CLOUD & DEVOPS

GCP Fundamentals

GCP's real resource hierarchy, IAM, Compute Engine, Cloud Storage, a genuinely global VPC network, Cloud SQL & Firestore, Cloud Functions & Cloud Run, global load balancing & Managed Instance Groups, Cloud Monitoring & cost management, and a closing capstone rebuilding AWS Fundamentals' and Azure Fundamentals' own capstone application a third time on GCP's genuinely different architecture — closing the full three-course Cloud Platform Specifics project.

10 Chapters | Philip Osztromok | Generated with Claude

GCP Fundamentals

Chapter 1 · Why GCP? Resource Hierarchy & Global Infrastructure

This is the third and final course in the site's own Cloud Platform Specifics split, deliberately kept structurally parallel to AWS Fundamentals and Azure Fundamentals chapter-for-chapter. GCP's own real chronology holds a genuine surprise: it isn't the youngest of the three.

GCP's Real Origin

In April 2008, Google announced App Engine — a real platform for developing and hosting web applications in Google-managed data centers, and Google's own real first cloud computing service. That real date matters for a genuinely interesting reason: AWS Fundamentals Chapter 1 placed AWS's own real launch in March 2006, and Azure Fundamentals Chapter 1 placed Azure's own real origin in October 2008 (announced) / February 2010 (launched as Windows Azure). GCP's own real April 2008 debut sits chronologically *between* the two — genuinely predating Azure's own public launch, even though GCP is commonly perceived as the youngest of the three major providers today.

Real Global Infrastructure

As of Q1 2024, GCP operates in 40 real regions and 121 real zones. A **region** is a specific geographical location; a **zone** is a real, isolated deployment area within a region — genuinely the failure domain, the same conceptual role an AWS AZ or (where supported) an Azure AZ plays. Google's own real documentation states that most regions have three zones.

> **A GENUINELY MORE UNIFORM REAL MODEL**
>
> Unlike Azure, where Azure Fundamentals Chapter 1 flagged a real, important gotcha — not every Azure region supports Availability Zones at all — GCP's own real model is more consistently uniform: the large majority of GCP regions have real zones (typically three) available. This doesn't mean every single GCP region is guaranteed identical zone support, but the real, common case is far less of a minefield to check than Azure's own split.

The Real GCP Resource Hierarchy

GCP organizes everything into a real, four-level hierarchy:

- **Organization** — the real root, representing a company or entity; created automatically when a Google Workspace or Cloud Identity account holder creates their first project. Acts as the parent to every folder and project.
- **Folder** (optional) — a real, intermediate grouping container between the Organization and Projects, modeling departments, teams, or legal entities, and providing isolation boundaries for delegated administration.
- **Project** — the real, fundamental organizing unit. Every real GCP resource — a Compute Engine VM, a storage bucket — belongs to exactly one Project, no exceptions.
- **Resource** — the individual real services and components living inside a Project.

Real IAM policies applied at any level cascade downward automatically — a role granted at the Organization level applies to every child Folder and Project beneath it, with no need to configure permissions individually at each one.

> **A REAL, DISTINCTIVE ORGANIZATIONAL PRINCIPLE**
>
> Google's own real documentation states this plainly: resources don't belong to the individual employee who created them — they belong to the Organization. This real design choice directly ensures continuity when personnel changes occur, since a Project's own real ownership was never actually tied to any one person's account in the first place.

Three Real Hierarchy Models, Compared

LEVEL	AWS	AZURE	GCP
Broadest	Organization (Chapter 2's scope, not covered in depth in this course)	Management Group	Organization
Intermediate grouping	Organizational Unit (OU)	(Management Groups can nest)	Folder (optional)
Billing/access boundary	Account	Subscription	Project

LEVEL	AWS	AZURE	GCP
Mandatory local grouping	None (tags-based)	Resource Group (every resource belongs to one)	Project (every resource belongs to exactly one)
Real Q1 2023 market share	~31%	~25%	~11%

A genuinely useful real observation: AWS Fundamentals never covered a formal multi-account hierarchy (that course scoped a single AWS account throughout); Azure's Resource Group and GCP's Project both play the same real role as the mandatory, per-resource grouping unit — just under different names, at a different real level of the overall hierarchy.

Hands-On Exercises

EXERCISE 1

A company grants a specific IAM role at the Organization level. Explain, in your own words, what real effect this has on a brand-new Project created inside a Folder underneath that Organization, without any additional role assignment being made.

View solution

EXERCISE 2

An employee who created several GCP Projects leaves the company. Explain, in your own words, why this real, documented organizational principle means those Projects and their resources are not automatically at risk of being lost or orphaned.

View solution

EXERCISE 3

A team with prior Azure experience assumes GCP's own regions will have the exact same "check per-region before assuming zone support" gotcha Azure has. Explain, in your own words, why this concern, while not completely baseless, is less critical in GCP's own real, more uniform model.

View solution

CHAPTER 1 QUICK REFERENCE

- **GCP's real origin** — App Engine, announced April 2008 — genuinely predates Azure's own October 2008 announcement / February 2010 launch
- **Real current scale** — 40 regions, 121 zones (Q1 2024); most regions have three zones
- **Real hierarchy** — Organization (root) → Folder (optional) → Project (every resource belongs to exactly one) → Resource
- Real IAM inheritance — a role granted at a higher level cascades automatically to everything beneath it
- Real principle: resources belong to the Organization, not the individual who created them
- **Real market position** — GCP ~11%, behind AWS's ~31% and Azure's ~25% (Synergy Research, Q1 2023)

CHAPTER 2: IAM & CLOUD IDENTITY

GCP Fundamentals

Chapter 2 · IAM & Cloud Identity

AWS Fundamentals Chapter 2 covered IAM's own user/group/role/policy structure; Azure Fundamentals Chapter 2 covered Entra ID's tenant-wide identity model and RBAC. GCP's own real IAM system shares the same underlying goal — controlling who can do what, on which resource — through a genuinely distinctive real vocabulary and mechanism.

Real Principals

GCP calls any identity being granted access a **principal**, split into two real categories:

- **Human users** — Google Accounts, Google groups, and federated identities in workforce identity pools.
- **Workloads** — service accounts (below), and federated identities in workload identity pools.

Real Three Role Types

ROLE TYPE	REAL DESCRIPTION
Basic (Primitive)	Owner, Editor, Viewer — real, highly permissive, broad access
Predefined	Google-managed collections of permissions for common real tasks (e.g. the Pub/Sub Publisher role)
Custom	User-created, containing only the specific real permissions you define — most control, most maintenance

A REAL, DIRECT WARNING FROM GOOGLE ITSELF

Google's own real, official documentation states plainly that basic roles "shouldn't be used in production environments." Owner and Editor grant sweeping, broad access across a Project —

genuinely the same real category of risk AWS's own root account and Azure's own Owner role represent, and exactly the kind of role least privilege exists to steer you away from by default.

