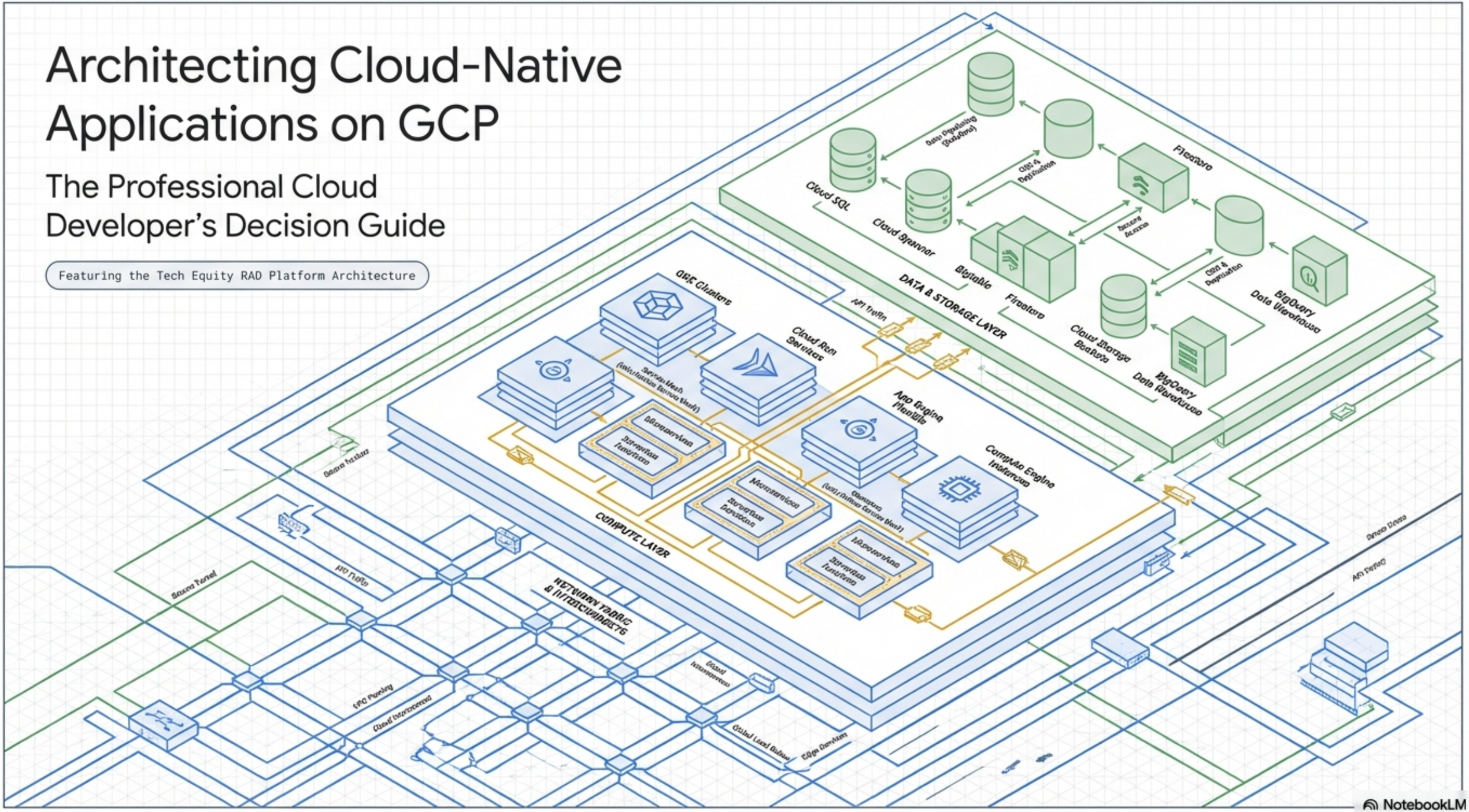
Architecting Cloud-Native Applications on GCP

