prem needs GCP access

Solution:

- Access level with on-prem IP ranges
- Ingress policy allowing on-prem SA
- Specific operations only



8. Workload Identity Federation - Key Patterns

The Problem It Solves

External workloads need GCP access WITHOUT service account keys

Architecture

External Identity (AWS/Azure/GitHub/OIDC)



External Token



Google STS (Security Token Service)



GCP Service Account Token (1 hour)



Access GCP Resources

Setup Components

1. **Workload Identity Pool** (container for external identities)
2. **Provider** (AWS/Azure/OIDC)
3. **Attribute Mapping** (map external claims to GCP)
4. **Attribute Conditions** (additional filters)
5. **IAM Binding** (grant `workloadIdentityUser` role)

Platform-Specific Patterns

AWS:

Provider: AWS (account ID: 123456789012)

Attribute Mapping:

```
google.subject = assertion.arn  
attribute.aws_role = assertion.arn.extract('assumed-role/{role}/')
```

IAM Binding Principal:

```
principalSet://iam.googleapis.com/.../attribute.aws_role/MyRole
```

Use Case: Lambda writing to BigQuery

Azure:

Provider: OIDC

Issuer: https://sts.windows.net/TENANT_ID/

Attribute Mapping:

```
google.subject = assertion.sub  
attribute.tenant_id = assertion.tid
```

IAM Binding Principal:

```
principal://iam.googleapis.com/.../subject/MANAGED_IDENTITY_OID
```

Use Case: Azure VMs accessing GCS

GitHub Actions:

Provider: OIDC

Issuer: <https://token.actions.githubusercontent.com>

Attribute Mapping:

```
google.subject = assertion.sub  
attribute.repository = assertion.repository
```

Attribute Condition:

```
assertion.repository == 'myorg/production-repo'
```

IAM Binding Principal:

```
principalSet://iam.googleapis.com/.../attribute.repository/myorg/myrepo
```

Use Case: GitHub deploys to Cloud Run

Exam Tip: If question mentions AWS/Azure/GitHub + "no keys" → Workload Identity Federation

9. Workload Identity (GKE) - Distinct from Federation

GKE-Specific Pattern

Kubernetes Service Account (K8s SA)

↓ (binding)

Google Service Account (GSA)

↓

GCP Resources

Required Configuration

1. Annotate K8s SA:

```
iam.gke.io/gcp-service-account: GSA_EMAIL
```

2. IAM Binding on GSA:

```
Role: roles/iam.workloadIdentityUser
```

Member: serviceAccount:PROJECT.svc.id.goog[NAMESPACE/KSA]

3. Grant GSA resource permissions:

Role: roles/cloudsql.client (example)

Exam Pattern: "GKE pod needs Cloud SQL" → Workload Identity with binding

10. Resource Hierarchy - Inheritance Patterns

Structure

Organization (one per company)

 └ Folders (multiple levels)

 └ Folders (can nest)

 └ Projects

 └ Projects

 └ Resources

Inheritance Rules (MEMORIZE!)

1. Policies propagate DOWN - Org grant applies to all descendants
2. Policies are ADDITIVE - Child adds, never removes
3. Most permissive wins - Viewer at org + Editor at project = Editor
4. Cannot remove parent grant - If granted at org, child cannot revoke

Folder Strategy Patterns

Environment-Based:

Org

 └ Production (strict policies)

 └ Staging (moderate)

 └ Development (relaxed)

Business Unit-Based:

Org

 └ Finance

 └ Engineering

 └ Sales

Hybrid (Most Common):

Org

```

└── Finance
    ├── Finance-Prod
    └── Finance-Dev
└── Engineering
    ├── Engineering-Prod
    └── Engineering-Dev

```

When to Grant Where

Level	Grant To	Use Case
Organization	Security team, Org admins	Company-wide roles
Folder	Business unit teams	Department/environment access
Project	Application teams	Project-specific access
Resource	Exceptions only	Fine-grained overrides

Exam Tip: "Most efficient" or "scalable" → Grant at highest appropriate level (folder > project > resource)

11. Common Exam Scenarios - Decision Trees

Scenario: Access Revocation Speed

Question Pattern: "Must revoke access in <5 minutes"

Decision Tree:

Individual users?

- NO: Use Groups ✓
- YES: Can take hours to update 100s of bindings ✗

Service Account?

- Disable SA ✓ (immediate)
- Delete keys ✗ (only stops key-based, not metadata server)
- Remove IAM bindings ✗ (slow if many)

Scenario: Temporary Access

Question Pattern: "Contractor needs access for 3 months"

Decision Tree:

Need automatic expiration?

- YES: IAM Condition with timestamp ✓
- NO: Manual revocation ✗ (will forget)

Need resource restrictions too?

- YES: IAM Condition with resource.name ✓
- NO: Grant full role ✗ (over-privileged)

Scenario: External Workload Authentication

Question Pattern: "AWS/Azure/GitHub needs GCP access"

Decision Tree:

Can use Workload Identity Federation?

- YES: Use WIF ✓ (no keys)
- NO: Why not? (rare - challenge this)

Must use keys?

- Compliance allows?
- YES: Keys + strict rotation + Secret Manager !
- NO: Must use WIF ✓

Scenario: Data Isolation

Question Pattern: "Prevent data from leaving production"

Decision Tree:

Need defense against compromised credentials?

- YES: VPC Service Controls ✓
- NO: IAM only ✗ (insufficient)

Need to allow specific external access?

- YES: VPC-SC + Egress Policy ✓
- NO: VPC-SC + No Egress ✓

Scenario: Role Selection

Question Pattern: "Service needs to [action] [resource]"

Decision Tree:

1. Can use predefined role?

- Check role list

- Minimum permissions?
- YES: Use predefined ✓
- NO: Too broad → Custom role

2. Primitive role mentioned?
 - Production/Compliance context?
 - YES: Never use primitive ✗
 - NO: Still avoid, use predefined ✓

Scenario: Permissions Not Working

Question Pattern: "Permission denied but IAM looks correct"

Troubleshooting Order:

1. Check Org Policies (can block IAM)
 2. Check VPC-SC (can block even with IAM)
 3. Check IAM Conditions (might not be met)
 4. Check Policy Version (must be 3 for conditions)
 5. Wait 2 minutes (propagation delay)
 6. Check resource name/project ID (typo?)
 7. Check API enabled
-

12. Best Practices - Exam Favorites

Always Do ✓

1. **Use Groups, Not Individuals**
 - Exam keyword: "scalable", "efficient", "100s of users"
 - Answer: Groups
2. **Separate Service Accounts**
 - Exam keyword: "multiple applications", "cost attribution"
 - Answer: One SA per app/env
3. **Avoid Service Account Keys**
 - Exam keyword: "GKE", "external workload", "secure"
 - Answer: Workload Identity / WIF
4. **Grant at Highest Appropriate Level**
 - Exam keyword: "most efficient", "100 projects"
 - Answer: Org or Folder level
5. **Use IAM Conditions for Temporary**
 - Exam keyword: "contractor", "temporary", "expire"
 - Answer: IAM Condition with timestamp
6. **Org Policies for Configuration Control**
 - Exam keyword: "enforce", "prevent", "ensure"

- Answer: Organization Policy
- 7. VPC-SC for Data Protection**
- Exam keyword: "prevent exfiltration", "HIPAA", "sensitive"
 - Answer: VPC Service Controls
- 8. Custom Roles for Precise Permissions**
- Exam keyword: "can X but not Y", "specific actions only"
 - Answer: Custom Role

Never Do

- 1. Primitive Roles in Production**
 - If see "Editor/Owner" + "production" → Wrong answer
 - 2. Default SA with Editor**
 - If see "default service account" → Disable automatic grants
 - 3. Individual User Grants**
 - If see "grant to each developer" → Use groups instead
 - 4. Download SA Keys When Alternatives Exist**
 - If see "GKE" or "GCE" + "keys" → Wrong, use Workload Identity
 - 5. Single SA for Everything**
 - If see "one service account" + "multiple apps" → Wrong, separate SAs
 - 6. Manual Access Revocation**
 - If see "remember to revoke" → Wrong, use automation/conditions
-

13. Quick Reference - Role to Permission Mapping

Common Permission Patterns

"Need to connect to Cloud SQL" → `roles/cloudsql.client` 

"Need to SSH to VM" → `roles/compute.osLogin` or
`roles/compute.osAdminLogin` 

"Need to deploy Cloud Run" → `roles/run.developer` 

"Need to invoke private Cloud Run" → `roles/run.invoker` 

"Need to read logs" → `roles/logging.viewer` 

"Need to read Data Access logs" → `roles/logging.privateLogViewer` 

"Need to impersonate service account" →
`roles/iam.serviceAccountTokenCreator` 

"Need to attach SA to resources" → `roles/iam.serviceAccountUser` 

"Need to upload to GCS (not read)" → `roles/storage.objectCreator` ✓

"Need to use Shared VPC subnets" → `roles/compute.networkUser` on host project/subnet ✓

"Need to query BigQuery" → `roles/bigquery.jobUser` +
`roles/bigquery.dataViewer` ✓

14. Compliance Patterns - Common Requirements

HIPAA

- ✓ VPC Service Controls (data perimeter)
- ✓ Access Transparency + Approval (Google access)
- ✓ CMEK encryption
- ✓ Data Access audit logs enabled
- ✓ No external IPs (org policy)
- ✓ No SA keys (org policy)
- ✓ Access from managed devices only

PCI-DSS

- ✓ Least privilege IAM
- ✓ Unique user accounts (no sharing)
- ✓ Regular access reviews
- ✓ Comprehensive audit logging
- ✓ Time-bound access (conditions)
- ✓ Separation of duties

GDPR

- ✓ Data residency (`gcp.resourceLocations` = EU only)
 - ✓ Audit trail (all logs enabled)
 - ✓ Data access restrictions (IAM + conditions)
 - ✓ Right to be forgotten (lifecycle policies)
-

15. Exam Keyword Triggers

Keyword in Question	Likely Answer Direction
"Scalable", "100s of users"	Use Groups
"Temporary", "Contractor", "Expire"	IAM Conditions

"No keys", "Secure", "GKE"	Workload Identity
"AWS", "Azure", "GitHub", "External"	Workload Identity Federation
"Prevent exfiltration", "HIPAA", "Sensitive"	VPC Service Controls
"Enforce", "Prevent", "Ensure configuration"	Organization Policy
"EU only", "Data residency", "Region restrict"	Org Policy: gcp.resourceLocations
"Production", "Compliance" + "Editor/Owner"	WRONG - Use predefined/custom
"Most efficient", "100 projects"	Folder-level grant
"Must revoke in <5 min"	Groups or Disable SA
"Automatically expire"	IAM Conditions
"Multiple applications", "Cost attribution"	Separate SAs per app
"Default service account"	Disable automatic grants, create custom
"Break-glass", "Emergency"	Separate account, IP restricted, monitored
"Audit", "Compliance report"	Cloud Asset Inventory
"Google personnel access"	Access Transparency + Approval

16. Critical Exam Traps

Trap 1: "Just add Editor role"

Trap: Fastest solution is primitive role **Reality:** In production/compliance context, always **WRONG** **Right:** Use specific predefined or custom role

Trap 2: "Create service account key"

Trap: Question mentions GKE/GCE and keys **Reality:** Almost never right answer for GCP resources **Right:** Workload Identity or metadata server

Trap 3: "Override organization policy at project"

Trap: "Child project needs exception to org policy" **Reality:** Cannot relax org policies at lower levels **Right:** Separate folder with different policy, or exception process

Trap 4: "Grant permission on every project"

Trap: "Grant access across 50 projects" **Reality:** Doesn't scale, hard to manage **Right:** Grant at folder or org level

Trap 5: "VPC-SC bridge for everything"

Trap: "Two perimeters need to communicate" **Reality:** Bridges weaken security **Right:** Use ingress/egress policies instead, or authorized views

Trap 6: "Delete service account immediately"

Trap: "Unused SA found" **Reality:** Might break something, hard to recover **Right:** Disable first, monitor, then delete after confirmation

Trap 7: "One permission missing, grant admin"

Trap: "SA can't do X, grant broader role" **Reality:** Violates least privilege **Right:** Grant specific missing permission via custom role

17. Last-Minute Memory Joggers

Policy Versions

- **Version 1:** No conditions
- **Version 3:** Required for conditions
- **Must request Version 3** to see conditions

Service Account Identifiers

- **Email:** `sa-name@project.iam.gserviceaccount.com` (changes if recreated)
- **Unique ID:** Numeric (permanent, doesn't change)

IAM vs Org Policy vs VPC-SC

- **IAM:** WHO can access WHAT
- **Org Policy:** WHAT configurations allowed
- **VPC-SC:** WHERE data can go

Propagation Times

- **IAM changes:** 60-120 seconds
- **Org Policy changes:** Up to 15 minutes
- **VPC-SC changes:** Can take several minutes

Special Role Names

- `iam.serviceAccountUser` → Attach SA to resources
- `iam.serviceAccountTokenCreator` → Generate tokens
- `iam.workloadIdentityUser` → For GKE/WIF
- `cloudsql.client` → Connect to Cloud SQL

Shared VPC Roles

- **Host project:** Grant `compute.networkUser` to service project SAs
- **Service project:** Use subnets from host project

Access Level Combining

- **Within access level:** Conditions can be AND or OR
 - **Multiple access levels on perimeter:** Combined with OR
-

18. Mental Models for Exam

Decision Model: "How to grant access?"

1. Who needs access?

Individual → Put in Group → Grant to Group
Application → Service Account

2. What level?

1 project → Project level
Multiple projects, same team → Folder level
Company-wide → Org level

3. Which role?

Check predefined roles first
Too broad? → Custom role
Production context? → Never primitive

4. Time-bound?

Yes → Add IAM condition
No → Regular grant

5. Additional restrictions?

IP → IAM condition
Resource pattern → IAM condition
Data protection → Consider VPC-SC

Decision Model: "Authentication for application?"

1. Where is application?

GKE → Workload Identity

GCE/Cloud Functions/Cloud Run → Metadata server (automatic)

AWS → Workload Identity Federation (AWS)

Azure → Workload Identity Federation (Azure OIDC)

GitHub Actions → Workload Identity Federation (GitHub OIDC)

On-prem K8s → Workload Identity Federation (K8s OIDC)

Other → Workload Identity Federation (generic OIDC)

2. Absolutely no alternative?

→ Service account key (with strict controls)

→ BUT exam answer: Almost always WIF/Workload Identity

Decision Model: "Enforcement needed?"

1. WHO can access? → IAM

2. WHAT can be configured? → Org Policy

3. WHERE can data go? → VPC-SC

4. All three? → Use all three (defense in depth)

19. Exam Day Checklist

Before answering each question:

✓ Read carefully - What is PRIMARY requirement? ✓ Eliminate obviously wrong answers first
✓ Production/Compliance context? → Never primitive roles ✓ Scalability mentioned?
→ Group/Folder level grants ✓ Security mentioned? → No keys, use WIF/Workload Identity
✓ Multiple correct answers? → Pick MOST secure/scalable/efficient ✓ "Should" vs "Must"?
→ "Must" = enforce with org policy/VPC-SC ✓ Tricky wording? → Reread what's actually being asked

20. Final Power Tips

1. **Groups solve 90% of user management questions**
2. **Keys are almost never the right answer** (except very specific legacy scenarios)
3. **VPC-SC for data exfiltration, not just IAM**
4. **Org policies enforce, IAM permits**
5. **Folder structure = scalability**
6. **Conditions = temporary/restricted access**
7. **Most permissive wins in IAM**
8. **Cannot relax org policies at lower levels**

-
9. Disable SA for immediate revocation
 10. Cloud Asset Inventory for org-wide IAM analysis
-

Print this, review before exam, and you'll be ready! 

21. IAM Policy Binding - Advanced Concepts

Policy Etag (Critical for Exam!)

What is it: Concurrency control mechanism

Format: Base64 encoded string

Changes: Every time policy is modified

Exam Scenario:

Q: "Two admins modify IAM policy simultaneously. What happens?"

A: Second update fails with etag mismatch error

Must re-read policy and retry

Key Point: ALWAYS include etag when updating policies

Policy Size Limits (Often Tested!)

Maximum policy size: 64 KB

Maximum members per binding: 1,500

Maximum bindings: No hard limit, but 64KB total

Exam Trap:

Q: "Need to grant access to 5,000 users"

Wrong: Add 5,000 individual users

Right: Create groups (groups have no member limit)

Deleted Member Behavior

When user deleted:

- IAM binding remains as "deleted:user:..."
- Binding becomes ineffective immediately
- Shows in policy as deleted member
- Manual cleanup recommended

Exam Pattern:

Q: "Employee left, what happens to their permissions?"

A: Permissions remain in policy but ineffective

Best practice: Use groups (remove from group = instant revocation)

22. Service Account Impersonation Chain (High Frequency!)

Maximum Chain Length

Limit: Up to 10 service accounts in chain

Format: SA1 → SA2 → SA3 → ... → SA10

Exam Scenario:

Q: "Multi-approval deployment: Developer → Approver → Executor"

Setup:

1. dev-sa can impersonate approver-sa
2. approver-sa can impersonate executor-sa
3. executor-sa has deployment permissions

Each needs: roles/iam.serviceAccountTokenCreator on next SA

Short-Lived vs Long-Lived Tokens

Default Token Lifetime: 1 hour

Maximum with extension: 12 hours (requires special org policy)

Org Policy: iam.allowServiceAccountCredentialLifetimeExtension

Exam Keyword:

"Token expires too quickly" → Check token lifetime extension policy

23. IAM Recommender (Critical for Professional!)

What It Does

Analyzes:

- Permission usage patterns (last 90 days)
- Over-permissioned roles
- Unused permissions
- Lateral movement risks

Provides:

- Role recommendations (remove/replace)
- Service account recommendations
- Policy insights

Recommendation Types

1. REMOVE_ROLE: Role not used at all
2. REPLACE_ROLE: Use more restrictive role
3. REVOKE_PERMISSION: Specific permission unused

Exam Pattern:

Q: "Identify unused permissions across 100 projects"

A: Use IAM Recommender

NOT: Manual audit (doesn't scale)

NOT: Audit logs (too manual)

Policy Intelligence Suite

Components:

1. Policy Analyzer: "Who has access to X?"
2. Policy Troubleshooter: "Why can't user access Y?"
3. Activity Analyzer: Actual permission usage
4. IAM Recommender: Optimization suggestions

Exam Scenario:

Q: "User can't access bucket, but IAM looks correct"

A: Use Policy Troubleshooter

Shows: Effective permissions, conditions, org policies, VPC-SC

24. Domain Restricted Sharing (Super Important!)

iam.allowedPolicyMemberDomains

Purpose: Restrict which domains can be granted IAM access

Scope: Organization-wide enforcement

Allowed Values Format:

- C0xxxxxx (Cloud Identity Customer ID)
- C0xxxxxx,C0yyyyyy (multiple domains)

Exam Scenario:

Q: "Prevent employees from sharing with personal Gmail accounts"

A: Org policy: iam.allowedPolicyMemberDomains = [Your Customer ID]

Result: Can only grant to @company.com accounts

Finding Customer ID:

```
gcloud organizations describe ORG_ID --format="value(owner.directoryCustomerId)"
```

Common Exam Question Pattern

Q: "Data exfiltration via external user grants. How to prevent?"

Options:

- A) VPC Service Controls
- B) IAM Conditions
- C) Organization Policy: iam.allowedPolicyMemberDomains
- D) Regular audits

Answer: C - Enforces at grant time, prevents the issue

25. Cross-Project Service Account Usage (Tested Often!)

Pattern 1: SA in Project A Accesses Resource in Project B

Setup:

Project A: sa@project-a.iam.gserviceaccount.com

Project B: Has bucket "bucket-b"

Required:

Grant SA roles/storage.objectViewer on:

- Bucket in Project B, OR
- Project B itself

Does NOT require:

- VPC peering
- Shared VPC
- VPC-SC bridge (unless in different perimeters)

Pattern 2: User Attaches Project A's SA to Project B's VM

Required Permissions:

User needs on SA (Project A):

- roles/iam.serviceAccountUser

User needs on VM (Project B):

- roles/compute.instanceAdmin (or similar)

Exam Trap:

Q: "User can create VM but can't attach SA from other project"

A: Missing iam.serviceAccountUser on the SA

26. Organization Constraints Deep Dive

constraints/iam.disableServiceAccountCreation

Effect: Prevents SA creation entirely

Use Case: Centralized SA management

Exam Pattern:

Q: "Security team must approve all SAs before creation"

A: Enable this constraint + approval workflow

constraints/iam.disableWorkloadIdentityClusterCreation

Effect: Prevents GKE cluster creation with Workload Identity

Use Case: Force non-Workload Identity setup (rare)

Exam: Usually the WRONG answer (WIF is best practice)

constraints/compute.requireOsLogin

Effect: Forces OS Login on ALL VMs

Impact: Metadata SSH keys stop working

Migration: Users need roles/compute.osLogin or osAdminLogin

Exam Scenario:

Q: "After enabling requireOsLogin, developers can't SSH"

A: Grant them compute.osLogin role

NOT: Add SSH keys to metadata (policy blocks it)

constraints/compute.requireShieldedVm

Effect: All VMs must be Shielded VMs

Shielded VM Features:

- Secure Boot (prevents rootkit)
- vTPM (virtual Trusted Platform Module)
- Integrity Monitoring

Exam: Compliance/security questions → Shielded VMs

27. Shared VPC Advanced Patterns (High Frequency!)

Roles Summary

In HOST Project:

- compute.xpnAdmin: Attach/detach service projects
- compute.networkAdmin: Manage VPC/subnets/firewall
- compute.securityAdmin: Manage firewall rules only

In SERVICE Project:

- compute.networkUser: Use shared subnets (granted to SAs)
- compute.instanceAdmin: Create VMs using shared subnets

Subnet-Level Permissions (Important!)

Can grant compute.networkUser on:

- Host project (all subnets)
- Specific subnet (more restrictive)

Exam Pattern:

Q: "Team A should use subnet-a only, Team B subnet-b only"

A: Grant compute.networkUser at SUBNET level, not project level

Service Project SA Permissions

Required for SA to create VM in shared subnet:

1. Grant to SA in HOST project:
 - roles/compute.networkUser on subnet
2. Grant to user/SA in SERVICE project:
 - roles/compute.instanceAdmin (or similar)
 - roles/iam.serviceAccountUser (if attaching SA)

Exam Trap:

"SA in service project can't create VM in shared subnet"

Check: Does SA have networkUser on HOST project subnet?

28. Binary Authorization Integration (Appears in Sec Engineer Exam)

How It Relates to IAM

Binary Authorization: Controls which container images can be deployed

IAM Roles Involved:

- binaryauthorization.attestors.verifyImageAttested: Check attestations
- binaryauthorization.policy.update: Modify policy
- iam.serviceAccountTokenCreator: Create attestations

Exam Scenario:

Q: "Only security-approved images can deploy to GKE production"

A: Binary Authorization policy + IAM + attestation

Service Account Flow:

1. CI/CD SA creates attestation (needs attestor admin)
 2. GKE SA verifies attestation (needs verifier role)
 3. Deployment proceeds if valid
-

29. IAM Deny Policies (NEW - Likely to be Tested!)

Critical Difference from Org Policies

IAM Deny Policies (New Feature):

- Explicit DENY in IAM (not just org policies)
- Deny specific principals from specific actions
- Overrides any ALLOW

Use Case:

"Alice has Editor at org level, but should NOT delete prod resources"

Solution:

IAM Deny Policy:

Principal: user:alice@example.com

Permission: compute.instances.delete

Resource: prod-project

Result: Alice's Editor role still works EXCEPT delete in prod

Deny vs Org Policy

Org Policy:

- Denies CONFIGURATIONS (no external IPs)
- Applies to everyone
- Hierarchy-based

IAM Deny:

- Denies PERMISSIONS (specific actions)
- Applies to specific principals
- Policy-based

Exam: "Block specific user from action while keeping role"

Answer: IAM Deny Policy (if available in exam)

30. Access Transparency vs Access Approval (Confused Often!)

Access Transparency

What: LOGS of Google personnel accessing your data

Cost: Included in Premium Support

Visibility: You see WHO, WHAT, WHEN, WHY

Control: NO - cannot prevent, only see

Use Case: Audit trail for compliance

Access Approval

What: Requires YOUR approval before Google accesses

Cost: Additional feature on Premium Support

Control: YES - you approve or deny each request

Timeout: If you don't respond, request expires

Use Case: HIPAA, highly sensitive data

Exam Pattern

Q: "Must approve before Google personnel access data"

A: Access Approval

NOT: Access Transparency (only logging)

Q: "Audit trail of Google personnel access"

A: Access Transparency

31. Resource Manager Roles (Often Overlooked!)

Critical Roles

resourcemanager.organizationAdmin:

- Full access to ORG and all descendants
- Can grant any role
- "God mode" - very few people should have

resourcemanager.folderAdmin:

- Manage specific folders
- Grant IAM within folders
- Cannot affect parent or sibling folders

resourcemanager.projectCreator:

- Create new projects
- Automatically becomes project Owner
- Can specify folder/org placement

resourcemanager.projectDeleter:

- Delete projects
- 30-day recovery period
- Should be highly restricted

resourcemanager.projectIamAdmin:

- Manage project IAM only
- Cannot modify project itself
- Good for security team delegation

Exam Scenario

Q: "Developer needs to create projects in Dev folder but not Prod"

A: Grant resourcemanager.projectCreator on Dev folder only

NOT: Org level (too broad)

NOT: Owner (too permissive)

32. Billing Account IAM (Professional Architect Must-Know)

Billing Roles

billing.admin:

- Manage billing account
- Link/unlink projects
- View costs
- Manage payment methods

billing.user:

- Link projects to billing account
- CANNOT view costs
- For project creators

billing.viewer:

- View costs only
- Cannot modify

billing.costManager:

- View costs
- Manage budgets and exports
- Cannot modify billing account settings

Common Pattern

Setup for new project creation:

User needs:

1. resourcemanager.projectCreator (to create project)
2. billing.user (to link to billing)

If missing billing.user:

- Can create project
- Project remains unbilled (cannot use paid resources)

Exam Trap:

Q: "User created project but can't enable APIs"

A: Likely missing billing.user role (project not linked to billing)

33. Security Command Center (SCC) Integration

IAM Findings

SCC Detects:

- Overly permissive IAM
- Public buckets/datasets
- SA key age
- Owner role grants
- External members
- Unused service accounts

Findings Categories:

- PRIVILEGE_ESCALATION
- OVERLY_PERMISSIVE_ACCOUNT
- PUBLIC_BUCKET_ACL
- SERVICE_ACCOUNT_KEY_NOT_ROTATED

Required Role

securitycenter.admin: Manage SCC

securitycenter.findingsViewer: View findings

securitycenter.findingsEditor: Update findings

Exam Pattern:

Q: "Automatically detect overly permissive SAs across org"

A: Enable Security Command Center

NOT: Manual audit (doesn't scale)

NOT: IAM Recommender alone (SCC more comprehensive)

34. Context-Aware Access (Beyond Basic Access Levels)

Access Levels - Advanced

Device Attributes (BeyondCorp Enterprise):

- Device encryption status
- OS version
- Security patches applied
- Corp-managed vs BYOD
- Compromised device detection

Example:

Access Level: high_security

Requires:

- Device: Encrypted
- Device: Latest OS patches
- Device: Corp-managed
- Device: Not compromised
- IP: Corporate range
- Time: Business hours

Exam Scenario

Q: "C-level execs need access to financial data only from secure devices"

A: Context-aware access level with device policy

Grant role with this access level in VPC-SC or IAM condition

NOT: IP restriction only (insufficient)

NOT: MFA only (device could still be compromised)

35. Workload Identity Pool Management

Pool Attributes

Can have multiple providers in same pool:

- aws-prod-provider (AWS prod account)
- aws-dev-provider (AWS dev account)
- azure-provider (Azure workloads)

Exam: "Different AWS accounts need different access"

Solution: Multiple providers in same pool

OR separate pools (more isolation)

Attribute Extraction

Advanced Mapping:

```
google.subject = assertion.sub  
attribute.role = assertion.arn.extract('assumed-role/{role}/')  
attribute.account = assertion.account  
attribute.env = assertion.arn.contains('prod') ? 'prod' : 'dev'
```

Exam: "Grant access based on AWS role name"

Answer: attribute.role mapping + principalSet binding

36. Conditional IAM - Advanced Patterns

Multiple Conditions on Same Role

Can have MULTIPLE bindings with SAME role but DIFFERENT conditions:

Binding 1:

Role: roles/storage.objectViewer
Member: user:alice@example.com
Condition: Business hours

Binding 2:

Role: roles/storage.objectViewer
Member: user:bob@example.com
Condition: Weekends only

Result: Alice weekdays, Bob weekends, both have same role

Condition Functions

String Functions:

- .startsWith('prefix')

- .endsWith('suffix')
- .contains('substring')
- .matches('regex')
- .split('delimiter')
- .extract('regex')

Time Functions:

- .getFullYear()
- .getMonth() (0-11)
- .getDate() (1-31)
- .getHours() (0-23)
- .getMinutes()
- .getDayOfWeek() (0-6, 0=Sunday)

Exam: Know .getHours() and .getDayOfWeek() patterns

37. Audit Logging Configuration (Pre-requisite for Next Section)

Log Types

Admin Activity:

- Always enabled (free)
- Cannot disable
- Project/resource modifications
- 400 days retention

Data Access:

- Disabled by default (except BigQuery)
- Costs money (large volume)
- Who read/wrote data
- 30 days default retention

System Events:

- Always enabled (free)
- GCP system actions
- Migrations, automatic scaling
- 400 days retention

Policy Denied:

- Logs denied API calls
- Disabled by default
- Useful for debugging
- 30 days default retention

Enabling Data Access Logs

Enable for:

- Cloud Storage (GCS)
- BigQuery (default ON)
- Cloud SQL
- Firestore

Exam Pattern:

Q: "HIPAA requires audit trail of who accessed patient data"

A: Enable Data Access logs on relevant services

NOT: Admin Activity (doesn't log reads)

38. Critical Exam Anti-Patterns (Last Check!)

Anti-Pattern Checklist

- X Granting roles/owner (except break-glass)
 - X Using default SA with Editor role
 - X Downloading SA keys when on GCP
 - X Individual user grants (not groups)
 - X Primitive roles in production
 - X No folder structure with 50+ projects
 - X Manual IAM management (not IaC)
 - X Ignoring IAM Recommender suggestions
 - X Not enabling Data Access logs for sensitive data
 - X Sharing service accounts across environments
 - X Not using conditions for temporary access
 - X Trying to relax parent org policy at child
 - X Using ACLs on GCS (should use uniform access)
 - X Not monitoring IAM changes
 - X Service account with iam.securityAdmin
-

39. Pro Exam-Specific Decision Matrix

AUTHENTICATION DECISION MATRIX:

Workload Location	Best Solution	
GKE	Workload Identity	
GCE	Metadata Server	
Cloud Functions	Metadata Server	
Cloud Run	Metadata Server	
AWS	WIF (AWS)	
Azure	WIF (OIDC)	
GitHub Actions	WIF (OIDC)	
On-prem K8s	WIF (OIDC)	
On-prem VM	WIF (OIDC) or Keys*	

Partner external	WIF (OIDC) or Keys*
------------------	---------------------

*Keys only if WIF impossible

ACCESS MANAGEMENT DECISION MATRIX:

Requirement	Solution
User management	Groups
App authentication	Service Accounts
Temporary access	IAM Conditions
Auto-expire access	IAM Conditions
IP restriction	IAM Conditions
Time restriction	IAM Conditions
Resource pattern	IAM Conditions
Config enforcement	Org Policies
Data exfiltration	VPC-SC
External collab	WIF or Temp Grants

HIERARCHY DECISION MATRIX:

Scope	Grant Level
Company-wide	Organization
Department-wide	Folder
Team/App-specific	Project
Exception only	Resource

Final "Must Remember" List

The Absolute Essentials (Memorize These!)

1. **IAM = WHO, Org Policy = WHAT, VPC-SC = WHERE**
2. **Policy evaluation: Org Policy DENY → VPC-SC BLOCK → IAM ALLOW**
3. **Groups are 90% of user management answers**
4. **Keys are almost never right (except specific legacy cases)**
5. **Policies are additive, org policies are restrictive**
6. **Most permissive IAM wins, most restrictive org policy wins**
7. **Disable SA = immediate revocation across all resources**
8. **Workload Identity for GKE, WIF for external**
9. **Condition version = 3 required**
10. **cloudsql.client for Cloud SQL connection**
11. **serviceAccountUser = attach, serviceAccountTokenCreator = impersonate**

12. **workloadIdentityUser** for GKE/WIF
 13. **compute.networkUser** for Shared VPC
 14. **iam.allowedPolicyMemberDomains** prevents external grants
 15. Cannot relax org policy at lower levels
 16. VPC-SC perimeters prevent data exfiltration despite credentials
 17. Access Transparency = logs, Access Approval = control
 18. Primitive roles → NEVER in production (except very specific cases)
 19. Default SA with Editor → Disable via org policy
 20. One SA per app per environment
-

What You Now Have

 Complete IAM theory (Sections 1-14)  50 Professional-level practice questions 
Critical additions (above)  Quick reference cheat sheet  Exam keyword triggers
 Decision matrices  Anti-patterns checklist

Questions

Question 1

Your company has 500 developers across 5 teams. Currently, each developer has been individually granted the Editor role on their team's projects. A new compliance requirement mandates that you must be able to revoke all access for any developer within 5 minutes. What is the MOST efficient solution?

- A. Create a Cloud Function that iterates through all projects and removes the user's IAM bindings
 - B. Create Google Groups for each team, add developers to groups, grant roles to groups instead of individuals
 - C. Use IAM Conditions with a short expiration time that requires daily renewal
 - D. Implement a custom service account per developer and disable the service account when needed
-

Question 2

You are designing IAM for a multi-tenant SaaS application on GCP. Each customer's data must be isolated in separate projects. Customers should never be able to access other customers' data, even if credentials are compromised. What combination provides the STRONGEST isolation?

- A. Separate projects per customer + IAM policies + VPC firewall rules
 - B. Separate projects per customer + VPC Service Controls perimeters + Organization policies
 - C. Single project with customer ID in resource names + Row-level security in BigQuery
 - D. Separate projects per customer + Customer-managed encryption keys per customer
-

Question 3

Your data science team needs to query production BigQuery datasets from their development environment. The production data contains PII and must never be copied to development projects. Security policy prohibits any data exfiltration. What is the BEST approach?

- A. Grant the data science team BigQuery Data Viewer role on production project
 - B. Create authorized views in the development project that query production datasets
 - C. Use VPC Service Controls with a bridge perimeter between production and development
 - D. Set up VPC Service Controls around production with an ingress policy allowing development service accounts
-

Question 4

A contractor needs temporary access to Cloud Storage buckets in your production project for 3 months. Access should automatically expire without manual intervention. The contractor should only access buckets with names starting with "contractor-". What is the MOST secure implementation?

- A. Grant roles/storage.objectViewer with IAM condition: `request.time < timestamp('2025-09-15T00:00:00Z') && resource.name.startsWith('projects/_/buckets/contractor-')`
 - B. Create a custom role with storage permissions, set calendar reminder to revoke on September 15
 - C. Use a service account with key expiration set to 3 months, share key with contractor
 - D. Grant roles/storage.objectViewer, create a Cloud Scheduler job to revoke access in 3 months
-

Question 5

Your application running on GKE needs to access Cloud SQL and Cloud Storage. Currently, you're using a downloaded service account key stored as a Kubernetes secret. Security team mandates eliminating all service account keys. What should you implement?

- A. Use Workload Identity to bind Kubernetes service accounts to Google service accounts
 - B. Store the service account key in Google Secret Manager instead of Kubernetes secrets
 - C. Use the GKE node's default service account for all workloads
 - D. Create a new service account key monthly and rotate the Kubernetes secret
-

Question 6

Your organization has 3 environments (dev, staging, prod) across 30 projects. You need to enforce that NO resources can be created in Europe regions across all projects. What is the MOST efficient approach?

- A. Create a custom Cloud Function that deletes European resources on creation
 - B. Set organization policy `gcp.resourceLocations` with denied values `in:eu-regions` at organization level
 - C. Use a Terraform policy to validate region before resource creation
 - D. Grant IAM permissions only for US regions using custom roles
-

Question 7

You're implementing break-glass access for production emergencies. Requirements: (1) Full access when needed, (2) Access only from secure admin workstation, (3) Every use generates immediate alert, (4) Comprehensive audit trail. What solution meets ALL requirements?

- A. Create break-glass service account with Owner role, store key in physical safe, CloudWatch alert on usage
 - B. Create break-glass user account with Owner role + IAM condition for IP restriction + Cloud Logging alert + Access Transparency
 - C. Use Google Cloud's emergency access feature with automatic revocation after 4 hours
 - D. Create multiple admin accounts with MFA, distribute across security team members
-

Question 8

Your AWS Lambda functions need to write data to Google BigQuery. Security policy prohibits downloading service account keys. AWS account ID is 123456789012, Lambda uses IAM role `lambda-bigquery-writer`. What is the correct setup?

- A. Create service account, grant BigQuery Data Editor, download key and store in AWS Secrets Manager
 - B. Create Workload Identity Pool with AWS provider, bind AWS role ARN to GCP service account with `workloadIdentityUser` role
 - C. Use Cloud VPN between AWS and GCP, authenticate via private network
 - D. Create a Cloud Function as a proxy that accepts requests from Lambda
-

Question 9

Your security team discovered that the Compute Engine default service account has Editor role on all projects. You need to fix this across 100 existing projects and prevent future occurrences. What actions should you take?

- A. Remove Editor role from default SA in each project manually, create monitoring alert
- B. Enable organization policy `iam.automaticIamGrantsForDefaultServiceAccounts`, use a script to remove Editor role from existing projects
- C. Delete all default service

accounts, recreate them without roles D. Create custom roles with minimal permissions, replace Editor role on default SAs

Question 10

A third-party audit firm needs read-only access to ALL projects in your organization for 2 weeks to perform a compliance audit. They should NOT see IAM policies or billing data. What is the BEST approach?

- A. Grant auditors `roles/viewer` at organization level with IAM condition expiring in 2 weeks
 - B. Create custom role excluding IAM/billing permissions, grant at organization level with time-bound condition
 - C. Grant `roles/viewer` at organization level, create organization policy to hide IAM and billing
 - D. Create temporary projects with copies of data, grant auditors access to temporary projects only
-

Question 11

Your GitHub Actions workflow deploys to Cloud Run. Currently using a service account key stored in GitHub Secrets. GitHub repository is `myorg/production-app`. Only the `main` branch should deploy to production. What is the MOST secure solution?

- A. Continue using service account key but rotate it monthly
 - B. Set up Workload Identity Federation with attribute condition `assertion.repository == 'myorg/production-app' && assertion.ref == 'refs/heads/main'`
 - C. Use Workload Identity Federation without attribute conditions for flexibility
 - D. Create a Cloud Build trigger that deploys, give Cloud Build service account permissions
-

Question 12

You have a Shared VPC setup with host project `network-host` and service project `app-service`. A service account in `app-service` needs to create VMs that attach to subnets in the host project. What IAM configurations are required?

- A. Grant `compute.networkUser` to the service account at the host project level
 - B. Grant `compute.networkUser` to the service account on specific subnets in host project
 - C. Grant `compute.admin` to the service account on host project
 - D. Enable API in service project only; no additional IAM needed
-

Question 13

Your company is migrating from on-premises AD to Google Cloud. Users should authenticate once and access both on-prem and GCP resources. What identity solution should you implement?

- A. Sync AD users to Cloud Identity, set up SAML federation for SSO
 - B. Recreate all users manually in Cloud Identity
 - C. Use Google Workspace for identity management
 - D. Create service accounts for each user to access GCP
-

Question 14

An application service account has been compromised. You need to immediately revoke ALL access. The service account is attached to 50 GCE instances and 20 Cloud Functions. What is the FASTEST way to revoke access?

- A. Disable the service account immediately
 - B. Delete all service account keys
 - C. Remove all IAM bindings from the service account
 - D. Stop all GCE instances and disable all Cloud Functions
-

Question 15

Your organization requires that all VMs must use OS Login instead of metadata-based SSH keys for compliance. Some legacy projects still use metadata keys. How do you enforce this going forward while identifying non-compliant resources?

- A. Set organization policy `compute.requireOsLogin` to true, use Security Command Center to find violations
 - B. Manually configure OS Login on each VM, document the process
 - C. Create a custom Cloud Function to remove metadata SSH keys daily
 - D. Grant users only `compute.osLogin` role and revoke `compute.instanceAdmin`
-

Question 16

You need to grant a vendor temporary access to upload files to a specific Cloud Storage bucket for 7 days. Vendor will access from IP range 203.0.113.0/24. They should NOT be able to download, list, or delete objects. What is the BEST solution?

- A. Grant `roles/storage.objectCreator` with IAM condition: `request.time < timestamp('2025-12-31T00:00:00Z') && origin.ip in ['203.0.113.0/24']`
 - B. Create signed URL with 7-day expiration for uploads
 - C. Grant `roles/storage.objectAdmin` temporarily, manually revoke after 7 days
 - D. Create a custom role with only `storage.objects.create` permission, grant to vendor
-

Question 17

Your data engineering team uses a shared service account for multiple ETL pipelines. You need to identify which pipeline generated excessive BigQuery costs last week. How can you attribute costs to specific pipelines?

- A. Review audit logs to see which Compute Engine instance used the service account
 - B. Create separate service accounts per pipeline for cost attribution
 - C. Use BigQuery labels on queries to identify pipelines
 - D. Check Cloud Billing reports filtered by service account email
-

Question 18

You're implementing defense in depth for a project containing PHI data. IAM policies are configured correctly. What additional controls should you implement? (Choose the MOST comprehensive approach)

- A. VPC Service Controls perimeter + Organization policies + CMEK + Access Transparency
 - B. VPC firewall rules + Cloud Armor + HTTPS only
 - C. Enable all audit logs + Set up alerting
 - D. Private Google Access + Cloud NAT + Cloud VPN
-

Question 19

A developer accidentally granted `roles/owner` to `allUsers` on a production project. You discovered this 2 hours later. What immediate actions should you take?

- A. Remove the public IAM binding, review audit logs for unauthorized access, rotate all credentials, assess data exposure
 - B. Delete the project immediately to contain the breach
 - C. Remove the IAM binding and continue normal operations
 - D. Contact Google Cloud support to restore to previous state
-

Question 20

Your application needs to access Cloud SQL from Cloud Run. Cloud Run service uses service account `app-sa@project.iam.gserviceaccount.com`. What is the MINIMUM IAM configuration required?

- A. Grant `roles/cloudsql.admin` to app-sa
 - B. Grant `roles/cloudsql.client` to app-sa on the project or Cloud SQL instance
 - C. Grant `roles/cloudsql.editor` to app-sa
 - D. No IAM needed; Cloud Run can access Cloud SQL by default
-

Question 21

You need to allow developers to deploy to Cloud Run in development projects but prevent them from deploying to production. Both environments are in separate folders. What is the MOST scalable solution?

- A. Grant `roles/run.developer` at the development folder level only
 - B. Use IAM conditions on `roles/run.developer` to check project ID
 - C. Create custom role for Cloud Run deployment, grant at folder level
 - D. Review each deployment manually and approve production deployments
-

Question 22

Your organization policy `iam.disableServiceAccountKeyCreation` is enforced at the organization level. A legacy application in one project requires a service account key to function. What should you do?

- A. Disable the organization policy temporarily to create the key
 - B. Create a separate folder without this policy, move the project to that folder
 - C. Request exception from security team, document risk, create time-bound key with rotation plan
 - D. Refactor the application immediately to remove key dependency
-

Question 23

You have a BigQuery dataset that should only be accessible during business hours (9 AM - 5 PM) from office IP (203.0.113.0/24). How do you implement this?

- A. Create IAM condition: `request.time.getHours() >= 9 && request.time.getHours() < 17 && origin.ip in ['203.0.113.0/24']`
 - B. Use BigQuery scheduled queries to copy data only during business hours
 - C. Implement application-level access control
 - D. Use VPC Service Controls with time-based access levels
-

Question 24

Your company uses Google Workspace. A user `alice@company.com` has `roles/editor` on a project. Alice leaves the company and her account is deleted from Workspace. What happens to her IAM permissions?

- A. Permissions automatically revoked when account deleted
 - B. Permissions remain in IAM policy as "deleted:user:alice@company.com" and become ineffective
 - C. Permissions transfer to her manager automatically
 - D. Project becomes inaccessible until permissions are manually updated
-

Question 25

You need to grant external partners access to specific Cloud Storage buckets. Partners authenticate via their own Okta instance. Partners should NOT have Google accounts. What solution should you implement?

- A. Create Google accounts for all partners
 - B. Use Workload Identity Federation with OIDC provider (Okta), map Okta groups to service accounts
 - C. Generate signed URLs for each partner
 - D. Create service accounts, send keys to partners via encrypted email
-

Question 26

Your security policy requires that all service accounts must be reviewed and approved before creation. How do you enforce this?

- A. Set organization policy to disable service account creation, use centralized request/approval workflow
 - B. Create custom Cloud Function to send approval requests when SAs are created
 - C. Review audit logs weekly and delete unauthorized service accounts
 - D. Grant `iam.serviceAccountAdmin` role only to security team
-

Question 27

An application running on GCE needs to access Cloud Storage. The VM is in project `app-project` and the bucket is in project `storage-project`. What IAM configuration is needed?

- A. Grant the VM's service account `roles/storage.objectViewer` on the bucket or `storage-project`
 - B. Enable VPC peering between projects
 - C. Grant `roles/storage.objectViewer` at organization level
 - D. No configuration needed; same organization resources can access each other
-

Question 28

You're designing a CI/CD pipeline where code review approval is required before production deployment. The pipeline should enforce: (1) developer initiates, (2) tech lead approves, (3) automated system deploys. How do you implement this with IAM?

- A. Single service account with `roles/run.admin`, manual approval in pipeline
- B. Three service accounts: dev-sa (trigger), approve-sa (approval), deploy-sa (deployment). Use service account impersonation chain where dev-sa → approve-sa → deploy-sa
- C. Use Cloud Build with manual approval step
- D. Grant developers `roles/run.admin` with approval workflow in ticketing system

Question 29

You need to provide a security auditor with read-only access to ALL audit logs including Data Access logs for 30 days. What is the MINIMUM permission required?

- A. `roles/logging.viewer`
 - B. `roles/logging.privateLogViewer`
 - C. `roles/viewer`
 - D. `roles/logging.admin`
-

Question 30

Your GKE cluster uses Workload Identity. A pod needs to access Cloud SQL. The Kubernetes service account is `app-ksa` in namespace `production`. The Google service account is `app-gsa@project.iam.gserviceaccount.com`. What configurations are required?

- A. Annotate K8s SA with GCP SA email, grant `roles/iam.workloadIdentityUser` to `serviceAccount:project.svc.id.goog[production/app-ksa]` on GCP SA, grant `roles/cloudsql.client` to GCP SA
 - B. Just grant `roles/cloudsql.client` to GKE node service account
 - C. Create service account key for GCP SA, mount as Kubernetes secret
 - D. Enable Cloud SQL proxy in the cluster
-

Question 31

Organization policy `compute.vmExternalIpAccess` is set to DENY ALL at organization level. A specific project needs to allow external IPs on NAT gateway VMs. What should you do?

- A. Override the policy at project level to allow all external IPs
 - B. Cannot be done; organization policy cannot be overridden
 - C. Create separate organization without this policy
 - D. Use IAM conditions to allow specific VMs to have external IPs
-

Question 32

You have 5 projects: dev1, dev2, dev3, staging, prod. Developers should have Editor access to all dev projects but only Viewer access to staging and prod. What is the MOST efficient folder structure?

- A. Flat structure: Grant roles individually per project
- B. Two folders: "Development" (dev1-3) and "Non-Production" (staging, prod); grant Editor on Development folder, Viewer on Non-Production folder
- C. Three folders: "Dev", "Staging", "Prod"; grant roles per folder
- D. No folders; use IAM conditions based on project name patterns

Question 33

Your application uses a service account to write to BigQuery. You want to ensure this service account CANNOT create or delete datasets, only insert rows into existing tables. What role should you grant?

- A. `roles/bigquery.dataEditor`
 - B. `roles/bigquery.admin`
 - C. `roles/bigquery.user`
 - D. Custom role with only `bigquery.tables.updateData` permission
-

Question 34

You need to implement least privilege for Cloud Functions that read from Pub/Sub and write to BigQuery. What roles should the Cloud Function's service account have?

- A. `roles/editor` on the project
 - B. `roles/pubsub.subscriber` on subscription, `roles/bigquery.dataEditor` on dataset
 - C. `roles/pubsub.admin` and `roles/bigquery.admin`
 - D. `roles/viewer` is sufficient for reading and writing
-

Question 35

A compliance requirement states: "No individual should have the ability to both create and delete production databases without approval." How do you implement this?

- A. Grant `roles/cloudsql.editor` which allows both create and delete
 - B. Create two custom roles: one with create permissions, one with delete permissions; assign to different people/groups
 - C. Use IAM conditions to require approval for delete operations
 - D. Implement approval workflow in application layer
-

Question 36

Your organization has 100 projects. You need to identify all service accounts that have the Owner role across all projects. What's the MOST efficient method?

- A. Manually check IAM page for each project
 - B. Use Cloud Asset Inventory to export IAM policies, query for Owner role assignments to service accounts
 - C. Write a script that calls `gcloud projects get-iam-policy` for each project
 - D. Use Security Command Center to find Owner role findings
-

Question 37

You're implementing HIPAA compliance. Google personnel should NOT access your data without your explicit approval. What GCP feature should you enable?

- A. VPC Service Controls
 - B. Access Transparency and Access Approval
 - C. Customer-Managed Encryption Keys (CMEK)
 - D. Private Google Access
-

Question 38

An application's service account needs to list all Cloud Storage buckets in a project but should NOT be able to read bucket contents. What is the MINIMUM role?

- A. `roles/storage.objectViewer`
 - B. `roles/storage.admin`
 - C. Custom role with `storage.buckets.list` permission
 - D. `roles/viewer`
-

Question 39

Your company operates in US, EU, and APAC. Data residency laws require EU customer data stay in EU regions. You have 50 projects. What's the BEST way to enforce this?

- A. Create EU-specific folder, set organization policy `gcp.resourceLocations` to allow only `in:eu-locations` on that folder
 - B. Train developers to create resources in correct regions
 - C. Use IAM conditions based on resource location
 - D. Manually audit resource locations monthly
-

Question 40

You need to grant a Cloud Function the ability to impersonate a service account to access BigQuery. What roles are required?

- A. Grant `roles/iam.serviceAccountTokenCreator` to Cloud Function's SA on target SA
 - B. Grant `roles/iam.serviceAccountUser` to Cloud Function's SA on target SA
 - C. Both `roles/iam.serviceAccountUser` and `roles/iam.serviceAccountTokenCreator`
 - D. Grant `roles/iam.serviceAccountAdmin` to Cloud Function's SA
-

Question 41

Your VPC Service Controls perimeter blocks all egress. A batch job needs to export data to an external partner's GCS bucket in a different organization. What's the CORRECT solution?

-
- A. Remove VPC Service Controls perimeter
 - B. Create egress policy allowing the specific service account to access the external bucket
 - C. Create a bridge perimeter to the partner's perimeter
 - D. VPC-SC cannot allow any egress; must use different approach
-

Question 42

You have a custom role `customRole1` created at the project level. You need to use this role in another project. What should you do?

- A. Copy the role definition and create it in the second project
 - B. Grant the role at organization level; it will be available in all projects
 - C. Custom roles created at project level can only be used in that project; must recreate or create at org level
 - D. Export the role and import it into the second project
-

Question 43

Your security team wants to detect when anyone grants the Owner role on any project. What should you implement?

- A. Create log sink with filter `protoPayload.methodName="SetIamPolicy"` AND `protoPayload.request.policy.bindings.role="roles/owner"`, route to Pub/Sub, trigger alert
 - B. Review IAM policies manually each day
 - C. Enable Security Command Center Premium
 - D. Use Cloud Asset Inventory snapshots and compare daily
-

Question 44

An external consultant needs to run a one-time data analysis on your BigQuery dataset. Access should be restricted to their specific query and should expire in 24 hours. What's the MOST secure approach?

- A. Create authorized view with consultant's query, grant them access to view with time-bound condition
 - B. Grant them `roles/bigquery.dataViewer` with time-bound condition
 - C. Extract data to CSV, send via encrypted email
 - D. Create temporary dataset copy, grant access to copy
-

Question 45

Your application uses multiple service accounts (app-sa, db-sa, storage-sa). You want to ensure that if one SA is compromised, it cannot escalate privileges by modifying IAM policies. What should you verify?

-
- A. None of the service accounts have `roles/iam.securityAdmin` or `roles/owner`
 - B. All service accounts use user-managed keys
 - C. Service accounts are in different projects
 - D. All service accounts have `roles/viewer` only
-

Question 46

You need to implement a "four-eyes principle" for production deployments: any deployment requires approval from two different people before execution. How do you enforce this with IAM?

- A. Not possible with IAM alone; requires workflow management system
 - B. Use IAM conditions that check for two different user approvals
 - C. Create deployment service account that only deployment tool can impersonate, require two approvers to authorize impersonation
 - D. Grant deployment permissions to a group, require two group members to approve
-

Question 47

Your organization is migrating from on-premises to GCP. During migration, on-prem Kubernetes clusters need to access GCP BigQuery. The on-prem clusters are not accessible from internet. What's the BEST solution?

- A. Set up Cloud VPN, use Workload Identity Federation with K8s OIDC
 - B. Create service account keys, securely transfer to on-prem
 - C. Use Cloud Interconnect for private connectivity, then configure Workload Identity Federation
 - D. Migrate Kubernetes to GKE first, then access BigQuery
-

Question 48

You discovered that a service account has 20 user-managed keys created over 3 years. Most keys are unused. What actions should you take?

- A. Delete all keys immediately
 - B. Identify which keys are actively used (check audit logs), disable unused keys, plan to migrate to Workload Identity, enable alerting for new key creation
 - C. Rotate all keys as they're probably compromised
 - D. Leave them; if not being used, no security risk
-

Question 49

Your Cloud Run service needs to invoke another Cloud Run service. Both services are private (require authentication). What's the MINIMUM IAM configuration?

- A. First service's SA needs `roles/run.invoker` on second service B. Both services need `allUsers` with `roles/run.invoker` C. First service needs `roles/run.admin` D. No IAM needed; same project services can invoke each other
-

Question 50

You're architecting a multi-tenant SaaS where each customer gets their own GCP project. You need a centralized way to enforce security policies across all customer projects. What should you implement?

- A. Organization policies at organization level + folder structure for customer tiers + VPC Service Controls for sensitive customers B. Manually configure each project with same settings C. Create master project with policies, copy to customer projects D. Use Terraform to maintain consistent configuration across projects
-
-

ANSWERS AND EXPLANATIONS

Answer 1: B

Explanation: Google Groups provide centralized management. Adding/removing users from groups instantly affects access across all projects. Options A and D are complex and error-prone. Option C requires daily maintenance. Groups are the industry standard for managing access at scale.

Key Concept: Use groups, not individual users, for scalable IAM management.

Answer 2: B

Explanation: VPC Service Controls prevent data exfiltration even with compromised credentials (defense in depth). Organization policies enforce configurations. Separate projects provide resource isolation. Option A lacks data exfiltration protection. Option C has weak isolation. Option D helps but doesn't prevent copying data to external projects.

Key Concept: VPC-SC provides data exfiltration protection beyond IAM.

Answer 3: D

Explanation: VPC Service Controls with ingress policy allows specific principals to query data without allowing data copies to leave the perimeter. Option A allows unrestricted access. Option B (authorized views) can still copy data. Option C (bridge) is too permissive.

Key Concept: VPC-SC ingress policies control who can access protected resources.

Answer 4: A

Explanation: IAM conditions provide automatic expiration and resource name filtering in a single binding. No manual intervention needed. Option B requires manual revocation. Option C is insecure (key download). Option D is complex and less reliable.

Key Concept: Use IAM conditions for time-bound and resource-specific access.

Answer 5: A

Explanation: Workload Identity eliminates service account keys by binding Kubernetes service accounts to Google service accounts. GKE metadata server provides tokens automatically. Option B still has keys. Option C violates least privilege. Option D still has key management burden.

Key Concept: Workload Identity is the recommended way to authenticate GKE pods to GCP.

Answer 6: B

Explanation: Organization policies are designed for this purpose - enforcing configuration constraints across resources. Setting at org level affects all projects. Option A is reactive, not preventive. Option C doesn't prevent. Option D is not how IAM works.

Key Concept: Use organization policies to enforce resource location restrictions.

Answer 7: B

Explanation: Meets all requirements: Owner role (full access), IAM condition for IP restriction (secure location), audit logs + Access Transparency (comprehensive audit), alerting (immediate notification). User accounts are better than SA for break-glass (human identity). Option A uses SA (less auditable). Option C doesn't exist. Option D lacks automation.

Key Concept: Break-glass accounts need full access, IP restrictions, and comprehensive logging.

Answer 8: B

Explanation: Workload Identity Federation allows AWS roles to impersonate GCP service accounts without keys. Bind specific AWS role ARN to GCP SA. Option A violates no-keys policy. Option C is complex. Option D adds unnecessary complexity.

Key Concept: Workload Identity Federation eliminates keys for external workloads.

Answer 9: B

Explanation: Organization policy prevents future automatic grants. Script remediates existing projects efficiently. Option A doesn't prevent future occurrences. Option C breaks existing workloads. Option D doesn't address root cause.

Key Concept: Use `iam.automaticIamGrantsForDefaultServiceAccounts` org policy.