Real IAM Policy Bindings

GCP's own real, distinctive mechanism is the **IAM policy binding** — a real, direct association between one role and a list of principals, attached to a specific resource. A resource's own "allow policy" contains a real list of these bindings; when a principal attempts an action, GCP checks that policy to confirm the principal holds a binding granting the required real permission.

```
# A real IAM policy binding, conceptually
{
  "role": "roles/storage.objectViewer",
  "members": [
    "user:alice@example.com",
    "serviceAccount:app-sa@my-project.iam.gserviceaccount.com"
  ]
}
```

Exactly as Chapter 1's own resource hierarchy promised, a binding applied at a Folder or Organization level cascades automatically to every Project and resource beneath it — the same real inheritance principle, applied here specifically to access control.

Service Accounts

A **service account** is a real identity for an application or workload, not a person — each with its own real, unique email address, commonly attached directly to a Compute Engine instance (Chapter 3) so that instance's own code can make authorized API calls.

THE SAME REAL LESSON, GCP'S OWN WORDS

Google's own real, official guidance is direct: "Service account keys are a security risk if not managed correctly." The recommended real alternatives — **attached service accounts** (automatically

obtaining short-lived credentials), **Workload Identity Federation** (for external workloads), and **domain-wide delegation** (for Google Workspace integration) — all avoid ever creating or storing a long-lived private key file at all. This is genuinely the same real lesson as AWS's own IAM-role-over-embedded-keys guidance (AWS Fundamentals Chapter 2) and Azure's own Managed Identity guidance (Azure Fundamentals Chapter 2) — three different real providers, the same underlying real principle.

AWS IAM vs. Azure RBAC vs. GCP IAM

PROPERTY	AWS	AZURE	GCP
Identity term	User / Role	Security principal	Principal
Broadest built-in role	Root account (avoid daily use)	Owner	Basic roles — Owner/Editor/Viewer (avoid in production)
Service identity mechanism	IAM role	Managed Identity	Service account (avoid long-lived keys)
Permission-grant mechanism	Policy attached to identity	Role assignment at a scope	Policy binding (role + principals) on a resource
Inheritance	None (flat, per-identity)	Downward through RBAC scope	Downward through resource hierarchy

Hands-On Exercises

EXERCISE 1

A developer is granted the Editor basic role on a Project so they can "do everything they need" quickly. Explain, in your own words, why Google's own real guidance recommends against this in production, and what real alternative they should use instead.

View solution

EXERCISE 2

A team downloads a service account key file and embeds it directly in their application's own source code, later accidentally committing it to a public repository. Explain, in your own words, what real, standard alternative would have avoided this specific risk entirely, using the chapter's own real guidance.

View solution

EXERCISE 3

A team binds a Viewer role to a specific principal at the Folder level. Explain, in your own words, what real access that principal has to a Project created later inside that same Folder, and why.

View solution

CHAPTER 2 QUICK REFERENCE

- **Principals** — human users (Google Accounts, groups) vs. workloads (service accounts, federated identities)
- **Three role types** — Basic/Primitive (Owner/Editor/Viewer, avoid in production), Predefined, Custom
- **IAM policy binding** — a role + a list of principals, attached to a resource; inherits down the Chapter 1 hierarchy
- **Service accounts** — real identities for workloads; Google's own real, official warning against long-lived keys
- Real alternatives to keys: attached service accounts, Workload Identity Federation, domain-wide delegation

CHAPTER 3: COMPUTE ENGINE

GCP Fundamentals

Chapter 3 · Compute Engine

AWS Fundamentals Chapter 3 covered EC2; Azure Fundamentals Chapter 3 covered Azure VMs' own real B-series credit model and IMDS story. Compute Engine, Google's own real virtual machine service, was announced 28 June 2012 at Google I/O in limited preview, reaching general availability by December 2013 — and it brings its own genuinely distinctive real pricing and security mechanisms.

Real Machine Families

FAMILY	REAL TUNED FOR
E2	Cost-effective, light workloads
N2 / N2D	General purpose, consistently high performance (Intel/AMD)
C2 / C3	Compute optimized — HPC, compute-bound workloads
M2 / M3	Memory optimized — large in-memory databases
A2 / A3	GPU-accelerated — ML training and inference (NVIDIA A100/H100)
T2A	ARM-based (Ampere Altra), cost-effective, scale-out

A Real, Genuinely Distinctive Feature: Automatic Sustained Use Discounts

ZERO ENROLLMENT, ZERO COMMITMENT

Unlike AWS's own Reserved Instances or Azure's own Reserved VM Instances — both requiring an explicit 1- or 3-year purchase decision — GCP's **Sustained Use Discount (SUD)** is real, entirely automatic: the more of a calendar month an eligible instance runs, the bigger the real discount, calculated and applied with zero enrollment or manual configuration at all. Real documented tiers:

0% for the first quarter of the month, rising in real steps to a maximum 30% net discount for resources that run the full month.

A REAL, EASY-TO-MISS ELIGIBILITY GAP

Not every machine family qualifies for the maximum real discount — and one real, genuinely important exception stands out: **E2**, the very family recommended above for cost-effective workloads, is *not eligible* for sustained use discounts at all. N1/M1/M2 real instances can reach a 30% discount; N2/N2D/C2 top out at 20%; E2 (and GPU/accelerator families) get none. E2's own lower baseline price doesn't stack with a real SUD the way an N2 instance's own price can.

Committed Use Discounts

Real Committed Use Discounts (CUDs) work more like AWS's or Azure's own reserved pricing — a real, upfront resource-based commitment for a defined term, in exchange for a larger discount than SUDs alone provide. CUDs and SUDs can be combined for the same instance.

Spot VMs

GCP's own real Spot VMs offer discounts of up to 91% off on-demand pricing — genuinely in the same real range as AWS Spot and Azure Spot VMs (both up to ~90%).

A REAL, MORE ABRUPT DEFAULT

GCP's own real default preemption notice is **0 seconds** — an instance can be reclaimed with no advance warning at all unless you explicitly configure up to 120 seconds of notice. This is a genuinely more abrupt real default than the warning period AWS and Azure Spot instances typically provide, and worth designing around explicitly for any workload that needs even a brief graceful-shutdown window.

A Real, Genuinely Different Firewall Model

NETWORK-LEVEL DEFINITION, PER-INSTANCE ENFORCEMENT

Neither AWS's per-instance security groups nor Azure's NSGs quite match GCP's own real model: firewall rules are defined at the **VPC network level**, not attached to any individual instance directly

— but enforcement still happens per-instance, via real targeting mechanisms: **network tags** (a label applied to matching instances) or **service accounts** (targeting instances using a specific identity). A rule with no target applies to every instance in the network by default.

GCP firewall rules are real and stateful — an allowed connection's own reply traffic is automatically permitted back, the same real behavior as AWS security groups and Azure NSGs. The platform enforces a real, implicit default-deny for inbound traffic, though GCP's own default network ships with pre-populated permissive rules (SSH, RDP, ICMP, internal traffic) that a production environment should review and tighten.