The Professional Cloud Developer's Decision Guide

Featuring the Tech Equity RAD Platform Architecture

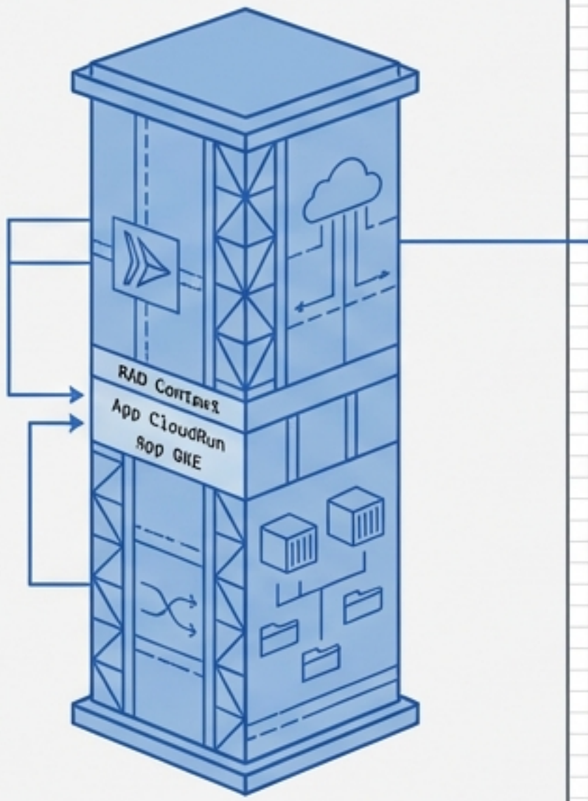


The Core Trinity of Cloud Architecture

Building cloud-native applications requires mastering three intersecting domains. This guide uses the Tech Equity RAD platform to map textbook theory to functional deployment variables.

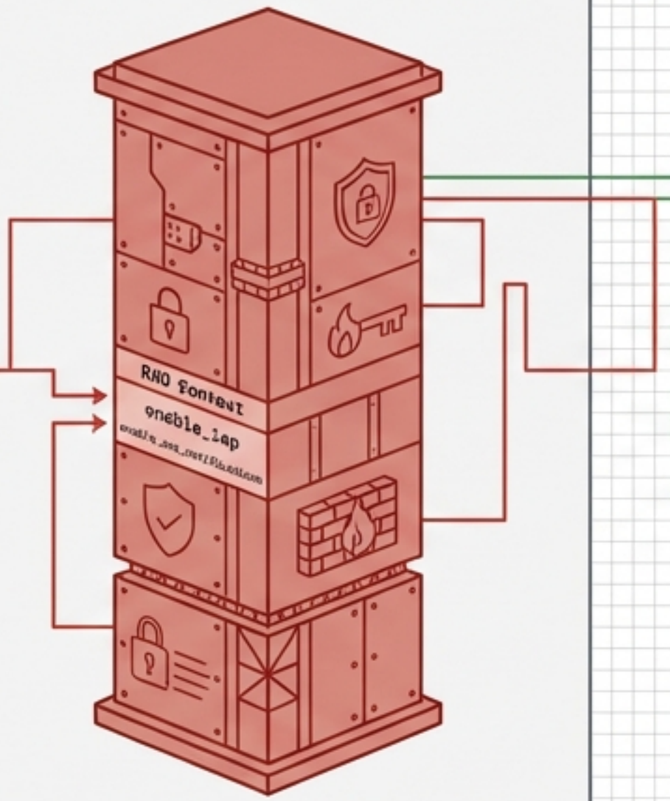
Performance & Compute

Routing traffic, scaling to zero, and orchestrating microservices.



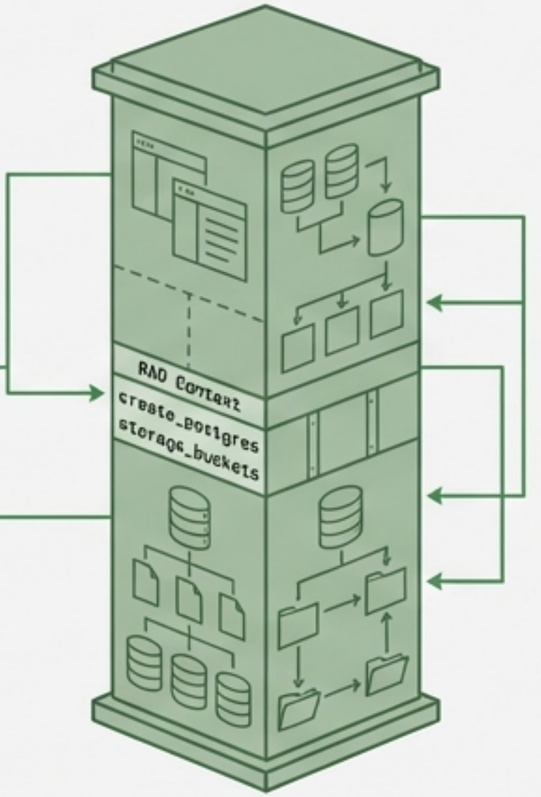
Security & Access

Zero-trust identity, supply chain defense, and compliance.



Data & State

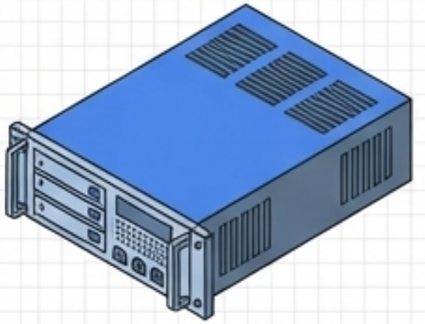
Consistency models, schema design, and high-throughput analytics.