Answer 10: B

Explanation: Custom role provides precise permissions (viewer minus IAM/billing). Time-bound condition ensures automatic revocation. Options A includes IAM/billing. Option C - org policies don't hide data from granted principals. Option D is operationally complex.

Key Concept: Custom roles enable precise permission control for specific scenarios.

Answer 11: B

Explanation: Workload Identity Federation with specific attribute conditions ensures only main branch can deploy. No keys involved. Option A still has keys. Option C allows any branch. Option D doesn't solve authentication issue.

Key Concept: Use WIF attribute conditions for branch/repo-specific access in GitHub Actions.

Answer 12: B

Explanation: Shared VPC requires `compute.networkUser` role on specific subnets (or host project). Subnet-level is more granular and secure. Option A works but too broad. Option C is excessive. Option D is incorrect.

Key Concept: Shared VPC requires networkUser role on host project or subnets.

Answer 13: A

Explanation: Cloud Identity syncs with AD (using GCDS or cloud sync). SAML federation provides SSO. Users authenticate once against AD, access both on-prem and GCP. Option B loses SSO. Option C isn't necessary if you don't need Gmail/Drive. Option D is for applications, not users.

Key Concept: Cloud Identity + SAML federation enables hybrid identity and SSO.

Answer 14: A

Explanation: Disabling service account immediately revokes ALL active tokens and prevents new token generation. Takes effect in seconds. Option B only stops key-based auth (metadata server still works). Option C is slow (many bindings). Option D doesn't revoke SA permissions.

Key Concept: Disable service account for immediate access revocation.

Answer 15: A

Explanation: Organization policy enforces going forward. Security Command Center detects existing violations. Option B is manual and doesn't scale. Option C is reactive. Option D doesn't enforce, only limits.

Key Concept: Org policies enforce; SCC detects existing violations.

Answer 16: A

Explanation: `roles/storage.objectCreator` allows only uploads (not download/list/delete). IAM conditions enforce time limit and IP restriction. Option B (signed URL) could work but less flexible. Option C too permissive. Option D doesn't include time/IP restrictions.

Key Concept: Combine minimal roles with IAM conditions for temporary, restricted access.

Answer 17: B

Explanation: Separate service accounts per pipeline enables cost attribution and security isolation. Option A is indirect. Option C helps but doesn't solve IAM attribution. Option D doesn't differentiate between pipelines using same SA.

Key Concept: Use separate service accounts for different workloads/applications.

Answer 18: A

Explanation: Defense in depth requires multiple layers: VPC-SC (data exfiltration), org policies (configuration), CMEK (encryption control), Access Transparency (Google access visibility). Options B, C, D are single-layer controls.

Key Concept: Defense in depth combines multiple security controls.

Answer 19: A

Explanation: Remove binding immediately (stop bleeding), investigate via audit logs (who accessed what), rotate credentials (assume compromise), assess data exposure (impact). Option B is too drastic. Option C ignores potential breach. Option D - Google can't restore IAM policies.

Key Concept: Incident response: contain, investigate, rotate, assess.

Answer 20: B

Explanation: `roles/cloudsql.client` is minimum permission needed to connect to Cloud SQL. Can be granted at project or instance level. Option A is excessive. Option C includes unnecessary permissions. Option D is incorrect.

Key Concept: `cloudsql.client` role enables connection to Cloud SQL instances.

Answer 21: A

Explanation: Folder-level grants scale efficiently. Grant deployment role only on dev folder. Production folder has no such grant. Option B works but less scalable. Option C doesn't differentiate environments. Option D doesn't scale.

Key Concept: Use folder hierarchy for environment-based access control.

Answer 22: C

Explanation: Document exception with business justification, assess risk, create key with rotation plan and expiration date. Option A violates policy. Option B undermines policy purpose. Option D may not be immediate option.

Key Concept: Exceptions should be documented, time-bound, with mitigation plans.

Answer 23: A

Explanation: IAM conditions can combine time and IP restrictions in single expression. Directly enforced by IAM. Option B doesn't restrict access. Option C is application-layer (not IAM). Option D - VPC-SC access levels don't support time-based natively in the same way.

Key Concept: IAM conditions support time and network-based access control.

Answer 24: B

Explanation: Deleted user IAM bindings remain as "deleted:user:..." but become ineffective. Manual cleanup recommended. Option A is not automatic. Option C doesn't happen. Option D is incorrect.

Key Concept: Deleted user IAM bindings remain but are ineffective; clean up manually.

Answer 25: B

Explanation: Workload Identity Federation with OIDC (Okta) allows external users to authenticate with their existing credentials, map to GCP service accounts. No Google accounts needed. Option A defeats purpose. Option C not suitable for ongoing access. Option D insecure.

Key Concept: Workload Identity Federation enables external identity integration.

Answer 26: A

Explanation: Organization policy disables SA creation. Centralized workflow provides approval process before creation. Option B is reactive (after creation). Option C is reactive. Option D doesn't enforce approval.

Key Concept: Disable SA creation via org policy; centralize through approval workflow.

Answer 27: A

Explanation: Cross-project access requires explicit IAM grant on target resource or project. Option B is networking, not IAM. Option C too broad. Option D is incorrect.

Key Concept: Cross-project resource access requires explicit IAM grants.

Answer 28: B

Explanation: Service account impersonation chain enforces roles: dev-sa initiates, impersonates approve-sa (tech lead controls), which impersonates deploy-sa (has deployment permissions). Clear separation and audit trail. Option A doesn't enforce approval. Option C is manual. Option D doesn't enforce via IAM.

Key Concept: Service account chaining enforces multi-level approvals.

Answer 29: B

Explanation: `roles/logging.privateLogViewer` grants access to Data Access logs (private logs). `roles/logging.viewer` doesn't include Data Access logs. Option C doesn't specifically grant log access. Option D is excessive (admin can modify logs).

Key Concept: privateLogViewer role required for Data Access log reading.

Answer 30: A

Explanation: Workload Identity requires: K8s SA annotation, `workloadIdentityUser` binding with format `serviceAccount:PROJECT.svc.id.goog[NAMESPACE/KSA]`, and resource permissions on GCP SA. All three are required. Options B, C, D don't implement Workload Identity correctly.

Key Concept: Workload Identity requires annotation, IAM binding, and resource permissions.

Answer 31: B

Explanation: Organization policies cannot be relaxed at lower levels (by design). Child can only be more restrictive. Option A impossible. Option C impractical. Option D - IAM conditions don't override org policies.

Key Concept: Organization policies cannot be overridden/relaxed at child levels.

Answer 32: B

Explanation: Two folders enable efficient policy management via inheritance. All dev projects in one folder (Editor granted once). Staging/prod in another folder (Viewer granted

once). Option A doesn't scale. Option C creates unnecessary folders. Option D - conditions are complex for this use case.

Key Concept: Folder structure enables efficient policy management via inheritance.

Answer 33: D

Explanation: Custom role with only `bigrquery.tables.updateData` (and `get/list` if needed for tables) provides minimum permissions. Option A includes dataset modification. Option B far too broad. Option C includes dataset creation.

Key Concept: Use custom roles for precise minimum permissions.

Answer 34: B

Explanation: `pubsub.subscriber` for reading messages, `bigrquery.dataEditor` for writing to tables. Scoped to specific resources (subscription, dataset). Option A too broad. Option C too broad (admin includes delete). Option D - viewer cannot write.

Key Concept: Grant minimum roles scoped to specific resources.

Answer 35: B

Explanation: Separation of duties via separate roles to different people. One person/group has create-only, another has delete-only (with approval). Option A violates requirement. Option C - conditions can't require approval. Option D is outside IAM.

Key Concept: Separation of duties via role separation across principals.

Answer 36: B

Explanation: Cloud Asset Inventory can export all IAM policies across organization, then query/filter efficiently. Option A doesn't scale. Option C works but less efficient. Option D - SCC for security findings, not comprehensive IAM queries.

Key Concept: Cloud Asset Inventory for organization-wide IAM policy analysis.

Answer 37: B

Explanation: Access Transparency shows when Google accesses your data. Access Approval requires your explicit approval before Google personnel access. Option A prevents data exfiltration. Option C encrypts data. Option D is networking.

Key Concept: Access Transparency + Approval for Google personnel access control.

Answer 38: C

Explanation: Custom role with only `storage.buckets.list` (and `storage.buckets.get`) provides minimum permission to list buckets without reading objects. Options A, B, D include unnecessary permissions.

Key Concept: Custom roles for specific, minimal permissions.

Answer 39: A

Explanation: Folder for EU projects + org policy restricting locations enforces at scale. Option B doesn't enforce. Option C - IAM conditions don't control resource creation location. Option D is reactive.

Key Concept: Folder + org policy combination for region-specific enforcement.

Answer 40: A

Explanation: `serviceAccountTokenCreator` allows generating access tokens for the target SA (impersonation). Option B (`serviceAccountUser`) is for attaching SA to resources. Option C - only TokenCreator needed for this scenario. Option D too broad.

Key Concept: `serviceAccountTokenCreator` for generating tokens/impersonation.

Answer 41: B

Explanation: Egress policy allows controlled data export to specific external destinations. Option A defeats security. Option C - bridges connect perimeters (both must be VPC-SC perimeters). Option D incorrect - egress policies exist for this.

Key Concept: VPC-SC egress policies allow controlled external data access.

Answer 42: C

Explanation: Project-level custom roles only exist in that project. Must recreate in second project or create at org level for org-wide availability. Options A, B, D - roles can't be moved or shared directly.

Key Concept: Project custom roles are project-specific; org custom roles are org-wide.

Answer 43: A

Explanation: Log sink with filter captures SetIamPolicy calls with Owner role, routes to Pub/Sub for real-time alerting. Option B doesn't scale/automate. Option C provides findings but custom alert is better. Option D is periodic, not real-time.

Key Concept: Log sinks + filters for real-time IAM change monitoring.

Answer 44: A

Explanation: Authorized view limits access to specific query results only. Time-bound condition on view access provides expiration. Most secure - consultant can't modify query or access raw data. Option B allows any query. Options C, D operationally complex.

Key Concept: Authorized views + time-bound conditions for restricted temporary access.

Answer 45: A

Explanation: Service accounts with IAM admin roles (`securityAdmin`, `owner`, `iam.serviceAccountAdmin`) can modify IAM policies = privilege escalation risk. Verify none have these. Option B is about keys (different concern). Option C doesn't prevent escalation. Option D too restrictive for applications.

Key Concept: Prevent service accounts from having IAM administrative permissions.

Answer 46: A

Explanation: IAM cannot enforce approval workflows. Requires external workflow system (Cloud Build with approvals, Spinnaker, custom solution) integrated with IAM. Options B, C, D - IAM doesn't support approval requirements.

Key Concept: IAM provides access control; workflow tools provide approval processes.

Answer 47: C

Explanation: Cloud Interconnect provides private connectivity to GCP. Then Workload Identity Federation (K8s OIDC) allows on-prem pods to authenticate to GCP without keys. Option A - VPN works but Interconnect is more robust for migration. Option B violates no-keys policy. Option D delays migration unnecessarily.

Key Concept: Private connectivity + Workload Identity Federation for hybrid cloud.

Answer 48: B

Explanation: Comprehensive approach: audit logs show usage, disable unused (not delete immediately - may break something), plan migration (long-term solution), prevent future issues (alerting). Option A risky (may break production). Options C, D incomplete.

Key Concept: Methodical approach to SA key remediation: identify, disable, migrate, prevent.

Answer 49: A

Explanation: Invoking private Cloud Run service requires `run.invoker` role. First service's SA needs this on second service. Option B makes services public (not desired). Option C too broad. Option D incorrect.

Key Concept: `run.invoker` role required to invoke private Cloud Run services.

Answer 50: A

Explanation: Organization policies at org level provide baseline for all projects. Folder structure enables tier-specific policies (enterprise customers get VPC-SC, etc.). VPC-SC for high-security customers. Option B doesn't scale. Options C, D don't provide centralized enforcement.

Key Concept: Org policies + folder hierarchy + VPC-SC for multi-tenant security.

Audit Logging

1. Audit Logging Fundamentals - The Essentials

The Core Concept

Every API call to GCP = Audit Log Entry (if enabled)

Who: principalEmail (user/SA)

What: methodName (API method called)

When: timestamp

Where: resource, callerIp

Result: status (success/failure)

Four Log Types (MEMORIZE!)

Log Type	Always On?	Cost	Retention	What It Logs
Admin Activity	✓ Yes	FREE	400 days	Configuration changes (create, delete, update)
Data Access	✗ No*	PAID 	30 days	Data reads/writes (*BigQuery READ on)
System Events	✓ Yes	FREE	400 days	GCP-initiated actions (migrations, auto-scaling)
Policy Denied	✗ No	PAID 	30 days	Blocked API calls (IAM/Org Policy denied)

Exam Trap: BigQuery Data Access READ logs are enabled by default (exception!)

2. Audit Log Types - Deep Details

Admin Activity Logs

What: Configuration/metadata changes **Cannot disable:** Always on **Volume:** Low-medium

Examples:

- ✓ compute.instances.insert (create VM)
- ✓ storage.buckets.delete (delete bucket)
- ✓ SetIamPolicy (change permissions)
- ✓ compute.firewalls.update (modify firewall)
- ✓ serviceusage.services.enable (enable API)

NOT in Admin Activity:

- ✗ storage.objects.get (read file - Data Access)
- ✗ bigquery.jobs.query (query data - Data Access)
- ✗ compute.instances.list (list VMs - Data Access ADMIN_READ)

Exam Pattern: "Track who deleted the VM" → Admin Activity logs

Data Access Logs

What: Reading or writing data **Default:** OFF (except BigQuery READ) **Cost:** Can be VERY expensive

Three Subtypes:

ADMIN_READ:

- Metadata reads (list, get properties)
- Example: storage.buckets.list, compute.instances.list
- Relatively lower volume

DATA_READ:

- Actual data reads
- Example: storage.objects.get, bigquery.tables.getData
- HIGHEST volume

DATA_WRITE:

- Data modifications
- Example: storage.objects.create, bigquery.tabledata.insertAll
- High volume

Services with Data Access Logs:

- Cloud Storage (GCS)
- BigQuery
- Cloud SQL
- Firestore/Datastore
- Cloud Spanner
- Secret Manager
- Pub/Sub
- Cloud Functions (executions)

Exam Pattern:

- "HIPAA requires tracking who accessed patient data" → Enable Data Access logs
- "Track all BigQuery queries" → Data Access READ logs
- "Logs too expensive" → Disable Data Access or use exclusion filters

System Events Logs

What: GCP-initiated automatic actions **Examples:**

- Live migration of VMs
- Automatic restarts
- GKE node auto-repairs
- System maintenance
- Internal VM state changes

Exam: Rarely the primary answer, but know they exist

Policy Denied Logs

What: API calls blocked by IAM or Organization Policies **Default:** OFF **Use:** Security monitoring, debugging permissions

Example Log:

```
{  
  "protoPayload": {  
    "status": {  
      "code": 7, // PERMISSION_DENIED  
      "message": "Permission denied"  
    },  
    "authenticationInfo": {  
      "principalEmail": "alice@example.com"  
    },  
    "methodName": "compute.instances.delete"  
  }  
}
```

Exam Pattern:

- "Detect unauthorized access attempts" → Enable Policy Denied logs
 - "User can't figure out why access denied" → Check Policy Denied logs
-

3. Log Structure & Critical Fields

Key Fields (Know These!)

```
json
{
  "protoPayload": {
    "authenticationInfo": {
      "principalEmail": "WHO"
    },
    "authorizationInfo": [
      "permission": "WHAT permission checked",
      "granted": true/false
    ],
    "requestMetadata": {
      "callerIp": "WHERE from",
      "callerSuppliedUserAgent": "WHAT tool"
    },
    "serviceName": "WHICH service",
    "methodName": "WHICH API method",
    "resourceName": "WHICH resource",
    "request": { "WHAT was requested" },
    "response": { "WHAT was returned" },
    "status": { "RESULT - success/error" }
  },
  "timestamp": "WHEN",
  "severity": "INFO/NOTICE/WARNING/ERROR",
  "resource": {
    "type": "Resource type",
    "labels": { "project_id", "zone", etc. }
  }
}
...
```

Most Important Fields for Exam

authenticationInfo.principalEmail: Who made the call
methodName: Which API method (e.g., `SetIamPolicy`, `compute.instances.insert`)
resourceName: Full resource path
status.code: 0 = success, 7 = permission denied
timestamp: When it happened
callerIp: Source IP address

4. Enabling & Configuring Audit Logs

Configuration Hierarchy

...
Organization Level (applies to all projects)
↓
Folder Level (applies to folder projects)
↓
Project Level (applies to project only)

Inheritance: Child inherits parent configuration (can make MORE restrictive)

Enabling Data Access Logs

Per Service, Per Project/Folder/Org:

yaml

Admin Activity: Cannot configure (always on)

Data Access:

Service: allServices (or specific like storage.googleapis.com)

Log Types:

- ADMIN_READ
- DATA_READ
- DATA_WRITE

Exemptions: (optional)

- principalEmail: "admin@example.com"
- principalEmail: "trusted-sa@project.iam.gserviceaccount.com"

Exemptions: Exclude specific users/SAs from Data Access logging (reduce cost)

Configuration Methods

Console: IAM & Admin → Audit Logs **gcloud**:

```
bash
# Get current config
gcloud projects get-iam-policy PROJECT_ID \
--format=json > policy.json

# Edit audit config in policy.json
# Set policy back

gcloud projects set-iam-policy PROJECT_ID policy.json
```

Terraform:

```
hcl
resource "google_project_iam_audit_config" "project" {
  project = "my-project"
  service = "allServices"
```

```
audit_log_config {  
    log_type = "DATA_READ"  
    exempted_members = [  
        "user:admin@example.com"  
    ]  
}  
}  
...  
---
```

5. Log Retention & Storage

Default Retention

Log Type	Default Retention	Can Change?
Admin Activity	400 days	✗ No
Data Access	30 days	✗ No
System Events	400 days	✗ No
Policy Denied	30 days	✗ No

CRITICAL: Cannot change retention in Cloud Logging directly!

Extending Retention (Exam Favorite!)

Method 1: Log Sinks to Cloud Storage

...

Retention: Configure bucket lifecycle policy (any duration)

Cost: Cloud Storage pricing (cheap)

Use: Long-term archival, compliance (7+ years)

Format: JSON files

Best Practice: Use Coldline/Archive storage class

...

Method 2: Log Sinks to BigQuery

...

Retention: Configure table expiration (any duration)

Cost: BigQuery storage pricing

Use: Analysis, querying, dashboards

Format: BigQuery table

Best Practice: Partition tables by date, set expiration per partition

...

Method 3: Log Sinks to Pub/Sub

Retention: N/A (streaming)

Cost: Pub/Sub + downstream processing

Use: Real-time processing, SIEM integration

Format: JSON messages

Best Practice: For real-time alerting/forwarding

****Exam Pattern**:**

- "Retain logs for 10 years" → Sink to GCS with lifecycle policy
- "Analyze logs with SQL" → Sink to BigQuery
- "Send to Splunk" → Sink to Pub/Sub → Splunk

6. Log Sinks & Routing - Critical for Exam!

Sink Basics

****Log Sink** = Filter + Destination**

Source: Cloud Logging



Filter: Log query (include/exclude)



Destination: GCS / BigQuery / Pub/Sub / Another project

Sink Destinations

Destination	Use Case	Retention Control	Cost
Cloud Storage	Long-term archive	Lifecycle policy	\$ (cheap)
BigQuery	Analysis, SQL queries	Table expiration	\$\$
Pub/Sub	Real-time streaming	N/A	\$ (+ processing)
Cloud Logging Bucket	Custom retention in Logging	1-3650 days	\$\$\$
Another Project	Centralized logging	Varies	Varies

Sink Scope Levels

Organization Sink: Exports logs from ALL projects/folders

Folder Sink: Exports logs from folder and child projects

Project Sink: Exports logs from single project only

****Exam Pattern**:** "Centralize logs from 100 projects" → Organization-level sink

Sink Filters (Super Important!)

****Common Filter Patterns**:**

All logs:

"" (empty string = all logs)

Specific log type:

logName:"logs/cloudaudit.googleapis.com%2Factivity"

IAM changes only:

protoPayload.methodName="SetiamPolicy"

Specific user:

protoPayload.authenticationInfo.principalEmail="alice@example.com"

Errors only:

severity >= ERROR

Exclude specific service:

NOT resource.type="gce_instance"

Combined:

protoPayload.methodName="SetiamPolicy"

AND severity="NOTICE"

Exclusion Filters

****Purpose**:** Reduce logging costs by excluding logs from ingestion

Example: Exclude GCS reads by monitoring system

NOT (resource.type="gcs_bucket")

AND

protoPayload.authenticationInfo.principalEmail="monitoring@project.iam.gserviceaccount.com"

AND protoPayload.methodName="storage.objects.get")

****Exam**:** "Reduce logging costs" → Use exclusion filters for noisy logs

7. Querying & Filtering Logs

Query Structure

FIELD OPERATOR VALUE

Operators:

- = : Exact match
- != : Not equal
- > : Greater than
- < : Less than
- >= : Greater or equal
- <= : Less or equal
- : : Contains (substring)
- =~ : Regex match

...

Critical Query Patterns (Memorize These!)

Find who deleted a VM:

```
...
resource.type="gce_instance"
protoPayload.methodName="v1.compute.instances.delete"
timestamp>="2025-12-01T00:00:00Z"
...
```

Find all IAM policy changes:

```
...
protoPayload.methodName="SetIamPolicy"
...
```

Find permission denied errors:

```
...
protoPayload.status.code=7
...
```

Find who accessed specific BigQuery table:

```
...
resource.type="bigquery_dataset"
protoPayload.resourceName:"tables/sensitive_data"
protoPayload.methodName="tabledata.list"
...
```

Find all Owner role grants:

```
...
protoPayload.methodName="SetIamPolicy"
protoPayload.serviceData.policyDelta.bindingDeltas.role="roles/owner"
protoPayload.serviceData.policyDelta.bindingDeltas.action="ADD"
...
```

Find access from specific IP:

```
...
protoPayload.requestMetadata.callerIp="203.0.113.45"
```

Find VPC Service Controls violations:

protoPayload.metadata.vpcServiceControlsUniqueId!=""

severity=ERROR

Find service account key creations:

protoPayload.methodName="google.iam.admin.v1.CreateServiceAccountKey"

Find failed login attempts (if using Cloud Identity):

protoPayload.methodName="google.login.LoginService.loginFailure"

8. Audit Logs for Specific Services

Cloud Storage (GCS)

Admin Activity:

- `storage.buckets.create`
- `storage.buckets.delete`
- `storage.buckets.update`
- `storage.buckets.setIamPolicy`

Data Access (enable for compliance):

- `storage.objects.create` (upload)
- `storage.objects.get` (download)
- `storage.objects.list` (list objects)
- `storage.objects.delete` (delete object)

Exam: "Track who downloaded file" → Enable Data Access READ

BigQuery

Admin Activity:

- `google.cloud.bigquery.v2.DatasetService.InsertDataset`
- `google.cloud.bigquery.v2.TableService.DeleteTable`
- `google.cloud.bigquery.v2.DatasetService.PatchDataset`

Data Access (READ enabled by default!):

- `google.cloud.bigquery.v2.JobService.InsertJob` (queries)
- `google.cloud.bigquery.v2.TableService.GetTable`
- `tabledata.list` (browse data)

****Query Details**:** Found in
`protoPayload.serviceData.jobCompletedEvent.job.jobConfiguration.query.query`

****Exam**:** "See which user ran what query" → Check Data Access logs, look in jobConfiguration

Compute Engine

****Admin Activity**:**

- `v1.compute.instances.insert`
- `v1.compute.instances.delete`
- `v1.compute.instances.start`
- `v1.compute.instances.stop`
- `beta.compute.firewalls.patch`

****Data Access**:**

- `v1.compute.instances.list`
- `v1.compute.instances.get`

****Exam**:** SSH access to VMs NOT logged (it's not an API call)

IAM

****Admin Activity** (critical for security):**

- `SetIamPolicy` - IAM policy changes
- `google.iam.admin.v1.CreateServiceAccount`
- `google.iam.admin.v1.DeleteServiceAccount`
- `google.iam.admin.v1.CreateServiceAccountKey` 

****Exam**:** "Alert on new service account keys" → Filter on `CreateServiceAccountKey`

Cloud SQL

****Admin Activity**:**

- `cloudsql.instances.create`
- `cloudsql.instances.delete`
- `cloudsql.users.create`
- `cloudsql.instances.update`

****Data Access**** (enable for compliance):

- Connection logs
- Query logs (via database flags, not audit logs)

****Exam**:** SQL queries NOT in audit logs, use database logs instead

Kubernetes Engine (GKE)

****Admin Activity**:**

- `io.k8s.core.v1.pods.create`
- `io.k8s.core.v1.secrets.get`
- `io.k8s.apps.v1.deployments.create`

****Important**:** GKE audit logs separate from Cloud Audit Logs

- Configure in GKE cluster settings
- Can be expensive (lots of API calls)

****Exam**:** "Audit Kubernetes API calls" → Enable GKE audit logging

9. Compliance & Security Monitoring

Security Monitoring Patterns

****Pattern 1: Privilege Escalation Detection****

...

Filter:

```
protoPayload.methodName="SetIamPolicy"  
protoPayload.serviceData.policyDelta.bindingDeltas.role=("roles/owner" OR "roles/editor"  
OR "roles/iam.securityAdmin")  
protoPayload.serviceData.policyDelta.bindingDeltas.action="ADD"
```

Alert: Immediate notification to security team

...

****Pattern 2: Service Account Key Creation****

...

Filter:

```
protoPayload.methodName="google.iam.admin.v1.CreateServiceAccountKey"
```

Alert: Security review required

...

****Pattern 3: Data Exfiltration Attempt****

...

Filter:

```
resource.type="gcs_bucket"
protoPayload.methodName="storage.objects.get"
protoPayload.requestMetadata.callerIp NOT IN ["CORPORATE_IP_RANGE"]
protoPayload.resourceName:"sensitive"
```

Alert: Potential data exfiltration

...

****Pattern 4: After-Hours Activity****

...

Filter:

```
timestamp.hour < 6 OR timestamp.hour > 18
protoPayload.methodName="SetIamPolicy"
```

Alert: Unusual timing for admin activity

...

****Pattern 5: Failed Access Attempts****

...

Filter:

```
protoPayload.status.code=7
protoPayload.authenticationInfo.principalEmail="specific-user@example.com"
```

Alert: Multiple failed attempts = possible compromised account

...

Compliance Requirements

****PCI-DSS**:**

...

Required Logs:

- All access to cardholder data (Data Access logs)
- All actions by privileged users (Admin Activity)
- All authentication attempts (Policy Denied)
- All IAM changes (Admin Activity - SetIamPolicy)

Retention: 1 year online, 3 years archive minimum

Solution: Sink to GCS with retention policy

...

****HIPAA**:**

...

Required Logs:

- All PHI access (Data Access logs)
- All configuration changes (Admin Activity)
- Failed access attempts (Policy Denied)

User authentication events

Retention: 6 years minimum

Solution: Sink to GCS Coldline/Archive storage

+ Immutable retention policy (Bucket Lock)

****SOC 2**:**

Required Logs:

- All system access
- All privileged operations
- All security-relevant events
- Log integrity (tamper-proof)

Solution: Sink to immutable GCS bucket

+ Access Transparency for Google personnel access

****GDPR**:**

Required Logs:

- All personal data access
- Data subject actions tracking
- Purpose of processing

Retention: As long as data exists, then delete

Solution: Lifecycle policies aligned with data retention

10. Cost Optimization

Cost Factors

Logging Costs:

1. Ingestion: \$0.50 per GB (first 50 GB/project/month free)
2. Storage: \$0.01 per GB per month
3. Queries: Usually minimal

Data Access Logs = Biggest Cost Driver

Cost Reduction Strategies

****Strategy 1: Selective Data Access Logging****

Don't enable for ALL services

Enable only for:

- Services with sensitive data (BigQuery, GCS)
- Services with compliance requirements
- Services with security concerns

Skip:

- High-volume, low-value logs (health checks)
- ...

****Strategy 2: Exemptions****

...

Exempt trusted service accounts:

- Monitoring systems
- Internal automation
- Known safe applications

Example:

Service: storage.googleapis.com

Log: DATA_READ

Exempt: monitoring-sa@project.iam.gserviceaccount.com

...

****Strategy 3: Exclusion Filters****

...

Exclude noisy, low-value logs:

```
resource.type="gcs_bucket"
protoPayload.methodName="storage.objects.get"
protoPayload.authenticationInfo.principalEmail="health-check@system.iam.gserviceaccount.com"
...
```

****Strategy 4: Sample High-Volume Logs****

...

Keep every 10th log entry:

```
sample(insertId, 0.1)
```

Use for: Very high volume, non-critical logs

Don't use for: Compliance-required logs

...

****Strategy 5: Sink to Cheaper Storage****

...

Don't keep everything in Cloud Logging (expensive)

Sink to:

- Cloud Storage (90% cheaper for long-term)
- BigQuery (query when needed, cheaper storage)

Keep in Cloud Logging:

- Recent logs only (30 days)
- High-value logs only
-

Strategy 6: Use Log Buckets with Retention

Create custom log buckets:

- Set specific retention (1-3650 days)
- Different retention for different log types
- Cheaper than default _Default bucket

Example:

- Admin Activity bucket: 400 days
- Data Access bucket: 90 days
- Less critical logs: 30 days
-

Cost Optimization Checklist:

- Enable Data Access logs only where needed
- Use exemptions for trusted service accounts
- Apply exclusion filters for noisy logs
- Sink to GCS for long-term retention
- Use custom log buckets with appropriate retention
- Monitor logging costs in Cloud Billing
- Regular review of what's being logged
-

Exam Pattern: "Reduce logging costs" → Exclusion filters + exemptions + sink to GCS

11. Integration with Other Services

Cloud Monitoring Alerts

Log-Based Metrics → **Alerting Policies**

Example: Alert on IAM Policy Changes

1. Create Log-Based Metric:

Name: iam_policy_changes

Filter: protoPayload.methodName="SetIamPolicy"

Metric Type: Counter

2. Create Alert Policy:

Condition: iam_policy_changes > 0

Notification: Email security team

Duration: 0 (immediate)

Common Alert Patterns:

1. Owner role granted:

Filter: SetIamPolicy + roles/owner + action=ADD

2. SA key created:

Filter: CreateServiceAccountKey

3. Org policy changed:

Filter: SetOrgPolicy

4. VPC-SC violation:

Filter: vpcServiceControlsUniqueId!="" AND severity=ERROR

5. Failed authentication spikes:

Filter: status.code=7

Condition: Rate > 10 per minute

Security Command Center (SCC)

Integration:

- SCC ingests audit logs automatically
- Creates security findings
- Detects anomalies

Findings from Audit Logs:

- Publicly accessible resources
- Overly permissive IAM
- Service account key usage
- Anomalous API activity
- Privilege escalation attempts

Exam: "Detect security issues from audit logs" → Security Command Center

Cloud Asset Inventory

****Integration**:**

- Tracks resource changes over time
- Uses audit logs as source
- Point-in-time snapshots

****Use Cases**:**

- "What resources existed on Dec 1?"
- "When was this VM created?"
- "Who changed IAM policy yesterday?"

SIEM Integration (Splunk, Datadog, etc.)

****Pattern**:** Pub/Sub Sink → SIEM

1. Create Pub/Sub topic
2. Create log sink to Pub/Sub
3. SIEM subscribes to topic
4. Real-time log forwarding

Benefits:

- Real-time analysis
- Correlation with other logs
- Advanced threat detection

****Exam**:** "Send logs to Splunk" → Pub/Sub sink

BigQuery for Log Analysis

****Pattern**:** BigQuery Sink → SQL Analysis

Common Queries:

-- Top 10 users by API calls

SELECT

```
protoPayload.authenticationInfo.principalEmail,  
COUNT(*) as api_calls
```

FROM `project.dataset.cloudaudit_googleapis_com_activity_*`

WHERE timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7 DAY)

GROUP BY 1

ORDER BY 2 DESC

LIMIT 10;

```
-- IAM changes by user
SELECT
    timestamp,
    protoPayload.authenticationInfo.principalEmail,
    protoPayload.resourceName
FROM `project.dataset.cloudaudit_googleapis_com_activity_*`
WHERE protoPayload.methodName = "SetIamPolicy"
ORDER BY timestamp DESC;

-- Failed API calls
SELECT
    timestamp,
    protoPayload.authenticationInfo.principalEmail,
    protoPayload.methodName,
    protoPayload.status.message
FROM `project.dataset.cloudaudit_googleapis_com_activity_*`
WHERE protoPayload.status.code != 0
ORDER BY timestamp DESC;
```

Terraform/IaC

Audit Log Configuration as Code:

```
hcl
resource "google_project_iam_audit_config" "default" {
    project = var.project_id
    service = "allServices"

    audit_log_config {
        log_type = "ADMIN_READ"
    }

    audit_log_config {
        log_type = "DATA_READ"
        exempted_members = [
            "serviceAccount:monitoring@${var.project_id}.iam.gserviceaccount.com"
        ]
    }

    audit_log_config {
        log_type = "DATA_WRITE"
    }
}

resource "google_logging_project_sink" "gcs_sink" {
```

```

name      = "audit-logs-archive"
destination = "storage.googleapis.com/${google_storage_bucket.logs.name}"
filter     = "logName:claudiaudit.googleapis.com"

unique_writer_identity = true
}

resource "google_storage_bucket_iam_member" "sink_writer" {
  bucket = google_storage_bucket.logs.name
  role   = "roles/storage.objectCreator"
  member = google_logging_project_sink.gcs_sink.writer_identity
}
...
---
```

12. Best Practices & Common Patterns

Essential Best Practices

****1. Enable Data Access Logs for Sensitive Data****

...

- BigQuery datasets with PII/PHI
- Cloud Storage buckets with sensitive files
- Secret Manager secrets
- Cloud SQL databases with customer data

✗ Don't enable for:

- Test/dev projects (unless required)
- Public datasets
- Non-sensitive data

...

****2. Centralize Logs****

...

Pattern: Organization-level sink to centralized project

Benefits:

- Single pane of glass
- Easier compliance audits
- Better security monitoring
- Cost optimization

Setup:

Organization → Sink → Dedicated logging project

...

****3. Immutable Long-Term Storage****

...

For compliance:

1. Sink to Cloud Storage
2. Enable Object Versioning
3. Set Retention Policy (Bucket Lock)
4. Use Coldline/Archive storage class

Result: Tamper-proof, cost-effective, long-term retention

...

****4. Separate Storage by Sensitivity****

...

Different sinks for different log types:

Sink 1: Admin Activity → GCS bucket (400 days)

Sink 2: Data Access → GCS bucket (90 days) + lifecycle to Coldline

Sink 3: Security logs → BigQuery (analysis) + GCS (archive)

Benefit: Optimized costs and retention

...

****5. Use Log-Based Metrics for Alerting****

...

Don't query logs repeatedly (expensive)

Instead:

1. Create log-based metric (one-time cost)
2. Alert on metric (free)

Example:

Metric: iam_owner_grants

Filter: SetIamPolicy + roles/owner + ADD

Alert: When count > 0

...

****6. Implement Least Privilege for Logs****

...

Roles:

- logging.viewer: Read logs
- logging.privateLogViewer: Read Data Access logs
- logging.admin: Manage logging configuration

Don't grant:

- Project Owner/Editor (too broad)
- logging.admin (unless needed)

Do grant:

- logging.privateLogViewer to security team
- logging.viewer to developers (filtered logs)

7. Regular Audit Log Reviews

Weekly:

- Review security alerts
- Check for privilege escalation
- Verify expected activity patterns

Monthly:

- Analyze cost trends
- Review exemptions (still valid?)
- Check sink configurations

Quarterly:

- Full security audit
- Compliance review
- Update alert policies

13. Troubleshooting & Common Issues

Issue 1: "Logs Not Appearing"

Checklist:

- Is logging enabled for this log type?
 - Admin Activity: Always on
 - Data Access: Check configuration
- Correct permissions?
 - Need logging.viewer or privateLogViewer
- Correct time range?
 - Default view: Last 1 hour
 - Try expanding time range
- Correct project/resource?
 - Verify project ID
 - Check resource filter
- Propagation delay?
 - Wait 2-3 minutes
 - Refresh view

****Solution Pattern**:**

1. Check audit log configuration (IAM & Admin → Audit Logs)
 2. Verify IAM permissions
 3. Expand time range to 24 hours
 4. Remove all filters, then add back one by one
 5. Check if logs in sink destination (if configured)
-

Issue 2: "Too Many Logs / High Costs"

****Troubleshooting**:**

1. Identify high-volume sources:
 - Go to Logs Explorer
 - Group by resource.type
 - Sort by count
 2. Check top service accounts:
`protoPayload.authenticationInfo.principalEmail`
 3. Check top methods:
`protoPayload.methodName`
 4. Check Data Access configuration:
 - Which services enabled?
 - Any exemptions needed?
-

****Solution Pattern**:**

1. Add exemptions for monitoring SAs
 2. Apply exclusion filters for noisy logs
 3. Disable Data Access for non-critical services
 4. Sample high-volume logs (if appropriate)
 5. Sink to cheaper storage (GCS)
 6. Use custom log buckets with shorter retention
-

Issue 3: "Can't Find Specific Event"

****Troubleshooting Steps**:**

1. Verify event generates audit log:

- SSH to VM: NOT logged (not API call)
- SQL query: Database logs, not audit logs
- Application errors: Not audit logs

2. Check correct log type:

- Configuration change: Admin Activity
- Data read/write: Data Access
- Failed access: Policy Denied (if enabled)

3. Correct timestamp:

- Check timezone (logs in UTC)
- Expand time range

4. Check if log excluded:

- Review exclusion filters
- Check sink filters

...

Issue 4: "Sink Not Working"

****Checklist**:**

...

- Sink writer has permissions?
 - GCS: roles/storage.objectCreator
 - BigQuery: roles/bigquery.dataEditor
 - Pub/Sub: roles/pubsub.publisher
- Destination exists and accessible?
 - Bucket created
 - Dataset created
 - Topic created
- Filter correct?
 - Test filter in Logs Explorer first
 - Verify logs match filter
- Include children (if org/folder sink)?
 - Check "Include children" option

...

****Solution Pattern**:**

...

1. Get sink writer identity:

```
gcloud logging sinks describe SINK_NAME
```

2. Grant permissions manually:

```
gsutil iam ch [WRITER_IDENTITY]:objectCreator gs://BUCKET
```

3. Test filter in Logs Explorer before creating sink

4. Monitor sink errors:

Check for sink error logs

...

Issue 5: "BigQuery Sink Table Issues"

****Common Problems**:**

...

1. Schema mismatch:

- Audit log schema can change
- BigQuery auto-updates schema
- Old queries may break

2. Table size growing fast:

- Partition tables by day
- Set partition expiration
- Use clustering (resource.type, timestamp)

3. Query performance slow:

- Use partitioned tables (_PARTITIONTIME)
- Filter on partition column first
- Use clustering for common filters

...

****Solution**:**

...

```
-- Create partitioned table
```

```
CREATE TABLE `project.dataset.cloudaudit_logs`  
PARTITION BY DATE(timestamp)  
CLUSTER BY resource.type, protoPayload.methodName  
OPTIONS(  
    partition_expiration_days=90  
) AS  
SELECT * FROM `project.dataset._AllLogs`  
WHERE logName LIKE '%cloudaudit%';
```

```
-- Query with partition filter (fast)
```

```
SELECT *  
FROM `project.dataset.cloudaudit_logs`  
WHERE DATE(timestamp) = '2025-12-09'  
    AND resource.type = 'gce_instance';
```

...

14. Exam Scenarios & Quick Answers

Scenario Decision Matrix

Requirement Solution
----- -----
Retain logs 7+ years Sink to GCS with retention policy
Analyze logs with SQL Sink to BigQuery
Real-time SIEM integration Sink to Pub/Sub
Track who deleted resource Admin Activity logs
Track who accessed data Data Access logs (enable first)
Track failed access attempts Policy Denied logs (enable first)
Reduce logging costs Exclusion filters + exemptions + selective Data Access
Centralize 100 projects Organization-level sink
Compliance archive GCS + Bucket Lock + Coldline/Archive
Alert on privilege escalation Log-based metric + alert on SetIamPolicy + Owner role
Debug permission issue Policy Denied logs + Policy Troubleshooter
Track BigQuery queries Data Access logs (already enabled for BQ)
Track SSH access NOT in audit logs (not API call)
Track SQL queries Database logs (not audit logs)
Immutable logs GCS with retention policy + Bucket Lock

Exam Keyword Triggers

Keyword Think
----- -----
"Who deleted" Admin Activity logs
"Who accessed" Data Access logs
"Failed attempts" Policy Denied logs
"7 years retention" Sink to GCS
"Analyze with SQL" Sink to BigQuery
"Real-time monitoring" Sink to Pub/Sub
"HIPAA" / "PCI-DSS" Data Access logs + long retention
"Reduce costs" Exclusion filters, exemptions
"Centralize logs" Org-level sink
"Tamper-proof" GCS Bucket Lock
"Track IAM changes" SetIamPolicy method
"Track SA keys" CreateServiceAccountKey method
"VPC-SC violations" vpcServiceControlsUniqued field
"Cannot find logs" Check if Data Access/Policy Denied enabled
"Too expensive" Disable Data Access or add exclusions

Quick Reference - Common Filters

...

IAM Changes:

protoPayload.methodName="SetIamPolicy"

Owner Role Grants:

protoPayload.methodName="SetIamPolicy"

protoPayload.serviceData.policyDelta.bindingDeltas.role="roles/owner"

protoPayload.serviceData.policyDelta.bindingDeltas.action="ADD"

SA Key Creation:

protoPayload.methodName="google.iam.admin.v1.CreateServiceAccountKey"

VM Deletions:

protoPayload.methodName="v1.compute.instances.delete"

Permission Denied:

protoPayload.status.code=7

BigQuery Queries:

resource.type="bigquery_dataset"

protoPayload.methodName="jobservice.insert"

GCS Downloads:

resource.type="gcs_bucket"

protoPayload.methodName="storage.objects.get"

Failed Logins:

protoPayload.status.code!=0

protoPayload.methodName:"login"

Specific User Activity:

protoPayload.authenticationInfo.principalEmail="user@example.com"

External IP Access:

protoPayload.requestMetadata.callerIp NOT: ("10.0.0.0/8" OR "172.16.0.0/12")

After Hours Activity:

timestamp.hour<6 OR timestamp.hour>18

Critical "Must Know" List

The 15 Essentials:

1. Admin Activity = Always On, Free, 400 days

2. **Data Access = Off by default (except BigQuery READ), Paid, 30 days**
 3. **Cannot change retention in Cloud Logging directly**
 4. **Extend retention: Sink to GCS/BigQuery**
 5. **methodName = Which API method was called**
 6. **principalEmail = Who made the call**
 7. **status.code=7 = Permission Denied**
 8. **Centralize: Organization-level sink**
 9. **Compliance: Sink to GCS + Bucket Lock**
 10. **Cost reduction: Exclusion filters + exemptions**
 11. **SetIamPolicy = IAM policy changes**
 12. **CreateServiceAccountKey = SA key creation**
 13. **BigQuery queries: Check jobConfiguration.query**
 14. **VPC-SC violations: vpcServiceControlsUniqueId field**
 15. **SIEM integration: Pub/Sub sink**
-

Final Exam Tips

When You See:

- "Track configuration changes" → Admin Activity (always on)
- "Track data access" → Data Access logs (enable first)
- "Long-term retention" → Sink to GCS
- "SQL analysis" → Sink to BigQuery
- "Real-time alerts" → Log-based metrics + Cloud Monitoring
- "Expensive logs" → Exclusion filters, exemptions, selective enablement
- "Compliance" → Data Access logs + GCS sink + retention policy
- "Centralized logging" → Org-level sink
- "Cannot find logs" → Check if log type enabled (Data Access / Policy Denied)
- "Debug permissions" → Policy Denied logs

Remember:

- Admin Activity captures "mutations" (create, delete, update)
- Data Access captures "reads" and "writes"
- Not everything generates audit logs (SSH, SQL queries)
- Cost control is key for Data Access logs
- Sinks are critical for retention beyond defaults

The Hybrid Cloud Problem

The "Why": The Hybrid Cloud Problem

Imagine you are a large company (like a bank or retailer). You have been running your own data centers (On-Premises) for 20 years. You are now moving some applications to Google Cloud (GCP).

You now have a "Hybrid" environment:

1. **On-Prem:** Your legacy database and Active Directory servers live here. They use a domain name like `corp.local`.
2. **GCP:** Your new web applications live here. They use GCP internal names like `app-server.c.my-project.internal`.

The Problem:

- When your new web app in GCP tries to connect to the legacy database (`db.corp.local`), it asks Google's DNS server. Google's server says: "*I don't know what `corp.local` is. That's not on the public internet.*"
- When your admin sitting in the office tries to SSH into the new GCP app (`app.internal`), their office computer asks the on-prem DNS server. The on-prem server says: "*I don't know what `.internal` is. That's not in my records.*"

The Solution:

We need a translator. We need to tell Google to "forward" questions about `corp.local` to the office, and tell the office to "forward" questions about `.internal` to Google.

The "What": The Two Flows

In the exam and in the real world, "DNS Forwarding" refers to setting up this two-way communication channel.

1. Outbound Forwarding (GCP \Rightarrow On-Prem)

- **The User:** A Virtual Machine (VM) inside Google Cloud.
- **The Goal:** It needs to talk to a database sitting in your office basement.
- **The Action:** You configure Cloud DNS to say: "*If anyone asks for `*.corp.local`, don't look on the internet. Send that question through the VPN tunnel to the DNS server at IP `10.0.0.5`.*"
- **Why use it?** So your cloud apps can consume legacy data without migrating the database yet.

2. Inbound Forwarding (On-Prem \Rightarrow GCP)

- **The User:** An employee sitting in the corporate office.
- **The Goal:** They need to open a dashboard hosted on a private Internal IP in GCP.
- **The Action:** You create a "listener" (Inbound Policy) in GCP. It gives you an IP address (like `10.128.0.2`). You then tell your office DNS server: "*If anyone asks for `*.internal`, forward that question to `10.128.0.2`.*"
- **Why use it?** So developers and admins can manage cloud resources using friendly names instead of memorizing IP addresses.

Summary for the Exam

Whenever you see a question about "**Hybrid Connectivity**" or "**Resolution across environments**," you are strictly dealing with these two concepts. The exam tests if you know *which direction* the traffic is going and *what components* make that direction work.

Revised Deep Dive

Now that we have established *why* we are doing this, let's revisit the **Core Concepts** and **Exam Scenarios** with this context in mind.

Part 1: The Core Concepts (The "Must-Knows")

1. **Inbound Forwarding (On-Prem ==> GCP):**
 - **Mechanism:** You create an **Inbound Server Policy**.
 - **IP Address:** This creates a specific internal IP (Entry Point) in your VPC subnets (usually the .2 address).
 - **The "Hidden" Range:** Cloud DNS proxies queries from On-Prem to GCP using the IP range **35.199.192.0/19**. You *must* allow this range on your on-premise firewall for ingress and ensure your on-prem router knows to route this range back to the VPN/Interconnect.
2. **Outbound Forwarding (GCP ==> On-Prem):**
 - **Mechanism:** You create a **Forwarding Zone** (e.g., **corp.local** points to **192.168.1.5**).
 - **Alternative:** You can use an **Outbound Server Policy** to override *all* resolutions (replace the metadata server entirely).
3. **Resolution Order (The "Tie-Breaker"):**
 - GCP resolves roughly in this order:
 1. Metadata Server policies (e.g., **/etc/hosts** on Linux).
 2. Cloud DNS Private Zones (Longest Suffix Match wins).
 3. Forwarding Zones.
 4. Peering Zones.
 5. Public Internet (unless restricted).
4. **Transitivity:**
 - **Cloud DNS Peering is NOT transitive.**
 - If VPC A peers DNS to VPC B, and VPC B peers DNS to VPC C, A *cannot* resolve names in C. A must peer directly to C.

Part 2: Deep Dive Exam Scenarios

Here are 5 difficult scenarios likely to appear on the exam, with the "Deep Dive" logic explaining why the answer is what it is.

Scenario 1: The "Alternative Name Server" Trap

Question: Your company requires all DNS traffic from GCP VMs to be filtered through an on-premise security appliance for auditing. You configure an **Outbound Server Policy** to point all DNS queries to your on-premise DNS server (**10.0.0.5**). Immediately after applying

this, your applications in GCP fail to resolve internal GCP resources (like `backend.internal`). Why?

- A. You forgot to open firewall port 53 on the on-premise router.
- B. Outbound Server Policies disable the local Metadata server resolution for internal zones.
- C. You need to create a peering zone between the VPC and the on-premise network.
- D. The `35.199.192.0/19` range is blocked.

Correct Answer: B

Deep Dive: This is a classic exam trap. When you configure an Outbound Server Policy (Alternative Name Server), you are telling the VM to stop using the GCP metadata server (169.254.169.254) for resolution and use only your custom server.⁴ Consequently, the VM loses the ability to resolve standard GCP internal names (like `.internal` or Private Service Connect endpoints) unless your on-prem server is configured to forward those specific queries back to GCP (Inbound Forwarding).⁵

Scenario 2: The "Split Horizon" Conflict

Question: You have a public zone `example.com` hosted on Cloud DNS. You also have an internal application that uses `app.example.com` hosted on a **Private Zone** in the same VPC. Users report they can access `app.example.com`, but they can no longer access the public website `www.example.com`. What is the cause?

- A. Private Zones cannot use the same domain name as Public Zones.
- B. The Private Zone takes precedence over the Public Zone for the *entire* domain.
- C. You need to create a CNAME record in the Private Zone for `www`.
- D. You must enable DNSSEC on the Private Zone.

Correct Answer: B (and C is the fix)

Deep Dive: Cloud DNS uses a "Split Horizon" approach. If a VPC is authorized to see a Private Zone (e.g., `example.com`), the Metadata server will only query that Private Zone for anything ending in `example.com`. It will not fall back to the public internet if the record is missing in the private zone.

- *The Fix:* You must copy the public records (like `www`) into the Private Zone or create a separate private zone for the specific subdomain `app.example.com` so the parent `example.com` continues to resolve publicly.

Scenario 3: Troubleshooting Connectivity (The 35.199... Range)

Question: You have configured a Cloud DNS **Forwarding Zone** to resolve `onprem.local` using your on-site DNS servers over Cloud VPN. The VPN is up, and firewalls allow traffic between the VM subnets and the on-premise DNS IP. However, `dig onprem.local` from a GCP VM times out. What is the most likely missing configuration?

- A. The GCP VMs do not have External IP addresses.

- **B.** You did not create an Inbound Server Policy.
- **C.** The on-premise firewall is blocking ingress from 35.199.192.0/19.
- **D.** You need to enable Private Google Access.

Correct Answer: C

Deep Dive: When GCP forwards a DNS query to on-prem, the source IP of the packet is NOT the VM's internal IP. It is an IP from the 35.199.192.0/19 proxy range. If your on-prem firewall only allows traffic from your VPC subnet (e.g., 10.10.0.0/24), it will drop the DNS packets coming from 35.199.x.x.

Scenario 4: Shared VPC and Service Projects

Question: You are using a Shared VPC architecture. The Host Project contains the VPC and VPN to on-prem.⁷ A Service Project contains a GKE cluster. You want the GKE pods in the Service Project to resolve on-premise hostnames. Where should you configure the Cloud DNS Forwarding Zone?

- **A.** In the Service Project.
- **B.** In the Host Project.
- **C.** In both projects.
- **D.** In the Service Project, peered to the Host Project.

Correct Answer: B (Usually)

Deep Dive: While you can create private zones in Service Projects, Forwarding Zones require network connectivity to the target.⁸ In a Shared VPC, the network interfaces live in the Host Project.⁹ The most streamlined architecture is to create the Forwarding Zone in the Host Project and ensure the Service Project is "authorized" (network binding) to use the Host Project's network.

- *Note:* If you create the zone in the Service Project, you must ensure the visibility is set to the Shared VPC Network (which physically resides in the Host Project). The exam often prefers centralizing network resources (DNS) in the Host Project.

Scenario 5: Transitive Peering Limitations

Question:

- **VPC A** (Hub) is connected to On-Prem via VPN.
- **VPC B** (Spoke) is peered to VPC A via VPC Network Peering.
- You want VMs in VPC B to resolve on-premise hostnames.
- You configured a Forwarding Zone in VPC A.
- VMs in VPC A can resolve on-prem names. VMs in VPC B cannot.

What is the required step to fix this?

- **A.** Configure a Forwarding Zone in VPC B pointing to VPC A's Inbound Policy IP.
- **B.** Configure a Cloud DNS Peering Zone in VPC B targeting VPC A.

- C. Enable "Export Custom Routes" on the VPC Peering connection.
- D. Add the [35.199.192.0/19](#) range to VPC B's firewall.

Correct Answer: B

Deep Dive: VPC Network Peering shares routes, but it does not automatically share Cloud DNS configurations. Just because B can ping A, doesn't mean B uses A's DNS rules. You must explicitly create a DNS Peering Zone in VPC B that targets VPC A. This tells VPC B: "Send all DNS queries for this domain to the Cloud DNS engine of VPC A."

Part 3: Summary Cheat Sheet for the Exam

Requirement	Solution	Key Constraint
GCP ==> On-Prem	Forwarding Zone	Requires VPN/Interconnect + Firewall allow for 35.199.192.0/19 .
On-Prem ==> GCP	Inbound Server Policy	Creates an IP in the VPC. On-prem must forward queries to this IP.
Override ALL DNS	Outbound Server Policy	WARNING: Breaks internal GCP resolution (metadata server) unless manually handled.
VPC A ==> VPC B	DNS Peering Zone	Only works 1 hop. Not transitive.
GCP ==> Public	Public Zone	If a Private Zone exists with the same name, the Private Zone wins (Split Horizon).

Cloud SQL Auth Proxy

What Cloud SQL Auth Proxy Actually Does

The Cloud SQL Auth Proxy handles authentication + encryption + IAM for Cloud SQL connections.

It solves these problems:

- ✓ 1. Automatic IAM authentication

It allows you to connect to Cloud SQL using your Google account instead of storing DB passwords.

- ✓ 2. Secure TLS encryption by default

No need to manage TLS certs.

- ✓ 3. Simplifies connecting from ANY environment

- Your laptop
- A VM
- A GKE pod
- A Cloud Run service
- A Bastion host

- ✓ 4. Integrates with Private IP or Public IP

Works regardless of IP mode.

“Where is Cloud SQL Auth Proxy used if I am using IAP Tunnel + Bastion?”

→ You can run it either on your laptop OR on the bastion.

MOST SECURE + MODERN:

✓ Run Cloud SQL Auth Proxy on your laptop
(IAP handles private access to Cloud SQL, proxy handles IAM auth)

TRADITIONAL:

✓ Run Cloud SQL Auth Proxy on the bastion, and connect to DB from within the bastion.

Security Advanced

Table of contents (click to jump)

1. Overview & security goals
 2. Reference architecture (diagram + flow)
 3. Networking & boundary controls (private cluster, ingress, Cloud NAT, hybrid connectivity)
 4. Identity & access (Workload Identity, IAM model, RBAC mapping, groups, IAP)
 5. Anthos Service Mesh deep dive (mTLS, AuthorizationPolicy, telemetry)
 6. Workload & node hardening (Shielded nodes, runtime security, Pod security)
 7. Data protection (Cloud SQL/AlloyDB, Secrets, KMS, CMEK)
 8. Software supply chain (Artifact Registry, Binary Authorization, attestations, SBOM)
 9. Edge protection & WAF (Cloud Armor, CDN, rate limits)
 10. Logging, monitoring, detection (SCC, Cloud IDS, logging exports, SIEM integration)
 11. Incident response, forensics, playbooks, automated remediation
 12. Pod egress/internet access & NAT deep dive (how pods reach internet safely)
 13. Advanced policies & enforcement (OrgPolicy, VPC-SC, Policy Controller/Gatekeeper)
 14. Performance, cost, and operational tradeoffs
 15. Practical migration checklist & testing plan
 16. Exam/Interview prep checklist + practice prompt bank
-

1 — Overview & security goals

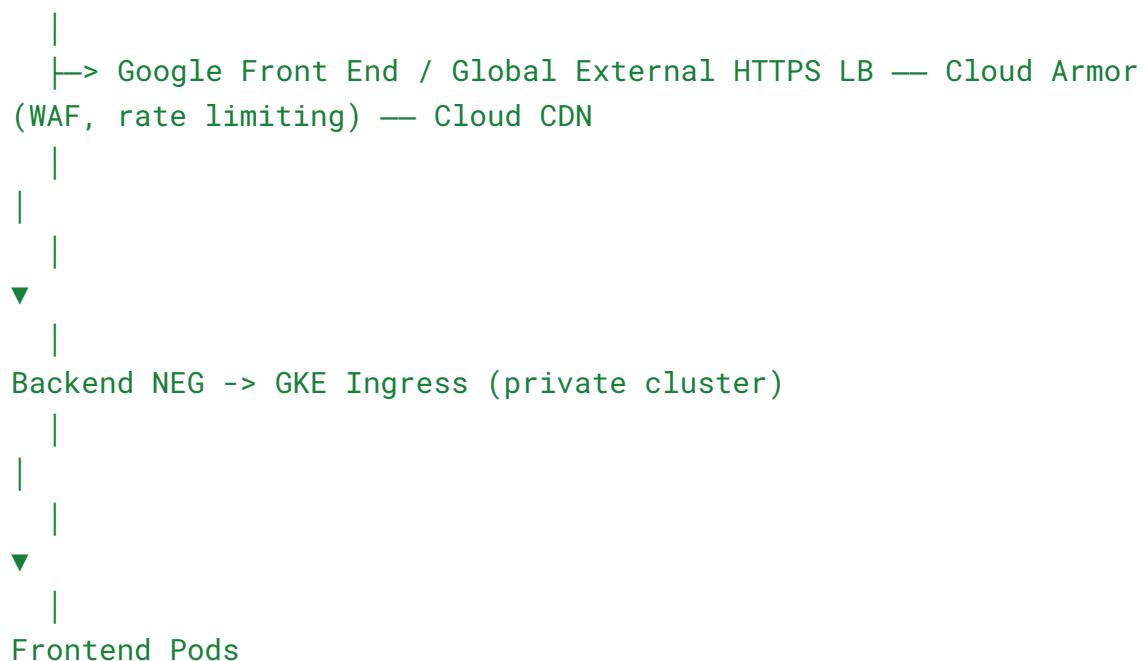
Architecture: Frontend (public-facing) → Backend API → Database (private). All services run in a **GKE private cluster** (no public node IPs). Control plane private endpoint. There is hybrid connectivity (VPN or Interconnect) to on-prem for legacy systems and admin access.

