Gemini Enterprise Agent Platform by Michaël Bettan

Definition: Gemini Enterprise Agent Platform is the evolution of Vertex AI for building, scaling, governing, and optimizing agents on Google Cloud. This guide covers the *build, retrieve, run, and optimize* lifecycle: choosing among the three build paths (**Agent Studio**, **Managed Agents API**, and code-first **ADK / open frameworks** with `agent-starter-pack` and `agents-cli`), hosting agents on **Agent Runtime** (formerly Agent Engine), managed **Sessions**, **Memory Bank**, **Example Store**, and the **Code Execution** sandbox, choosing between Agent Runtime, Cloud Run, and GKE, grounding and retrieval services, caching and external state, the **Agent Evaluation** suite (2-step user simulation, environment simulation, adaptive rubrics, trajectory metrics, online monitors, and golden-dataset flywheels), **Agent Observability** (Agent Registry telemetry views, dual-attribution audit logs, failure cluster analysis, and programmatic prompt optimization), and cost/quota management.

Out of scope here: cryptographic identity, IAM, Principal Access Boundary (PAB), VPC Service Controls, Model Armor, Sensitive Data Protection, Agent Gateway, and governance policies are covered under **Agent Platform: Identity, Security & Governance**; ADK code-level patterns, vendor-neutral RAG design, Gemini model selection, and A2A wire mechanics are covered in their dedicated topic guides.

MENTAL MODEL (5-PART ANATOMY MAPPED TO GOOGLE CLOUD)

The **model** (Gemini 4 Argon, Gemini 3.8 Flash…) is the reasoning brain; the **orchestration layer** (ADK, LangGraph, Managed Agents harness, or Agent Studio) wires the Think → Act → Observe loop; **Agent Runtime** is the managed body that keeps long-running tasks alive; **Sessions** are short-term working memory for one conversation while **Memory Bank** is consolidated long-term memory across conversations; **MCP servers, retrieval services, and the Code Execution sandbox** are the hands and library; **Evaluation + Observability + Optimizer** form the continuous AgentOps quality flywheel.

RULE 0 (MANAGED BY DEFAULT)

Choose the most managed component that meets the requirement. Move down the stack (**Managed Agents API / Agent Studio** → **Agent Runtime** → **Cloud Run** → **GKE**) only for a concrete engineering need: custom orchestration code, non-Python stacks, hosting stateless Streamable HTTP MCP servers, GPUs/self-hosted open weights, or fine-grained kernel/network control.

NOTE

Product names, preview status, quotas, and regions evolve continuously. Items marked **Preview** are Pre-GA and do not carry GA SLAs. Always verify launch stage, regional availability, and pricing against current Google Cloud documentation before production commitment.

01. Platform Map: Naming, Pillars & Build Paths

The four pillars

Pillar	What it covers	Key components
Build	Create agents, attach skills/tools, and connect to enterprise data	**ADK** (Python, Go, Java, TypeScript), **Agent Studio** (low-code visual builder), **Managed Agents API** + **Interactions API** (**Preview**), **Skill Registry**, **Agent Garden** (samples/templates), **Model Garden** (200+ models), **RAG Engine**, **Agent Search**, **Agent Retrieval**, `agent-starter-pack`, `agents-cli`
Scale	Run stateful, multi-step agents reliably in production	**Agent Runtime** (`reasoningEngines`), **Sessions**, **Memory Bank**, **Example Store** (**Preview**), **Code Execution** sandbox, **A2A Agent Hosting**
Govern	Control identity, discovery, egress, and security policy	**Agent Registry**, **Agent Identity** (SPIFFE), **Agent Gateway**, **Model Armor**, governance policies → see **Security guide**
Optimize	Simulate, evaluate, observe, and continuously improve quality	**Gen AI Evaluation Service** (2-step user simulation, environment simulation, adaptive rubrics, trajectory metrics, **Online Monitors**), **Agent Registry Observability**, **Failure Cluster Analysis**, **Prompt Optimization / Agent Optimizer**

Billing model: Agent Platform is **pay-as-you-go** consumption (tokens, vCPU/GiB-seconds on runtimes, storage/queries on indexes) with optional **Provisioned Throughput** (GSUs) for developers; the **Gemini Enterprise app** (formerly Agentspace) is a separate per-seat subscription workspace where business users discover and invoke published agents.

Three build paths on Agent Platform

Build path	Target builder & control level	How the loop & state work	Best fit
1. Agent Studio (low-code)	Citizen developers, domain experts, rapid prototypers	Visual canvas to configure instructions, Gemini model, grounding data stores, MCP tools, and sub-agents; test in-console and publish 1-click to **Agent Runtime** or **Gemini Enterprise Workflow Builder**	Internal knowledge assistants, departmental workflows, rapid validation before pro-code handoff
2. Managed Agents API (Preview)	Developers wanting a declarative, harness-managed agent via REST/SDK	Combines the **Agents API** (reusable agent definition + mounted **Skill Registry** skills) with the stateful **Interactions API** (`client.interactions.create` , server-side `previous_interaction_id` , `background=true`). Google owns the Think-Act-Observe loop, automatic restarts, artifacts, and an isolated Linux sandbox (**no network egress by default**). Powers managed endpoints like **Antigravity Agent** and **Deep Research**.	Autonomous coding, file analysis, research, and document-producing tasks where you want zero harness maintenance
3. Code-first ADK & Open Frameworks	Software & AI engineers needing deterministic control over control flow, callbacks, and CI/CD	You author orchestration code in **ADK**, **LangGraph**, **CrewAI**, **AG2**, or **LlamaIndex**, scaffold production CI/CD + Terraform via **`agent-starter-pack`** / **`agents-cli`** , and deploy to **Agent Runtime**, **Cloud Run**, or **GKE**	Complex multi-agent topologies, strict compliance callbacks, custom deterministic guardrails, hybrid A2A meshes

Name changes you must recognize

Current name	Former name	Exam / API note
Gemini Enterprise Agent Platform	Vertex AI	Evolution of the same platform; IAM permissions still use `aiplatform.*`
Gemini Enterprise (end-user app)	Google Agentspace	Unified enterprise search + multi-agent workspace supporting A2A and **A2UI**
Agent Runtime	Vertex AI Agent Engine / Reasoning Engine	Underlying REST/gRPC resource remains `projects/…/locations/…/reasoningEngines/<id>`
Agent Search	Vertex AI Search	Turnkey managed search/RAG (`discoveryengine.googleapis.com`) with connectors and native ACLs
Agent Retrieval	Vector Search 2.0	Collections-based, self-tuning vector store with schema and auto-embeddings
Vector Search 1.0	Matching Engine	High-scale ScaNN ANN index-as-a-service
Workflow Builder (Gemini Enterprise)	Agent Designer	No-code builder for end-user chat agents and scheduled/triggered workflows in Gemini Enterprise
Gemini 4 Argon / Gemini 3.x	Gemini 1.5 / PaLM 2	Active frontier (`gemini-4-argon`) and workhorse (`gemini-3.8-flash`) generations

MODEL GARDEN POINTER

Model Garden catalogs Google first-party (Gemini 4 Argon, Gemini 3.8 Flash, Nano Banana 2.1, Gemini Omni 1.1 Flash, Veo 3.1, Lyria 3.5, Chirp 3), partner MaaS (Anthropic Claude, Mistral, DeepSeek), and open-weight models (Gemma 4, Gemma 3n, Llama). **Gemini itself is served only through managed Google endpoints** — you cannot self-host Gemini weights on Cloud Run or GKE; open models such as Gemma 4 can be self-hosted there.

02. Agent Runtime: Deploying & Scaling Stateful Agents

Agent Runtime is Google Cloud's fully managed, opinionated compute runtime purpose-built for stateful agents: sub-second cold starts, optional `min_instances` to keep instances warm, long-running and multi-day asynchronous execution, IAM and **VPC Service Controls** perimeter enforcement, private egress to on-prem/VPC resources via **Private Service Connect (PSC) interface** network attachments, cryptographic **Agent Identity** (`spiffe://…`), and zero-plumbing integration with **Sessions**, **Memory Bank**, **Example Store**, **Code Execution**, **A2A**, and OpenTelemetry observability.

Framework & protocol support

Support level	Frameworks & protocols
First-class native integration	**ADK** (Python via `adk deploy agent_engine` or `agents-cli` ; Go via `adkgo` ; Java & TypeScript via container deployment)
Built-in SDK templates	**LangChain**, **LangGraph**, **AG2**, **LlamaIndex** (**Preview**)
Custom Python class	**CrewAI** or any custom Python class implementing `query` / `async_query` / `stream_query`
Agent2Agent (A2A) hosting	Native **`A2aAgent`** wrapper (**Preview**) exposing an `Agent Card` , task store, and A2A task endpoints directly on Agent Runtime
Any language / runtime	Custom **Dockerfile** or pre-built **Artifact Registry container image** conforming to the Agent Runtime HTTP contract

Five deployment methods

Method	Languages	Best for
Developer Connect (Git repository)	Python	GitOps CI/CD — automatically builds and deploys directly from a connected Git repo without managing local build artifacts
Source files (inline directory)	Python	Automated CI/CD pipelines and Terraform; packages local source directly with no Cloud Storage staging bucket required
Dockerfile	Any	Non-Python agents (Go, Java, TypeScript/Node) or Python agents needing OS-level system packages
Container image (Artifact Registry)	Any	Pre-built, vulnerability-scanned immutable images promoted from staging to prod; **lowest deployment latency**
SDK object (in-memory pickle)	Python	Rapid notebook/Colab prototyping (`client.runtimes.create(agent=my_agent)`); object must be serializable

SDK, `agent-starter-pack` & `agents-cli` production workflow

- **Agent Platform SDK (`google-cloud-agentplatform`):** Exposes `agentplatform.Client(project=…, location=…)` with `client.runtimes` , `client.sessions` , `client.sandboxes` , and `client.memory_banks` (superseding the legacy `vertexai.agent_engines` namespace).
- **`agent-starter-pack` & `agents-cli` :** Production scaffolding toolkit that generates standardized repo layouts:
- Infrastructure-as-Code (**Terraform**) for staging/prod projects, least-privilege IAM, Artifact Registry, and VPC-SC/PSC wiring.
- **Cloud Build** or GitHub Actions CI/CD pipelines with **evaluation gates** (`agents-cli eval` runs golden + trajectory evalsets on every PR and blocks promotion if scores regress).
- Pre-wired **OpenTelemetry** tracing, BigQuery log sinks, and feedback collection for the post-deployment data flywheel.
- **Managed Agents API (`Agents API` + `Interactions API` , Preview):** Instead of deploying container code to `reasoningEngines` , you register an Agent config + **Skill Registry** bundles (`SKILL.md` + helper scripts/assets) and invoke it via `Interactions API` . Server-side state (`previous_interaction_id`) and `background=true` allow multi-hour execution without client polling loops, inside an isolated Linux container with **network egress disabled by default**.

> **COMMON TRAP — RESERVED ENVIRONMENT VARIABLES**
> Never override platform-managed environment variables such as `GOOGLE_CLOUD_PROJECT` or `PORT` in your deployment spec, and never use the reserved `GOOGLE_CLOUD_AGENT_ENGINE*` prefix for custom application variables (other than documented flags like `GOOGLE_CLOUD_AGENT_ENGINE_ENABLE_TELEMETRY=true`).

> **EXAM TIP**
> **Agent Runtime hosts stateful agents, NOT standalone MCP tool servers.** Deploy custom **Streamable HTTP MCP servers** on **Cloud Run** (for serverless scale-to-zero) or **GKE**, register them in **Agent Registry**, and invoke them from your Agent Runtime agent.

