Architecture & Planning

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

FROM REQUIREMENTS TO ARCHITECTURE

- **Goals** — Business outcomes
- → **Constraints** — Measurable needs
- → **Choices** — Service trade-offs
- → **Validate** — KPIs and tests

01 Designing and planning a cloud solution architecture 25%

Requirements and architectural trade-offs

- **Functional requirements** describe behavior. **Nonfunctional requirements** specify latency, availability, throughput, security and recoverability. Make each important requirement measurable before selecting products.
- **Business KPIs** connect technology to outcomes: adoption, conversion, dealer wait time or vehicle downtime. Track regional traffic, error rates and latency together when evaluating expansion.
- **Well-Architected Framework** balances operational excellence, security, reliability, performance optimization, cost optimization and sustainability. Record trade-offs and ownership in architecture decisions.
- **Managed services** reduce infrastructure work but still require access control, data design, monitoring and recovery planning. Compare total operational effort, portability and application compatibility.
- **Case-study reading** starts with the business goal, existing constraints and explicit technical requirements. Separate the immediate migration deadline from the long-term modernization strategy.

Choose the execution platform

- **Compute Engine** provides OS and kernel control for custom Linux, legacy software and specialized VMs. **Managed instance groups (MIGs)** add repeatable replicas, autoscaling and autohealing.
- **GKE** provides managed Kubernetes for container orchestration. **Autopilot** reduces node administration; **Standard** offers more infrastructure control. Kubernetes still requires sound workload configuration.
- **Cloud Run** hosts managed containerized services with request-driven scaling and revision traffic splitting. Scale-to-zero can reduce idle compute cost; consider cold starts, concurrency and downstream limits.
- **App Engine** supplies managed application hosting with services, versions and traffic splitting. Standard and flexible environments have different runtime, networking and scaling characteristics.
- **Cloud Run functions** suits event-triggered code. Use a managed scheduler or event source to invoke work; choose a VM or container job for execution requirements outside a function runtime.
- **Stateless frontends** keep durable sessions and business data outside individual instances. WebSockets need connection and reconnect planning; session affinity does not replace durable shared state.

Architecture & Planning

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

FROM REQUIREMENTS TO ARCHITECTURE

Goals — Business outcomes → **Constraints** — Measurable needs → **Choices** — Service trade-offs → **Validate** — KPIs and tests

Match the database to the access pattern

- **Cloud SQL** provides managed relational databases for compatible MySQL, PostgreSQL and SQL Server applications. **AlloyDB** serves PostgreSQL-compatible workloads needing its performance and availability architecture.
- **Spanner** combines relational transactions, strong consistency and horizontal scaling. Choose regional or multi-region placement according to availability, write latency and cost requirements.
- **Bigtable** serves high-throughput, low-latency key and range access to wide-column data, including telemetry. Design around row-key access; it is not a general relational join engine.
- **Bigtable row keys** should distribute writes while supporting prefix/range scans. Put frequently filtered identifiers before a time suffix when appropriate; a leading sequential timestamp can create hotspots.
- **Firestore** stores application documents such as profiles and game state. Design queries and indexes together; Datastore mode uses entities and keys. Batch key lookups avoid separate requests for known identifiers.
- **Composite indexes** support combinations of filters and ordering in Firestore/Datastore queries. Deploy required index definitions and allow index creation to complete before relying on the query.
- **BigQuery** is an analytical warehouse for large SQL scans and reporting. Separate analytical workloads from latency-sensitive transactional serving.
- **Memorystore** caches frequently read data and session information. Set expiration and invalidation policies, and retain authoritative data in durable storage when cache loss is unacceptable.

Build the data path

- **Pub/Sub -> Dataflow -> BigQuery** separates event ingestion, transformation and SQL analytics. Keep raw objects in **Cloud Storage** when replay and schema evolution matter.
- **Dataflow** runs Apache Beam batch and streaming pipelines. Event-time windows, watermarks and late-data handling accommodate delayed mobile or vehicle events.
- **Dataproc** runs Spark and Hadoop workloads with less platform migration effort. Use transient processing capacity and durable object storage to separate compute lifetime from data lifetime.
- **Datastream** captures supported database changes for downstream use. **Application Integration** connects SaaS and business systems; **Workflows** orchestrates service calls; each solves a different integration problem.