Primary security goals

- Protect public edge (DDoS + WAF) so attacks don't reach VPC
- Ensure zero-trust east-west communications (mTLS + authz)
- Enforce least privilege: no long-lived keys, fine-grained IAM & WIs
- Protect data at rest and in transit; prevent exfiltration
- Secure software supply chain (only signed images)
- Provide robust detection, logging, and automated response
- Allow on-prem (and remote) admin and users to access tools securely (IAP, private connectivity) without VPN where possible

2 — Reference architecture (textual diagram + flow)

Internet





Key datapaths:

- Edge filtering (Cloud Armor) blocks DDoS and OWASP attacks.
- Ingress -> NEG targets GKE pods via NodePort/Service or serverless NEG.
- Egress: Pods → Nodes → Cloud NAT → Internet (outbound only)
- On-prem access: either via Interconnect/VPN to private control plane or via IAP TCP tunnel (zero-trust).

3 — Networking & boundary controls

3.1 Private GKE cluster essentials

- Private cluster: nodes have internal IPs only.
- Private control plane endpoint (optional) — control plane has private IP in VPC or authorized networks.

- Master Authorized Networks (MAN): restrict who can call the control plane (CIDR ranges or IAP).
- Recommended: disable public endpoint entirely; use authorized networks + IAP.

Commands

```
gcloud container clusters create priv-cluster \
--region=us-central1 \
--enable-private-nodes \
--enable-private-endpoint \
--master-authorized-networks=10.0.0.0/8 \
--enable-ip-alias
```

3.2 Hybrid connectivity

- For stable on-prem → VPC: use Dedicated Interconnect (higher throughput) or Cloud VPN for smaller links.
- Use Cloud Router for dynamic BGP routes.
- Ensure BGP communities and route advertisements obey org rules.

Commands (VPN)

```
gcloud compute vpn-tunnels create vpn1 --peer-address=ONPREM_IP \
--ike-version=2 --shared-secret=XXXX \
--router=cloud-router-1 --region=us-central1 \
--local-traffic-selector=0.0.0.0/0
```

3.3 Ingress / Load Balancing

- Use Global External HTTPS Load Balancer (GFE) + Cloud Armor + CDN.
- Backend: NEG (Network Endpoint Group) targeting GKE service. NEG lets LB talk to individual pods (or node ports) and integrate with Cloud CDN.

High level steps

1. Create GKE Service of type [NodePort](#) or [ClusterIP](#) with backend [Ingress](#).

2. Configure GCE Ingress (GKE Ingress v2 supports NEG).
3. Attach Cloud Armor policy to the LB.

Cloud Armor example

```
gcloud compute security-policies create my-waf --description "WAF for frontend"
gcloud compute security-policies rules create 1000 --security-policy my-waf \
--expression "origin.region_code != 'US'" --action "deny(403)"
```

3.4 Egress control & NAT

- Private clusters: to let pods call the Internet (package downloads, external APIs), use Cloud NAT or Private Service Connect.
- Cloud NAT is recommended for controlled outbound connectivity without exposing nodes.

Why Cloud NAT?

- Provides outbound connectivity for resources without external IPs.
- NAT addresses are controlled, logged, and can be filtered at external APIs.

Cloud NAT setup

```
gcloud compute routers create nat-router --network=my-vpc
--region=us-central1
gcloud compute routers nats create nat-config --router=nat-router
--region=us-central1 \
--nat-all-subnet-ip-ranges --enable-logging
```

Pod egress flow

Pod → Node SNAT → Cloud Router → Cloud NAT → Internet

3.5 Network policies

- Use Kubernetes [NetworkPolicy](#) (Calico or VPC-native) to micro-segment.
- Deny all by default and explicitly allow necessary flows.

Example

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-allow-frontend
  namespace: prod
spec:
  podSelector:
    matchLabels:
      app: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
  ports:
  - protocol: TCP
    port: 8080
```

4 — Identity & access (deep)

4.1 Workload Identity (WID) — no keys approach

Pattern:

- Create Google Service Account (GSA).
- Create Kubernetes Service Account (KSA).
- Allow KSA to actAs GSA (IAM binding).
- Pods use KSA; GKE token exchange returns GCP access tokens for GSA.

Commands

```
# create GSA
```

```
gcloud iam service-accounts create my-backend-sa
--project=my-project

# create KSA in namespace
kubectl create serviceaccount ksa-backend --namespace=prod

# allow KSA to impersonate GSA
gcloud iam service-accounts add-iam-policy-binding
my-backend-sa@my-project.iam.gserviceaccount.com \
--member="serviceAccount:my-project.svc.id.goog[prod/ksa-backend]"
--role="roles/iam.workloadIdentityUser"
```

Pod spec

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
  namespace: prod
spec:
  serviceAccountName: ksa-backend
  containers:
  - name: app
    image: gcr.io/my-project/backend:latest
```

Security notes

- Avoid mounting GCP keys.
- Limit GSA IAM roles to least privilege.
- Use separate GSA per microservice.

4.2 Kubernetes RBAC + IAM integration

- Use `gcloud` to map GCP IAM principals to Kubernetes groups (GKE Autoprovisioned Cloud IAM).
- Use `kubectl` with `gcloud auth login` or `gcloud container clusters get-credentials` (tokens expire).

Best practice

- Use Google Workspace groups for role bindings.
- Use `ClusterRole` for cross-namespace admin tasks, but avoid `cluster-admin` for services.

4.3 Identity-Aware Proxy (IAP) for apps and kubectl

IAP uses OAuth2 and enforces policies based on IAM. Use cases:

- Secure internal web UI (e.g., admin portal, Grafana).
- IAP TCP forwarding: allow `kubectl` access over IAP for private clusters.

Enable IAP

- Put app behind external LB.
- Enable IAP on LB resource.
- Grant `roles/iap.tunnelResourceAccessor` or `roles/iap.httpsResourceAccessor` to users/groups.

kubectl over IAP (gcloud as helper)

```
gcloud container clusters get-credentials priv-cluster --zone us-central1
gcloud auth login
# Use gcloud proxy or iap tunneling patterns, or configure 'gcloud iap tcp-tunnel'.
```

Exam tip: When question says "no VPN, remote workforce," IAP is likely part of the solution.

5 — Anthos Service Mesh (ASM) deep dive

ASM (managed Istio) adds security, policy, telemetry.

5.1 mTLS (mutual TLS)

- Automatic between sidecar proxies (Envoy).
- Identity is SPIFFE URI (`spiffe://cluster/ns/sa`).
- Enforce mTLS in [PeerAuthentication](#).

PeerAuthentication example

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: prod
spec:
  mtls:
    mode: STRICT
```

5.2 AuthorizationPolicy (fine-grained)

- Allow only specific principals to call services.

AuthorizationPolicy example

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: backend-policy
  namespace: prod
spec:
  selector:
    matchLabels:
      app: backend
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/prod/sa/ksa-frontend"]
```

5.3 Request authentication (JWT)

Validate JWT tokens at the mesh boundary or on service.

RequestAuthentication example

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt
  namespace: prod
spec:
  selector:
    matchLabels:
      app: frontend
  jwtRules:
  - issuer: "https://accounts.google.com"
    jwksUri: "https://www.googleapis.com/oauth2/v3/certs"
```

5.4 Telemetry & Policy

- Use Cloud Trace, Cloud Monitoring, and Cloud Logging connectors for ASM.
- Capture Envoy access logs and Istio metrics.
- Generate alerts for unexpected service calls.

5.5 Sidecar / egress guard

- Optionally configure egress whitelists (only allow backend to call DB and specific external services). ASM can enforce egress using [ServiceEntry](#).

6 — Workload & node hardening

6.1 Shielded GKE Nodes

- Use secure boot, measured boot, and integrity reporting.
- Enable Shielded GKE nodes on node pools.

Command

```
gcloud container node-pools create pool-1 --cluster=priv-cluster  
--enable-shielded-nodes
```

6.2 Pod Security Admission & PodSecurity Standards

- Use `restricted` policies: disallow privileged containers, host networking, hostPath, etc.
- In GKE >= 1.25 use built-in PSA levels (`privileged`, `baseline`, `restricted`).

Example

```
kubectl apply -f - <<EOF  
apiVersion: policy/v1beta1  
kind: PodSecurityAdmission  
# use namespace labels to enforce  
EOF
```

(Exact YAML varies; exam expects you to know to enable PSA and deny privileged)

6.3 Runtime security (Falco / GKE Sandbox)

- Use GKE Sandbox (gVisor) for untrusted workloads.
- Use Runtime Threat Detection (in Security Command Center) and third-party tools (Falco) for suspicious syscalls.

6.4 Image hardening

- Use minimal base images (distroless / COS).
- Scan images using Container Analysis.
- Enforce Binary Authorization.

7 — Data protection (deep)

7.1 Databases

Cloud SQL / AlloyDB:

- Use Private IP.
- Use Cloud SQL IAM DB Auth (Postgres IAM auth).
- Enforce SSL/TLS between services.
- Use private ALLOW lists for which subnets can connect.

Network path

- Backend Pod → VPC internal IP → Cloud SQL private IP

7.2 Secrets

- Use Secret Manager + Secret Manager CSI driver to mount secrets as files in pods.
- Audit Secret Manager access via Cloud Audit Logs.

Create secret

```
gcloud secrets create db-password --replication-policy="automatic"
echo -n "supersecret" | gcloud secrets versions add db-password
--data-file=-
```

K8s CSI driver sample

```
apiVersion: v1
kind: Pod
metadata: ...
spec:
  volumes:
  - name: secret-volume
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: "gcp-secrets"
```

7.3 Key Management

- Use Cloud KMS for CMEK for storage and database encryption.
- For HSM/BYOK, use Cloud HSM; store key material per compliance (FIPS, HIPAA).
- Audit KMS key access via `cloudkms.googleapis.com` logs.

CMEK sample for GCS bucket

```
gsutil kms encryption -k
projects/my-project/locations/global/keyRings/my-kr/cryptoKeys/my-ke
y gs://my-bucket
```

8 — Software supply chain (very important)

8.1 Artifact Registry & Container Analysis

- Push images to Artifact Registry (private).
- Enable vulnerability scanning (Container Analysis API).

8.2 Binary Authorization

- Create attestors in Binary Authorization, sign images in CI, enforce policy in production clusters.

High-level Binary Authorization flow

1. CI builds image → stores in Artifact Registry.
2. CI signs attestation (e.g., using KMS / Google Cloud Build).
3. Binary Authorization policy enforces signed images for a cluster.

Create policy (concept)

```
gcloud container binauthz policy import --project=my-project
--policy-file=binauthz-policy.yaml
```

8.3 SBOM and provenance

- Record SBOM (software bill of materials).
- Store provenance and attestations in Binary Authorization and Container Analysis.

Exam tip: If Q says "prevent unverified images from running", answer = Binary Authorization.

9 — Edge protection & WAF (Cloud Armor)

9.1 Cloud Armor capabilities

- OWASP CRS built-in rules
- Custom rules (IP ranges, geolocation)
- Rate-limiting / thresholding
- Adaptive Protection (machine learning detection)
- Integration with CDN and LB to keep malicious traffic off the VPC

Sample deny-by-ip rule

```
gcloud compute security-policies rules create 2000 \
--security-policy=my-waf \
--expression="inIpRange(origin.ip, '203.0.113.0/24')" \
--action=deny(403)
```

Rate limit

```
gcloud compute security-policies rules create 3000 \
--security-policy=my-waf \
--expression="true" \
--action="throttle" \
 \
--throttle-config="rateLimitThreshold.count=100,rateLimitThreshold.intervalSec=60"
```

9.2 Integrating with CDN

- Cache static assets with Cloud CDN; configure cache keys and TTL.
 - Bypass Cloud Armor for cache hits — Cloud Armor still inspects at edge for cacheable requests depending on configuration.
-

10 — Logging, monitoring, detection

10.1 Logs & exports

- Send Cloud Audit Logs & VPC Flow Logs to:
 - Logging buckets (for retention)
 - BigQuery (for analytics)
 - Pub/Sub (for near-real-time SIEM ingest)
- Export Cloud Armor logs to Logging and BigQuery for correlation.

Example sink

```
gcloud logging sinks create sink-to-pubsub \
pubsub.googleapis.com/projects/my-project/topics/my-topic \
--log-filter='resource.type="http_load_balancer"'
```

10.2 Security Command Center (SCC)

- SCC Premium can surface misconfigurations, vulnerabilities, and runtime threats.
- Enable Container Threat Detection.

10.3 Cloud IDS

- Deployed in VPC to monitor east-west or north-south traffic patterns (layer 3/4).
- Good complement to Cloud Armor (which is L7/edge).

10.4 SIEM architecture

- Logging sink → Pub/Sub → Dataflow (transform) → SIEM (Splunk/Chronicle/Elastic)
- Enrich logs with asset inventory (instance name, pod name, service account) for correlation.

10.5 Alerts & automation

- Use Cloud Monitoring alerting policies to trigger Cloud Functions or Run for automated mitigations (e.g., add Cloud Armor deny, quarantine GSA).
-

11 — Incident response & forensics

11.1 Preparation

- Runbooks for common incidents: DDoS, data exfil, image compromise.
- Keep playbooks stored in a controlled repo.
- Access revocation plan: how to revoke SA, rotate keys, deploy deny lists quickly.

11.2 Containment steps (example: suspicious outbound exfil)

1. Identify source pod / service account via logs + VPC Flow Logs.
2. Apply NetworkPolicy or delete pod to prevent further exfil.
3. Revoke GSA access with `gcloud iam service-accounts disable`.
4. Add Cloud Armor deny policies if traffic coming from outside.
5. Snapshot nodes/pods for forensics (kubectl exec to dump process lists, capture memory if allowed).

11.3 Forensics artifacts to collect

- Pod logs (stdout/stderr)
- Node / container runtime logs
- VPC Flow Logs

- Container image provenance & attestation
- Binary Authorization attestations
- KMS audit logs
- Cloud Armor logs

11.4 Post-incident

- Root cause analysis
 - Revoke compromised identities
 - Roll forward CI/CD (patched images) and sign
 - Update policies & org constraints
-

12 — Pod egress / how pods connect to the Internet (detailed)

12.1 Default behavior (GKE private nodes)

- Pods have private cluster IPs.
- Nodes may have no external IP.
- Outbound traffic from pods is SNAT'ed (source NAT) at node level, then at Cloud NAT.

Concrete flow

Pod IP (10.0.0.10) -> Node (10.0.0.5) SNATs to Node ephemeral IP -> Cloud NAT (NAT IP) -> Internet

If you enable Cloud NAT with `--nat-all-subnet-ip-ranges`, NAT will map requests and record logs.

12.2 Fine-grained egress controls

- Use egress `NetworkPolicy` to block unknown outbound traffic.
- Use `ExternalName` services or egress gateways in ASM to control and monitor external calls.
- Use `Private Service Connect` or `VPC-SC` for private access to Google APIs without egress to the public internet.

12.3 Restricting egress to specific external APIs

- Allowlist external IPs at Cloud Armor / WAF (if traffic goes via LB).
 - For API clients, use proxy (e.g., Squid in a managed instance group) with logging.
 - ASM egress gateway can route & log outbound traffic.
-

13 — Advanced policies & enforcement

13.1 OrgPolicy (Organization Policy Service)

Use constraints:

- `constraints/compute.skipDefaultNetworkCreation` to avoid default network
- `constraints/compute.restrictPublicIp` to prevent public IPs on VMs
- `constraints/gcp.restrictCmekCryptoKeyProjects` to restrict CMEK usage
- `constraints/iam.disableServiceAccountKeyCreation` to prevent key creation

Example to disable service account keys

```
gcloud org-policies set-policy disable-sa-keys-policy.yaml
```

13.2 VPC Service Controls (VPC-SC)

- Create service perimeters around Google-managed services (GCS, BQ, Secret Manager).
- Use with Access Context Manager (conditions) for fine-grain access.
- Important: does not protect VM traffic or GKE: used to prevent data exfiltration from managed services.

Exam traps: If the question asks “block data exfiltration from BigQuery,” choose VPC-SC. If ask “prevent DDoS,” choose Cloud Armor.

13.3 Policy Controller / OPA / Gatekeeper

- Enforce Kubernetes manifest constraints (deny certain images, hostPath usage).
- Combine with Binary Authorization: reject image if unauthenticated.

Sample constraint example (deny privileged containers)

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPPrivilegedContainer
metadata:
  name: disallow-privileged
spec:
  match:
    kinds:
      - apiGroups: [ "" ]
        kinds: [ "Pod" ]
```

14 — Performance, cost, tradeoffs

14.1 Using Cloud Armor + CDN vs internal edge

- Pros: defense at edge; cost for LB + Armor + CDN
- Cons: additional latency for WAF evaluation, cost for traffic
- Mitigation: tune caching, use cache fill for static assets

14.2 Private cluster + Cloud NAT

- Cost: NAT egress and Cloud Router costs
- Benefit: nodes have no public IPs (huge security win)

14.3 Binary Authorization strict policy

- Pro: prevents unapproved images
 - Con: developer friction; need reliable CI signing
 - Mitigation: staging exceptions and attestation automation
-

15 — Migration checklist & testing plan

Pre-migration

- Inventory application dependencies (external APIs, DNS)
- Build artifact registry & CI signing
- Create VPC and subnets, enable Private Google Access
- Prepare private GKE cluster + NAT
- Prepare Cloud SQL (private IP) or equivalent
- Configure LB + Cloud Armor + CDN
- Set up Workload Identity, KSA → GSA mappings
- Enable logging & SIEM sinks

Cutover

- Start with read-only traffic to GCP (split traffic) using LB traffic rules
- Canary deploy frontend in GKE, point a subset of users
- Monitor Cloud Armor metrics & LB logs

Post-cutover

- Revoke on-prem public access to old servers
- Enable Binary Authorization enforcement
- Run security tests (OWASP ZAP, web scanners)
- Incident response drill

Testing plan

- Penetration test (with authorization)
 - DDoS spike emulation (simulate high request volumes; verify Cloud Armor rate limits)
 - Fault injection (pod deletion, network partition)
 - Exfiltration test (attempt to access GCS from unauthorized VPC to verify VPC-SC)
 - Least privilege audit: ensure RBAC and IAM inverse/sanity checks
-

16 — Exam & interview prep checklist + practice prompts

Must-know (memorize these)

- Workload Identity steps and why it replaces keys
- How Cloud NAT maps pod egress (pod→node→NAT)
- Binary Authorization, what it enforces & where it fits
- Cloud Armor capabilities vs Cloud IDS vs VPC-SC differences
- IAP use cases and role bindings
- How to implement mTLS and AuthorizationPolicy in ASM
- How to prevent data exfil with VPC-SC and Access Context Manager
- Log routing: Logging sink → Pub/Sub → Dataflow → SIEM pattern

Practice prompts (answer in exam style)

1. You must migrate a legacy web app to GKE private cluster with no downtime and ensure only signed images run — design the steps and explain security tradeoffs.
 2. How do you allow remote sales users (Google Workspace) to administer the cluster without VPN and while enforcing MFA and device posture?
 3. A backend service is exfiltrating data to an external host. What logs show this? How do you contain it and what permanent controls do you add?
 4. A public-facing GKE frontend is under a slow-loris L7 attack. Which GCP services and configurations mitigate the issue immediately and in the medium term?
 5. Build an incident response runbook that automates blocking an attacking IP and rolling back to a safe container image.
-

Appendix: Useful commands & YAML snippets (copy-paste ready)

Create private cluster (basic)

```
gcloud container clusters create priv-cluster \
--project=my-project \
--region=us-central1 \
--enable-ip-alias \
--enable-private-nodes \
--enable-private-endpoint \
--master-authorized-networks=10.0.0.0/8 \
--no-enable-master-authorized-networks
```

Workload Identity binding

```
gcloud iam service-accounts create backend-gsa --project=my-project
kubectl create serviceaccount backend-ksa -n prod
gcloud iam service-accounts add-iam-policy-binding
backend-gsa@my-project.iam.gserviceaccount.com \
--member="serviceAccount:my-project.svc.id.goog[prod/backend-ksa]" \
--role="roles/iam.workloadIdentityUser"
```

Cloud NAT

```
gcloud compute routers create nat-router --network=my-vpc  
--region=us-central1  
gcloud compute routers nats create nat-config --router=nat-router  
--region=us-central1 \  
--nat-all-subnet-ip-ranges --enable-logging
```

Cloud Armor deny rule (by IP)

```
gcloud compute security-policies create my-waf  
gcloud compute security-policies rules create 100 \  
--security-policy my-waf \  
--expression "inIpRange(origin.ip, '203.0.113.0/24')" \  
--action "deny(403)"
```

PeerAuthentication (ISTIO mTLS)

```
apiVersion: security.istio.io/v1beta1  
kind: PeerAuthentication  
metadata:  
  name: default  
  namespace: prod  
spec:  
  mtls:  
    mode: STRICT
```

AuthorizationPolicy (allow frontend → backend)

```
apiVersion: security.istio.io/v1beta1  
kind: AuthorizationPolicy  
metadata:  
  name: allow-frontend  
  namespace: prod  
spec:  
  selector:  
    matchLabels:  
      app: backend  
  rules:  
  - from:  
    - source:  
      principals: ["cluster.local/ns/prod/sa/backend-ksa"]
```

Final practical study plan (30 days to pro)

Week 1 — Fundamentals

- Read official docs: Workload Identity, Private clusters, Cloud Armor, Binary Authorization, VPC Service Controls.
- Hands-on: create private cluster, enable NAT, deploy a sample app.

Week 2 — Identity & Mesh

- Implement Workload Identity for sample services.
- Install Anthos Service Mesh (or Istio) and enable mTLS. Create AuthorizationPolicy rules.
- Practice token exchange and inspect SPIFFE identities.

Week 3 — Supply chain & edge

- Configure Artifact Registry, sign an image in CI, create Binary Authorization attestor and policy.
- Build an HTTPS LB, attach Cloud Armor policy, and test rate limits and geoblocking.

Week 4 — Monitoring and IR

- Enable SCC and Container Threat Detection, export logs to BigQuery, write queries for incident detection.
- Simulate incidents and run through playbooks.
- Take practice questions (scenario based); write concise exam-style answers

GCP - Project Details

In short: **You (the human)** use the Name, **the CLI/API** uses the ID, and **Google Internals** use the Number.

1. Project Name ("The Friendly Label")

- **Purpose:** A human-readable label to help you identify the project in the Console.
- **Mutability:** **Mutable.** You can change it at any time.
- **Uniqueness:** **Not Unique.** You can have 10 different projects all named "My Dev Project".
- **Format:** User-defined string (e.g., "My E-Commerce Dev").
- **Use Case:** Displayed in the Dashboard and Billing Reports to make things look pretty.

2. Project ID ("The Technical Identifier")

- **Purpose:** The global unique identifier used to reference the project in scripts, CLI, and APIs.
- **Mutability:** **Immutable.** Once created, it cannot be changed.
- **Uniqueness:** **Globally Unique** across all of Google Cloud (not just your organization). If someone at another company took `my-app-prod`, you cannot have it.
- **Format:** Lowercase, numbers, hyphens (e.g., `my-app-prod-8823`).
- **Use Case:**
 - Used in `gcloud config set project [ID]`.
 - Used in Terraform code.
 - Used in resource URLs (e.g., `projects/my-app-prod/global/networks/vpc-1`).

3. Project Number ("The Internal ID")

- **Purpose:** An auto-generated numerical identifier used by Google's backend systems.
 - **Mutability:** **Immutable.**
 - **Uniqueness:** **Globally Unique.**
 - **Format:** A long string of numbers (e.g., `123456789012`).
 - **Use Case:**
 - **Crucial for Service Accounts:** Default Google-managed Service Accounts always use the Project Number in their email address.
 - **Example:** `123456789012-compute@developer.gserviceaccount.com`
-

Comparison Table

Feature	Project Name	Project ID	Project Number
Example	"Bank App Dev"	bank-app-dev-554	9823746123
Created By	You	You (or auto-generated based on name)	Google (Auto-generated)
Changeable?	Yes	No (Permanent)	No (Permanent)
Unique?	No	Yes (Globally)	Yes (Globally)
Primary Use	Human UI / Billing	CLI / APIs / Terraform	Internal / Default Service Accounts

CLI Command to see all three:

```
Bash
gcloud projects list
```

Output:

```
Plaintext
PROJECT_ID      NAME      PROJECT_NUMBER
my-app-dev-882  My App Dev  1029384756
prod-network-host Prod Network 5647382910
```

Why does this matter for the Exam/Real World?

- Deletion:** If you delete a project, you can shut down the resources, but the **Project ID** is reserved for some time (usually 30 days). You cannot immediately create a new project with the exact same Project ID.
- Legacy:** Older APIs sometimes ask for Project Number, while newer ones ask for Project ID.
- Service Accounts:** If you see an email like 12345@cloudbuild.gserviceaccount.com, that **12345** is the Project Number. This tells you exactly which project that Service Account belongs to.

gcloud commands

```

#!/bin/bash

# --- 1. Configuration & Variables ---
# Dynamically get the current Project ID
export PROJECT_ID=$(gcloud config get-value project)
export REGION_1="us-central1"
export ZONE_1="us-central1-a"
export SA_NAME="my-custom-sa"

echo "Starting Complete GCP Setup for Project: $PROJECT_ID..."

# --- 2. IAM & Service Accounts ---
echo "Creating Service Account and assigning Roles..."
# Create Service Account
gcloud iam service-accounts create $SA_NAME \
--display-name "My Custom App Service Account" --quiet

# Assign 'Storage Admin' Role to the Service Account
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:$SA_NAME@$PROJECT_ID.iam.gserviceaccount.com" \
--role="roles/storage.admin" --quiet

# --- 3. VPC Creation (Custom Mode) ---
echo "Creating 3 Custom VPCs..."
gcloud compute networks create vpn-1 --subnet-mode=custom --quiet
gcloud compute networks create vpn-2 --subnet-mode=custom --quiet
gcloud compute networks create vpn-3 --subnet-mode=custom --quiet

# --- 4. Subnet Creation ---
echo "Creating Subnets with distinct ranges..."
# Subnet for VPC 1
gcloud compute networks subnets create subnet-1 \
--network=vpn-1 --region=$REGION_1 --range=192.168.1.0/24 --quiet

# Subnet for VPC 2
gcloud compute networks subnets create subnet-2 \
--network=vpn-2 --region=$REGION_1 --range=192.168.2.0/24 --quiet

# Subnet for VPC 3
gcloud compute networks subnets create subnet-3 \
--network=vpn-3 --region=$REGION_1 --range=192.168.3.0/24 --quiet

# --- 5. Firewall Rules (SSH & ICMP) ---
echo "Creating Firewall Rules for all VPCs..."
# VPC 1 Rules
gcloud compute firewall-rules create vpn1-allow-ssh --network=vpn-1 --allow=tcp:22
--source-ranges=0.0.0.0/0 --target-tags=allow-ssh --quiet

```

```

gcloud compute firewall-rules create vpc1-allow-icmp --network=vpc-1 --allow=icmp
--source-ranges=0.0.0.0/0 --target-tags=allow-icmp --quiet

# VPC 2 Rules
gcloud compute firewall-rules create vpc2-allow-ssh --network=vpc-2 --allow=tcp:22
--source-ranges=0.0.0.0/0 --target-tags=allow-ssh --quiet
gcloud compute firewall-rules create vpc2-allow-icmp --network=vpc-2 --allow=icmp
--source-ranges=0.0.0.0/0 --target-tags=allow-icmp --quiet

# VPC 3 Rules
gcloud compute firewall-rules create vpc3-allow-ssh --network=vpc-3 --allow=tcp:22
--source-ranges=0.0.0.0/0 --target-tags=allow-ssh --quiet
gcloud compute firewall-rules create vpc3-allow-icmp --network=vpc-3 --allow=icmp
--source-ranges=0.0.0.0/0 --target-tags=allow-icmp --quiet

# --- 6. VPC Peering ---
echo "Establishing Peering between VPC-1 and VPC-2..."
# Peer VPC-1 -> VPC-2
gcloud compute networks peerings create peer-1-to-2 \
--network=vpc-1 --peer-network=vpc-2 --quiet

# Peer VPC-2 -> VPC-1 (Required for handshake)
gcloud compute networks peerings create peer-2-to-1 \
--network=vpc-2 --peer-network=vpc-1 --quiet

# --- 7. Compute Engine (VMs) ---
echo "Creating Virtual Machines..."

# VM 1: In VPC-1 with Startup Script
gcloud compute instances create instance-1 \
--zone=$ZONE_1 \
--subnet=subnet-1 \
--machine-type=e2-micro \
--tags=allow-ssh,allow-icmp \
--metadata=startup-script='#!/bin/bash
apt-get update
apt-get install -y apache2' --quiet

# VM 2: In VPC-2 with Custom Service Account attached
gcloud compute instances create instance-2 \
--zone=$ZONE_1 \
--subnet=subnet-2 \
--machine-type=e2-micro \
--tags=allow-ssh,allow-icmp \
--service-account="$SA_NAME@$PROJECT_ID.iam.gserviceaccount.com" \
--scopes=https://www.googleapis.com/auth/cloud-platform --quiet

```

```

# VM 3: In VPC-3 (Isolated)
gcloud compute instances create instance-3 \
--zone=$ZONE_1 \
--subnet=subnet-3 \
--machine-type=e2-micro \
--tags=allow-ssh,allow-icmp --quiet

echo "All resources created successfully!"

#!/bin/bash
=====
# --- 1. Configuration & Variables ---
export PROJECT_ID=$(gcloud config get-value project)
export ZONE="us-central1-a"
export REGION="us-central1"
export REPLICA_ZONES="us-central1-a,us-central1-b"

echo "Starting Disk Management Demo for Project: $PROJECT_ID..."

# --- 2. Create Zonal Persistent Disks ---
echo "Creating Zonal Disks..."

# 2.1 Standard HDD (Low cost, for backups/logs)
gcloud compute disks create disk-data-hdd \
--size=10GB \
--type=pd-standard \
--zone=$ZONE \
--quiet

# 2.2 SSD Balanced (General purpose, best value)
gcloud compute disks create disk-data-ssd \
--size=20GB \
--type=pd-balanced \
--zone=$ZONE \
--quiet

# --- 3. Create Regional Persistent Disk (High Availability) ---
echo "Creating Regional HA Disk (Replicated in 2 zones)..."

# Note: Minimum size for Regional PD is 200GB
gcloud compute disks create disk-regional-ha \
--size=200GB \
--type=pd-balanced \
--region=$REGION \
--replica-zones=$REPLICA_ZONES \
--quiet

```

```

# --- 4. Create VM with Local SSD (Ephemeral) ---
echo "Creating VM with attached Local SSD (Scratch disk)..."
# This creates a VM with a boot disk AND a super fast local scratch disk
gcloud compute instances create vm-with-disks \
    --zone=$ZONE \
    --machine-type=e2-medium \
    --local-ssd=interface=NVME \
    --quiet

# --- 5. Attach the Zonal Disk to the VM ---
echo "Attaching the Zonal SSD to the VM..."
gcloud compute instances attach-disk vm-with-disks \
    --disk=disk-data-ssd \
    --zone=$ZONE \
    --device-name=my-extra-data \
    --quiet

# --- 6. Resize the Disk (Upscale) ---
echo "Resizing disk-data-ssd from 20GB to 50GB..."
# Note: You can only increase size, never decrease.
gcloud compute disks resize disk-data-ssd \
    --size=50GB \
    --zone=$ZONE \
    --quiet

# --- 7. Backup & Restore (Snapshots) ---
echo "Creating a Snapshot of the Zonal SSD..."
gcloud compute disks snapshot disk-data-ssd \
    --snapshot-names=snap-backup-v1 \
    --zone=$ZONE \
    --storage-location=$REGION \
    --quiet

echo "Restoring a new disk from the Snapshot..."
gcloud compute disks create disk-restored-from-backup \
    --source-snapshot=snap-backup-v1 \
    --type=pd-balanced \
    --zone=$ZONE \
    --quiet

# --- 8. Verification ---
echo "Listing all created disks..."
gcloud compute disks list

echo "-----"
echo "Disk Setup Complete!"
echo "Note: To use the resized space or the attached disk,"
echo "you must still SSH into 'vm-with-disks' and run Linux commands"

```

```
echo "(lsblk, mkfs, mount, resize2fs) to format/mount the drive."
echo "-----"
```

Create Machine With SSH Keys

```
gcloud compute instances create instance-1 \
--zone=$ZONE_1 \
--subnet=subnet-1 \
--machine-type=e2-micro \
--tags=allow-ssh,allow-http \
--metadata-from-file startup-script=test.sh \
--metadata "ssh-keys=admin:ssh-rsa"
```

```
AAAAB3NzaC1yc2EAAAQABAAQACQDEDM6IzRG4zKytTt4h3aMW8pNL4CJWDxRM
AA+1yaOIL4/sSmZvAxId7BvJ5zB3IPyw9ZXX1OzINXII+kOL3fdMQI+VhWcoGR0RzB/jyZNdL
NyNDNsYzO8Qi1qEvf+p8epAEs/eT/Pe3cXsOPidYGFGkxreRgvfADlfoRmr8pW8IVyCKVYIu
AbgWI+Y2Gt0FG6G0TBUj4fBdTKVKz1aPKw4+B9pkZeBeC0XfuYLDsBX6fQr1yAAij29A2G
woYvuvl9Zv+b3INgFP0kRVp9xzPGozP4tXtvEKCMYgDDj3N+rpey7Dm9VAu+Yjy51bfwCN
hSNPZP73EFuzXIFwgO0sIWsoxHsbMO0QVwn7+DplHkNlpHLlqBJS+UjnJfqyVg7MKCud1/r
fSomdaQuAJ2qn8r7DAFG5anp3IxASFqbL11NC8R1IuwxyT+AD2xn/HDBnFFwvnPKFWvIZ
HNIS9Z2TO/7CeUFR4u7+A82sUv8ZO3erzCbJKB8V+njCqTcCycH0AvULFVvnLTioqz01Da
NfPvDLX2shA0oQ1wYFLLDwVgsbsRlm48T2ai7lsWNkbzaJHnGjAo2BThXca6dX2YhdDIKS
PELx/Rg1gPH1/KmECWcOdqeUI3DdwFYXPNkoT1ygtrzCC5i+p3iqDf7vicgCCEC6gZE8Vi
VPnSgaCbzmQ== Key for GCP"
```

```
gcloud compute instances create instance-1 \
--zone=$ZONE_1 \
--subnet=subnet-1 \
--machine-type=e2-micro \
--tags=allow-ssh,allow-http \
--metadata-from-file startup-script=test.sh,ssh-keys=my-key-formatted.txt
```

Save the key in my-key-formatted.txt

GCP key must be like below:

```
kollurisahan:ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQACQDFWHhfZ7A6/EHzecrb/Aab+K5oCDfAlhMMbNr
yAZUVL4JGRKKfHzWodnYC0A8RyQqZdSlgVDY8xobdUZ91KOR2HO6gRCdMWhW3wmY
fJcjEsy14agFKclnyo+qbiwuqQJdgDflsicW+gUg7pnfrEPmPi3wHfBYYTHCokNJBrP0TteKpqv
I10Vo27MvQXfOsGDwYo0iKL5jTYu4Fm2QQ7vKtWQ65vWNBZ5APatQMEQ3Lym9gyqKtW
Pfz+tIG2bPOKiDYc54aRTdID0Madj5eW8a1QwB0k9D2+3c/MQ0j5zfiaUbTMUG6EkypqHMq
k356UOGxJ2NGHgovqxF+9UivCB2VJRY+Mpn1eB3m5LRCBy0c35xhfM3nRD6ksgGd2Ktg
XvwzWb/VieLT6czWk1KR0QIkQM7htVrvrwetFIA26U8IBI9O9zBleS2PyR5eZskv8B1WVUQ6
1x/nZwoars6Y+rxsCY2jXyN8ESzfATOAXxtt8DZyrt8A125/LibJ4uhCbE0pmNtPOMAqvqlIkP
KKMr9E7aQ90RSNvfNLsnKj7Z5gf/Dps3+XHLhSbOWOs1ImEwYBxwL+1aQhtclgcoGSM9iX
Vg7igjOmZ28PJLdLZjjTQYG3UYwmFUTDZk/ZKu6Kr91GLMmlKF5OSvV457Z3odaH0ZPfX
Yzvu/0xiWErEPdw== kollurisahan
```

```
gcloud compute project-info add-metadata \
--metadata-from-file ssh-keys=~/ssh/kollurisahan-key-gcp.pub
```

Add created Boot Disk to VM

```
gcloud compute instances create instance-1 --zone=us-central1-a --subnet=subnet-1
--disk=name=disk-instance1,mode=rw,boot=no,auto-delete=yes --machine-type=e2-micro
--tags=allow-ssh,allow-http --metadata-from-file startup-script=test.sh
```

Mount Disk to a VM:

Part 1: Initial Setup & Persistence

(Run these when you first attach the disk)

1. Check Device Name: (Find the 10G/20G disk, e.g., sdb or sda)
lsblk
2. Format Disk: (Only for new disks! Replace /dev/sdb with your device)
sudo mkfs.ext4 -m 0 -E lazy_itable_init=0,lazy_journal_init=0,discard /dev/sdb
3. Create Folder:
sudo mkdir -p /mnt/disks/my-data-disk
4. Mount Temporary:
sudo mount -o discard,defaults /dev/sdb /mnt/disks/my-data-disk
5. Grant Permissions:
sudo chmod a+w /mnt/disks/my-data-disk
6. Create Test File:
echo "I survived the reboot" > /mnt/disks/my-data-disk/survivor.txt
7. Get UUID: (Copy the text inside UUID="...")
sudo blkid /dev/sdb
8. sudo cp /etc/fstab /etc/fstab.backup
9. Edit fstab:
sudo nano /etc/fstab
Paste at bottom: UUID=YOUR_COPIED_UUID /mnt/disks/my-data-disk ext4
discard,defaults,nofail 0 2
10. Reload System Config:
sudo systemctl daemon-reload
11. Test Config: (Should show no errors)
sudo mount -a

Part 2: Reboot Verification

(Run this to prove persistence)

11. Reboot:
sudo reboot
12. Checkpoint (After login): Check if disk is mounted.
df -h | grep my-data-disk
(You should see it mounted with size ~10G/20G).

13. Checkpoint (Data): Check if file exists.

```
cat /mnt/disks/my-data-disk/survivor.txt  
(You should see "I survived the reboot").
```

Part 3: Resizing the Disk

(Run this AFTER you increase the size in Google Cloud Console, e.g., from 20GB to 50GB)

14. Checkpoint (Identify Mismatch):

- Run `lsblk` => You should see the physical disk is **50G**.
- Run `df -h /mnt/disks/my-data-disk` => You should see the file system is still **20G**.
- Note the device name in `df` output (e.g., `/dev/sda` or `/dev/sdb`).

15. Resize the File System:

(Replace `/dev/sda` with whatever device name you saw in step 14).

```
sudo resize2fs /dev/sda
```

16. Final Checkpoint (Verify):

```
df -h | grep my-data-disk
```

(You should now see Size: 50G).

**** Important ****

1. Service Account User (`roles/iam.serviceAccountUser`)

- **What it does:** It allows a human (or another system) to "impersonate" or "act as" a Service Account.
- **The Scenario:** You are creating a VM. You want that VM to have permission to read from a Storage Bucket. You created a Service Account (SA) called `my-app-sa` that has storage permissions.
- **The Rule:** To attach `my-app-sa` to your new VM, **YOU** (the human) must have the `Service Account User` role on that specific Service Account.
- **Why?** Security. If you didn't need this role, any developer could create a VM, attach the "Administrator Service Account" to it, SSH into the VM, and delete the whole project. This role prevents privilege escalation.

2. Compute Image User (`roles/compute.imageUser`)

- **What it does:** It allows a user to **list** and **read** disk images so they can create a VM from them.
 - **The Scenario:**
 - **Project A:** Contains your "Golden Images" (Hardened Linux images with security software installed).
 - **Project B:** Is where Developers build VMs.
 - **The Rule:** For a developer in Project B to create a VM using the image in Project A, they need the `Compute Image User` role on Project A (or the specific image).
 - **Why?** By default, you can't see resources in other projects. This role bridges the gap for "Shared Image Projects."
-

3. "Are they created by User or GCP?"

There are two parts to this question: The **Roles** themselves, and the **Resources** they protect.

A. The Roles (The Permissions)

Created by: GCP.

- `roles/iam.serviceAccountUser` and `roles/compute.imageUser` are **Predefined Roles**.
- Google created these roles and maintains the list of permissions inside them. You cannot change what permissions are inside these roles (unless you create a custom role).

B. The Resources (The things being used)

Created by: Both.

- **Service Accounts:**
 - **Default SAs:** Created by **GCP** automatically (e.g., `project-number-compute@...`).
 - **User Managed SAs:** Created by **You** (e.g., `my-app-sa@...`).
 - **Note:** You can assign the `Service Account User` role to both types.
- **Images:**
 - **Public Images:** Created by **GCP/OS Vendors** (Debian, Ubuntu, Windows, RedHat). Everyone has `Image User` access to these by default.
 - **Custom Images:** Created by **You**. You must explicitly give `Image User` access to others if you want them to use your custom images.

Regions

us-central1-a

<https://docs.cloud.google.com/compute/docs/regions-zones>

For Terraform

1. Identity & Access Management (IAM)

Resource	Mandatory CLI Flags / Terraform Arguments	Why? (The Logic)
Service Account	<code>name</code> (display name) <code>account-id</code> (email prefix)	You need to create the identity before you can give it roles.
IAM Policy Binding (Assign Role)	<code>project</code> <code>member</code> (user:email, serviceAccount:email) <code>role</code> (roles/viewer)	You must answer: Who gets What Access on Which Project ?
Custom Role	<code>id</code> (role_id) <code>permissions</code> (list) <code>stage</code> (GA, ALPHA)	A custom role is just a container for a list of permissions (e.g., <code>compute.instances.start</code>).

2. VPC & Networking

Resource	Mandatory CLI Flags / Terraform Arguments	Why? (The Logic)
VPC Network	<code>name</code> <code>--subnet-mode=custom</code> (Recommended)	If you don't say <code>custom</code> , Google makes <code>auto</code> and spams 40+ subnets across the world.
Subnet	<code>name</code> <code>network</code> (VPC Name) <code>region</code> (e.g., us-central1) <code>range</code> (CIDR, e.g., 10.0.1.0/24)	A subnet <i>must</i> belong to a VPC and live in a specific Region with a specific IP block.
Firewall Rule	<code>name</code> <code>network</code> <code>action</code> (ALLOW/DENY) <code>rules</code> (protocol:port) <code>source-ranges</code> OR <code>source-tags</code>	A firewall is useless if it doesn't know Who is coming in, on What Port , and What to do (Allow/Block).
VPC Peering	<code>name</code> <code>network</code> (Local) <code>peer-network</code> (Remote)	Peering is a handshake. You need to name the local VPC and the one you want to talk to.
Shared VPC	<code>host-project</code> <code>service-project</code>	You must designate one project as the "Host" (Router) and link "Service" projects to it.

3. Compute Engine (VMs & Access)

Resource	Mandatory CLI Flags / Terraform Arguments	Why? (The Logic)
VM Instance	<code>name</code> <code>zone</code> (e.g., us-central1-a) <code>machine-type</code> (e.g., e2-medium) <code>image-family</code> or <code>image</code>	A VM needs a Location , a Size (CPU/RAM), and an OS (Boot Disk) to start.
SSH Keys	<code>metadata key: ssh-keys</code> Value: <code>username:ssh-rsa</code> ...	SSH keys are injected via Metadata. The <code>username:</code> prefix is mandatory for GCP to create the Linux user.
Startup Script	<code>metadata key: startup-script</code> (Text) OR <code>startup-script-url</code> (Storage)	Tells the VM what to do <i>immediately</i> after booting. Must use the specific key <code>startup-script</code> .

4. Storage & Images (Disks/Snapshots)

Resource	Mandatory CLI Flags / Terraform Arguments	Why? (The Logic)
Persistent Disk	<code>name</code> <code>zone</code> (Zonal) or <code>region</code> (Regional) <code>size</code> (e.g., 10GB)	A physical disk needs a location and a capacity. Default type is usually <code>pd-standard</code> or <code>pd-balanced</code> .
Snapshot	<code>name</code> <code>source-disk</code> <code>source-disk-zone</code> (Where is the disk?)	You can't take a photo (snapshot) without a subject (disk). You must tell Google where the disk lives.
Custom Image	<code>name</code> <code>source-disk</code> OR <code>source-snapshot</code>	An image is created <i>from</i> something. You must point to the source (a VM's disk or a backup snapshot).

5. Instance Groups (MIGs)

Resource	Mandatory CLI Flags / Terraform Arguments	Why? (The Logic)
Instance Template	<code>name</code> <code>machine-type</code> <code>disk</code> (boot image) <code>network</code>	The Blueprint . This looks exactly like creating a VM, but it saves the config instead of launching it.
Health Check	<code>name</code> <code>port</code> (80/8080) <code>request-path</code> (e.g., /health)	The MIG needs to know where to ping your app to see if it's alive.
Managed Instance Group	<code>name</code> <code>template</code> (Link to above) <code>zone</code> or <code>region</code> <code>base-instance-name</code>	The Factory . It needs the Blueprint (Template) and a location to start building VMs.
Autoscaling Policy	<code>min-num-replicas</code> <code>max-num-replicas</code> <code>target</code> (CPU utilization)	The Brain . It needs boundaries (Min/Max) and a trigger (CPU %) to know when to scale.

IAM - Identity Access Management

Identity Access Management

1. IAM Core Concept

Definition: IAM defines "Who can do what on which resources."

2. The "Who": Members (Identities/Principals)

These are the entities requesting access to GCP resources.

2.1 Individual Accounts:

- **Google Accounts:** (@gmail.com) - Personal usage.
- **Google Workspace:** (@company.com) - Corporate usage.
- **Cloud Identity:** Used when we don't want Google Workspace (email/drive) features, but need to manage users (e.g., Active Directory sync) to access GCP.

2.2 Collections:

- **Google Groups:** A collection of users. Best practice is to assign roles to Groups, not individuals.

2.3 Special Identifiers:

- **allAuthenticatedUsers:** Anyone who has authenticated with a Google account (very broad).
- **allUsers:** Anyone on the internet (including unauthenticated users).

2.4 Non-Humans:

- **Service Accounts:** Identities for machines/applications.

3. The "What": Permissions & Roles

3.1 Permissions:

- The smallest unit in IAM.
- **Syntax:** `Service.Resource.Verb`
- **Example:** `compute.instances.start`
- **Note:** We **cannot** assign permissions directly to users. We must group them into Roles.

3.2 Roles:

- A collection of permissions.
- **Rule:** Roles are assigned to a Person (Member).

4. Types of Roles

4.1 Primitive Roles (Basic): Historic roles existing before IAM was fully granular. Applies at the **Project Level**.

- **Owner:** Full control + Manage Roles + Billing.
- **Editor:** Create, Modify, Delete resources (No IAM/Billing).
- **Viewer:** Read-only access.

4.2 Predefined Roles: Created and managed by Google.

- **Granular:** Gives specific access to specific services.
- *Example:* `Compute Instance Admin` (Can manage VMs but not Storage).

4.3 Custom Roles: Fully tailored by the admin.

- **Purpose:** Least Privilege Principle.
- *Example:* A role allowing *only* VM start/stop, but preventing delete or create.

5. Service Accounts (SAs)

5.1 Definition:

- Special accounts for **Machines/Non-humans** (Apps, VMs).
- Used for authenticating with Google APIs.
- Not attached to any specific human user.
- We do not "login" to SAs; applications **authenticate** as the SA.

5.2 Types & Formats:

- **Default (Google Managed):** Created automatically by Google when you enable APIs.
 - *Format:* `PROJECTNUMBER-compute@developer.gserviceaccount.com`
- **User Managed:** Created explicitly by you.
 - *Format:* `SERVICE_ACCOUNT_NAME@PROJECT_ID.iam.gserviceaccount.com`

6. Governance

6.1 Organization Policies:

- These are different from IAM roles. They are rules/constraints set at the top level to restrict configurations.
 - *Example:* "Restrict all VMs to only use `us-central1` zone."
-

7. Lab/Project Task

Goal: Create a network infrastructure.

7.1 Network Setup:

- Create **3 VPCs**.
- Create **3 Subnets** (one inside each VPC).
- Create **Firewall rules** (to allow traffic).

7.2 Compute Setup:

- Create **3 VMs**, mapping one VM to each Subnet.

Google Cloud SDK

Google Cloud SDK

1. Gcloud Setup & Configuration

Basic commands to get started and configure your environment.

- **Check Version:** `gcloud --version`
- **Login:** `gcloud auth login`
- **Set Active Project:** `gcloud config set project <PROJECT_ID>`
- **Check Active Configuration:** `gcloud config list` (*Note: This shows your current settings, like active account, zone, and project.*)

2. Project Management Variables

How to list projects and store the Project ID in a variable for easier scripting.

- **List All Projects:** `gcloud projects list`
- **Get Current Project ID:** `gcloud config get-value project`

Export Project ID to a Variable:

Bash

```
export PROJECT_ID=$(gcloud config get-value project)
```

- (*Note: Using `export` makes this variable available to other commands in your session.*)

3. Cloud Shell Basics

Understanding where to save files and how to edit them.

- **Persistence:** Only the home directory `/home/username` is persistent (5GB storage). Files saved outside this folder (like in `/tmp` or system folders) are deleted when the session ends.
- **Cloud Shell Editor:** `edit iam.yaml` (*Note: This command opens the specific file in the Cloud Shell Editor GUI above the terminal.*)

4. IAM Theory: Bindings vs. Policy

Understanding the structure of permissions.

4.1 Role Binding

A binding connects **One Role** to **One or more Members** (Principals).

- *Example Binding 1:*
 - **Role:** `roles/storage.admin`
 - **Members:** `sahithi@hey.com, sahan@hey.com`
- *Example Binding 2:*
 - **Role:** `roles/compute.admin`

- **Members:** sahithi@hey.com, sahan@hey.com

4.2 IAM Policy A Policy is the **collection of all Role Bindings** attached to a resource.

- *Formula:* Binding 1 + Binding 2 + ... = **IAM Policy**

5. IAM Command Line

How to view and manipulate policies using CLI.

- **CLI Tip:** `gcloud iam [TAB][TAB]` (*Note: Pressing Tab twice ("double tap") shows available commands/autocomplete*).

View Project Policy (JSON format):

Bash

```
gcloud projects get-iam-policy $PROJECT_ID --format=json
```

- (*Note: We can use YAML or JSON files to define these policies and apply them*).
-

6. High Level Task Execution

Goal: Create a Service Account, generate a key, and assign the `storage.admin` role.

Step 1: Create the Service Account

Bash

```
gcloud iam service-accounts create my-app-sa \
--display-name "My App Service Account"
```

Step 2: Assign Role (Create the Binding)

Bash

```
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:my-app-sa@$PROJECT_ID.iam.gserviceaccount.com" \
--role="roles/storage.admin"
```

Step 3: Create and Download Key (JSON)

Bash

```
gcloud iam service-accounts keys create ~/my-key.json \
--iam-account=my-app-sa@$PROJECT_ID.iam.gserviceaccount.com
```

VPC

Virtual Private Cloud

1. VPC & Quotas

- **Default Quota:** By default, you can create **5 VPC networks** per project (this is a "Soft Limit" and can be increased by requesting a quota increase from Google).
- **Global vs. Regional:**
 - **Google Cloud VPC:** Is a **Global** resource. A single VPC spans all regions (e.g., us-central1, asia-south1).
 - **AWS VPC / Azure VNet:** Are **Regional** resources. You must create a separate VPC for every region.

2. Subnets & IP Ranges

Subnets are created inside a VPC. They are **Regional** resources (defined by a Region + CIDR Range).

2.1 Private IP Ranges (RFC 1918):

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

2.2 IP Calculation & Reserved IPs:

- **Example:** 10.1.1.0/24
 - **Total IPs:** 256 (2^8 because $32 - 24 = 8$ bits for host).
 - **Usable IPs:** 252.
 - **Google Reserves 4 IPs in every subnet:**
 1. **Network Address:** (First IP, e.g., .0)
 2. **Default Gateway:** (Second IP, e.g., .1)
 3. **Reserved for Future:** (Second to last IP, e.g., .254)
 4. **Broadcast Address:** (Last IP, e.g., .255)

3. Communication Rules

- **Same VPC, Different Regions:** VM1 (us-central1) and VM3 (asia-south1) are in the **same VPC**. They **can** communicate using internal IPs by default (because the GCP VPC is global). No peering is needed.
- **Different VPCs:** They are isolated. They cannot communicate unless you set up **VPC Peering** or **VPN**.

4. VPC Modes (Auto vs. Custom)

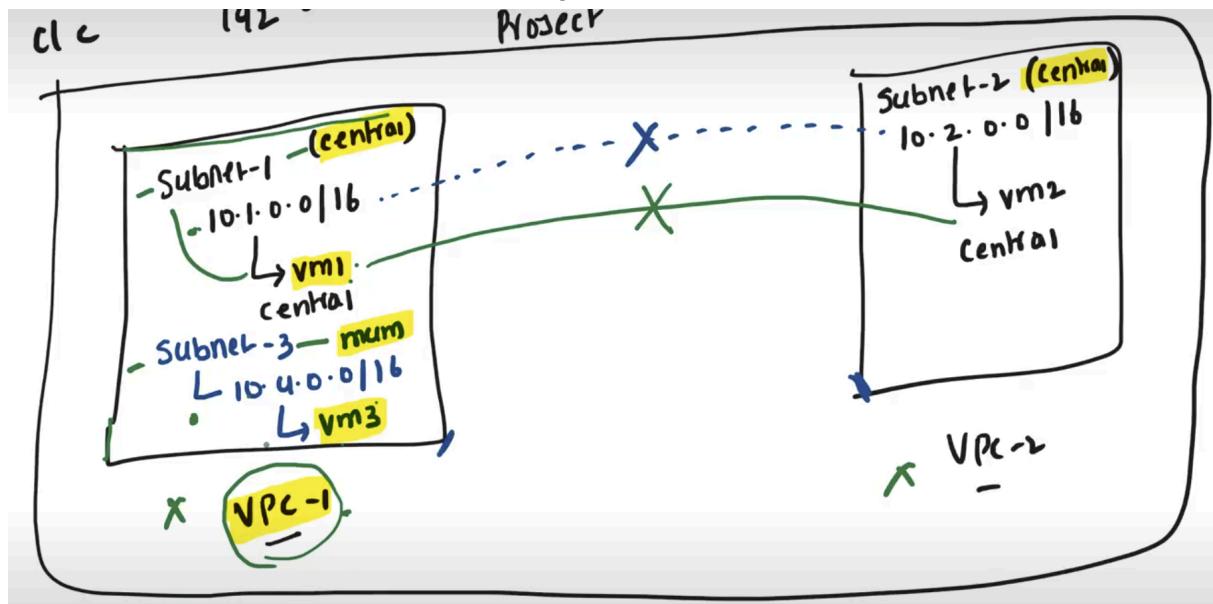
4.1 Auto Mode VPC:

- **Behavior:** Automatically creates one subnet in **every** Google Cloud region.
- **Default CIDR:** Uses a fixed range (`10.128.0.0/9`) split into `/20` subnets.
- **Drawback:** It creates 40+ subnets (one per region) even if you don't use them. This can cause IP conflicts if you later try to peer with other networks.
- **Naming:** In Auto-mode, the subnet name is the **same** as the VPC name.