03. Choosing a Runtime: Agent Runtime vs. Cloud Run vs. GKE

Criterion	Agent Runtime	Cloud Run	GKE (Standard / Autopilot)
Primary workload	Stateful single- or multi-agent orchestrators (ADK, LangGraph, A2A)	Stateless A2UI/APIs, webhook handlers, **Streamable HTTP MCP servers** (`--functional-type=mcp-server`), async Jobs	Self-hosted open-weight models (Gemma 4 via vLLM/TGI), GPU/TPU workloads, dense multi-service meshes
Ops overhead	Lowest (fully managed agent harness)	Low (serverless containers)	Highest (cluster, node-pool, and networking lifecycle)
State & memory	Native **Sessions**, **Memory Bank**, **Example Store**, **Code Execution** & **Computer Use** (`Preview`)	Ephemeral local disk/RAM; externalize state to Agent Platform Sessions, Memorystore, Firestore, Spanner, or Cloud SQL	StatefulSets, local SSDs, or external managed databases
Scaling, revisions & duration	Autoscales up to **`1,000` instances** (**`100` in VPC-SC**); `min_instances` & `container_concurrency` (`Preview`); up to **`6,000` revisions** with `traffic-split` (*bypassed if Agent Gateway routes*); multi-day async workflows	Scales to zero (`$0` idle) up to `1,000` instances (`80-1,000` concurrency); HTTP timeout up to **60 min** (Services) / **24h** (Jobs); `--to-revisions` canary splits	Full control over HPA/VPA, custom metrics, GPU time-slicing, Istio/Gateway API L7 splits, and spot nodes
Isolation & security	IAM, VPC-SC, PSC interface (`AgentConnectivityTemplate`), Agent Identity, isolated Code Execution sandbox	IAM, Direct VPC egress, VPC-SC, built-in Agent Identity (`--functional-type=agent`) & Agent Registry integration	Workload Identity Federation, **GKE Sandbox (`RuntimeClassName: gvisor` / Kata)** kernel isolation, NetworkPolicies
Hosts MCP servers?	**No**	**Yes** (ideal for Streamable HTTP MCP servers)	**Yes** (high-density or stateful MCP fleets)
Main trade-off	**No custom GPUs/TPUs inside the runtime container**; non-Python stacks require custom containers	In-process memory vanishes on scale-in or instance recycle (optional L4 / RTX PRO 6000 GPUs for bursts)	Highest operational and Kubernetes platform engineering cost (full H100/H200/B200 & TPU control)

Architectural decision rules:

1. **Python/ADK/LangGraph agent + minimal ops + managed Sessions & Memory Bank → Agent Runtime.**
2. **Custom remote MCP server (Streamable HTTP), polyglot TypeScript/Java/Go webhook agent, or spiky scale-to-zero workload → Cloud Run.**
3. **Self-hosting Gemma 4 / Llama on GPUs/TPUs, strict kernel-level multi-tenant sandboxing (`gVisor`), or existing enterprise K8s platform → GKE.**
4. **Zero-infrastructure autonomous coding/file/research agent where Google manages the loop and Linux sandbox → Managed Agents API (Preview).**
5. **Hybrid regulated GPU topology (stateful ADK orchestrator + private self-hosted Gemma 4 SLM):** Host the orchestrator on **Agent Runtime** (for turnkey Sessions, Memory Bank, and SPIFFE Agent Identity) and invoke a private **GKE** GPU `vLLM` worker (or **Model Garden** `PrivateEndpoint`) over a **PSC interface** (`Agent ConnectivityTemplate` with `vpcEgress: ALL_TRAFFIC` and `restricted.googleapis.com` `199.36.153.4/30`) inside **VPC-SC**—never attempt to attach GPUs directly to an Agent Runtime container.

> **COMMON TRAP — KEEPING CONVERSATION HISTORY IN CLOUD RUN RAM**
> Cloud Run instances scale to zero or recycle unpredictably, wiping in-process dictionaries and local SQLite files. Always externalize conversational state to **Agent Platform Sessions** (`--session_service_uri agentengine://<id>`), **Memorystore**, **Firestore**, **Spanner**, or **Cloud SQL**.

04. Context Engineering: Sessions, Memory Bank & Example Store

Effective agents practice **context engineering** — dynamically assembling only the task-relevant history, user facts, and few-shot demonstrations needed for the current turn rather than stuffing entire raw logs into the prompt window (which causes **context rot**, latency spikes, and quadratic attention cost).

Core state vocabulary

Abstraction	Scope	Lifetime	Typical contents
Session	Chronological append-only log of **Events** (user messages, model thoughts, tool calls, tool outputs)	Single conversation thread	Turn-by-turn dialogue and tool trace
State	Mutable key-value scratchpad (`session.state`, `user:`, `app:`, `temp:` prefixes in ADK)	Current turn (`temp:`) or current session	Cart IDs, workflow step flags, extracted slot values
Memory Bank	LLM-extracted, consolidated semantic facts & preferences	Persistent across sessions (with configurable TTL & revisions)	User role, preferred coding language, dietary/travel rules, recurring project context
Example Store (Preview)	Dynamically retrieved input→trajectory/output few-shot pairs	Persistent per agent/task corpus	Edge-case tool routing demonstrations, strict formatting exemplars

Managed Sessions, Context Compaction & Prefix Caching (Tier 1 + Tier 2): ADK agents on Agent Runtime automatically persist sessions (`VertexAiSessionService`), while Cloud Run, GKE, or local dev workstations attach via `--session_service_uri agentengine://<id>` (or `DatabaseSessionService` on Spanner, AlloyDB, or Cloud SQL). Configure ADK `EventsCompactionConfig(token_threshold=..., event_retention_size=...)` to summarize older tool trajectories before hitting token limits, and pair with `ContextCacheConfig` (or implicit prefix caching). **Prefix-cache trap:** placing dynamic timestamps or per-turn user IDs at the **top** of the system prompt invalidates prefix caching on every turn—keep static instructions and tool schemas first.

Memory Bank (long-term semantic & episodic memory)

- **Asynchronous LLM extraction & consolidation (`GenerateMemories`):** Extracts durable facts from session events in the background (zero hot-path latency penalty) and **consolidates** them with existing memories — merging duplicates, updating stale preferences, and resolving contradictions while keeping **memory revisions** for audit/rollback.
- **Explicit agent-triggered writes (`CreateMemory`):** Exposed as a tool ("memory-as-a-tool") when the user explicitly says *"Always format my SQL queries in GoogleSQL"* and immediate persistence is desired.
- **Scoped retrieval (`RetrieveMemories`):** Retrieves by exact scope or **semantic similarity search** against the active user turn.
- **Strict tenant isolation:** Every memory is keyed by an explicit identity scope such as `{user_id, app_name}` — never share an unscoped memory namespace across users.
- **Enterprise governance:** Custom **Memory Topics** (defining *what* is worth remembering and *what* PII must be ignored), multimodal event ingestion, **TTL** policies (default or per-memory expiration for GDPR/retention compliance), IAM Conditions, `roles/aiplatform.memoryViewer` / `roles/aiplatform.memoryEditor`, CMEK, and regional data residency (`gcp.resourceLocations`).
- **ADK integration:** `VertexAiMemoryBankService`; trigger incremental extraction from an `after_agent_callback` via `add_events_to_memory` (preferred over full-session `add_session_to_memory` to avoid redundant token processing).

> **COMMON TRAP — MEMORY POISONING**
> If untrusted external content (e.g., an email or web page fetched by a tool) tricks the extraction LLM into storing a malicious instruction in Memory Bank, that instruction is injected into *future* sessions (**persistent cross-session prompt injection**). Screen untrusted inputs with **Model Armor**, constrain extraction with strict **Memory Topics**, and audit memory revisions (**Security guide**).

Example Store (Preview)

Stores curated few-shot demonstrations and dynamically injects only the top-*k* semantically similar examples at inference time — improving edge-case accuracy without inflating every prompt.

- **Immutable embedding binding:** The embedding model (e.g., `text-embedding-005`, `text-multilingual-embedding-002`, or `gemini-embedding-001`) is **locked when the Example Store instance is created**; switching embedding models requires creating a new instance and re-indexing.
- **Quota awareness:** Up to 50 Example Store instances per project per region (verify current limits).

Requirement	Right component
Recall what *this user* asked three turns ago	**Session** events / scratchpad `state`
Remember *this user's* preferred timezone and reporting style next month	**Memory Bank**
Show the model 3 similar complex SQL/tool trajectories at runtime	**Example Store** (**Preview**)
Answer questions grounded in 50,000 corporate policy PDFs with ACLs	**Agent Search** (Section 06)

05. Code Execution Sandbox

The **Code Execution** sandbox executes untrusted, model-generated Python code inside a hardened micro-container completely separate from the agent's own runtime process:

- **Sub-second provisioning & stateful sessions:** Spins up and executes in `<1s`, supports file I/O up to **100 MB**, and preserves stateful sandbox files/variables across turns for up to **14 days** (configurable TTL).
- **ADK integration:** `AgentEngineSandboxCodeExecutor` binds **one isolated sandbox per session**, enabling multi-turn data science, chart generation, CSV reconciliation, and deterministic math verification. Callable even if the agent orchestrator itself runs on Cloud Run or GKE.
- **Managed Agents sandbox:** In the **Managed Agents API**, the built-in Linux execution environment runs with **no outbound network access by default** and mounts approved **Skill Registry** scripts/files cleanly into the workspace.

> MENTAL MODEL — RUNTIME VS. SANDBOX
> **Agent Runtime** runs *trusted* orchestration code written and reviewed by engineers (and holds IAM/MCP credentials). The **Code Execution sandbox** runs *untrusted* code synthesized on the fly by an LLM. Running `exec()` inside the agent's own container exposes service-account tokens, environment variables, and VPC network paths.

06. Grounding, Retrieval & Enterprise Tooling

Retrieval service decision matrix

Service	Architecture	Choose when	Key limits & watch-outs
Agent Search (formerly Vertex AI Search)	Turnkey managed search & RAG (`discoveryengine`)	Enterprise documents across SharePoint, Drive, Confluence, Jira, GCS, or BigQuery needing built-in layout parsing, hybrid search, reranking, citations, and **document-level source ACL enforcement**	Less low-level control over custom chunking or bespoke embedding pipelines; max **10 data stores** per grounding request
RAG Engine	Code-first modular RAG orchestration over **corpora** (`RagCorpus` / `RagFile`)	Engineers need explicit control over chunk size/overlap, parser choice (default, LLM parser, **Document AI Layout Parser**), embedding model, and backing store (**RagManagedDb**, Agent Retrieval, Vector Search 1.0, Pinecone, Weaviate)	`ImportRagFiles` Layout Parser caps PDFs at **20 MB / 500 pages**; **Serverless mode** lacks CMEK while **Spanner mode** supports CMEK (**one-way mode lock-in**: corpora cannot migrate in place without re-indexing); Agent Retrieval corpora require `COSINE` distance
Agent Retrieval (formerly Vector Search 2.0)	Self-tuning, schema-aware collection store	Storing structured + unstructured **Data Objects** with automatic managed embeddings (`vertex_embedding_config`) or BYO vectors without cluster sizing	Exact brute-force KNN up to ~100k objects; create an ANN **Collection Index** beyond ~100k objects; CMEK (`--kms-key`) must be set at collection creation
Vector Search 1.0	High-scale ScaNN ANN index-as-a-service	100k to billions of BYO dense/sparse vectors at sub-10ms latency with fine-grained ScaNN tree/hashing tuning and `STREAM_UPDATE` or `BATCH_UPDATE` pipelines	`IndexEndpoint` bills **24/7 VM node-hours even at 0 QPS**; `dimensions` and `distanceMeasureType` are immutable; stores IDs + vectors + restricts only (hydrate payload text externally)
AlloyDB AI / Cloud SQL (`pgvector` / ScaNN)	Transactional operational DB vectors	Low-latency hybrid SQL filters (`WHERE tenant_id = $1 AND status = 'OPEN'`) joined directly with live ACID rows (`scann` on AlloyDB; HNSW/IVFFlat on Cloud SQL)	Pin an **explicit embedding model version** in `google_ml_integration` `embedding()` calls so model updates never cause silent vector drift; size connection pools via `MCP Toolbox`
BigQuery (`VECTOR_SEARCH`)	In-warehouse SQL vector search (`IVF` `< 10M` rows; `TREE_AH` up to `200M` rows)	Batch enrichment, analytical SQL joins over petabyte warehouse tables, and offline evaluation pipelines	Hundreds of milliseconds to seconds cold analytical startup latency; wrong fit for sub-50ms online chat QPS