OS Login: IAM-Integrated SSH Access

Where AWS and Azure rely on a real SSH key pair generated at instance launch, GCP offers a genuinely more integrated real alternative: **OS Login** ties SSH access directly to IAM (Chapter 2) rather than a static credential. Real, critical difference: OS Login performs a permission check on *every single login attempt* — revoking access immediately the moment an IAM role is changed or removed, unlike a traditional metadata-based SSH key, which keeps working regardless of any later IAM change until someone manually removes it.

THE SAME REAL THEME, ONE LEVEL DEEPER

This is genuinely the same real "avoid static, long-lived credentials" principle Chapter 2's own service-account-key warning taught — applied here specifically to SSH access instead of API credentials.

EC2 vs. Azure VM vs. Compute Engine

PROPERTY	AWS EC2	AZURE VM	GCP COMPUTE ENGINE
Real launch	2006	June 2012	Announced June 2012, GA Dec 2013
Discount for steady use	Reserved Instances (explicit purchase)	Reserved VM Instances (explicit purchase)	Sustained Use Discounts — real, automatic, zero enrollment
Real Spot/preemptible discount	Up to ~90%	Up to 90%	Up to 91%

PROPERTY	AWS EC2	AZURE VM	GCP COMPUTE ENGINE
Firewall model	Per-instance security group + subnet NACL	One stateful NSG (subnet or NIC)	Network-level rules, per-instance enforcement via tags/service accounts
SSH access model	Static key pair	Static key pair (typically)	OS Login — real, per-login IAM permission check

Hands-On Exercises

EXERCISE 1

A team picks E2 machines for a workload specifically because they expect to also benefit from Sustained Use Discounts once the instances run continuously all month. Explain, in your own words, why this expectation is incorrect.

View solution

EXERCISE 2

A workload needs at least a few seconds to save state gracefully before shutting down, and the team is deploying it on GCP Spot VMs using the real default configuration. Explain, in your own words, why this default is a genuine risk here, and what real, specific change would fix it.

View solution

EXERCISE 3

An employee's IAM permissions are revoked immediately after they leave a team, but the team is unsure whether this genuinely blocks their SSH access to production instances right away. Explain, in your own words, why the answer depends on whether OS Login or traditional metadata-based SSH keys are being used.

View solution

CHAPTER 3 QUICK REFERENCE

- **Compute Engine** — announced June 2012, GA by December 2013
- **Machine families** — E2 (economical, no SUD), N2/N2D (general purpose), C2/C3 (compute), M2/M3 (memory), A2/A3 (GPU), T2A (ARM)
- **Sustained Use Discounts** — real, automatic, zero-enrollment, up to 30% (N1/M1/M2) or 20% (N2/N2D/C2); E2 not eligible
- **Spot VMs** — up to 91% off; real 0-second default eviction notice (configurable up to 120 seconds)
- Real firewall model: rules defined at the network level, enforced per-instance via network tags or service accounts — genuinely unlike AWS/Azure
- **OS Login** — real IAM-integrated SSH, checked on every login attempt; immediate revocation, unlike static SSH keys

CHAPTER 4: CLOUD STORAGE

GCP Fundamentals

Chapter 4 · Cloud Storage

AWS Fundamentals Chapter 4 covered S3's real, fixed 11-nines durability. Azure Fundamentals Chapter 4 covered Azure Blob Storage's own real choice of durability level, and a genuine 2023 SAS-token leak case study. GCP's own real Cloud Storage borrows the AWS-style "one fixed durability figure" approach — but delivers a genuinely distinctive real twist on how its coldest tier actually behaves.

Real Storage Classes

CLASS	REAL MIN. DURATION	REAL AVAILABILITY SLA	USE CASE
Standard	None	99.9%–99.95%	Frequently accessed ("hot") data
Nearline	30 days	99.0%–99.9%	Accessed roughly once a month or less
Coldline	90 days	99.0%–99.9%	Accessed at most once a quarter
Archive	365 days	99.0%–99.9%	Accessed less than once a year, incl. disaster recovery

A Real, Genuinely Distinctive Feature: Uniform Millisecond Retrieval

EVERY TIER RETRIEVES IN MILLISECONDS — EVEN ARCHIVE

Neither AWS Fundamentals Chapter 4's own Glacier tiers (real retrieval times measured in hours) nor Azure Fundamentals Chapter 4's own Archive tier (up to 15 real hours to rehydrate) match GCP's real behavior here: even GCP's coldest class, **Archive**, is real, genuinely available for retrieval "within milliseconds, not hours or days." Every one of GCP's four storage classes differs in cost, minimum storage duration, and retrieval fees — but *not* in real access speed, unlike the AWS and Azure models this course's own two sibling chapters covered.

Real, Uniform Durability

All four GCP storage classes share the same real, documented durability figure: 99.999999999% annual durability — 11 nines, identical to S3's own real figure, and applied uniformly across every class rather than varying by a real, chosen redundancy configuration the way Azure Blob Storage's own LRS/ZRS/GRS choice does.

Public Access Prevention

GCP's own real, direct equivalent of S3's Block Public Access is **Public Access Prevention**: enforced at the bucket level or, per Chapter 1's own real Organization hierarchy, across an entire Organization via policy constraint. Once enforced, any request relying on the real `allUsers` or `allAuthenticatedUsers` principals fails outright (HTTP 401/403) — existing public permissions are overridden, not deleted, meaning enforcement doesn't require hunting down and manually removing every individual public grant first.

Signed URLs

GCP's own real equivalent of AWS pre-signed URLs and Azure SAS tokens is the **Signed URL** — a real, time-limited credential embedded directly in a URL, granting temporary access to a specific resource without the requester needing standing IAM credentials at all.

> **A REAL, PLATFORM-ENFORCED SAFETY CEILING**
>
> Azure Fundamentals Chapter 4's own 2023 case study involved a SAS token real, effectively permanent expiration (valid until 2051) with no platform-level ceiling stopping it. GCP's own real Signed URLs cannot make that same mistake at the platform level: the maximum real expiration is capped at 604,800 seconds — exactly 7 days — enforced by GCP itself, not merely a best-practice recommendation. A real, common practical pattern uses a much shorter window still, often 15 minutes, for actual uploads and downloads.

S3 vs. Blob Storage vs. Cloud Storage

PROPERTY	AWS S3	AZURE BLOB STORAGE	GCP CLOUD STORAGE
Durability	Fixed, 11 nines	Chosen — 11/12/16 nines	Fixed, 11 nines (uniform across classes)
Coldest-tier retrieval	Hours (Glacier)	Up to 15 hours (Archive)	Milliseconds (Archive) — real, genuinely distinctive
Public-exposure guard	Block Public Access	Block Public Access	Public Access Prevention
Scoped temporary access	Pre-signed URLs	SAS tokens (no platform-enforced max)	Signed URLs, real 7-day platform-enforced max

Hands-On Exercises

EXERCISE 1

A team moves rarely-accessed compliance records to GCP's Archive class, then worries their disaster-recovery plan needs data restorable within minutes if ever required. Explain, in your own words, why GCP's own real Archive class behavior addresses this concern in a way AWS Glacier and Azure's own Archive tier genuinely couldn't.