4.2 Custom Mode VPC (Best Practice):

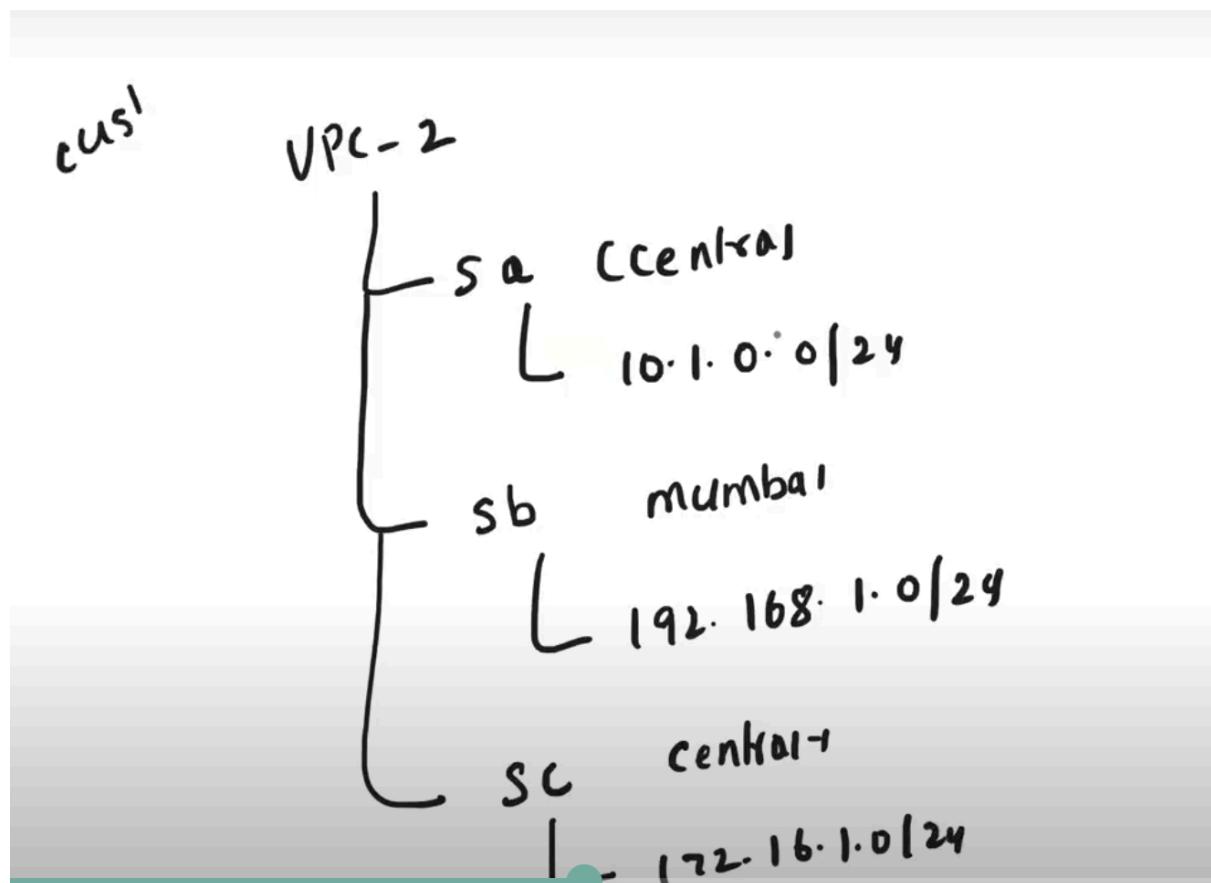
- **Behavior:** Creates a VPC with **zero** subnets.
- **Control:** You manually create only the subnets you need in the regions you want.
- **Naming Convention:** You can follow a clear pattern like `vpcname-region-subnet`.

Below yellow highlighted are in same network but different regions but VM1 and VM3 are communicated each other without VPC peering



1. Networks are classified as two subnets
 - a. Auto subnet - By default, it will create 42 subnets. But, in real time we don't use it and by default it creates CIDR ranges as well. So it deploys network for all regions and there will conflicts between regions
 - b. Custom subnet - User creates this network

In VPC we can create multiple subnets. It should be in private ranges. But, in AWS and Azure VPCs are freezed at region level. If we have application, we can have multiple subnets like mumbai, delhi etc.,



CLI Commands:

Gcloud compute networks create my-auto-vpc => It creates VPC (But, it will create auto subnet by default)

Gcloud config set project (project_id)

By default, subnet names should be “Unique” in project and in region. It will be vpcname-region_name-

Firewall

Firewall

Here is your structured and corrected guide to GCP Networking, Firewalls, and the specific tasks you outlined.

1. Networking Basics & Ports

Standard Ports we use:

- **SSH:** Port 22
- **HTTP:** Port 80
- **HTTPS:** Port 443
- **MySQL:** Port 3306

The Golden Rule: By default, **0 Firewalls = Deny All Ingress**. (If you create a custom VPC, no traffic can enter until you create a firewall rule).

2. Firewall Concepts

2.1 Scope:

- Firewalls are defined at the **VPC Network level**.
- They are "Stateful" (If you allow a request in, the reply is automatically allowed out).

2.2 Traffic Types:

- **Ingress (Inbound):** Traffic coming *into* the VM (e.g., User accessing website). Default: **Deny**.
- **Egress (Outbound):** Traffic going *out of* the VM (e.g., VM downloading updates). Default: **Allow**.

2.3 Targeting (Who gets the rule?): We can apply a firewall rule to:

1. **All Instances:** Every VM in the network.
2. **Network Tags:** Specific VMs with a label (e.g., `web-server`, `allow-ssh`).
3. **Service Accounts:** Specific VMs running as a specific identity.

3. Setting up the Infrastructure (CLI)

Here are the corrected commands for creating the VPC and Subnets.

3.1 Create VPC

Bash

```
# --subnet-mode=custom is crucial to prevent auto-creation of subnets in all regions
gcloud compute networks create my-vpc-1 \
    --subnet-mode=custom \
```

```
--bgp-routing-mode=regional \
--mtu=1460
```

3.2 Create Subnets

```
Bash
# Subnet A (Central)
gcloud compute networks subnets create subnet-a \
    --network=my-vpc-1 \
    --range=10.2.1.0/24 \
    --region=us-central1

# Subnet B (Singapore)
gcloud compute networks subnets create subnet-b \
    --network=my-vpc-1 \
    --range=10.2.2.0/24 \
    --region=asia-southeast1
```

4. Lab Task: Complex Connectivity

Scenario:

- **Instance 1a, 1b:** Can ping Instance 2.
- **Instance 1c:** Cannot ping Instance 2.
- **No one:** Can ping Instance 3.

4.1 Strategy using Network Tags:

- Tag **1a, 1b** with `source-ping`.
- Tag **1c** with `no-ping` (or just don't tag it).
- Tag **2** with `target-ping`.
- Tag **3** with nothing.

4.2 Create VMs with Tags:

```
Bash
# Instance 1a (Can ping)
gcloud compute instances create instance-1a \
    --zone=us-central1-a --subnet=subnet-a --tags=source-ping

# Instance 1b (Can ping)
gcloud compute instances create instance-1b \
    --zone=us-central1-a --subnet=subnet-a --tags=source-ping

# Instance 1c (Cannot ping)
gcloud compute instances create instance-1c \
    --zone=us-central1-a --subnet=subnet-a

# Instance 2 (Accepts ping)
```

```
gcloud compute instances create instance-2 \
    --zone=asia-southeast1-a --subnet=subnet-b --tags=target-ping

# Instance 3 (Rejects ping)
gcloud compute instances create instance-3 \
    --zone=asia-southeast1-a --subnet=subnet-b
```

4.3 Create the Firewall Rules:

Allow SSH (for everyone to login):

Bash

```
gcloud compute firewall-rules create allow-ssh-all \
    --network=my-vpc-1 \
    --action=ALLOW \
    --direction=INGRESS \
    --rules=tcp:22 \
    --source-ranges=0.0.0.0/0
```

1.

Allow Specific Ping (The Logic):

Bash

```
gcloud compute firewall-rules create allow-custom-ping \
    --network=my-vpc-1 \
    --action=ALLOW \
    --direction=INGRESS \
    --rules=icmp \
    --source-tags=source-ping \
    --target-tags=target-ping
```

2. Result: Only VMs with `source-ping` (1a, 1b) can talk to VMs with `target-ping` (2). Instance 1c fails. Instance 3 fails.

5. Service Account Firewalls & Web Server

Instead of Tags, we can use Service Accounts (SA) for better security (SAs are strictly managed by IAM, whereas any editor can change a Tag).

5.1 Setup Web Server:

Bash

```
# 1. Update OS
sudo apt update -y
# 2. Install Apache
sudo apt install apache2 -y
# 3. Check file exists
ls /var/www/html/
```

5.2 Open Port 80 using Service Account: Identify the Service Account email attached to the VM (found in API & Identity Management section of VM details).

Bash

```
gcloud compute firewall-rules create allow-http-sa \
    --network=my-vpc-1 \
    --action=ALLOW \
    --direction=INGRESS \
    --rules=tcp:80 \
    --source-ranges=0.0.0.0/0 \
    --target-service-accounts=<YOUR_VM_SA_EMAIL>
```

5.3 Deletion: When deleting via CLI, use `--quiet` to skip the prompt. `gcloud compute instances delete instance-1a --quiet`

6. Firewall Priorities & Egress

6.1 Priority Logic:

- **Lower Number = Higher Priority.**
- Priority `100` overrides Priority `1000`.
- **Conflict Rule:** If two rules have the exact same priority (e.g., both `1000`), specific "Deny" usually takes precedence over "Allow", but it is best practice to **never** have overlapping rules with the same priority number. Always use gaps (e.g., `100, 200, 1000`).

6.2 Egress Blocking (The Google Ping Test): By default, VMs can ping Google (`8.8.8.8`) because Egress is allowed. Let's block it.

Bash

```
# Deny Ping to Google DNS
gcloud compute firewall-rules create deny-google-ping \
    --network=my-vpc-1 \
    --action=DENY \
    --direction=EGRESS \
    --rules=icmp \
    --destination-ranges=8.8.8.8/32 \
    --priority=900
```

- If there is a default "Allow Egress" rule at priority `65535`, our rule at `900` will win (because `900 < 65535`), and the ping will fail.

7. Dependency Management (Terraform vs CLI)

- **Terraform:** Automatically figures out what to delete first.
- **CLI / Manual:** You must delete in reverse order of dependency:
 1. VMs
 2. Firewall Rules
 3. Subnets
 4. VPC

VPC Peering

VPC Peering

Here is your structured, corrected, and expanded guide to **VPC Peering** in Google Cloud.

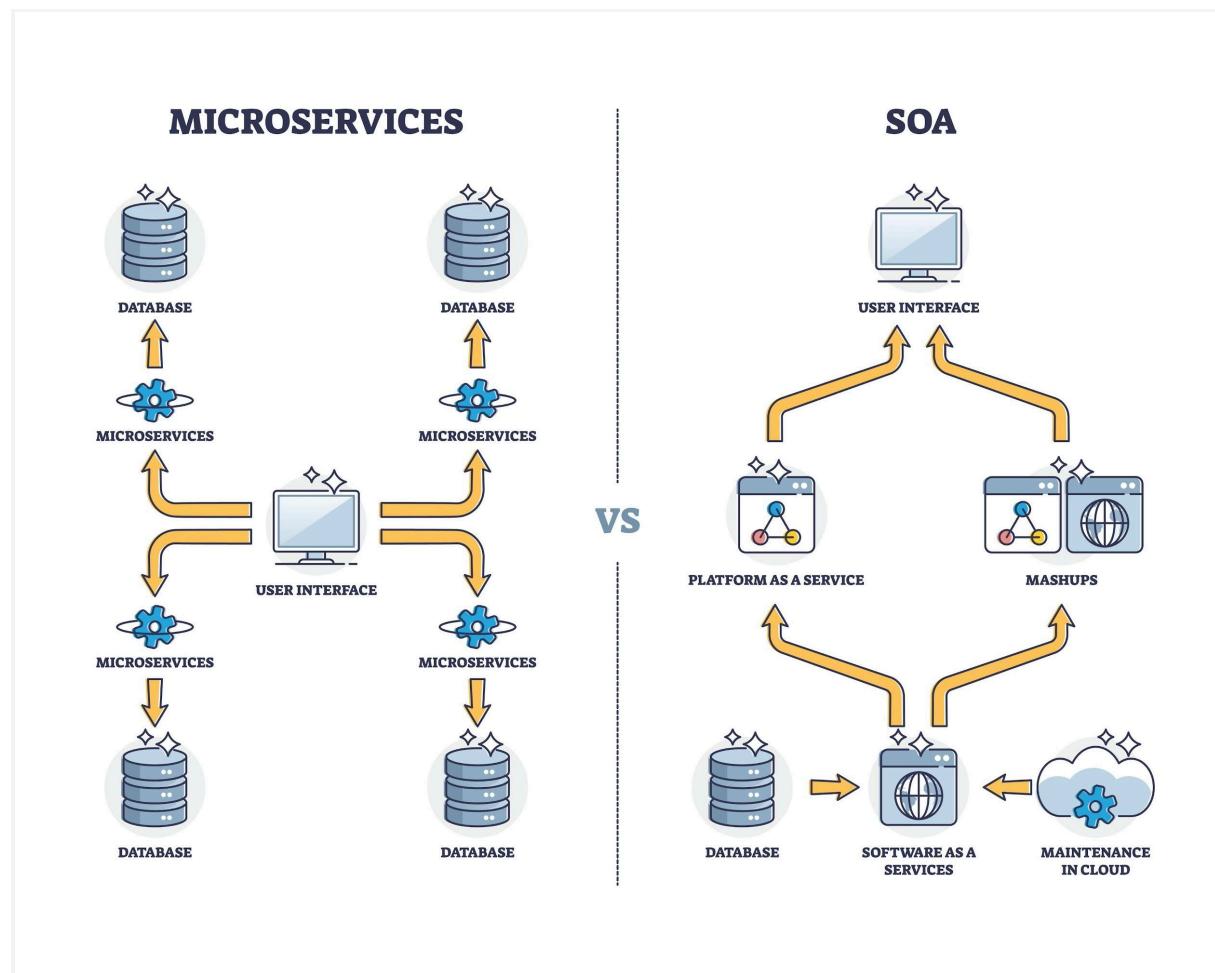
1. VPC Peering Concepts

1.1 Default Behavior:

- **Correction:** By default, two different VPCs **cannot** communicate with each other. They are completely isolated network domains.
- **Solution:** To enable communication, we use **VPC Peering**.

1.2 Types of Peering:

- **VPC Peering:** Connects two VPCs inside GCP (Same project or Different projects; Same Organization or Different Organizations).
- **Cloud Peering:** This is different. It is used to connect your on-premise network to Google's Edge network (for accessing G-Suite/Google APIs), not for connecting two VPCs.



2. Private IP Ranges (RFC 1918)

When creating subnets for peering, you usually choose from these standard private ranges:

1. 10.0.0.0 - 10.255.255.255
2. 172.16.0.0 - 172.31.255.255
3. 192.168.0.0 - 192.168.255.255

3. Restrictions & Rules

3.1 CIDR Overlap (Critical):

- No two subnets in the peered VPCs can have the same IP range.
 - Example: If VPC-A has 10.0.1.0/24, VPC-B **cannot** use 10.0.1.0/24.

3.2 Transitive Peering:

- Rule: VPC Peering is **Non-Transitive**.
- Scenario: VPC-1 peers with VPC-2. VPC-2 peers with VPC-3.
- Result: VPC-1 **cannot** talk to VPC-3.
- Solution: You must create a "Full Mesh" (3-way communication). VPC-1 must peer directly with VPC-3.

3.3 Firewall Boundaries:

- Firewall rules are **not** exchanged. You must create rules in each VPC to allow traffic.
- Tag Limitation: You cannot reference a Network Tag or Service Account from another VPC.
 - Explanation: If creating a rule in VPC-1 to allow traffic from VPC-2, you must use VPC-2's **IP Range** as the source, not its tags.

4. Lab Execution: Setup

We will create 3 VPCs, 3 Subnets (using 192.168.x.x), and 3 VMs.

4.1 Create Custom VPCs

Bash

```
gcloud compute networks create vpc-1 --subnet-mode=custom
gcloud compute networks create vpc-2 --subnet-mode=custom
gcloud compute networks create vpc-3 --subnet-mode=custom
```

4.2 Create Subnets (Distinct Ranges)

Bash

```
# Subnet 1 (192.168.1.0/24)
gcloud compute networks subnets create subnet-1 \
    --region=us-central1 --network=vpc-1 --range=192.168.1.0/24
```

```
# Subnet 2 (192.168.2.0/24)
gcloud compute networks subnets create subnet-2 \
    --region=us-central1 --network=vpc-2 --range=192.168.2.0/24

# Subnet 3 (192.168.3.0/24)
gcloud compute networks subnets create subnet-3 \
    --region=us-central1 --network=vpc-3 --range=192.168.3.0/24
```

4.3 Create VMs

Bash

```
# Instance 1 in VPC 1
gcloud compute instances create instance-1 \
    --zone=us-central1-a --subnet=subnet-1 --tags=allow-ssh,allow-icmp

# Instance 2 in VPC 2
gcloud compute instances create instance-2 \
    --zone=us-central1-a --subnet=subnet-2 --tags=allow-ssh,allow-icmp

# Instance 3 in VPC 3
gcloud compute instances create instance-3 \
    --zone=us-central1-a --subnet=subnet-3 --tags=allow-ssh,allow-icmp
```

5. Firewalls (Security)

You need to open SSH (to login) and ICMP (to test ping). Note: We must create these rules for ALL 3 VPCs. Showing VPC-1 as example.

Bash

```
# Allow SSH (Port 22)
gcloud compute firewall-rules create vpc1-allow-ssh \
    --network=vpc-1 \
    --action=ALLOW \
    --direction=INGRESS \
    --rules=tcp:22 \
    --target-tags=allow-ssh

# Allow ICMP (Ping)
# Note: Source range 0.0.0.0/0 allows ping from anywhere (Internet + Peered VPCs)
gcloud compute firewall-rules create vpc1-allow-icmp \
    --network=vpc-1 \
    --action=ALLOW \
    --direction=INGRESS \
    --rules=icmp \
    --target-tags=allow-icmp
```

6. Establishing the Peering

Peering requires a "Handshake". You must configure it on **both** sides. Until both sides are created, the status will remain **INACTIVE** or **Waiting for Peer**.

Step 6.1: Peer VPC-1 to VPC-2 (Outbound Request)

Bash

```
gcloud compute networks peerings create peer-1-to-2 \
    --network=vpc-1 \
    --peer-project=$PROJECT_ID \
    --peer-network=vpc-2
```

Status: Inactive/Waiting

Step 6.2: Peer VPC-2 to VPC-1 (Inbound Acceptance)

Bash

```
gcloud compute networks peerings create peer-2-to-1 \
    --network=vpc-2 \
    --peer-project=$PROJECT_ID \
    --peer-network=vpc-1
```

Status: Active (Connected)

Now, **instance-1** (192.168.1.x) can ping **instance-2** (192.168.2.x), provided the firewalls allow the IP ranges.

Shared VPC

Shared VPC: Concepts, Architecture, and Workflow

1. The Problem: Why Shared VPC?

- **The Default VPC Risk:** By default, new Google Cloud projects come with a "Default VPC" with pre-populated firewall rules (allowing SSH, RDP, ICMP). In an enterprise environment, this is a security risk. Organization Policies usually enforce **disabling the creation of Default VPCs**.
- **The Permission Dilemma:**
 - If an Admin gives a developer **Instance Admin** rights in a project with no network, the developer cannot create a VM (no network interface).
 - If the Admin gives the developer permissions to create networks (**Network Admin**), the developer can also create Firewall rules, Routes, and Peering. This breaks the "Least Privilege" principle and fragments network security.
- **The Solution:** Shared VPC allows the separation of **Network Administration** (Host Project) from **Instance Administration** (Service Project).

2. Architecture & Terminology

- **XPN (Cross Project Network):** The legacy name for Shared VPC. You may still see this in API calls or older documentation.
- **Host Project:**
 - Owned by the **Networking Team**.
 - Contains the VPC Network and all Subnets.
 - Centralized location for Firewalls, Routes, and Cloud VPN/Interconnects.
 - Scenario: One Host Project containing three subnets: **dev-subnet**, **prod-subnet**, and **private-subnet**.
- **Service Projects:**
 - Owned by **App/Dev Teams** (e.g., **dev-project**, **prod-project**).
 - These projects **do not** have their own local VPCs.
 - They consume the network resources shared by the Host Project.

3. IAM Roles & Permissions

To make this work, specific roles must be granted at specific levels.

Role Name	Scope	Assigned To	Purpose
Compute Shared VPC Admin	Organization / Folder	Network Admin	Allows the user to configure a project as a "Host Project" and attach Service Projects.

Compute Network User	Host Project (or specific Subnet)	Service Project Users (Devs)	Allows the developer to use the subnets to launch VMs, but not modify network settings (Firewalls/Routes).
Compute Instance Admin	Service Project	Service Project Users (Devs)	Allows the developer to create and manage VM instances within their own project.

4. Configuration Workflow

Step 1: Enable Shared VPC (Networking Team)

1. The Organization Admin grants the Networking Team the **Compute Shared VPC Admin** role.
2. The Networking Team designates the **Host Project**.
3. They create the VPC and the required subnets (**dev**, **prod**, **private**).

Step 2: Attach Service Projects

1. The Networking Team attaches **dev-project** and **prod-project** to the Host Project.
2. This authorizes those projects to communicate with the Host's network.

Step 3: Grant User Permissions (The "Sharing" Step)

We must grant the **Compute Network User** role to the specific identities (Principals) managing the Service Projects.

- **Granularity:** We can share the *entire* Host VPC, or specific subnets.
- **Scenario:**
 - Grant **dev-team-group** the **Network User** role on **dev-subnet** *only*.
 - Grant **prod-team-group** the **Network User** role on **prod-subnet** *only*.
 - *Result:* Devs cannot see or use the Prod subnet, and vice versa.

5. The User Experience (Service Project Admin)

When "User X" (the developer) logs into the **dev-project** to create a VM:

1. **Creating the VM:** The user goes to create an instance.
2. **Network Selection:** Under the Networking interface, they will see two tabs:
 - **Networks in this project:** (Empty, as there is no local VPC).
 - **Networks shared with me:** (This is where **dev-subnet** from the Host Project appears).
3. **Launch:** The user selects the shared subnet and launches the VM. The VM now lives in the **dev-project** (billing/management) but sits on the **Host Project** network (IP addressing/connectivity).

6. Operational Limitations & Security

- **Firewalls:** The Service Project user **cannot** create firewall rules.

- **Process:** If "User X" creates a web server and needs Port 80 opened:
 1. They verify the port is blocked.
 2. They must raise a ticket/request with the **Networking Team** (Host Project Owners).
 3. The Networking Team creates the Firewall Rule in the **Host Project**.
- **Benefit:** This ensures the Networking Team maintains a strict security perimeter across the entire Organization, regardless of how many Service Projects exist.

Google Computing

Google Cloud Computing Services

1. Compute Engine (GCE)

Type: IaaS (Infrastructure as a Service)

- **Concept:**
 - Provides virtualized hardware (VMs) similar to VMware on-premise.
 - You have full control over the **Operating System (OS)**, kernel, and software stack.
- **Best For:**
 - **"Lift and Shift"**: Moving existing on-prem applications to the cloud without rewriting code.
 - **Legacy Apps**: Applications that require specific OS configurations or kernels.
 - **Databases**: Running SQL/NoSQL databases where you need fine-tuned control over the file system.
 - **Long-running workloads**: Processes that run 24/7.

2. App Engine (GAE)

Type: PaaS (Platform as a Service)

- **Concept:**
 - **"Code-First" approach**: You provide the source code (Java, Python, Go, etc.); Google handles the build, deployment, and infrastructure.
 - Developers focus only on writing code, not managing servers.
- **Two Environments**:
 - **Standard Environment**: Fast scaling (down to zero), specific language runtimes, sandboxed.
 - **Flexible Environment**: Runs in Docker containers, supports any language, longer startup time, does not scale to zero (minimum 1 instance).
- **Best For:**
 - Web Websites and Mobile Backends.
 - Developers who want to focus purely on code.

3. Cloud Run

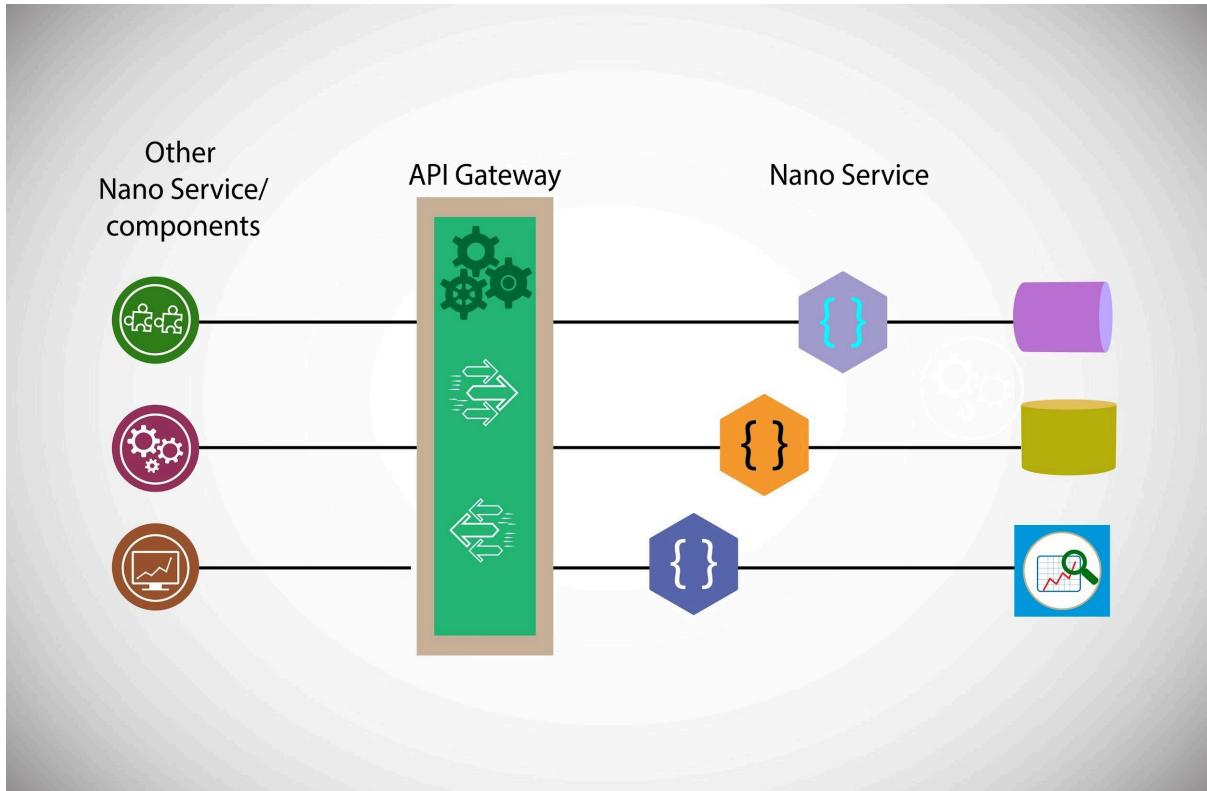
Type: Serverless Container Platform (PaaS)

- **Concept:**
 - **"Container-First" approach**: You provide the **Container Image**; Google runs it.
 - Fully managed serverless platform (No server management).
 - **Scales to Zero**: You pay **only** when a request is being processed. If no one visits, you pay \$0.
- **Best For:**
 - **Stateless Containers**: HTTP/Web APIs, microservices.
 - **Event-Driven Apps**: Can be triggered by Pub/Sub or Storage events.
 - **Any Language**: If you can containerize it (Docker), you can run it.

- *Note:* It is often preferred over App Engine now because it is portable (based on Knative).

4. Google Kubernetes Engine (GKE)

Type: CaaS (Container as a Service) / Orchestration



Shutterstock

- **Concept:**
 - Managed **Kubernetes** service for running containerized applications at scale.
 - **Shared Responsibility:**
 - **Control Plane (Master):** Managed by Google (API, scheduling, health checks).
 - **Worker Nodes:** Managed by you (though GKE Autopilot can manage these too).
 - Allows control over hardware (GPUs, specific machine types) for your containers.
- **Best For:**
 - **Containerized Microservices:** Complex architectures with many interacting services.
 - **Stateful Applications:** Databases or apps requiring persistent disks attached to specific pods.
 - **Heavy Workloads:** Applications requiring custom machine types or GPUs.
 - **Hybrid/Multi-cloud:** Using Anthos to run workloads on-prem and in Cloud.

5. Cloud Functions (GCF)

Type: FaaS (Function as a Service)

- **Concept:**
 - Single-purpose, small snippets of code (functions).
 - Triggered purely by **Events** (HTTP request, File uploaded to Storage, Message in Pub/Sub).
- **Best For:**
 - **"Glue" Code:** Connecting GCP services together.
 - **Lightweight Automation:** E.g., "When a file is uploaded to Storage, resize the image."

5.1 Cloud Functions 2nd Generation (Important)

Concept:

The modern version of Cloud Functions built on top of Cloud Run and Eventarc. It combines the simplicity of functions with the power of Cloud Run infrastructure.

Key Improvements over Gen 1:

1. **Concurrency:**
 - Gen 1: 1 Instance = 1 Request.
 - Gen 2: 1 Instance = **Multiple Requests** (Much cheaper for high traffic).
2. **Execution Time:**
 - Gen 1: Max 9 minutes.
 - Gen 2: Max **60 minutes** (for HTTP triggers).
3. **Traffic Splitting:** Since it runs on Cloud Run, you can split traffic between versions (e.g., 90% to stable, 10% to test) for safer deployments.

Summary Comparison Table

Feature	Compute Engine	GKE	App Engine	Cloud Run	Cloud Functions (Gen 2)
Control	High (OS Level)	Medium (Cluster Level)	Low (App Level)	Low (Container Level)	Low (Code Level)
Input	VM Image	Containers (Pods)	Source Code	Container Image	Code Function
Scaling	Manual / Autoscaler	Cluster Autoscaler	Automatic	Automatic (to 0)	Automatic (to 0)
Primary Use	Legacy / Lift & Shift	Orchestration / Complex Apps	Web Apps	Stateless Microservices	Event Triggers / Glue Code

Google Compute Engine

Google Compute Engine

1. VM - It is Infrastructure as Service (IaaS)
2. Multiple VMs
 - a. Instance template
 - b. Instance Group
 - c. Autoscaling
 - d. Healthchecks
3. Backends
4. LoadBalancers
5. Frontend Config
6. DNS
7. CDN - Content Delivery Network
8. Https

Google Compute Engine:

1. General Information & Identity

- **Naming Convention:** VM names must be in lowercase letters only; no capital letters or special characters are allowed.
- **Project Scope:** VMs are deployed within a specific Google Cloud Project.

2. Location & Scope

- **Region:** The specific data center location (geographic area) where the server is deployed.
- **Zone (Zonal Scope):**
 - VMs are strictly Zonal resources.
 - You must select one specific zone within a region.
 - **Permanence:** Once selected, the zone cannot be changed.
 - **Availability:** If the specific zone goes down, the application goes down.

3. Machine Configuration (Compute)

- **Machine Families:** Machine types are grouped into families based on workload needs.
 - **General Purpose:** Best for day-to-day computing and low cost (e.g., E2, N1, N2, N3).
 - **Specialized Families:** Compute Optimized, Storage Optimized, and Memory Optimized.
- **Configuration Types:**
 - **Preset:** Pre-defined configurations given by Google (e.g., `e2-standard-2`, where "2" indicates the CPU count).
 - **Custom:** You can define specific RAM and CPU.
 - **CPU:** Must be an even number.

- **RAM:** Can be increased in 0.25 GB increments (CLI requires bytes, e.g., **4*1024**).
- **Scalability:** You can resize the CPU and Memory configuration after the deployment is complete.

4. Boot Disk & Operating System

- **Image Types:**
 - **Public Images:** Standard OS images provided by Google.
 - **Custom Images:** Created by the user.
 - **Snapshots:** Backups of previous disks.
- **OS Flavors:** Linux or Windows.
 - **Default:** Debian GNU/Linux.
- **Disk Size Requirements:**
 - **General Minimum:** 10 GB.
 - **CentOS:** Minimum 20 GB.
 - **Windows:** Minimum 50 GB.
 - **Maximum:** Up to 65 TB.
- **Containers:** Deployment of containers on VMs is possible but the specific "Deploy Container" feature noted is deprecated.

5. Storage & Data Persistence

- **Disk Types:**
 - **Persistent Disk (PD):** Can be Zonal or Regional.
 - **Local SSD:** Faster but strictly tied to the hardware (ephemeral).
 - **Cost:** SSDs are more expensive; Standard HDDs are cheaper.
- **Encryption:**
 - **Default:** Google Managed Encryption Keys (GMEK).
 - **Custom:** Customer-Managed Encryption Keys (CMEK) via Cloud KMS.
- **Lifecycle & Rules:**
 - **Deletion Rule:** You must specify if the boot disk should be kept or deleted when the VM is deleted.
 - **Resizing:** Disks can be increased in size, but **cannot** be decreased.
 - **Additional Disks:** You can attach multiple additional disks to a single VM.
- **Backup (Snapshots):**
 - Can be manual or scheduled.
 - Supports incremental backups (saving only changes since the last snapshot).

6. Networking

- **Traffic Rules:** Options to allow HTTP (Port 80) and HTTPS (Port 443).
- **Load Balancing:** Can allow load balancer health checks.
- **IP Addressing:**
 - **Subnet:** Must be chosen (CIDR block).
 - **Private IP:** Every VM has one.
 - **Public IP:**
 - **Ephemeral:** Changes if the VM is stopped and started.

- **Static:** Reserved IP that does not change.
- **Best Practice:** Only DNS/Front-end should hold a Public IP; backend servers should rely on Private IPs.
- **Network Tier:** Defaults to "Premium" tier.

7. Observability

- **Ops Agent:** Can be installed to collect advanced metrics and logs for monitoring.

8. Security

- **Service Accounts (SA):** Defines the identity of the VM.
 - **Default SA:** Can be used with Access Scopes (e.g., "Full" access needed to copy logs to storage buckets).
 - **Custom SA:** Recommended for granular control using IAM roles.
- **Access & Authentication:**
 - **OS Login:** Enables managing access via IAM rather than manual keys.
 - **SSH Keys:**
 - **Manual:** You generate a Private/Public key pair.
 - The **Public Key** is given to the VM configuration.
 - The **Private Key** is retained by the user to connect.

9. Advanced & Lifecycle Management

- **Provisioning Models:**
 - **On-Demand:** Standard pricing and availability.
 - **Spot/Preemptible:** Significantly cheaper but can be reclaimed by Google at any time.
- **Automation:**
 - **Startup Scripts:** Automate tasks upon boot (e.g., `sudo apt-get update`, installing Apache).
- **Protection:**
 - **Deletion Protection:** Prevents accidental deletion. Must be edited and unchecked to delete the VM.
- **Reservations:** Ability to reserve specific capacity (e.g., 5 VMs with 2 CPU/4GB RAM) to ensure availability.
- **Cost:** Billing is calculated on a pro-rated basis (pay for what you use).

Below is the gcloud command:

```
gcloud compute instances create my-complex-vm \
--project=my-project-id \
--zone=us-central1-a \
--machine-type=e2-custom-4-4096 \
--image-family=centos-7 \
--image-project=centos-cloud \
--boot-disk-size=20GB \
--boot-disk-type=pd-ssd \
--boot-disk-device-name=my-vm-boot \
--boot-disk-kms-key=projects/my-project/locations/global/keyRings/my-ring/cryptoKey
s/my-key \
--boot-disk-auto-delete \
--create-disk=name=my-data-disk,size=50GB,mode=rw,type=pd-standard,auto-delete=
no \
--network-interface=subnet=default,network-tier=PREMIUM,address=1.2.3.4 \
--tags=http-server,https-server \
--service-account=my-custom-sa@my-project.iam.gserviceaccount.com \
--scopes=https://www.googleapis.com/auth/cloud-platform \
--deletion-protection \
--preemptible \
--reservation-affinity=any \
--metadata=startup-script='#!/bin/bash
yum update -y
yum install -y httpd
systemctl start httpd',block-project-ssh-keys=TRUE,ssh-keys='username:ssh-rsa
AAAAB3NzaC1yc2E...'
```

WebServer SSH

1. VM Lifecycle Management

1.1 Stop VM:

```
gcloud compute instances stop first-vm-cli --zone=us-central1-a
```

1.2 Delete VM:

```
gcloud compute instances delete first-vm-cli
```

1.3 Quiet Mode:

We can use --quiet (or -q) to skip the confirmation prompt (e.g., "Do you want to continue?").

2. Configuration

2.1 Set Default Zone:

```
gcloud config set compute/zone us-central1-a
```

⇒ It takes zone us-central1-a as default for future commands.

3. SSH & Security Concepts

3.1 Login Types: We can login using key-based login (Public/Private) or Username/Password.

3.2 Google Managed Keys: By default, Google creates keys for us when we use the browser SSH or gcloud ssh. (Google uses google_compute_engine for this).

3.3 Key Placement: We place Public keys in the VMs (or GCP Metadata) and keep Private keys on our local machine.

3.4 Security Rules:

- Use per-user keys (not shared).
- Rotate keys regularly.
- **Never** share Private keys in Git, etc.

4. Generating Custom Keys

4.1 Command:

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/siva-key -C "hey there"
```

- **-t rsa**: The algorithm type (Note: **rsa** is standard).
- **-b 4096**: The number of bits (higher = harder to hack).
- **-f**: The folder/filename.
- **-C**: The comment.

5. Managing Metadata (Applying Keys)

5.1 Project Level (Global Access):

There is a metadata section in the project. If we add keys here, we can access all VMs in the project.

- `gcloud compute project-info add-metadata --metadata-from-file ssh-keys=~/ssh/my-ssh-key.pub`

5.2 Instance Level (Single VM Access):

We can add keys specifically to one VM.

- `gcloud compute instances add-metadata <VM-NAME> --zone=<ZONE> --metadata-from-file ssh-keys=~/ssh/my-ssh-key.pub`

6. Connecting

6.1 Connect Command:

`ssh -i <private-key> username@ipaddress`

- `-i`: Identity file (this tells SSH which private key to use).

Project:

-
1. Create 3 VMs and 1 key should be placed in one VM and it should be only connected
 2. Use project level and it should be accessed all VMs

Start Up Scripts and Reservations

1. Viewing Logs (Debugging)

1.1 The Command:

To view the output of a startup script, SSH into the VM and run:

Bash

```
sudo journalctl -u google-startup-scripts.service
```

1.2 Breakdown:

- `sudo`: Requires root privileges.
- `journalctl`: Systemd log viewer.
- `-u`: Filter by specific unit/service.

2. Service Account (SA) Lifecycle & Recovery

2.1 Deletion & Recovery Rules:

- **Time Limit:** You can "undelete" (recover) a Service Account if it was deleted less than **30 days** ago.
- **Past 30 Days:** The account is permanently deleted and cannot be recovered by Google.
- **Audit Logs:** To find out *who* deleted the SA, you must check the **Cloud Audit Logs** (Activity tab).

2.2 Recreating SAs with the Same Name:

- **User-Managed SAs:**
 - **Yes**, you can create a new SA with the same email name (e.g., `app-sa@project...`).
 - **Critical Warning:** Even if the name is the same, the internal **Unique ID (UID)** will be different.
 - **Impact:** Any IAM bindings (permissions) attached to the *old* SA will **not** apply to the *new* SA. You must re-assign roles to the new identity.
- **Google-Managed SAs:**
 - **No**, you cannot manually recreate these (e.g., the default compute engine SA). You typically have to disable and re-enable the specific API to trigger Google to regenerate them.

3. Startup Scripts (Metadata)

3.1 Definition:

- **Key:** Must be exactly `startup-script` (defined by Google).
- **Value:** The actual shell script content or a file path.

3.2 Scope & Precedence (Project vs. VM):

- **Project Level:**
 - If defined here, the script runs on **every** VM created in the project.

- Useful for installing security agents or compliance tools needed everywhere.
- **VM (Instance) Level:**
 - Defined on a specific VM.
 - Useful for app-specific installations (e.g., installing Apache on a web server).
- **Execution Order:**
 - If both exist, **BOTH** are executed.
 - Project-level scripts run *first*.
 - Instance-level scripts run *second*.

4. Creating VMs with Startup Scripts (CLI)

Here is the corrected command.

4.1 Syntax Corrections:

- **Zone:** `us-central-a` ⇒ `us-central1-a` (Missing the number 1).
- **Flag:** `-` (dash) ⇒ `--` (double dash).
- **Metadata:** ensuring the syntax `key=filename` is correct.

4.2 The Command:

Bash

```
gcloud compute instances create startup-script-vm \
--zone=us-central1-a \
--metadata-from-file=startup-script=./my-script.sh
```

4.3 Using direct string (for short scripts):

If you don't have a file, you can pass the script directly:

Bash

```
gcloud compute instances create simple-vm \
--zone=us-central1-a \
--metadata=startup-script='#!/bin/bash
apt-get update
apt-get install -y apache2'
```

Key `startup-script` This is the specific "reserved name" Google looks for to know "I should run this code." **File Path `./my-script.sh`** This is where the file lives on your local computer. **Value** The Content The text/code inside the file becomes the value.

If I have from bucket level then it will start-script-url and we need to give the endpoint

There are 4 options

1. VM level
2. Project level
3. CLI level
4. Blob level

If you run `gcloud compute instances create my-vm` without specifying a machine type, the CLI defaults to:

N1-standard-1

5. Creating Zonal Reservations

Concept: A Reservation is **not** a data backup. It is a **Capacity Assurance**.

- **The Problem:** During a big product launch or Black Friday, you might need 100 VMs. If the Google Data Center is full, you might get a "Resource Exhausted" error.
- **The Solution:** You "book" the hardware in advance to guarantee that the capacity is available exactly when you need it.

5.1 Scope & Sharing

When creating a reservation, you must define **who** can use it:

1. **Single Project (Local):** Only the current project can use these reserved machines.
2. **Shared Reservation:** You can share this reservation across multiple projects (within an Organization).
 - *Benefit:* Great for GKE clusters or Vertex AI jobs that span multiple projects but need guaranteed hardware.

5.2 Consumption Policies (How VMs use it)

You must decide how VMs "find" this reservation:

- **Use Reservation Automatically (Default):**
 - Any VM created in this zone that matches the machine type (e.g., `n2-standard-4`) will automatically use this reservation.
 - *Good for:* General capacity safety.
- **Select Specific Reservation:**
 - A VM will **only** use this reservation if you explicitly flag that VM to use it (using `--reservation-affinity`).
 - *Good for:* Critical Prod workloads. It prevents a random Dev/Test VM from accidentally "stealing" your reserved spot.

5.3 Billing (Critical Warning)

- **The Clock Starts Immediately:** As soon as you click "Create," you are billed for the machines.
- **Usage Irrelevant:** Whether you turn the VMs on or off, **you pay for the reservation 24/7.**
 - *Analogy:* It is like reserving a table at a restaurant. You pay for the seat whether you show up to eat or not.
- **Discounts:** Reservations often combine well with **Committed Use Discounts (CUDs)** to lower the price.

5.4 Steps to Create

1. **Select Region & Zone:** Reservations are strictly Zonal (e.g., `us-central1-a`).
2. **Select Properties:**
 - Machine Type (e.g., `n2-standard-4`).
 - Number of VMs (e.g., 50).
3. **Select Consumption:** Automatic or Specific.
4. **Create:** Billing begins instantly.

Disks

Compute Engine Storage Overview

To run a VM application or database, we need CPU, Memory, and Storage.

- Boot Disk: Every VM requires a boot disk (OS Image) to start.
- Images: can be Public (provided by Google, e.g., Ubuntu, Windows) or Custom (created by you).

1. Block Storage (Persistent Disks)

Definition: Virtual hard drives attached to VMs over the network (Network Attached Storage).

- Behavior:
 - Network Attached: They are not physically plugged into the server; they connect via Google's network (located in different racks).
 - Flexibility: One Disk can be attached to one VM in Read/Write mode. (Multi-writer is possible only for specific high-end configurations).
 - Scalability: You can increase disk size anytime (even while running), but you cannot decrease it.
 - Lifecycle: Independent of the VM. If you delete the VM, the disk can remain (if "Delete boot disk when instance is deleted" is unchecked).
 - Limits: Min size 10GB; Max size 65TB.

Types of Persistent Disks (PD):

A. Zonal Persistent Disk (Standard)

- Scope: Single Zone (e.g., us-central1-a).
- Restriction: The VM must be in the exact same zone to attach this disk.
- Durability: High, but if the zone fails, data is inaccessible.

B. Regional Persistent Disk (High Availability)

- Scope: Replicated across two zones in the same region.
- Use Case: High Availability (HA) databases or critical apps. If one zone goes down, the disk is still available in the other zone.
- Supported Machines: E2, N1, N2, N2D.
- Minimum Size: 200 GB.
- Constraint: Performance is slightly lower than Zonal due to replication overhead.

C. Disk Performance Types

1. pd-standard (HDD): Low cost, sequential access (Logs, Backups).
2. pd-balanced (SSD): Best value, general purpose (Boot disks, LOB apps).
3. pd-ssd (SSD): Performance critical (Databases, Low latency).
4. pd-extreme: High-end database workloads (SAP HANA, Oracle).

2. Other Storage Options

Local Disk (Local SSD)

- Definition: Physically attached to the server hosting your VM (not network attached).
- Performance: Extremely fast (High IOPS, very low latency).
- Volatility: Ephemeral. If you Stop/Shutdown the VM, data is lost.
- Use Case: Temporary caches, scratch processing, or NoSQL databases (like Cassandra) that handle replication themselves.
- Capacity: Fixed size (375 GB chunks). We can attach up to 24 Local SSD
- We can use CMEK, CSEK encryption keys
- Local SSD can't be your boot disk
- Local SSD can't be your extra disk
 - We need to attach local SSD as extra disk during the instance creation
 - Every VM will not support local SSD

Object Storage (Cloud Storage / Blob)

- Definition: Stores "Objects" (files) in "Buckets". It has no file system hierarchy (no directories, just keys).
- Access: Accessed via API (HTTP/HTTPS), not mounted as a drive.
- Scalability: Infinite capacity.
- Use Case: Storing backups, images, videos, logs, and static website hosting.
- Classes: Standard (Hot), Nearline (Rare), Coldline (Very Rare), Archive (Long-term).

File Storage (Filestore)

- Definition: Fully managed NFS (Network File System) server.
- Access: Can be mounted by multiple VMs simultaneously in Read/Write mode.
- Use Case: Shared workloads, content management systems (CMS), lift-and-shift apps requiring a shared drive.
- Performance: Ranges from Basic (HDD) to High Scale (SSD).

3. Encryption

Data is always encrypted at rest by default. You have three choices for key management:

1. Google-Managed Keys (Default):
 - Google manages the entire lifecycle (rotation, storage). You don't see the keys.
2. Customer-Managed Encryption Keys (CMEK):
 - You manage the keys using Cloud KMS (Key Management Service).
 - You control rotation and permissions, but keys are stored in GCP.
3. Customer-Supplied Encryption Keys (CSEK):
 - You generate the key on your own premise (laptop/HSM).
 - You provide the raw key to Google when creating the disk.
 - Correction: Google does allow providing these keys via the Console (UI), CLI, or API. Google creates a hash of the key to verify requests but never stores the raw key. If you lose this key, the data is lost forever.

4. Creating a Disk (Step-by-Step)

1. Location: Select Zonal or Regional.
2. Source:
 - Blank Disk: Empty storage.
 - Image: Create from an OS image.
 - Snapshot: Create from a backup of another disk.
3. Type: Choose (Standard, Balanced, SSD, Extreme).
4. Size: Enter size (Min 10GB for Zonal, 200GB for Regional).
5. Encryption: Choose Google-managed (default) or CMEK/CSEK.
6. Create: The disk is created but not attached.
7. Attach: You must go to the VM instance settings to attach this existing disk.

5. Local SSD Constraints

Local SSDs are physically attached to the server (not network-attached), offering extreme speed but with strict limitations:

- **Cannot be a Boot Disk:** You cannot install the OS on a Local SSD.
- **Cannot be "Detached":** Local SSDs are tied to the specific physical server. You cannot create one separately and attach it later; it must be created **at the same time** as the VM.
- **Ephemeral:** If the VM stops, the data is lost.

6. Hyperdisk Overview

Definition: Hyperdisk is Google's **Next-Generation** block storage. While Persistent Disk (PD) is the standard, Hyperdisk is the "Advanced/Extended" version designed for high-end workloads.

Key Features:

- **Decoupled Performance:** Unlike Standard PDs, where speed depends on disk size, Hyperdisk allows you to configure **Capacity, IOPS, and Throughput** independently.
 - *Example:* You can have a small 100GB disk with massive IOPS for a database.
- **Workload Types:** Ideal for high-performance databases (SAP HANA, SQL Server) and massive data processing (Hadoop/Spark).
- **Machine Compatibility:** Only supported on newer machine series (e.g., **C3, N4, C3D**). It requires newer CPU architectures.

3. Comparison: Persistent Disk (PD) vs. Hyperdisk

Feature	Persistent Disk (PD)	Hyperdisk (Next-Gen)
Type	Network Attached Storage	Network Attached Storage
Boot Disk Support	Yes (All types)	Limited (Only Hyperdisk Balanced; Extreme/Throughput are data-only)
Scope	Zonal & Regional	Zonal Only
Performance Model	Dependent on Disk Size (Bigger disk = Faster)	Provisioned (You pay specifically for the IOPS/Throughput you define)
Min Size	10 GB	4 GB
Use Case	General purpose, Web Apps, Standard DBs	High-end Enterprise DBs, High Throughput Analytics

Here are your notes on **Advanced Disk Management (Hyperdisk & Local SSD)** rephrased and structured for clarity.

1. Local SSD Constraints

Local SSDs are physically attached to the server (not network-attached), offering extreme speed but with strict limitations:

- **Cannot be a Boot Disk:** You cannot install the OS on a Local SSD.
- **Cannot be "Detached":** Local SSDs are tied to the specific physical server. You cannot create one separately and attach it later; it must be created **at the same time** as the VM.
- **Ephemeral:** If the VM stops, the data is lost.

2. Hyperdisk Overview

Definition: Hyperdisk is Google's **Next-Generation** block storage. While Persistent Disk (PD) is the standard, Hyperdisk is the "Advanced/Extended" version designed for high-end workloads.

Key Features:

- **Decoupled Performance:** Unlike Standard PDs, where speed depends on disk size, Hyperdisk allows you to configure **Capacity**, **IOPS**, and **Throughput** independently.
 - *Example:* You can have a small 100GB disk with massive IOPS for a database.
- **Workload Types:** Ideal for high-performance databases (SAP HANA, SQL Server) and massive data processing (Hadoop/Spark).
- **Machine Compatibility:** Only supported on newer machine series (e.g., **C3**, **N4**, **C3D**). It requires newer CPU architectures.

3. Comparison: Persistent Disk (PD) vs. Hyperdisk

Feature	Persistent Disk (PD)	Hyperdisk (Next-Gen)
Type	Network Attached Storage	Network Attached Storage
Boot Disk Support	Yes (All types)	Limited (Only Hyperdisk Balanced; Extreme/Throughput are data-only)
Scope	Zonal & Regional	Zonal Only
Performance Model	Dependent on Disk Size (Bigger disk = Faster)	Provisioned (You pay specifically for the IOPS/Throughput you define)
Min Size	10 GB	4 GB
Use Case	General purpose, Web Apps, Standard DBs	High-end Enterprise DBs, High Throughput Analytics

4. Project Task: Resize & File System Expansion

Scenario: You create a VM with an extra data disk. You decide later that the disk is too small and increase the size in the Google Cloud Console.

The Problem:

Just clicking "Resize" in the console (Infrastructure level) does not automatically make the space available to the Operating System.

The Solution (Step-by-Step):

1. **GCP Console:** Edit the disk and increase size (e.g., 100GB ==> 200GB).
2. **OS Level (Inside VM):** The OS still sees the old partition size. You must manually expand it.
 - *Command 1: growpart* (Expands the partition table).
 - *Command 2: resize2fs* (Extends the file system to fill the partition).
3. **Result:** No reboot is required. The space becomes instantly available.

lsblk

```
sudo growpart /dev/sda 1
```

```
sudo resize2fs /dev/sda1
```

```
df -h
```

Custom Images

1. Concepts: Public vs. Custom Images

1.1 Public Images (Base Layer):

- When we create a VM, we start with a "Boot Disk."
- Google provides **Public Images** (Vanilla/Bare Metal). These are basic operating systems like Debian, Ubuntu, CentOS, or Windows Server.
- They contain only the OS kernel and basic packages.

1.2 Custom Images (Golden Images):

- **The Problem:** Installing Java, Python, and Web Servers manually on every single VM is slow and error-prone.
- **The Solution:** You create a **Custom Image**.
 1. Start a Base VM with a Public Image.
 2. Install all your software (Java 17, Apache, etc.).
 3. "Bake" this disk into a new Custom Image.
- **The Benefit:** You can now launch 100 VMs from this Custom Image, and they will all have Java/Apache pre-installed instantly.

1.3 How to Create a Custom Image (The Process):

1. **Configure:** Set up your "Golden VM" with all necessary software.
2. **Stop:** Stop the VM to ensure data consistency (Filesystem flush).
3. **Create:** Run the command to create an image from that VM's boot disk.
 - *Note:* You can also create images from Snapshots or other Images.

2. Image Management Features

2.1 Global Availability:

- Images are **Global Resources**.
- You can create an image in Project A (USA) and use it to launch a VM in Project B (Singapore).
- **Requirement:** The user in Project B needs the **Compute Image User** role (`roles/compute.imageUser`) on Project A.

2.2 Image Families:

- A "Family" is a pointer to the **latest, non-deprecated** version of an image.
- *Example:* You name your family `web-server-family`.
 - You create `image-v1`.
 - You create `image-v2`.
- If a script requests `web-server-family`, Google automatically gives them `image-v2`.

2.3 Image States (Lifecycle):

- **Active:** The image is healthy and ready for use.
- **Deprecated:** A warning state. You can still create VMs from it, but users get a warning message. (Used to phase out old versions).
- **Obsolete:** You cannot create new VMs from this image. It is effectively retired.
- **Deleted:** The image is permanently removed.

2.4 Moving Zones:

- You **cannot change the zone** of a running VM.
 - **Workaround:** Create a Custom Image (or Snapshot) of the VM → Create a **new VM** in the new zone using that Image.
-

3. Task Execution (CLI Commands)

Here is the step-by-step execution of your scenario.

Step 1: Create Base VM & Install Software

Bash

```
# A. Create the Base VM
gcloud compute instances create base-vm \
--zone=us-central1-a \
--machine-type=e2-medium \
--image-family=debian-11 \
--image-project=debian-cloud
```

```
# B. Login to install software
```

```
# (Run these inside the VM)
# sudo apt update -y
# sudo apt install openjdk-17-jdk apache2 -y
# exit
```

Step 2: Create Custom Image (Image 1)

Note: We must stop the VM first to ensure the image is clean.

Bash

```
# 1. Stop the VM
gcloud compute instances stop base-vm --zone=us-central1-a

# 2. Create Image1 from the VM's disk
gcloud compute images create image-1 \
--source-disk=base-vm \
--source-disk-zone=us-central1-a \
--family=my-web-app
```

Step 3: Create VM in Singapore using Image 1

Notice we don't need to install Java/Apache. It's already there.

Bash

```
gcloud compute instances create sg-vm \
--zone=asia-southeast1-a \
--image=image-1
```

Step 4: Update Base, Create Image 2, Deprecate Image 1

We need to add Docker/Git to the original base VM and make a new version.

Bash

```
# 1. Start the base VM again
gcloud compute instances start base-vm --zone=us-central1-a
```

```
# 2. SSH and Install Docker/Git
# (Run inside VM)
# sudo apt install git docker.io -y
# exit
```

```
# 3. Stop the VM again
gcloud compute instances stop base-vm --zone=us-central1-a
```

```
# 4. Create Image2 (The newer version)
gcloud compute images create image-2 \
--source-disk=base-vm \
--source-disk-zone=us-central1-a \
--family=my-web-app
```

```
# 5. Deprecate Image1 (Mark it as old)
gcloud compute images deprecate image-1 \
--state=DEPRECATED \
--replacement=image-2
```

Step 5: Create VM with Custom Image 2

We can point to the specific image name OR the family. Since `image-1` is deprecated, the family now points to `image-2`.

Bash

```
gcloud compute instances create new-feat-vm \
--zone=us-central1-a \
--image-family=my-web-app
```

Step 6: Reactivate Image 1 (Un-deprecate)

We realized `image-1` was actually good and we want to remove the warning. We change the state back to ACTIVE.

Bash

```
gcloud compute images deprecate image-1 --state=ACTIVE
```

(Note: Although the command is called `deprecate`, setting state to `ACTIVE` effectively un-deprecates it).

Snapshots

1. Compute Engine Snapshots: Core Concepts

- **Definition:** Snapshots are the primary mechanism for backing up Persistent Disks (PDs). They are used for **High Availability (HA)** and **Disaster Recovery (DR)**.
- **Incremental Backups:**
 - Snapshots are **incremental** by default.
 - *Snapshot 1*: Contains all data on the disk (Full Backup).
 - *Snapshot 2*: Contains **only** the data that changed since Snapshot 1 (Delta).
 - *Benefit*: This saves storage costs and reduces backup time.
- **Global Access:** unlike the Disks themselves (which are Zonal or Regional), Snapshots are global resources. A snapshot taken in `us-central1` can be used to restore a VM in `asia-southeast1`.
- **Application:** Applicable to Standard PD, Balanced PD, SSD PD, and Hyperdisk.

2. Types of Snapshots

When creating a snapshot, you must choose the type based on your recovery needs:

1. **Standard Snapshot:**
 - Best for Disaster Recovery (DR).
 - Stored in a separate location from the disk (Geo-redundant).
2. **Instant Snapshot:**
 - Best for Rapid Restoration.
 - Stored in the **same** location/disk hardware as the VM.
 - Very fast recovery but **not** good for DR (if the zone fails, you lose the snapshot too).
3. **Archive Snapshot:**
 - Best for Compliance/Long-term storage.
 - Stored in a separate location.
 - Cheapest storage cost, but retrieval is slower.

3. Snapshot Storage Locations

- **Regional:** Stored in one specific region (Cheaper, data stays within region).
- **Multi-Regional:** Stored across a geo-location like the US or `EU` (Higher availability, survives total region failure).

4. How Snapshot Deletion Works (The "Chain" Logic)

Snapshots form a chain. If you delete a snapshot in the middle of the chain, you do **not** lose data required for later snapshots.

- **Scenario:** You have Snap A (10GB) -> Snap B (Change: 2GB) -> Snap C (Change: 1GB).
- **Action:** You delete Snap B.
- **Result:** The unique data blocks from Snap B are **moved/merged** into Snap C. Snap C grows larger, and the chain remains unbroken. You can still restore Snap C perfectly.

5. Lab Scenario: Understanding Incremental Sizes

Here is the breakdown of the scenario you described:

1. **Initial State:** VM created with 10GB Disk.
2. **Snapshot 1:** You download a 1GB file.
 - *Snapshot Size:* ~1.93GB (Compressed OS + 1GB file). This is the baseline.
3. **Snapshot 2:** You modify/add 500MB of data.
 - *Snapshot Size:* ~566MB (Only the changes are stored).
4. **Snapshot 3:** You modify/add 200MB of data.
 - *Snapshot Size:* ~207MB (Only the small change is stored).
5. **Snapshot 4 (New Region):** You take a snapshot of this same disk but store it in **asia-southeast1**.
 - **Snapshot Size: 2.72GB (Full Size).**
 - **Reason:** Snapshots are only incremental **per storage location**. Since **asia-southeast1** had no previous history of this disk, it must take a full backup again.

6. Instance Templates

- **Definition:** A blueprint for creating VMs. It defines the machine type, boot disk image, labels, startup scripts, and service accounts.
- **Use Case:** required for creating **Managed Instance Groups (MIGs)** (Autoscaling groups).
- **Scope:** Instance Templates are **Global** resources. You can use one template to launch VMs in **us-central1** and **eu-west1**.

7. Gcloud Commands (CLI)

Here are the commands for Manual Snapshots and Scheduled Snapshots.

A. Manual Snapshots (On-Demand)

Use this when you want to take a backup immediately.

Bash