Service	Architecture	Choose when	Key limits & watch-outs
Firestore Vector Search	Serverless operational NoSQL document + flat/KNN vector store	Mobile/web or operational agent documents already stored in Firestore needing integrated metadata + vector lookups	Max **2,048 dimensions** (truncate 3,072d embeddings via `output_dimensionality ⩽ 2048` + L2 re-normalize), max **1,000 results** per query, and **no real-time snapshot listeners** on vector queries
Grounding with Google Search / Maps / Web	Managed live public-index grounding	Fresh public facts, news, places/geospatial context, or third-party enterprise web grounding (Parallel, Exa)	Must render required **Search Suggestions** / attribution chips from `groundingMetadata`

COMMON TRAP: `DOT_PRODUCT` METRIC FAILURE UNDER RAG ENGINE
When backing a **RAG Engine** corpus with an **Agent Retrieval** or Vector Search index, configure the index distance metric as **`COSINE`**. RAG Engine issues cosine-distance queries, so leaving the index at `DOT_PRODUCT` fails at runtime with `FAILED_PRECONDITION`.

Embedding models, normalization & distance metric maths

Model	Modalities & language scope	Max dimensions & input cap	`task_type` & Matryoshka support	Key operational trap
`text-embedding-005` (`GA`)	English text and source code	Up to **768** dims; **2,048-token** max input	Supports **8 `task_type` values** (`RETRIEVAL_DOCUMENT`, `RETRIEVAL_QUERY`, `QUESTION_ANSWERING`, `FACT_VERIFICATION`, `CLASSIFICATION`, `CLUSTERING`, `SEMANTIC_SIMILARITY`, `CODE_RETRIEVAL_QUERY`) and Matryoshka `output_dimensionality`; unit L2-normalized at 768d	Accuracy degrades on non-English corpora; switch to `text-multilingual-embedding-002` or `gemini-embedding-001` for multilingual text
`gemini-embedding-001` (`GA`)	Multilingual text (100+ languages) and code	Up to **3,072** dims; **2,048-token** max input	Asymmetric **`RETRIEVAL_DOCUMENT`** (corpus indexing) vs. **`RETRIEVAL_QUERY`** (user query); Matryoshka output slicing (`768`, `1,536`, `3,072`)	Exceeds Firestore's 2,048-dimension ceiling at default 3,072d; set `output_dimensionality ⩽ 2048` (e.g., `768` or `1536`) and **L2 re-normalize** the truncated slice before indexing
`gemini-embedding-2` (`GA` / rolling multimodal)	Unified omni-modal space: text, images, audio, video, and PDFs	Up to **3,072** dims; **8,192-token** max input	Cross-modal retrieval in one shared vector space; supports managed auto-embeddings in Agent Retrieval (`global`, `us`, `eu`)	Long recordings or PDFs exceeding 8,192 tokens must be segmented into timestamped or page-bounded clips (`start_offset` / `end_offset` metadata per `DataObject`)

- **L2 normalization and distance metric equivalence:** When vectors are unit L2-normalized ($\|u\|_2 = \|v\|_2 = 1$), cosine similarity equals the inner dot product ($u \cdot v$) and produces the exact same monotonic neighbor ordering as squared Euclidean distance ($\|u - v\|_2^2 = 2 - 2(u \cdot v)$). Under unit normalization, **`DOT_PRODUCT_DISTANCE`** is the fastest hardware SIMD execution path in ScaNN.
- **Un-normalized and Matryoshka-sliced vectors:** Truncating a Matryoshka vector via `output_dimensionality` (or importing third-party encoders) breaks unit length ($\|u\|_2 < 1$). Using raw `DOT_PRODUCT_DISTANCE` on un-normalized vectors biases top-`k` toward high-magnitude vectors; either L2 re-normalize vectors before upsert, configure `featureNormType: "UNIT_L2_NORM"` on the Vector Search index, or select `COSINE_DISTANCE`.

Vector Search 1.0 index & endpoint tuning

Knob / configuration	Purpose & mechanics	Architectural guidance & exam traps
`treeAhConfig` vs. `bruteForceConfig`	`treeAhConfig` combines two-level ScaNN Tree partitioning with Asymmetric Hashing for fast ANN; `bruteForceConfig` runs exact exhaustive linear scan	Use `treeAhConfig` for `100k` to billions of vectors at sub-10ms latency; reserve `bruteForceConfig` for small datasets (`< ~50k` vectors) or `100%`-recall ground-truth benchmarking
`leafNodeEmbeddingCount`	Target number of vectors per ScaNN tree leaf partition (default ~ `1000`)	Smaller leaf sizes build finer-grained partitions and raise recall at the cost of higher index build time and memory footprint
`leafNodesToSearchPercent`	Percentage of tree leaf partitions probed per query (e.g., `5` – `10` %)	Primary runtime latency-vs-recall dial; doubling the searched leaf percentage increases vCPU consumption roughly linearly

Knob / configuration	Purpose & mechanics	Architectural guidance & exam traps
approximateNeighborsCount	Pre-reordering candidate pool scored via fast asymmetric hashing before exact full-precision rescoring	Always set strictly greater than requested neighbor_count (> k , e.g., 100 – 200 for top- 10) so quantization noise does not drop true nearest neighbors
restricts , **numeric_restricts** & crowding	Categorical token filters (namespace with allow / deny lists), typed numeric range predicates, and per_crowding_attribute_num_neighbors	Enforce tenant/metadata pre-filtering inside the ANN scan; set **crowding** (e.g., max 2 neighbors per document_id or brand_id) so one verbose document never monopolizes top- k
STREAM_UPDATE vs. **BATCH_UPDATE**	STREAM_UPDATE accepts real-time UpsertDatapoints / deletes within seconds with background compaction; BATCH_UPDATE rebuilds immutable shards from GCS	STREAM_UPDATE carries higher hourly node cost for live freshness; BATCH_UPDATE lowers serving cost when minutes-to-hours refresh lag is acceptable
IndexEndpoint deployment & networking	Hosts deployed index shards on dedicated machine types (minReplicaCount to maxReplicaCount) via public endpoint, VPC Peering, or **Private Service Connect**	**Four classic traps:** (1) IndexEndpoint bills **24/7 VM node-hours even at 0 QPS** (minReplicaCount ⩾ 1); (2) dimensions and distanceMeasureType are **immutable** after creation; (3) prefer **PSC (pscAutomationConfigs)** over VPC Peering to avoid /26+ subnet CIDR exhaustion; (4) stores IDs + vectors + restricts only (fetch chunk text from GCS, Firestore, AlloyDB, or BigQuery)

Vector Search 1.0 vs. Agent Retrieval vs. RAG Engine

When the matrix above leaves two vector options close, compare them on these dimensions.

- **Object model:** Vector Search 1.0 holds datapoints (an ID, the embedding, optional filter metadata) in an index deployed to an endpoint, so the source text or record lives in a separate store. Agent Retrieval keeps JSON **Data Objects** (payload fields plus one or more vector fields) inside a schema-validated **Collection**. RAG Engine exposes RagCorpus and RagFile resources and hands the vectors to a backing vector database.
- **Embedding pipeline:** Vector Search 1.0 is bring-your-own: you compute dense (and, for hybrid, sparse) vectors before upserting. Agent Retrieval accepts your own vectors or fills a vector field automatically from a vertex_embedding_config (model ID, text template built from payload fields, task type) whenever a Data Object arrives without that vector. RAG Engine parses, chunks, and embeds files for you at import time (ImportRagFiles with Document AI Layout Parser supports up to **20 MB / 500 pages** per PDF).
- **Hybrid search and reranking:** Vector Search 1.0 fuses a dense and a sparse index with Reciprocal Rank Fusion (RRF), weighted by rrf_ranking_alpha , and you build the sparse vectors (for example SPLADE or TF-IDF) yourself. Agent Retrieval combines semantic search with built-in keyword text search through RRF and can rerank results with the Ranking API. RAG Engine sets reranking in RagRetrievalConfig (rank_service for a semantic ranker model or llm_ranker for a Gemini model); its hybrid_search.alpha setting applies only when Weaviate is the backing store.
- **Agent and MCP integration:** Vector Search 1.0 needs a custom function tool or your own MCP server. Agent Retrieval has a Google-managed stateless remote MCP server (Preview, read-only tools; see the MCP catalogue below). RAG Engine attaches a corpus to Gemini requests as a vertex_rag_store retrieval tool.
- **Scaling and cost:** Vector Search 1.0 bills deployed index endpoints by node-hour, even when idle. Agent Retrieval serves brute-force KNN with no index; add a Collection Index once a collection nears 100,000 Data Objects or KNN latency is too high for your query rate. Its pricing is usage-based for small workloads or resource-based for tuned performance.
- **CMEK on Agent Retrieval:** set the Cloud KMS key (same region as the collection) when you create the Collection (--kms-key). An existing collection cannot be switched to CMEK, so export its Data Objects to Cloud Storage and import them into a new CMEK collection.
- **RAG Engine deployment modes (one-way lock-in):** one RAG API routes to whichever mode the project has active; corpora created under the other mode stay stored but hidden until you switch back, and modes **cannot be migrated in place** without re-indexing.
- **Serverless mode** needs no tier management, autoscales, and by default provisions an Agent Retrieval collection in your project as the vector database (Pinecone and Weaviate are alternatives; RagManagedDb is not offered). Storage is not isolated and **CMEK is not supported**.
- **Spanner mode** is the default when no mode is chosen. It runs a dedicated Spanner instance at the **Basic** tier (default, fixed low compute) or the **Scaled** tier (autoscaling, production data volumes); the **Unprovisioned** tier deletes the instance and its data. It isolates storage and performance, **supports CMEK**, and offers RagManagedDb (default), Agent Retrieval, Vector Search 1.0, Pinecone, and Weaviate as vector stores.
- **Both modes** support VPC Service Controls.
- **When to choose: Vector Search 1.0** for existing pipelines with your own embedders and tuned, always-on ANN endpoints; **Agent Retrieval** for a new custom agent store that keeps payload and vectors together with auto-embeddings, keyword plus semantic search, and an MCP server; **RAG Engine Serverless mode** for the fastest managed document RAG when CMEK is not required; **RAG Engine Spanner mode** (or a CMEK-enabled Agent Retrieval collection) when policy demands CMEK or isolated storage and performance.
- **Anti-pattern (Serverless mode under a CMEK mandate):** the mode has no CMEK option, so a key-management policy rules it out. Pick Spanner mode before importing data, because corpora do not carry over between modes.