View solution

EXERCISE 2

A developer tries to create a Signed URL intended to remain valid for 30 days, to avoid regenerating it repeatedly. Explain, in your own words, why this specific request cannot succeed on GCP, and what real, structural protection this provides compared to Azure's own SAS token model.

View solution

EXERCISE 3

A bucket has several existing public IAM grants when Public Access Prevention is enforced on it. Explain, in your own words, whether those existing grants need to be manually deleted first for the enforcement to actually take effect, and why.

View solution

CHAPTER 4 QUICK REFERENCE

- **Storage classes** — Standard (no min.), Nearline (30 days), Coldline (90 days), Archive (365 days)
- Real, genuinely distinctive: every class retrieves in milliseconds — even Archive — unlike AWS Glacier or Azure Archive
- **Durability** — real, fixed 11 nines across every class, applied uniformly
- **Public Access Prevention** — GCP's real Block-Public-Access equivalent, enforceable at bucket or Organization level
- **Signed URLs** — real, time-limited access; platform-enforced 7-day (604,800-second) maximum expiration, unlike Azure's own unbounded SAS tokens

CHAPTER 5: VPC NETWORKING

GCP Fundamentals

Chapter 5 · VPC Networking

AWS Fundamentals Chapter 5 built a VPC scoped to one Region, with subnets each bound to one AZ. Azure Fundamentals Chapter 5 loosened that one notch — a VNet's own subnets span every AZ within a Region. GCP goes a full step further still, and this chapter's own real structural difference is the biggest of the entire three-course arc.

A Real, Major Structural Leap: The VPC Itself Is Global

GENUINELY BEYOND WHAT AWS OR AZURE OFFER

A GCP VPC network — including its own routes and firewall rules — is a real, **global** resource, not tied to any particular region or zone at all. Only **subnets** are regional. This is a genuinely bigger structural difference than even Azure's own subnet-spans-all-AZs model: a single GCP VPC can span every real region on the planet at once, with regional subnets created inside it wherever actually needed — never multiple separate VPCs peered together just to cover more than one region, the way a real multi-region AWS or Azure architecture typically requires.

One real, practical consequence: firewall rules and routes, both attached to the VPC itself, automatically apply consistently everywhere that VPC has subnets — a single global ruleset, not one per region.

Auto Mode vs. Custom Mode

- **Auto mode** — GCP automatically creates one real subnet per region, using predefined ranges inside `10.128.0.0/9`, and automatically provisions a new subnet whenever Google Cloud adds a region.
- **Custom mode** — you create every subnet manually, choosing exactly which regions get one and what IP range each uses.

GOOGLE'S OWN REAL RECOMMENDATION

> Google's own real documentation recommends custom mode for production specifically — auto mode's automatic subnet creation as new regions launch can silently create IP ranges that conflict with your own deliberate network planning, without any explicit action on your part.

Global Routes

Because the VPC itself is global, its own real routes are too — a route created once applies across the entire network, in every region the VPC has a subnet in, rather than needing to be duplicated per region the way an AWS route table must be attached separately to each subnet.

Cloud NAT

Cloud NAT gives private instances real outbound-only internet access, without any instance needing its own external IP — the same real capability as an AWS or Azure NAT Gateway. Its own real configuration model is genuinely simple: you configure a NAT gateway on a **Cloud Router** (which supplies the real control plane), specifying which subnets it serves — no separate route table entry is required at all, matching Azure's own simpler no-route-table NAT Gateway attachment (Chapter 5 of Azure Fundamentals) rather than AWS's own explicit route table requirement.

Firewall Rules, Revisited

Chapter 3 already introduced GCP's own real firewall model: rules defined at the network level, enforced per-instance via network tags or service accounts. Now that this chapter has established the VPC's own real global scope, the full picture is clear: a single set of firewall rules, attached to one global VPC, governs traffic consistently across every region that VPC touches — genuinely different from both AWS (per-instance security groups plus per-subnet NACLs, each scoped to one Region's own VPC) and Azure (one stateful NSG type, but still scoped to one Region's own VNet).

Three Real Network Models, Compared

PROPERTY	AWS VPC	AZURE VNET	GCP VPC NETWORK
VPC/network scope	One Region	One Region	**Global** — every region at once
Subnet scope	One specific AZ	All AZs in the Region	One specific Region
Firewall scope	Per-VPC (one Region)	Per-VNet (one Region)	Per-VPC (global)
NAT attachment	Requires a route table entry	No route table entry needed	No route table entry needed (Cloud Router-based)
Multi-region networking	Multiple VPCs, peered/connected manually	Multiple VNets, peered manually	One VPC, naturally spanning regions

Hands-On Exercises

EXERCISE 1

A team with prior AWS and Azure experience creates three separate VPC networks in GCP - one for each of three regions they plan to deploy into - expecting this to be necessary the way it was for their own AWS/Azure architectures. Explain, in your own words, why this is genuinely unnecessary on GCP.

View solution

EXERCISE 2

A production team uses an Auto mode VPC network, and months later, Google adds a new real region their organization has no plans to ever deploy into. Explain, in your own words, what real, automatic consequence this has for their network, and why Custom mode would have avoided it.

View solution

EXERCISE 3

A team updates one firewall rule on their single GCP VPC network, which has subnets in four different regions. Explain, in your own words, whether they need to repeat that same update three more times

for the other regions, and why - contrasting this with how the equivalent update would work on AWS.

View solution

CHAPTER 5 QUICK REFERENCE

- Real, major structural leap: a GCP **VPC network is global** — only subnets are regional, genuinely unlike AWS's or Azure's own per-region network scope
- **Auto mode** (one subnet per region, automatic) vs. **Custom mode** (manual, Google's own recommendation for production)
- Real **global routes and firewall rules** — one ruleset applies consistently across every region the VPC touches
- **Cloud NAT** — configured on a Cloud Router, no route table entry needed, matching Azure's own simplicity over AWS's explicit requirement
- Full three-way firewall recap: AWS's split security-group/NACL model, Azure's single-but-regional NSG, GCP's single-and-global network-level rules

CHAPTER 6: CLOUD SQL & FIRESTORE

GCP Fundamentals

Chapter 6 · Cloud SQL & Firestore

AWS Fundamentals Chapter 6 covered RDS (managed relational) alongside DynamoDB (managed NoSQL). Azure Fundamentals Chapter 6 covered Azure SQL Database alongside Cosmos DB. This chapter closes the third leg of that same relational-plus-NoSQL pairing — **Cloud SQL** and **Firestore** — and delivers this three-course arc's own most striking consistency-model finding yet.

Cloud SQL: Managed Relational Databases

Cloud SQL is GCP's managed relational database service, supporting three real engines: **MySQL**, **PostgreSQL**, and **SQL Server** — the same three-engine spread as AWS RDS's own core offering, and directly comparable to Azure SQL Database's own single-engine (SQL Server-based) scope.

