Organization & Identity

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

SECURE ENGINEERING FOUNDATIONS

- **Hierarchy** — Boundaries
- **Identity** — Least privilege
- **Secrets** — Controlled access
- **Workstations** — Consistent tools

01 Organization, identity and development foundations

Resource hierarchy and isolation

- **Organizations, folders and projects** define policy and access boundaries. Group production and development projects under appropriate folders so inherited controls apply consistently to new projects.
- **Separate environments** keep experiments, load tests and feature development away from production. Use separate projects and clusters where stronger isolation is required; namespaces alone do not isolate network traffic.
- **Shared VPC** centralizes network administration while service projects own workloads. A project lien helps prevent deletion; the Shared VPC lien-removal organization policy restricts who can remove that protection.
- **VPC Network Peering** connects private networks with non-overlapping ranges and is non-transitive. **Private Service Connect** exposes selected services privately. Choose connectivity boundaries around ownership and access requirements.

Least privilege and workload identity

- **Service accounts** represent workloads. Attach a dedicated identity to a VM or build, grant only required resource permissions, and use short-lived credentials through Application Default Credentials.
- **Workload Identity Federation for GKE** gives each Kubernetes service account its own cloud access. Grant permissions to the workload principal or configure Google service account impersonation; avoid sharing node credentials among applications.
- **IAM and Kubernetes RBAC** control different operations. Cloud IAM grants Google Cloud access; namespace-scoped Roles and RoleBindings authorize Kubernetes actions. Add network policies and workload security controls for tenant isolation.
- **Organization policies** can restrict service account key creation and upload. Prevent new keys, inventory and retire existing keys, and prefer federation. If an unavoidable exception is approved, scope it narrowly and audit it.
- **Groups and narrow resource scopes** simplify access reviews. Assess IAM changes before applying them, review effective inheritance, and keep build execution permissions separate from human approval permissions.

Organization & Identity / SRE & Incidents

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

SECURE ENGINEERING FOUNDATIONS

- **Hierarchy** — Boundaries
- **Identity** — Least privilege
- **Secrets** — Controlled access
- **Workstations** — Consistent tools

Secrets, keys and rotation

- **Secret Manager** stores versioned passwords and API keys. Grant the runtime or build identity **Secret Accessor** on required secrets; keep secret values out of source, images, command output and build logs.
- **Cloud KMS** manages cryptographic keys and cryptographic operations. An encrypted secret still needs protected ciphertext, decrypt permission and rotation handling. **Base64** is an encoding; **HMAC** authenticates data and does not encrypt it.
- **Cloud Run secret environment variables** resolve at instance startup. Pin a version for repeatable releases. A mounted secret using **latest** can expose rotated values when read; the application must reread the file and refresh connections.
- **Rotation** must update the external credential, publish the secret version, refresh consumers and retire the old credential safely. A rotation schedule or new secret version alone does not change a database password.
- **Secret placement** must meet resource-location policy. Use user-managed replication to select allowed locations when automatic replication conflicts with residency requirements.

Managed development environments

- **Cloud Workstations** standardizes IDEs, tools and development images. A private workstation cluster supports private access; grant users workstation access rather than broad project administration.
- **Idle timeout** stops unused workstations; running timeout caps a session even if active. Use authenticated IDE web previews for local web apps and provision supported GPUs with sufficient quota when development needs them.

02 Site reliability engineering and incident management

SLIs, SLOs and user experience

- **SLI** measures service behavior; **SLO** sets its target over a defined window; **SLA** is a contractual commitment with consequences. Agree on user journeys and business needs before setting targets.
- **Availability SLI = successful eligible requests / all eligible requests.** Define which requests count and what success means. Raw success counts change with traffic and do not express reliability by themselves.
- **Latency SLI = requests below a threshold / eligible requests.** For 95% below 200 ms, count qualifying requests against the full eligible population. Averages can hide slow users and long tails.