Reranking, hybrid RRF & citation verification

Mechanism	Latency & candidate scale	How it works	Key trap & operational guidance
Agent Platform Ranking API (`semantic-ranker-*` , e.g., `semantic-ranker-default@latest`)	**`< 100 ms`** over up to ~200 candidate chunks	Stateless cross-encoder jointly attends to the query and each candidate passage, narrowing noisy recall down to high-precision top- `k`	Requires **`discoveryengine.googleapis.com`** enabled (`roles/discoveryengine.viewer` / ranking permission) even when called from custom RAG pipelines, or requests fail with **`403 PERMISSION_DENIED`**
Agent Search built-in ranking (`BoostSpec`)	Zero extra network hop (inline in `discoveryengine`)	Combines managed semantic + keyword retrieval with declarative condition-based score multipliers (freshness decay, authoritative source boosts)	Applies only inside Agent Search queries; configure bounded positive/negative boost weights so boosted low-relevance docs do not crowd out exact answers
LLM reranker (`llm_ranker` in RAG Engine / Gemini listwise reranking)	**`1–2+ s`** **TTFT**; high input-token cost	Prompts a generative Gemini model to reason across candidate passages for complex multi-constraint relevance	Prohibitive latency and token spend on high-QPS chat or large `top_k` pools; reserve for low-QPS research synthesis or cascade after cross-encoder pruning
Hybrid sparse + dense + RRF (`rrf_ranking_alpha`)	Single-digit ms fusion overhead	Reciprocal Rank Fusion merges lexical sparse rank (exact SKUs, part numbers, error codes, acronyms) with dense semantic rank on a rank-order scale	`rrf_ranking_alpha` spans `[0.0, 1.0]` : **`0.5`** weights dense and sparse rankings equally, whereas **`1.0`** **weights dense vectors `100%` and silently ignores sparse keyword matches** (`0.0` uses sparse keywords only)

- **Check Grounding API:** Evaluates each claim in a candidate answer against supplied source facts, returning a support score and per-sentence citations to block ungrounded hallucinations.
- **Grounding metadata:** Inspect `groundingChunks` , `groundingSupports` , `webSearchQueries` , and `searchEntryPoint` to render verifiable inline citations.
- **Agent Search grounding limits:** Attach up to **10 data stores** per request; callers require `roles/discoveryengine.viewer` (and user identity propagation via Workforce Identity or OAuth when enforcing source document ACLs).

Google Cloud MCP servers & enterprise integration layers

Google Cloud provides both **first-party remote `StreamableHTTP` MCP servers** (`https://<service>.googleapis.com/mcp` or regional `.rep.googleapis.com/mcp` , auto-registered in `global` **Agent Registry** when the underlying API is enabled, supporting stateless MCP spec **`2026-07-28`** without `initialize` or `Mcp-Session-Id` state) and four complementary enterprise tool integration layers for databases, SaaS/ERPs, OpenAPI microservices, and custom logic.

First-party Google Cloud & database MCP servers catalogue

MCP server / project	Capabilities	Hosting & discovery mode	Required IAM / auth	Operational traps
`bigquery.googleapis.com/mcp`	Dataset/table discovery, schema inspection, dry-run validation, and SQL execution	Google-hosted remote `StreamableHTTP` (`global` Agent Registry)	`roles/mcp.toolUser` + `roles/bigquery.jobUser` + `roles/bigquery.dataViewer` (row/column ACLs apply)	Unbounded analytical scans spike slot/byte bills; enforce `maximumBytesBilled` or expose curated authorized views
`container.googleapis.com/mcp`	GKE cluster, node-pool, workload, and pod diagnostics	Google-hosted remote `StreamableHTTP` (`global` Agent Registry)	`roles/mcp.toolUser` + `roles/container.viewer` (or `roles/container.developer` for mutations)	Scope troubleshooting agents to `container.viewer` so prompt injection cannot mutate live cluster specs
`compute.googleapis.com/mcp`	Compute Engine VM inventory, metadata inspection, and instance lifecycle actions	Google-hosted remote `StreamableHTTP` (`global` Agent Registry)	`roles/mcp.toolUser` + `roles/compute.viewer` / `roles/compute.instanceAdmin.v1`	Gate stop/reset/delete tools with IAM UAP `iap.googleapis.com/mcp.tool.isReadOnly` CEL predicates or HITL approval
`run.googleapis.com/mcp`	Inspect and deploy Cloud Run services, revisions, and traffic splits	Google-hosted remote `StreamableHTTP` (`global` Agent Registry)	`roles/mcp.toolUser` + `roles/run.viewer` / `roles/run.developer`	Manages Cloud Run itself; *custom* MCP servers hosted *on* Cloud Run deploy as Services with `--functional-type=mcp-server` + `--no-allow-unauthenticated` + IAP
`logging.googleapis.com/mcp` (plus Monitoring / Trace MCP)	Query Cloud Logging entries, metrics, and trace spans during incident triage	Google-hosted remote `StreamableHTTP` (`global` Agent Registry)	`roles/mcp.toolUser` + `roles/logging.viewer` / `roles/logging.viewAccessor`	Require bounded `timestamp` filters on every query to prevent full-bucket scans and context-window overflow

MCP server / project	Capabilities	Hosting & discovery mode	Required IAM / auth	Operational traps
Cloud SQL / AlloyDB / Spanner / Firestore / Cloud Storage MCP	Instance management, `execute_sql` / `execute_sql_readonly`, NoSQL ops, and `storage.googleapis.com/storage/mcp` object R/W	Google-hosted remote `StreamableHTTP` (`global` Agent Registry; AlloyDB regional Preview)	`roles/mcp.toolUser` + underlying DB/bucket IAM roles	Prefer `execute_sql_readonly` or parameterized **MCP Toolbox** tools over unrestricted SQL write endpoints
`developerknowledge.googleapis.com/mcp` (plus Maps Grounding Lite)	Search public English Google developer documentation (and live place/geospatial facts)	Google-hosted non-Cloud remote `StreamableHTTP`	API key or OAuth (no `roles/mcp.toolUser`; not governed by Cloud IAM policies)	**VPC-SC trap:** indexes public English docs only and does **not** support the `restricted.googleapis.com` VIP inside VPC-SC; route egress via **`private.googleapis.com`**
Agent Retrieval MCP (`vectorsearch.googleapis.com/mcp`, `Preview`)	Nine read-only tools for collection/index inspection plus vector, semantic, hybrid, and keyword search (no write tools)	Google-hosted stateless `Streamable HTTP` (MCP spec `2026-07-28`)	`roles/mcp.toolUser` + `roles/vectorsearch.viewer` + `roles/agentregistry.viewer`	Client must attach target project/routing headers on every `tools/call`; strictly read-only
MCP Toolbox for Databases (`googleapis/genai-toolbox` / `mcp-toolbox`, `GA`)	Prebuilt (`--prebuilt=<db>`) and declarative `tools.yaml` parameterized SQL/Cypher/NoSQL with connection pooling and OTel across **30+ DBs**	Self-hosted on Cloud Run / GKE (`StreamableHTTP`) or local `stdio` (`Toolbox Toolset` / `Mcp Toolset`)	IAM DB auth + Cloud Run Invoker / IAP + optional per-tool OAuth/OIDC	Never expose a generic raw `run_any_sql` string parameter; bind typed parameters in `tools.yaml` to prevent SQL injection

> **NOTE**
> Additional Google-managed remote MCP servers also cover Bigtable, Memorystore, Pub/Sub, Apigee API Hub, and Knowledge Catalog. Verify per-service launch stage and supported regions in the Google Cloud MCP documentation before production rollout.

Four-way enterprise tool integration matrix

When connecting beyond native first-party Google Cloud MCP endpoints, select the integration layer matching your backend protocol and governance boundary:

Integration layer	ADK toolset class	Target systems & strengths	Latency, auth & operational traps
Integration Connectors + Application Integration (`GA`)	`Application IntegrationToolset`	**100+ SaaS, ERP, and CRM platforms** (Salesforce, ServiceNow, SAP, Workday, Jira) with visual schema mapping, payload transformation, and **Auth Manager 3LO** user-delegated consent	Higher per-invocation hop latency and connector execution cost than direct database or REST calls
MCP Toolbox for Databases (`GA`)	`ToolboxToolset` / `Mcp Toolset`	**Sub-50ms pooled database queries** across 30+ relational, graph, and analytical engines (AlloyDB, Cloud SQL, Spanner, BigQuery, Postgres, Neo4j, Snowflake) via declarative `tools.yaml` and OTel tracing	Database-centric only; provides no prebuilt SaaS/ERP business-object adapters
Apigee + API Hub (`GA`)	`ApiHubToolset` (or `OpenAPIToolset` for small direct specs)	**Cataloged OpenAPI REST microservices** fronted by enterprise rate limits, LLM token quotas, mTLS, semantic caching, and inline Model Armor	**Tool-bloat trap:** always filter specific OpenAPI operations rather than importing 200 raw REST endpoints into one agent's prompt context
Custom Remote MCP on Cloud Run (`GA`)	`McpToolset` (`Streamable HTTPConnection Params`)	**Bespoke polyglot domain logic** and multi-API composition deployed with `--functional-type=mcp-server`, `--no-allow-unauthenticated`, and **Cloud IAP + Agent Gateway**	**Header-split trap:** Cloud Run consumes `Authorization` to verify the caller's OIDC token; pass Cloud Run OIDC in **`X-Serverless-Authorization`** when forwarding a user OAuth token in `Authorization`, or delegate token brokering to **Agent Gateway + Auth Manager**

- **Transport & principal rule:** Use `stdio` only for local IDE/CLI subprocesses; production remote tools require **`StreamableHTTP`** over HTTPS authenticated via cryptographic **SPIFFE Agent Identity** (for agent-scoped cloud/DB actions) or **Auth Manager 3LO OAuth** (when downstream SaaS/workspace tools must enforce per-user ACLs).

Agent Registry end-to-end governance workflow

Agent Registry serves as the centralized control-plane catalog for agents (`AgentCard`), MCP servers (`McpServer`), and standalone skills across your organization:

Governance dimension	Platform behavior & workflow	Key IAM roles & exam traps
Auto-discovery vs. manual registration	**Auto-discovered:** Agent Runtime (`reasoningEngines`), Cloud Run Services deployed with `--functional-type=mcp-server` (Cloud Run Jobs cannot be MCP servers), GKE workloads labeled `registry.gke.io/functional-type: "MCP_SERVER"` with `modelcontextprotocol.info/*` annotations, and 1P Google Cloud MCP servers (registered only in `--location=global`, with regional URLs inside `interfaces`). **Manual (`gcloud agent-registry`):** external, on-prem, cross-project (outside management boundary), or regional-residency endpoints; MCP registration requires uploading a **`toolspec.json`** (`⩽ 10 KB`, declaring tool schemas plus `readOnlyHint` / `destructiveHint`) because the registry does not crawl external servers	Querying 1P Google Cloud MCP servers with `--location=us-central1` returns `NOT_FOUND` (use `--location=global`). Manual registration requires `roles/agentregistry.editor` or `roles/agentregistry.admin`
Discovery vs. invocation authorization	Catalog visibility and live network invocation are strictly separated	**`roles/agentregistry.viewer`** grants **catalog discovery and metadata read only**; it never permits calling the live A2A or MCP endpoint. Runtime invocation requires **`roles/iap.egressor`** (evaluated with IAM UAP CEL predicates at Agent Gateway / IAP) plus target endpoint IAM (`roles/run.invoker` or `roles/mcp.toolUser`)
Gemini Enterprise Agent Gallery alignment	Match the Agent Registry location (`global`, `us`, `eu`, or specific region) to your **Gemini Enterprise** app location so published enterprise agents populate the workforce **Agent Gallery**	Business users discover governed organizational agents via *Agent Gallery → Request access → "From your organization"* and invoke them inline with `@agent_name`; mismatched registry/app regions hide the agent from the gallery
CI/CD gating & version cutovers	Built-in pre-registration CI approval workflows and a formal `DEPRECATED` lifecycle flag on MCP/agent endpoints are **not publicly documented** in Agent Registry itself	Gate catalog writes by restricting `roles/agentregistry.admin` / `editor` to your **Cloud Build** CI service account (unlocked only after `agents-cli eval` regression gates pass), and perform zero-downtime version cutovers or retirements via semantic **`AgentCard.version`** fields paired with **Agent Gateway** routing
ADK runtime resolution & `.clone()`	Resolve `AgentRegistry.get_remote_a2a_agent()` and `get_mcp_toolset()` **once at process startup** rather than inside per-turn request handlers	Avoids per-turn control-plane discovery latency and quota burn; always call **`.clone()`** when attaching the same resolved remote A2A agent across multiple ADK workflow branches so parent-pointer tree state does not collide

Skill Registry lifecycle

Skill Registry (Preview) is the Agent Platform repository for agent skills in the open Agent Skills layout: a `SKILL.md` file plus optional `scripts/`, `references/`, and `assets/` folders. Agents load a skill only when the task calls for it, instead of carrying every instruction in the prompt. **Agent Registry** can also hold the same packages as *standalone skills*, alongside agents and MCP servers. That second path has its own API, roles, and tighter payload limits, so keep the two apart.