```
# Syntax:  
gcloud compute snapshots create [SNAPSHOT_NAME] --source-disk [DISK_NAME]  
--snapshot-type [STANDARD|ARCHIVE]
```

```
# Example: Create a snapshot of 'disk-1' in 'us-central1-a'  
gcloud compute snapshots create snap-manual-v1 \  
--source-disk=disk-instance1 \  
--source-disk-zone=us-central1-a \  
--storage-location=us-central1 \  
--description="Manual backup before upgrade"
```

B. Scheduled Snapshots (Automated)

To take a snapshot every hour automatically, we use **Resource Policies**.

Step 1: Create the Schedule Policy

This creates a rule to take a snapshot every hour, keep it for 10 days, and store it in the US multi-region.

Bash

```
gcloud compute resource-policies create snapshot-schedule policy-hourly-backup \
--description="Hourly backup policy" \
--max-retention-days=10 \
--start-time=00:00 \
--hourly-schedule=1 \
--region=us-central1 \
--on-source-disk-delete=apply-retention-policy \
--snapshot-labels=env=dev \
--storage-location=US
```

Regional

```
gcloud beta compute resource-policies create snapshot-schedule snapshot-1 \
--description="This is a scheduled snapshot for instance-1" \
--max-retention-days=10 \
--start-time 15:00 \
--hourly-schedule 1 \
--region us-central1 \
--on-source-disk-delete apply-retention-policy \
--snapshot-labels env=dev \
--storage-location US
```

Step 2: Attach Policy to Disk

The policy does nothing until you attach it to a specific VM disk.

Bash

```
gcloud compute disks add-resource-policies disk-instance1 \
--resource-policies=snapshot-1 \
--zone=us-central1-a
```

Step 3: Verify

To check if the policy is attached:

Bash

```
gcloud compute disks describe disk-instance1 \
--zone=us-central1-a \
--format="value(resourcePolicies)"
```

Instance Groups

1. Instance Templates (The Blueprint)

- **Definition:** A configuration resource that acts as a "blueprint" or "cookie cutter" for virtual machines.
- **Purpose:** It defines the machine type, boot disk image, labels, startup scripts, and network settings.
- **Usage:** You cannot "run" a template; you use it to create identical VMs (specifically for Managed Instance Groups).

2. Instance Groups (Overview)

- **Definition:** A collection of VM instances that you can manage as a single entity.
- **Types:**
 1. **Managed Instance Groups (MIGs):** For identical, automated, scalable apps.
 2. **Unmanaged Instance Groups (UMIGs):** For different, manually managed apps.

3. Unmanaged Instance Groups (UMIGs)

Concept:

A group of dissimilar (non-identical) VMs that are manually created and managed.

Key Characteristics:

- **Heterogeneous:** You can group an **e2-micro** and an **e2-medium** together.
- **Manual Lifecycle:** The group does **not** create the VMs for you. You must create the VMs first, then add them to the group.
- **Constraints:** All VMs in the group must be in the **Same Zone** and the **Same VPC Network**.
- **No Automation:** UMIGs do **not** support Auto-scaling or Auto-healing. If a VM dies, it stays dead.

Use Case:

- Used solely to group existing VMs to put them behind a **Passthrough Load Balancer**.
- Legacy applications where VMs are unique "pets" rather than disposable "cattle".

4. Managed Instance Groups (MIGs)

Concept:

A group of identical VMs created automatically from an Instance Template.

Key Characteristics:

- **Homogeneous:** All VMs are clones of the template.
- **Automated:** Supports **Auto-scaling** (add/remove VMs based on load) and **Auto-healing** (recreate failed VMs).

- **Deployment:** Can be **Zonal** (all VMs in `us-central1-a`) or **Regional** (VMs spread across `us-central1-a, b, c`).
 - **Modes:**
 - *Stateless*: Web servers (replaceable).
 - *Stateful*: Databases (preserve disk/IP).
-

5. Scaling Concepts

A. Horizontal Scaling (Scaling Out / In)

- **Action:** Changing the **number** of VMs.
- **Scale Out:** Adding more VMs (e.g., 2 VMs => 5 VMs) to handle increased traffic.
- **Scale In:** Removing VMs (e.g., 5 VMs => 2 VMs) when traffic drops.
- **Example:** Google Cloud Spanner, MIGs.

B. Vertical Scaling (Scaling Up / Down)

- **Action:** Changing the **size** of the VM.
- **Scale Up:** Increasing power (e.g., `e2-medium` => `e2-highcpu`).
- **Scale Down:** Decreasing power.
- **Example:** Cloud SQL (requires restart), Resizing a VM.

6. Lab Task: Creating Unmanaged Instance Groups

Step 1: Prerequisite (Create VMs)

- Create **VM-1** (e.g., `e2-medium` with a "Blue" startup script).
- Create **VM-2** (e.g., `e2-micro` with a "Green" startup script).
- **Note:** Ensure both are in the **same Zone** (e.g., `us-central1-a`) and **same Network**.

Step 2: Create the Group

- Go to **Compute Engine => Instance Groups**.
- Click **Create Instance Group**.
- Select **New Unmanaged Instance Group**.

Step 3: Configuration

- **Name:** `my-unmanaged-group`.
- **Region/Zone:** Must match the VMs created in Step 1.
- **Network:** Must match the VMs.
- **Port Mapping:** Specify the port your application listens on (e.g., Port 80 for Apache, 8080 for Java). This helps the Load Balancer know where to send traffic.

Step 4: Add Instances

- Select **VM-1** and **VM-2** from the dropdown list.

Step 5: Deletion Behavior

- If you delete the **Unmanaged Instance Group**, the group is destroyed, but the **VMs remain running**. (Because the group didn't create them, it doesn't delete them).
-

CLI Command for Unmanaged Group

Since you asked for Gcloud commands previously, here is how to do this via CLI:

Bash

```
# 1. Create the Unmanaged Instance Group  
gcloud compute instance-groups unmanaged create my-umig \  
--zone=us-central1-a
```

```
# 2. Add the existing VMs to the group
```

```
gcloud compute instance-groups unmanaged add-instances my-umig \  
--zone=us-central1-a \  
--instances=vm-blue,vm-green
```

```
# 3. Set the named port (Port Mapping)
```

```
gcloud compute instance-groups unmanaged set-named-ports my-umig \  
--zone=us-central1-a \  
--named-ports=http:80
```

7. The Workflow: Creating a Managed Instance Group (MIG)

To create a MIG, you must follow a strict order of operations because MIGs require a blueprint.

Step 1: Create the Instance Template (The Blueprint)

- You cannot create a MIG without this.
- Define the Machine Type ([e2-medium](#)), Image ([Debian](#)), Disks, and Startup Scripts.
- Note: Templates are global and immutable (cannot be changed once created).

CLI Command:

Bash

```
gcloud compute instance-templates create my-template \
--machine-type=e2-medium \
--image-family=debian-11 \
--image-project=debian-cloud
```

Step 2: Create the Managed Instance Group

- **Type:** Select **Stateless** (for web servers) or **Stateful** (for databases).
- **Location:**
 - **Zonal:** (Single zone, lower cost, lower availability).
 - **Regional:** (Multi-zone, high availability, survives zone failure).
- **Size:** Define the initial number of instances (e.g., 2).

2. Health Checks & Auto-Healing

MIGs don't just "keep VMs running"; they keep "healthy applications running."

2.1 The Concept:

- The MIG periodically "pings" your application.
- If the app fails to respond, the MIG **deletes** that VM and **recreates** a fresh one automatically. This is called **Auto-Healing**.

2.2 Firewall Rules (Crucial Step):

- Your VMs must allow traffic from Google's Health Check probes. If you block these IPs, Google thinks your VMs are dead and will delete them in a loop.
- **Allow these Source Ranges:**
 - [35.191.0.0/16](#)
 - [130.211.0.0/22](#)

2.3 Health Check Configuration:

- **Scope:** Global or Regional (Match your Load Balancer type).
- **Port:** Where your app lives (e.g., Django on [8080](#), Nginx on [80](#)).

- **Parameters:**
 - **Check Interval:** How often to check? (e.g., Every 5 seconds).
 - **Timeout:** How long to wait for a reply? (e.g., 5 seconds).
 - **Healthy Threshold:** How many successes to be "Green"? (e.g., 2 successes).
 - **Unhealthy Threshold:** How many failures to be "Red"? (e.g., 2 failures => Delete VM).
-

3. Autoscaling Strategies

Autoscaling adds or removes VMs dynamically based on load.

3.1 Basic Settings:

- **Min Instances:** (e.g., 3) Never go below this, ensuring baseline availability.
- **Max Instances:** (e.g., 8) Never go above this to control costs.

3.2 Scaling Signals (Triggers):

1. **CPU Utilization (Most Common):**
 - *Target:* 60%.
 - *Logic:* If average CPU across VMs > 60%, add a VM. If < 60%, remove a VM.
2. **Load Balancing Serving Capacity:**
 - Scale based on Requests Per Second (RPS).
3. **Cloud Monitoring Metric:**
 - Scale based on custom metrics (e.g., "Number of messages in Pub/Sub queue").

3.3 Advanced Scaling Modes:

- **Predictive Autoscaling:**
 - Uses Machine Learning to analyze the last 3 days of traffic.
 - *Logic:* "Traffic usually spikes at 9 AM, so let's boot up VMs at 8:50 AM."
 - *Best for:* Predictable daily patterns.
- **Scheduled Autoscaling:**
 - You manually define the schedule.
 - *Logic:* "I know we have a sale on Friday at 6 PM. Set min instances to 50 at 5 PM."

4. Initialization & Cool-Down Periods (The "Warm-up" Time)

This is the most common reason for "**Autoscaling Loops**" (where VMs keep restarting).

- **The Issue:** A VM boots up in 10 seconds, but your Java/Django app might take 2 minutes to actually start listening on Port 8080.
- **Initialization Period (Cool Down):**
 - You tell GCP: "**Please wait 120 seconds after boot before checking health.**"

- If you set this too short (e.g., 10s), GCP checks at 11s, sees the app is dead, deletes the VM, and tries again. The VM never gets a chance to live.
 - *Recommendation:* Time your startup script exactly and add a buffer (e.g., if it takes 2 mins, set this to 180 seconds).
-

5. CLI Cheat Sheet for MIGs

Create a Health Check:

Bash

```
gcloud compute health-checks create http my-health-check \
--port=8080 \
--check-interval=5s \
--timeout=5s \
--unhealthy-threshold=2 \
--healthy-threshold=2
```

Create a Firewall Rule for Health Checks:

Bash

```
gcloud compute firewall-rules create allow-health-check \
--allow=tcp:8080 \
--source-ranges=35.191.0.0/16,130.211.0.0/22
```

Create the MIG (Zonal):

Bash

```
gcloud compute instance-groups managed create my-app-mig \
--template=my-template \
--size=3 \
--zone=us-central1-a \
--health-check=my-health-check \
--initial-delay=120
```

(Note: *--initial-delay* is the Initialization Period mentioned in point 4).

Configure Autoscaling:

Bash

```
gcloud compute instance-groups managed set-autoscaling my-app-mig \
--zone=us-central1-a \
--max-num-replicas=8 \
--min-num-replicas=3 \
--target-cpu-utilization=0.60
```

Summary

Resource	Scope (Location)	Types / Variants	When to use which Type (Real Life)	Key Constraints / "Gotchas"
Disks	Zonal (Standard)Regional (HA)	1. pd-standard: HDD (Spinning disk) 2. pd-balanced: SSD (General) 3. pd-ssd: SSD (Fast) 4. pd-extreme: SSD (Top Tier) 5. Hyperdisk: Next-Gen (Throughput/IOPS) 6. Local SSD: Ephemeral	pd-standard: Cold logs, backups, rarely accessed data. pd-balanced: Boot disks, Web servers, most Line-of-Business apps (Best Value). pd-ssd: Production Databases (MySQL, Postgres). Hyperdisk: SAP HANA, Oracle, Massive Hadoop jobs. Local SSD: Temporary Cache (Redis), Scratch space (Data lost on stop).	You cannot change a Local SSD to a Persistent Disk later. Hyperdisk requires specific machine types (like C3).
Snapshots	Global Access(Stored in Region or Multi-Region)	1. Standard: Separate storage. 2. Instant: Local storage. 3. Archive: Cold storage.	Standard: Daily backups for Disaster Recovery (DR). Instant: Take this 5 mins before a risky OS upgrade (Restores in seconds). Archive: "Keep data for 7 years" for legal compliance (Cheapest).	Instant Snapshots are expensive and lost if the zone fails. Always move important Instant snaps to Standard eventually.
Images	Global	1. Public: Provided by Google (Debian, Ubuntu, Windows). 2. Custom: Created by You.	Public: When starting a fresh experiment or learning. Custom: Production scaling. You pre-install Java, Agents, and Firewall rules so new VMs boot up ready-to-work.	You cannot change the OS family (e.g., Linux to Windows) of an image.
Instance Templates	Global	Deterministic: Specific inputs.(Not many "types", but "Versions")	Daily Ops: You update the template version (v1 → v2) when you patch the OS. The MIG then rolls out v2 to all VMs.	You cannot edit a template. You must copy → modify → save as new.
Managed Instance Groups (MIG)	Zonal or Regional	1. Stateless: Web/App servers. 2. Stateful: Databases (Preserves disk names).	Stateless: 99% of use cases. Web servers that can be deleted/recreated anytime. Stateful: Kafka or Elasticsearch nodes where Disk ID matters.	Regional MIGs cost more (cross-zone traffic) but survive a total zone outage. Zonal MIGs die if the Zone dies.
Unmanaged Instance Groups	Zonal	N/A	Legacy: You manually built 3 Pet VMs (not from a template) and just need to put a Load Balancer in front of them.	Avoid it if possible. No Auto-healing (if VM crashes, it stays crashed).

Load Balancers

Load Balancers

1. What is a Load Balancer?

- **Definition:** A service that distributes incoming network traffic across multiple backend targets (instances).
- **Purpose:** Ensures **High Availability (HA)** and reliability for applications. It acts as the "Single Point of Entry" for users.
- **Backend Targets:** Traffic can be distributed to VMs (Compute Engine), Containers (GKE Pods), App Engine, Cloud Run, or Cloud Functions.
- **Cloud Native:** Unlike traditional hardware load balancers (like F5 used on-premise), Google Cloud Load Balancers are software-defined. They require no physical hardware management and scale from zero to full capacity automatically.

2. Key Features

- **Global Anycast IP:**
 - A single IP address is used globally.
 - **Low Latency Routing:** It uses "Anycast" technology to route users to the nearest Google data center.
 - *Example:* A user in London is automatically routed to a UK region backend, while a user in the USA is routed to a US region backend, all using the exact same IP address.
- **Proxy-Based Architecture (SSL Termination):**
 - Client traffic terminates at the Load Balancer (Frontend).
 - The Load Balancer handles the SSL/TLS decryption.
 - A *new* connection is established from the Load Balancer to your Backend instances.

3. Types of Load Balancers

A. Based on Traffic Type (OSI Layer)

1. **Application Load Balancer (Layer 7):**
 - Handles HTTP/HTTPS traffic.
 - Ideal for Web Applications.
 - Supports smart routing (URL paths, Host headers).
2. **Network Load Balancer (Layer 4):**
 - Handles TCP/UDP traffic.
 - Ideal for non-HTTP traffic, gaming, or raw throughput.

B. Based on Scope (Geography)

1. **Global Load Balancer:**
 - Distributes traffic to backends in **multiple regions** (e.g., US, Europe, Asia).
 - Used when your users are distributed worldwide.
 - Requires a Global Anycast IP.
2. **Regional Load Balancer:**
 - Distributes traffic to backends within a **single region** (e.g., `us-central1`).
 - Used when all users and data residency requirements are local.
 - Requires a Regional IP.

C. Based on Visibility

1. **External Load Balancer:**
 - Accessible from the Internet (Public IP).
2. **Internal Load Balancer:**
 - Accessible only within the VPC network (Private IP).
 - Used for internal communication between microservices (e.g., Frontend App talking to Backend Database).

4. Load Balancer Components (The "Flow")

Traffic flows through these logical components in order:

1. Frontend (The Entry Point)

- Defines the IP Address (Global or Regional).
- Defines the Protocol (HTTP, HTTPS, TCP, UDP).
- **Session Affinity:** Configured here.
 - *Concept:* Ensures a user is "stuck" to the same backend VM for the duration of their session.
 - *Use Case:* Shopping carts or banking transactions where the session state is stored locally on one server.

2. Host and Path Rules (Routing)

- Primarily for Application Load Balancers (Layer 7).
- **Host-Based Routing:** Routes traffic based on the domain name.
 - *Example:* `shop.xyz.com` ==> Shop Backend; `blog.xyz.com` ==> Blog Backend.
- **Path-Based Routing:** Routes traffic based on the URL path.
 - *Example:* `xyz.com/images` ==> Backend Bucket; `xyz.com/api` ==> App Backend.
- **Rewrite Rules:** Modifies the URL before sending it to the backend.
 - *Example:* User requests `xyz.com/images` ==> LB rewrites to `backend.com/static-assets`.

3. Backends (The Destination)

- **Backend Service:** Groups of computing resources (Instance Groups, NEGs).
 - Includes **Health Checks** to ensure traffic only goes to healthy instances.
- **Backend Bucket:** A Cloud Storage bucket used to serve static content (images, videos) directly via the Load Balancer (cheaper and faster than using VMs).

5. Summary Flowchart

**User (DNS) ==> Frontend (IP/SSL Termination) ==> Host/Path Rules (Logic)
==> Backend Service (Health Checks) ==> VMs / Pods**

6. Quick Decision Matrix

Scenario	Recommended Load Balancer
Web App with Global Users	Global External Application LB
Database/Internal API	Regional Internal Application LB
High throughput non-HTTP traffic	Network Load Balancer
Serving Static Images/Videos	Application LB with Backend Bucket

Load Balancers - 1

This architecture allows a user in Singapore to hit the Singapore servers, and a user in the US to hit the US servers, all using **one single IP address**.

Phase 1: Network Foundation

Before creating VMs, we need the network plumbing.

1. Create the VPC

- Create a custom VPC network.

2. Create Subnets (Dual Region)

- **Subnet A:** Region `us-central1` (e.g., `10.0.1.0/24`)
- **Subnet B:** Region `asia-southeast1` (e.g., `10.0.2.0/24`)

3. Configure Firewall Rules

We need two specific rules:

- **Allow-SSH-Web:** Allow `tcp:22` (SSH) and `tcp:80` (Web) from `0.0.0.0/0`.
- **Allow-Health-Checks (Crucial):** The Load Balancer (Google) probes your VMs to see if they are alive. You **must** allow traffic from these Google-owned IP ranges:
 - `35.191.0.0/16`
 - `130.211.0.0/22`

Phase 2: Compute Infrastructure (The Backends)

We need two Instance Templates because templates define the **Subnet**. Since Subnets are regional, Template A must point to Subnet A, and Template B to Subnet B.

4. Define Startup Script

- Prepare your script (e.g., `apt update && apt install apache2...`).

5. Create Instance Templates

- **Template-US:** Uses `us-central1` subnet. Includes startup script.
- **Template-Asia:** Uses `asia-southeast1` subnet. Includes startup script.

6. Create Managed Instance Groups (MIGs)

- **MIG-US:**
 - Region: `us-central1`
 - Template: `Template-US`
- **MIG-Asia:**
 - Region: `asia-southeast1`
 - Template: `Template-Asia`

7. Set Named Ports (Critical Step)

- Load Balancers do not talk to "Port 80" directly; they talk to a "Service Name" (e.g., `http`).
 - You must map the name to the port on **both** MIGs.
 - **CLI Command:** `gcloud compute instance-groups set-named-ports MIG-US --named-ports http:80`
-

Phase 3: Load Balancer Configuration

Now we tie everything together.

8. Reserve Global Static IP

- Reserve a Global External IP Address so your website IP doesn't change.

9. Create the Load Balancer

- Go to Network Services ==> Load Balancing ==> Create Load Balancer.
- **Type:** Application Load Balancer (HTTP/HTTPS).
- **Visibility:** Public facing (External).
- **Mode:** Global (Single Anycast IP).

10. Configure Backend Configuration

- **Create Backend Service:** This is the logical group that handles traffic.
- **Protocol:** HTTP.
- **Named Port:** `http` (Must match what you set in Step 7).
- **Add Backends:** Add **both** `MIG-US` and `MIG-Asia` to this service.
- **Balancing Mode:** Set to **Utilization (80%) or Rate (RPS)**.
 - **Logic:** If US servers hit 80% CPU, overflow traffic to Asia.
- **Health Check:** Create a **Global Health Check** (TCP or HTTP) here.

11. Advanced Backend Settings (Session Affinity)

- Inside the Backend Service, under "Advanced Configuration," you can enable **Session Affinity**.
- **Setting:** **Generated Cookie or Client IP**.
- **Purpose:** Ensures a user stays connected to the same VM for their entire session (useful for shopping carts).

12. Host and Path Rules (Routing Rules)

- **Simple Routing (Default):**
 - **Host:** `*` (Any domain).
 - **Path:** `/*` (Any path).
 - **Backend:** `Your-Backend-Service`.
 - **Meaning:** Send **all** traffic to your backends.
- **Advanced Routing (Optional):**

- You can send `/video` to a different backend or `/static` to a Bucket.

13. Configure Frontend

- **Protocol:** HTTP (or HTTPS if you have a cert).
- **IP Address:** Select the Static IP you reserved in Step 8.
- **Port:** 80.

14. Review and Create

- Click Create. It takes ~5-10 minutes for the Global Anycast IP to propagate worldwide.

Phase 4: DNS Integration (GoDaddy & Cloud DNS)

Now that your Load Balancer is live, we need to point your domain name (e.g., [example.com](#)) to it. We will use **Google Cloud DNS** to manage the records, but we must first tell **GoDaddy** to trust Google.

15. Retrieve Your Load Balancer IP

Before configuring DNS, you need the specific IP address that listens for traffic.

1. Go to **Network Services ==> Load Balancing**.
2. Click on the Load Balancer you just created (from Phase 3).
3. Look for the **Frontend** section.
4. Copy the **IP Address** (e.g., [34.102.x.x](#)). *Keep this handy, you will need it for the A Record.*

16. Create a Cloud DNS Zone in GCP

This sets up a "bucket" in Google Cloud to hold your DNS records.

1. Go to **Network Services ==> Cloud DNS**.
2. Click **Create Zone**.
3. **Zone Type:** Public.
4. **Zone Name:** Enter a unique ID (e.g., [my-website-zone](#)).
5. **DNS Name:** Enter your actual domain name bought on GoDaddy (e.g., [example.com](#)).
6. Click **Create**.
7. **Crucial Step:** Once created, look for the **NS (Name Server)** record type. You will see 4 servers listed (e.g., [ns-cloud-a1.googledomains.com](#), [ns-cloud-b1...](#), etc.). **Keep this tab open.**

17. Update Name Servers in GoDaddy

We now need to tell GoDaddy to let Google manage the traffic.

1. Log in to your **GoDaddy** account and go to your **Domain Portfolio**.
2. Select your domain and scroll down to **DNS** (or select "Manage DNS").
3. Look for the **Nameservers** section.
4. Click **Change**.
5. Select "**I'll use my own nameservers**" (Advanced).

6. **Copy and Paste** the 4 Name Servers from your GCP Cloud DNS console (Step 16) into the GoDaddy fields.
 - *Line 1: ns-cloud-a1.googledomains.com*
 - *Line 2: ns-cloud-b1.googledomains.com*
 - *(Add lines for c1 and d1 as well).*
7. Click **Save**. *Note: It may take minutes to hours for this switch to propagate globally.*

18. Configure DNS Records in GCP

Now that GoDaddy is pointing to Google, go back to the **GCP Cloud DNS** console to create your specific records.

A. Create the A Record (Root Domain)

This points example.com directly to your Load Balancer.

1. Click **Add Standard**.
2. **DNS Name:** Leave blank (this represents the root **@**).
3. **Resource Record Type:** Select **A**.
4. **IPv4 Address:** Paste the **Frontend IP Address** you copied in Step 15.
5. Click **Create**.

B. Create the CNAME Record (Subdomain)

This acts as an alias. We will point www to the root domain.

1. Click **Add Standard**.
2. **DNS Name:** Type **www**.
3. **Resource Record Type:** Select **CNAME**.
4. **Canonical Name:** Type your root domain (e.g., **example.com**).
5. Click **Create**.

C. Create a Wildcard Record (Catch-All)

This ensures that if someone types a random subdomain (e.g., blog.example.com or shop.example.com) that doesn't exist, they still get sent to your website.

1. Click **Add Standard**.
2. **DNS Name:** Type ***** (asterisk).
3. **Resource Record Type:** Select **A**.
4. **IPv4 Address:** Paste the **Frontend IP Address** (same as the A record).
5. Click **Create**.

Summary of Records:

- **example.com => A ==> 34.xxx.xxx.xxx** (LB IP)
- **www.example.com ==> CNAME ==> example.com**
- ***.example.com ==> A ==> 34.xxx.xxx.xxx** (LB IP)

Gcloud Commands

Phase 1: Network Foundation

1. Create a Custom VPC

Bash

```
gcloud compute networks create lb-vpc --subnet-mode=custom
```

2. Create the Two Subnets

Bash

```
# Subnet for US Central
gcloud compute networks subnets create subnet-us \
--network=lb-vpc \
--region=us-central1 \
--range=10.0.1.0/24
```

```
# Subnet for Singapore
gcloud compute networks subnets create subnet-asia \
--network=lb-vpc \
--region=asia-southeast1 \
--range=10.0.2.0/24
```

3. Create Firewall Rules

Bash

```
# Allow SSH and HTTP from anywhere
gcloud compute firewall-rules create allow-ssh-web \
--network=lb-vpc \
--action=allow \
--direction=INGRESS \
--rules=tcp:22,tcp:80,icmp \
--source-ranges=0.0.0.0/0
```

```
# Allow Google Health Checks (Mandatory IP ranges)
gcloud compute firewall-rules create allow-health-check \
--network=lb-vpc \
--action=allow \
--direction=INGRESS \
--rules=tcp:80 \
--source-ranges=130.211.0.0/22,35.191.0.0/16
```

Phase 2: Compute Infrastructure (Instance Groups)

4. Create Instance Templates We use a simple startup script to install Apache so the Health Check passes.

Bash

```
# Template for US Region
gcloud compute instance-templates create template-us \
--machine-type=e2-micro \
--image-family=debian-11 \
--image-project=debian-cloud \
--network=lb-vpc \
--subnet=subnet-us \
--region=us-central1 \
--tags=allow-health-check \
--metadata-from-file=startup-script=startup-script.sh
```

```
# Template for Asia Region
```

```
gcloud compute instance-templates create template-asia \
--machine-type=e2-micro \
--image-family=debian-11 \
--image-project=debian-cloud \
--network=lb-vpc \
--subnet=subnet-asia \
--region=asia-southeast1 \
--tags=allow-health-check \
--metadata-from-file=startup-script=startup-script.sh
```

```
#!/bin/bash
```

```
# Stop script on error to prevent partial installs
set -e
```

```
# Update and Install Nginx
```

```
apt-get update
apt-get install -y nginx
```

```
# Start and Enable Nginx
```

```
systemctl start nginx
systemctl enable nginx
```

```
# Set Permissions
```

```
chmod -R 755 /var/www/html
```

```
# Get Hostname and IP (awk ensures we only get the first IP)
```

```
HOSTNAME=$(hostname)
IP_ADDR=$(hostname -I | awk '{print $1}')
```

```

# Create the index.html file using a Here-Doc for safety
cat <<EOF > /var/www/html/index.html
<!DOCTYPE html>
<html>
<head>
<style>
  body { font-family: Arial, sans-serif; color: #333333; margin: 0; padding: 0; display: flex; justify-content: center; align-items: center; height: 100vh; background-color: #f2f2f2; }
  h1 { color: #005580; }
  p { font-size: 1.2em; color: #555555; }
  .hostname { color: #ff7500; }
</style>
</head>
<body>
<div>
  <h1>i27academy Google Cloud Platform</h1>
  <p><strong>Server Hostname:</strong> <span class='hostname'>$HOSTNAME</span></p>
  <p><strong>Server IP Address:</strong> $IP_ADDR</p>
</div>
</body>
</html>
EOF

```

5. Create Managed Instance Groups (MIGs) before that create health check in Step 8

Bash

```

# MIG in US
gcloud compute instance-groups managed create mig-us \
--template=template-us \
--size=2 \
--zone=us-central1-a \
--health-check=http-detailed-check \
--initial-delay=300

```

MIG in Asia

```

gcloud compute instance-groups managed create mig-asia \
--template=template-asia \
--size=2 \
--zone=asia-southeast1-a \
--health-check=http-detailed-check \
--initial-delay=300

```

6. Set Named Ports *This maps the logical name "http" to port 80 on the VMs.*

Bash

```
gcloud compute instance-groups set-named-ports mig-us \
--named-ports http:80 \
--zone=us-central1-a
```

```
gcloud compute instance-groups set-named-ports mig-asia \
--named-ports http:80 \
--zone=asia-southeast1-a
```

Phase 3: Load Balancer Components

7. Reserve a Global Static IP

Bash

```
gcloud compute addresses create lb-ipv4-1 \
--ip-version=IPV4 \
--global
```

(Note: Run `gcloud compute addresses list` to see the IP you got).

8. Create a Health Check

Bash

```
gcloud compute health-checks create http http-detailed-check \
--port=80 \
--check-interval=5s \
--timeout=5s \
--healthy-threshold=2 \
--unhealthy-threshold=3 \
--global \
--description="Detailed health check for backend API"
```

9. Create the Backend Service

Bash

```
gcloud compute backend-services create web-backend-service \
--global \
--protocol=HTTP \
--port-name=http \
--health-checks=http-detailed-check \
```

```
--load-balancing-scheme=EXTERNAL_MANAGED \
--locality-lb-policy=ROUND_ROBIN \
--connection-draining-timeout=300 \
--session-affinity=NONE \
--timeout=30
```

10. Add Instance Groups to Backend Service

Bash

```
# Add US Group
gcloud compute backend-services add-backend web-backend-service \
--instance-group=mig-us \
--instance-group-zone=us-central1-a \
--global
```

```
# Add Asia Group
```

```
gcloud compute backend-services add-backend web-backend-service \
--instance-group=mig-asia \
--instance-group-zone=asia-southeast1-a \
--global
```

11. Create URL Map (Routing Rules) *This directs all incoming traffic to our backend service.*

Bash

```
gcloud compute url-maps create web-map-http \
--default-service web-backend-service
```

12. Create Target HTTP Proxy *This connects the URL Map to the Frontend.*

Bash

```
gcloud compute target-http-proxies create http-lb-proxy \
--url-map web-map-http
```

13. Create Global Forwarding Rule (The Frontend) *This ties the IP address, Port 80, and the Proxy together.*

Bash

```
gcloud compute forwarding-rules create http-content-rule \
--address=lb-ipv4-1 \
--global \
--target-http-proxy=http-lb-proxy \
--ports=80
```

Validation

Wait about 5–7 minutes for the global configuration to propagate. Then get your IP:

Bash

```
gcloud compute forwarding-rules describe http-content-rule --global
```

Copy the **IPAddress** and paste it into your browser. You should see "Hello from US-Central" or "Hello from Singapore" depending on where you are!

Phase 4: Cloud DNS Configuration

14. Create the Cloud DNS Managed Zone This creates the container in Google Cloud to manage your domain's records.

Bash

```
gcloud dns managed-zones create gcpstudy-zone \  
  --description="DNS zone for gcpstudycircle" \  
  --dns-name="gcpstudycircle.online." \  
  --visibility=public
```

15. Retrieve Your Load Balancer IP We need the IP address you reserved in **Phase 3, Step 7**. We will store it in a variable called **\$LB_IP** so we don't have to type it manually.

Bash

```
# Capture the IP address into a variable  
LB_IP=$(gcloud compute addresses describe lb-ipv4-1 \  
  --global \  
  --format="value(address)")  
  
# Verify it was captured correctly  
echo "Your Load Balancer IP is: $LB_IP"
```

16. Create DNS Records Now we create the three critical records using the IP we just captured.

A Record (Root Domain): Points **gcpstudycircle.online** directly to the Load Balancer.

Bash

```
gcloud dns record-sets create gcpstudycircle.online. \  
  --zone=gcpstudy-zone \  
  --type=A \  
  --ttl=300 \
```

```
--rrdatas=$LB_IP
```

CNAME Record (www): Points www.gcpstudycircle.online to the root domain.

Bash

```
gcloud dns record-sets create www.gcpstudycircle.online. \
--zone=gcpstudy-zone \
--type=CNAME \
--ttl=300 \
--rrdatas=gcpstudycircle.online.
```

Wildcard A Record (*): Catch-all for any other subdomain (e.g., blog.gcpstudycircle.online) and sends it to the Load Balancer.

Bash

```
gcloud dns record-sets create "*.gcpstudycircle.online." \
--zone=gcpstudy-zone \
--type=A \
--ttl=300 \
--rrdatas=$LB_IP
```

17. Retrieve Name Servers for GoDaddy Google Cloud has now assigned 4 specific Name Servers to your zone. You need to copy these to GoDaddy.

Bash

```
gcloud dns managed-zones describe gcpstudy-zone \
--format="value(nameServers)"
```

(This will output 4 lines looking like ns-cloud-c1.googledomains.com, etc.)

Manual Step: GoDaddy Configuration

The [gcloud](#) commands cannot touch your GoDaddy account, so you must do this part in your browser:

1. **Log in to GoDaddy** and go to the **DNS Management** page for gcpstudycircle.online.
2. Find the **Nameservers** section.
3. Select "**I'll use my own nameservers**".
4. **Copy and Paste** the 4 server names outputted by the command in **Step 17**.

- *Note: Remove the trailing dot . at the end if GoDaddy complains (e.g., paste ns-cloud-c1.googledomains.com instead of ns-cloud-c1.googledomains.com.).*

5. Click **Save**.

It may take **15 minutes to 1 hour** for the DNS changes to propagate globally. Once done, visiting <http://gcpstudycircle.online> will hit your Global Load Balancer, which will route you to the nearest healthy server (US or Asia).

Phase 6: HTTPS Configuration (SSL/TLS)

Prerequisite: Ensure you have completed the **Manual GoDaddy Step** (pointing Name Servers to Google) and wait 15 minutes.

You now have **two options** to secure your Load Balancer. Choose **ONE**.

Option A: Self-Signed Certificate (For Testing)

Use this if you just want to test HTTPS and don't care that the browser shows a "Not Secure / Warning" red screen.

1. Generate the Private Key (Corrected 2028 to standard 2048)

Bash

```
openssl genrsa -out gcpstudycircle.key 2048
```

2. Generate the Certificate Signing Request (CSR)

Bash

```
openssl req -new -key gcpstudycircle.key -out gcpstudycircle.csr \
-subj "/C=US/ST>NewYork/L=Buffalo/O=gcpstudycircle/OU=IT/CN=gcpstudycircle.online"
```

3. Generate the Certificate (CRT) (Corrected -re to -req)

Bash

```
openssl x509 -req -days 365 \
-in gcpstudycircle.csr \
-signkey gcpstudycircle.key \
-out gcpstudycircle.crt
```

4. Upload to Google Cloud Crucial Correction: You mentioned mixing up Key and Crt. Do not mix them up!

- `--certificate = .crt` file (Public)

- `--private-key` = `.key` file (Private)

Bash

```
gcloud compute ssl-certificates create self-signed-cert \
--certificate=gcpstudycircle.crt \
--private-key=gcpstudycircle.key \
--global
```

Option B: Let's Encrypt (For Production)

Use this if you want the "Green Padlock" and valid trust from browsers.

1. Install Certbot

Bash

```
sudo apt-get update
sudo apt-get install -y certbot
```

2. Request Certificate (DNS Challenge)

Bash

```
sudo certbot certonly \
--manual \
--preferred-challenges dns \
-d gcpstudycircle.online \
-d www.gcpstudycircle.online
```

- *Action:* It will give you a **TXT string**. Copy it. Do not press Enter yet.

3. Create Verification DNS Record (Open a new Cloud Shell tab)

Bash

```
gcloud dns record-sets create _acme-challenge.gcpstudycircle.online. \
--zone=gcpstudy-zone \
--type=TXT \
--ttl=60 \
--rrdatas="[PASTE_STRING_FROM_CERTBOT]"
```

4. Finish Certbot

Go back to the first tab and press Enter. It will save files to `/etc/letsencrypt/live/gcpstudycircle.online/`.

5. Upload to Google Cloud

Bash

```
sudo gcloud compute ssl-certificates create lets-encrypt-cert \
--certificate=/etc/letsencrypt/live/gcpstudycircle.online/fullchain.pem \
--private-key=/etc/letsencrypt/live/gcpstudycircle.online/privkey.pem \
--global
```

Final Step: Attach Certificate to Load Balancer

Now that you have uploaded a certificate (either `self-signed-cert` from Option A **OR** `lets-encrypt-cert` from Option B), you must attach it to the proxy.

- 1. Create the HTTPS Proxy** Replace `[CERT_NAME]` with either `self-signed-cert` or `lets-encrypt-cert`.

Bash

```
gcloud compute target-https-proxies create https-lb-proxy \
--url-map=web-map-http \
--ssl-certificates=[CERT_NAME]
```

- 2. Create the Global Forwarding Rule (Port 443)** This opens the secure port on your Load Balancer.

Bash

```
gcloud compute forwarding-rules create https-content-rule \
--address=lb-ipv4-1 \
--global \
--target-https-proxy=https-lb-proxy \
--ports=443
```

Summary of What Happens Now

1. **User** types `https://gcpstudycircle.online`.
2. **DNS** (GoDaddy -> Google Cloud DNS) resolves to your Load Balancer IP.
3. **Load Balancer** sees traffic on Port 443.
4. It uses the **Key** to decrypt the traffic (SSL Offloading).
5. It forwards the request as plain HTTP (Port 80) to your backend servers in US or Asia.
6. The Server responds, LB encrypts it again, and sends it back to the User.

Cloud Storage

1. Object Versioning (Data Protection)

Concept: By default, if you upload `deploy.yaml` and then upload a new version of `deploy.yaml`, the old file is overwritten and lost forever.

- **Without Versioning:** Overwriting/Deleting is permanent.
- **With Versioning:** Overwriting hides the old file (making it "Noncurrent") and puts the new file on top ("Live"). Deleting a file simply marks it as a "Delete Marker," keeping the data in the background.

Key Rules:

1. **Generations:** Every time you overwrite a file, Google assigns it a **Generation Number** (e.g., `1589034...`). This ID allows you to retrieve that specific version later.
2. **Timing:** Versioning only protects files modified **after** you enable it. It cannot recover files deleted before the setting was turned on.
3. **Cost Warning: Be careful.** If you have a 1GB file and you overwrite it 10 times, you are paying for **11GB** of storage (1 Live + 10 Noncurrent versions). You usually pair Versioning with Lifecycle rules to clean up old versions automatically.

2. Lifecycle Management (Automation & Cost)

Concept: Instead of manually deleting or moving files, you set a "Policy" on the bucket. Google scans the bucket and applies actions automatically.

Common Use Case (The Waterfall):

1. **Day 1:** File uploaded (Standard Class - Hot data).
2. **Day 30:** Move to **Nearline** (Cheaper, slightly slower access).
3. **Day 90:** Move to **Coldline/Archive** (Long-term backup).
4. **Day 365:** **Delete** the object (Expire).

Key Rules:

1. **Scope:** Applied at the Bucket level; affects all matching objects inside.
 2. **Latency:** Lifecycle rules are eventually consistent. It may take up to **24 hours** for a rule to actually delete or move a file after the condition is met.
-

3. Command Cheat Sheet (CLI)

Here are the commands you need for daily operations.

A. Managing Versioning

1. Enable Versioning

Bash

```
gcloud storage buckets update gs://my-bucket-name --versioning
```

2. List All Versions (See Old & New) This is how you see the history. Look for the **Generation** column or # numbers.

Bash

```
# This shows the live object AND the hidden noncurrent versions  
gcloud storage ls --all gs://my-bucket-name/deploy.yaml
```

Output Example:

```
gs://my-bucket/deploy.yaml#123456... (Old Version)  
gs://my-bucket/deploy.yaml#987654... (Live Version)
```

3. Read/View an Old Version To see the content of a deleted/overwritten file, use its Generation ID.

Bash

```
gcloud storage cat gs://my-bucket-name/deploy.yaml#123456789
```

4. Restore an Old Version To "undo" a mistake, you copy the old generation onto the live path.

Bash

```
gcloud storage cp gs://my-bucket-name/deploy.yaml#123456789  
gs://my-bucket-name/deploy.yaml
```

B. Managing Lifecycle

Lifecycle rules are usually defined in a **JSON file** rather than a long single command.

Step 1: Create the Policy File (`lifecycle.json`) Create a file named `lifecycle.json` with your rules:

JSON

```
{  
  "rule": [  
    {  
      "action": {"type": "SetStorageClass", "storageClass": "NEARLINE"},  
      "condition": {"age": 30}  
    },  
    {  
      "action": {"type": "Delete"},  
      "condition": {"age": 365}  
    }  
  ]  
}
```

```
    },
    {
      "action": {"type": "Delete"},
      "condition": {"numNewerVersions": 3}
    }
  ]
}
```

(The last rule is critical for versioning: "If there are 3 newer versions of this file, delete this old one" - this prevents storage costs from exploding).

Step 2: Apply the Policy

Bash

```
gcloud storage buckets update gs://my-bucket-name --lifecycle-file=lifecycle.json
```

Step 3: View Current Policy To check what rules are currently running on a bucket:

Bash

```
gcloud storage buckets describe gs://my-bucket-name --format="json(lifecycle)"
```

Cloud SQL

1. Cloud SQL Concepts (Rephrased)

Definition: Cloud SQL is a fully managed relational database service for MySQL, PostgreSQL, and SQL Server. It handles the "undifferentiated heavy lifting" (backups, patches, failover) so you can focus on the application code.

Key Configurations:

- **Editions:**
 - **Enterprise (Standard):** 99.95% availability. 7 days of backup retention. (Standard choice for 99% of workloads).
 - **Enterprise Plus:** 99.99% availability. 35 days backup retention. Includes **Data Cache** (Flash storage) for higher performance. Expensive.
- **Availability (HA):**
 - **Zonal:** Database lives in one zone. If the zone fails, the DB is down.
 - **High Availability (Regional):** Google maintains an Active node in Zone A and a Standby node in Zone B. Failover is automatic.
 - **Note:** Cloud SQL is **Regional**. You cannot change the Region later, but you can change the Zones.
- **Storage:**
 - **Auto-scaling:** Recommended for Prod. Checks storage every 30 seconds; if utilization > 90%, it adds space.
 - **Constraint:** Storage size can increase, but **never decrease**.
- **Security (SSL):**
 - **Allow only SSL:** Encrypts data in transit.
 - **Require Trusted Client Certificates:** Enforces Mutual TLS (mTLS). You must download the keys to the client (VM) for it to connect.

2. Architecture: Private IP & Private Services Access

The Challenge:

Cloud SQL instances do not live inside your VPC. They live in a Google-Managed VPC. To let your VM talk to the Database via a Private IP (without going over the internet), you must peer these two networks.

The Solution:

This peering mechanism is called Private Services Access (PSA).

Step-by-Step Guide to Enable Private IP:

1. **Allocate IP Range:**
 - Go to **VPC Network ==> VPC Networks ==>** Select your VPC.
 - Tab: **Private Service Connection ==> Allocated IP Ranges for Services.**
 - Action: Allocate a CIDR block (e.g., **10.50.0.0/24**) specifically for Google Services.
2. **Create Connection (Peering):**
 - Tab: **Private Connections to Services.**

- Action: Create connection ==> Select the range you just allocated ==> Connect.
- Result: Google creates a VPC Peering link between your VPC and the Cloud SQL VPC.

3. Configure DB:

- When creating the Cloud SQL instance, select **Private IP**.
- Select your VPC from the dropdown.

3. Cheat Sheet: Mandatory Parameters

When using CLI or Terraform, you must define these core attributes.

Feature	Terraform Argument	Gcloud Flag	Note
Identity	name	INSTANCE_NAME	Must be unique.
Version	database_version	--database-version	e.g., MYSQL_8_0 , POSTGRES_14 .
Hardware	tier	--tier	e.g., db-custom-2-3840 (2 vCPU, 3.8GB RAM).
Location	region	--region	e.g., us-central1 .
Storage	disk_size, disk_type	--storage-size, --storage-type	PD SSD or PD HDD .
Network	ip_configuration	--network	Must point to your VPC ID.
Security	require_ssl	--require-ssl	Enforces encryption.

4. Gcloud Commands

A. Prerequisites (Network Plumbing)

Before creating a Private IP instance, you must set up the network peering.

Bash

```
# 1. Create a Private IP Range for Google Services
gcloud compute addresses create google-managed-services-range \
    --global \
    --purpose=VPC_PEERING \
    --addresses=192.168.0.0 \
    --prefix-length=24 \
    --description="Peering range for Cloud SQL" \
    --network=my-custom-vpc

# 2. Create the VPC Peering (Private Services Access)
gcloud services vpc-peerings connect \
    --service=servicenetworking.googleapis.com \
    --ranges=google-managed-services-range \
    --network=my-custom-vpc \
    --project=$PROJECT_ID
```

B. Create the Cloud SQL Instance

Bash

```
gcloud sql instances create sample-my-sql-instance \
    --database-version=MYSQL_8_0 \
    --cpu=2 \
    --memory=4GB \
    --region=us-central1 \
    --root-password="Admin123#" \
    --storage-size=10GB \
    --storage-type=SSD \
    --network=projects/$PROJECT_ID/global/networks/my-custom-vpc \
    --no-assign-ip \
    --require-ssl
```

(Note: `--no-assign-ip` disables Public IP, forcing Private IP usage via the network flag).

Migration and Replicas

1. Migration Strategy: On-Prem to Cloud SQL

Scenario: You have a MySQL database (`employee-db`) running On-Premise and you want to move it to Google Cloud SQL.

Option A: Dump & Restore (Offline / Maintenance Window)

Best for non-critical applications where downtime is acceptable.

1. Preparation:

- Notify customers of maintenance/downtime (e.g., "System down from 8:00 AM to 1:00 PM").
- **Lock** the On-Prem database (Set to Read-Only) to ensure no new data is written during migration.

2. Export (On-Prem):

- Generate a dump file.
- Formats: `.sql` (Schema + Data) or `.csv` (Data only).

3. Transfer:

- Upload the dump file to a **Google Cloud Storage (GCS)** bucket.

4. Import (Cloud SQL):

- Go to your Cloud SQL instance (`dev-sql-instance`).
- Select **Import**==> Source: GCS Bucket==> File: `dump.sql`.

5. Cutover:

- Once the import finishes (e.g., at 1:00 PM), update your application configuration to point to the new Cloud SQL IP address.
- Resume operations.

Option B: Live Migration (Zero Downtime)

Best for critical applications where data changes every second (e.g., 100,000 records ==> 100,500 records).

- Uses **Database Migration Service (DMS)**.
- Sets up continuous replication so the Cloud DB stays in sync with On-Prem until you are ready to switch over instantly.

2. Operational Management: Import & Export

Exporting Data (Backup/Transfer)

- **Source:** You can export the entire instance or specific databases (e.g., just `employee-db` out of 3 apps).
- **Format:**
 - **SQL:** Contains commands to recreate tables and data.
 - **CSV:** Contains raw data (no schema).
- **Destination:** Data is sent directly to a **GCS Bucket**.
- **Security Note:** By default, the `root` user has access to all databases. Best practice is to create specific users and grant them permissions only for their specific database (e.g., `GRANT ALL ON db1.* TO 'user1'@'%'`).

Importing Data

- **Source:** Upload from your local computer (slow) or select a file residing in a **GCS Bucket** (fast/recommended).
 - **Configuration:**
 - **SQL File:** Just select the file; it runs the commands.
 - **CSV File:** You must specify exactly which **Database** and **Table** the data belongs to.
-

3. Read Replicas (Performance Scaling)

The Problem:

If you have one instance (test-primary-instance) handling both Writes (New data) and Reads (Reporting/Analytics), performance will drop due to high load.

The Solution:

Create Read Replicas.

- **Nature:** These are **Read-Only** copies of the primary instance.
 - **Sync:** Google automatically syncs new data from Primary ==> Replica.
 - **Traffic Splitting:** Configure your application to send:
 - **INSERT/UPDATE** queries ==> **Primary Endpoint**.
 - **SELECT** (Report) queries ==> **Read Replica Endpoint**.
-

4. Disaster Recovery (DR) & Promotion

Cross-Region Replication:

- **Primary:** `us-central1`.
- **Replica:** Create a Read Replica in `us-east1` or `asia-southeast1`.
- **Benefit:** If the entire `us-central1` region goes down, your data is safe in the other region.

Promoting a Replica (The DR Switch):

If the Primary fails, you Promote the Read Replica.

1. **Action:** Click "Promote Replica" in the console.
 2. **Result:** The Replica stops being a "follower" and becomes a standalone **Primary Instance** (Read & Write capable).
 3. **Broken Link:** The sync between the Old Primary and New Primary is broken permanently.
 4. **IP Address:** The promoted instance keeps its own IP address. You must update your application to point to this new IP.
-

5. CLI Cheat Sheet (Gcloud)

1. Create a Read Replica

Bash

```
gcloud sql instances create my-replica \
--master-instance-name=test-primary-instance \
--region=us-east1 \
--tier=db-custom-2-3840
```

2. Import SQL from GCS

Bash

```
gcloud sql import sql test-primary-instance gs://my-bucket/backup.sql \
--database=employee-db
```

3. Export SQL to GCS

Bash

```
gcloud sql export sql test-primary-instance gs://my-bucket/export-data.sql \
--database=employee-db
```

4. Promote Replica to Primary

Bash

```
gcloud sql instances promote-replica my-replica
```

HA and DR

1. High Availability (HA) - Zonal Protection

Concept: HA protects your database against a **Zone Failure** (e.g., `us-central1-a` goes down).

- **Architecture:**
 - **Primary Instance:** Located in the primary zone (e.g., `us-central1-a`).
 - **Standby Instance:** Located in a secondary zone (e.g., `us-central1-b`) within the **same region**.
 - **Data Sync:** Google handles synchronous replication. Every write to the Primary is written to the Standby before success is confirmed.
 - **Failover Behavior:**
 - If Zone A fails, Google automatically switches traffic to the Standby in Zone B.
 - **IP Address:** The Static IP address of the instance **remains the same**. Your application does *not* need to change its connection string.
 - **Billing:** You pay for **two instances** (Primary + Standby) + Storage.
 - **Use Case:** Mandatory for **Stage** and **Production**. (Dev instances usually run as "Single Zone" to save money).
 - **Testing:** You can manually click "Failover" in the console to simulate a zonal outage and verify your application recovers.
-

2. Disaster Recovery (DR) - Regional Protection (Standard Edition)

Concept: DR protects your database against a **Total Region Failure** (e.g., the entire `us-central1` region goes offline).

- **Architecture:**
 - **Primary:** `test-central` in `us-central1` (IP: 34.10.10.5).
 - **Read Replica:** `test-east` in `us-east4` (IP: 34.10.10.6).
 - **Failover Process (Manual Promotion):**
 - Region `us-central1` goes down.
 - You manually **Promote** the `test-east` Read Replica.
 - `test-east` becomes a standalone **Primary Instance** (Read/Write).
 - **Connection String:** The IP address is different (34.10.10.6). You must update your application to use the new endpoint.
 - **Failback (The Hard Part):**
 - Once `us-central1` comes back online, the old primary is outdated.
 - To go back, you must create a *new* replica in Central from the East instance, wait for sync, and promote it again.
-

3. Cloud SQL Enterprise Plus (Advanced DR)

Concept: Enterprise Plus is the premium tier offering higher performance (Data Cache) and advanced, reversible DR capabilities.

- **Machine Types:** Uses "Tiers" (Shared/Dedicated) with optimized hardware capabilities compared to Standard.
 - **Designated DR Replica:**
 - In Enterprise Plus, you can explicitly mark a Cross-Region Replica as a "**DR Replica**".
 - **Switchover (Planned Failover):**
 - Unlike the Standard "Promotion" (which breaks the link), Enterprise Plus performs a **Role Swap**.
 - **Action:** You trigger a Switchover.
 - **Result:** East becomes **Primary**. Central becomes the **DR Replica**. The replication link is maintained, just reversed.
 - **Benefit:** Zero data loss during planned switches, and much easier "Failback" to the original region later.
-

4. Comparison Summary

Feature	Standard HA	Standard DR (Cross-Region)	Enterprise Plus DR
Scope	Zonal (Same Region)	Regional (Different Region)	Regional (Different Region)
Failover Type	Automatic	Manual Promotion	Manual Switchover
IP Address	Same (Retained)	New (Replica's IP)	New (Replica's IP)*
Replication	Synchronous (Zero Data Loss)	Asynchronous (Potential Lag)	Asynchronous
Fallback	Automatic	Difficult (Manual Re-build)	Easy (Role Swap)

Export to Sheets

**Note: Even in Enterprise Plus switchover, the underlying instances typically retain their specific IPs. You usually need a DNS CNAME or internal proxy to handle the endpoint change seamlessly.*

5. CLI Commands

Create HA Instance (Standard):

Bash

```
gcloud sql instances create prod-db-instance \
--region=us-central1 \
--availability-type=REGIONAL \
--tier=db-custom-2-3840
```

(Note: *REGIONAL* flag enables HA/Failover replica).

Trigger Manual HA Failover (Test):

Bash

```
gcloud sql instances failover prod-db-instance
```

Create Cross-Region Replica (DR):

Bash

```
gcloud sql instances create test-east-replica \
--master-instance-name=prod-db-instance \
--region=us-east4
```

Promote Replica (Standard DR):

Bash

```
gcloud sql instances promote-replica test-east-replica
```

Enterprise Plus Switchover (Advanced DR): (*This performs the reversible role swap*)

Bash

```
gcloud sql instances switchover test-east-replica
```

Cloud Run

Initial Deployment

Here is the rephrased and organized study guide based on your notes, including the specific commands and code for the Python/FastAPI lab.

1. Cloud Run Overview

What is it?

Cloud Run is a fully managed serverless platform that allows you to run stateless containers without managing infrastructure (servers or clusters). It abstracts away the underlying hardware and OS.

Key Use Cases:

- **Services:** Frontend web apps, backend APIs, microservices.
- **Jobs:** Batch data processing, script execution.
- **LLMs:** Hosting lightweight AI/ML models.
- **Event Processing:** Asynchronous queue processing (e.g., from Pub/Sub).

Evolution of Compute:

1. **Bare Metal:** Applications deployed directly on physical servers (difficult to scale/manage).
2. **Virtualization:** Applications deployed on Virtual Machines (VMs) (better resource utilization).
3. **Containerization:** Applications packaged with dependencies (Docker), ensuring consistency across environments.
4. **Serverless (Cloud Run):** You provide the container; Google manages the runtime, scaling, and networking.

2. Artifact Management (GAR vs GCR)

What is an Artifact?

An artifact is the "build output" of your application code.

- **Java:** JAR or WAR files.
- **Node.js:** Distributable folders/packages.
- **Docker:** Container Images.

Google Container Registry (GCR):

- **Status: Deprecated/Obsolete.** This was Google's original image storage service.
- **Behavior:** It stored images in Cloud Storage buckets but lacked advanced features.

Google Artifact Registry (GAR):

- **Status: Current Standard.** This is the evolution of GCR.
- **Function:** It works like Docker Hub, JFrog, or Nexus but is integrated into GCP.
- **Supported Formats:** Docker images, Maven (Java), npm (Node), Python (PyPI), and more.

- **Features:** Vulnerability scanning, granular IAM access control, and region-specific storage.
-

3. Deploying to Cloud Run (The Workflow)

The Workflow:

Source Code ==> Dockerfile ==> Build Image ==> Push to Artifact Registry ==> Deploy to Cloud Run

Cloud Run Service Features:

- **Endpoint:** Automatically assigns a stable HTTPS URL (e.g., <https://service-name-xyz.run.app>).
- **Autoscaling:** Scales automatically based on traffic (can scale down to zero to save costs).
- **Load Balancing:** Built-in global load balancing.
- **Authentication:**
 - **Allow Unauthenticated (Public):** The service is accessible to anyone on the internet (e.g., a public website).
 - **Require Authentication (IAM):** Only specific users or other Google services can access it (e.g., an internal API).

Comparison: Cloud Run vs. GKE

- **Cloud Run:** Best for stateless apps, "scale-to-zero," and simplicity. No cluster management.
- **GKE (Google Kubernetes Engine):**
 - **Standard:** You manage the worker nodes (VMs). Best for complex architectures needing full control.
 - **Autopilot:** Google manages the nodes/infrastructure. You just deploy pods. A middle ground between GKE Standard and Cloud Run.

4. Lab: Python FastAPI Application

Here is the step-by-step implementation of the lab you requested involving FastAPI, OpenTelemetry, and Redis.

Step 1: Create the Source Code

Create a file named `main.py`.

Python

```
from fastapi import FastAPI
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import ConsoleSpanExporter, SimpleSpanProcessor
import redis
import os
```

```

# 1. Setup OpenTelemetry (Simple Console Exporter for demo)
trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer(__name__)
trace.get_tracer_provider().add_span_processor(
    SimpleSpanProcessor(ConsoleSpanExporter()))
)

app = FastAPI()

# 2. Setup Redis Connection (Uses Environment Variables)
# In Cloud Run, we inject these env vars
redis_host = os.getenv("REDIS_HOST", "localhost")
redis_port = os.getenv("REDIS_PORT", "6379")
r = redis.Redis(host=redis_host, port=redis_port, decode_responses=True)

# Endpoint 1: Root
@app.get("/")
def read_root():
    with tracer.start_as_current_span("root_endpoint"):
        return {"message": "Hello from Cloud Run!"}

# Endpoint 2: Write to Redis
@app.get("/set/{key}/{value}")
def set_redis(key: str, value: str):
    with tracer.start_as_current_span("redis_set"):
        r.set(key, value)
        return {"status": "set", "key": key, "value": value}

# Endpoint 3: Read from Redis
@app.get("/get/{key}")
def get_redis(key: str):
    with tracer.start_as_current_span("redis_get"):
        val = r.get(key)
        return {"key": key, "value": val}

```

Step 2: Create the requirements.txt

```

Plaintext
fastapi
unicorn
redis
opentelemetry-api
opentelemetry-sdk

```

Step 3: Create the Dockerfile

```

Dockerfile
# Use official lightweight Python image

```

```
FROM python:3.9-slim

# Set working directory
WORKDIR /app

# Copy requirements and install
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY main.py .

# Command to run the app using Uvicorn
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Step 4: Create Repository in Artifact Registry

You must create the "folder" (repository) before you can push images to it.