High Availability: The Regional Instance

A Cloud SQL instance configured for high availability is called a real **regional instance** — it runs a primary and a standby in two different zones within the same region, kept in sync via real **synchronous replication** to each zone's own persistent disk, so every committed write genuinely exists in both zones before the write is confirmed.

Failover itself is real and automatic: a heartbeat system checks the primary's health every second, and if several heartbeats are missed, failover begins. The standby takes over as the new primary using the instance's own shared static IP address, so a connected application reconnects with zero configuration change — real observed unavailability during failover is around 60 seconds. The old primary is then destroyed and rebuilt as the new standby.

A REAL, EASY-TO-MISS LIMITATION

The HA standby instance cannot serve read queries — it exists purely for failover. This is a genuinely important distinction from a **read replica**, a separate Cloud SQL feature that does serve real read traffic, matching the same HA-vs-read-replica split AWS RDS and Azure SQL Database both already draw in their own equivalent chapters.

Firestore: A Serverless NoSQL Document Database

Firestore is GCP's fully managed, serverless NoSQL **document database** — data is organized into collections of documents rather than tables of rows, the same real document model as AWS DynamoDB and Azure Cosmos DB's own Core (SQL) API.

A new Firestore database is created in one of two real modes: **Native mode**, the modern, recommended choice with real-time client sync and offline support, or **Datastore mode**, kept for compatibility with applications originally built on the older Cloud Datastore service — both modes share the same real underlying infrastructure.

Firestore scales automatically and horizontally, partitioning data across many servers and load-balancing requests with no manual capacity planning, and bills on a real pay-per-request model that can scale down to zero when unused.

A Real, Major Consistency Difference

> **FIRESTORE'S OWN REAL, SIMPLER GUARANTEE**
>
> Firestore provides real **strong consistency** for every read and write, always — every write is fully committed before the operation returns, and every subsequent read is guaranteed to see it. There is no eventually-consistent read option to choose, and none to accidentally get wrong.

This is a genuinely different, and genuinely simpler, real design than either sibling course's own NoSQL database. DynamoDB (AWS Fundamentals Chapter 6) makes the reader choose per-request — eventually consistent by default, with a real, roughly one-in-three documented chance of returning briefly stale data, or strongly consistent at double the read cost if explicitly requested. Cosmos DB (Azure Fundamentals Chapter 6) goes even further the other way, exposing a real five-level consistency spectrum the developer configures at the account or container level. Firestore skips that whole decision entirely: consistency is always strong, with nothing to configure and nothing to get wrong.

Three Managed NoSQL Databases, Compared

PROPERTY	AWS DYNAMODB	AZURE COSMOS DB	GCP FIRESTORE
Consistency model	Choose per-request (eventual default, strong optional)	Five-level spectrum, set at account/container level	**Always strong** — no choice to make
Real API surface	DynamoDB API	Six wire-compatible APIs	Native mode or Datastore-compatible mode
Scaling model	Provisioned or on-demand capacity	Provisioned or serverless throughput (RU/s)	Fully automatic, pay-per-request

Relational vs. NoSQL: When to Use Which

- **Cloud SQL (relational)** — structured, related data that benefits from real joins, transactions, and a fixed schema — the same use case AWS Fundamentals and Azure Fundamentals both already reach for RDS and Azure SQL Database.
- **Firestore (NoSQL)** — flexible, hierarchical, or rapidly-changing data, especially where real-time client sync or offline support matters, such as a mobile app's own live data feed.

Hands-On Exercises

EXERCISE 1

A team scales up read traffic on their Cloud SQL instance and routes some of that traffic to the HA standby instance to spread the load. Explain, in your own words, why this will not work, and what they should use instead.

View solution

EXERCISE 2

A developer moving from DynamoDB to Firestore looks for a way to explicitly request an eventually consistent read, expecting it to be cheaper than a strongly consistent one, the way it is on DynamoDB. Explain, in your own words, what they will find, and why.

View solution

EXERCISE 3

A team is migrating an existing application that was built directly on the older Cloud Datastore service, and does not want to rewrite its data-access code. Explain, in your own words, which Firestore mode lets them do this, and why.

View solution

CHAPTER 6 QUICK REFERENCE

- **Cloud SQL** — managed MySQL, PostgreSQL, or SQL Server; HA via a **regional instance** (primary + standby in different zones, synchronous replication, ~60s automatic failover)
- The HA standby **cannot serve reads** — use a separate read replica for that
- **Firestore** — serverless NoSQL document database; **Native mode** (modern) or **Datastore mode** (compatibility)
- Real, major finding: Firestore is **always strongly consistent** — genuinely simpler than DynamoDB's per-request choice or Cosmos DB's five-level spectrum
- Relational (Cloud SQL) for structured/related data; NoSQL (Firestore) for flexible, hierarchical, or real-time-synced data

CHAPTER 7: CLOUD FUNCTIONS & CLOUD RUN

GCP Fundamentals

Chapter 7 · Cloud Functions & Cloud Run

AWS Fundamentals Chapter 7 covered Lambda and its real, hard 15-minute execution ceiling. Azure Fundamentals Chapter 7 covered Azure Functions, whose higher hosting plans remove that ceiling almost entirely. GCP's own serverless story goes further than either — its two compute services aren't just similar, they're **architecturally the same thing**.

Cloud Run: A Fully Managed Container Platform

Cloud Run runs your code as a real container — no cluster to create, no infrastructure to manage. It supports deploying an existing container image directly, or a real source-based deployment (Go, Node.js, Python, Java, .NET, Ruby) where the platform builds the container for you automatically.

- **Scaling** — real, fast request-based autoscaling up to thousands of instances, with genuine **scale-to-zero**: if no requests arrive, even the last instance is removed. Minimum instances can be configured to avoid this where cold starts matter.
- **Timeout** — a real default of 5 minutes, extendable up to a real maximum of **60 minutes**.
- **Billing** — two real models: request-based (pay only while actively processing a request, nothing while idle) or instance-based (pay for an instance's full lifetime, no per-request fee).

A Real, Major Structural Finding: Cloud Functions 2nd Gen *Is* Cloud Run

> **NOT JUST SIMILAR — LITERALLY THE SAME UNDERLYING SERVICE**
>
> Google's own real documentation now calls 2nd-generation Cloud Functions **"Cloud Run functions"**. Deploying a 2nd-gen function doesn't run it on some separate functions-only runtime — it's "automatically built as a container and deployed as a Cloud Run service." Cloud Functions 2nd gen is a convenience layer for going from source code straight to a running Cloud Run service, not a genuinely separate compute product underneath.

This is a real, deeper architectural merge than either AWS (Lambda and Fargate are genuinely separate compute engines under the hood) or Azure (Azure Functions and Container Apps are related but distinct services) offer. On GCP, once you're on 2nd gen, "function" and "container service" describe the same real deployed artifact viewed two different ways.