Stage	Skill Registry (Agent Platform API)	Agent Registry standalone skills
Author	`SKILL.md` front matter: `name` (required, max 64 characters, lowercase letters, digits, and hyphens, no leading or trailing hyphen), `description` (required, max 1,024 characters), optional `license` (max 1,024). Instructions body max 500,000 characters. The skill ID (1 to 63 characters, starts with a letter) stays reserved even after deletion, and the `gcp-` prefix is reserved for built-in skills	Same `SKILL.md`, placed at the zip root with valid front matter per the Agent Skills specification
Publish and validate	`CreateSkill` long-running operation (REST `v1beta1`, or `client.skills.create` in the `agentplatform` SDK) with a base64 zip. Asynchronous validation rejects a zip over **10 MB**, over 500 MB unzipped, over **10,000** entries, nested deeper than 8 folders, compression ratio over 100, any **symbolic link**, duplicate names, `..` or absolute paths, or a missing `SKILL.md`	`gcloud alpha agent-registry skills create` with `--payload` (local zip) or `--gcs-source-uri`. Limits: **500 KB** compressed, 10 MB uncompressed, 1 MB per file, 8 folder levels. The ID gets a `private-` prefix, and a Cloud Storage import needs the Agent Registry service agent to hold `storage.objects.get`
Revisions	Every create or update stores an immutable **skill revision**; `GetSkill` returns the latest one, and `ListSkillRevisions` / `GetSkillRevision` expose history	New skills start `DISABLED`; set the default revision and switch the target state to Active before agents can discover it
Discover	`ListSkills` returns name, display name, and description; `RetrieveSkills` runs semantic search over display name and description. The built-in **`gcp-skill-registry`** skill (Google-managed) lets an agent create, search, and manage skills itself; it appears after the first API call	Agent Registry search across agents, MCP servers, and skills
Authorize	Inherits project IAM: `roles/aiplatform.user` to manage skills, `roles/aiplatform.viewer` for read-only access	`roles/agentregistry.viewer` (discover and read), `roles/agentregistry.user` (read plus create, update, delete skills and revisions), `roles/agentregistry.editor` (skills plus agents, MCP servers, endpoints), `roles/agentregistry.admin` (everything plus bindings and skill IAM policies)
Govern at runtime	**Semantic governance policies** (Preview) intercept the ADK skill tools and judge each call against natural language constraints (NLCs)	Same policies apply, because loading a skill is itself a tool call

Progressive disclosure at runtime runs through four ADK skill tools, and each one is a point where a policy can return a verdict:

1. **`list_skills`** shows the model only the catalog of skill names and descriptions.
2. **`load_skill`** pulls one skill's full instructions and tools into context.
3. **`load_skill_resource`** fetches a referenced file or asset on demand.
4. **`run_skill_script`** executes the skill's bundled code.

```
---
name: invoice-triage
description: Sorts supplier invoices by urgency and flags missing
PO numbers. Use when a user shares or asks about an invoice.
license: Apache-2.0
---
# Invoice triage
1. Extract the PO number, amount, and due date.
2. Run `scripts/check_po.py` to confirm the PO exists before
routing.
```

- **Compliance (Skill Registry):** Access Transparency and US/EU data residency are supported; CMEK, VPC Service Controls, and HIPAA are not. Regions: `us-central1`, `europe-west4`, `us-east5`.
- **Tainted-session control:** an NLC can deny `load_skill` for any skill that sends messages or writes externally once the history contains web, email, or uploaded content, so the outbound tools never enter context.
- **Supply-chain control:** an NLC can inspect the arguments of an install tool triggered by skill instructions and deny packages outside an allowlist.
- **Caveat:** semantic governance uses an LLM, so verdicts are probabilistic. Keep IAM and network egress controls as the deterministic layer.
- **Anti-pattern (registry admin roles on agents):** granting `roles/agentregistry.admin` or `roles/agentregistry.editor` to an agent identity lets a compromised agent alter tool definitions or enroll a malicious agent. On the Agent Registry standalone-skills path, give consuming agents `roles/agentregistry.viewer` and publishers or CI pipelines `roles/agentregistry.user`; on the Skill Registry path, use `roles/aiplatform.viewer` for consumers and `roles/aiplatform.user` for publishers.
- **Anti-pattern (IAM as the only skill control):** IAM decides who may publish or read a package, but an authorized agent can still be steered by injected content into loading a powerful skill. Pair IAM with semantic governance policies.

07. Caching & External State

Backing store	Primary role in an agentic architecture
Gemini Context Caching	**Implicit caching** (on by default, zero storage fee; **the only context cache mode on `gemini-4-argon`**) and **Explicit caching** (`cachedContents` with TTL + storage fee on supported Gemini 2.5/3.x models) for massive repeated system instructions, tool schemas, or static corpora
Memorystore for Redis / Valkey	Sub-millisecond (`< 1 ms`) hot state, sliding-window rate-limit/budget counters, Pub/Sub `messageId` idempotency locks, and **semantic response caching** via Redis HNSW/FLAT vector search

Backing store	Primary role in an agentic architecture
Firestore	Serverless multi-region (`99.999%`) strong-consistency document store for real-time UI state sync, HITL approval queues, and subcollection turn logs
Cloud SQL / AlloyDB	ACID relational workflow checkpoints; backs ADK's `DatabaseSessionService` (with PgBouncer / AlloyDB connection pooling) when custom SQL querying over sessions is required
Agent Platform Sessions	Zero-ops managed event & scratchpad persistence (`VertexAiSessionService` , default 365-day TTL, regional CMEK) for ADK and Agent Runtime
Cloud Storage (`GcsArtifactService`)	Versioned binary blob store (`{app}/{user}/{session}/{filename}/{version}` and `user:` scope) for large PDFs, audio, images, and sandbox outputs outside `session.state`

Backing store	Primary role in an agentic architecture
BigQuery	Long-term telemetry sink, offline evaluation datasets, failure-cluster analytics, and FinOps token attribution; **never hot-path turn state**

> **COMMON TRAP: CROSS-TENANT SEMANTIC CACHE LEAKAGE**
> If User A asks *"What is the Q4 executive bonus pool?"* and the semantic cache keys only on the query embedding, User B asking a similar question can receive User A's restricted answer. **Always partition semantic cache keys by tenant/role/ACL scope** (or bypass semantic caching entirely for user-authorized data).

Choosing data services by agent data need

Split agent data by access pattern rather than putting everything in one database. Each row names the default pick and the usual fallback.

Data need	Default service	Common alternative	Why the default fits
Conversation sessions (events + state)	**Agent Platform Sessions** via `VertexAiSessionService`	Cloud SQL or AlloyDB via `DatabaseSessionService`	No schema to own; same store reachable from Agent Runtime, Cloud Run, GKE, or a laptop; feeds Memory Bank generation
Long-term user memory	**Memory Bank** (`VertexAiMemoryBankService` , Section 04)	Your own facts table with vectors in AlloyDB/Cloud SQL (`pgvector`) or Firestore vector search	Managed LLM extraction, consolidation, identity-scoped similarity retrieval, TTL, and revisions
Semantic cache	**Memorystore for Redis** vector search (Redis 7.2; HNSW approximate or FLAT exact)	AlloyDB vector index	Sub-millisecond in-memory lookups with native key expiry, so repeated or paraphrased prompts skip the model call
Rate-limit, budget & idempotency locks	**Memorystore** (Redis/Valkey atomic increments & `SET NX` TTL keys)	Firestore transactional documents (higher latency, ~1 write/s/doc limit)	Sub-millisecond atomic operations suit per-user throttling and Pub/Sub `messageId` deduplication on the hot path
Transactional data that tools read and write	**Cloud SQL** or **AlloyDB**	**Spanner** (global scale) or **Firestore** (document model)	ACID transactions, joins, and vector columns next to live rows; reachable through managed MCP servers or MCP Toolbox
Analytics and agent telemetry	**BigQuery**	Cloud Logging Log Analytics	Serverless SQL over prompt/response logs, traces, eval results, and token cost; never the per-turn state store
Documents, media, and artifacts	**Cloud Storage** (ADK `GcsArtifactService` for agent outputs)	BigQuery object tables (SQL metadata over Cloud Storage objects)	Durable object storage (including Zonal Rapid buckets) that also feeds Agent Search, RAG Engine, and Agent Retrieval ingestion
Vector embeddings	**Agent Retrieval** or **Vector Search 1.0** (Section 06 matrix)	AlloyDB/Cloud SQL `pgvector` , Firestore vector search, BigQuery `VECTOR_SEARCH`	Purpose-built ANN and hybrid search; pick the operational database when vectors must be filtered with live transactional rows

ADK state services (`SessionService` , `MemoryService` , `ArtifactService`) & backing stores

ADK interface / backing store	Persistence scope & features	Production limits & architectural traps
`VertexAiSessionService` (Agent Platform Sessions, `GA`)	Managed chronological `Event` log + `session.state` (`--session_service_uri agentengine://<id>`); **365-day default TTL** (configurable), regional data residency, and **CMEK**	Default on Agent Runtime and callable from Cloud Run or GKE; never stuff OAuth tokens, secrets, unredacted PII, or multi-MB blobs into `session.state`
`VertexAiMemoryBankService` (Agent Platform Memory Bank, `GA`)	Asynchronous `GenerateMemories` consolidation (`add_events_to_memory` in `after_agent_callback`) and explicit `CreateMemory` tool writes with revisions, Memory Topics, TTL, and CMEK	Always scope by `{user_id, app_name}` to prevent cross-tenant leakage; screen external inputs with Model Armor to block memory poisoning
`DatabaseSessionService` (AlloyDB / Cloud SQL / Spanner, `GA`)	Relational ACID persistence in customer-operated PostgreSQL/MySQL/Spanner (or local SQLite for dev) when SQL reporting, row-level security, or custom VPC residency is required	**Connection-exhaustion trap:** autoscaling Cloud Run containers quickly exhaust PostgreSQL `max_connections` unless fronted by **PgBouncer**, **AlloyDB Auth Proxy pooling**, or bounded SQLAlchemy pools