Architecture & Planning / Infrastructure & Data

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

FROM REQUIREMENTS TO ARCHITECTURE

Goals — Business outcomes → **Constraints** — Measurable needs → **Choices** — Service trade-offs → **Validate** — KPIs and tests

- **Data quality** checks validate schema, nulls, ranges, uniqueness and business rules before publishing curated data. Quarantine invalid records and retain enough context to repair or replay them.
- **Data locality** reduces latency and transfer cost. Preprocess regional datasets near their storage and move smaller results when central analysis does not require every raw record.

AI, ML and generative applications

- **Vertex AI** supports model development, training, deployment and MLOps. **Model Garden** helps select models; choose managed model APIs or deployable models according to control, cost and residency needs.
- **Pretrained APIs** can extract text, speech, image or video features without training a custom model. Generative models support summarization and conversation; evaluate them against task-specific examples.
- **Retrieval-augmented generation (RAG)** supplies relevant authorized content at inference time. Enforce document access during retrieval and evaluate grounding, citations and answer quality.
- **Training-serving skew** compares serving feature distributions with training data. **Drift** tracks changes over time. Monitor model quality and collect recommendation outcomes to guide retraining.
- **Explainable AI** helps inspect feature contributions for supported models; explanations do not prove correctness. Version datasets, features, model artifacts and evaluations for reproducibility.
- **Regional processing** requires checking the complete path: model availability, endpoint, storage, logs and external calls. A regional bucket alone does not constrain where inference executes.
- **Responsible AI** combines harmful-content screening, sensitive-data controls, evaluation and human review. Treat prompts and retrieved content as untrusted input; limit the actions an agent can perform.
- **Edge inference** supports disconnected vehicles and low-latency decisions. Plan signed model delivery, local fallback and later telemetry synchronization; accelerator choice does not itself improve model accuracy.

02 Managing and provisioning a cloud solution infrastructure 17.5%

VPC topology and connectivity boundaries

- **VPC networks** are global; **subnets** are regional. Allocate non-overlapping primary, secondary, hybrid and peering ranges, including room for GKE Pods and Services.
- **Shared VPC** centralizes networking in a host project while service projects own workloads. Separate network administration from workload and data permissions. Host and service projects belong to the same organization.
- **VPC Network Peering** connects eligible networks across projects or organizations through internal IPs. Peering is non-transitive and requires compatible, non-overlapping subnet ranges.

Infrastructure & Data

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

PROVISION THE COMPLETE DATA PATH

- **Network** — Private reachability
- **Compute** — Right runtime
- **Storage** — Access semantics
- **Data** — Ingest and analyze

- **Hub-and-spoke peering** does not automatically connect the spokes to one another. Plan firewall rules, advertised routes and access to shared services explicitly.
- **Overlapping networks** need renumbering or a carefully scoped translation/routing design. A firewall rule cannot resolve address ambiguity; VPN connectivity alone does not fix overlapping destinations.
- **Multiple NICs** can attach a supported VM to distinct VPC networks. Configure guest routing and firewalls deliberately; the VM does not automatically become a router between them.

Hybrid connectivity and routing

- **HA VPN** provides IPsec tunnels over the internet. Design redundant tunnels, peer devices and BGP sessions; a single peer remains a failure domain even with multiple tunnels.
- **Dedicated Interconnect** provides dedicated private connectivity; **Partner Interconnect** uses a service provider. Choose by throughput, location, deployment time, availability and recurring cost.
- **Cloud Router** exchanges dynamic routes with BGP for supported hybrid connections. It is a control-plane component, not a packet-forwarding appliance.
- **Interconnect redundancy** spans independent connections, edge availability domains and metros according to the availability target. Test failure paths and verify remaining bandwidth during an outage.
- **Private transport** and **encryption** are separate requirements. Plan TLS or a supported encrypted connectivity design where confidentiality in transit is required.
- **Direct Peering** reaches Google services; it is not a substitute for routed private connectivity into a VPC. Choose the connection based on the actual destination and address requirements.