SRE & Incidents

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

RELIABILITY FEEDBACK LOOP

Measure SLIs and SLOs → **Decide** Error budget → **Respond** Mitigate impact → **Learn** Improve systems

- **Quality** measures complete, non-degraded responses; **freshness** measures sufficiently recent data. Measure important features separately so a healthy homepage cannot hide broken checkout or reporting.
- **Request-based SLOs** evaluate event populations. **Window-based SLOs** evaluate the fraction of time intervals that meet a condition. Choose rolling or calendar periods deliberately and set achievable targets with useful operating margin.
- **Measurement location** determines coverage. Server or proxy logs reveal observed requests; client telemetry and synthetic journeys can also detect failures before traffic reaches the backend, including CDN or load-balancer problems.

Error budgets and release decisions

- **Allowed bad fraction = 1 - SLO target.** A 99.9% target permits 0.1% bad eligible events. A time-based 30-day window at 99.9% permits 43.2 minutes; request-based budgets are measured in requests.
- **Burn rate = observed bad-event fraction / allowed bad fraction.** A burn rate of 1 consumes the budget at the sustainable rate; higher rates consume it faster. Use multiple alert windows to detect both sharp and sustained degradation.
- **Error budget policy** is agreed with product, development and operations before an incident. Define release freezes, reliability priorities and exceptions. Exhaustion calls for following that policy, even when a feature passes unit tests.
- **Excess reliability** has opportunity cost. When targets and user satisfaction are sound and budget remains, invest in delivery or other reliability gaps instead of automatically tightening the SLO. Revisit objectives when user needs change.

Shared ownership and toil

- **SRE engagement starts during design.** Review architecture, dependencies and operational risks with developers early. Shared SLOs and error budgets align feature delivery with production reliability.
- **Production readiness reviews** assess monitoring, capacity, dependencies, failure handling, runbooks and ownership. Track required reliability improvements before transferring operational responsibility.
- **Toil** is manual, repetitive, automatable work with little enduring value that grows with service scale. Measure it, prioritize automation by impact, and keep engineering time for durable improvements.
- **Actionable paging** demands an urgent human response to user impact or imminent budget loss. Remove duplicate and self-resolving noise, route nonurgent work to tickets, and maintain sustainable on-call rotations.

SRE & Incidents / CI/CD & Supply Chain

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

RELIABILITY FEEDBACK LOOP

Measure SLIs and SLOs → **Decide** Error budget → **Respond** Mitigate impact → **Learn** Improve systems

Incident roles and mitigation

- **Declare an incident early** and establish a shared response channel and live state document. The **Incident Commander** coordinates decisions; the **Operations Lead** directs mitigation; the **Communications Lead** manages updates.
- **Mitigate user impact first.** Roll back a harmful release, disable a feature, fail over or reroute traffic when safe. Preserve useful evidence, then investigate contributing causes after service is stable.
- **Communicate intent before production changes.** Check remaining capacity before draining a failing node or region, coordinate responders, and avoid independent changes that conflict or worsen overload.
- **Status updates** state impact, known facts, mitigation progress and the next update time. Update internal and external stakeholders regularly without inventing a resolution estimate; hand over command explicitly.
- **Runbooks and exercises** prepare responders for common failures. Track detection and recovery times separately: **MTTD** measures detection delay, **MTTR** recovery, **MTBF** failure intervals; include affected user share when estimating risk.

Blameless learning and follow-through

- **Postmortem policy** defines triggers such as significant user impact, data loss, unusually difficult recovery or a stakeholder request. Invite incident participants and leadership to support a learning culture.
- **A useful postmortem** records impact, timeline, detection, mitigation, contributing causes, what worked and what failed. Explain system conditions and decisions using information available at the time.
- **Action items** need one accountable owner, collaborators, priority and a completion target. Improve detection, tests, capacity or architecture; track completion and share lessons widely while protecting sensitive information.

03 Continuous integration, delivery and supply chain security

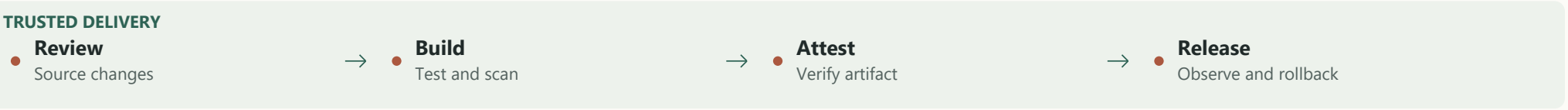
Source control and fast feedback

- **Trunk-based development** uses small changes and short-lived branches. Enforce peer review and automated checks through protected branches; keep source, build configuration and infrastructure definitions versioned.
- **CI sequence:** run unit checks, build an immutable artifact, test it in an isolated environment, then promote the same artifact. Integration and acceptance tests validate dependencies and configuration; post-deployment smoke tests confirm basic operation.

