Agent Platform Security & Governance by Michaël Bettan

The four governance components

Component	Answers	Product
Identity	*Who* is acting?	Agent Identity (SPIFFE-based per-agent principal), or service accounts (legacy/default)
Registry	*What* destinations exist and are approved?	Agent Registry (agents, MCP servers/tools, endpoints, skills)
Policies	*What* actions are allowed?	IAM allow/deny, Principal Access Boundary (PAB), Unified Access Policies on the gateway, Semantic Governance
Gateway	*Where* is it enforced?	Agent Gateway (Client-to-Agent ingress, Agent-to-Anywhere egress)

Content controls (Model Armor, SDP, Gemini safety filters), network/data controls (VPC-SC, PSC, CMEK, residency), detection (Audit Logs, SCC), and humans (HITL) sit on top of these four as defense in depth. In Google's **Secure AI Framework (SAIF)** (`saif.google`), secure agent design rests on three pillars: **well-defined human controllers** (Agent Identity + HITL approvals), **carefully limited powers** (IAM, PAB, and gateway policies), and **observable actions and planning** (dual-attribution audit logs + SCC AI Protection).

"Which control prevents X?" (master decision table)

Threat / requirement	Primary control	Why not the others
Shared service-account key leaked; you need per-agent, non-impersonable credentials	**Agent Identity**	Service accounts are shared by default and can be impersonated
Stolen agent access token replayed from another machine	**Agent Identity** (certificate-bound tokens + default Context-Aware Access: mTLS + DPoP)	IAM roles don't stop replay of a valid token
Agent must act on *a user's* Jira/GitHub data with that user's consent	**Auth manager, 3-legged OAuth** (user-delegated)	An API key or the agent's own identity would act with the agent's authority, not the user's
API key hard-coded in agent source	**Secret Manager** (secret env var or ADK `SecretManagerClient`) + `secret Accessor` for the agent	Env-var plaintext and code are both visible to anyone who can read the deployment
Agent can reach *any* project in the org despite narrow intent	**Principal Access Boundary** (boundary on eligible resources)	Allow policies only grant; PAB caps eligibility no matter what roles exist
A broad role must never include one dangerous permission	**IAM deny policy**	PAB limits *resources*, not specific permissions
Agent may call MCP tool `get_order` but not `delete_order`	**Unified Access Policy** on Agent Gateway (CEL on `tool.name` / `destructive_ hint`)	IAM on the Google Cloud API can't see MCP tool names
Agent calls an unregistered external host	**Agent Gateway + Agent Registry** (default-deny egress; UAP on `destination. unregistered.*`)	Model Armor screens content, not destinations
An email in context instructs the agent to forward mail to an attacker (indirect injection → rogue tool call)	**Semantic Governance** (intent alignment) + **Model Armor** prompt-injection filter	UAP sees an allowed tool name and can't judge intent
Refund over a business limit (e.g., >$75) must never auto-execute	**Semantic Governance** natural-language constraint, or deterministic check + **HITL tool confirmation**	Model Armor has no business rules
Jailbreak / prompt injection / malicious URL in user input	**Model Armor** (input template)	Gemini safety filters only cover harm categories
Every team's Model Armor templates must keep prompt-injection detection on, whatever each team configures	**Model Armor floor settings** at org or folder level (template conformance)	A template protects only the workload that references it
Credit-card numbers leak in responses	**Model Armor SDP filter** (advanced mode with de-identify template) / **Sensitive Data Protection**	Safety filters don't detect custom infoTypes
Hateful output from the model	**Gemini safety filters** (model level) and/or **Model Armor RAI filter** (model-agnostic)	—
Data exfiltration from Google APIs to an outside project	**VPC Service Controls** perimeter	IAM alone can't stop a legitimately authorized identity copying data out
Agent must reach on-prem/private VPC services	**PSC interface** (network attachment)	Agent Runtime can't reach your private VPC/on-prem by default (Google-managed tenant)
Regulator requires customer-controlled keys	**CMEK** (`encryption_spec` , same-region key)	Google default encryption isn't customer-controlled
Data and ML processing must stay in the EU	**Data residency**: EU multi-region/regional endpoint; block the global endpoint	The global endpoint has no residency guarantee
"Who changed the agent's IAM?"	**Cloud Audit Logs** (Admin Activity, always on)	Data Access logs must be enabled explicitly
Detect anomalous grants to agent identities / over-privileged agents	**SCC AI Protection** + Event Threat Detection	Audit logs record but don't alert
Irreversible, high-stakes action needs sign-off	**HITL**: ADK tool confirmation (`require_confirmation`)	Automated filters can be wrong; accountability needs a human
Hundreds of unknown agents/tools across teams (agent sprawl)	**Agent Registry** (central catalog) + gateway default-deny + org policy	Ad-hoc spreadsheets aren't enforced

Launch-stage status (as documented, 2026-10)