1st Gen vs. 2nd Gen

PROPERTY	1ST GEN	2ND GEN (CLOUD RUN FUNCTIONS)
Max timeout	9 minutes	60 min (HTTP-triggered); 9 min (event-driven)
Max resources	8 GB RAM / 2 vCPU	16 GB RAM / 4 vCPU
Concurrency per instance	1 request at a time	Up to 1,000 concurrent requests
Event sources	7 direct sources	90+ sources via Cloud Audit Logs
Underlying infrastructure	Legacy functions runtime	Cloud Run itself

Three Serverless Compute Models, Compared

PROPERTY	AWS LAMBDA	AZURE FUNCTIONS	GCP CLOUD RUN / FUNCTIONS
Max execution time	Hard 15-minute ceiling, every plan	Unbounded on Flex Consumption/Premium/Dedicated	60 minutes (HTTP), any Cloud Run service
Relationship to containers	Lambda and Fargate are separate services	Functions and Container Apps are separate, related services	2nd-gen functions **are** Cloud Run services
Scale-to-zero	Yes (Lambda itself is inherently request-driven)	Yes, on Consumption plan	Yes, on Cloud Run's own default scaling

Hands-On Exercises

EXERCISE 1

A developer deploys a 2nd-generation Cloud Function and later wants to view or manage it directly through the Cloud Run console instead of the Cloud Functions console. Explain, in your own words, whether this is possible, and why.

View solution

EXERCISE 2

A team needs an event-driven function (not HTTP-triggered) to run for up to 20 minutes on 2nd-generation Cloud Functions. Explain, in your own words, whether this is possible, and what real limit stands in the way.

View solution

EXERCISE 3

A team migrating from AWS explains that on Lambda, moving past the 15-minute execution ceiling meant switching to an entirely different service (Fargate). Explain, in your own words, whether the equivalent move on GCP - going from a short function to a long-running container workload - requires switching services in the same way.

View solution

CHAPTER 7 QUICK REFERENCE

- **Cloud Run** — fully managed containers, real scale-to-zero, up to 60-minute timeout, request-based or instance-based billing
- Real, major finding: 2nd-gen Cloud Functions is literally **called "Cloud Run functions"** and deploys as a real Cloud Run service underneath
- 1st gen: 9-min max, 1 concurrent request, 8GB/2vCPU. 2nd gen: 60-min max (HTTP), 1,000 concurrent, 16GB/4vCPU
- Genuinely deeper merge than AWS (Lambda/Fargate separate) or Azure (Functions/Container Apps separate) offer

CHAPTER 8: LOAD BALANCING & MANAGED INSTANCE GROUPS

GCP Fundamentals

Chapter 8 · Load Balancing & Managed Instance Groups

AWS Fundamentals Chapter 8 covered ELB's own ALB/NLB split and Auto Scaling Groups. Azure Fundamentals Chapter 8 went further with a genuinely more specialized four-way load-balancing split, plus the honest warning that "a scale set alone can't protect you against data center failures." GCP's own load-balancing story picks up directly from Chapter 5's own real finding that a VPC network is global — because on GCP, the load balancer itself inherits that same global-by-default character.

Cloud Load Balancing: Global by Default

A REAL, DISTINCTIVE ARCHITECTURE

A GCP global load balancer presents **a single anycast IP address as the frontend for backend instances in regions around the world** — one IP, genuinely spanning multiple regions at once, with automatic cross-region failover built in. This is delivered through Google's own real edge network, **Google Front Ends**, present in over 80 distinct locations worldwide, and built on real internal Google infrastructure (Maglev, Andromeda, Envoy proxies) rather than instance-based load balancer software.

This is a genuinely different starting point from both siblings. AWS's ALB/NLB are inherently regional constructs — genuine global routing needs a separate service layered on top (Route 53 or CloudFront). Azure's own Load Balancer and Application Gateway are similarly regional, with Front Door as the separate global layer. On GCP, a global external load balancer is *already* multi-region, with no separate global-routing service required to get there.

Layer 4 vs. Layer 7, and Global vs. Regional

GCP splits load balancers along the same real Layer 4 / Layer 7 line as its siblings — **Network Load Balancers** (Layer 4, routing on TCP/UDP/other transport-layer data) and **Application Load Balancers** (Layer 7, routing on HTTP headers and URI paths) — then further splits each by scope: global external, regional external, or classic, with internal variants for private-only traffic.

TYPE	LAYER	REAL SCOPE OPTIONS
Application Load Balancer	7 (HTTP-aware)	Global external, regional external, classic, internal (regional/cross-region)
Network Load Balancer	4 (TCP/UDP)	Proxy (global/regional/classic/internal) or passthrough (global/regional/internal)

Managed Instance Groups (MIGs)

A Managed Instance Group keeps a fleet of identical VMs — built from one shared instance template — running as a single manageable entity, with automatic recreation of failed instances, integration with load balancers, and automated rolling updates.

- **Zonal MIG** — every instance lives in one real zone, supporting up to 1,000 VMs.
- **Regional MIG** — instances spread automatically across multiple zones in one region, supporting up to 2,000 VMs, giving real protection against a single zone's own failure.

> **THE SAME WARNING AZURE'S OWN SCALE SETS CARRY**
>
> A **zonal** MIG, by itself, gives no protection at all against a zone going down — every instance in it lives in that one zone. This is the exact same real gap Azure Fundamentals Chapter 8 already flagged for a Virtual Machine Scale Set on its own: high availability against a genuine zone/data-center failure has to be a deliberate choice (a **regional** MIG here), not an assumption that comes free with autoscaling.

Real Autoscaling Signal Types

- **CPU utilization** — scale based on average CPU across the group.
- **Load balancing capacity** — scale directly on the backend service's own real traffic load.
- **Cloud Monitoring metrics** — scale on any custom metric.
- **Schedule-based** — scale on a predetermined timetable.
- **Queue-based** (zonal MIGs only) — scale based on a workload queue, e.g. Pub/Sub backlog.

Two Real, Distinct Health-Check Mechanisms

GCP names this distinction explicitly, where it's often left implicit elsewhere: a real **load-balancing health check** exists to redirect traffic away from an instance that's failing, while a real, separate **autohealing health check** exists to proactively recreate that instance entirely. Same underlying idea as AWS's and Azure's own health-check-driven instance replacement, but GCP names and configures the two purposes as two distinct checks rather than one shared mechanism.

Hands-On Exercises

EXERCISE 1

A team wants one single IP address to front backend instances deployed across three different GCP regions, with automatic failover between regions if one goes down. Explain, in your own words, whether GCP's global load balancer can do this natively, and how that compares to what AWS's ALB alone would need.

View solution

EXERCISE 2

A team deploys a zonal Managed Instance Group and assumes it is already protected against a single zone going down, since it autoscales and autoheals. Explain, in your own words, why this assumption is wrong, and what real change would fix it.

View solution

EXERCISE 3

An instance in a Managed Instance Group is briefly slow to respond to the load balancer's own health check, but its own application-level autohealing check still passes. Explain, in your own words, what real, different action GCP takes as a result of each of these two checks.