Private access and traffic delivery

- **Cloud NAT** gives eligible private VMs and nodes outbound internet access without individual external IPs. It does not provide unsolicited inbound SSH access.
- **Private Google Access** lets eligible clients without external IPs reach Google APIs. **Private Service Connect** exposes supported services through private endpoints; plan DNS and service-specific access.
- **Private services access** allocates connectivity for supported producer-hosted services. It differs from Private Google Access and from Private Service Connect.
- **Serverless VPC access** or supported **Direct VPC egress** connects serverless workloads to private destinations. Verify routes, DNS, database permissions and hybrid return paths.
- **Application Load Balancers** provide HTTP(S) host/path routing and URL maps. Route API versions to separate backends while retaining a common DNS name and TLS certificate.

Infrastructure & Data

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

PROVISION THE COMPLETE DATA PATH

- **Network** — Private reachability
- **Compute** — Right runtime
- **Storage** — Access semantics
- **Data** — Ingest and analyze

- **Network Load Balancers** serve suitable TCP/UDP workloads. Match protocol, scope, proxy behavior and source-IP needs before choosing a load balancer; regional game traffic need not be HTTP.
- **Cloud CDN** caches eligible HTTP content near users; **Media CDN** addresses media delivery workloads. Set cache headers and keys deliberately, and avoid caching private responses across users.

Object, block and file storage

- **Cloud Storage** stores objects, **Persistent Disk** supplies VM block storage, and **Filestore** provides shared NFS file storage. Select by application semantics as well as capacity and throughput.
- **Cloud Storage FUSE** exposes objects through a filesystem interface for suitable workloads such as ML datasets. It does not provide every POSIX behavior or replace a shared transactional filesystem.
- **Standard, Nearline, Coldline and Archive** trade capacity cost against access charges and minimum storage duration. All provide online object access; estimate retrieval and operation costs as well as stored bytes.
- **Lifecycle management** transitions or deletes objects when rules match. **Retention policies** prevent premature deletion; locking retention is a separate, consequential action.
- **Object Versioning** retains noncurrent versions for recovery. **Soft delete** provides a separate recovery window. Coordinate lifecycle rules so old versions and protected copies do not grow without limit.
- **Regional Persistent Disk** replicates across two zones in one region. An active-passive application still needs failover orchestration and fencing to prevent simultaneous writers.
- **Disk performance** depends on disk type, provisioned capacity or performance settings, VM limits and access patterns. After enlarging a disk, expand the partition/filesystem as needed; Local SSD is scratch storage.

Provision resilient compute

- **Custom images** preinstall stable dependencies and reduce VM startup time. **Instance templates** define repeatable VM configuration; a MIG uses templates to create replicas.
- **Autoscaling** changes capacity; **autohealing** recreates unhealthy instances. Load-balancer health checks control traffic eligibility. Allow health-check traffic and give applications enough initialization time.
- **Regional MIGs** distribute instances across zones. Retain enough healthy-zone capacity and quota for the intended failure scenario; a regional MIG does not span multiple regions.
- **GKE Deployments** manage replicas and rolling updates. **Services** provide stable discovery and load balancing; **Ingress/Gateway** handles suitable external HTTP(S) routing.
- **StatefulSets** provide stable Pod identities and ordered lifecycle behavior. Persistent volumes provide storage; neither feature alone implements application replication or database failover.