Capability	Status	Note
Agent Identity	**GA** (since 2026-04-22)	Falls back to service accounts if not enabled
Agent Identity auth manager	**GA** (since 2026-08-22, with the `agentidentity` / `agentidentitycredentials` APIs)	Older IAM Connectors API stays Preview and will not reach GA
Agent Gateway	**GA** for Agent Runtime + Gemini Enterprise (GE: egress only)	GKE, Cloud Run, cross-cloud: coming soon / early preview
Agent Registry	GA (verify); **standalone skills in Registry = Preview**	—
Skill Registry	**Preview**	—
Semantic Governance policies	**Preview**	—
Token-management policies, extension bindings	Coming soon	—
Agent Platform Threat Detection (SCC)	**Preview**, Premium tier	—
ADK tool confirmation (HITL)	**Experimental**	—
Model Armor image screening, template modality selection, exclusion rules	**Preview**	Text and document screening carry no Preview label
Model Armor floor settings for Google Cloud MCP servers	**Preview** (as labeled on the floor settings page)	Labeled Preview on the floor settings page
Model Armor spans in Cloud Trace	**Preview**	Span attribute names may change
Agent Gateway Cloud Trace integration	**Preview**	Gateway request logs and the Observability dashboard carry no Preview label
Model Armor insights (Agent Platform Security tab)	**Preview**	Needs tracing set up for the agent
Agent Anomaly Detection	**Preview**, allowlisted users only	Findings go to SCC (Standard, Premium or Enterprise)

> EXAM TRAP.
> "Preview" isn't a reason to reject a control in a design question, but you must know which features are not GA. Some security controls (e.g., CMEK/VPC-SC support) aren't available for preview models.

01. Agent Identity Model

Three ways an agent proves who it is

Option	What it is	Strengths	Weaknesses
Default service agent	AI Platform Reasoning Engine Service Agent `service-PROJECT_NUMBER@gcp-sa-aiplatform-re.iam.gserviceaccount.com` (`roles/aiplatform.reasoningEngineServiceAgent`)	Zero setup	Shared by every agent in the project, so no per-agent least privilege
Custom service account	You attach your own SA	Per-workload roles possible	Can be shared and impersonated; key-management risk
Agent Identity (GA)	Per-agent, SPIFFE-based, strongly attested principal tied to the agent's lifecycle and resource URI	Not shared, **can't be impersonated**, no long-lived keys; tokens bound to the agent's X.509 cert (24h, auto-rotated); Context-Aware Access enforces **mTLS + DPoP** by default	GA since 2026-04-22; new principal on every redeploy (see traps)
Workload Identity Federation (WIF)	Federates external/other-cloud workloads to Google Cloud without keys	Used by Agent Gateway / auth manager for cross-cloud and external identity exchange	Not an agent-native identity

Agent Identity is supported on **Agent Runtime, Gemini Enterprise, and Cloud Run**.

Principal formats

- SPIFFE ID: `spiffe://agents.global.org-ORG_ID.system.id.goog/resources/aiplatform/projects/PROJECT_NUMBER/locations/LOCATION/reasoningEngines/AGENT_ID`
- Single agent (IAM): `principal://agents.global.org-ORG_ID.system.id.goog/resources/aiplatform/projects/PROJECT_NUMBER/locations/LOCATION/reasoningEngines/AGENT_ID`
- All agents in a project: `principalSet://agents.global.org-ORG_ID.system.id.goog/attribute.platformContainer/aiplatform/projects/PROJECT_NUMBER`
- All agents in the org: `principalSet://…/attribute.platform/aiplatform`

Enabling it

- SDK (`api_version="v1beta1"`): `client.runtimes.create(config={"identity_type": types.IdentityType.AGENT_IDENTITY, ...})`. You can create the identity **before** deploying code so you can set up IAM first, then add code with `client.runtimes.update(...)`.
- CLI: `agents-cli deploy --agent-identity`; for `adk deploy`, set it in `.agent_engine_config.json`.
- If the setting is absent, the agent runs as a service account.
- Default roles granted to the identity: `roles/aiplatform.agentContextEditor`, `roles/aiplatform.agentDefaultAccess`.
- **Minimum least-privilege runtime role baseline (never grant `roles/editor` or `roles/owner`):** `roles/aiplatform.user` (invoke Gemini models, Managed Sessions, and Memory Bank), `roles/iap.egressor` (pass Agent Gateway Cloud IAP `REQUEST_AUTHZ` checks), `roles/agentregistry.viewer` (resolve governed A2A agents and MCP servers), `roles/modelarmor.user` (call Model Armor sanitization endpoints), plus `roles/secretmanager.secretAccessor` only on required startup secrets (`roles/iam.principalAccessBoundaryAdmin` stays restricted to central security for PAB ceilings).

Lifecycle and CI/CD traps

Trap	Consequence	Fix
Agent redeployed (new resource ID)	New principal, so **old bindings don't apply**	After deploy, read `spec.effectiveIdentity`; put baseline roles on the project `principalSet`, which survives redeploys
Agent deleted	IAM bindings **remain** (orphaned)	Remove bindings as part of teardown
Granting legacy Cloud Storage bucket roles	Not supported for agent identities	Use the IAM predefined roles
Parent agent delegates to a subagent (A2A)	Parent needs discovery + invoke rights	Grant the **parent's agent identity** `roles/agentregistry.viewer` and `roles/aiplatform.user` on the subagent; don't grant them to the user

> **EXAM TRAP.**
> "Use a service account key stored in the container" is never the answer. Prefer Agent Identity, or an attached SA with no exported keys.