View solution

CHAPTER 8 QUICK REFERENCE

- Real, major finding: GCP's **global load balancer** is one anycast IP spanning multiple regions at once, natively — no separate global-routing layer needed, unlike AWS (Route 53/CloudFront) or Azure (Front Door)
- Layer 4 (Network LB) vs. Layer 7 (Application LB), each further split by global/regional/classic scope
- **Zonal MIG** (single zone, up to 1,000 VMs) vs. **Regional MIG** (multi-zone, up to 2,000 VMs, real zone-failure protection)
- A zonal MIG alone offers no zone-failure protection — the same real gap Azure's own scale sets carry without explicit multi-zone configuration
- Two distinct, explicitly named health checks: **load-balancing** (redirects traffic) and **autohealing** (recreates the instance)

CHAPTER 9: CLOUD MONITORING & COST MANAGEMENT

GCP Fundamentals

Chapter 9 · Cloud Monitoring & Cost Management

AWS Fundamentals Chapter 9 covered CloudWatch's real memory-and-disk-space collection gap and Budgets' alert-only behavior. Azure Fundamentals Chapter 9 covered Azure Monitor's own richer four-layer model and Budgets' identically alert-only behavior, paired with Action Groups for real automated response. GCP's own story confirms the same recurring pattern a third time — and then genuinely goes one step further on the automation side.

Cloud Monitoring: Automatic Metrics vs. the Ops Agent

Cloud Monitoring automatically collects real system metrics for Compute Engine VMs — CPU utilization and disk usage, with no setup required. But **detailed system metrics, application metrics, third-party application metrics** (Apache, Nginx, MongoDB, PostgreSQL, and others), and any custom instrumentation all require installing the real **Ops Agent** on the VM first.

> **THE SAME REAL GAP, A THIRD TIME**
>
> This completes the exact same recurring finding across all three providers in this arc: memory and deeper metrics are never collected automatically by default — AWS CloudWatch needed the CloudWatch Agent, Azure Monitor needed the Azure Monitor Agent, and now GCP Cloud Monitoring needs the Ops Agent. Three genuinely independent platforms, the same real architectural choice each time.

Alerting Policies

An alerting policy triggers when a chosen metric crosses a defined threshold (e.g. CPU above 80%), routing notifications to email, Slack, PagerDuty, or other channels, and creates a real, persistent **incident** record for tracking and troubleshooting — a distinctly named concept GCP uses explicitly, where AWS and Azure both lean more on the underlying alarm/alert object itself. Alerting policies can be configured via the console, API, CLI, or Terraform, and many Google Cloud services come with pre-configured alerting recommendations already available.

Cloud Billing Budgets: Alerts, Not a Spending Cap

A standard Cloud Billing budget is **alerts-only** by default. Google's own real documentation states plainly that an alerts-only budget "doesn't automatically cap Google Cloud... usage or spending" — it exists purely to inform, triggering email notifications at real, configurable thresholds (the real default set is **50%, 90%, and 100%** of the budget) as actual or forecasted spend crosses them.

> **THE SAME REAL FINDING, A THIRD TIME**
>
> This is now a fully confirmed three-way pattern: AWS Budgets alerts but never stops spending; Azure Budgets' own documentation states resources "aren't affected" and consumption "isn't stopped"; GCP's own alerts-only budget explicitly does not cap usage either. No major provider's own default budget mechanism actually enforces a spending ceiling.

Budgets can also publish real **Pub/Sub notifications** at each threshold — a downstream Cloud Function (Chapter 7) can subscribe to that topic to automate a real response, such as programmatically disabling billing on a project, echoing Azure's own Action Group-attached-to-a-budget pattern, but built from GCP's own Pub/Sub-plus-function pieces rather than one native attachment point.

A Genuine Step Further: Spend Cap Budgets

> **MORE BUILT-IN THAN EITHER SIBLING NATIVELY OFFERS**
>
> GCP also offers a real, separate **spend cap** budget type (currently in preview) for supported services — one that can actually pause a service once its threshold is reached, rather than only notifying. Lifting the cap afterward still requires manual intervention, but this genuinely goes further out of the box than AWS's typical SNS-plus-Lambda assembly or Azure's Action Group attachment, both of which still require the team to build the automated response themselves.

Three Monitoring & Budget Models, Compared

PROPERTY	AWS	AZURE	GCP
Memory metrics by default	No — needs CloudWatch Agent	No — needs Azure Monitor Agent	No — needs Ops Agent
Default budget behavior	Alerts only	Alerts only	Alerts only
Built-in automated stop option	None native — build via SNS + Lambda	None native — attach an Action Group	**Spend cap budgets (preview)** can pause a service directly

Hands-On Exercises

EXERCISE 1

A team notices their Cloud Monitoring dashboard shows no memory-usage data for their Compute Engine VMs, despite CPU and disk usage showing up correctly. Explain, in your own words, why this is happening, and what real step fixes it.

View solution

EXERCISE 2

A team sets a standard, alerts-only Cloud Billing budget of $1,000/month, and is surprised when their actual spend reaches $1,500 with no service interruption. Explain, in your own words, why this happened, and what real alternative budget type would have actually stopped spending.

View solution

EXERCISE 3

A team wants their Cloud Billing budget to automatically disable billing on a project once spend hits 100%, without using GCP's own preview spend-cap feature. Explain, in your own words, what real combination of services they would need to build this themselves.

View solution

CHAPTER 9 QUICK REFERENCE

- **Cloud Monitoring** — automatic CPU/disk metrics; memory, detailed, and third-party metrics need the **Ops Agent** (completing a real three-way "memory needs an agent" pattern with AWS and Azure)
- Alerting policies create a named **incident** record and route to email/Slack/PagerDuty
- Cloud Billing budgets are **alerts-only by default** (50/90/100% thresholds) — confirmed the same real behavior as AWS Budgets and Azure Budgets, none of which stop spending by default
- Pub/Sub-triggered budget notifications can drive a real Cloud Function to programmatically respond
- Real, distinguishing extra: a preview **spend cap** budget type that can actually pause a service — more built-in than AWS's or Azure's own native options

CHAPTER 10: CAPSTONE — DEPLOYING A SIMPLE WEB APPLICATION ON GCP

GCP Fundamentals

Chapter 10 · Capstone: Deploying a Simple Web Application on GCP

AWS Fundamentals and Azure Fundamentals each closed with the same small photo-upload web application, built using each provider's own real architecture. This capstone rebuilds it a third time, on GCP — the same nine dependency-ordered steps, each one using GCP's own genuinely different mechanisms explored across this course's own nine prior chapters, closing the full three-course, 30-chapter Cloud Platform Specifics project.

Step 1: Organization, Folder & Project Planning (Ch. 1)

Before creating anything, we place the project inside GCP's own real resource hierarchy — an Organization, a Folder for this application, and a Project underneath it — and choose a Region with real Availability Zone support, keeping in mind Chapter 1's own caution that not every GCP region necessarily offers the same zone count.