Infrastructure & Data / Security & Compliance

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

LAYERS OF PROTECTION

- **Identity** — Least privilege
- **Network** — Enforce boundaries
- **Data** — Encrypt and minimize
- **Evidence** — Audit and retain

- **HPA** changes Pod count; **cluster autoscaler** adds or removes nodes based on scheduling needs. Use queue backlog for queue-bound work and requests-based resource planning for efficient scheduling.
- **Requests** guide scheduling; **limits** cap resource use. CPU limits can throttle and memory limits can cause termination. Tune from representative measurements; isolate special workloads with node pools, affinity and taints.
- **VM Manager** uses the OS Config agent for fleet patching. Rebuild container images after base-image updates. **Sole-tenant nodes** and node affinity support physical host placement and eligible licensing requirements.

03 Designing for security and compliance 17.5%

Identity, hierarchy and least privilege

- **Organization -> folders -> projects -> resources** provides policy and administration scopes. Group by business unit and environment; use separate projects where production and development permissions must diverge.
- **IAM allow policies** inherit downward and grants are additive. A narrower role at a child does not cancel an ancestor grant. Remove overbroad grants or apply supported deny controls deliberately.
- **Google Groups** simplify employee access management and preserve individual authentication. Prefer predefined roles at the narrowest useful scope; use custom roles for a real permissions gap.
- **Organization Policy** constrains supported configurations such as resource locations or external IP use. **IAM** controls who may act. Restrict project creation to the intended development or sandbox scope.
- **Cloud Identity** manages identities; **Directory Sync** provisions directory objects. **SAML federation** delegates sign-in to an identity provider; provisioning and authentication are distinct processes.
- **IAM Conditions** can constrain supported access by time or other attributes. Temporary auditor access should use individual identities, a managed group and a scoped read role.

Workload credentials and remote access

- **Attached service accounts** and Application Default Credentials let Google Cloud workloads obtain short-lived credentials. Grant only the target API permissions needed by that workload.
- **Compute Engine access scopes** can further restrict OAuth access despite IAM grants. For API failures, check the attached identity, IAM, access scopes, enabled APIs and network reachability.
- **Workload Identity Federation** exchanges trusted external workload identity for short-lived access. **Workload Identity Federation for GKE** avoids distributing service account keys to Pods.
- **Workforce Identity Federation** serves human users from external identity providers. **Service account impersonation** lets authorized principals act through a service account without a downloaded private key.

Security & Compliance

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

LAYERS OF PROTECTION

- **Identity** — Least privilege
- **Network** — Enforce boundaries
- **Data** — Encrypt and minimize
- **Evidence** — Audit and retain

- **Authenticated service invocation** uses a dedicated caller identity, the target invoker role and an ID token with the correct audience. A shared secret in environment variables is harder to govern and rotate.
- **IAP TCP forwarding** provides identity-controlled access to private VMs with the required firewall and IAM permissions. **OS Login** governs supported Linux login; Cloud Shell is a client environment, not a network bypass.

Network controls and exfiltration protection

- **VPC firewall rules** control traffic independently of subnet membership. Target tiers with supported service accounts or tags, and allow only the required upstream identities, protocols and ports.
- **Firewall priority** uses lower numbers first. Plan explicit egress restrictions when needed; the implied egress rule allows outbound traffic. Routes choose paths, while firewalls permit or deny traffic.
- **Hierarchical firewall policies** apply centrally at supported hierarchy scopes. Use governed targets and rule ordering so project teams cannot weaken required production controls.
- **Cloud Armor** applies edge protection and WAF policies to supported load-balancer backends. **VPC Service Controls** limits supported service access across perimeters to reduce data exfiltration; it complements IAM.
- **Access levels** can incorporate trusted source networks or other context for perimeter access. VPC firewall rules do not directly protect a Cloud Storage API request made from an employee laptop.
- **Cloud Run ingress** can require internal traffic or load balancing. With an Application Load Balancer and serverless NEG, restrict direct public service access so requests follow the intended WAF path.
- **Private GKE nodes** reduce exposure, and restricted control-plane access reduces the administration attack surface. Private deployment runners still need network reachability, DNS and authorization.