Choosing the Compute Foundation

Workload Requirements

Requires Custom OS / Kernel access / Legacy hardware?



Compute Engine

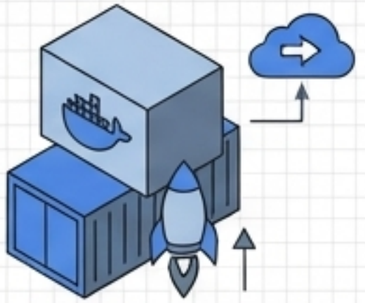
`Provisioned via Services GCP`

Requires persistent connections, StatefulSets, or fine-grained networking?

GKE Autopilot

`Provisioned via App GKE`

Event-driven HTTP workload? Needs to scale to zero?



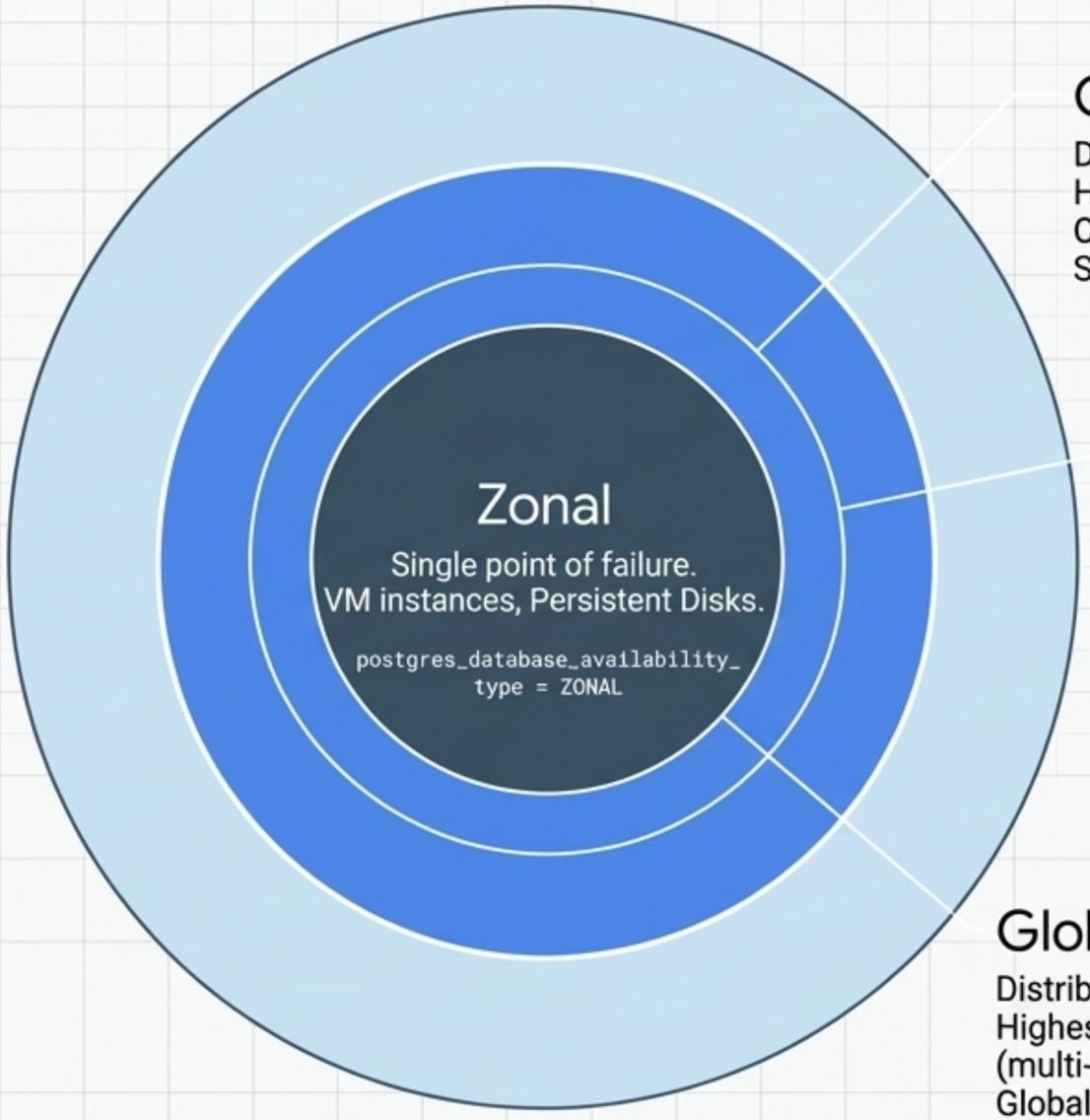
Cloud Run

`Provisioned via App CloudRun`

Cost Optimization

Right-sizing resources dictates cost. Set min_instance_count = 0 to incur zero compute cost when idle. Use max_instance_count to bound scaling costs during spikes, and container_resources to prevent CPU/memory over-provisioning.