Step 2: A Scoped IAM Role, No Long-Lived Keys (Ch. 2)

A dedicated service account is created for the application, granted a real **predefined role** scoped to exactly what it needs (never a basic role like Editor or Owner), and **OS Login** is enabled for SSH access instead of managing static SSH keys — avoiding the exact real risk category, long-lived service account keys, that Chapter 2's own case study covered.

Step 3: A Global VPC With One Regional Subnet & Cloud NAT (Ch. 5)

A Custom mode VPC network is created (per Google's own real production recommendation from Chapter 5) with one subnet in the chosen Region, and a Cloud NAT gateway attached via a Cloud Router for outbound-only internet access from private instances — no route table entry required.

Step 4: A Web Tier on a Regional Managed Instance Group (Ch. 3, Ch. 8)

Web servers run on Compute Engine, deployed via a **regional** Managed Instance Group (Chapter 8's own real zone-failure protection, not a zonal group alone), with a firewall rule scoped only to the necessary inbound ports and enforced via network tags.

Step 5: Cloud Storage for Uploaded Photos (Ch. 4)

Uploaded photos land in a Cloud Storage bucket with **object versioning** and **Public Access Prevention** enabled. Any temporary, shareable photo link uses a Signed URL — automatically capped at GCP's own real, platform-enforced 7-day maximum, the exact structural protection Chapter 4 contrasted against Azure's own unbounded SAS token model.

Step 6: Cloud SQL for Metadata (Ch. 6)

Photo metadata (uploader, caption, timestamp) is stored in a Cloud SQL **regional instance** — a PostgreSQL primary with an HA standby in a different zone of the same region — reachable only from the web tier's own subnet via a scoped firewall rule, never from the public internet.

Step 7: A Storage-Triggered Cloud Function for Thumbnails (Ch. 7)

A 2nd-generation Cloud Function — genuinely deployed as a real Cloud Run service under the hood, per Chapter 7's own central finding — triggers automatically on each new object landing in the Cloud Storage bucket, generating a thumbnail and writing it back to a separate bucket.

Step 8: A Global Load Balancer in Front of the Regional MIG (Ch. 8)

A global external Application Load Balancer fronts the web tier's regional Managed Instance Group behind one real anycast IP address — Chapter 8's own central finding, delivered here with no separate global-routing service layered on top the way an equivalent AWS or Azure build would need.

Step 9: Monitoring & an Honestly-Framed Budget (Ch. 9)

The Ops Agent is installed on every web tier instance so real memory metrics are actually collected (not just the automatic CPU/disk figures), an alerting policy watches CPU and error-rate thresholds, and a Cloud Billing budget is configured with the real default 50/90/100% alert thresholds — framed honestly per Chapter 9: this budget alerts, it does not by itself cap spending. A spend cap budget is noted as a genuine, GCP-specific option worth evaluating for this workload once it leaves preview.

Chapter Attribution: Every Step's Real Cross-Cloud Equivalent

STEP	GCP CHAPTER	AWS FUNDAMENTALS EQUIVALENT	AZURE FUNDAMENTALS EQUIVALENT
1. Org/Folder/Project & region planning	Ch. 1	Ch. 1 — Region/AZ planning	Ch. 1 — Region/AZ-support planning
2. Scoped IAM role, no static keys	Ch. 2	Ch. 2 — scoped IAM role	Ch. 2 — scoped Managed Identity
3. Global VPC, one regional subnet, Cloud NAT	Ch. 5	Ch. 5 — VPC, public/private subnets	Ch. 5 — single VNet spanning every AZ
4. Regional MIG web tier, scoped firewall	Ch. 3, Ch. 8	Ch. 3 — EC2, scoped security group	Ch. 3 — web VM, scoped NSG
5. Cloud Storage, versioning, capped Signed URLs	Ch. 4	Ch. 4 — S3, versioning, Block Public Access	Ch. 4 — Blob Storage, ZRS, scoped SAS tokens
6. Cloud SQL regional instance for metadata	Ch. 6	Ch. 6 — Multi-AZ RDS	Ch. 6 — General Purpose Azure SQL Database
7. Storage-triggered thumbnail function	Ch. 7	Ch. 7 — S3-triggered Lambda	Ch. 7 — Blob-triggered Azure Function
8. Global LB + regional MIG	Ch. 8	Ch. 8 — ALB + Auto Scaling Group	Ch. 8 — Application Gateway + Flexible-mode VMSS
9. Ops Agent, alerts, honest budget	Ch. 9	Ch. 9 — CloudWatch Agent, honest Budget	Ch. 9 — Azure Monitor Agent, Budget + Action Group

What's Still Out of Scope

- No CI/CD pipeline for deploying application code
- No CDN or WAF layer (Cloud CDN, Cloud Armor)
- No DNS layer (Cloud DNS)
- No multi-Region disaster recovery
- No containerization of the web tier itself (beyond what Cloud Run implicitly provides for the thumbnail function)

THE SAME DISCIPLINE, THREE GENUINELY DIFFERENT TOOLKITS

Across all three courses, the same underlying security and architecture discipline held: least privilege (IAM roles / IAM RBAC / Managed Identities), private-by-default networking (scoped firewall rules / security groups / NSGs, with GCP's own global VPC and load balancer going further than either sibling), and honest monitoring and budgeting (alerts that inform, not silently enforce, unless deliberately wired to). The real lesson of this three-course arc isn't that one provider is "better" — it's that the same real principles get expressed through three genuinely different, verifiable mechanisms, and knowing which is which is what actually transfers between them.

Hands-On Exercises

EXERCISE 1

A reviewer asks why the web tier uses a regional Managed Instance Group instead of a zonal one for this capstone. Explain, in your own words, the real risk a zonal MIG alone would leave unaddressed.

View solution

EXERCISE 2

A teammate familiar with the AWS Fundamentals capstone asks why this GCP build doesn't need a separate service like Route 53 or CloudFront to get one global entry point across regions. Explain, in your own words, why not.

View solution

EXERCISE 3

A stakeholder assumes that once the Cloud Billing budget in Step 9 is configured, GCP will automatically stop the application's own services from running if spend exceeds the budget. Explain, in your own words, whether this assumption is correct, and what real GCP feature would actually be needed to make it true.

View solution

CHAPTER 10 QUICK REFERENCE — COURSE COMPLETE

- Nine dependency-ordered steps, each using GCP's own real architecture: resource hierarchy → IAM → global VPC/Cloud NAT → regional MIG → Cloud Storage → Cloud SQL → Cloud Function → global load balancer → monitoring/budget
- Chapter-attribution table names each step's real AWS Fundamentals and Azure Fundamentals equivalent
- Honest scope boundary: no CI/CD, no CDN/WAF, no DNS layer, no multi-Region DR, no full containerization
- **GCP Fundamentals is now complete — 10/10 chapters.** This closes the full three-course Cloud Platform Specifics project: AWS Fundamentals, Azure Fundamentals, and GCP Fundamentals, 30 chapters total.