Bash

```
# Create a Docker repository in us-central1
gcloud artifacts repositories create my-repo \
--repository-format=docker \
--location=us-central1 \
--description="My FastAPI Docker Repo"
```

Step 5: Build and Tag the Image

Docker images for GAR must follow this specific naming format:

[LOCATION]-docker.pkg.dev/[PROJECT_ID]/[REPOSITORY]/[IMAGE]:[TAG]

Bash

```
# 1. Build the image locally (generic name)
docker build -t my-fastapi-app .
```

```
# 2. Tag the image for Google Artifact Registry
# Replace 'my-project-id' with your actual project ID
docker tag my-fastapi-app
us-central1-docker.pkg.dev/my-project-id/my-repo/my-fastapi-app:v1
```

Step 6: Push to Artifact Registry

Bash

```
# Authentication (run once)
gcloud auth configure-docker us-central1-docker.pkg.dev
```

```
# Push the tagged image
```

```
docker push us-central1-docker.pkg.dev/my-project-id/my-repo/my-fastapi-app:v1
```

Step 7: Deploy to Cloud Run

You can do this via the GUI or CLI.

CLI Command:

Bash

```
gcloud run deploy fastapi-service \
--image=us-central1-docker.pkg.dev/my-project-id/my-repo/my-fastapi-app:v1 \
--region=us-central1 \
--allow-unauthenticated \
--port=8080 \
--set-env-vars=REDIS_HOST=10.0.0.5,REDIS_PORT=6379
```

(Note: Replace **10.0.0.5** with your actual Redis/Memorystore IP address).

Blue-Green Deployment

1. Core Concepts & "Gotchas"

- **Cloud Shell is Ephemeral:**
 - The Docker images you build in Cloud Shell are **not persistent**. If you restart the shell, they disappear.
 - Only data in your `$HOME` directory is persistent.
 - *Best Practice:* Always push your images to **Artifact Registry** immediately after building.
 - **Port Requirement:**
 - By default, Cloud Run containers must listen on **Port 8080**.
 - Google sends traffic to 8080. If your app listens on port 3000, the health check will fail, and the deployment will crash.
 - **Pricing Models:**
 - **Request-based:** You pay only while a request is being processed. If no one visits, you pay \$0.
 - **Instance-based:** You keep a minimum number of instances running (e.g., `min-instances=1`) to avoid "Cold Starts." You pay for the CPU/Memory allocation even if idle.
 - **Scaling:**
 - **Scale to Zero:** Unique feature of Cloud Run. If there is no traffic, Google turns off all containers.
 - **Execution Environments:**
 - **First Generation:** Standard. Faster cold starts for small apps.
 - **Second Generation:** Full Linux compatibility. Better performance for heavy CPU tasks.
-

2. Lab Scenario: Blue/Green/Purple Deployments

This lab demonstrates how to safely release new versions without breaking the live site.

Phase 1: The Blue Deployment (Version 1)

1. **Action:** Deploy the "Blue" image.
2. **Settings:** Ensure "Serve this revision immediately" is **Checked**.
3. **Result:** 100% of traffic goes to Blue.

Phase 2: The Green Deployment (Version 2)

1. **Action:** Update code, rebuild image with `green` tag, push to GAR.
2. **Deploy:** Click "Edit & Deploy New Revision." Select Green image.
3. **Traffic:** Leave "Serve immediately" **Checked**.
4. **Result:** Cloud Run automatically shifts 100% of traffic from Blue to Green.

Phase 3: The Purple Deployment (Canary Test / Hotfix)

Goal: Deploy the Purple version, but DO NOT let the public see it yet. Only we want to test it.

1. **Action:** Build/Push Purple image.
 2. **Deploy:** Click "Edit & Deploy New Revision."
 3. **Crucial Step: UNCHECK** the box "Serve this revision immediately."
 4. **Traffic:** Public traffic stays 100% on Green. Purple gets 0% traffic.
 5. **Tagging:** In the deployment settings, give this revision a specific tag name, e.g., `prod-test`.
 6. **Result:**
 - o **Main URL:** Shows Green (Public).
 - o **Tagged URL:** Google generates a specific *hidden* URL (e.g., <https://prod-test--service-xyz.run.app>) that loads the Purple version.
 7. **Testing:** You verify the Purple version using the hidden URL.
 8. **Rollout:** Once tested, go to "Manage Traffic" and gradually move traffic: 0% → 10% → 50% → 100% Purple.
-

3. Task 1: VM to Artifact Registry Workflow (CLI)

Goal: Create a VM, install Docker, build an image, and push it to Google Artifact Registry (GAR).

Bash

```
# 1. Create a VM (Standard Debian)
gcloud compute instances create builder-vm \
--zone=us-central1-a \
--machine-type=e2-medium \
--image-family=debian-11 \
--image-project=debian-cloud \
--scopes=https://www.googleapis.com/auth/cloud-platform
```

```
# 2. SSH into the VM
gcloud compute ssh builder-vm --zone=us-central1-a
```

```
# --- INSIDE THE VM ---
```

```
# 3. Install Docker
sudo apt-get update
sudo apt-get install -y docker.io
```

```
# 4. Authenticate Docker with Google Cloud
gcloud auth configure-docker us-central1-docker.pkg.dev
```

```
# 5. Create a dummy Dockerfile
echo "FROM nginx" > Dockerfile
echo "RUN echo 'Hello from Task 1' > /usr/share/nginx/html/index.html" >> Dockerfile
```

```
# 6. Build the image
# Format: LOCATION-docker.pkg.dev/PROJECT-ID/REPO-NAME/IMAGE:TAG
docker build -t
us-central1-docker.pkg.dev/$DEVSHELL_PROJECT_ID/my-repo/task-image:v1 .

# 7. Push to Registry
docker push us-central1-docker.pkg.dev/$DEVSHELL_PROJECT_ID/my-repo/task-image:v1
```

4. Task 2: Cloud Run Management (CLI)

Goal: Deploy revisions, manage 0% traffic deployments, and traffic splitting via CLI.

Step 1: Create Service (Initial Deployment)

Bash

```
gcloud run deploy my-web-app \
--image=us-central1-docker.pkg.dev/$DEVSHELL_PROJECT_ID/my-repo/task-image:v1 \
--region=us-central1 \
--allow-unauthenticated
```

Step 2: Deploy New Revision with 0% Traffic (The "Purple" Scenario) We use `--no-traffic` to ensure it doesn't go live immediately. We use `--tag` to create the hidden test URL.

Bash

```
gcloud run deploy my-web-app \
--image=us-central1-docker.pkg.dev/$DEVSHELL_PROJECT_ID/my-repo/task-image:v2 \
--region=us-central1 \
--no-traffic \
--tag=prod-test
```

Step 3: Check the Status

Bash

```
gcloud run services describe my-web-app --region=us-central1
```

You will see the traffic split is still 100% to the old revision, but the new revision exists with a specific URL.

Step 4: Manage Traffic (Split 50/50) You can refer to revisions by name (e.g., `my-web-app-00002-xqz`).

Bash

```
gcloud run services update-traffic my-web-app \  
--region=us-central1 \  
--to-revisions=my-web-app-00001-xyz=50,my-web-app-00002-xqz=50
```

Step 5: Promote New Version to 100% (Go Live) Alternatively, you can use `--to-latest` to just move everything to the newest built revision.

Bash

```
gcloud run services update-traffic my-web-app \  
--region=us-central1 \  
--to-latest
```

Private Database Access

1. Serverless VPC Access (The Bridge)

Concept: Serverless services (Cloud Run, Cloud Functions, App Engine Standard) run *outside* your VPC. By default, they cannot reach your internal resources (like a Private SQL IP or a VM on [10.0.0.5](#)). To fix this, we create a **VPC Access Connector**. This acts as a bridge (or jump server pipeline) that tunnels traffic from the serverless environment into your VPC.

Setup Steps:

1. **Network Prerequisite:** You must have a **Custom Subnet** (usually a [/28](#) CIDR range) dedicated *only* to this connector. The connector creates managed connector instances (VMs) in this subnet to forward traffic.
 2. **Create Connector:**
 - Go to **VPC Network** → **Serverless VPC Access**.
 - Select the **Region** and **Network** (must match your Cloud Run region).
 - **Scaling:** Set Min/Max instances (e.g., 2 to 10). *Warning: This service is expensive because these connector instances run 24/7.*
 3. **Configure Service (Cloud Run):**
 - Go to your Cloud Run service → **Networking** tab.
 - Under **VPC Access**, select "Connect to a VPC".
 - Choose your connector.
 - **Traffic Routing:** Select "Route only requests to private IPs" (Split Tunneling) or "Route all traffic" (Force Tunneling).
-

2. Identity-Aware Proxy (IAP) for SSH

Concept: IAP allows you to SSH into **Private VMs** (VMs with no Public IP) securely over the internet without using a VPN or Bastion Host. It wraps the SSH connection in an HTTPS tunnel authenticated by your Google Identity.

The "Red to Green" Setup:

1. **Enable API:** Go to **Security** → **Identity-Aware Proxy** and enable the API.
2. **Check Status:** Click on the **SSH and TCP Resources** tab. You will likely see a "Warning/Red" status.
3. **The Firewall Rule (Critical Fix):**
 - The "Red" status means the firewall is blocking Google's IAP traffic.
 - You must create an Ingress Firewall Rule.
 - **Source IP Range:** [35.235.240.0/20](#) (This is the specific range Google uses for IAP tunneling. *Note: 130.211... is for Load Balancers, not IAP*).
 - **Ports:** [22](#) (SSH) or [3389](#) (RDP).
4. **Result:** The status turns **Green**. You can now click "SSH" in the console, and Google creates a tunnel to your private instance.

3. Cloud NAT (Outbound Internet for Private VMs)

Problem: A Private VM (no Public IP) is secure, but it cannot download updates (e.g., `apt-get update`, `pip install`) or call external APIs (e.g., OpenAI).

Solution: **Cloud NAT (Network Address Translation)** allows private VMs to initiate connections **out** to the internet but prevents the internet from initiating connections **in**.

Setup Steps:

1. **Create Cloud Router:**
 - Cloud NAT requires a **Cloud Router** to handle the control plane (routing packets).
 - Create one in the same region as your VMs.
 2. **Create Cloud NAT Gateway:**
 - Go to **Network Services → Cloud NAT**.
 - Link it to your Cloud Router.
 3. **IP Configuration:**
 - **Public (Manual/Auto):** 99% of the time, you use **Public NAT**.
 - **Static IP (Recommended for Enterprise):** Instead of "Auto-allocating" IPs, reserve a **Static External IP** for the NAT Gateway.
 4. **Mapping:**
 - Select which subnets can use this NAT (usually "All subnets" or specific "Private subnets").
-

4. Real-World Scenario: Multi-Cloud Whitelisting

Scenario:

- **GCP:** Hosting a backend application on Private VMs.
- **Azure:** Hosting an OpenAI/ML Service.
- **Security:** The Azure firewall blocks all traffic unless it comes from a specific "Allowed IP".

How Cloud NAT solves this:

1. Since your GCP VMs are private, they don't have their own public IPs to whitelist.
 2. You configure **Cloud NAT** with a **Manual Static IP** (e.g., `34.55.x.x`).
 3. All traffic leaving your GCP VMs is masked behind this single NAT IP.
 4. You tell the Azure team: "*Please whitelist `34.55.x.x`.*"
 5. Now, your private GCP servers can securely talk to the Azure service.
-

Summary Table

Feature	Serverless VPC Access	IAP (Identity-Aware Proxy)	Cloud NAT
Goal	Allow Cloud Run/Functions to reach Private Internal IPs.	Allow Admins to SSH into Private VMs.	Allow Private VMs to access the Internet (Outbound only).
Direction	Inbound to VPC.	Inbound to VM (via Tunnel).	Outbound to the Internet.
Key Config	Dedicated /28 Subnet.	Firewall Allow 35.235.240.0/20 .	Cloud Router + Static IP.
Cost	High (Instance-based billing).	Free (Ingress traffic).	Low (Data processing fees).

Cloud Build

1. What is Cloud Build?

Cloud Build is a **serverless, fully managed CI/CD platform**. It allows you to build, test, and deploy code without provisioning or managing any servers.

- **The Problem (Jenkins Scenario):**
 - To build with Jenkins, you typically need to manage infrastructure: 1 Master VM and multiple Slave VMs.
 - You must manually install software (Java, Maven, Docker) on every machine.
 - You pay for these VMs 24/7 even if no builds are running (unless you configure complex dynamic scaling).
 - Maintenance (OS patching, plugin updates) is your responsibility.
- **The Solution (Cloud Build):**
 - **Serverless:** No VMs to manage. Google spins up a temporary container, runs your build, and destroys it immediately.
 - **Scalable:** If 10 developers submit builds at once, Cloud Build spins up 10 parallel workers instantly.
 - **Cost-Effective:** You pay only for the minutes the build actually runs.

2. How Cloud Build Works: "Cloud Builders"

This is the most important concept to understand.

- **Concept:** Cloud Build executes your pipeline steps inside **temporary containers**.
- **The Tools:** If your build needs **maven**, Cloud Build spins up a container that already has Maven installed. If you need **docker** commands, it uses a container with Docker installed.
- **Cloud Builders:** These standard containers (e.g.,
`gcr.io/cloud-builders/docker`, `gcr.io/cloud-builders/git`,
`gcr.io/cloud-builders/gcloud`) are maintained by Google and available for you to use instantly.

3. Build Methods

There are two primary ways to trigger a build:

Method A: Tag-Based (Simple)

This is for simple "Build & Push" scenarios where you just have a **Dockerfile**.

- **Action:** It reads your **Dockerfile**, builds the image, tags it, and pushes it to Artifact Registry (GAR).

CLI Command:

Bash

```
# Automatically builds Dockerfile in current dir and pushes to registry
gcloud builds submit --tag us-central1-docker.pkg.dev/project-id/repo/my-image:v1
```

Method B: Config-Based (Advanced)

This uses a configuration file (usually `cloudbuild.yaml`) to define a multi-step pipeline. Use this when you need to do more than just build a container (e.g., run unit tests, copy files to GCS, deploy to Cloud Run).

4. The `cloudbuild.yaml` Configuration

This file tells Cloud Build exactly what steps to take. Each "step" runs in a specific container (Cloud Builder).

Scenario:

1. **Build** a Docker image from Python code.
2. **Push** the image to the Artifact Registry.
3. **Deploy** the image to Cloud Run.

The `cloudbuild.yaml` File:

steps:

```
# STEP 1: Build the container image
# Uses the 'docker' cloud builder
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t',
    'us-central1-docker.pkg.dev/sampleprojecttesting-478502/python-app-fastapi-docker:v1', '.']

# STEP 2: Push the image to Artifact Registry
# Uses the 'docker' cloud builder again
- name: 'gcr.io/cloud-builders/docker'
  args: ['push',
    'us-central1-docker.pkg.dev/sampleprojecttesting-478502/python-app-fastapi-docker:v1']

# STEP 3: Deploy to Cloud Run
# Uses the 'gcloud' cloud builder
- name: 'gcr.io/cloud-builders/gcloud'
  args:
    - 'run'
    - 'deploy'
    - 'my-python-service'
    - '--image'
    - 'us-central1-docker.pkg.dev/sampleprojecttesting-478502/python-app-fastapi-docker:v1'
    - '--region'
    - 'us-central1'
```

```
- '--allow-unauthenticated'

# Optional: Define timeout or substitutions
timeout: '1200s'
```

5. Execution & Permissions

1. **Submit the Build** Once you have created the `cloudbuild.yaml` file, run this command in your terminal:

Bash
gcloud builds submit --config cloudbuild.yaml

2. **Critical: IAM Permissions** By default, the **Cloud Build Service Account** (which looks like `[project-number]@cloudbuild.gserviceaccount.com`) has permissions to build and push images, but it **does not have permission to deploy to Cloud Run**.

To make the pipeline work, you must go to IAM and grant the Cloud Build Service Account these roles:

- **Cloud Run Admin** (To deploy the service)
- **Service Account User** (To act as the runtime identity for the Cloud Run service)

Summary Comparison

Feature	Jenkins (Static Slave)	Cloud Build
Infrastructure	You manage Master + Slave VMs.	Fully Managed (Serverless).
Scaling	Limited by the number of slave VMs.	Auto-scales to meet demand instantly.
Maintenance	You must update Java/Docker manually.	Google maintains the "Cloud Builder" images.
Cost	24/7 VM cost (even if idle).	Pay-per-minute of build time only.

External Build

Cloud Build Triggers: Automating CI/CD

Concept:

Instead of manually running commands (`gcloud builds submit`) every time you change your code, Cloud Build Triggers automate the process. They "listen" for changes in your source code repository and automatically kick off a build pipeline when specific events occur.

The Workflow:

1. **Developer Action:** A developer commits and pushes code to a Git repository (e.g., GitHub, Bitbucket, GitLab).
2. **Trigger Detection:** Cloud Build detects the change (e.g., a push to the `main` branch).
3. **Execution:** Cloud Build automatically executes the instructions in your `cloudbuild.yaml` file.
4. **Result:** The build runs, images are pushed to Artifact Registry, and the application is optionally deployed to Cloud Run/GKE.

1. Setting Up the Connection (One-Time Setup)

Before you can create a trigger, you must connect your external repository to Google Cloud.

- **Action:** Go to **Cloud Build ==> Repositories ==> Connect Repository**.
- **Supported Providers:** GitHub, GitHub Enterprise, Bitbucket, GitLab.
- **Process:** You will authenticate with your provider (e.g., log in to GitHub) and select the specific repository you want to link.

2. Creating a Trigger (Step-by-Step)

Once connected, you define *when* the build should happen.

1. **Name & Region:** Give your trigger a clear name (e.g., `prod-deploy-trigger`) and select the region.
2. **Event (The "When"):** Choose what action starts the build.
 - **Push to a branch:** (Most common) Runs every time code is pushed to a specific branch (e.g., `main` or `develop`).
 - **Push new tag:** Runs when a Git tag is created (often used for releases like `v1.0`).
 - **Pull Request:** Runs when a PR is opened (useful for running tests *before* merging code).
3. **Source:** Select the Repository and the specific Branch (Regex is supported, e.g., `feature-.*`).
4. **Configuration (The "What"):**
 - **Type:** Select **Cloud Build configuration file (yaml or json)**.
 - **Location:** Specify the path, usually `cloudbuild.yaml` (root directory).
5. **Substitutions (Dynamic Variables):**
 - These allow you to pass variables into your `cloudbuild.yaml` at runtime.

- **Format:** Variables in the trigger setup must start with an underscore (e.g., `_REGION`, `_IMAGE_NAME`).
- **Usage:** In your YAML file, you refer to them as `$_['REGION']`.

6. Service Account:

- Select the custom Service Account that has the necessary permissions (Cloud Run Admin, etc.). Do not use the default one for production pipelines.

3. Gated Deployments (Approvals)

You might not want every single commit to go straight to production. You can add a manual "Gate."

- **Require Approval:** In the Trigger settings, check the box "**Require approval before build executes**".
- **The Flow:**
 1. Developer pushes code.
 2. The Trigger starts but enters a "**Pending**" state (Orange icon in the console).
 3. A user with the **Cloud Build Approver** (or Editor/Owner) role must log in to the GCP Console and click "**Approve**".
 4. Only then does the build/deploy step run.

4. Verification

- **Commit ID:** Every build is linked to the specific Git Commit SHA that triggered it. You can click the Commit ID in Cloud Build to see the exact code change in GitHub.
- **Logs:** You can watch the build logs in real-time. If it fails, the logs will show exactly which step (Build, Push, or Deploy) caused the error.

5. CLI Command to Create a Trigger

While the Console is easier for the initial OAuth connection to GitHub, you can create the trigger logic via CLI if the repo is already connected.

Bash

```
gcloud builds triggers create github \
--name="my-prod-trigger" \
--repo-name="my-repo" \
--repo-owner="my-github-username" \
--branch-pattern="^main$" \
--build-config="cloudbuild.yaml" \
--region="us-central1" \
--substitutions="_REGION=us-central1,_DEPLOY_ENV=prod" \
--require-approval
```

Summary of Key Features

Feature	Description
Automation	Removes the need for manual <code>gcloud builds submit</code> .
Substitutions	Inject variables (Region, Env) without changing the code.
Gating	"Require Approval" stops accidental production deployments.
Traceability	Every build links back to the exact code commit.

KMS

1. Encryption Basics

Encryption converts **Plain Text** into **Cipher Text** to protect data. In GCP, protection happens in two states:

- **Encryption in Transit:** Protects data while it is moving (e.g., transferring a file from your laptop to Cloud Storage).
 - **Encryption at Rest:** Protects data while it is stored on the disk (e.g., sitting in a Storage Bucket or Persistent Disk). By default, **all** data in GCP is encrypted at rest.
-

2. The Three Levels of Encryption

Google offers three tiers of control over the encryption keys:

A. Google Managed Encryption Keys (GMEK)

- **Default:** This is the standard setting for all GCP services.
- **Management:** Google handles everything—key creation, storage, rotation, and lifecycle.
- **User Action:** None. It just works.

B. Customer Managed Encryption Keys (CMEK)

- **Scenario:** You want more control (compliance/auditing) but don't want to build your own physical hardware.
- **Mechanism:** Google creates the keys, but they live in **your** Cloud KMS project.
- **User Control:** You control the **Key Rotation** schedule, permissions (who can use the key), and you can revoke the key to lock data instantly.
- **Setup:** Configured via Cloud KMS GUI or CLI.

C. Customer Supplied Encryption Keys (CSEK)

- **Scenario:** Your company requires that Google **never** sees the key material.
 - **Mechanism:** You generate the key on your own premise (laptop/server). You supply the key purely for the duration of the upload/download operation. Google uses it in memory to encrypt/decrypt and then discards it.
 - **Constraint:** Google cannot store or manage this key. If you lose this key, **your data is lost forever**.
 - **Usage:** Only supported via CLI (`gcloud/gsutil`) or API, not the GUI.
-

3. Setting up Cloud KMS (For CMEK)

If you choose **CMEK**, you must set up the Key Ring and Key.

1. **Create Key Ring:**
 - A logical grouping of keys.
 - **Location:** Must match the resource location (e.g., if your Bucket is in `us-central1`, the Key Ring must be in `us-central1` or `global`).
 2. **Create Key:**
 - **Protection Level:**
 - **Software:** Standard software-backed keys.
 - **HSM (Hardware Security Module):** Keys stored in FIPS 140-2 Level 3 validated hardware.
 - **External (EKM):** Keys stored outside Google (e.g., on-premise) but accessed by Google.
 - **Purpose:** "Symmetric Encrypt/Decrypt".
 - **Rotation:** You set the schedule (e.g., automatically generate a new version every 90 days).
 3. **Integration:**
 - When creating a GCS Bucket, you select "Customer Managed Key" and point to this Key Resource ID.
 - **Note:** A bucket has a default encryption setting (GMEK or CMEK). All new objects inherit this, but you can override it for specific individual files.
-

4. Customer Supplied Encryption Keys (CSEK) Implementation

Since CSEK requires you to provide the raw key material, here is the workflow using Python and `gsutil`.

Step A: Generate the Key (Python)

CSEK requires a **256-bit AES key** encoded in Base64.

```
Python
import base64
import os

# 1. Generate 32 bytes (256 bits) of random data
key = os.urandom(32)

# 2. Encode it to Base64 (Standard for GCP headers)
encoded_key = base64.b64encode(key).decode("utf-8")

print("Your CSEK Key:")
print(encoded_key)
```

Copy the output string. This is your key.

Step B: Uploading a File with CSEK

You must pass the key in the `.boto` configuration headers using the `-o` flag.

Bash

```
# Syntax: gsutil -o "GSUtil:encryption_key=[YOUR_BASE64_KEY]" cp [FILE] [DESTINATION]
gsutil -o "GSUtil:encryption_key=k73...=" cp confidential.txt gs://my-secure-bucket/
```

- **Result:** The file is stored. Even if you open the GCP Console, you cannot view/download this file because Google doesn't have the key to decrypt it for the UI.

Step C: Reading/Decrypting the File

To read the file back, you must provide the `decryption_key` flag (which matches the key you used to encrypt it).

Bash

```
# Syntax: gsutil -o "GSUtil:decryption_key1=[YOUR_BASE64_KEY]" cat [FILE]
gsutil -o "GSUtil:decryption_key1=k73...=" cat gs://my-secure-bucket/confidential.txt
```

5. Mixing Encryption Types in a Bucket

- **Bucket Default:** You set a default at the bucket level (e.g., GMEK or CMEK).
- **Object Override:** You can mix objects in the same bucket.
 - *Example:* A bucket can have Default GMEK.
 - File A (Uploaded normally): Uses GMEK.
 - File B (Uploaded with KMS flag): Uses CMEK.
 - File C (Uploaded with CSEK flag): Uses CSEK.
- **Limitation:** You cannot enforce "GMEK and CMEK" simultaneously on the *same object*. It is one method per object.

Database Migration System

1. Concept: Database Live Migration

- **Goal:** Migrate an application's database from On-Premises (or IaaS) to a managed Cloud SQL instance with **Minimal to Zero Downtime**.
- **Tool:** **Database Migration Service (DMS)**.
- **Mechanism:**
 1. **Full Dump:** It takes an initial snapshot of the source.
 2. **CDC (Change Data Capture):** It continuously syncs new changes (binlogs) from Source => Destination.
- **Versioning Rule:** The Source MySQL version must be **less than or equal to** the Destination version (e.g., Source 5.7 => Destination 8.0 is OK; Source 8.0 => Destination 5.7 is NOT).

2. Step-by-Step Implementation

Phase 1: Source Database Setup ("On-Prem" Simulation)

We will create a Debian VM and install MySQL to act as the "On-Prem" source.

1. Create the VM (Gcloud)

Bash

```
gcloud compute instances create on-prem-db \
--image-family=debian-11 \
--image-project=debian-cloud \
--zone=us-central1-a \
--subnet=subnet-1 \
--machine-type=e2-medium \
--tags=db-server
```

Phase 1: Configure the "On-Prem" Database (VM)

This section sets up the database VM with the correct network settings and database permissions immediately, preventing the connection and "Definer" errors.

1. VM Network & Firewall Setup

Before touching the database, ensure the VM itself allows traffic.

Bash

```
# 1. Open the Firewall (Allow traffic on port 3306)
```

```
gcloud compute firewall-rules create allow-mysql-3306 \
--network=vpc-dms \
--allow=tcp:3306 \
--source-ranges=0.0.0.0/0 \
```

```
--description="Allow DMS to connect to MySQL"
```

```
# 2. Check internal OS firewall (UFW) just in case
```

```
sudo ufw allow 3306/tcp
```

2. Configure MariaDB Settings

This script configures the bind address (to listen externally) and the binary logging format (required for migration).

Bash

```
# Backup the config file first
```

```
sudo cp /etc/mysql/mariadb.conf.d/50-server.cnf /etc/mysql/mariadb.conf.d/50-server.cnf.bak
```

```
# 1. Set Bind Address to 0.0.0.0 (Listen on all IPs)
```

```
sudo sed -i 's/^bind-address.*$/bind-address = 0.0.0.0/'  
/etc/mysql/mariadb.conf.d/50-server.cnf
```

```
# 2. Enable Binary Logging with ROW format (CRITICAL for DMS)
```

```
# We use 'tee -a' to append these lines safely
```

```
echo "[mysqld]" | sudo tee -a /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
echo "server-id = 1" | sudo tee -a /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
echo "log_bin = /var/log/mysql/mysql-bin.log" | sudo tee -a  
/etc/mysql/mariadb.conf.d/50-server.cnf
```

```
echo "binlog_format = ROW" | sudo tee -a /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
echo "expire_logs_days = 7" | sudo tee -a /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
# 3. Restart MariaDB to apply changes
```

```
sudo systemctl restart mariadb
```

```
# 4. Verify the settings
```

```
sudo netstat -tulpn | grep 3306 # Should show 0.0.0.0:3306
```

3. Create Users and Data (The "Golden Script")

This SQL script creates the necessary users (`admin` and `root` with `%` host) and ensures the stored procedure has the correct "Definer" to avoid migration errors.

Log in to MySQL:

Bash

```
sudo mysql -u root
```

Run this SQL block:

SQL

```
-- 1. Create the Admin User for Migration
```

```
CREATE USER 'admin'@'%' IDENTIFIED BY 'Admin123#';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%' WITH GRANT OPTION;
```

```
-- 2. Create a Root User accessible from Google Cloud (Fixes "Definer" errors)
```

```
CREATE USER 'root'@'%' IDENTIFIED BY 'your_root_password_here';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION;
```

```
FLUSH PRIVILEGES;
```

```
-- 3. Create Database
```

```
CREATE DATABASE employee_db;
```

```
USE employee_db;
```

```
-- 4. Create Tables & Dummy Data
```

```
CREATE TABLE departments (id INT AUTO_INCREMENT PRIMARY KEY, name  
VARCHAR(50));
```

```
INSERT INTO departments (name) VALUES  
('HR'), ('IT'), ('Sales'), ('Marketing'), ('Finance'),  
('Legal'), ('Operations'), ('Support'), ('Engineering'), ('R&D');
```

```
CREATE TABLE employees (id INT AUTO_INCREMENT PRIMARY KEY, name  
VARCHAR(50), dept_id INT);
```

```
-- 5. Create Procedure with the CORRECT DEFINER (Fixes "Definer root@localhost" error)
```

```
DELIMITER $$
```

```
CREATE DEFINER='root'@'%' PROCEDURE generate_data()
```

```
BEGIN
```

```
DECLARE i INT DEFAULT 1;
```

```
WHILE i <= 50 DO
```

```
    INSERT INTO employees (name, dept_id) VALUES (CONCAT('Employee_', i), FLOOR(1  
+ (RAND() * 10)));
```

```
    SET i = i + 1;
```

```
END WHILE;
```

```
END$$
```

```
DELIMITER ;
```

```
-- Generate the data
```

```
CALL generate_data();
```

Phase 2: Network Peering (The Bridge)

This connects your VPC to Google's internal network so Cloud SQL can talk to your VM via Private IP.

Run these commands in Cloud Shell:

Bash

```
# 1. Reserve an IP range for Google Services
```

```
gcloud compute addresses create google-managed-services-vpc-dms \
```

```
--global \
```

```
--purpose=VPC_PEERING \
```

```
--prefix-length=16 \
```

```
--description="Peering for Google Services" \
```

```
--network=vpc-dms
```

```
# 2. Create the Private Connection
```

```
gcloud services vpc-peerings connect \
```

```
--service=servicenetworking.googleapis.com \
```

```
--ranges=google-managed-services-vpc-dms \
```

```
--network=vpc-dms
```

Phase 3: DMS Configuration (Google Cloud Console)

Now you are ready to configure the migration job in the UI.

1. Create Connection Profile (The Source)

- **Go to:** Database Migration Service > Connection Profiles.
- **Create Profile:** MySQL.
- **Hostname:** [Public IP or Internal IP of your "On-Prem" VM].
 - *Note: Since we set up peering, you can technically use the Internal IP if running from another VM in the same VPC, but for this lab, Public IP is easiest unless you have VPN set up. If using Public IP, ensure the firewall allows it.*
- **Port:** 3306.

- **Username:** admin.
- **Password:** Admin123#.
- **Encryption:** None.
- **Region:** us-central1.

2. Create Migration Job

- **Go to:** Migration Jobs > Create Migration Job.
- **Job Name:** migrate-employee-db.
- **Source Engine:** MySQL.
- **Migration Type:** Continuous (for CDC/Real-time sync).
- **Destination Region:** us-central1.
- **Destination Version:** Cloud SQL for MySQL 8.0 (or matching version).
- **Networking:** Private IP (Select your vpc-dms network).

2.a. Define Source & Test

- Select the Connection Profile you created.
- Click **Run Test**.
 - If you followed Phase 1 correctly, this will pass immediately (Green Check).
- If it passes, click **Create & Start**.

3. Define Destination

- Select "Create a new instance".
- **Destination ID:** cloud-sql-dest.
- **Password:** Set a root password.
- **Region:** Same as source or different (your choice).
- **Connectivity:** Choose **Public IP** (Simplest for labs) or Private Service Access.
- Click "Create & Continue".

4. Connectivity Method

- Select **VPC Network Peering** (or Allowlist if using Public IPs).
- **VPC:** Select default.
- DMS will configure the necessary routes so the Cloud SQL instance can "pull" data from the VM.

5. Test & Create

- Click **Test Connection**.
 - If successful (all checks green), click **Create & Start Job**.
-

Phase 3: Verification & Promotion (Cutover)

1. Verify Sync (The Replication Test)

- **Wait:** Wait for the job status to change from "Full Dump in Progress" to "**CDC in Progress**" (or "Ready to Promote").
- **Test:**

Connect to your **On-Prem VM** and run:

SQL

```
INSERT INTO employees (name, dept_id) VALUES ('Sync_Test_User', 1);
```

- 1.
2. Connect to your **Cloud SQL Instance** (Destination). You will see **Sync_Test_User** appear almost instantly.
3. *Note:* You cannot write to the Cloud SQL instance yet; it is effectively a Read Replica.

2. Promotion (The Cutover)

- Once you are ready to switch traffic to Google Cloud:
 - **Stop writing** to the On-Prem database (Read-Only mode).
 - Wait for Replication Lag to hit **0 seconds**.
 - In DMS, click **Promote**.
- **Result:**
 - The replication link is broken.
 - The Cloud SQL instance becomes a **Standalone Primary** (Read/Write).
 - You update your application connection string to use the Cloud SQL IP.

Hybrid-On Prem

1. Hybrid Connectivity Options

This refers to connecting your **On-Premises** infrastructure (or another cloud) to your **Google Cloud VPC** over a private connection.

A. Cloud VPN (Virtual Private Network)

- **Mechanism:** Creates a secure, encrypted tunnel (IPsec) over the public internet.
- **Pros:** Cost-effective, fast setup, sufficient for most use cases (up to 3 Gbps per tunnel).
- **Cons:** Traffic goes over the public internet (though encrypted), which some strict compliance policies may not allow.

B. Cloud Interconnect

- **Mechanism:** A dedicated physical cable connection between your on-prem data center and Google.
- **Dedicated Interconnect:** You run a cable directly to Google (requires you to be in a Google Colocation facility). Expensive.
- **Partner Interconnect:** You connect to a Service Provider (e.g., Equinix, Verizon) who already has a cable to Google. Useful if you are far from a Google facility.
- **Pros:** Private traffic (bypasses internet), consistent latency, high bandwidth (10G/100G).

C. Cloud Peering (IMPORTANT DISTINCTION)

- **What it is:** A direct connection to Google's *Public Services* (YouTube, Gmail, Drive).
- **What it is NOT:** It does **NOT** give access to your private VPC (VMs). It is not the same as VPC Peering.

2. Cloud VPN Concepts

Types of VPN Connections

1. **Site-to-Site VPN:** Connects an entire network (On-Prem) to another network (GCP). This is the standard for hybrid cloud.
2. **Remote Access (Client-to-Site) VPN:** Connects a single laptop to the GCP network (useful for admins working remotely).

Routing Types: Static vs. Dynamic

1. **Static Routing (Classic VPN):**
 - You manually modify the Routing Table every time you add a subnet.
 - **Scenario:** If you add **10.1.4.0/24** to GCP, you must manually log in to the On-Prem router and add a route for it.
2. **Dynamic Routing (HA VPN):**
 - Uses **Cloud Router** and **BGP (Border Gateway Protocol)**.
 - **Scenario:** You add a new subnet. Cloud Router automatically tells the On-Prem router: "Hey, I have a new range, send traffic here." No manual updates required.

3. Policy-Based vs. Route-Based VPNs

This is the specific distinction you asked about. It determines **how** traffic is selected to go into the tunnel.

A. Policy-Based VPN

- **Concept:** The VPN tunnel is strictly defined by a "Policy" (Traffic Selectors).
- **Traffic Selectors:** You must explicitly list the IP ranges allowed on both sides.
 - *Local Selector:* `10.1.0.0/16` (GCP side)
 - *Remote Selector:* `192.168.1.0/24` (On-Prem side)
- **Behavior:** The firewall/VPN gateway checks every packet. If it matches the specific source/destination pair, it encrypts it. If you change a subnet IP, you must delete and recreate the tunnel.
- **Limit:** Supports specific "Phase 2" selectors. Less flexible.

B. Route-Based VPN (Recommended)

- **Concept:** The VPN tunnel is "wide open."
- **Traffic Selectors:**
 - *Local Selector:* `0.0.0.0/0` (Any)
 - *Remote Selector:* `0.0.0.0/0` (Any)
- **Behavior:** The VPN accepts *anything* you throw at it. The decision of "what goes into the tunnel" is handled purely by the **Routing Table** (Static Routes or BGP).
- **Benefit:** Highly flexible. If you add a new subnet, you just update the route (or let BGP do it). You do not need to touch the VPN configuration itself.

Feature	Policy-Based	Route-Based
Traffic Selectors	Specific CIDRs (e.g., <code>10.1.0.0/24</code>)	Universal (<code>0.0.0.0/0</code>)
Routing Decision	Encoded in the VPN Policy	Decided by Routing Table / BGP
Flexibility	Low (Recreate tunnel on change)	High (Update routes only)
Use Case	Legacy Firewalls	Modern / HA VPN

4. Setting up HA VPN (Dynamic Routing) via CLI

In a Dynamic (HA) setup, we use **Cloud Router** to manage the routes automatically.

Step 1: Create the Cloud Router

This "brain" will speak BGP to your On-Prem router.

Bash

```
gcloud compute routers create my-cloud-router \
--network=vpc-1 \
--region=us-central1 \
--asn=65001 # Google Side ASN
```

Step 2: Create the HA VPN Gateway

This creates the physical-like interface in GCP.

Bash

```
gcloud compute vpn-gateways create my-ha-gateway \
--network=vpc-1 \
--region=us-central1
```

Step 3: Create the VPN Tunnel (Route-Based)

We connect our gateway to the On-Prem Peer IP.

- *Note:* HA VPN requires IKEv2.
- We link it to the router we created in Step 1.

Bash

```
gcloud compute vpn-tunnels create my-tunnel-1 \
--peer-address=[ON_PREM_PUBLIC_IP] \
--ike-version=2 \
--shared-secret=[YOUR_SECRET_KEY] \
--router=my-cloud-router \
--vpn-gateway=my-ha-gateway \
--interface=0 \
--region=us-central1
```

Step 4: Configure BGP Interface and Peer

Now we tell the Cloud Router to actually talk to the On-Prem router over that tunnel.

Bash

```
# 1. Add an interface to the Cloud Router for this tunnel
gcloud compute routers add-interface my-cloud-router \
--interface-name;if-tunnel-1 \
--vpn-tunnel=my-tunnel-1 \
--ip-address=169.254.0.1 \
--mask-length=30 \
--region=us-central1
```

2. Add the BGP Peer (The On-Prem Router's details)

```
gcloud compute routers add-bgp-peer my-cloud-router \
--peer-name=on-prem-peer \
--interface;if-tunnel-1 \
```

```
--peer-ip-address=169.254.0.2 \
--peer-asn=65002 \
--region=us-central1
```

Summary Scenario (Based on your notes)

If you add **10.1.4.0/24** (VM4) to **vpc-1**:

- **Static/Policy-Based VPN:** You must manually log into the On-Prem router and add a route for **10.1.4.0/24** pointing to the VPN tunnel.
- **Dynamic/Route-Based VPN:** The **Cloud Router** automatically sends a BGP update: "*Hello On-Prem, I have a new route 10.1.4.0/24.*" The On-Prem router updates itself instantly.

Disaster Recovery

GCP Disaster Recovery & High Availability Cheat Sheet

Complete Guide for Professional Cloud Architect Exam

[RTO - How fast you must recover](#)

[RPO - How much data loss is acceptable](#)

DR Fundamentals & Terminology {#dr-fundamentals}

Key Metrics

Metric	Definition	Typical Values	Impacts
RTO (Recovery Time Objective)	Maximum acceptable downtime	Minutes to hours	Architecture complexity, cost
RPO (Recovery Point Objective)	Maximum acceptable data loss	Seconds to hours	Backup frequency, replication
SLO (Service Level Objective)	Internal target for availability	99.9% - 99.99%	Service design
SLA (Service Level Agreement)	Contractual guarantee	99.5% - 99.95% (GCP)	Financial penalties if missed

Availability Calculations

Availability % | Downtime/Year | Downtime/Month | Downtime/Week

99%	3.65 days	7.2 hours	1.68 hours
99.5%	1.83 days	3.6 hours	50.4 minutes
99.9%	8.76 hours	43.2 minutes	10.1 minutes
99.95%	4.38 hours	21.6 minutes	5.04 minutes
99.99%	52.56 minutes	4.32 minutes	1.01 minutes
99.999%	5.26 minutes	25.9 seconds	6.05 seconds

DR Strategy Cost & Complexity

Strategy	Cost	Complexity	RTO	RPO	Data Loss Risk
Backup & Restore	\$	Low	Hours	Hours	High
Pilot Light	\$\$	Medium	Minutes	Minutes	Medium
Warm Standby	\$\$\$	Medium	Minutes	Seconds	Low
Hot Standby	\$\$\$\$	High	Seconds	Near-0	Very Low
Active-Active	\$\$\$\$\$	Very High	None	None	None

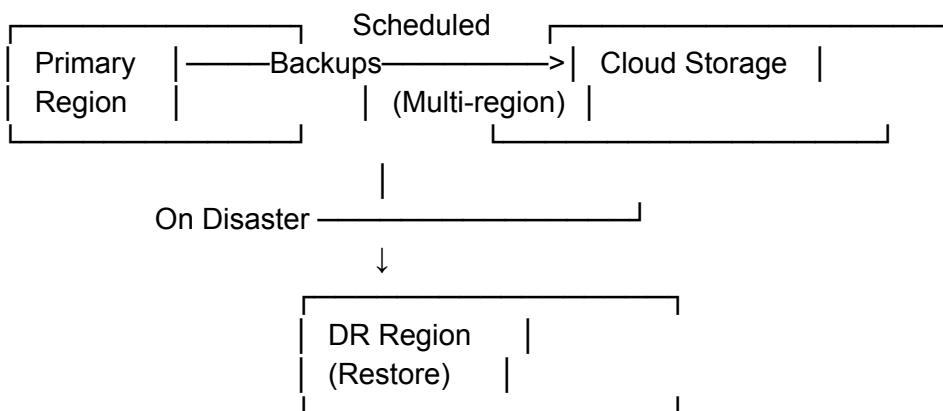
DR Strategy Patterns {#dr-strategy-patterns}

1. Backup & Restore (Cheapest)

When to Use:

- Non-critical workloads
- RTO: 1-24 hours acceptable
- RPO: Hours acceptable
- Cost is primary concern

Architecture:



Implementation:

- Automated backup to Cloud Storage (multi-region or dual-region)
- Infrastructure as Code (Terraform) for rapid provisioning
- Documented runbooks for restoration
- Regular DR testing (quarterly)

Typical Metrics:

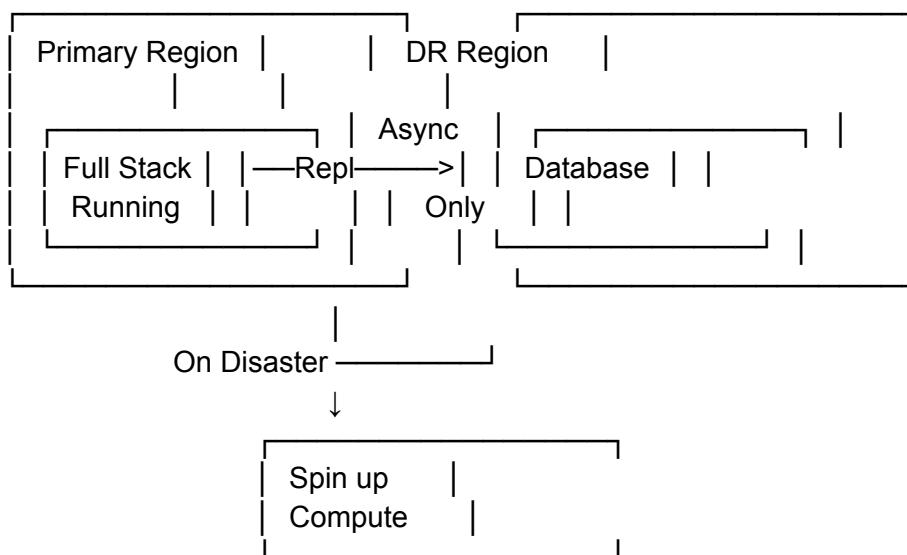
- RTO: 1-4 hours
 - RPO: 1-24 hours
 - Cost: 10-20% of production
-

2. Pilot Light (Minimal Active Resources)

When to Use:

- Moderate criticality
- RTO: 15-60 minutes
- RPO: 5-15 minutes
- Balance of cost and recovery speed

Architecture:



Implementation:

- Database continuously replicated to DR region (read replica)
- Compute resources defined in IaC, not running
- DNS/Load balancer ready to switch
- Automated scripts to provision compute on failure

Typical Metrics:

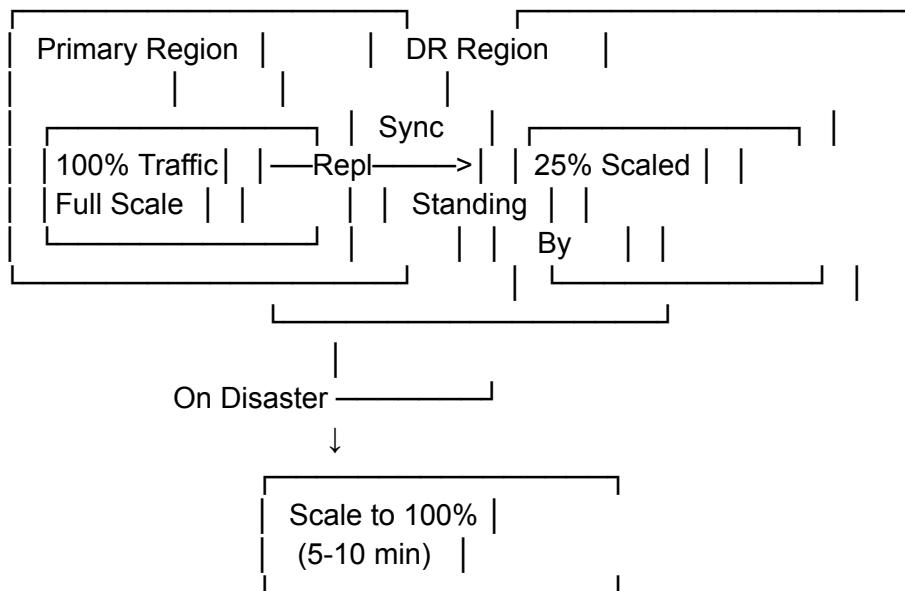
- RTO: 15-60 minutes
 - RPO: 5-15 minutes
 - Cost: 30-40% of production
-

3. Warm Standby (Scaled-Down Active)

When to Use:

- Business-critical applications
- RTO: 5-15 minutes
- RPO: 1-5 minutes
- Can accept reduced capacity initially

Architecture:



Implementation:

- Full stack running in DR region at reduced capacity (25-50%)
- Continuous replication (async or sync)
- Autoscaling configured to scale up on demand
- Load balancer ready to shift traffic

Typical Metrics:

- RTO: 5-15 minutes (scale-up time)
- RPO: 1-5 minutes
- Cost: 50-70% of production

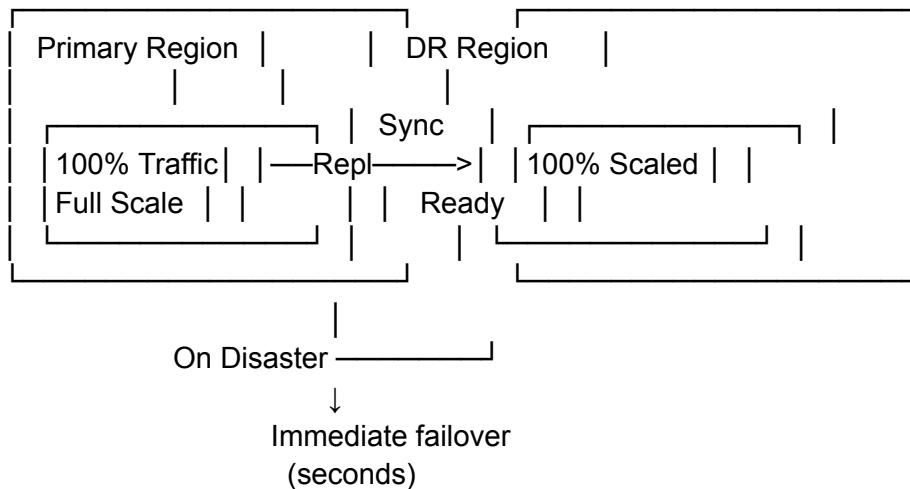
4. Hot Standby (Full Active)

When to Use:

- Mission-critical applications

- RTO: 1-5 minutes
- RPO: Seconds
- Zero data loss acceptable risk

Architecture:



Implementation:

- Full production capacity in both regions
- Synchronous or near-synchronous replication
- Automated health checks and failover
- One region idle (not serving traffic)

Typical Metrics:

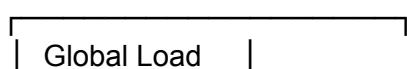
- RTO: 1-5 minutes
- RPO: <1 minute
- Cost: 90-100% of production

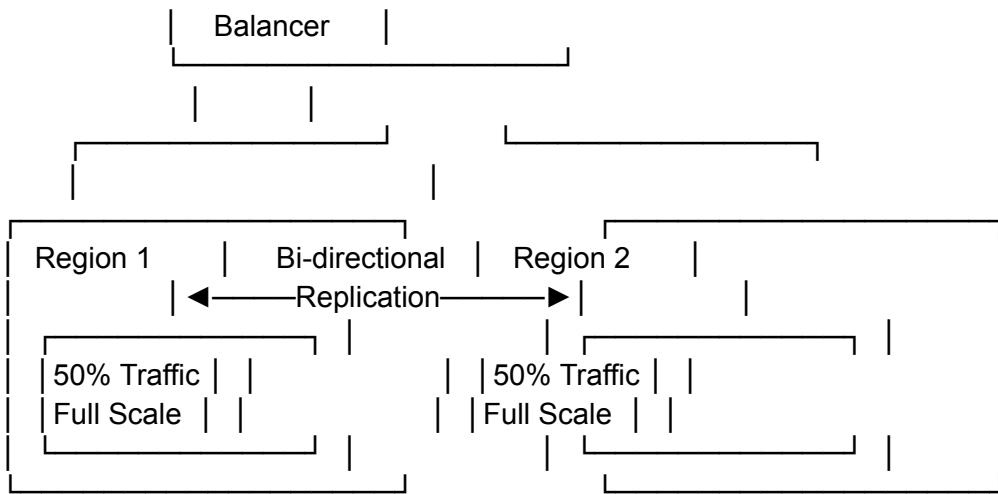
5. Active-Active (Multi-Region)

When to Use:

- Zero-downtime requirements
- Global user base
- RTO: 0 (no downtime)
- RPO: 0 (no data loss)

Architecture:





Implementation:

- Traffic distributed across multiple regions
- Multi-region database (Cloud Spanner, multi-region Firestore)
- Global load balancing
- Conflict resolution for writes

Typical Metrics:

- RTO: 0 (transparent failover)
- RPO: 0 (synchronous replication)
- Cost: 200%+ of single-region

Compute Resources DR/HA {#compute-resources}

Google Kubernetes Engine (GKE)

High Availability Configuration

Feature	Single-Zone	Multi-Zonal	Regional	Multi-Region
Control Plane	Single zone	Single zone	Multi-zone (HA)	Multiple clusters
Worker Nodes	Single zone	Multiple zones	Multiple zones	Multiple regions
Availability	99.5% SLA	99.9% SLA	99.95% SLA	99.99%+ (custom)
RTO	Hours	10-30 min	5-10 min	<5 min

RPO	Depends on backup	Depends on backup	Depends on backup	Near-0
Use Case	Dev/Test	Production	Critical Production	Global/Mission-critical

DR Strategies for GKE

1. Backup & Restore

Configuration

Tool: Backup for GKE (or Velero)

Backup Frequency: Every 4-12 hours

Backup Scope:

- Kubernetes resources (Deployments, Services, ConfigMaps, Secrets)
- Persistent Volumes
- Custom Resource Definitions

Storage: Cloud Storage (multi-region bucket)

Implementation

```
apiVersion: gkebackup.cnrm.cloud.google.com/v1beta1
```

```
kind: BackupPlan
```

```
spec:
```

```
  cluster: my-gke-cluster
```

```
  retentionPolicy:
```

```
    backupRetainDays: 30
```

```
  backupSchedule:
```

```
    cronSchedule: "0 */6 * * *" # Every 6 hours
```

```
  backupConfig:
```

```
    includeVolumeData: true
```

```
    includeSecrets: true
```

Metrics:

- RTO: 1-2 hours (restore time)
- RPO: 4-12 hours (backup frequency)
- Cost: ~\$0.026/GB/month (storage)

2. Regional Cluster (Recommended for Production)

```
# Regional cluster spans 3 zones automatically
gcloud container clusters create production-cluster \
--region=us-central1 \
--num-nodes=1 \ # Per zone = 3 total
--enable-autorepair \
--enable-autoupgrade \
```

```
--enable-autoscaling --min-nodes=1 --max-nodes=10

# Regional persistent disks
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: regional-pd
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  replication-type: regional-pd # Replicated across 2 zones
```

Metrics:

- RTO: <5 minutes (automatic failover)
 - RPO: 0 (synchronous replication)
 - SLA: 99.95%
 - Cost: Same as zonal (nodes) + ~2x for regional PDs
-

3. Multi-Region Active-Active

```
# Architecture
Primary Cluster: us-central1 (regional)
Secondary Cluster: europe-west1 (regional)
Global Load Balancer: Routes based on proximity
Database: Cloud Spanner (multi-region) or replicated Cloud SQL
```

```
# Config Sync for consistency
apiVersion: configmanagement.gke.io/v1
kind: ConfigManagement
spec:
  git:
    syncRepo: https://github.com/company/k8s-configs
    syncBranch: main
    secretType: none
  policyController:
    enabled: true
```

Metrics:

- RTO: 0 (transparent failover)
 - RPO: 0 (depends on database replication)
 - Cost: 2x infrastructure + Global LB
-

GKE Backup Checklist

What to Backup	Tool	Frequency	Retention
Cluster config	Terraform/Config Connector	On change	Version control
Workload resources	Backup for GKE	6-12 hours	30 days
Persistent volumes	Backup for GKE	6-12 hours	30 days
ConfigMaps/Secrets	Backup for GKE	6-12 hours	30 days
Helm charts	Git/Artifact Registry	On release	Indefinite
Custom CRDs	Backup for GKE	6-12 hours	30 days

Compute Engine (VMs)

High Availability Configuration

1. Single Zone (No HA)

```
# Standard VM
Availability: ~99.5% (no SLA without committed use)
SPOF: Entire zone
RTO: Manual VM recreation (10-30 min)
RPO: Last snapshot
```