Encryption, secrets and data in use

- **Default encryption at rest** protects stored data. **CMEK** adds customer control through Cloud KMS permissions, rotation and auditability; it does not replace data-access controls.
- **Cloud HSM** provides HSM-backed Cloud KMS keys. **Imported key material** originates outside Google Cloud; **Cloud EKM** keeps key control in an external key manager. Match the actual key-custody requirement.
- **CSEK** requires supplying key material with supported service operations. It has different operational responsibilities from a Cloud KMS-managed key; service support varies.
- **Key rotation** creates or selects new key versions for future use. Existing ciphertext may still depend on old versions; plan re-encryption and recovery before disabling or destroying keys.
- **Secret Manager** stores application secrets with IAM and versioning. Keep credentials out of source code, container images and logs. Separate key administration, secret administration and data access.

Security & Compliance / Migration & Optimization

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

LAYERS OF PROTECTION

- **Identity** — Least privilege
- **Network** — Enforce boundaries
- **Data** — Encrypt and minimize
- **Evidence** — Audit and retain

- **Confidential Computing** protects supported workloads while data is in use. Encryption at rest and TLS alone do not protect plaintext during processing; verify workload and hardware compatibility.
- **Digital signatures** support origin and integrity verification. TLS protects a connection. Tamper-evident logs also need protected storage, controlled signing keys and a verification process.

Privacy, retention and audit evidence

- **Shared responsibility** requires mapping applicable obligations to the application, data flows, identities and operational processes. Provider certifications do not automatically make an application compliant.
- **Sensitive Data Protection** identifies and de-identifies sensitive content. Redact or tokenize before durable ingestion when raw PII must never be stored; include temporary files, retries and logs in that design.
- **Tokenization** substitutes a token for sensitive values and can reduce exposure. Deterministic transformations support matching but reveal equality; scope access to the token vault and plan rotation.
- **Deletion workflows** need subject identifiers, data inventory and propagation across stores and derived copies. Hiding rows in a view does not delete underlying personal data.
- **Immutable evidence** uses an appropriate retention policy and access separation. Version history, lifecycle deletion and encryption keys serve different purposes and do not independently enforce retention.
- **Compliance Reports Manager** supplies relevant provider reports. Confirm covered services and contractual requirements with compliance owners, and retain your own configuration, access and operational evidence.

04 Analyzing and optimizing technical and business processes 15%

Plan a controlled migration

- **Assess -> plan -> deploy -> optimize** separates discovery from execution and improvement. Use **Migration Center** for inventory, dependency assessment and cost modeling where appropriate.
- **Dependency mapping** groups tightly coupled components into migration waves. Start with a low-risk, representative workload; validate latency, security and functionality before expanding.
- **Rehost, replatform, refactor, retain or retire** are workload decisions. A rapid data-center exit may favor rehosting now and phased modernization later.

Migration & Optimization

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

MOVE SAFELY AND IMPROVE

Assess Dependencies and TCO → **Plan** Waves and recovery → **Validate** Load and failures → **Optimize** Cost and skills

- **Hybrid transition** requires addressing, DNS, identity, routing, firewall and bandwidth plans. Co-locate latency-sensitive dependencies where feasible, and define rollback criteria before cutover.
- **License assessment** checks software eligibility, OS support and required tenancy before import. VMware Engine preserves suitable VMware environments; Compute Engine provides a different operating model.

Cost, capacity and team readiness

- **TCO** includes service charges, licenses, network transfer, operations, recovery and engineering effort. Facilitate stakeholder decisions with measured requirements and explicit trade-offs.
- **Billing export to BigQuery** supports cost analysis by project and labels. Standardize ownership labels and budgets; budget notifications do not by themselves stop spending.
- **Rightsizing** uses representative utilization and performance tests. Schedule development VMs off outside work hours; stopped VMs can still incur storage and other retained-resource charges.
- **Spot VMs** suit interruption-tolerant batch work with checkpoints, retries and durable output. Keep reliable capacity for workloads that cannot tolerate eviction.
- **Committed use discounts** fit predictable usage commitments; **reservations** address capacity availability. Shared reservations can make eligible capacity available to approved consumer projects.
- **Skills readiness** maps upcoming responsibilities to training, labs and support needs. Use governed sandboxes with limited APIs, quotas, permissions and automatic expiry to support safe experimentation.