CI/CD & Supply Chain

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET



- **Triggers** distinguish pull requests, branch pushes and release tags. Match production builds to the intended branch or anchored semantic-version tag pattern; broad tag patterns can also match development tags.
- **Approval gates** serve distinct purposes. Repository approval governs code merging; a Cloud Build approval governs build execution; a delivery approval governs promotion. Grant approver roles to the required group.
- **Ephemeral test namespaces** isolate feature work on a shared development cluster. Apply RBAC, quotas and network policies, test appropriate dependency versions, and clean up resources when tests or branches end.

Cloud Build execution and efficiency

- **Cloud Build** runs containerized build steps with a selected service account. Grant registry write, target deployment and state-bucket access only where needed; obtain GKE credentials and authorize Kubernetes actions for the deployment identity.
- **Private pools** provide managed build workers with private network connectivity. Configure VPC connectivity, DNS, routes and firewall rules for private APIs or database migrations; IAM alone does not establish a network path.
- **Step dependencies** use IDs and **waitFor**. Independent steps can start with **waitFor: ['-']**; a later step waits for their IDs. Parallelism shortens the critical path when CPU, I/O and quotas permit it.
- **Caching** avoids repeated downloads and builds. Reuse dependency or image-layer caches; copy stable manifests before frequently changed source in a Dockerfile. Compare larger workers against elapsed-time savings and total cost.
- **Build secrets** use **availableSecrets.secretManager** with narrowly scoped access for the actual build identity. Expose secrets only to required steps and avoid echoing them. Use build events and Pub/Sub for external integrations.

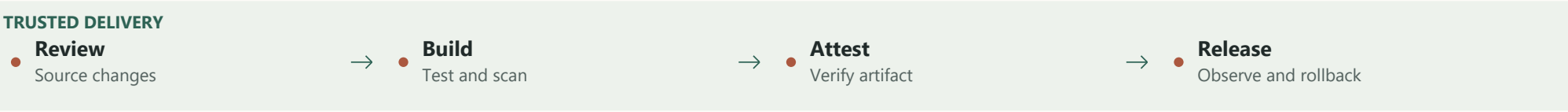
Terraform and immutable infrastructure

- **Terraform configuration** declares desired resources; **state** maps those resources to real objects. Use a persistent **gcs backend** with locking, controlled bucket access and object versioning so ephemeral runners share the correct state.
- **Plan before apply**, review changes, and serialize updates to each state. Separate environment configurations and state, reuse tested modules, and run infrastructure integration tests in disposable projects before merging.
- **State is sensitive.** Protect it with IAM and encryption rather than committing it to Git. A backend 403 points to the runner identity or bucket access; state loss can make existing infrastructure appear unmanaged.

CI/CD & Supply Chain

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET



- **Instance templates are immutable.** Create a replacement template, update the managed instance group, roll instances safely and retire the old template. Terraform **create_before_destroy** requires names that allow old and new resources to coexist.
- **Packer** creates repeatable machine images, including across cloud providers. Version images and templates, deploy through controlled rollouts, and use reusable foundation modules for repeatable project provisioning.

GitOps and environment configuration

- **GitOps** keeps reviewed desired configuration in Git and uses reconciliation to converge live resources. Separate application source from deployment configuration when ownership and access differ; use explicit environment directories or overlays.
- **Config Sync** reconciles Kubernetes configuration from a repository. **Policy Controller** evaluates constraints and constraint templates. **Config Connector** reconciles supported Google Cloud resources through Kubernetes custom resources.
- **Kustomize** overlays patch environment differences onto shared bases. **Skaffold profiles** select environment-specific rendering and deployment settings for Cloud Deploy, keeping shared configuration consistent.
- **Drift management** requires a defined controller and source of truth. Coordinate tools so Terraform and a Kubernetes controller do not independently own the same cloud resource.