ADK interface / backing store	Persistence scope & features	Production limits & architectural traps
`InMemorySessionService` (Process RAM)	Ephemeral in-process dictionary for unit tests and local `adk web`	Wiped on container restart, scale-to-zero, or horizontal load-balancing across replicas; never use in production
Firestore (Custom / serverless state, `GA`)	`99.999%` multi-region strong consistency, real-time client listeners for UI streaming/HITL queues, and automatic document TTL policies	Hard **1 MiB max document size** and **~1 sustained write/sec per document** hotspot limit; **store turn events in subcollections** (`sessions/{id}/events/{eventId}`) rather than appending unbounded arrays inside one root document
Memorystore for Redis / Valkey (`GA`)	**`< 1 ms`** HNSW/FLAT vector semantic cache, sliding-window RPM/TPM rate limiters, and Pub/Sub `message Id` idempotency keys; scales up to **250 nodes / 25+ TB**	**Compliance audit trap:** relies on **asynchronous replication** and optional mutually exclusive **RDB snapshots or AOF (`appendfsync`) persistence** (legacy standalone Redis tiers are RDB-only); replica failover or key eviction can still lose recent in-flight writes, so never use Memorystore as the sole system of record or immutable audit ledger
`GcsArtifactService` (Cloud Storage, `GA`)	Versioned binary object hierarchy (`{app}/{user}/{session}/{filename}/{version}`), `user:` prefix for cross-session persistence, and **Zonal Rapid Storage buckets** for low-latency streaming	**Initialization trap:** leaving `Runner(artifact_service=None)` raises **`ValueError`** at runtime whenever a tool calls `ctx.save_artifact()`. Pass `gs://` URIs across A2A rather than multi-MB base64 A2A `Part` blobs that choke Agent Gateway `CONTENT_AUTHZ`

Service trade-offs

Service	Latency profile	Consistency	Cost model	Ops burden
Agent Platform Sessions + Memory Bank	API calls on the hot path; memory generation runs asynchronously	Session reads/writes are immediate; memories converge after background consolidation	Managed usage-based pricing	Lowest
Firestore	Low (serverless document reads/writes)	Strong (`99.999%` multi-region), with ACID transactions	Per operation + storage	Very low (scales to zero)
Memorystore	Sub-millisecond (`< 1 ms`) in-memory	Strong on the primary; replicas are asynchronous (mutually exclusive RDB or AOF on Cluster/Valkey; RDB-only on legacy standalone Redis)	Provisioned capacity per hour	Low to medium (capacity sizing)
Cloud SQL	Low within the VPC	Strong relational ACID	Provisioned vCPU/RAM + storage	Medium (instances, pooling, indexes)
AlloyDB	Low, with `scann` vector and columnar analytical acceleration	Strong relational ACID	Provisioned primary and read pool nodes	Low to medium
BigQuery	Hundreds of ms to seconds per query	Strong after commit	Per data scanned or reserved slots + storage	Very low
Cloud Storage	Tens of ms to first byte (lower on Zonal Rapid buckets)	Strong read-after-write	Per GB-month by class + operations + egress	Lowest
Agent Retrieval / Vector Search	Single-digit to tens of ms ANN	Near real time for streaming updates	Usage-based (Agent Retrieval) or 24/7 node-hours (Vector Search 1.0)	Low

> **NOTE**
> Latency labels are directional. Benchmark with your own payload sizes and regions before committing to an SLO.

Communication protocols & event-driven triggers

Match the transport or event broker to directionality, barge-in latency, delivery guarantees, and duration ceilings:

Protocol or trigger	Direction & concurrency model	Timeout / duration ceiling	Primary fit & operational traps
SSE (`Server-Sent Events`) (`StreamingMode.SSE`, A2A `/message:stream`)	Unidirectional `Server → Client` stream over HTTP/1.1 or HTTP/2 for incremental text tokens, tool status, and chunked A2A artifacts	Up to **60 minutes** on Cloud Run / HTTP gateways	Ideal for chat UIs and short-to-medium A2A `SendStreaming Message` progress (`SubscribeToTask` on reconnect); **cannot** support real-time voice barge-in (half-duplex) or multi-hour background jobs

Protocol or trigger	Direction & concurrency model	Timeout / duration ceiling	Primary fit & operational traps
MCP `StreamableHTTP` (`2026-07-28` stateless spec)	Stateless HTTP `POST` per request (`Mcp-Method` , `Mcp-Name` , `Mcp-Param-Region` headers) replying with JSON or a per-call SSE stream	Request-scoped tool RPC (seconds to minutes)	Standard remote MCP transport scaling horizontally behind load balancers with **no sticky `Mcp-Session-Id`** ; legacy standalone `HTTP+SSE` dual-endpoint transport is deprecated
Bidirectional Streaming (`Gemini Live API`) (`gemini-3.8-live` over WebSockets / gRPC)	Full-duplex audio + video frame streaming (`runner.run_live`) with server-side VAD **barge-in** (`event.interrupted == True`)	Stateful streaming connection; persist **session resumption handles** to reconnect cleanly across WebSocket drops	Mandatory for sub-second voice/video front-doors on Agent Runtime or Cloud Run; **`gemini-4-argon` does not support Live WebSockets**, so delegate deep reasoning from `3.8-live` via `AgentTool`
Pub/Sub Push & Eventarc (`--trigger_sources pubsub,eventarc` → Cloud Run or Agent Runtime)	Decoupled 1-to-N event fan-out with **at-least-once delivery**; tune Cloud Run `container_concurrency` (e.g., multiples of 9 such as `36` for async I/O-bound agent loops)	**600s (10-minute)** max Pub/Sub ack deadline (Cloud Run HTTP timeout up to 60 min)	**Duplicate-mutation trap:** slow multi-step turns exceeding the 600s ack deadline trigger redelivery and duplicate tool side effects. Enforce atomic **`messageId` idempotency checks** in Memorystore/Spanner/Firestore and attach a **Dead-Letter Topic (`maxDeliveryAttempts ⩾ 5`)**
Cloud Tasks	Point-to-point queue with explicit **`maxDispatchesPerSecond`** , `max ConcurrentDispatches` , `scheduleTime` delayed execution, and OIDC token injection	Up to **30 minutes** per HTTP target attempt	Best for **rate-limiting / shock-absorbing** agent bursts so fragile legacy ERPs and project Gemini TPM/RPM quotas are never overwhelmed; **no 1-to-N pub-sub fan-out**
Cloud Workflows & Cloud Run Jobs	Deterministic serverless saga state machine (**Workflows**, `$0` idle wait + HTTP callbacks) vs. non-HTTP batch shards (**Cloud Run Jobs**, up to `10,000` parallel tasks)	Up to **1 year** (**Workflows**) / **24 hours** per shard (**Cloud Run Jobs**)	Use **Workflows** for multi-day human approval sleeps and deterministic compensation sagas invoking ADK / Agent Runtime for reasoning sub-steps; use **Cloud Run Jobs** for offline batch enrichment or eval suites (exposes no HTTP/SSE endpoint)

Reference architectures

Component	Customer support agent	Data analysis agent	High-QPS public FAQ agent	Regulated-industry agent
Runtime	Agent Runtime	Agent Runtime	Cloud Run (autoscaled)	Agent Runtime (`AgentConnectivity Template`) + private GKE GPU / Model Garden `PrivateEndpoint` in VPC-SC
Sessions	`VertexAiSessionService`	`VertexAiSessionService`	Firestore subcollections or Memorystore (short-lived state)	`DatabaseSessionService` on AlloyDB/Spanner with PgBouncer, CMEK, and PSC
Memory	Memory Bank (device history, preferences)	Memory Bank (analyst preferences)	None (anonymous traffic)	Memory Bank with strict topics, SDP surrogates, and TTL, or none
Knowledge and vectors	Policy PDFs in Cloud Storage indexed by Agent Search or Agent Retrieval	BigQuery tables through `bigquery.googleapis.com/mcp`	Agent Retrieval behind a Memorystore semantic cache	AlloyDB `scann` vectors with row-level security next to transactional rows
Tools and transactions	Order/refund lookups via `ToolboxToolset` + CRM via `ApplicationIntegration Toolset`	Code Execution sandbox for Python; charts saved with `GcsArtifactService`	Memorystore sliding-window rate limits per IP or API key	Custom Cloud Run MCP (`--functional-type=mcp-server`) behind Cloud IAP + Agent Gateway
Telemetry	Cloud Trace + BigQuery eval sink	BigQuery cost and eval analytics	Cloud Monitoring cache hit rate + BigQuery logs	Dual-attribution Audit Logs to BigQuery; locked Cloud Storage retention bucket

COMMON TRAP: MEMORYSTORE AS THE SYSTEM OF RECORD.
Because Memorystore uses asynchronous replication (with optional mutually exclusive RDB snapshots or AOF on Redis Cluster/Valkey, and RDB-only on legacy standalone Redis), replica failover or key eviction can still drop recent in-flight writes. Keep authoritative orders, approvals, and audit records in Cloud SQL, AlloyDB, Spanner, or Firestore, and treat Memorystore entries as rebuildable.

08. Optimize Part 1: Agent Evaluation, Simulation & Golden Flywheels

Evaluating non-deterministic, multi-turn agents requires grading **both the final outcome and the intermediate decision trajectory** (`client.evals.evaluate()` / `agents-cli eval`).

Metric hierarchy

Metric family	How it grades	Built-in examples & exam discriminators
Adaptive rubrics (recommended default)	An autorater dynamically generates prompt-specific pass/fail testable rubrics for each eval item, then grades the response against them	`GENERAL_QUALITY` adapts automatically across diverse open-ended prompts without brittle static checklists
Static rubrics & custom LLM judges	Applies fixed organization-wide pass/fail criteria or custom `types.LLMMetric` (`prompt_template` + `result_parsing_function`)	Brand tone, regulatory disclaimers, domain rubrics (**waste trap:** never use `LLMMetric` for deterministic schema, regex, or math checks that belong in `CustomMetric`)
Agent outcome, grounding & pairwise metrics	Specialized autoraters for task quality, context faithfulness, and head-to-head model comparison	`FINAL_RESPONSE_QUALITY` , `TOOL_USE_QUALITY` , **`GROUNDING`** (entailment against retrieved chunks; **passes even if retrieved docs are outdated**), **`HALLUCINATION`** (`supported` / `unsupported` / `contradictory` / `disputed` span labels; **missing tool-output span causes false-positive hallucination flags**), and **`PairwiseMetric`** (swaps A/B candidate order automatically to cancel LLM judge position bias)
Trajectory metrics	Deterministic comparison between actual tool-call sequence/arguments and a golden reference trajectory	`trajectory_exact_match` (strict compliance pipelines), `trajectory_in_order_match` (required steps in order, extra read steps allowed), `trajectory_any_order_match` (parallel gathering), `trajectory_precision` / `trajectory_recall`
Computation / code metrics (`CustomMetric` / `CodeExecutionMetric`)	Deterministic Python/programmatic graders (sub-millisecond, zero LLM judge cost, 100% reproducible)	JSON Schema validation, regex/SQL AST checks, exact numeric tolerances, latency/token budget assertions, ROUGE/BLEU

2-Step Agent Simulation & Environment Simulation

Static single-turn test prompts miss multi-turn clarification loops, adversarial drift, and tool outages. Agent Platform provides two simulation engines:

1. **2-Step Agent Simulation (Multi-Turn User Simulator):** - **Prerequisite:** Requires an immutable **Agent Registry version snapshot** (capturing instructions, model config, and tool declarations) so runs are reproducible. - **Step 1 (Scenario Synthesis):** Inspects the agent's instructions and tool schemas to synthesize realistic **`Starting Prompts`** paired with a hidden **`Conversation Plan`** (`UserPersona` , goal, edge-case constraints, **`violation_rubrics`** for adversarial red-teaming, and secrets to reveal only if asked). - **Step 2 (Multi-Turn Execution):** `User Simulator` conducts a live multi-turn dialogue against the target agent until the goal is achieved, abandoned, or hits `max_turns` , emitting OpenTelemetry traces for trajectory scoring plus **`per_turn_user_simulator_quality_v1`** (meta-metric verifying the simulated user stayed faithful to its persona and instructions).
2. **Environment Simulation (Fault & Tool Mocking):** - Intercepts outbound tool calls during evaluation to inject **deterministic faults** (HTTP `503 Service Unavailable` , `429` rate limits, timeout spikes, malformed JSON, or synthetic edge-case database rows) verifying fallback logic, retry budgets, and graceful degradation **without mutating production systems**.