Availability and the Blast Radius



Global

Distributed across regions.
Highest availability.
Cloud Storage (multi-region), Spanner, Global Load Balancing.

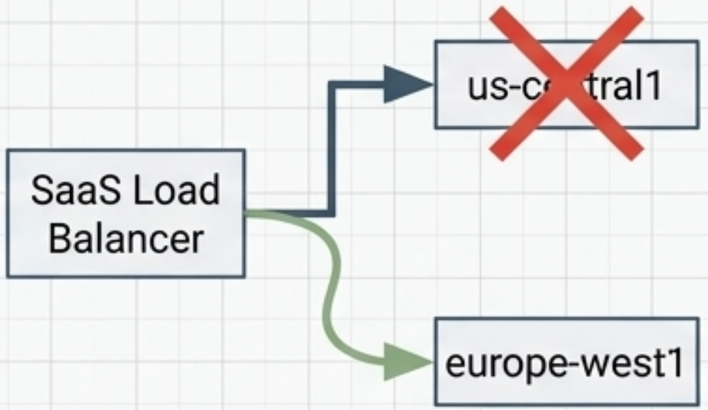
Regional

Survives single zone failure.
Cloud Run, GKE Autopilot, Cloud SQL HA.

Automatic failover in ~60s if `availability_type = REGIONAL`

Global

Distributed across regions.
Highest availability. Cloud Storage (multi-region), Spanner, Global Load Balancing.

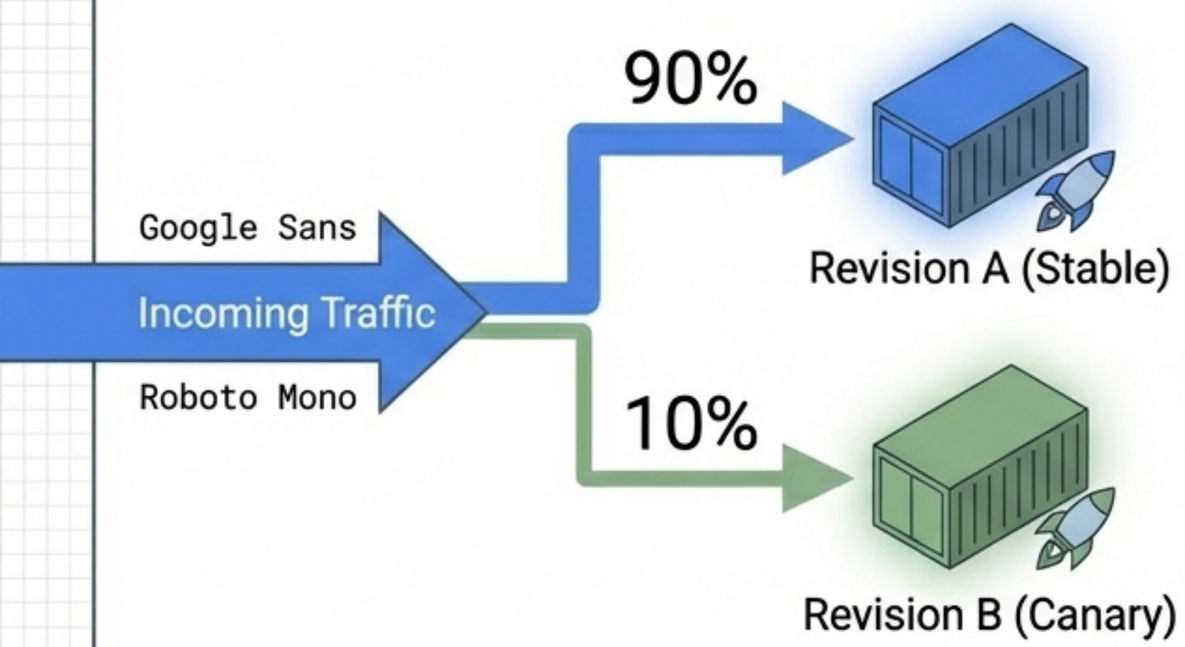


Failover Routing

Health checks detect regional degradation instantly. Traffic shifts to the nearest healthy region with zero application downtime, controlled by the `availability_regions` variable.