Progressive releases and rollback

- **Rolling updates** replace replicas incrementally with configured surge and unavailability limits. **Blue/green** keeps two environments and switches routing, allowing fast reversal while the previous environment remains ready.
- **Canary releases** expose a small share of real traffic, observe user-facing metrics and advance gradually. Compare with a representative baseline under similar load and cache warmup; a fresh baseline avoids misleading cold-start comparisons.
- **A/B tests** compare behavior across cohorts. **Shadow tests** copy production traffic to a candidate without returning candidate responses; isolate writes and other side effects. Load tests and shadow tests answer different questions.
- **Cloud Run revisions** support tagged test URLs and deployment with **--no-traffic**. Test the tagged revision, then adjust traffic percentages; restore routing to a healthy revision for rapid mitigation. Tags do not bypass service authentication.
- **GKE blue/green rollback** switches the Service selector to the healthy Deployment. A single Deployment rollout rollback changes that Deployment as a whole; use separate rollout units when independent zone or cohort control is required.

CI/CD & Supply Chain / Observability

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

TRUSTED DELIVERY

Review Source changes → **Build** Test and scan → **Attest** Verify artifact → **Release** Observe and rollback

- **Cloud Deploy** models pipelines, targets, releases and rollouts. Configure verification and approval gates; a **repairRollout** automation can retry failures and roll back to the latest successful release. An ordinary rolling update does not automatically undo itself.
- **API and database compatibility** must survive mixed versions and rollback. Use additive schema changes before removing old fields, migrate carefully, publish replacement APIs and deprecation timelines, and monitor remaining consumers before retirement.

Artifact trust and deployment enforcement

- **Artifact Registry** stores container images and OCI Helm charts with IAM controls. Place repositories near consumers where practical. **Remote repositories** cache upstream dependencies; **virtual repositories** present a unified read endpoint.
- **Image tags** are convenient release labels and may move. **SHA-256 digests** identify immutable image content. Record the source commit and build metadata, then deploy and roll back using the intended digest.
- **Artifact Analysis** detects known vulnerabilities in supported image content. Use on-demand scanning for a pre-push gate and automatic scanning for registry images; define severity policy and handle updated findings.
- **Binary Authorization** enforces deployment policy, such as approved registry patterns and required attestations. Scanning reports findings; admission policy decides whether the image may run.
- **Attestations** bind an image digest to a verified condition, such as trusted build provenance or successful load tests. Let an authorized pipeline sign only after checks pass; protect signing keys with KMS and use short-lived workload identity.
- **Provenance** links an artifact to its source and build process. Audit registry uploads to identify the acting principal. Approval, provenance, vulnerability checks and runtime detection protect different stages of the supply chain.
- **Emergency deployment exceptions** should use the documented, narrowly scoped Binary Authorization break-glass mechanism, with approval, audit review and cleanup. Keep normal admission enforcement active for other workloads.

04 Monitoring, logging, tracing and alerting

Telemetry and monitoring scope

- **Metrics** summarize numerical behavior, **logs** capture discrete events, **traces** connect work across services, and **profiles** attribute resource consumption to code. Correlate them using time, resource labels and request context.
- **Four golden signals** are latency, traffic, errors and saturation. Measure user-visible symptoms for SLOs and pages; internal CPU, queue, disk and dependency metrics help explain causes and predict capacity pressure.
- **A metrics scope** has a scoping project and monitored projects. Use dedicated scopes for relevant production or organizational boundaries, add projects explicitly, and grant **Monitoring Viewer** at the intended scope.

Observability

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

CONNECT THE EVIDENCE

- **Metrics** — Detect symptoms
- **Logs** — Inspect events
- **Traces** — Follow requests
- **Profiles** — Find code cost

- **Dashboard sharing** requires both a URL and authorized access to its data. Export dashboard JSON when another team needs a reusable definition. A copied dashboard does not grant access to the original metrics.
- **OpenTelemetry** instruments metrics and traces across environments. Use supported collectors and exporters for central analysis; control metric-label cardinality to prevent excessive time series and cost.