Availability, backup and regional recovery

- **RTO** is the target time to restore service; **RPO** is the tolerated recovery-point gap. Choose recovery architecture and exercises against both targets.
- **Zonal resilience** uses multiple zones and adequate spare capacity. **Regional resilience** needs another region, reachable dependencies, replicated or recoverable data and a tested traffic transition.
- **Cloud SQL HA** handles supported zonal failures within a region. An asynchronous cross-region read replica supports a different recovery path; promotion, connection changes and failback require planning.
- **Backups and point-in-time recovery** protect historical states. Replication can propagate corruption or deletion, so replicas do not replace backups. Monitor replica lag against the RPO.

Migration & Optimization / Delivery & Integration

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

MOVE SAFELY AND IMPROVE

- **Assess** — Dependencies and TCO
- → **Plan** — Waves and recovery
- → **Validate** — Load and failures
- → **Optimize** — Cost and skills

- **Backup and DR** can centralize supported workload protection and enforced backup-vault retention. Separate backup access from production administration and test restores into the intended recovery environment.
- **Cold, warm and hot recovery** trade ongoing cost for recovery speed. Provisioning automation helps restore infrastructure, but cannot recreate data that was never copied or backed up.

Test the system and the recovery plan

- **Unit tests** isolate code behavior; **integration tests** check component interactions; **acceptance tests** validate business behavior; **smoke tests** check essential behavior after deployment.
- **Load tests** reproduce representative concurrency, arrival rate, data volume and user paths. Stress beyond expected demand and measure latency percentiles, error rates and all dependent services.
- **Resilience tests** combine realistic load with controlled instance, zone, network or database failures. Verify capacity recovery, replica promotion and measured RTO/RPO in addition to traffic routing.
- **Fault injection** introduces delays or errors to test timeouts, retries and isolation. Define a blast radius, monitoring and stop conditions; service-mesh fault injection and node failure test different behaviors.
- **Firebase Test Lab** exercises mobile applications across device configurations. Endpoint coverage alone does not replace unit, load, failure or client-device testing.

05 Managing implementation 12.5%

Repeatable infrastructure and configuration

- **Terraform** declares infrastructure and produces a reviewable plan. Use reusable modules, controlled state access and a deployment identity that developers cannot bypass in production.
- **Config Connector** represents supported Google Cloud resources as Kubernetes objects. **Config Controller** provides a managed environment for declarative resource management.
- **Config Sync** reconciles configuration from a source repository. **Policy Controller** enforces admission constraints using policy rules; synchronization and policy validation solve different needs.
- **Service Catalog** can offer approved infrastructure patterns for self-service use. Encode networking, IAM, observability and environment separation in the supplied templates.
- **GitOps and IaC** improve repeatability and reviewability. Plan provider/API lifecycle changes, drift, rollback and team skills rather than assuming automation eliminates operational risk.

Delivery & Integration

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

DELIVER A VERIFIED ARTIFACT

Source Review change → **Build** Test and attest → **Promote** Controlled rollout → **Verify** Observe and recover

Build, attest and promote one artifact

- **Cloud Build** triggers builds and tests from source changes. **Artifact Registry** stores versioned images and packages. Promote the tested artifact instead of rebuilding different binaries for production.
- **Commit identifiers and image digests** link source to deployed artifacts. Mutable tags such as latest are insufficient deployment evidence; retain build provenance and release records.
- **Artifact Analysis** identifies supported vulnerabilities and metadata. An **SBOM** inventories dependencies; build provenance records how an artifact was produced. Enforce security gates in the pipeline.
- **Binary Authorization** checks deployment policy and trusted attestations. Scanning discovers findings; authorization enforces whether an artifact may deploy. Shielded nodes do not replace image validation.
- **Cloud Deploy** manages delivery to supported targets and promotion stages. Private GKE endpoints can require a reachable Cloud Build private pool configured as the execution environment.
- **Keyless CI/CD** uses an attached build identity, federation or impersonation. Separate plan, approval and apply permissions, and restrict direct production deployment to the intended delivery mechanism.
- **Container build caching** benefits from installing stable dependencies before copying frequently changed source. Use a minimal compatible base image and rebuild for security updates; do not remove runtime dependencies.