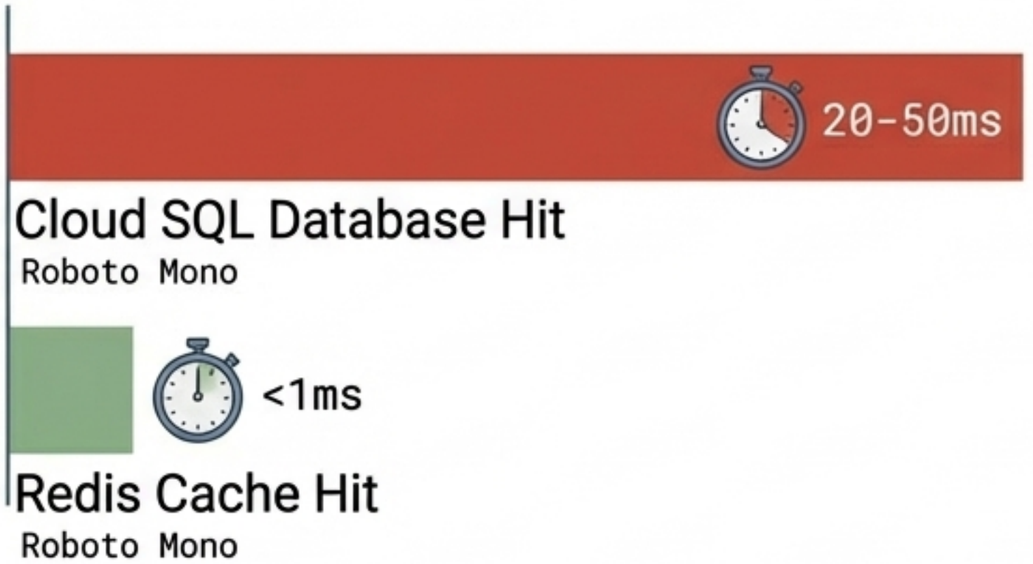
Traffic Rollouts and Microsecond Caching

Immutable Revisions & Splitting



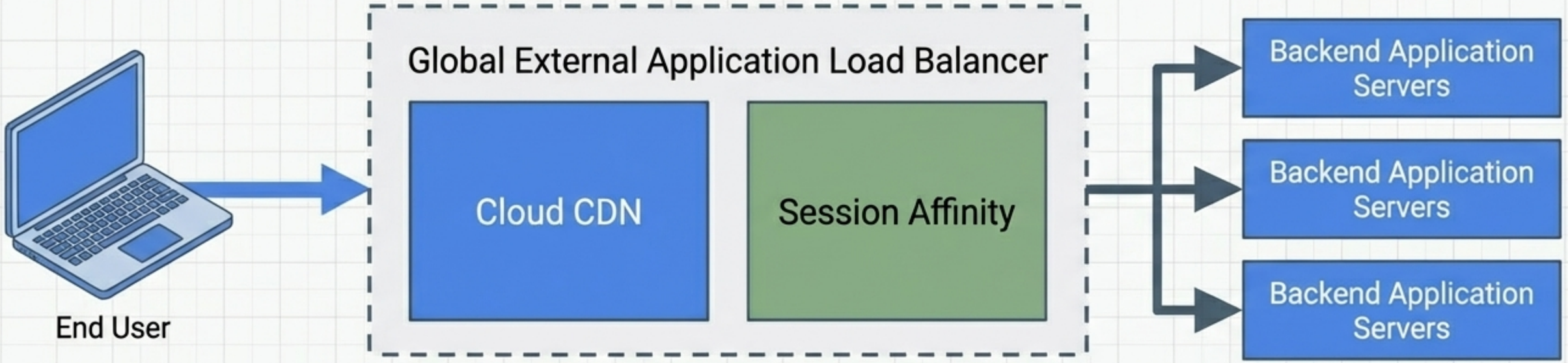
Cloud Run revisions are immutable. Use the `traffic_split` variable for A/B testing. If the 10% canary algorithm underperforms, rollback is instant by routing 100% back to stable.

Memorystore Redis Caching



Enabled via `create_redis` and `redis_tier`. Passing the private Redis IP directly into the container drops cache-hit latency under 1ms, skipping Cloud SQL entirely and reducing DB CPU load by 60%.

Edge Routing and State Preservation



Cloud CDN Caching Modes

Caches at global Points of Presence (PoPs).
- `CACHE_ALL_STATIC`: Auto-caches images/CSS.
- `USE_ORIGIN_HEADERS`: Respects app Cache-Control.
- `FORCE_CACHE_ALL`: Overrides headers (use with caution).

Session Affinity Options

Critical when legacy apps store state in local memory.
- `CLIENT_IP`: Simple, but breaks behind NAT/proxies.
- `GENERATED_COOKIE`: Load balancer sets a cookie on the first response. Recommended for browser apps.
- `HEADER_FIELD`: Routes based on specific HTTP header values.

API Architecture & Management

Protocol Choice

HTTP REST	gRPC
Standard **JSON** payloads and standard HTTP methods. Native to Cloud Run. Best for browser clients.	**Protocol Buffers** and HTTP/2 transport. Highly efficient serialization. Best for service-to-service communication.

Gateway Matrix

Cloud API Gateway	Apigee
Lightweight, low-latency managed gateway. Defined via `OpenAPI 2.0` specifications. Enforces basic rate limits (e.g., `100 req/min`) and `JWT auth`. Best for simple API surfaces.	Enterprise API management platform. Provides CIAM, monetization, and OAuth 2.0 flows. Advanced traffic analytics and transformation policies. Best for complex, multi-tier external developer ecosystems.