2. Multi-Zone with Managed Instance Groups (MIG)

```
# Create regional MIG
gcloud compute instance-groups managed create web-mig \
--region=us-central1 \
--template=web-template \
--size=3 \
--health-check=http-health-check \
--initial-delay=300

# Autoscaling
gcloud compute instance-groups managed set-autoscaling web-mig \
--region=us-central1 \
--max-num-replicas=10 \
--min-num-replicas=3 \
--target-cpu-utilization=0.6

# Distribution across zones
zones: us-central1-a, us-central1-b, us-central1-c (automatic)
```

Metrics:

- Availability: 99.9%+
 - RTO: <5 minutes (auto-healing)
 - RPO: Depends on data replication
 - Cost: Same as single zone (just distributed)
-

DR Strategies for Compute Engine

1. Snapshot-Based DR

```
# Scheduled snapshots
gcloud compute resource-policies create snapshot-schedule daily-snapshot \
--region=us-central1 \
--max-retention-days=14 \
--on-source-disk-delete=keep-auto-snapshots \
--daily-schedule \
--start-time=02:00

# Attach to disk
gcloud compute disks add-resource-policies my-disk \
--resource-policies=daily-snapshot \
--zone=us-central1-a

# Cross-region snapshot copy (for DR)
gcloud compute snapshots create my-snapshot \
--source-disk=my-disk \
--storage-location=us
```

Metrics:

- RTO: 15-45 minutes (VM creation from snapshot)
 - RPO: Snapshot frequency (hours to daily)
 - Cost: \$0.026/GB/month (standard) or \$0.013/GB/month (archive)
-

2. Custom Image + Terraform

```
# Create custom image
resource "google_compute_image" "app_image" {
  name      = "app-image-v1"
  source_disk = google_compute_disk.boot_disk.self_link

  storage_locations = ["us"] # Multi-region
```

```

}

# DR VM from image (IaC)
resource "google_compute_instance" "dr_vm" {
  name      = "dr-vm"
  machine_type = "n2-standard-4"
  zone      = "us-west1-a"

  boot_disk {
    initialize_params {
      image = google_compute_image.app_image.self_link
    }
  }
}

# Only create on disaster
count = var.disaster_mode ? 1 : 0
}

```

Metrics:

- RTO: 10-20 minutes
 - RPO: Image creation frequency
 - Cost: \$0.085/GB/month (image storage)
-

3. Active-Passive with Load Balancer

```

# Primary region MIG
gcloud compute instance-groups managed create primary-mig \
--region=us-central1 \
--size=3

# DR region MIG (can be size=0 for pilot light)
gcloud compute instance-groups managed create dr-mig \
--region=us-east1 \
--size=1 # Pilot light

# Global load balancer with failover
gcloud compute backend-services create web-backend \
--global \
--load-balancing-scheme=EXTERNAL \
--health-checks=http-health-check \
--failover-ratio=0.2

# Add backends with failover
gcloud compute backend-services add-backend web-backend \

```

```
--instance-group=primary-mig \
--instance-group-region=us-central1 \
--balancing-mode=UTILIZATION \
--max-utilization=0.8 \
--global
```

```
gcloud compute backend-services add-backend web-backend \
--instance-group=dr-mig \
--instance-group-region=us-east1 \
--failover \
--global
```

Metrics:

- RTO: 5-15 minutes (failover + scale)
 - RPO: Depends on data replication
 - Cost: 50-100% extra for standby capacity
-

Compute Engine Backup Checklist

Component	Backup Method	Frequency	Retention	Cross-Region
Boot disk	Snapshot	Daily	14-30 days	Yes (multi-region)
Data disk	Snapshot	4-12 hours	7-90 days	Yes
Custom images	Image creation	On update	Indefinite	Yes
Instance metadata	Terraform state	On change	Git history	Yes (code repo)
Startup scripts	Git/Cloud Storage	On change	Indefinite	Yes
SSH keys	Secret Manager	On change	Indefinite	Yes (auto-replicated)

Cloud Run

High Availability (Built-in)

Default HA Features:

```
# Cloud Run is inherently HA
- Multi-zonal deployment (automatic)
- Auto-scaling (0 to N instances)
```

- Global load balancing (automatic)
- Zero-downtime deployments
- SLA: 99.95% for services with ≥ 2 instances

Configuration for HA:

```
# Deploy with minimum instances (prevents cold starts)
gcloud run deploy myservice \
--image=gcr.io/project/image:tag \
--region=us-central1 \
--min-instances=2 \      # Always  $\geq 2$  for SLA
--max-instances=100 \
--cpu=2 \
--memory=4Gi \
--timeout=300 \
--concurrency=80

# Multi-region deployment
gcloud run deploy myservice --region=us-central1 --image=...
gcloud run deploy myservice --region=europe-west1 --image=...
gcloud run deploy myservice --region=asia-east1 --image=...

# Global Load Balancer with Cloud Run
gcloud compute backend-services create cr-backend --global
# Add each regional Cloud Run as backend (via Serverless NEG)
```

Metrics:

- Availability: 99.95% (with min-instances ≥ 2)
- RTO: 0 (auto-scaling and traffic shifting)
- RPO: 0 (stateless, data in external services)

DR Strategies for Cloud Run

1. Container Image Versioning

```
# Strategy: Keep all image versions
Artifact Registry retention: Indefinite for production tags
Tagging strategy:
- latest (current production)
- v1.2.3 (semantic version)
- git-abc123 (commit SHA)
- stable (last known good)
```

```
# Rollback
```

```
gcloud run deploy myservice \  
--image=gcr.io/project/image:stable \  
--region=us-central1
```

Metrics:

- RTO: 2-5 minutes (deploy previous version)
 - RPO: 0 (stateless)
 - Cost: ~\$0.10/GB/month (Artifact Registry)
-

2. Multi-Region Active-Active

Architecture:

- Deploy identical services in 3+ regions
- Global HTTPS Load Balancer routes traffic
- Cloud CDN for static content
- Cloud Armor for DDoS protection

Configuration:

Primary: us-central1

Secondary: europe-west1, asia-east1

Database: Cloud Spanner (multi-region) or Firestore (multi-region)

Traffic Distribution:

- Proximity-based routing (automatic)
- Automatic failover on health check failure
- No manual intervention needed

Metrics:

- RTO: 0 (transparent failover)
 - RPO: 0 (depends on database)
 - Cost: N x single region + Global LB
-

3. Blue-Green Deployment

```
# Deploy new version (green) with traffic split  
gcloud run deploy myservice-green \  
--image=gcr.io/project/image:v2 \  
--region=us-central1 \  
--no-traffic # Don't send traffic yet
```

```
# Gradually shift traffic
```

```
gcloud run services update-traffic myservice \
--to-revisions=myservice-green=10 # 10% to new version

# Monitor metrics, then increase
gcloud run services update-traffic myservice \
--to-revisions=myservice-green=50

# Full cutover or rollback
gcloud run services update-traffic myservice \
--to-revisions=myservice-green=100 # Success

# OR rollback
gcloud run services update-traffic myservice \
--to-revisions=myservice-blue=100 # Rollback
```

Metrics:

- RTO: <2 minutes (traffic shift)
- RPO: 0
- Cost: Minimal (only during transition)

App Engine

High Availability (Built-in)

Regional Service:

```
# App Engine is regional by default
- Automatic distribution across zones in region
- Auto-scaling (0 to N instances)
- Built-in load balancing
- Zero-downtime deployments
- SLA: 99.95% (Standard), 99.5% (Flexible)

# Cannot change region after creation
# For multi-region: Deploy separate App Engine apps in different projects
```

DR Strategies for App Engine

1. Version Management

```
# Deploy new version without traffic
gcloud app deploy --no-promote --version=v2
```

```
# Traffic splitting (canary)
gcloud app services set-traffic default \
--splits=v1=90,v2=10

# Full cutover
gcloud app services set-traffic default \
--splits=v2=100

# Rollback
gcloud app services set-traffic default \
--splits=v1=100
```

Metrics:

- RTO: 1-3 minutes
 - RPO: 0 (stateless)
 - Cost: No extra cost for versions
-

2. Multi-Region (Multi-Project)

```
# Architecture (complex)
Project 1: App Engine in us-central1
Project 2: App Engine in europe-west1
Global Load Balancer: Routes between projects (requires serverless NEG)
```

Simpler alternative: Cloud Run instead of App Engine for multi-region

```
# Shared data
Database: Cloud Spanner (multi-region)
Storage: Cloud Storage (multi-region buckets)
```

Metrics:

- RTO: Minutes (manual failover between projects)
 - RPO: 0 for application, depends on data
 - Cost: 2x App Engine + shared database
-

3. Backup & Restore

```
# Backup source code
Git repository: Always maintain source control
Build artifacts: Store in Artifact Registry
```

```
# Application configuration  
app.yaml: Version controlled  
Environment variables: Documented and version controlled  
Cron jobs: cron.yaml backed up
```

```
# Data backup  
Firestore: Scheduled exports to Cloud Storage  
Cloud Storage: Dual-region or multi-region buckets
```

Metrics:

- RTO: 30-60 minutes (redeploy)
 - RPO: Depends on data backups
 - Cost: Minimal (source code storage)
-

Database Services DR/HA {#database-services}

Cloud SQL (MySQL, PostgreSQL, SQL Server)

High Availability Configurations

Configuration	Availability	RTO	RPO	Cost Multiplier	Use Case
Single Zone	~99.5%	10-30 min	Minutes to hours	1x	Dev/Test
HA (Same Region)	99.95%	<60 sec	0	~2x	Production
HA + Read Replica	99.95%	<60 sec (primary) 5-15 min (replica promotion)	Seconds	2-3x	Read-heavy workloads
HA + Cross-Region Replica	99.95%	5-15 min	Seconds to minutes	2-3x	DR + read scaling
Multi-Region (Manual)	99.99%+	5-15 min	Seconds	3x+	Mission-critical

1. High Availability Configuration (Regional)

```
# Create HA instance  
gcloud sql instances create prod-db \
```

```
--database-version=POSTGRES_14 \
--tier=db-custom-4-16384 \
--region=us-central1 \
--availability-type=REGIONAL \ # Key: Enable HA
--enable-bin-log \           # For point-in-time recovery
--backup-start-time=03:00 \
--retained-backups-count=7 \
--retained-transaction-log-days=7
```

How it works

Primary: us-central1-a (example)
Standby: us-central1-b (automatic)
Replication: Synchronous (0 RPO)
Failover: Automatic (~60 seconds)

Metrics:

- SLA: 99.95%
- RTO: <60 seconds (automatic failover)
- RPO: 0 (synchronous replication)
- Cost: ~2x single zone

Failover Scenarios:

- Zone failure: Automatic (~60 sec)
- Instance maintenance: Automatic, zero data loss
- User-initiated: `gcloud sql instances failover prod-db`

2. Cross-Region Read Replica (DR)

```
# Create read replica in different region
gcloud sql instances create prod-db-replica \
  --master-instance-name=prod-db \
  --tier=db-custom-4-16384 \
  --region=us-east1 \          # Different region
  --replica-type=READ_POOL \
  --availability-type=REGIONAL # HA for replica too

# Replica lag monitoring
SELECT NOW() - pg_last_xact_replay_timestamp() AS replication_lag;

# Promote replica to standalone (DR scenario)
gcloud sql instances promote-replica prod-db-replica

# This breaks replication - replica becomes primary
```

Metrics:

- RTO: 5-15 minutes (promotion time)
 - RPO: Seconds to minutes (async replication lag)
 - Cost: ~2x primary cost (full instance)
 - Replication lag: Typically <1 second, can be seconds under load
-

3. Point-in-Time Recovery (PITR)

```
# Enabled automatically with transaction logs  
# Can restore to any point in last 7 days (configurable up to 35 days)
```

```
# Restore to specific timestamp  
gcloud sql backups create --instance=prod-db --description="pre-migration"
```

```
# Clone instance to specific point in time  
gcloud sql instances clone prod-db prod-db-restored \  
--point-in-time '2024-01-15T14:30:00.000Z'
```

```
# Or restore from backup  
gcloud sql instances restore-backup prod-db \  
--backup-id=1234567890
```

Metrics:

- RTO: 15-60 minutes (restore time depends on size)
 - RPO: Transaction log frequency (minutes)
 - Cost: ~\$0.09/GB/month (transaction logs)
-

4. Multi-Region DR Strategy

```
# Architecture
```

Primary Region: us-central1

- HA instance (primary + standby)
- Serves all write traffic
- Automatic backups to multi-region Cloud Storage

DR Region: europe-west1

- Cross-region read replica (HA enabled)
- Serves read traffic for EU users
- Ready for promotion on disaster

```
# Failover procedure
```

1. Health check detects primary region failure
2. Promote DR replica: gcloud sql instances promote-replica
3. Update application connection strings (or use Cloud SQL Proxy)

4. Application now writes to DR region
5. Create new replica in original region when recovered

Metrics:

- RTO: 5-15 minutes (manual or automated promotion)
 - RPO: <1 minute (typical replication lag)
 - Cost: 3x single zone (HA primary + HA replica)
-

Cloud SQL Backup Best Practices

```
# Automated backups
gcloud sql instances patch prod-db \
--backup-start-time=03:00 \          # Daily backup time
--retained-backups-count=30 \         # Keep 30 days
--retained-transaction-log-days=7 \   # 7 days of PITR
--enable-bin-log                      # Required for PITR

# On-demand backup before changes
gcloud sql backups create \
--instance=prod-db \
--description="pre-upgrade-backup"

# Export to Cloud Storage (for long-term retention or cross-project)
gcloud sql export sql prod-db gs://my-backups/backup-$(date +%Y%m%d).sql \
--database=mydb

# Verify backups
gcloud sql backups list --instance=prod-db

# Test restoration (quarterly)
gcloud sql instances clone prod-db test-restore-$(date +%Y%m%d) \
--backup-id=BACKUP_ID
```

Backup Strategy:

Backup Type	Frequency	Retention	Storage Location	RPO
Automated backup	Daily	7-365 days	Regional	24 hours
Transaction logs	Continuous	7-35 days	Regional	Minutes
Manual backup	Before changes	As needed	Regional	N/A

Exports to GCS	Weekly	90+ days	Multi-region	Weekly
----------------	--------	----------	--------------	--------

Cloud Spanner (Globally Distributed SQL)

Configuration Options

Configuration	Regions	Availability	RTO	RPO	Write Latency	Use Case
Regional	1 region (3 zones)	99.99%	Seconds	0	<10ms	Single region, low latency
Multi-Region	3+ regions	99.999%	0	0	<100ms	Global, read-heavy
Dual-Region	2 regions	99.99%	0	0	<50ms	Regional redundancy

1. Regional Configuration

```
# Create regional instance
gcloud spanner instances create prod-spanner \
--config=regional-us-central1 \ # 3 zones in us-central1
--nodes=3 \ # Or processing-units=1000
--description="Production database"
```

Characteristics

Zones: 3 in single region (automatic distribution)

Replication: Synchronous across zones

Availability: 99.99% SLA

RTO: <10 seconds (automatic failover)

RPO: 0 (synchronous replication)

Best for:

- Applications in single region
 - Lowest latency requirements (<10ms)
 - Lower cost than multi-region
 - Still globally distributed if needed (but higher latency)
-

2. Multi-Region Configuration

```
# Create multi-region instance
gcloud spanner instances create global-spanner \
--config=nam-eur-asia1 \ # North America + Europe + Asia
```

```
--nodes=6 \          # Distributed across regions
--description="Global database"

# Available multi-region configs
nam3: Iowa, South Carolina, Oregon
nam6: Iowa, South Carolina, Oregon, Los Angeles, Montreal, Toronto
nam-eur-asia1: Iowa, Belgium, Taiwan
eur3: Netherlands, Belgium, Finland
eur6: Netherlands, Belgium, Finland, London, Frankfurt, Zurich

# Characteristics
Replication: Synchronous within leader region, async to others
Read: Fast everywhere (local reads)
Write: Slower (consensus across regions) ~100ms
Availability: 99.999% SLA (five nines)
RTO: 0 (transparent failover)
RPO: 0 (no data loss)
```

Best for:

- Global applications
 - Read-heavy workloads with global users
 - Mission-critical applications
 - Can tolerate ~100ms write latency
-

3. Dual-Region Configuration

```
# Create dual-region instance
gcloud spanner instances create dr-spanner \
--config=nam10 \          # Iowa + South Carolina
--nodes=2
```

```
# Other dual-region options
nam10: Iowa + South Carolina
nam11: Iowa + Oregon
nam12: Oregon + Los Angeles
eur10: Belgium + Finland
eur11: Netherlands + Finland
asia1: Tokyo + Osaka
```

```
# Characteristics
Similar to multi-region but only 2 regions
Lower cost than 3+ region configs
Availability: 99.99% SLA
RTO: 0
RPO: 0
```

Best for:

- Regional redundancy without global distribution cost
 - Balance of HA and cost
 - Moderate geographical distribution
-

Cloud Spanner Backup & Recovery

```
# Create on-demand backup
gcloud spanner backups create prod-backup \
--instance=prod-spanner \
--database=prod-db \
--retention-period=30d

# Create scheduled backups using Cloud Scheduler + Cloud Functions
# Cloud Function code:
from google.cloud import spanner

def backup_spanner(event, context):
    client = spanner.Client()
    instance = client.instance('prod-spanner')
    database = instance.database('prod-db')

    backup_id = f"scheduled-backup-{datetime.now():%Y%m%d-%H%M%S}"
    backup = instance.backup(backup_id, database=database)
    backup.create()

# Restore from backup
gcloud spanner databases create prod-db-restored \
--instance=prod-spanner \
--backup=prod-backup \
--backup-instance=prod-spanner

# Point-in-time recovery (version retention)
# Enable version retention on database
ALTER DATABASE prod_db SET OPTIONS (
    version_retention_period = '7d'
);

# Read historical data
SELECT * FROM Users
FOR SYSTEM_TIME AS OF TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1
HOUR);
```

Backup Metrics:

Feature	Details	RPO	RTO	Cost
On-demand backup	Manual trigger	Backup time	15-60 min	~\$0.036/GB/month
Scheduled backup	Automated (Cloud Functions)	Schedule frequency	15-60 min	~\$0.036/GB/month
Version retention	Historical reads	Seconds	Instant (query)	~\$0.10/GB/month

Cloud Spanner DR Patterns

Pattern 1: Multi-Region (Recommended)

Configuration: nam-eur-asia1

Natural DR: Built-in, no additional setup

Failover: Automatic, transparent

RTO: 0

RPO: 0

Cost: Standard multi-region pricing

Pattern 2: Regional + Backup for DR

Primary: regional-us-central1

DR: Restore from backup to different region if needed

RTO: 15-60 minutes (restore time)

RPO: Backup frequency (hours)

Cost: Lower (regional + backup storage)

AlloyDB (PostgreSQL-compatible)

High Availability Configuration

```
# Create HA cluster
gcloud alloydb clusters create prod-cluster \
--region=us-central1 \
--password=SECURE_PASSWORD
```

```
gcloud alloydb instances create prod-primary \
--cluster=prod-cluster \
--region=us-central1 \
--instance-type=PRIMARY \
--cpu-count=8 \
```

```
--availability-type=REGIONAL # Enables HA
```

How HA works

Primary: Active instance in one zone

Standby: Synchronous standby in different zone

Replication: Synchronous (0 RPO)

Failover: Automatic in <60 seconds

SLA: 99.99% for HA instances

Metrics:

- RTO: <60 seconds
 - RPO: 0 (synchronous)
 - Cost: ~2x single availability
-

Read Replicas for Scaling

Create read pool (up to 20 read replicas)

```
gcloud alloydb instances create prod-read-pool \
--cluster=prod-cluster \
--region=us-central1 \
--instance-type=READ_POOL \
--read-pool-node-count=3 # 3 read replicas
--cpu-count=8
```

Characteristics

Replication: Asynchronous from primary

Distribution: Across zones (automatic)

Load balancing: Automatic across read replicas

Lag: Typically <10ms

Cross-Region Replica (DR)

Create cross-region secondary cluster (DR)

```
gcloud alloydb clusters create dr-cluster \
--region=us-east1 \
--cluster-type=SECONDARY \
--primary-cluster=projects/PROJECT/locations/us-central1/clusters/prod-cluster
```

Characteristics

Replication: Asynchronous (continuous)

Lag: Seconds to minutes (depends on distance and load)

Read-only: Cannot write to secondary

Promotion: Manual process to make primary

```
# Promote secondary to primary (DR scenario)
gcloud alloydb clusters promote dr-cluster \
--region=us-east1
```

Metrics:

- RTO: 5-15 minutes (promotion + app failover)
 - RPO: Seconds to minutes (replication lag)
 - Cost: Full secondary cluster cost
-

AlloyDB Backup & Recovery

```
# Automated backups (default)
gcloud alloydb clusters update prod-cluster \
--region=us-central1 \
--automated-backup-days-of-week=MONDAY,WEDNESDAY,FRIDAY \
--automated-backup-start-times=03:00 \
--automated-backup-window=4h \
--automated-backup-retention-count=14
```

```
# On-demand backup
gcloud alloydb backups create manual-backup-20240115 \
--cluster=prod-cluster \
--region=us-central1
```

```
# Restore to new cluster
gcloud alloydb clusters restore prod-cluster-restored \
--region=us-central1 \
--backup=manual-backup-20240115
```

```
# Point-in-time recovery
gcloud alloydb clusters restore prod-cluster-pitr \
--region=us-central1 \
--cluster=prod-cluster \
--point-in-time=2024-01-15T14:30:00Z
```

Backup Features:

Feature	Details	RPO	RTO
Automated backup	Daily backups, retained 14 days	24 hours	30-60 min
On-demand backup	Manual, retained until deleted	Backup time	30-60 min
Continuous backup	Enabled automatically	Seconds	30-60 min

Point-in-time recovery	Up to 35 days	Seconds	30-60 min
------------------------	---------------	---------	-----------

AlloyDB DR Strategy (Recommended)

Architecture:

- Primary Region: us-central1
 - Primary instance (HA enabled)
 - Read pool (3 replicas)

DR Region: us-east1

- Secondary cluster (continuous replication)
- Read pool (optional, for read traffic)

Backup:

- Automated daily backups (14 days retention)
- Continuous backup for PITR (35 days)
- Cross-region backup copies (optional)

Failover Procedure:

1. Detect primary region failure
2. Promote secondary cluster in us-east1
3. Update DNS or application endpoints
4. Verify data integrity
5. Monitor replication lag to new secondary

Metrics:

- RTO: 5-15 minutes
 - RPO: Seconds to minutes
 - Cost: ~3x primary (HA + full DR cluster)
-

Firestore

High Availability (Multi-Region by Default)

Firestore is multi-region by default

Location Types:

- Multi-region: nam5 (US), eur3 (Europe)
- Regional: us-central1, europe-west1, etc.

Availability:

- Multi-region: 99.999% SLA (5 nines)
- Regional: 99.99% SLA (4 nines)

Replication:

- Multi-region: Replicated across multiple regions
- Regional: Replicated across 3 zones

RTO: 0 (transparent failover)
RPO: 0 (synchronous replication)

Backup & Export

```
# Scheduled export (Cloud Scheduler + Cloud Functions)
gcloud firestore export gs://my-backups/firestore-$(date +%Y%m%d) \
--collection-ids='users,orders'

# OR export all collections
gcloud firestore export gs://my-backups/firestore-$(date +%Y%m%d)

# Import from export
gcloud firestore import gs://my-backups/firestore-20240115

# Point-in-time recovery
# Firestore does NOT have built-in PITR
# Use frequent exports for recovery
# Typical strategy: Export every 6-12 hours
```

Export Strategy:

Frequency	Use Case	Retention	Storage Location
Daily	General backup	30 days	Multi-region bucket
Hourly	Critical data	7 days	Multi-region bucket
Before changes	Deployment safety	90 days	Multi-region bucket
Monthly	Long-term archive	1 year+	Archive storage

Metrics:

- RTO: 30-60 minutes (import time depends on size)
 - RPO: Export frequency (hours)
 - Cost: ~\$0.026/GB/month (Cloud Storage)
-

Bigtable

High Availability Configuration

```
# Create instance with replication
gcloud bigtable instances create prod-bigtable \
```

```
--display-name="Production Bigtable" \
--cluster-config=id=prod-c1,zone=us-central1-a,nodes=3 \
--cluster-config=id=prod-c2,zone=us-central1-b,nodes=3 # HA

# Add additional cluster for multi-region
gcloud bigtable clusters create prod-c3 \
--instance=prod-bigtable \
--zone=europe-west1-b \
--nodes=3

# Replication
Single-cluster: No replication (SPOF)
Multi-cluster (same region): HA within region
Multi-cluster (different regions): Global HA and DR
```

Replication Modes:

Configuration	Clusters	Availability	RTO	RPO	Write Latency	Use Case
Single-cluster	1	99.9%	Hours	Hours	6ms	Dev/Test
2 clusters (same region)	2	99.99%	<1 min	0	10ms	Production HA
3+ clusters (multi-region)	3+	99.999%	0	0	50ms+	Global, mission-critical

Replication Strategies

```
# Single-cluster replication (for HA)
gcloud bigtable app-profiles create ha-profile \
--instance=prod-bigtable \
--route-any \ # Route to any available cluster
--description="HA profile"
```

```
# Multi-cluster replication (for DR)
gcloud bigtable app-profiles create dr-profile \
--instance=prod-bigtable \
--route-to=prod-c1 \ # Primary cluster
--description="DR profile with failover"
```

```
# Eventually consistent replication
All writes replicated to all clusters
Replication lag: Milliseconds to seconds
Conflict resolution: Last-write-wins timestamp
```

Metrics:

- RTO: <60 seconds (automatic failover with route-any)
 - RPO: 0 (writes replicated to all clusters)
 - Replication lag: <1 second typical
 - Cost: N x single cluster cost
-

Bigtable Backup

```
# Create on-demand backup
gcloud bigtable backups create prod-backup-20240115 \
--instance=prod-bigtable \
--cluster=prod-c1 \
--table=my-table \
--retention-period=30d

# Scheduled backups (Cloud Scheduler + gcloud script)
# Cloud Scheduler job runs daily with:
gcloud bigtable backups create backup-$(date +%Y%m%d) \
--instance=prod-bigtable \
--cluster=prod-c1 \
--table=my-table \
--retention-period=30d

# Restore from backup
gcloud bigtable backups restore prod-backup-20240115 \
--destination=restored-table \
--destination-instance=prod-bigtable \
--destination-cluster=prod-c1

# Copy backup to different instance (DR)
gcloud bigtable backups copy prod-backup-20240115 \
--source-instance=prod-bigtable \
--source-cluster=prod-c1 \
--destination-instance=dr-bigtable \
--destination-cluster=dr-c1 \
--destination-project=dr-project
```

Backup Metrics:

Backup Type	RPO	RTO	Cost	Notes
On-demand backup	Hours	15-60 min	~\$0.036/GB/month	Manual trigger

Scheduled backup	Backup frequency	15-60 min	~\$0.036/GB/month	Automated
Multi-cluster replication	0	<1 min	N x cluster cost	No backup needed for HA

Bigtable DR Strategy

Pattern 1: Multi-Cluster Active-Active (Recommended)

Configuration:

- Cluster 1: us-central1-a (3 nodes)
- Cluster 2: us-central1-b (3 nodes) - Same region HA
- Cluster 3: europe-west1-b (3 nodes) - DR region

App Profile:

Production: route-any (automatic failover)

Characteristics:

- Writes go to all clusters
- Reads from nearest/healthiest cluster
- Automatic failover
- No manual intervention

Metrics:

- RTO: 0 (transparent failover)
- RPO: 0 (synchronous to all clusters)
- Cost: 3x single cluster

Pattern 2: Primary + Backup DR

Configuration:

- Primary: Single cluster with periodic backups
- DR: Restore from backup if primary region fails

Backup Strategy:

- Daily full backups to Cloud Storage
- Retain 30 days
- Cross-region backup copies

Metrics:

- RTO: 1-2 hours (restore + validation)
- RPO: 24 hours (daily backup)
- Cost: 1x cluster + backup storage

Memorystore (Redis/Memcached)

Redis High Availability

```
# Standard tier (HA)
gcloud redis instances create prod-redis \
--size=5 \
--region=us-central1 \
--tier=STANDARD \          # HA enabled
--redis-version=redis_7_0 \
--replica-count=1
```

How HA works

Primary: Active instance in one zone

Replica: Standby in different zone (same region)

Replication: Asynchronous

Failover: Automatic (2-3 minutes)

Data Persistence: RDB snapshots + AOF logs

SLA: 99.9%

Tiers:

Tier	Availability	RTO	RPO	Cost	Use Case
Basic	99%	Hours (manual recreation)	Full data loss	1x	Dev/Test, stateless cache
Standard	99.9%	2-3 min	<1 min	~2x	Production, session store

Redis Backup & Export

Import RDB file (initial setup)

```
gcloud redis instances import prod-redis gs://my-backups/redis.rdb \
--region=us-central1
```

Export to Cloud Storage

```
gcloud redis instances export prod-redis gs://my-backups/redis-export-$(date
+%Y%m%d).rdb \
--region=us-central1
```

Automated export using Cloud Scheduler

Schedule daily export at 3 AM

Cloud Scheduler job calls Cloud Functions that executes export command

Point-in-time recovery

Redis Standard tier maintains:

```
# - RDB snapshots (persistence)
# - AOF (append-only file) logs
# Automatic recovery on failover
```

Backup Strategy:

Method	Frequency	Use Case	RPO	RTO
Automatic persistence	Continuous (AOF)	Automatic recovery	<1 min	2-3 min
Manual export	Weekly	Long-term backup	1 week	30-60 min
Pre-change export	Before upgrades	Safety net	N/A	30-60 min

Memcached (No HA)

Memcached characteristics
 Tier: Basic only (no Standard/HA tier)
 Persistence: None (in-memory only)
 Replication: None
 Failover: None
 SLA: None

Use for:

- Pure caching (data can be lost)
- Stateless applications
- Data easily reconstructed

DR Strategy:

- Use Redis Standard instead for HA
- OR accept data loss and repopulate cache
- OR implement application-level redundancy

Database DR Comparison Matrix

Database	Min HA Config	Best HA Config	Typical RTO	Typical RPO	Multi-Region	Auto-Failover	Cost Factor
Cloud SQL	Regional HA	HA + Cross-region replica	1 min / 5-15 min	0 / Second s	Yes (replica)	Yes / Manual	2-3x

Cloud Spanner	Regional	Multi-region	Seconds	0	Yes (native)	Yes	2-4x
AlloyDB	Regional HA	HA + Secondary cluster	1 min / 5-15 min	0 / Seconds	Yes (secondary)	Yes / Manual	2-3x
Firestore	Multi-region	Multi-region	0	0	Yes (native)	Yes	1x
Bigtable	2 clusters	3+ clusters	Seconds	0	Yes (replication)	Yes	2-3x
Memorystore Redis	Standard	Standard	2-3 min	<1 min	No	Yes	2x

Data Analytics DR/HA {#data-analytics}

BigQuery

High Availability (Built-in)

BigQuery is multi-zonal by default

Dataset Locations:

- Multi-region: US, EU (data replicated across regions)
- Regional: us-central1, europe-west1, etc.

Availability:

- Multi-region: 99.99% SLA (for tables in US/EU multi-region)
- Regional: 99.99% SLA

Characteristics:

- Zero infrastructure management
- Automatic replication within location
- No manual failover needed
- Globally distributed query execution

RTO: 0 (transparent to users)

RPO: 0 (synchronous replication)

Dataset Location Strategy

Create multi-region dataset (recommended for DR)

```
bq mk --location=US prod_dataset      # Replicated across US regions
```

```
bq mk --location=EU analytics_dataset # Replicated across EU regions
```

```
# Regional dataset (lower cost, single region)
bq mk --location=us-central1 regional_dataset

# Cross-region copy for DR
bq cp
--destination_kms_key=projects/PROJECT/locations/EU/keyRings/RING/cryptoKeys/KEY \
US:source_dataset.table \
EU:dr_dataset.table
```

Location Best Practices:

Location Type	Availability	Use Case	Cost	DR Strategy
Multi-region (US/EU)	99.99%	Production, mission-critical	Standard	Built-in redundancy
Regional	99.99%	Data residency requirements	Standard	Cross-region table copy
Multi-region + Regional copy	99.99%+	Compliance + DR	2x storage	Automated sync

Backup Strategies for BigQuery

1. Table Snapshots (Time Travel)

```
-- BigQuery retains 7 days of history (configurable up to 7 days)
-- Query historical data
SELECT * FROM `project.dataset.table`
FOR SYSTEM_TIME AS OF TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 HOUR);

-- Restore table from snapshot
CREATE OR REPLACE TABLE `project.dataset.restored_table`
AS SELECT * FROM `project.dataset.table`
FOR SYSTEM_TIME AS OF TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 2 HOUR);

-- OR use bq command
bq cp --snapshot_epoch=1673827200000 \ # Unix timestamp
dataset.source_table \
dataset.restored_table
```

Metrics:

- RPO: Up to 7 days (time travel window)
 - RTO: Minutes (query/copy time)
 - Cost: Included (no additional charge for time travel)
-

2. Table Clones (Zero-cost Backups)

```
-- Create clone (metadata only, no data copy)
CREATE SNAPSHOT TABLE `project.dataset.table_backup`
CLONE `project.dataset.table`;

-- Restore from clone
CREATE OR REPLACE TABLE `project.dataset.table`
CLONE `project.dataset.table_backup`;

-- Schedule clone creation (Cloud Scheduler + bq command)
bq cp --clone \
    project:dataset.table \
    project:dataset_backups.table_$(date +%Y%m%d)
```

Metrics:

- RPO: Clone creation frequency
 - RTO: Seconds (metadata operation)
 - Cost: \$0 (until clone modified or 7 days expire)
 - Storage: Only delta changes after clone creation
-

3. Table Exports (Long-term Backup)

```
# Export to Cloud Storage (Avro recommended for backup)
bq extract --destination_format=AVRO \
    'project:dataset.table' \
    'gs://backups/table-$(date +%Y%m%d)/*.avro'

# Export as Parquet (efficient compression)
bq extract --destination_format=PARQUET \
    'project:dataset.table' \
    'gs://backups/table-$(date +%Y%m%d)/*.parquet'

# Scheduled exports using Cloud Scheduler + Cloud Functions
def export_bigquery_table(event, context):
    from google.cloud import bigquery
    client = bigquery.Client()

    table_ref = 'project.dataset.table'
```

```

destination = f'gs://backups/table-{date.today():%Y%m%d}/*.avro'

job_config = bigquery.ExtractJobConfig()
job_config.destination_format = bigquery.DestinationFormat.AVRO

extract_job = client.extract_table(table_ref, destination, job_config=job_config)
extract_job.result() # Wait for completion

```

Export Strategy:

Frequency	Format	Retention	Storage Class	Use Case
Daily	Avro	30 days	Standard	Recent backup
Weekly	Avro	90 days	Nearline	Medium-term
Monthly	Parquet	7 years	Coldline	Long-term archive, compliance

Metrics:

- RPO: Export frequency
 - RTO: 30-60 minutes (restore from Cloud Storage)
 - Cost: Cloud Storage pricing (~\$0.020/GB/month for Standard)
-

4. Cross-Region Replication

```

# Use scheduled queries to replicate data across regions
bq query --use_legacy_sql=false --destination_table='EU:dr_dataset.table' \
--replace \
--schedule='every 6 hours' \
'SELECT * FROM `US:prod_dataset.table`'

# OR use Data Transfer Service
# Create transfer config for cross-region table copy
gcloud transfer configure \
--source-table=US:prod_dataset.table \
--destination-table=EU:dr_dataset.table \
--schedule='every 6 hours'

```

Metrics:

- RPO: Replication frequency (hours)
- RTO: 0 (data already in DR region)
- Cost: 2x storage + query processing

BigQuery DR Best Practices

Strategy 1: Multi-Region + Time Travel (Recommended for most)

Location: US or EU multi-region

Backup: 7-day time travel (built-in)

DR: Automatic within multi-region

Cost: Standard pricing

RPO: 7 days (time travel)

RTO: Minutes

Strategy 2: Multi-Region + Scheduled Clones

Location: US or EU multi-region

Backup: Daily/weekly clones to backup dataset

Retention: 30-90 days of clones

Cost: Only delta storage

RPO: Clone frequency

RTO: Seconds

Strategy 3: Regional + Cross-Region Replication

Primary: us-central1

DR: europe-west1 (scheduled query every 6 hours)

Backup: Daily exports to Cloud Storage

Cost: 2x storage + query processing

RPO: 6 hours (replication lag)

RTO: 0 (DR data ready)

Strategy 4: Compliance/Long-term

Primary: Multi-region dataset

Backup:

- Daily exports to Cloud Storage (30 days Standard)
- Monthly exports to Coldline (7 years)
- Cross-region copy for critical tables

Cost: Primary + Cloud Storage (tiered)

RPO: 24 hours (daily export)

RTO: 30-60 minutes

Dataflow

High Availability (Managed)

Dataflow is fully managed with built-in HA

Characteristics:

- Regional service
- Automatic worker management
- Automatic retry on worker failure
- No instance management

- Checkpointing for streaming jobs

Availability: 99.9% for streaming jobs (no SLA for batch)

RTO: Minutes (automatic worker restart)

RPO: Depends on checkpointing frequency

Streaming Pipeline DR

Create streaming pipeline with HA

```
python dataflow_job.py \
--project=my-project \
--region=us-central1 \
--runner=DataflowRunner \
--streaming \
--enable-streaming-engine \      # Better performance and HA
--num-workers=3 \
--max-num-workers=10 \
--autoscaling-algorithm=THROUGHPUT_BASED
```

DR strategy for streaming pipelines

Architecture:

Primary Region: us-central1 (Dataflow job)

DR Region: us-east1 (standby job)

Source: Pub/Sub (multi-region topic)

Sink: BigQuery (multi-region dataset)

Failover approach

1. Active-Passive:

- Primary job runs in us-central1
- Standby job ready to start in us-east1
- Pub/Sub subscriptions can be switched

2. Active-Active:

- Both regions running
- Use regional Pub/Sub subscriptions
- Idempotent writes to BigQuery (avoid duplicates)

Metrics:

- RTO: 5-10 minutes (start new job in DR region)
 - RPO: Minutes (based on checkpointing)
 - Cost: 1x for active-passive, 2x for active-active
-

Batch Pipeline DR

Batch jobs are stateless and easily rerunnable

DR Strategy:

- Store job definitions in Git/Cloud Source Repositories
- Use Terraform for job orchestration
- Retry failed jobs automatically
- No special DR setup needed

Recovery:

1. Job fails in primary region
2. Automatic retry (built-in)
3. OR trigger job in different region (Cloud Scheduler/Composer)

RTO: Job re-execution time (minutes to hours)

RPO: 0 (batch jobs reprocess data)

Dataproc (Managed Hadoop/Spark)

High Availability

Standard cluster (no HA)

```
gcloud dataproc clusters create standard-cluster \
--region=us-central1 \
--zone=us-central1-a \
--master-machine-type=n2-standard-4 \
--worker-machine-type=n2-standard-4 \
--num-workers=3
```

HA cluster (3 master nodes)

```
gcloud dataproc clusters create ha-cluster \
--region=us-central1 \
--num-masters=3 # HA mode \
--master-machine-type=n2-standard-4 \
--worker-machine-type=n2-standard-4 \
--num-workers=5
```

HA Characteristics

Masters: 3 masters in different zones

High availability: Automatic failover for master

Metadata: Stored in Hive Metastore (HA)

Data: Stored in Cloud Storage (not HDFS)

SLA: None (regional availability)

Metrics:

- RTO: Minutes (master failover)

- RPO: 0 (data in Cloud Storage)
 - Cost: ~3x master cost
-

Dataproc DR Strategy

Recommended: Ephemeral clusters + Cloud Storage

Architecture:

Data: Cloud Storage (multi-region buckets)

Compute: Ephemeral Dataproc clusters

Metadata: Separate Hive Metastore (Cloud SQL)

Jobs: Defined in code (Git/Cloud Source Repositories)

DR Approach:

1. Store all data in Cloud Storage (not HDFS)
2. Use ephemeral clusters (create/destroy per job)
3. Metastore in Cloud SQL with HA and cross-region replica
4. On DR: Spin up new cluster in DR region, same Cloud Storage

Benefits:

- No persistent cluster to back up
- Data already in durable storage
- Can recreate in any region
- Cost-effective (pay only when running)

Metrics:

RTO: 5-15 minutes (cluster creation)

RPO: 0 (data in Cloud Storage)

Cost: Minimal (no idle clusters)

Pub/Sub

High Availability (Multi-Region)

Pub/Sub is global by default

Characteristics:

- Global service (not regional)
- Automatic replication across zones
- No cluster management
- Messages stored redundantly

Availability: 99.95% SLA

RTO: 0 (automatic failover)

RPO: 0 (at-least-once delivery guarantee)

Message Retention & Recovery

```
# Create topic with message retention
gcloud pubsub topics create prod-topic \
--message-retention-duration=7d      # Max: 31 days

# Create subscription with retention
gcloud pubsub subscriptions create prod-subscription \
--topic=prod-topic \
--message-retention-duration=7d \
--retain-acked-messages           # Keep after ack
--ack-deadline=60

# Seeking (replay messages)
# Replay from timestamp
gcloud pubsub subscriptions seek prod-subscription \
--time=2024-01-15T14:30:00Z

# Replay from snapshot
gcloud pubsub snapshots create backup-snapshot \
--subscription=prod-subscription

# Later, seek to snapshot
gcloud pubsub subscriptions seek prod-subscription \
--snapshot=backup-snapshot
```

DR Strategy:

Pattern: Active-Active (Natural)

- Pub/Sub is global and multi-region
- Publishers and subscribers can be anywhere
- No special DR setup needed

Message Durability:

- Messages replicated across zones automatically
- Retention: Up to 31 days
- Snapshots: For replay/recovery

Recovery Scenarios:

- Publisher failure: Retry with exponential backoff
- Subscriber failure: Messages redelivered (unacked)
- Regional failure: Transparent (Pub/Sub is global)

Metrics:

RTO: 0 (transparent failover)

RPO: 0 (at-least-once delivery)

Cost: No extra for HA (built-in)

Storage Services DR/HA {#storage-services}

Cloud Storage

Storage Classes & Availability

Storage Class	Availability SLA	Use Case	Monthly Cost (per GB)
Standard	99.99% (multi-region) 99.9% (regional)	Hot data, frequent access	\$0.020 (multi) \$0.020 (regional)
Nearline	99.9% (multi-region) 99.0% (regional)	Infrequent access (30+ days)	\$0.010
Coldline	99.9% (multi-region) 99.0% (regional)	Rare access (90+ days)	\$0.004
Archive	99.9% (multi-region) 99.0% (regional)	Long-term archive (365+ days)	\$0.0012

Location Types

Multi-region (recommended for DR)

```
gsutil mb -l US gs://my-bucket-us      # US multi-region
```

```
gsutil mb -l EU gs://my-bucket-eu      # EU multi-region
```

```
gsutil mb -l ASIA gs://my-bucket-asia  # Asia multi-region
```

Dual-region (two specific regions)

```
gsutil mb -l US-CENTRAL1+US-EAST1 gs://my-dual-bucket
```

```
gsutil mb -l EUROPE-NORTH1+EUROPE-WEST4 gs://my-dual-bucket-eu
```

Regional (single region)

```
gsutil mb -l us-central1 gs://my-regional-bucket
```

Location Comparison:

Location Type	Regions	Availability	Use Case	Cost
---------------	---------	--------------	----------	------

Multi-region	2+ automatically	99.99%	Maximum HA, global access	Standard
Dual-region	2 specific	99.99%	Compliance + HA	Standard
Regional	1	99.9%	Low latency, data residency	~10% lower

DR Strategies for Cloud Storage

1. Multi-Region Buckets (Recommended)

```
# Create multi-region bucket with versioning
gsutil mb -l US -c STANDARD gs://prod-data
gsutil versioning set on gs://prod-data
```

Characteristics

Replication: Automatic across regions in US
 Durability: 99.99999999% (11 nines)
 Availability: 99.99% SLA
 RPO: 0 (synchronous replication)
 RTO: 0 (automatic failover)

Best for:

- Mission-critical data
- Global access requirements
- Maximum availability

2. Cross-Region Replication

```
# Transfer between buckets in different regions
gsutil -m rsync -r gs://us-bucket gs://eu-bucket
```

```
# Scheduled replication (Cloud Scheduler + Cloud Functions/Workflows)
# Or use Storage Transfer Service
gcloud transfer jobs create \
  gs://source-bucket gs://destination-bucket \
  --schedule-start-date=2024-01-15 \
  --schedule-repeats-every=1d
```

```
# One-way sync preserves DR copy
```

Metrics:

- RPO: Replication frequency (minutes to hours)
 - RTO: 0 (data already in DR region)
 - Cost: 2x storage + transfer cost
-

3. Object Versioning

```
# Enable versioning
gsutil versioning set on gs://my-bucket

# List versions
gsutil ls -a gs://my-bucket/myfile.txt

# Restore previous version
gsutil cp gs://my-bucket/myfile.txt#1673827200000000 \
  gs://my-bucket/myfile.txt

# Lifecycle rule to delete old versions
{
  "lifecycle": {
    "rule": [
      {
        "action": {"type": "Delete"},
        "condition": {
          "numNewerVersions": 3,
          "isLive": false
        }
      }
    ]
  }
}
```

```
gsutil lifecycle set lifecycle.json gs://my-bucket
```

Metrics:

- RPO: Seconds (version created on write)
 - RTO: Minutes (restore operation)
 - Cost: Storage for all versions (use lifecycle to limit)
-

4. Lifecycle Management + Archive

```
// lifecycle-config.json
{
  "lifecycle": {
```

```

"rule": [
  {
    "action": {"type": "SetStorageClass", "storageClass": "NEARLINE"},
    "condition": {"age": 30, "matchesStorageClass": ["STANDARD"]}
  },
  {
    "action": {"type": "SetStorageClass", "storageClass": "COLDLINE"},
    "condition": {"age": 90, "matchesStorageClass": ["NEARLINE"]}
  },
  {
    "action": {"type": "SetStorageClass", "storageClass": "ARCHIVE"},
    "condition": {"age": 365, "matchesStorageClass": ["COLDLINE"]}
  },
  {
    "action": {"type": "Delete"},
    "condition": {"age": 2555} # 7 years
  }
]
}
}

```

```
# Apply lifecycle policy
gsutil lifecycle set lifecycle-config.json gs://my-bucket
```

Use for:

- Compliance (7-10 year retention)
 - Cost optimization
 - Automatic tiering
-

Object Lock for Compliance

```
# Create bucket with retention policy (WORM)
gsutil mb -l US --retention 7d gs://compliance-bucket
```

```
# Lock retention policy (irreversible!)
gsutil retention lock gs://compliance-bucket
```

```
# Objects cannot be deleted or modified until retention period expires
# Use for: Financial records, healthcare data, legal compliance
```

Persistent Disk (Compute Engine)

High Availability

```
# Persistent Disk Types  
Standard PD: HDD, no replication  
Balanced PD: SSD, no built-in replication  
Performance PD: SSD, no built-in replication  
Regional PD: SSD, replicated across 2 zones in region (HA)
```

Regional Persistent Disk (HA)

```
# Create regional PD  
gcloud compute disks create regional-disk \  
--size=500GB \  
--type=pd-balanced \  
--region=us-central1 \  
--replica-zones=us-central1-a,us-central1-b
```

```
# Attach to VM  
gcloud compute instances attach-disk my-vm \  
--disk=regional-disk \  
--mode=rw \  
--zone=us-central1-a
```

Characteristics

Replication: Synchronous across 2 zones
Failover: Attach to VM in different zone if primary fails
Availability: 99.99% SLA (vs 99.9% for zonal)
Performance: Same as zonal PD

Metrics:

- RTO: 5-10 minutes (attach to new VM)
 - RPO: 0 (synchronous replication)
 - Cost: 2x zonal disk cost
-

Snapshot-Based DR

```
# Create snapshot schedule  
gcloud compute resource-policies create snapshot-schedule daily-snapshot \  
--region=us-central1 \  
--max-retention-days=14 \  
--on-source-disk-delete=keep-auto-snapshots \  
--daily-schedule \  
--start-time=02:00
```

```
# Attach policy to disk  
gcloud compute disks add-resource-policies my-disk \<
```

```
--resource-policies=daily-snapshot \
--zone=us-central1-a

# Create on-demand snapshot
gcloud compute disks snapshot my-disk \
--snapshot-names=pre-upgrade-snapshot \
--zone=us-central1-a \
--storage-location=us          # Multi-region

# Restore from snapshot
gcloud compute disks create restored-disk \
--source-snapshot=pre-upgrade-snapshot \
--zone=us-east1-a             # Can be different region
```

Snapshot Strategy:

Frequency	Retention	Storage Location	Use Case	RPO
Hourly	24 hours	Regional	Fast recovery	1 hour
Daily	14 days	Multi-region	Standard backup	24 hours
Weekly	90 days	Multi-region	Long-term	1 week
Pre-change	Ad-hoc	Multi-region	Safety net	N/A

Metrics:

- RTO: 15-45 minutes (disk creation from snapshot)
 - RPO: Snapshot frequency
 - Cost: \$0.026/GB/month (standard), \$0.013/GB/month (archive)
-

Filestore (NFS)

High Availability

```
# Zonal instance (no HA)
gcloud filestore instances create basic-instance \
--tier=BASIC_HDD \
--file-share=name=share1,capacity=1TB \
--network=name=default \
--zone=us-central1-a

# Regional instance (HA) - Enterprise tier only
```

```
gcloud filestore instances create enterprise-instance \
--tier=ENTERPRISE \
--file-share=name=share1,capacity=1TB \
--network=name=default \
--region=us-central1      # Regional, not zonal
```

Enterprise tier characteristics

Replication: Synchronous across zones

Failover: Automatic

SLA: 99.99%

Performance: Up to 100,000 IOPS, 10 GB/s

Tier	Availability	HA	Capacity	Performance	Cost
Basic HDD	99.9%	No	1TB-63.9TB	1800 IOPS	\$
Basic SSD	99.9%	No	2.5TB-63.9TB	60,000 IOPS	\$\$
High Scale SSD	99.9%	No	10TB-100TB	100,000 IOPS	\$\$\$
Enterprise	99.99%	Yes	1TB-10TB	100,000 IOPS	\$\$\$\$

Backup & Recovery

Create backup (Enterprise and High Scale SSD tiers only)

```
gcloud filestore backups create backup-20240115 \
--instance=enterprise-instance \
--instance-zone=us-central1 \
--file-share=share1 \
--region=us-central1
```

Schedule backups (Cloud Scheduler + gcloud script)

Daily backup job

Restore from backup

```
gcloud filestore instances create restored-instance \
--tier=ENTERPRISE \
--file-share=name=share1,capacity=1TB,source-backup=backup-20240115 \
--region=us-central1
```

Cross-region DR: Manual rsync to Cloud Storage

Schedule rsync job to backup to GCS

```
gsutil -m rsync -r /mnt/filestore gs://backup-bucket/filestore-backup/
```

Metrics:

- RTO: 30-60 minutes (restore time)
 - RPO: Backup frequency (hours to daily)
 - Cost: Backup storage (similar to PD snapshots)
-

Quick Decision Matrix {#decision-matrix}

RTO/RPO Requirements → Strategy

RTO Requirement	RPO Requirement	Recommended Strategy	Typical Services	Cost Factor
Hours	Hours	Backup & Restore	All services	1x
15-60 minutes	5-15 minutes	Pilot Light	Compute + DB replica	1.3-1.5x
5-15 minutes	1-5 minutes	Warm Standby	Compute + DB	1.5-2x
1-5 minutes	<1 minute	Hot Standby	Full stack	2-3x
0 (no downtime)	0 (no data loss)	Active-Active	Multi-region	2-4x

Service Selection Based on HA/DR Needs

High Availability (Single Region):

- Compute: GKE regional cluster, Compute Engine MIG, Cloud Run (min-instances ≥ 2)
- Database: Cloud SQL Regional HA, AlloyDB HA, Bigtable multi-cluster
- Storage: Regional persistent disk, Filestore Enterprise

Multi-Region DR (Manual Failover):

- Compute: GKE multi-region + Global LB, Multi-region Cloud Run
- Database: Cloud SQL + cross-region replica, AlloyDB + secondary cluster
- Storage: Multi-region Cloud Storage, cross-region GCS sync

Multi-Region Active-Active (Automatic):

- Compute: Cloud Run multi-region, GKE multi-cluster with Global LB
- Database: Cloud Spanner multi-region, Firestore multi-region
- Storage: Multi-region Cloud Storage buckets

Cost Optimization for DR

Cost-Effective Strategies:

1. Use regional HA for primary (99.95% SLA)

2. Pilot light in DR region (DB replica only)
3. Backup to multi-region Cloud Storage
4. Infrastructure as Code for rapid rebuild
5. Test DR quarterly, not monthly

Premium Strategies (High Cost):

1. Multi-region active-active
 2. Hot standby at full capacity
 3. Multi-region databases (Spanner, Firestore)
 4. Continuous replication
 5. Global load balancing
-

Exam Scenario Patterns {#exam-scenarios}

Pattern 1: "Minimize Downtime"

Scenario Keywords:

- "Minimize downtime"
- "High availability required"
- "Must stay operational"

Solution Approach:

1. Check RTO requirement (if not specified, assume <15 min)
2. Use regional HA for single-region (99.95%)
3. Use multi-region for global (99.99%+)
4. Enable auto-healing/auto-scaling
5. Use managed services (less operational overhead)

Common Answer Patterns:

- GKE: Regional cluster with autoscaling
 - Database: Cloud SQL Regional HA or Cloud Spanner
 - Compute: Managed Instance Groups across zones
 - Storage: Multi-region Cloud Storage
-

Pattern 2: "Minimize Data Loss"

Scenario Keywords:

- "Cannot lose data"
- "Zero data loss"
- "Ensure data durability"

Solution Approach:

1. Check RPO requirement (if not specified, assume near-zero)
2. Use synchronous replication (Regional HA, Cloud Spanner)
3. Enable continuous backup (transaction logs, PITR)
4. Multi-region storage
5. Versioning for object storage

Common Answer Patterns:

- Database: Cloud SQL HA (synchronous), AlloyDB HA
 - Storage: Regional persistent disks, multi-region Cloud Storage
 - Avoid: Async replication for critical data
 - Enable: PITR, versioning, retention policies
-

Pattern 3: "Cost-Effective DR"

Scenario Keywords:

- "Minimize cost"
- "Budget-conscious"
- "Cost-effective"
- RTO: Hours acceptable

Solution Approach:

1. Use backup & restore strategy
2. Store backups in multi-region Cloud Storage (Nearline/Coldline)
3. Use Infrastructure as Code for rebuild
4. No standby resources
5. Regular testing (quarterly vs monthly)

Common Answer Patterns:

- Snapshots to Cloud Storage Archive
 - Database automated backups (built-in)
 - No cross-region replicas (save cost)
 - Terraform/IaC for rapid provisioning
 - Accept longer RTO for cost savings
-

Pattern 4: "Global Users"

Scenario Keywords:

- "Users in multiple regions"
- "Global application"

- "Low latency worldwide"

Solution Approach:

1. Use multi-region architecture
2. Global Load Balancer (Cloud CDN)
3. Multi-region database (Spanner, Firestore)
4. Deploy compute to multiple regions
5. Proximity-based routing

Common Answer Patterns:

- Cloud Spanner multi-region or Firestore multi-region
 - Cloud Run in 3+ regions with Global LB
 - GKE clusters in multiple regions
 - Multi-region Cloud Storage
 - Cloud CDN for static content
-

Pattern 5: "Compliance/Regulation"

Scenario Keywords:

- "Must comply with..."
- "Regulatory requirements"
- "Data residency"
- "Audit trail"
- Long retention (7-10 years)

Solution Approach:

1. Check data residency requirements (region constraints)
2. Enable comprehensive audit logging
3. Implement encryption (CMEK if required)
4. Use retention policies and Object Lock
5. Regular compliance audits

Common Answer Patterns:

- Regional databases with Organization Policy constraints
 - Cloud Audit Logs with Log Analytics
 - Customer-Managed Encryption Keys (Cloud KMS)
 - Object Lock for immutable storage
 - VPC Service Controls for data perimeter
 - Long-term archives in Coldline/Archive storage
-

Pattern 6: "Migrate from On-Prem"

Scenario Keywords:

- "Migrating from on-premises"
- "Hybrid cloud"
- "Gradual migration"

Solution Approach:

1. Start with lift-and-shift (Migrate for Compute Engine)
2. Use Cloud Interconnect for hybrid connectivity
3. Replicate databases (Database Migration Service)
4. Plan for pilot light initially
5. Gradually refactor to cloud-native

Common Answer Patterns:

- VM migration with Migrate for Compute Engine
 - Database Migration Service for live migration
 - Cloud Interconnect for private connectivity
 - Pilot light DR initially, then warm standby
 - Gradual adoption of managed services
-

Quick Reference: Service-Specific Commands

GKE Backup

```
# Create backup plan
gcloud container backup-restore backup-plans create PLAN_NAME \
--cluster=projects/PROJECT/locations/REGION/clusters/CLUSTER \
--location=REGION \
--all-namespaces \
--include-volume-data \
--include-secrets
```

```
# Restore
gcloud container backup-restore restores create RESTORE_NAME \
--location=REGION \
--backup-plan=PLAN_NAME \
--backup=BACKUP_NAME \
--cluster=projects/PROJECT/locations/REGION/clusters/CLUSTER
```

Cloud SQL HA Setup

```
gcloud sql instances create prod-db \
```

```
--database-version=POSTGRES_14 \
--tier=db-custom-4-16384 \
--region=us-central1 \
--availability-type=REGIONAL \
--backup-start-time=03:00 \
--enable-bin-log
```

```
# Create read replica in different region
gcloud sql instances create dr-replica \
--master-instance-name=prod-db \
--tier=db-custom-4-16384 \
--region=us-east1 \
--availability-type=REGIONAL
```

Bigtable Multi-Cluster

```
# Create instance with multiple clusters
gcloud bigtable instances create prod-bt \
--display-name="Production" \
--cluster-config=id=c1,zone=us-central1-a,nodes=3 \
--cluster-config=id=c2,zone=us-east1-b,nodes=3
```

```
# Create app profile with routing
gcloud bigtable app-profiles create ha-profile \
--instance=prod-bt \
--route-any
```

Cloud Storage Multi-Region

```
# Create multi-region bucket with versioning
gsutil mb -l US -c STANDARD gs://prod-data
gsutil versioning set on gs://prod-data
gsutil lifecycle set lifecycle.json gs://prod-data
```

Key Takeaways for Exam

1. Know the SLAs:

- Single-zone: ~99.5%
- Regional HA: 99.9-99.95%
- Multi-region: 99.99-99.999%

2. Understand trade-offs:

- Cost vs Availability

- Consistency vs Latency (CAP theorem)
- RTO/RPO vs Complexity

3. Managed services have built-in HA:

- Cloud Run, App Engine: Multi-zonal by default
- BigQuery: Multi-region replication
- Cloud Storage: 11 nines durability

4. Regional HA is cost-effective for most production workloads:

- Cloud SQL Regional HA: 99.95%, 2x cost
- GKE Regional: 99.95%, same cost as zonal
- Regional persistent disks: 99.99%, 2x cost

5. Multi-region for mission-critical or global:

- Cloud Spanner: 99.999% for multi-region
- Firestore multi-region: 99.999%
- Active-active deployment: No downtime

6. Backup strategies:

- Frequent automated backups (daily minimum)
- Test restores quarterly
- Cross-region copies for DR
- Long-term archives in Coldline/Archive

7. Common exam gotchas:

- Cloud SQL cross-region replica requires manual promotion
- GKE Autopilot doesn't support some advanced features
- Memorystore Redis Basic tier has no HA
- Bigtable single-cluster has no SLA

Remember: The exam tests your ability to:

- Choose the right service for availability requirements
- Balance cost, complexity, and RTO/RPO
- Design resilient architectures
- Implement proper backup and recovery procedures

Good luck! 