Online Evaluation Monitors & the Production Data Flywheel

- **Online Evaluation (`evaluate-online` / `OnlineEvaluator`):** Continuously samples live production traces (requiring Cloud Trace + OpenTelemetry GenAI spans, including retrieved/tool outputs so `HALLUCINATION` does not false-positive) and runs asynchronous autoraters (`HALLUCINATION` , `TOOL_USE_QUALITY` , `GENERAL_QUALITY` , `safety_v1`) wired to Cloud Monitoring **Quality Alerts** on score drift.
- **Golden-Dataset Flywheel:** Capture explicit human feedback (thumbs-down, user re-prompts, HITL escalations) and low-scoring online monitor traces → group failure modes via **`client.evals.generate_loss_clusters()`** → de-identify PII with **Sensitive Data Protection (SDP)** (`ReplaceWithInfoTypeConfig` or `CryptoDeterministicConfig` when multi-turn entity co-reference must hold) → curate expected trajectories into **Example Store** (dynamic few-shot) and the **5-bucket `.evalset.json` regression suite** gated in CI/CD (`agents-cli eval compare --fail-on-regression`).

EXAM TIP
`FINAL_RESPONSE_QUALITY` alone cannot catch **"right answer, wrong path"** bugs (e.g., an agent refunded a customer without calling `check_return_eligibility` first). Always combine outcome rubrics with **`trajectory_in_order_match`** / **`trajectory_exact_match`** when policy mandates specific tool ordering.

COMMON TRAP (OVER-PRIVILEGED ONLINE MONITOR CREATION)
Online evaluation monitors execute under a platform service agent identity that reads trace data and invokes judge models; restrict IAM permissions to create/modify monitors to prevent unauthorized trace inspection or quota spend.

09. Optimize Part 2: Observability, Audit & Prompt Optimization

OpenTelemetry & Agent Registry observability views

- **Standards:** Built on **OpenTelemetry (OTel)** using **GenAI semantic conventions** (hierarchical spans for `invoke_agent`, `chat` / LLM calls, `execute_tool` / MCP calls, and `Model Armor` screening).
- **Enabling telemetry on Agent Runtime:** Set `GOOGLE_CLOUD_AGENT_ENGINE_ENABLE_TELEMETRY=true` (exports traces, metrics, and structured logs **without** raw prompt/response bodies by default) and `OTEL_SEMCONV_STABILITY_OPT_IN=gen_ai_latest_experimental`.
- **Privacy-safe payload capture:** Prompt/completion text capture is opt-in: set `OTEL_INSTRUMENTATION_GENAI_CAPTURE_MESSAGE_CONTENT=EVENT_ONLY` and keep `ADK_CAPTURE_MESSAGE_CONTENT_IN_SPANS=false` (the value `true` is invalid with the latest conventions and collects nothing). Enable only in approved environments paired with **SDP de-identification** and restricted Cloud Logging buckets (`roles/logging.privateLogViewer`).
- **Eight Agent Registry Observability tabs:** 1. **Overview:** Sessions, turns, token throughput, error rates, p50/p95/p99 end-to-end latency. 2. **Evaluation:** Historical offline benchmark trends + live **Online Monitor** quality scores. 3. **Models:** Per-model latency, input/thinking/output token breakdown, and `429` error rates. 4. **Tools:** Per-tool and per-MCP-server **p95 latency**, tool error rates, and **"no-tool interactions"** (spotting tools the agent ignores due to poor descriptions). 5. **Usage:** Active users, tenant consumption, and quota utilization. 6. **Logs:** Correlated structured Cloud Logging entries (`trace_id` linked). 7. **Traces:** Waterfall / DAG span inspector showing exact LLM thinking time vs. tool wait time vs. Model Armor overhead. 8. **Topology:** Auto-discovered live dependency graph of orchestrator agents, A2A sub-agents, models, and MCP servers.

Dual-Attribution Agentic Audit Logging

When an agent acts on behalf of a human user, traditional service-account logs lose *who* triggered the action or *which* agent made the autonomous decision. Agent Platform's **Cloud Audit Logs** (`is_agentic=true` metadata) record **dual attribution**:

1. The cryptographic **Agent Identity** (`spiffe://…` / `principal://…` in Agent Registry).
2. The initiating **human user principal** (propagated OAuth / Workforce Identity context).

This prevents "confused deputy" forensic blind spots during security incident response (**Security guide**).

Auto-Loss Analysis & Programmatic Prompt Optimization

- **Auto-Loss Analysis (`client.evals.generate_loss_clusters()`):** Aggregates failing offline simulation traces and low-scoring production traces, using LLM clustering to group thousands of raw errors into ranked, actionable root-cause buckets (e.g., *"ambiguous date parameter in `book_flight` tool schema"*, *"premature termination before pagination completes"*).
- **Data-Driven Prompt Optimization (`adk optimize` / `optimize-agent` / GEPA):** Closed-loop automated optimization that takes your current system instructions, tool descriptions, and labeled evaluation dataset, iteratively mutates instructions and few-shot examples, scores candidates via `client.evals`, and outputs a Pareto-optimal revision when fixing loss clusters or migrating models (e.g., Gemini 2.5 → Gemini 3.8 Flash or Gemini 4 Argon). Always partition labeled examples into a `train` **split** (used to propose prompt mutations) and a strictly **held-out** `validation` **split** (used to accept/reject candidates) so the optimizer cannot overfit instructions to test prompts.

Production symptom	First place to inspect & remediate
High p95 turn latency	**Traces waterfall** → check whether `thinking_level=HIGH` decode time, serial MCP tool calls, or cross-region retrieval dominates
Agent hallucinates instead of calling a tool	**Tools tab** ("no-tool interactions") → refine tool docstring/schema + add `trajectory_in_order_match` eval
Sudden quality drop on live traffic	**Evaluation tab** (Online Monitors + Quality Alerts) + `generate_loss_clusters()` → SDP-scrub samples into `.evalset.json` and run `adk optimize` (`train` + held-out `validation`)
Unexpected token bill spike	**Models / Usage tabs** → check uncompacted session history growth, dynamic prefix breaking implicit caching, or runaway retry loops

10. Cost, Quotas & Capacity Engineering

Symptom / error	Root cause	Architectural fix
`429 RESOURCE_EXHAUSTED` on Standard PayGo	Shared regional pool contention or project TPM/QPM rate limit	Route stateless calls to the `global` **endpoint**, add exponential backoff with jitter, enable **Priority PayGo** for bursty critical paths, or reserve **Provisioned Throughput (GSUs)** for steady baseline traffic (spills over to PayGo by default unless `X-Vertex-AI-LLM-Request-Type: dedicated` is set)
"Region temporarily at capacity" during Agent Runtime deploy	Physical compute pool exhaustion in that region — **NOT** an IAM or project quota limit	Deploy to an alternate supported region, reduce requested vCPU/memory per instance, or retry off-peak (a quota increase ticket will **not** resolve physical pool exhaustion)

Representative Agent Runtime default quotas (per project per region; verify current docs): `query` / `streamQuery` 90 RPM, session writes 100 RPM, append events 300 RPM, session reads 10,000 RPM, max 100 agent resources, Memory Bank writes 100 RPM / reads 300 RPM, Code Execution sandbox runs 1,000 RPM.

Six high-impact FinOps levers:

1. **Tiered model routing:** Use **Gemini 3.5 Flash-Lite** for high-QPS intent routing/guardrails, **Gemini 3.8 Flash** for main agentic tool loops, and **Gemini 4 Argon** only for complex planning/synthesis.

2. **Right-size `thinking_level`:** Default to `LOW` or `MEDIUM` on routine tool turns; reserve `HIGH` for ambiguous planning steps.
3. **Prefix-aligned Context Caching:** Keep static system instructions and tool definitions byte-identical at the start of the prompt to maximize **implicit cache** hits (95% discount on Gemini API; zero storage cost) or use `cachedContents` where supported.
4. **Session compaction & observation masking:** Summarize older turns and truncate multi-megabyte raw MCP JSON responses after extraction.
5. **Tune `min_instances` & delete idle Vector Search 1.0 endpoints:** Deployed Vector Search 1.0 indexes and warm `min_instances` accrue hourly charges even at zero traffic.
6. **Sampled Online Evaluation & Batch/Flex offline evals:** Sample 1–5% of live traffic for `OnlineEvaluator`, and run nightly 10,000-case golden suites on **Flex PayGo** or **Batch inference**.

11. Failure Modes & Exam Traps

Trap	Why it fails	Correct architectural move
Deploying a custom MCP server onto Agent Runtime	Agent Runtime hosts agent orchestrators (`reasoningEngines`), not arbitrary MCP servers	Deploy **Streamable HTTP MCP servers** on **Cloud Run** or **GKE** and register in **Agent Registry**
Storing chat history in Cloud Run in-memory dicts	Serverless instances scale to zero or recycle without warning	Persist state in **Agent Platform Sessions**, **Memorystore**, **Firestore**, or **Cloud SQL**
Attempting to self-host Gemini 4 Argon or 3.8 Flash on GKE	Proprietary Gemini weights are served exclusively via managed Google endpoints	Consume Gemini via API (with VPC-SC / PT if needed), or self-host open-weight **Gemma 4** on GKE
Executing LLM-generated Python via `exec()` in the agent container	Exposes IAM credentials, env vars, and internal VPC endpoints	Execute untrusted generated code in the **Code Execution sandbox** (or **GKE Sandbox / gVisor**)
Building custom chunking + pgvector when SharePoint/Drive ACLs are required	Re-implementing connectors and document-level ACL sync is brittle and error-prone	Use **Agent Search** with native identity propagation and connector ACLs
Using default `DOT_PRODUCT` on an index backing RAG Engine	RAG Engine queries with cosine distance, triggering `FAILED_PRECONDITION`	Create the index with **`COSINE`** distance
Indexing 3,072d `gemini-embedding-001` directly in Firestore or slicing Matryoshka dims without re-normalizing	Exceeds Firestore's **2,048-dimension** cap, and truncated sub-vectors lose unit norm ($\|u\|_2 < 1$), skewing raw `DOT_PRODUCT_DISTANCE`	Set `output_dimensionality` $\leqslant$ `2048` and **L2 re-normalize** (or set `featureNormType: "UNIT_L2_NORM"` / `COSINE_DISTANCE`)
Selecting RAG Engine Serverless mode under a CMEK mandate or importing >20 MB / >500p PDFs	Serverless mode lacks CMEK (**one-way lock-in**: modes cannot migrate in place), and `ImportRagFiles` Layout Parser caps at **20 MB / 500 pages**	Choose **RAG Engine Spanner mode** upfront when CMEK is required, and split or batch-parse oversized PDFs before import
Setting `rrf_ranking_alpha = 1.0` in hybrid search or calling Ranking API without `discoveryengine` enabled	`alpha = 1.0` weights dense vectors `100%` and drops sparse SKU/acronym matches; missing `discoveryengine.googleapis.com` returns `403`	Set `rrf_ranking_alpha = 0.5` for balanced lexical + dense RRF, and enable `discoveryengine.googleapis.com` for `semantic-ranker-*`
Leaving an idle Vector Search 1.0 `IndexEndpoint` deployed or peering via small `/28` VPC subnets	Deployed replicas bill **24/7 VM node-hours even at `0` QPS**, and VPC Peering exhausts `/26+` CIDR ranges	Undeploy unused endpoints (or use **Agent Retrieval** for elastic scale) and connect via **PSC (`pscAutomation` `Configs`)**
Routing `developerknowledge.googleapis.com/mcp` through `restricted.googleapis.com` inside VPC-SC	Developer Knowledge MCP does not support the `restricted` VIP (and indexes public English docs only)	Route egress via **`private.googleapis.com`** VIP
Forwarding a user OAuth token in `Authorization` to an IAP/IAM-protected Cloud Run MCP server	Cloud Run consumes `Authorization` to verify the caller's OIDC identity token, stripping or rejecting the downstream OAuth token	Pass Cloud Run OIDC in **`X-Serverless-Authorization`** or broker credentials through **Agent Gateway + Auth Manager**