VM, container and Prometheus collection

- **Ops Agent** collects supported VM logs and host metrics. Fleet agent policies can automate installation and updates. Grant the attached identity **Logs Writer** and **Monitoring Metric Writer**, then check access scopes and API reachability.
- **Missing VM telemetry** requires checking the agent process, configuration, file permissions, application output, service identity, egress and ingestion errors. Native platform metrics do not imply guest logs are being collected.
- **GKE workload logs** normally come from container stdout and stderr. For an unmodifiable app that logs to a file, a sidecar can tail a shared volume into stdout. A node collector cannot automatically read every private container filesystem.
- **Managed Service for Prometheus** provides managed metric ingestion and PromQL querying across clusters. Managed collection uses **PodMonitoring** or **ClusterPodMonitoring** resources; verify selectors, named ports and scrape paths when metrics are absent.
- **System and workload logging** can be selected separately in GKE. Retain operational logs while reducing verbose development application collection where those application logs are not needed centrally.

Metric design and log-derived signals

- **Gauge** represents a current value. **Delta** represents change over an interval; **cumulative** represents a running accumulation. Metric kind and value type are separate choices and must match the measurement.
- **Distribution metrics** retain observations in buckets for latency histograms and heatmaps. Choose useful bucket boundaries; averages or a single percentile cannot describe every user experience.
- **Logs-based counter metrics** count matching events such as cache misses or blocked requests. **Distribution metrics** extract numerical values such as duration. Filter using actual structured fields and choose bounded labels.
- **Alignment** transforms samples within a series; **reduction** combines series across resources. Keep numerator and denominator filters consistent when computing ratios, and retain labels needed to isolate a service or region.

Observability

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

CONNECT THE EVIDENCE

- **Metrics** — Detect symptoms
- **Logs** — Inspect events
- **Traces** — Follow requests
- **Profiles** — Find code cost

Log routing, retention and analysis

- **Structured JSON logs** expose searchable fields such as severity, service, request and trace identifiers. Cloud Run and GKE collect supported JSON written to stdout; Logging client libraries can send structured entries directly.
- **Log Router sinks** filter and route matching incoming entries. Use aggregated sinks with child-resource inclusion at the organization or folder to cover existing and future projects; configure the sink writer identity on the destination.
- **Logging buckets** support operational querying and configured retention. **Cloud Storage** suits long-term archives; **BigQuery** supports SQL analysis; **Pub/Sub** streams to SIEMs and other consumers. A sink does not backfill historical logs.
- **Archive protection** combines appropriate storage class, retention policy and, when required, Bucket Lock. A locked retention policy is irreversible; configure and validate it carefully before locking.
- **Retention and transport are separate.** Pub/Sub is not a substitute for a multi-year archive. A sink routes copies; exclusions on one sink do not remove copies in other destinations or erase already stored records.
- **Cost control** starts with required operational and security evidence. Reduce unnecessary verbosity, sample suitable repetitive events, choose sink filters and retention deliberately, and preserve critical logs. Shorter retention does not reduce ingestion volume.

Log access, audit logs and sensitive fields

- **Logs Viewer** reads standard logs; **Private Logs Viewer** additionally covers Data Access audit logs. Use groups and **log views** to grant teams access only to their own entries in centralized buckets.
- **Restricted fields** can hide supported sensitive fields while retaining useful event data. Grant **Log Field Accessor** only to authorized readers. Field access restriction differs from removing PII before ingestion.
- **PII minimization** starts in the application or collection pipeline. Redact or omit sensitive values before sending logs. Sampling, encryption at rest and export sinks do not themselves sanitize an ingested record.
- **Audit log categories** include Admin Activity, Data Access, System Event and Policy Denied. **_Required** retains mandated audit categories; Data Access logs generally use **_Default** or custom routing. Enable Data Access logging where needed.
- **Audit configurations inherit** down the hierarchy. A child cannot simply disable an inherited Data Access logging requirement; enable it at selected folders or projects when an excluded subtree is required. Principal exemptions have a different scope.
- **Sink management** requires configuration permissions, such as those in **Logs Configuration Writer**; viewing logs does not grant permission to create exports. Review destination permissions and sensitive-data exposure independently.