The Orchestration Matrix

Cloud Workflows	Cloud Tasks	Cloud Scheduler	Eventarc
Primary Use Case: Multi-step, sequential coordination. **Statefulness:** Stateful. **Key Capability:** Handles branching logic, retries, and sequential HTTP service calls (e.g., Inventory → Payment → Shipping).	**Primary Use Case:** Rate-controlled asynchronous execution. **Statefulness:** Stateless. **Key Capability:** Dispatches single tasks to targets (e.g., sending at most 50 emails/minute). Supports deduplication and future scheduling.	**Primary Use Case:** Fully managed cron service. **Statefulness:** Stateless. **Key Capability:** Time-based invocation of endpoints or topics (e.g., nightly batch jobs).	**Primary Use Case:** Event routing. **Statefulness:** Stateless. **Key Capability:** Routes triggers via CloudEvents (Storage finalized, Audit Logs, Pub/Sub) directly to compute targets. The glue for event-driven systems.

Synthesis: Unified Event-Driven Architecture

Stage 1: Ingest

PDF uploaded to GCS.

Stage 2: Trigger

Eventarc detects `object.v1.finalized`.

Stage 3: Process

Cloud Run extracts text via Document AI and saves to Firestore.

Stage 4: Decouple

Cloud Run publishes `document.processed` message to Pub/Sub.

`Zero service coupling.` Independent downstream subscribers.

Stage 5: Fan-Out

Email Notification

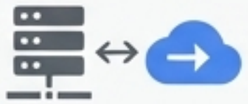
Search Index Update

Compliance Workflow

Securing Foundational Access

Authentication Source

Application Default Credentials (ADC)



The gold standard for code running on GCP. Automatically uses the service account bound to the workload (Workload Identity). No credential files needed.

Workload Identity Federation

For code outside GCP (AWS, GitHub Actions).
Exchanges external OIDC tokens for short-lived Google access tokens.

JWT & OAuth 2.0

For service-to-service calls.

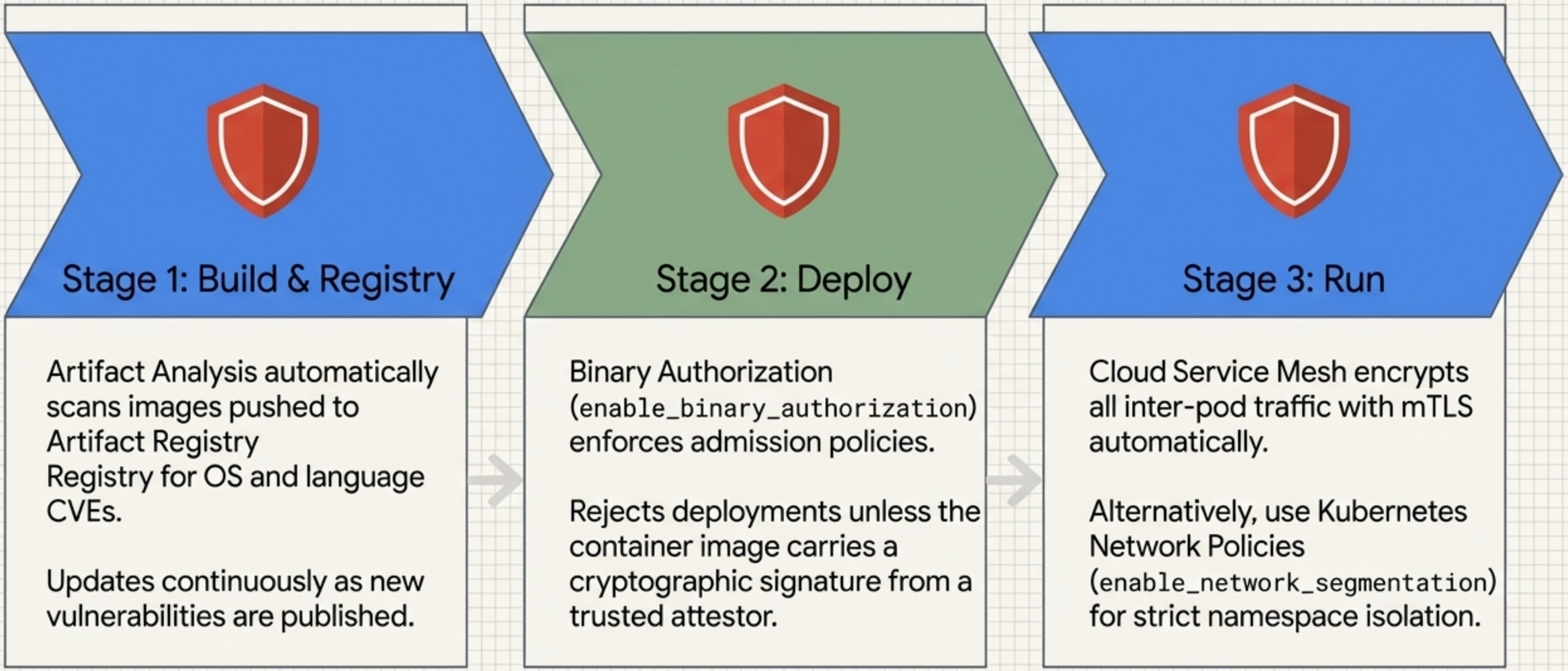
A service account generates a signed JWT, exchanged for a short-lived bearer token.