API contracts and asynchronous work

- **API versioning** preserves existing clients during incompatible changes. A stable facade can hide backend modernization; publish contracts and a deliberate deprecation process.
- **Apigee** provides API lifecycle and policy management. **Quota** controls allowed calls over an interval; burst protection, authentication and authorization address separate requirements.
- **OAuth 2.0** supports delegated authorization; **OpenID Connect** adds identity assertions. API keys identify calling applications but do not establish end-user identity.
- **Pub/Sub** decouples producers and consumers. Independent subscriptions give consumers their own backlog. Acknowledge after successful processing, and plan redelivery, dead letters and idempotency.
- **Publish batching** trades throughput efficiency against per-message delay. Tune batch thresholds for latency-sensitive ingestion, and distinguish publishing delay from subscriber backlog.
- **Ordering** is scoped to supported ordering-key configurations, not a global timestamp sequence. Exactly-once delivery features do not automatically guarantee exactly-once external business effects.

Delivery & Integration

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

DELIVER A VERIFIED ARTIFACT

Source Review change → **Build** Test and attest → **Promote** Controlled rollout → **Verify** Observe and recover

- **Cloud Scheduler** starts recurring work; **Cloud Tasks** manages individually scheduled/retried task delivery. Event-driven deployment notifications avoid assuming a release always occurs at a fixed time.
- **Timeouts, bounded retries and jittered exponential backoff** handle transient failures such as throttling and server errors. Retry non-idempotent operations only with a safe deduplication strategy.

Transfer, migrate and govern analytical access

- **Transfer planning** estimates bytes / usable throughput plus validation and contingency time. **Transfer Appliance** suits large offline transfers when the network cannot meet the deadline.
- **Storage Transfer Service** automates supported cloud and agent-based transfers. **gcloud storage** supports direct object operations. Use parallelism, resumability and appropriate compression for online transfer.
- **Checksums** such as CRC32C compare local content with object metadata without downloading every object again. Confirm completion and integrity before deleting the source.
- **Database Migration Service** supports eligible migration paths. For low-downtime cutover, seed data, replicate changes, stop writes, confirm catch-up, promote or finalize, and redirect clients.
- **Cloud SQL connectors and Auth Proxy** provide supported authenticated, encrypted connections. They still need network reachability, the correct workload identity, IAM permissions and database authorization.
- **BigQuery jobUser** permits job creation in the execution project; **dataViewer** grants read access to data at the selected scope. Query compute is charged to the execution project, separately from storage.
- **Partitioning** prunes data when predicates use the partition column; **clustering** narrows scans within storage blocks. Partition expiration ages out data without deleting a whole rolling-history table.
- **BigQuery INFORMATION_SCHEMA job views** expose query execution details for authorized analysis. Use job metadata and audit logs to investigate users, bytes processed and costly queries; billing export serves broader cost reporting.
- **Authorized views** expose approved query results without granting direct source-table access. Use supported row/column controls for fine-grained access and preserve a separate ability to run query jobs.
- **Signed URLs** delegate limited object operations for a bounded time. Browser uploads can also require CORS configuration. They are bearer credentials; use identity-based IAM when individual audit attribution is required.