Observability

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

CONNECT THE EVIDENCE

- **Metrics** — Detect symptoms
- **Logs** — Inspect events
- **Traces** — Follow requests
- **Profiles** — Find code cost

Diagnosing slow requests and code

- **Cloud Trace** follows a request across service spans. Propagate trace context through HTTP and messaging boundaries, inspect waterfalls for slow dependencies, and add spans around important business operations.
- **Trace-log correlation** requires attaching the trace identifier and span context to structured logs. Filter Logs Explorer by the slow request trace, or open correlated logs from trace details when the context is populated.
- **Sampling and export** balance diagnostic detail with overhead. Target an affected API proxy when possible. Background telemetry exporters on Cloud Run need CPU availability or a request-safe flush strategy to avoid losing buffered data.
- **Cloud Profiler** samples supported production runtimes with low overhead. CPU profiles reveal expensive computation; wall-clock time also includes waiting. Large wall/CPU gaps suggest blocking, I/O or dependency waits.
- **Heap growth and OOMKilled** suggest memory pressure or a leak. Compare profiles, limits, garbage collection and restart history. More memory can provide temporary headroom while the underlying defect is fixed.
- **Error Reporting** groups supported error events and stack traces. Database timeouts need connection counts, pool limits, query latency and server load as well as application release history; stable user counts do not imply stable database demand.

Alerts, synthetic checks and network evidence

- **Alert policies** combine measured conditions, duration, missing-data handling and notification channels. Use SLO burn-rate alerts for budget risk, tune thresholds to actionability, and test the complete delivery path.
- **Uptime checks** test endpoint reachability. **Private uptime checks** cover supported internal targets. **Synthetic monitors** exercise multi-step user journeys such as checkout and verify function beyond a single healthy response.
- **Private notification endpoints** require reachable transport. A VPC-local subscriber can pull Pub/Sub notifications and invoke an internal webhook; a managed intermediary with VPC access is another option. Public push delivery needs an accessible endpoint.
- **Connectivity Tests** analyze supported paths, routes and firewall configuration without shell access to workloads. **VPC Flow Logs** provide sampled flow metadata; even maximal secondary sampling does not capture every packet.
- **Firewall Rules Logging** records connections associated with logged rules. **Packet Mirroring** captures packet copies for deep analysis. Load-balancer and **Cloud Armor** logs reveal HTTP requests and policy decisions; inspect actual action and matched-field data.
- **Personalized Service Health** helps correlate relevant Google Cloud incidents with service symptoms. Treat provider events as diagnostic context alongside your own telemetry.

Capacity & Cost

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

OPTIMIZE WITH EVIDENCE

- **Measure** — Load and limits
- → **Plan** — Failure headroom
- → **Scale** — Pods and capacity
- → **Review** — Cost and objectives

05 Reliability, capacity, performance and cost optimization

Capacity and failure headroom

- **Load testing** establishes throughput, latency, bottlenecks and saturation under realistic traffic. Test expected peaks and dependency limits in an isolated environment; autoscaling does not replace measurement.
- **Compound growth** multiplies demand: 10% monthly growth for six months is about **1.77 times** the baseline. With three evenly loaded zones, surviving one-zone loss at full demand requires enough capacity in the remaining two.
- **Quotas, pool limits and warmup** constrain scaling. Validate regional CPU, GPU, address and service quotas, maximum nodes and instances, startup time and dependency capacity before a launch. Quota approval does not reserve physical capacity.
- **Resilience** combines replicas across failure domains, healthy routing and adequate remaining capacity. Use multiple regional deployments for regional failure; a regional managed instance group or regional GKE cluster stays within one region.