Anti-Pattern Alert

Avoid long-lived JSON keys (hard to audit, no auto-rotation).

Use Identity-Aware Proxy (`enable_iap`) for zero-trust end-user access.

Securing the Supply Chain & Microservices



Stage 1: Build & Registry

Artifact Analysis automatically scans images pushed to Artifact Registry
Registry for OS and language CVEs.

Updates continuously as new vulnerabilities are published.

Stage 2: Deploy

Binary Authorization (`enable_binary_authorization`) enforces admission policies.

Rejects deployments unless the container image carries a cryptographic signature from a trusted attestor.

Stage 3: Run

Cloud Service Mesh encrypts all inter-pod traffic with mTLS automatically.

Alternatively, use Kubernetes Network Policies (`enable_network_segmentation`) for strict namespace isolation.

Data Compliance and Retention

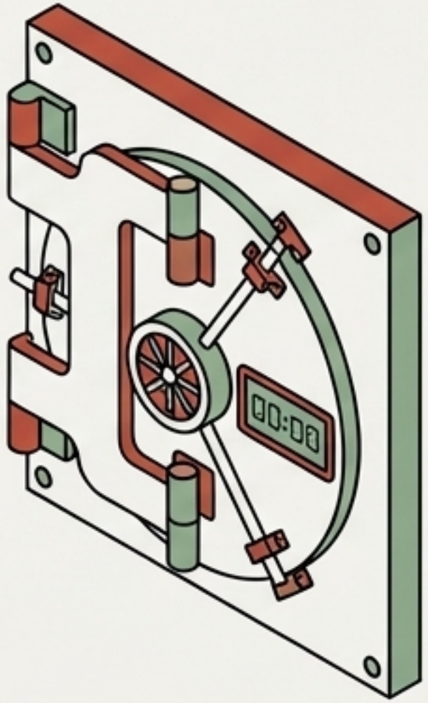
Crypto-Shredding with CMEK

Customer-Managed Encryption Keys allow instant data revocation.

Encrypt tenant data with a unique `KMS key`.

To comply with deletion requests, simply destroy the `KMS key`—rendering all associated data permanently inaccessible in minutes.

WORM Storage with Bucket Lock



For regulatory compliance (**SEC 17a-4, HIPAA**), use `Cloud Storage Object Lifecycle Management` combined with `Bucket Lock`.

Enforces a permanent retention period where objects cannot be deleted or overwritten by anyone—even project owners—until expiry.

The Master Database Selection Matrix

Structure: Relational vs. NoSQL

Workload: Transactional (OLTP) vs. Analytical (High-Throughput)

Cloud SQL

Standard web apps (`create_postgres`). Single-primary, strong consistency. Max 64TB.

Cloud Spanner

Limitless horizontal scaling. Global reads/writes with TrueTime external consistency.

AlloyDB

HTAP processing. Disaggregated storage. 4x OLTP and 100x analytical throughput vs standard PostgreSQL.

Firestore

Serverless, flat document maps. Strong single-document consistency. Ideal for real-time app listeners.

Bigtable

Wide-column. Petabyte scale, single-digit ms latency. Best for IoT and time-series. The row key is the only index.