Reliability & Operations

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

OPERATE FROM USER OUTCOMES

- **Observe** — Logs and metrics
- → **Evaluate** — SLO and budget
- → **Respond** — Mitigate impact
- → **Learn** — Improve runbooks

06 Ensuring solution and operations excellence 12.5%

Metrics, logs, traces and profiles

- **Cloud Monitoring** collects metrics, dashboards and alerts. **Cloud Logging** searches events; **Error Reporting** groups application errors. Install the **Ops Agent** for required VM guest telemetry.
- **Cloud Trace** follows request latency across services. Propagate trace context with supported instrumentation or auto-instrumentation, and correlate structured logs with trace IDs.
- **Cloud Profiler** identifies supported code-level CPU or memory costs. Traces explain where request time is spent; profiles explain where program resources are consumed.
- **GKE logs** from container stdout/stderr reveal application failures; Kubernetes events and previous-container logs help with restart loops. Node system logs answer different questions.
- **Managed Service for Prometheus** supports scalable Prometheus metrics and PromQL. Reuse dashboards through configuration export and adjust project, resource and environment filters.
- **Cloud Audit Logs** record administrative and supported data-access activity. Verify the required Data Access configuration, including Kubernetes API interactions; container application logs do not replace audit logs.
- **Aggregated log sinks** at folder or organization scope route matching descendant logs to central destinations. Grant the sink writer identity access and choose the narrowest scope covering existing and future projects.
- **Retention design** separates active query from long-term evidence. Use appropriate log-bucket retention, BigQuery or Cloud Storage destinations; a dashboard does not extend underlying retention.
- **Log Analytics** supports SQL analysis of eligible log buckets and linked BigQuery datasets. Use **Looker Studio** for reporting, and route selected events to Pub/Sub for automated response.

SLIs, SLOs and safe release operations

- **SLIs** measure user-visible behavior such as successful-request ratio and latency. **SLOs** set targets; an **SLA** is a contractual commitment. CPU utilization is useful capacity telemetry, not a complete user experience.
- **Error budgets** express permitted unreliability. Burn-rate alerts identify rapid consumption and support actionable paging; use dashboards or tickets for low-urgency diagnostics.
- **Canary releases** shift a small share of real traffic to new code; **blue-green releases** switch between complete environments. Compare success and latency before wider rollout.
- **Cloud Run revisions** and **App Engine versions** support traffic splitting and fast traffic rollback. **GKE rolling updates** use readiness, surge and unavailability settings to manage replacement.
- **Readiness probes** gate traffic; **liveness probes** trigger restarts; **startup probes** protect slow initialization. Overly aggressive health checks can create a restart loop.

Reliability & Operations

Google Cloud architecture, design and operations

KNOWLEDGE CHEATSHEET

OPERATE FROM USER OUTCOMES

Observe Logs and metrics → **Evaluate** SLO and budget → **Respond** Mitigate impact → **Learn** Improve runbooks

- **Database schema changes** should remain compatible with old and new application versions during rollout. Use expand/migrate/contract sequencing; rolling back code does not automatically roll back data mutations.
- **Multi-region services** require regional compute deployments and an appropriate global entry point. GKE multi-cluster Gateway or serverless NEGs can expose suitable backends; data placement and failover remain separate decisions.

Diagnose, mitigate and learn

- **Incident response** prioritizes restoring service with a known-good rollback or safe capacity action. Preserve logs, timelines and change evidence, then investigate root cause.
- **Scaling failures** can come from quotas, maximum replicas, initialization delay, unschedulable Pods, unhealthy backends or downstream saturation. Check each limit before changing the scaling signal.
- **Network diagnosis** checks DNS, routes, firewall decisions and application listeners. **VPC Flow Logs** summarize flows; **Firewall Rules Logging** records rule matches and supports firewall insight analysis.
- **SSH failures** warrant identity, firewall and daemon checks. Use serial-console output or an isolated copy of a disk for investigation when needed; avoid destroying a still-serving database to restore management access.
- **Uptime checks** test an endpoint from outside the application. Ensure the intended probe sources can reach it, and verify the notification and remediation path rather than relying only on a dashboard.
- **Shutdown hooks** are best effort under termination and should finish quickly. Use durable, idempotent cleanup or reconciliation rather than relying on a hook as the sole record of instance deletion.
- **Blameless postmortems** record impact, contributing conditions and corrective actions with owners. Test the fix, update runbooks and exercises, and use official service status or security bulletins for relevant incidents.