GKE scaling and resource pressure

- **Horizontal Pod Autoscaler (HPA)** adjusts replica count using resource or supported custom/external metrics. **Cluster autoscaler** adjusts node capacity for schedulable Pods. Enable and tune both for demand-driven scaling in Standard clusters.
- **Resource requests** influence scheduling and CPU-utilization targets. **Limits** bound consumption; exceeding memory limits can cause OOM termination. **Vertical Pod Autoscaler** recommends or adjusts requests and may require Pod recreation.
- **Queue-driven scaling** can use backlog and oldest unprocessed message age. Increase consumers only if the downstream system can absorb more work; apply backpressure or admission control to protect scarce inventory or dependencies.
- **DiskPressure** often involves ephemeral storage, container logs or emptyDir volumes. Compare per-container ephemeral-storage usage to identify consumers; set requests, limits, cleanup and storage placement appropriately.
- **Cloud Run scaling** depends on concurrency, CPU, resource settings and instance bounds. Set minimum instances for latency needs and maximum instances to protect cost and dependencies; ensure quotas and database connections support the range.

Capacity & Cost

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

OPTIMIZE WITH EVIDENCE

Measure Load and limits → **Plan** Failure headroom → **Scale** Pods and capacity → **Review** Cost and objectives

Health checks and safe containment

- **Readiness probes** decide whether a Pod receives traffic. **Liveness probes** restart an unhealthy container. **Startup probes** allow slow initialization before liveness checks begin. Avoid restarts caused solely by a failing external dependency.
- **Managed instance group autohealing** recreates unhealthy VM instances using health checks. Load-balancer health checks control routing; they do not by themselves recreate failed VMs.
- **Cordon** blocks new scheduling; **drain** evicts eligible Pods and respects disruption rules. Neither isolates network traffic. For suspected compromise, coordinate security response, restrict communications, preserve evidence and replace affected resources safely.
- **GKE release channels** balance update maturity and feature access. Choose a suitably stable production channel and validate upcoming versions in staging with representative workloads; schedule controlled maintenance and monitor upgrades.

Bottlenecks and graceful degradation

- **Optimize the measured bottleneck.** CPU saturation suggests computation; high I/O wait suggests storage or blocking; peak VM network throughput suggests machine/network limits. Adding NICs does not automatically multiply a VM bandwidth limit.
- **Timeouts, bounded retries and jitter** prevent indefinite waits and synchronized retry storms. Set deadlines within the caller latency budget; retry only safe or idempotent work.
- **Circuit breakers and bulkheads** stop repeatedly calling unhealthy dependencies and contain resource exhaustion. Cache, queue or omit optional work where safe so a slow auxiliary service cannot consume the entire request budget.
- **Warm capacity** reduces cold-start and queueing delays. Tune minimum instances, worker pools or idle capacity from measurements. Extra replicas do not automatically fix a per-process memory leak or a saturated shared database.

Capacity & Cost

Google Cloud Professional Cloud DevOps Engineer

KNOWLEDGE CHEATSHEET

OPTIMIZE WITH EVIDENCE

Measure Load and limits → **Plan** Failure headroom → **Scale** Pods and capacity → **Review** Cost and objectives

Cost attribution and economic tradeoffs

- **Billing export to BigQuery** supports detailed cost attribution. Apply consistent project and resource labels for services or teams, then query supported billing label fields. Resource metadata and names are not substitutes for billing dimensions.
- **Budgets and alerts** notify on actual or forecast spending; they do not automatically cap usage. Scope the budget to supported billing dimensions and use governed project or label selection for departmental totals.
- **Committed use discounts** suit predictable baseline usage. Resource-based commitments and flexible spend-based commitments have different eligible-resource and location rules; choose terms around expected changes. Discounts and capacity reservations solve different needs.
- **Spot VMs** suit interruption-tolerant, restartable work with checkpoints and durable outputs. Use regular capacity for workloads that cannot tolerate eviction or must meet strict completion guarantees.
- **Rightsizing** uses representative CPU, memory, network and workload metrics plus recommendations. Revalidate peak and failure scenarios before downsizing; a recommendation is input to a decision rather than an automatic production change.
- **Storage and network cost** depend on access frequency, retention, retrieval and data transfer. Use lifecycle policies for suitable archives, keep artifacts near consumers and compare supported Standard versus Premium network tiers against latency and resilience needs.
- **Reliability investment** balances avoided user and business loss against engineering and infrastructure cost. Estimate impact, failure probability and recovery time explicitly; an extra nine has value only in the context of the service.