Schema Design & Consistency Trade-Offs

Consistency Spectrum

Strong / Serialisable — External Consistency — Eventually Consistent

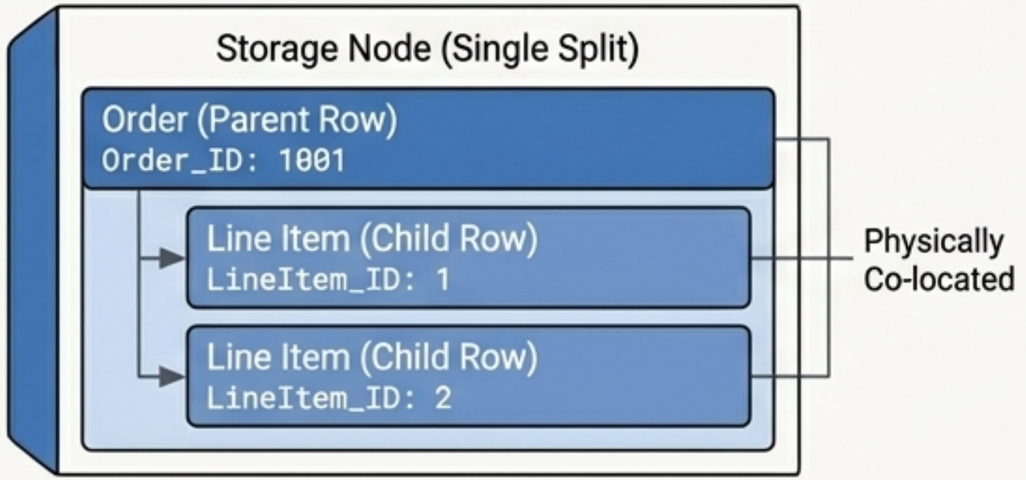
Roboto Regular, Mono
Cloud SQL, Spanner

Matte Azure Blue
Spanner TrueTime

Roboto Regular, Mono
Bigtable multi-cluster replica

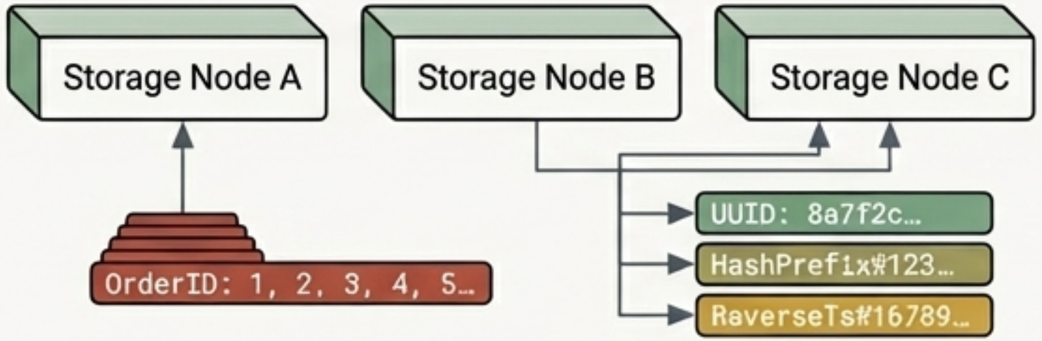
Schema Design Rules

Spanner Schema Interleaving



Prevent cross-server joins by physically co-locating parent and child rows. E.g., `INTERLEAVE IN PARENT Orders`.

Preventing Hotspots



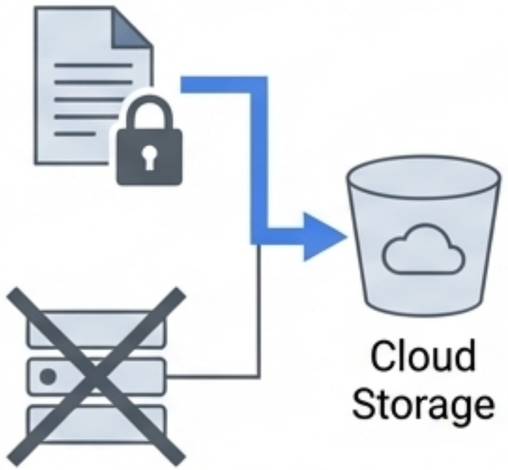
BAD: Auto-incrementing IDs
Hotspot on single node.

GOOD: Distributed Keys
Writes evenly distributed across all nodes.

Never use auto-incrementing integer IDs. In Spanner, use UUIDs. In Bigtable, use hash-prefixed row keys or reverse-timestamps (`<device-id>#<reverse-timestamp>`) to evenly distribute writes.

Analytics Ingestion and BigQuery

Secure Edge Ingestion



Use **Signed URLs** to grant time-limited, unauthenticated PUT/GET access directly to GCS, bypassing the app server for heavy uploads.

Streaming Inserts

Real-time rows, queryable instantly (no updates for 30 mins).

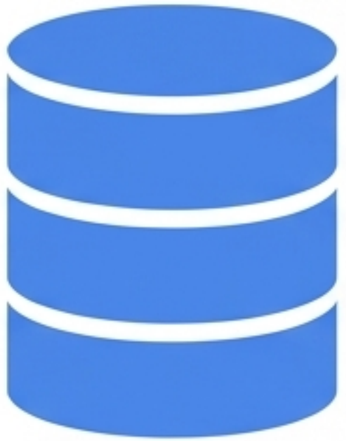
Storage Write API

High-throughput streaming with exactly-once semantics (Recommended).

Batch Loads

Zero-cost ingestion for nightly ETL via CSV/Parquet.

BigQuery Data Warehouse



Vertex AI Integration

BigQuery integrates directly with Vertex AI. Use raw event telemetry (e.g., 50k events/sec) for ML model training immediately without copying files.