Trap	Why it fails	Correct architectural move
Granting `roles/agentregistry.viewer` expecting runtime A2A/MCP invocation, or sharing uncloned ADK remote agents	`agentregistry.viewer` grants catalog discovery only, and reusing one `get_remote_a2a_agent()` across workflow branches collides parent pointers	Grant `roles/iap.egressor` + endpoint IAM for invocation, resolve once at startup, and call `.clone()` per workflow branch
Appending all turn events into one Firestore document array or using Memorystore as an audit ledger	Hits Firestore's **1 MiB** max document size and **~1 write/s/doc** hotspot limit; Memorystore uses **asynchronous replication** (mutually exclusive RDB or AOF on Cluster/Valkey; RDB-only on legacy standalone Redis) and can lose recent writes on failover	Store turn events in **Firestore subcollections** and persist compliance records in AlloyDB, Cloud SQL, Spanner, or Firestore
Calling `ctx.save_artifact()` when `Runner(artifact_service=None)`	ADK raises `ValueError` at runtime when no `ArtifactService` is bound	Configure `GcsArtifactService` on the `Runner` and pass `gs://` URIs across A2A instead of multi-MB base64 blobs
Triggering slow mutative agents via Pub/Sub Push without idempotency or Dead-Letter Topics	Turns exceeding the **600s max ack deadline** redeliver at-least-once, repeating external writes or looping poison messages forever	Deduplicate `messageId` atomically in Memorystore/Firestore, attach a **DLT (`maxDeliveryAttempts ⩾ 5`)**, or rate-limit via **Cloud Tasks (`maxDispatchesPerSecond`)**
Filing a quota increase for *"Region temporarily at capacity"*	Physical regional compute pool saturation is unrelated to project quota caps	Switch region, right-size container CPU/RAM, or retry
Grading compliance agents only with `FINAL_RESPONSE_QUALITY`	Misses skipped verification tools ("right answer, wrong trajectory")	Pair outcome rubrics with `trajectory_exact_match` or `trajectory_in_order_match`
Testing fault tolerance against live production payment/ERP APIs	Causes real side effects and cannot deterministically trigger 503/timeout edge cases	Use **Environment Simulation** + **2-Step Agent Simulation** against an Agent Registry snapshot
Sharing a semantic cache across users on ACL-protected RAG	Leaks confidential answers across permission boundaries	Partition cache keys by user/group ACL scope or disable semantic caching on restricted corpora
Querying an auto-registered Google Cloud MCP server with `--location=us-central1`	Auto-registered `McpServer` resources exist only in `global`, so the call returns `NOT_FOUND`	Query with `--location=global`, or register the server manually in the region when metadata residency matters
Registering an external MCP server without a `toolspec.json`	Manual registration records only the endpoint; the registry does not inspect the server, so no tools are discoverable	Upload a `toolspec.json` (MCP tool schema, 10 KB max) during registration
Shipping with `InMemorySessionService` on Cloud Run	Sessions vanish on restart and are not shared across replicas	Switch to `VertexAiSessionService` or `DatabaseSessionService` (with PgBouncer / AlloyDB pooling) before first deployment
Using `reasoning-engines revisions update --traffic-split=...` when **Agent Gateway** fronts the agent	Agent Gateway routes directly to target endpoints and **bypasses** runtime-level revision traffic splits (and `InMemorySessionService` drops sessions across revisions)	Shift canary weights at the **Agent Gateway / load-balancer layer** and persist state in `VertexAiSessionService` or `DatabaseSessionService`
Setting `AgentConnectivityTemplate vpcEgress: PRIVATE_RANGES_ONLY` or DNS to `private.googleapis.com` (`199.36.153.8/30`) under VPC-SC	`PRIVATE_RANGES_ONLY` leaks public internet egress and `private.googleapis.com` exposes APIs outside VPC-SC (plus VPC-SC caps Agent Runtime at `max_instances=100` vs. `1,000` outside, with no transitive VPC peering)	Configure `accessPath: AGENT_TO_ANYWHERE`, `vpcEgress: ALL_TRAFFIC`, PSC Interface `/28` `network Attachment`, and `restricted.googleapis.com` (`199.36.153.4/30`)
Using unidirectional **SSE** (`run_async`) for real-time voice agents requiring mid-sentence **barge-in**	SSE streams `Server → Client` only and cannot stream full-duplex PCM audio frames or trigger server-side VAD interruption	Use **Gemini Live API** (`gemini-3.8-live`, `runner.run_live`) over bidirectional WebSockets/gRPC with **session resumption handles**, delegating deep reasoning via `AgentTool`

12. Glossary

- **Agent Runtime:** Fully managed stateful compute runtime for agents (formerly Vertex AI Agent Engine); API resource `reasoningEngines`.
- **Managed Agents API (Preview):** Declarative harness (**Agents API** + **Interactions API** + **Skill Registry**) where Google runs the agent loop and isolated no-egress Linux sandbox.

- **`agent-starter-pack` / `agents-cli`:** Production scaffolding and CLI tool generating Terraform, Cloud Build CI/CD, telemetry wiring, and `agents-cli eval` gates.
- **Session / Event / State:** Single-conversation event log and mutable key-value scratchpad.
- **Memory Bank (`VertexAiMemoryBankService`):** Asynchronous LLM-extracted, consolidated, user-scoped long-term semantic memory with Memory Topics, TTL, CMEK, and revisions.
- **Example Store (Preview):** Dynamic semantic retrieval of few-shot trajectories/examples (embedding model fixed at creation).
- **Code Execution sandbox:** Sub-second isolated Python container (100 MB file I/O, up to 14-day TTL) for untrusted LLM-generated code.
- **Agent Search vs. RAG Engine:** Turnkey connector + native-ACL enterprise RAG (`discoveryengine`) vs. code-first modular corpus pipeline (`20 MB / 500p` `ImportRagFiles` Layout Parser cap).
- **Agent Retrieval vs. Vector Search 1.0:** Self-tuning schema-aware collection store vs. ScaNN (`treeAhConfig`) billion-scale ANN index-as-a-service (`24/7` `IndexEndpoint` node-hour billing).
- **`text-embedding-005` / `gemini-embedding-001` / `gemini-embedding-2`:** Agent Platform embedding models covering 768d English/code (8 `task_type` values), 3,072d multilingual Matryoshka text/code, and 3,072d 8,192-token omni-modal embeddings.
- **Matryoshka (`output_dimensionality`) & `UNIT_L2_NORM`:** Prefix-slicing embeddings (`768` / `1,536` / `3,072`) paired with L2 re-normalization so `DOT_PRODUCT_DISTANCE` preserves cosine ranking on ScaNN SIMD paths and fits Firestore's `2,048`-dim limit.
- **ScaNN crowding (`per_crowding_attribute_num_neighbors`):** Vector Search 1.0 diversity cap limiting how many nearest neighbors can share a metadata attribute such as `document_id` or `brand_id`.
- **Reciprocal Rank Fusion (`rrf_ranking_alpha`) & Ranking API (`semantic-ranker-*`):** Rank-order fusion of sparse lexical and dense vector search (`0.5` balanced; `1.0` dense-only) followed by `< 100 ms` stateless cross-encoder re-scoring (`discoveryengine.googleapis.com`).
- **2-Step Agent Simulation:** Synthesizes `Starting Prompt` + `Conversation Plan` from an Agent Registry snapshot, then runs multi-turn simulated user sessions.
- **Environment Simulation:** Injects deterministic tool faults (503s, latency, mocked payloads) during evaluation.
- **Dual-Attribution Audit Log:** Cloud Audit Log (`is_agentic=true`) recording both the SPIFFE **Agent Identity** and the initiating human principal.
- **Failure Cluster Analysis & Prompt Optimization:** Automated root-cause grouping of failed traces paired with closed-loop programmatic system-prompt tuning.
- **Google Cloud remote MCP server:** Google-hosted stateless HTTPS MCP endpoint (`2026-07-28` spec), auto-registered in `global` Agent Registry and gated by `roles/mcp.toolUser` plus resource IAM.
- **MCP Toolbox for Databases (`ToolboxToolset`):** Open-source MCP server (`googleapis/genai-toolbox` / `mcp-toolbox`) providing pooled, parameterized `tools.yaml` queries and OTel tracing across 30+ databases.
- **`ApplicationIntegrationToolset` & `ApiHubToolset`:** ADK enterprise toolsets wrapping 100+ SaaS/ERP Integration Connectors (with Auth Manager 3LO) and Apigee/API Hub governed OpenAPI catalogs.
- **`toolspec.json`:** Tool specification file (MCP tool schema, 10 KB max, including `readOnlyHint` / `destructiveHint`) uploaded when manually registering an external MCP server in Agent Registry.
- **Streamable HTTP vs. stdio:** Remote HTTPS MCP transport for cloud servers vs. local child-process transport for servers on the same machine.
- **`VertexAiSessionService` vs. `DatabaseSessionService` vs. `InMemorySessionService`:** ADK session backends for managed Agent Platform Sessions (365d TTL, CMEK), customer-operated SQL databases (requiring PgBouncer/AlloyDB pooling on Cloud Run), and non-persistent process memory.
- **`GcsArtifactService`:** Cloud Storage-backed ADK `ArtifactService` storing versioned binary blobs (`{app}/{user}/{session}/{filename}/{version}` and `user:` scope, including Zonal Rapid buckets) outside `session.state`.
- **Pub/Sub Push + Eventarc vs. Cloud Tasks vs. Cloud Workflows vs. Cloud Run Jobs:** Event fan-out (`600s` max ack deadline + `messageId` idempotency + DLT) vs. rate-limited point-to-point dispatch (`maxDispatchesPerSecond`) vs. deterministic multi-day sagas vs. 24-hour parallel batch shards.
- **Skill Registry (Preview):** Agent Platform repository that validates, versions, and serves `SKILL.md`-based agent skill packages; inherits project IAM (`roles/aiplatform.*`).
- **Skill revision:** Immutable snapshot of one version of a skill; the skill resource points to a default revision that agents load unless a specific revision is requested.
- **`gcp-skill-registry`:** Google-managed built-in skill that lets an agent create, search, and manage skills in Skill Registry.
- **Semantic governance policy (Preview):** LLM-evaluated natural language constraint that can allow or deny skill tool calls such as `load_skill` and `run_skill_script`.
- **RAG Engine Serverless mode vs. Spanner mode:** Zero-tier autoscaled backend without CMEK vs. dedicated Spanner instance with Basic or Scaled tiers, isolation, and CMEK (one-way mode lock-in).
- **Collection / Data Object:** Agent Retrieval container with a schema (similar to a table) and the individual JSON record it stores alongside its vector fields.
- **Semantic cache:** Vector-indexed response cache (for example Memorystore for Valkey / Redis vector search) that serves stored answers to semantically similar prompts; partition keys by tenant or ACL scope.