Terraform

Terraform Workflows

1. Core Concepts: How Terraform Works

Terraform is an Infrastructure as Code (IaC) tool that manages cloud resources via APIs.

- **Terraform Core (The Binary):** The `terraform.exe` or binary file on your laptop. It reads your code and calculates what needs to be done. It cannot talk to Google Cloud directly.
 - **The Provider (The Plugin):** To talk to a specific cloud (like GCP, AWS, Azure), Terraform needs a plugin called a **Provider**.
 - *Role:* It acts as a translator. It takes Terraform commands and converts them into Google Cloud API calls (CRUD operations).
 - *Location:* When you run `init`, Terraform downloads the provider (e.g., `registry.terraform.io/hashicorp/google`) into a hidden `.terraform` folder in your working directory.
 - **The Registry:** The public library (`registry.terraform.io`) where HashiCorp hosts all available providers.
-

2. Configuration & Syntax (HCL)

You write code in **HashiCorp Configuration Language (HCL)**.

File Structure:

- `main.tf`: The primary file where you define resources (VPCs, VMs).
- `provider.tf`: (Optional but recommended) Where you configure the Google provider.
- `version.tf`: (Optional) Where you lock specific versions of Terraform or Providers.

Basic Syntax:

Terraform

```
# This is a comment in Terraform
```

```
# 1. Configure the Provider
```

```
provider "google" {  
  project = "my-test-project"  
  region  = "us-central1"  
  # Credentials can be set here, but environment variables are safer (see Section 4)  
}
```

```
# 2. Define a Resource (The actual infra)
```

```
# Syntax: resource "type" "local_name"  
resource "google_compute_network" "vpc1" {  
  name          = "terraform-vpc-demo"  
  auto_create_subnetworks = false
```

```
}
```

3. The Terraform Workflow (5 Main Commands)

1. `terraform init` (Initialize)

- **Action:** Scans your code to see which providers are needed.
- **Result:** Connects to the Registry, downloads the specific provider executable (e.g., `terraform-provider-google`), and stores it in the `.terraform` folder.
- **Note:** You cannot run any other command until you run this.

2. `terraform validate` (Check Syntax)

- **Action:** Checks for syntax errors (missing brackets, typos).
- **Limit:** It does **not** check if the resource logic is valid (e.g., it won't check if "us-central1" is a real region, only that you formatted the string correctly).

3. `terraform plan` (Dry Run)

- **Action:** Compares your code (Desired State) with the real world (Actual State) and the State File.
- **Result:** Shows a "Execution Plan" of what *will* happen.
- **Symbols:** + (Create), - (Destroy), ~ (Update in-place).

4. `terraform apply` (Execute)

- **Action:** Executes the plan against the Google Cloud API.
- **Safety:** It shows the plan again and asks for confirmation (`yes/no`).
- **Result:** Creates the real infrastructure.

5. `terraform destroy` (Teardown)

- **Action:** Reads the state file to find everything it created and deletes it from the cloud.
-

4. Authentication: Connecting Local Terraform to GCP

You asked for the options to authenticate. There are two primary ways:

Option A: Service Account Key (The Hard Way)

- **Best for:** CI/CD Pipelines (Jenkins, Cloud Build).
- **Method:**
 1. Create a Service Account in the GCP Console.
 2. Grant it permissions (e.g., `Owner` or `Editor`).
 3. Download the JSON key file (e.g., `key.json`).

Reference it in your code:

```
Terraform
provider "google" {
  credentials = file("key.json")
```

```
project  = "my-project"
}
```

- **Risk:** Securing the `key.json` file is difficult. If you accidentally push this to GitHub, your cloud is compromised.

Option B: Application Default Credentials (ADC) (The Recommended Way for Locals)

- **Best for:** Developers working on their local laptop.
- **Method:**
 1. Install the Google Cloud SDK (`gcloud`).

Run this specific command in your terminal:

Bash

```
gcloud auth application-default login
```

2. A browser window opens. Login with your Google Gmail/Corporate account.
3. **Done.** Terraform automatically detects the credentials saved on your laptop.
You do **not** need to put `credentials = file(...)` in your Terraform code.

5. State Management (`terraform.tfstate`)

- **Desired State:** What is written in your `main.tf` file (The Dream).
- **Actual State:** What actually exists in Google Cloud (The Reality).
- **The State File (`terraform.tfstate`):** A JSON file on your local machine that maps your code to the real resources.
 - When you run `terraform apply`, Terraform writes the IDs of created resources into this file.
 - When you run `terraform plan` next time, it checks this file to know "Oh, I already created that VPC, I don't need to create it again."
 - **Warning:** Never edit this file manually.

Terraform Resources

Terraform Fundamentals & Blocks

Terraform is an Infrastructure as Code (IaC) tool that allows you to define cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.

1. Fundamental Blocks

A typical Terraform configuration consists of four primary block types:

Block Type	Description
Provider Block	Configures the cloud provider (e.g., Google, AWS) and authentication settings.
Resource Block	The core block used to create, modify, or delete infrastructure components (e.g., VMs, VPCs).
Terraform Block	Configures Terraform itself, including version constraints and backend state storage.
Data Block	Fetches information from existing resources outside of Terraform (read-only).

2. Resource Block

The resource block is the most important component. It instructs Terraform to create specific infrastructure.

Syntax

```
Terraform
resource "<PROVIDER>_<TYPE>" "<LOCAL_NAME>" {
    # Arguments (Inputs)
    name      = "my-resource"
    machine_type = "e2-medium"
}
```

- **Provider Type:** The specific resource kind (e.g., `google_compute_network`).
- **Local Name:** A unique identifier for the resource *within your Terraform code* (e.g., `tf_vpc`). It is not the name of the resource in the cloud.

- **Arguments:** Input parameters you provide (e.g., name, region, size).
- **Attributes:** Output values generated after creation (e.g., the IP address assigned by Google).

Dependencies

Terraform creates resources in parallel unless a dependency exists.

1. **Implicit Dependency:** Automatically created when one resource references an attribute of another.
 - *Example:* `network = google_compute_network.tf_vpc.id` (Terraform knows it must create the VPC first to get the ID).
2. **Explicit Dependency:** Manually defined using the `depends_on` meta-argument.
 - *Example:* `depends_on = [google_storage_bucket.my_bucket]`

3. Terraform Block (Settings)

This block ensures consistency across your team by enforcing specific versions of Terraform and Providers, and configuring where the State File is stored.

```
Terraform
terraform {
  # 1. Terraform Version Constraint
  required_version = ">= 1.2.0"

  # 2. Provider Constraints
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 5.0"
    }
  }

  # 3. Backend Configuration (Remote State)
  backend "gcs" {
    bucket = "my-terraform-state-bucket"
    prefix = "state/dev"
  }
}
```

Version Constraints Reference Table

Constraint	Meaning	Example Logic
= 1.4.2	Exact	Must be exactly version 1.4.2.

<code>!= 1.4.2</code>	Not Equal	Any version except 1.4.2.
<code>>= 1.2.0</code>	Greater/Equal	Version 1.2.0 or newer.
<code><= 1.4.0</code>	Less/Equal	Version 1.4.0 or older.
<code>~> 1.3.0</code>	Pessimistic Constraint	Allows updates to the <i>right-most</i> version number. <code>1.3.x</code> is allowed (e.g., 1.3.9), but <code>1.4.0</code> is NOT.
<code>>= 1.3, < 1.5</code>	Range	Any version between 1.3 (inclusive) and 1.5 (exclusive).

Important Backend Note

- **No Variables:** You **cannot** use variables (e.g., `${var.bucket_name}`) inside the `backend` block. Values must be hardcoded or provided via partial configuration files during initialization.

4. Resource Behavior (Lifecycle)

When you run `terraform apply`, resources undergo one of the following lifecycle events:

Behavior	Description
<code>Creation (+)</code>	The resource does not exist and will be created.
<code>Update In-Place (~)</code>	The resource exists, and the change (e.g., adding a label) can be applied without deleting it.
<code>Destroy and Recreate (-/+)</code>	The change cannot be applied to the existing resource (e.g., changing the Region or Image). Terraform must delete the old one and build a new one.

Destroy (-)	The resource is removed from the configuration and will be deleted from the cloud.
--------------------	--

5. Functions & Dynamic References

Working with Lists (`tolist`)

Sometimes outputs are returned as "Sets" (unordered), but you need a specific item (ordered "List"). You can cast it:

Terraform

```
# Convert a Set of tags to a List and grab the first one (Index 0)
tag_one = tolist(google_compute_firewall.tf_ssh.target_tags)[0]

# Grab the second tag (Index 1)
tag_two = tolist(google_compute_firewall.tf_ssh.target_tags)[1]
```

The `file()` Function & `path.module`

When defining startup scripts or user data, you should load them from external files rather than writing massive strings in your `main.tf`.

- **`path.module`**: A built-in variable that returns the filesystem path of the module (folder) where the current `.tf` file is located. This ensures the path is relative and works on any computer.

Example: Loading a Startup Script

Terraform

```
resource "google_compute_instance" "vm_instance" {
  name = "frontend-vm"

  metadata = {
    # Dynamically reads the content of "startup.sh" located in the same folder
    startup-script = file("${path.module}/startup.sh")
  }
}
```

Why use `${path.module}` ?

If you just use `file("startup.sh")`, Terraform looks for the file relative to where you ran the command (root), which might break if you are running Terraform from a different directory or using modules. Using `${path.module}` guarantees it looks in the same folder as your code.

Terraform Resources and Variables

1. Terraform Resource Lifecycle: How Changes are Applied

When you modify your Terraform code (`main.tf`) and run `terraform apply`, Terraform compares your code against the real infrastructure. Depending on *what* you changed, it chooses one of three actions:

A. Update In-Place (~)

This happens when you change a property that **does not require** the resource to be deleted.

- **Scenario:** You added a new network tag (e.g., `allow-https`) to an existing VM.
- **Behavior:** Terraform simply makes the API call to update the resource. The Resource ID (VM ID) remains the same.

Plan Output:

Terraform

```
~ resource "google_compute_instance" "instance-tf" {
  id  = "projects/.../instances/instance-1"
  name = "instance-1"
~ tags = [
  "allow-http",
  + "allow-https", # "+" indicates this is being added
]
}
```

Plan: 0 to add, 1 to change, 0 to destroy.

-

B. Update with Stop (`allow_stopping_for_update`)

Some changes in Google Cloud (like changing `machine_type` or `min_cpu_platform`) require the VM to be **Stopped** and then **Restarted**.

- **The Problem:** By default, Terraform interprets a required stop as a failure or a need to destroy the resource. It creates an error.
- **The Fix:** You must explicitly add the argument `allow_stopping_for_update = true` inside your `google_compute_instance` block.
- **Behavior:** Terraform stops the VM, updates the RAM/CPU, and starts it again. It does **not** delete the VM.

C. Destroy and Create / Force Replacement (-/+)

This happens when you change a property that **cannot be changed** on an existing resource (Destructive Change).

- **Scenario:** Changing the `zone` of a VM (e.g., `us-central1-a` `\to` `us-central1-b`). You cannot physically move a running VM to a new datacenter building instantly.
- **Behavior:** Terraform **Destroys** the old resource (-) and **Creates** a new one (+) in the new location.

Plan Output:

```
Terraform  
-/+ destroy and then create replacement  
~ zone = "us-central1-a" -> "us-central1-b" # forces replacement  
•
```

2. Terraform Variables: Parameterizing Configuration

Variables allow you to make your code dynamic and reusable across environments (Dev, Test, Prod) without rewriting the `main.tf` file.

Defining Variables (`variables.tf`)

You typically define all variable blocks in a separate file named `variables.tf`.

- **Reserved Keywords:** You cannot name a variable `source`, `version`, `providers`, `count`, `for_each`, `lifecycle`, `depends_on`, or `locals`.

Syntax Structure:

Terraform

```
variable "region" {  
    description = "The GCP region to deploy resources"  
    type        = string      # string, number, bool, list, map  
    default     = "us-central1" # Fallback value if none is provided  
  
    # Optional: Validation block to restrict input  
    validation {  
        condition  = length(var.region) > 4  
        error_message = "The region name is too short."  
    }  
}
```

Calling Variables (`main.tf`)

In your resource configuration, you reference variables using `var.variable_name`.

Terraform

```
provider "google" {  
    region = var.region  
}
```

3. Variable Assignment Strategies

How do you actually pass values to these variables?

1. `terraform.tfvars` (The Auto-Loader)

- **Purpose:** This file is automatically loaded by Terraform when you run `plan` or `apply`.
- **Use Case:** Storing common values used across all environments.

Content:

Terraform

region = "us-central1"

project_id = "my-project-123"

2. Environment-Specific Files (`dev.tfvars`, `prod.tfvars`)

- **Purpose:** Handling different configurations for different environments (e.g., `e2-micro` for Dev, `n1-standard-1` for Prod).
- **Use Case:** Real-time / Production pipelines.

Execution: You must explicitly tell Terraform which file to use via the CLI.

Bash

```
terraform plan -var-file="dev.tfvars"
```

3. CLI Flags (`-var`)

- **Purpose:** Passing values directly in the command line (Runtime).

Use Case: Overriding a single value quickly or testing.

Bash

```
terraform plan -var="machine_type=f1-micro"
```

4. Variable Precedence (The Hierarchy)

If you define the same variable in multiple places, which one wins? Terraform follows a strict order of precedence (Last one wins).

Order from Highest Priority (Winner) to Lowest:

1. **CLI Arguments (`-var` or `-var-file`):**
 - *Example:* `terraform apply -var="machine_type=f1-micro"` overrides everything else.
2. **Auto-loaded variable files (`*.auto.tfvars`):**
 - Files ending in `.auto.tfvars` are loaded alphabetically.
3. **`terraform.tfvars`:**
 - The standard definitions file.
4. **Environment Variables:**
 - System variables starting with `TF_VAR_variable_name`.
5. **default value:**
 - The value written inside the `variable "{}"` block in `variables.tf`.

Scenario Example:

- **Default:** `e2-micro`
- **terrafrom.tfvars:** `e2-small`
- **dev.tfvars:** `n1-standard-1`
- **CLI Runtime:** `f1-micro`

If you run: `terraform plan -var-file="dev.tfvars" -var="machine_type=f1-micro"`

- **Winner:** `f1-micro` (CLI `-var` beats `-var-file`).

If you run: `terraform plan -var-file="dev.tfvars"`

- **Winner:** `n1-standard-1` (Specific file beats `terraform.tfvars`).

If you run: `terraform plan`

- **Winner:** `e2-small` (Takes from `terraform.tfvars`).

Variables & Validations

1. Conditional Logic (Ternary Operator)

Terraform uses ternary operators to make decisions within your configuration. This allows you to set values dynamically based on a condition (True/False).

Syntax: `condition ? "value_if_true" : "value_if_false"`

Real-World Example: You want to run a startup script only if a specific variable is set to `true`. For example, in a "Stage" environment, you might need High Availability scripts, but not in "Dev".

Configuration:

```
Terraform
variable "enable_startup_script" {
  description = "Toggle to enable/disable the startup script"
  type        = bool
  default     = false
}

resource "google_compute_instance" "vm" {
  name = "test-vm"
  # If true, load the script file. If false, set it to null (do nothing).
  metadata_startup_script = var.enable_startup_script ? file("${path.module}/startup.sh") :
  null
}
```

2. Lists (`list(string)`)

A list is an ordered sequence of values. This is essential when a resource argument expects multiple values, such as **Network Tags** or **Firewall Rules**.

Configuration:

```
Terraform
variable "network_tags" {
  type  = list(string)
  default = ["web-server", "http-server", "https-server"]
}

resource "google_compute_instance" "vm" {
  # ...
  tags = var.network_tags
}
```

3. Maps (`map(string)`) – The Lookup Strategy

A map is a collection of **Key-Value pairs**. This is the standard best practice for managing environment-specific configurations (e.g., using small machines for Dev and large machines for Prod).

Scenario: Select the machine type automatically based on the deployment environment.

Step 1: Define the Map (The Dictionary)

Terraform

```
variable "machine_type_map" {  
  description = "Map of environments to machine types"  
  type        = map(string)  
  default     = {  
    "dev"   = "e2-micro"  
    "stage" = "e2-small"  
    "prod"  = "n1-standard-1"  
  }  
}  
  
variable "environment" {  
  description = "Current deployment environment"  
  type        = string  
  default     = "dev"  
}
```

Step 2: Use the Lookup in Resource In your `main.tf`, you dynamically fetch the value using the environment key.

Terraform

```
resource "google_compute_instance" "app_server" {  
  name      = "my-app-vm"  
  # This translates to: var.machine_type_map["dev"] -> "e2-micro"  
  machine_type = var.machine_type_map[var.environment]  
}
```

Step 3: Runtime Override If you run the following command, Terraform overrides `environment` to "prod", which forces the map to look up "prod" and select `n1-standard-1`:

Bash

```
terraform apply -var="environment=prod"
```

4. Objects (**object**) – Grouping Configuration

Instead of creating 10 separate variables for a single VM (e.g., `var.vm_name`, `var.vm_zone`, `var.vm_size`), you can group them into a single **Object**. This keeps your code clean and organized.

Definition (`variables.tf`):

```
variable "vm_config" {
  description = "Configuration settings for the application VM"
  type = object({
    name      = string
    machine_type = string
    zone      = string
    tags      = list(string)
  })

  default = {
    name      = "default-vm"
    machine_type = "e2-medium"
    zone      = "us-central1-a"
    tags      = ["web"]
  }
}
```

Input Values (`terraform.tfvars`):

```
vm_config = {
  name      = "frontend-vm"
  machine_type = "n1-standard-1"
  zone      = "us-east1-b"
  tags      = ["http-server", "frontend"]
}
```

Usage (`main.tf`): Access attributes using dot notation (`var.object.attribute`).

```
resource "google_compute_instance" "vm" {
  name      = var.vm_config.name
  machine_type = var.vm_config.machine_type
  zone      = var.vm_config.zone
  tags      = var.vm_config.tags
}
```

5. List of Objects (`list(object)`) – Multiple Complex Resources

This is the most powerful structure. It allows you to define a **List** of multiple VMs, where each item in the list is a complex **Object** containing that specific VM's details.

Definition (`variables.tf`):

Terraform

```
variable "vm_list" {
  description = "List of VMs to create with their specific configurations"
  type = list(object({
    name      = string
    machine_type = string
    zone      = string
    tags      = list(string)
  }))
}
```

Input Values (`terraform.tfvars`):

Terraform

```
vm_list = [
  # VM 1 Configuration (Index 0)
  {
    name      = "web-server-01"
    machine_type = "e2-micro"
    zone      = "us-central1-a"
    tags      = ["web"]
  },
  # VM 2 Configuration (Index 1)
  {
    name      = "db-server-01"
    machine_type = "n1-highmem-2"
    zone      = "us-central1-b"
    tags      = ["db", "private"]
  }
]
```

Usage (`main.tf`): You access specific VMs using the list index `[0]`, `[1]`, etc. (Note: In real-world scenarios, you would typically loop through this list using `for_each` or `count`, but here is the manual access method).

Terraform

```
# Accessing the first VM defined in the list
resource "google_compute_instance" "first_vm" {
  name      = var.vm_list[0].name      # "web-server-01"
```

```
machine_type = var.vm_list[0].machine_type # "e2-micro"
zone      = var.vm_list[0].zone
tags      = var.vm_list[0].tags
}
```

6. Variables with Validation ([variables.tf](#))

This file defines the rules. If a user tries to input `us-west1` (not in the list) or `5GB` (too small), Terraform will fail immediately before even trying to plan.

Terraform

```
variable "region" {
  description = "The GCP region to deploy resources"
  type       = string
  default    = "us-central1"

  # Validation: Only allow specific allowed regions
  validation {
    condition  = contains(["us-central1", "us-east4", "asia-south1"], var.region)
    error_message = "Region must be one of: us-central1, us-east4, asia-south1."
  }
}

variable "disk_size_gb" {
  description = "Size of the boot disk in GB"
  type       = number
  default    = 20

  # Validation: Must be greater than 10 GB
  validation {
    condition  = var.disk_size_gb > 10
    error_message = "Disk size must be strictly greater than 10GB."
  }
}
```

7. Implementation with String Appending (`main.tf`)

This demonstrates how to dynamically create the **Zone** (`us-central1-a`) by appending `-a` to the **Region** variable (`us-central1`).

Syntax: `"${var.variable_name}-string"`

Terraform

```
resource "google_compute_instance" "vm" {
  name      = "validated-vm"
  machine_type = "e2-medium"

  # -----
  # STRING INTERPOLATION (Appending "-a" to the region)
  # If var.region is "us-central1", this becomes "us-central1-a"
  # -----
  zone = "${var.region}-a"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"

      # Using the validated variable
      size = var.disk_size_gb
    }
  }

  network_interface {
    network = "default"
  }
}
```

Meta-Arguments-1

1. Authentication: Application Default Credentials (ADC)

This is the standard, secure method for local development. It avoids the risk of managing long-lived JSON key files manually.

- **Command:** `gcloud auth application-default login`
 - **Workflow:**
 1. Run the command; a browser window opens.
 2. Login with your Google identity.
 3. A temporary credential file is generated on your machine.
 4. **Terraform's Behavior:** When you run `terraform plan` or `apply`, the `google` provider automatically detects and uses this file. You do **not** need to put `credentials = file("key.json")` in your code.
-

2. Terraform Outputs

Outputs allow you to extract and display information about the resources you just created. This is critical because you often don't know values (like a Public IP or Resource ID) until *after* the resource is created by the cloud provider.

- **Arguments:** Input values you defined (e.g., `name = "my-vm"`).
- **Attributes:** Output values returned by GCP (e.g., `id`, `self_link`, `nat_ip`).

Configuration:

```
Terraform
# 1. Output the VPC ID
output "vpc_id" {
  description = "The unique ID of the VPC"
  value      = google_compute_network.tf_vpc.id
}

# 2. Output the Subnet ID
output "subnet_id" {
  description = "The unique ID of the subnet"
  value      = google_compute_subnetwork.tf_subnet.id
}

# 3. Output the VM's Public IP
output "instance_external_ip" {
  description = "Public IP address of the instance"
  # Accessing nested attributes:
  # network_interface[0] = The first NIC card
  # access_config[0]    = The external configuration block
  # nat_ip            = The actual public IP attribute
```

```
    value = google_compute_instance.tg_gce_vm.network_interface[0].access_config[0].nat_ip
}
```

Viewing Outputs:

- Outputs are printed to the CLI automatically after `terraform apply`.
 - You can view them anytime later by running `terraform output`.
 - *Note:* Use `sensitive = true` in the output block to hide secrets (like passwords) from the logs.
-

3. Terraform Meta-Arguments

Meta-arguments are special configuration blocks available to **all** resource types. They alter the default behavior of Terraform resources.

There are 5 core meta-arguments:

1. `provider`: Manage resources across multiple projects/regions (using aliases).
 2. `count`: Create N identical copies of a resource.
 3. `for_each`: Create resources by iterating over a map or set (Key-Value based).
 4. `lifecycle`: Change create/destroy behavior (e.g., `prevent_destroy`).
 5. `depends_on`: Define explicit dependencies (Order of creation).
-

A. The `provider` Meta-Argument (Multi-Region/Project)

By default, resources use the root `provider "google"` block. To deploy resources to a *different* project or region within the same file, use an **Alias**.

Step 1: Define Providers

```
Terraform
# Default Provider (Project A - US Central)
provider "google" {
  project = "project-a"
  region  = "us-central1"
}

# Alias Provider (Project B - Europe)
provider "google" {
  alias   = "project_b"    # This alias name is the key reference
  project = "project-b"
  region  = "europe-west1"
}
```

Step 2: Apply to Resources

Terraform

```
# Resource in Project A (Uses Default)
resource "google_compute_network" "vpc_project_a" {
  name = "vpc-a"
}

# Resource in Project B (Uses Alias)
resource "google_compute_network" "vpc_project_b" {
  provider = google.project_b # Reference the alias here
  name    = "vpc-b"
}

resource "google_compute_subnetwork" "subnet_project_b" {
  provider = google.project_b
  name    = "subnet-b"
  network = google_compute_network.vpc_project_b.id
}
```

B. The `count` Meta-Argument (Scaling Resources)

`count` allows you to create multiple instances of a resource without duplicating code. It exposes a `count.index` object (0, 1, 2...) to help create unique names.

Scenario: Create 1 Custom VPC, 3 Dynamic Subnets, and 3 VMs.

Step 1: Define Variables

Terraform

```
variable "subnet_cidrs" {
  type  = list(string)
  default = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
}
```

Step 2: Implementation

Terraform

```
# 1. VPC (Single Resource)
resource "google_compute_network" "custom_vpc" {
  name          = "count-demo-vpc"
  auto_create_subnetworks = false
}
```

```

# 2. Multiple Subnets (Dynamic)
resource "google_compute_subnetwork" "subnets" {
  count      = length(var.subnet_cidrs) # Returns 3

  # Uses index to create unique names: subnet-1, subnet-2, subnet-3
  name      = "subnet-${count.index + 1}"

  # Picks the CIDR from the list at index 0, 1, 2
  ip_cidr_range = var.subnet_cidrs[count.index]

  region      = "us-central1"
  network     = google_compute_network.custom_vpc.id
}

# 3. Multiple VMs (Dynamic)
resource "google_compute_instance" "vm" {
  count      = 3

  # Creates names: vm-0, vm-1, vm-2
  name      = "vm-${count.index}"
  machine_type = "e2-micro"
  zone      = "us-central1-a"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }

  network_interface {
    network  = google_compute_network.custom_vpc.id

    # Assigns VM-0 to Subnet-0, VM-1 to Subnet-1, etc.
    subnetwork = google_compute_subnetwork.subnets[count.index].id
  }
}

```

Meta-Arguments-2

1. **for** Expressions (Data Transformation)

The **for** expression is used inside `[...]` (Lists) or `{ ... }` (Maps) to loop over data and transform it. It does **not** create resources; it creates **values** (like a list of names or a map of IPs).

A. List Transformation (Outputting a List)

Goal: Extract just the names of all VMs created.

Terraform

```
output "vm_names" {
  description = "A list of all VM names"
  # Syntax: [for <item> in <collection> : <output_value>]
  value = [for instance in google_compute_instance.tf_gce_vm : instance.name]
}
```

B. Map Transformation (Outputting a Map)

Goal: Create a lookup table where **Key** = VM Name and **Value** = Public IP.

Terraform

```
output "vm_name_to_ip" {
  description = "Map of VM Names to their Public IPs"
  # Syntax: {for <item> in <collection> : <key> => <value>}
  value = {
    for instance in google_compute_instance.tf_gce_vm :
      instance.name => instance.network_interface[0].access_config[0].nat_ip
  }
}
```

C. Complex Map (Object Values)

Goal: **Key** = VM Name, **Value** = Object (containing ID, Zone, IP).

Terraform

```
output "vm_details" {
  value = {
    for instance in google_compute_instance.tf_gce_vm :
      instance.name => {
        id      = instance.instance_id
        zone    = instance.zone
        external_ip = instance.network_interface[0].access_config[0].nat_ip
      }
  }
}
```

2. The `for_each` Meta-Argument (Resource Creation)

If you want to create multiple resources based on a collection (Map or Set), use `for_each`.

Key Rules:

1. **Map vs. Set:** `for_each` only accepts a `map` or a `set(string)`. It does **not** accept a `list`.
 - o *Tip:* If you have a list, wrap it in `toset(var.list)`.
2. **Uniqueness:** Keys must be unique (e.g., unique subnet names).
3. **Conflict:** You cannot use `count` and `for_each` in the same resource block.
4. **Access:**
 - o `each.key`: The key of the map (or the value itself if using a Set).
 - o `each.value`: The value associated with that key (if Map).

Scenario A: Creating Subnets using `map(object)`

Best for when resources have different properties (different regions, different CIDRs).

Variables:

Terraform

```
variable "subnets" {  
  type = map(object({  
    cidr_block = string  
    region    = string  
  }))  
  
  default = {  
    "us-central-subnet" = {  
      cidr_block = "192.168.1.0/24"  
      region    = "us-central1"  
    }  
    "us-east-subnet" = {  
      cidr_block = "192.168.2.0/24"  
      region    = "us-east4"  
    }  
  }  
}
```

Resource Implementation:

Terraform

```
resource "google_compute_subnetwork" "test_subnet" {
```

```

for_each = var.subnets

# each.key = "us-central-subnet"
name = each.key

# each.value = { cidr_block = "...", region = "..." }
region      = each.value.region
ip_cidr_range = each.value.cidr_block
network      = google_compute_network.my_vpc.id
}

```

Scenario B: Creating VMs using `list(string)`

Best for identical resources where only the name changes. Note: Since `for_each` doesn't support lists, we use `toset()`.

Variables:

Terraform

```

variable "vm_names" {
  type  = list(string)
  default = ["vm-instance-1", "vm-instance-2"]
}

```

Resource Implementation:

Terraform

```

resource "google_compute_instance" "tf_gce_vm" {
  # Convert list to set: ["vm-1", "vm-2"]
  for_each = toset(var.vm_names)

  # For sets, each.key and each.value are the same
  name      = each.key
  machine_type = "e2-micro"
  zone      = "us-central1-a"

  boot_disk { initialize_params { image = "debian-cloud/debian-11" } }
  network_interface { network = "default" }
}

```

3. Dependencies: Implicit vs. Explicit (`depends_on`)

Terraform builds a "Dependency Graph" (DAG) to decide which order to create resources.

A. Implicit Dependency (Automatic)

This happens when you reference an attribute of one resource inside another.

- *Example:* Subnet uses `network = google_compute_network.vpc.id`.
- *Result:* Terraform knows it **must** create the Network first, then the Subnet. You do not need to do anything.

B. Explicit Dependency (`depends_on`)

This is used when there is a hidden dependency that Terraform cannot see in the code.

- *Scenario:* An application inside a VM relies on a Cloud Storage bucket existing, but the VM resource itself doesn't reference the bucket.
 - *Usage:* You must manually tell Terraform to wait.
-

4. Real-World Scenario: The IAM Propagation Delay

The Problem: Creating a Service Account (SA) is fast. However, assigning that SA to a VM or creating IAM bindings often fails immediately because IAM systems are "eventually consistent." It takes time (10-60 seconds) for the new Identity to propagate globally.

The Fix: We introduce a `null_resource` to act as a "Sleep Timer" between creation and usage.

Step 1: Create the Service Account

Terraform

```
resource "google_service_account" "app_sa" {  
  account_id  = "my-app-sa"  
  display_name = "Application Service Account"  
}
```

Step 2: The "Sleep Timer" (Implicitly depends on SA)

We use `local-exec` to run a shell command that sleeps. We reference `google_service_account.app_sa.id` (even if we don't need it) to ensure this resource waits for the SA to be created first.

Terraform

```
resource "null_resource" "delay_iam_propagation" {  
  provisioner "local-exec" {  
    command = "echo 'Waiting for IAM propagation...' && sleep 30"  
}
```

```

}

# Implicit dependency: This trigger ensures the sleep happens
# AFTER the SA is created.
triggers = {
  sa_id = google_service_account.app_sa.id
}
}
}

```

Step 3: The VM (Explicitly depends on Sleep)

Now we create the VM. We use `depends_on` to say "Do not start creating this VM until the sleep timer is finished."

Terraform

```

resource "google_compute_instance" "vm" {
  name      = "app-vm"
  machine_type = "e2-micro"
  zone      = "us-central1-a"

  service_account {
    email = google_service_account.app_sa.email
    scopes = ["cloud-platform"]
  }

  # CRITICAL: Wait for the delay to finish before assigning the SA
  depends_on = [
    null_resource.delay_iam_propagation
  ]

  boot_disk { initialize_params { image = "debian-cloud/debian-11" } }
  network_interface { network = "default" }
}

```

Flow:

1. **Start:** Terraform sees VM needs Sleep, Sleep needs SA.
2. **Action:** Create `google_service_account.app_sa`.
3. **Action:** Start `null_resource` (Sleep 30s).
4. **Wait:** VM creation pauses...
5. **Action:** Sleep finishes.
6. **Action:** Create `google_compute_instance.vm` (IAM is now propagated and ready).

Meta-Arguments-3

1. Lifecycle Meta-Arguments

By default, if a resource needs to be replaced (like changing a VM's zone), Terraform deletes the old one first and then creates the new one. Lifecycle arguments allow you to alter this default behavior.

A. `create_before_destroy`

- **Default Behavior:** Destroy old resource ==> Create new resource. (Downtime involved).
- **Modified Behavior:** Create new resource ==> Destroy old resource. (Zero/Reduced Downtime).
- **Scenario:** You are moving a VM from `us-central1-a` to `us-central1-b`.
 1. **Without this flag:** Terraform stops VM-A, deletes it, and then boots up VM-B. The app is down during this time.
 2. **With `create_before_destroy = true`:** Terraform boots up VM-B in the new zone first. Once it is running, it deletes VM-A.

B. `prevent_destroy`

- **Purpose:** Safety lock for critical resources (e.g., Production Database).
- **Behavior:** If you run `terraform destroy` or make a change that forces replacement, Terraform will **fail with an error** and refuse to proceed.
- **Error Message:** "Instance cannot be destroyed".

C. `ignore_changes`

- **Purpose:** Tells Terraform to ignore differences between your code and the actual cloud resource for specific attributes.
- **Real-World Use Case (The Startup Script):**
 1. **Day 1:** You deploy a VM with a startup script: `echo "v1" > /tmp/version`.
 2. **Day 2:** An Admin manually SSHs into the VM or uses the GCP Console to change the metadata script to `echo "v1-hotfix" > /tmp/version` to fix a bug quickly.
 3. **Day 3:** You run `terraform plan`.
 4. **The Conflict:** Terraform sees the manual change and says, "Hey, the cloud state is different from my code! I will overwrite the hotfix back to 'v1'."
 5. **The Fix:** You use `ignore_changes`. Terraform will say, "I see the metadata changed, but I am instructed to ignore it."

Configuration Example:

Terraform

```
resource "google_compute_instance" "vm" {
  name = "prod-vm"
  zone = "us-central1-a"

  lifecycle {
    # 1. Zero downtime replacement
    create_before_destroy = true

    # 2. Prevent accidental deletion
  }
}
```

```

# prevent_destroy = true

# 3. Ignore manual changes to the startup script or tags
ignore_changes = [
  metadata_startup_script,
  tags,
]
}
}

```

2. Data Blocks (`data`)

Resources (`resource`) **create** new infrastructure. Data blocks (`data`) **read** information about existing infrastructure.

Scenario:

You want to create 2 VMs, but you want to distribute them dynamically across available zones (e.g., VM1 in Zone A, VM2 in Zone B) without hardcoding the zone names.

Step 1: Define the Data Source

Fetch the list of available zones in a region.

Terraform

```

data "google_compute_zones" "available_zones" {
  region = "us-central1"
  status = "UP" # Only fetch zones that are online
}

```

Step 2: Use the Data in a Resource

The data source returns a list attribute called `names` (e.g., `["us-central1-a", "us-central1-b", "us-central1-c"]`).

Terraform

```

resource "google_compute_instance" "vm_instance" {
  count = 2
  name = "web-server-${count.index + 1}" # web-server-1, web-server-2

  # -----
  # DYNAMIC ZONE ASSIGNMENT
  # count.index 0 -> picks names[0] (us-central1-a)
  # count.index 1 -> picks names[1] (us-central1-b)
  # -----
  zone = data.google_compute_zones.available_zones.names[count.index]

  machine_type = "e2-micro"

```

```
boot_disk { initialize_params { image = "debian-cloud/debian-11" } }
network_interface { network = "default" }
}
```

3. Local Values (**locals**)

Locals act as **temporary constants** or helper variables within your module. Unlike input variables (**variable**), **locals** **cannot** be overridden by users at runtime (CLI/tfvars). They are calculated inside the code.

Primary Uses:

1. **Simplify complex logic:** Combine multiple variables into one easy-to-read name.
2. **Enforce Naming Conventions:** Ensure every resource follows the company standard.
3. **Common Tags:** Define tags once, apply them everywhere.

Example Scenario:

Your company requires all resources to follow the naming format:
project-environment-app-resource.

Terraform

```
variable "project" { default = "my-company" }
variable "env"    { default = "dev" }
variable "app"    { default = "payment-api" }

# -----
# LOCALS BLOCK
# -----
locals {
  # 1. Construct the standard prefix once
  name_prefix = "${var.project}-${var.env}-${var.app}"

  # 2. Define common labels/tags
  common_tags = {
    owner      = "devops-team"
    cost_center = "101"
    environment = var.env
  }

  # 3. Complex logic (e.g., determine region short code)
  region_short = var.env == "prod" ? "us" : "asia"
}

# -----
# USAGE
# -----
```

```

resource "google_storage_bucket" "main" {
  # Result: "my-company-dev-payment-api-bucket"
  name = "${local.name_prefix}-bucket"

  location = "US"

  # Apply the common tags map
  labels = local.common_tags
}

resource "google_compute_instance" "vm" {
  # Result: "my-company-dev-payment-api-vm"
  name = "${local.name_prefix}-vm"

  machine_type = "e2-micro"
  zone        = "us-central1-a"

  # Merging common tags with specific tags
  labels = merge(local.common_tags, { component = "backend" })

  boot_disk { initialize_params { image = "debian-cloud/debian-11" } }
  network_interface { network = "default" }
}

```

Summary Comparison

Feature	Input Variables (variable)	Local Values (locals)	Data Sources (data)
Source	User Input / <code>tfvars</code> / CLI	Calculated inside code	Fetched from Cloud API
Mutable?	Yes (Overridable)	No (Fixed logic)	Read-Only
Purpose	Parameterize configuration	DRY (Don't Repeat Yourself) logic	Dynamic Querying
Example	<code>var.region</code>	<code>local.name_prefix</code>	<code>data.google_compute_zones.names</code>

Statefile, import and workspaces

1. The Terraform State File (`terraform.tfstate`)

The **State File** is the absolute source of truth for Terraform. It is the "brain" that remembers what infrastructure Terraform has created and how it maps to real-world cloud resources.

- **Function:** It maps your local configuration (code) to the actual resources IDs in the cloud (e.g., mapping `google_compute_instance.vm` to `i-0123456789abcdef0`).
- **Risk:** If this file is lost or corrupted, Terraform loses track of the infrastructure. The resources in the cloud become **orphans** (they still exist, but Terraform cannot manage or delete them).
- **Storage:**
 - **Default:** Stored locally (`local` backend) as `terraform.tfstate`.
 - **Recommended:** Stored in a remote **Backend** (e.g., GCS Bucket, S3) which supports version control, locking, and collaboration.

2. State Synchronization and Drift

"Drift" occurs when the actual infrastructure in the cloud changes outside of Terraform's knowledge (e.g., someone manually deletes a VM in the GCP Console).

How `terraform plan` works

When you run `terraform plan`, Terraform performs three checks:

1. **Refreshes State:** It queries the cloud provider API to see the current status of resources.
2. **Compares:** It compares the **Refresh Result** (Actual Cloud) vs. **State File** vs. **Manifest (Code)**.
3. **Proposes Changes:** It calculates what needs to be done to make the *Cloud* match the *Code*.

Drift Scenarios Table

Below are the detailed scenarios of what happens when there is a mismatch between the **Code**, **State**, and **Actual Cloud**.

Scenario	Manifest (Code)	State File	Actual Cloud Infra	Terraform Plan Result	Explanation
1. Steady State	✓ Resource Exists	✓ Resource Exists	✓ Resource Exists	No Changes	Everything is in sync.
2. Drift (Manual Deletion)	✓ Resource Exists	✓ Resource Exists	✗ Deleted Manually	+ Create (1 to add)	Terraform sees the resource is missing in the cloud but required in code. It will attempt to recreate it.
3. Drift (Manual Creation)	✗ No Code	✗ No State Entry	✓ Created Manually	No Changes	Terraform ignores resources it doesn't know about (unless imported).
4. State Deletion (state rm)	✓ Resource Exists	✗ Removed via CLI	✓ Resource Exists	+ Create (1 to add)	DANGER: Terraform thinks the resource doesn't exist. Running <code>apply</code> will fail with a 409 Conflict / Already Exists error because it tries to create a resource that is already there.
5. Code Deletion	✗ Removed from Code	✓ Resource Exists	✓ Resource Exists	- Destroy (1 to remove)	Terraform sees the resource in State/Cloud but not in Code, so it plans to delete it.

3. Managing Unhealthy Resources (Tainting)

Sometimes a resource exists and is "healthy" according to the cloud API, but it isn't functioning correctly inside (e.g., a VM where the startup script failed, or a Load Balancer failing health checks).

- **The Problem:** `terraform apply` says "No changes" because the ID exists in the state and cloud.
- **The Solution: Tainting.** You force Terraform to mark the resource as "damaged."

Workflow:

1. Identify the bad resource: `terraform state list`.
2. Taint it: `terraform taint google_compute_instance.vm_instance`.
3. Run Plan: Terraform sees the "Tainted" status and plans **1 to destroy** and **1 to add** (Replace).
4. Run Apply: The old resource is destroyed, and a fresh one is created.
5. Note: You can use `terraform untaint` to undo this if it was a mistake.

4. Advanced State Commands

These commands allow you to manipulate the "brain" directly without changing the cloud infrastructure.

Command	Purpose	Micro Detail
<code>terraform refresh</code>	Updates the state file to match real-world infra.	Does not modify infrastructure. It strictly updates <code>terraform.tfstate</code> with the latest data from the cloud API. Note: <code>terraform plan</code> runs this automatically.
<code>terraform state list</code>	Lists all resources tracked in the state file.	Useful to find the exact resource name for tainting or removing.
<code>terraform state show <name></code>	Shows detailed attributes of a specific resource.	Displays IP addresses, IDs, and metadata stored in the state.

terraform state rm <name>	Removes a resource from the state file only .	The resource remains running in the cloud, but Terraform "forgets" it. It is no longer managed.
terraform import <addr> <id>	Brings an existing cloud resource into Terraform state.	Used when you manually created a resource (e.g., Firewall) and want Terraform to manage it. Note: It syncs the State, but you must manually write the Code block .

5. Targeted Operations (-target)

Use this when you want to apply or destroy **specific** resources without scanning or touching the entire infrastructure (e.g., waiting for a huge database to update when you only want to fix a firewall rule).

- **Syntax:** `terraform apply -target=resource_type.resource_name`
- **Scenario:** You have VPC, Subnet, GCE, and Bucket. You want to destroy only the Firewall.
 - Command: `terraform destroy -target=google_compute_firewall.allow_ssh`
 - Result: Only the firewall is checked and destroyed; other resources are ignored.

6. Terraform Workspaces

Workspaces solve the problem of managing multiple environments (Dev, Test, Prod) using a **single code base** without state files overwriting each other.

- **Without Workspaces:** If you use `main.tf` for Dev and then for Prod, the `terraform.tfstate` would get overwritten, or Terraform would try to change Dev resources to match Prod configurations.
- **With Workspaces:** Terraform creates a separate state file directory for each environment.

Workspace Structure

When you enable workspaces, Terraform changes the backend structure:

- **default:** Uses standard `terraform.tfstate`.
- **dev:** Creates a folder `terraform.tfstate.d/dev/terraform.tfstate`.
- **prod:** Creates a folder `terraform.tfstate.d/prod/terraform.tfstate`.

Workspace Commands & Workflow

Command	Description
terraform workspace list	Lists all workspaces. The current one is marked with an asterisk (*).
terraform workspace new <name>	Creates a new workspace (e.g., <code>dev</code>) and instantly switches to it.
terraform workspace select <name>	Switch context to another workspace.
terraform workspace delete <name>	Deletes a workspace (must be empty/destroyed first).

Example Workflow

1. **Initialize:** `terraform workspace new dev`
2. **Deploy Dev:**
 - o `terraform apply -var-file="dev.tfvars"`
 - o Terraform creates Dev resources and saves state in `terraform.tfstate.d/dev/`.
3. **Switch to Prod:** `terraform workspace new prod`
4. **Deploy Prod:**
 - o `terraform apply -var-file="prod.tfvars"`
 - o Terraform creates separate Prod resources and saves state in `terraform.tfstate.d/prod/`.
5. **Result:** You have two independent environments from one set of `.tf` files. changing Dev does **not** impact Prod.

Provisioners

1. Provisioners: The "Last Resort"

In the real world, Terraform is designed for **Infrastructure as Code (IaC)**—creating the skeleton (VPC, VM, DB). It is *not* ideal for **Configuration Management** (installing software, patching, managing files).

- **Best Practice:** Use **Ansible**, **Chef**, **Puppet**, or **Google Startup Scripts**.
- **Terraform Provisioners:** These are a "last resort" feature. Use them only if standard images or startup scripts cannot solve your problem.
 - **Limitation:** Provisioners **only run during resource creation**. If you update the code later, the provisioner does *not* run again unless you destroy and recreate the resource.

2. The Three Types of Provisioners

1. **local-exec:**
 - Runs a command on **your local machine** (where Terraform is running).
 - **Use Case:** writing IP addresses to a local Ansible inventory file.
 2. **file:**
 - Copies files or directories from your **local machine** to the **remote VM**.
 - **Use Case:** Uploading a script or config file.
 3. **remote-exec:**
 - Runs commands on the **remote VM**.
 - **Use Case:** Installing Apache, setting permissions (`chmod +x`), or running a script.
-

3. Scenario Implementation (Step-by-Step)

Goal: Create a VPC, Subnets, Firewall, dynamic SSH keys, and use all three provisioners to set up a VM.

Step 1: Network & Firewall Setup

Terraform

```
resource "google_compute_network" "vpc" {
  name = "custom-vpc"
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "subnets" {
  for_each = var.subnets # Assuming a map variable
  name      = each.key
  ip_cidr_range = each.value.cidr
  region     = each.value.region
  network    = google_compute_network.vpc.id
```

```

}

resource "google_compute_firewall" "allow_rules" {
  name    = "allow-ports"
  network = google_compute_network.vpc.id
  allow {
    protocol = "tcp"
    ports    = ["22", "8080", "9000"]
  }
  source_ranges = ["0.0.0.0/0"]
}

```

Step 2: Dynamic SSH Key Generation

Instead of using a static key file, we generate one on the fly using the `tls` provider.

Terraform

```

# 1. Generate the RSA Key Pair in memory
resource "tls_private_key" "ssh_key" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

# 2. Save Private Key to local file (sensitive)
resource "local_file" "private_key" {
  content      = tls_private_key.ssh_key.private_key_pem
  filename     = "${path.module}/id_rsa"
  file_permission = "0600" # Important for SSH security
}

# 3. Save Public Key to local file
resource "local_file" "public_key" {
  content = tls_private_key.ssh_key.public_key_openssh
  filename = "${path.module}/id_rsa.pub"
}

```

Step 3: VM Creation & Metadata Injection

We must inject the **Public Key** into the VM's metadata so Google allows the connection.

Terraform

```

resource "google_compute_instance" "vm" {
  name        = "ansible-vm"
  machine_type = "e2-medium"
  zone        = "us-central1-a"

  boot_disk {

```

```

initialize_params { image = "debian-cloud/debian-11" }
}

network_interface {
  network = "default"
  access_config {} # Ephemeral Public IP
}

# Inject the generated public key
metadata = {
  ssh-keys = "${var.vm_user}:${tls_private_key.ssh_key.public_key_openssh}"
}

# --- PROVISIONERS START HERE ---

# A. Connection Block (Prerequisite for File/Remote-Exec)
connection {
  type      = "ssh"
  user      = var.vm_user
  host      = self.network_interface[0].access_config[0].nat_ip
  private_key = tls_private_key.ssh_key.private_key_pem
}

# B. File Provisioner (Copy Script)
provisioner "file" {
  # Ternary logic: If vm is 'ansible', copy real script, else copy dummy
  source    = var.vm_type == "ansible" ? "ansible.sh" : "empty.sh"
  destination = "/home/${var.vm_user}/setup.sh"
}

# C. Remote-Exec Provisioner (Run Script)
provisioner "remote-exec" {
  inline = [
    "chmod +x /home/${var.vm_user}/setup.sh",
    var.vm_type == "ansible" ? "./setup.sh" : "echo 'Skipping setup'"
  ]
}

# D. Local-Exec Provisioner (Generate Ansible Inventory)
# Runs on your LAPTOP, not the VM.
provisioner "local-exec" {
  command = "echo '${self.network_interface[0].access_config[0].nat_ip}' >> inventory.ini"
}

```

4. The "Taint" Problem (Forcing Re-runs)

Why Taint? Provisioners only run when the resource is **created**.

- If you modify `ansible.sh` and run `terraform apply`, Terraform says "No changes" because the VM already exists.
- To force Terraform to re-run the provisioner, you must mark the VM as "Tainted". This forces Terraform to **Destroy** and **Recreate** the VM.

A. Taint a Single Resource

Bash

```
terraform taint google_compute_instance.vm
```

Result: The resource is marked for destruction in the state file. The next `terraform apply` will destroy it and create a new one (triggering provisioners).

B. Tainting Multiple Resources (Loop Script) If you used `count` or `for_each`, you might have 10 VMs. Tainting them one by one is tedious. Use this shell script:

Bash

```
#!/bin/bash
# 1. List all instances from state
# 2. Filter for your VM resource name
# 3. Loop through and taint each one
```

```
for instance in $(terraform state list | grep "google_compute_instance.tf-vm-instances"); do
  echo "Tainting instance: $instance"
  terraform taint "$instance"
done
```

After execution: Run `terraform apply`. Terraform will show: **2 to add, 2 to destroy**.

Summary of Best Practices

1. **Avoid Provisioners** if possible. Use standard Images (Packer) or Startup Scripts.
2. **Use `local-exec`** only for things like updating local config files or triggering external scripts.
3. **Use `taint` (or `-replace`)** carefully. It destroys data. Always backup persistent data before tainting a database or stateful VM.
4. **Modern Alternative:** In Terraform v0.15+, `terraform taint` is deprecated in favor of: `terraform apply -replace="google_compute_instance.vm"`

Modules

1. The Concept: Why Modules?

The Problem (Without Modules): Imagine you are an SRE team supporting 4 different application teams.

- **App Team A** needs a VPC, Subnet, Firewall, VM, SQL, Bucket.
- **App Team B** needs the exact same stack.
- **Result:** You copy-paste the same `main.tf`, `variables.tf`, and `outputs.tf` four times. If you need to fix a security bug in the Firewall, you must manually edit 4 different repositories. This violates the **DRY (Don't Repeat Yourself)** principle.

The Solution (With Modules): The SRE team writes a **Standardized Module** (a template) once.

- **Module 1 (Network):** Defines how a secure VPC, Subnet, and Firewall should look.
 - **Module 2 (Compute):** Defines how a standard VM should look.
 - **Application Teams:** They simply "Call" these modules. They don't write resource blocks; they just provide inputs (like "Project ID" and "Region").
-

2. The Folder Structure (Standard Layout)

We separate the logic (Modules) from the implementation (Root).

Plaintext

```
/my-terraform-project
├── main.tf      # (Root) Calls the modules
├── variables.tf # (Root) Defines variables for the teams
├── outputs.tf   # (Root) Displays final IPs/IDs
└── terraform.tfvars # (Root) Team-specific values (e.g., App 1 inputs)

└── modules/    # Directory holding all reusable logic
    ├── vpc/       # --- VPC MODULE ---
    │   ├── main.tf    # Resource: google_compute_network
    │   ├── variables.tf # Inputs: vpc_name, project_id
    │   └── outputs.tf  # Outputs: vpc_id (Crucial for linking)

    ├── subnet/     # --- SUBNET MODULE ---
    │   ├── main.tf    # Resource: google_compute_subnetwork
    │   ├── variables.tf # Inputs: vpc_id, cidr_range, region
    │   └── outputs.tf  # Outputs: subnet_id

    └── gce/        # --- VM MODULE ---
        ├── main.tf    # Resource: google_compute_instance
        ├── variables.tf # Inputs: subnet_id, machine_type
        └── outputs.tf  # Outputs: instance_ip
```

3. Module Implementation (The Code)

A. The VPC Module ([./modules/vpc](#))

This module creates the network and **exports** its ID so others can use it.

[modules/vpc/main.tf](#)

Terraform

```
resource "google_compute_network" "vpc" {
    name          = var.vpc_name
    auto_create_subnetworks = false
}
```

[modules/vpc/outputs.tf](#)

Terraform

```
output "vpc_id" {
    description = "The ID of the VPC created"
    value      = google_compute_network.vpc.id
}
```

B. The Subnet Module ([./modules/subnet](#))

This module needs the VPC ID to know where to place the subnet.

[modules/subnet/variables.tf](#)

Terraform

```
variable "vpc_id" {
    description = "The ID of the VPC where this subnet belongs"
    type      = string
}
# ... define region, subnet_name, cidr ...
```

[modules/subnet/main.tf](#)

Terraform

```
resource "google_compute_subnetwork" "subnet" {
    name          = var.subnet_name
    ip_cidr_range = var.subnet_cidr
    region        = var.region
    network       = var.vpc_id # Using the input variable
```

```
}
```

4. The Root Configuration (Connecting the Dots)

This is where the application team "calls" the modules. The critical part is passing the **Output** of the VPC module as the **Input** to the Subnet module.

.**main.tf** (Root)

Terraform

```
# 1. Call the VPC Module
module "vpc" {
  source  = "./modules/vpc"
  vpc_name = var.vpc_name # Passing value from root variable
}

# 2. Call the Subnet Module
module "subnet" {
  source = "./modules/subnet"

  # Pass basic variables
  subnet_name = var.subnet_name
  subnet_cidr = var.subnet_cidr
  region      = var.region

  # CRITICAL: Pass the Output from the VPC module
  vpc_id = module.vpc.vpc_id

  # Implicit dependency is handled by the reference above,
  # but explicit dependency is good practice for clarity.
  depends_on = [module.vpc]
}

# 3. Call the VM Module
module "gce" {
  source = "./modules/gce"

  # Pass the Output from the Subnet module
  subnet_id  = module.subnet.subnet_id
  machine_type = "e2-medium"
}
```

5. Input Values (**terraform.tfvars**)

Each application team will have their own `terraform.tfvars` file to define their specific environment.

`terraform.tfvars` (App Team 1)

```
Terraform
vpc_name    = "app1-vpc-prod"
subnet_name = "app1-subnet-us"
subnet_cidr = "10.0.1.0/24"
region      = "us-central1"
```

6. Real-World Usage: Remote Modules (Git)

In a professional environment, you do not keep the modules folder inside the application repo. You store modules in a centralized **SRE GitHub Repository**.

- **SRE Repo:** `github.com/my-org/terraform-modules`
- **App Repo:** Just contains `main.tf`.

Versioning Strategy (Tags): We use Git Tags (e.g., `v1.0.0`) to ensure stability. If SREs update the module to `v2.0.0`, it doesn't break App Team 1 because they are pinned to `v1.0.0`.

Calling a Remote Module:

```
Terraform
module "vpc" {
  # Double slash // is required to point to a specific folder in the repo
  source = "git::https://github.com/my-org/terraform-modules//vpc?ref=v1.0.0"

  vpc_name = "app1-vpc"
}
```

Summary Checklist for Modules

1. **Source:** Always define where the code lives (`./local` or `git::remote`).
2. **Variables:** Pass inputs *into* the module.
3. **Outputs:** Return IDs/IPs *out* of the module so other modules can use them.
4. **Versioning:** Always use `?ref vX.Y.Z` for remote modules to prevent breaking changes.

Tab 53