02. Tool Authentication & Secrets

Two authorities

Model	Agent acts as…	Mechanism	Use when
User-delegated	the end user (with consent)	**3-legged OAuth** via the auth manager	Per-user data in SaaS (Jira, GitHub, Salesforce, ServiceNow, Workspace)
Agent's own authority	itself	Agent Identity → Google Cloud APIs; **OIDC ID tokens** → external services (federation); **2-legged OAuth** (machine-to-machine); **API key**; HTTP basic (not recommended)	Shared back-end services, batch jobs, service-to-service calls

Audit logs record **both the agent and the user** when the agent acts on a user's behalf, and only the agent when it acts on its own authority.

Identities in one agent call

Identity	What it represents	Where it comes from	Typical use
End user	The person typing the prompt	Cloud Identity or an external IdP through Workforce Identity Federation	Carries the user's own data entitlements (document sharing, row-level rules)
Agent Identity	One deployed agent	Provisioned by Agent Runtime, Gemini Enterprise or Cloud Run; SPIFFE ID with a certificate-bound credential	Calls Google Cloud APIs and the gateway as the agent; is granted `roles/agentidentity.user` on the auth providers it may use
Service account	A workload, not a specific agent	Attached by you, or the default Reasoning Engine service agent	Fallback when Agent Identity is off; also the runtime identity of a self-hosted MCP server (e.g., on Cloud Run)
OAuth client in an auth provider	The registered app at a third-party service (Jira, Salesforce, GitHub)	Client ID and secret stored in an auth manager auth provider	The "application" side of 2LO and 3LO flows

Acting as itself vs. On behalf of the user:

- **As itself:** the data or action is not tied to who is asking (reference data, shared queues, batch jobs, service-to-service calls). Use Agent Identity for Google Cloud APIs, or a 2LO/API-key auth provider for a third party.
- **On behalf of the user:** the result must differ per user, or the action must be attributable to that user (their mailbox, their tickets, CRM records filtered by owner). Use a 3LO auth provider so the downstream service applies the user's own permissions.
- **Mixed SPIFFE + 3LO within one turn:** an orchestrator routinely uses both authorities inside a single turn, presenting its own **SPIFFE Agent Identity** for internal BigQuery queries or peer A2A subagent calls while invoking **Auth Manager 3LO** (`adk_request_credential` / `consent_nonce`) for user-scoped Jira, Drive, or Salesforce mutations.
- **Anti-pattern (shared super-credential):** one 2LO token or broad service account answers every user's request. The agent then decides who sees what, which sets up a confused deputy (see §03).

Agent Identity auth manager

- A centralized **credential vault and auth broker** (**GA** since `2026-08-22`). "Auth providers" (`authProviders/`) hold API keys, OAuth client IDs/secrets and delegated user tokens.
- Automates the OAuth dance with **PKCE**: consent (`adk_request_credential` carrying an auto-generated anti-session-fixation `consent_nonce` verified when the callback posts `"consentNonce"` to `credentials:finalize`, plus registered `continue_uri`), authorization-code exchange, and refresh-token rotation using **minimized OAuth scopes** (e.g., `drive.readonly` or `read:jira-work`, never broad admin scopes).
- ADK binds providers either directly via **`GcpAuthProviderScheme`** (`name="projects/<PROJECT>/locations/<REGION>/authProviders/<ID>"`, passed to `McpToolset(..., auth_scheme=...)` or `AuthenticatedFunctionTool`) or automatically through a same-region outbound binding via `AgentRegistry(project_id=..., location="<REGION>").get_mcp_toolset(mcp_server_name=..., continue_uri=...)`, injecting `Authorization` / `X-Goog-Api-Key` headers at the tool/gateway boundary so raw credentials never enter LLM context, `session.state`, or agent source code (**location trap:** `authProviders/` are **strictly regional** and unavailable in `global`; mismatching regions across the auth provider, MCP server, and `AgentRegistry` client fails outbound binding).
- Access to auth providers is governed by IAM (`roles/agentidentity.user`) using the agent's SPIFFE identity. User grants can be viewed and revoked without redeploying the agent.
- With **Agent Gateway + Gemini Enterprise**, end-user credentials are encrypted by the auth manager and decrypted at the gateway, so the agent process never sees the raw credential.
- Constraint: some providers (e.g., GitHub, Microsoft) support only a single scope.
- **API migration (`iamconnectors.googleapis.com` deprecated):** auth providers were first exposed through the IAM Connectors API (`iamconnectors.googleapis.com`, Preview), which is deprecated and will not reach GA. Use the Agent Identity API (`agentidentity.googleapis.com`; credential calls on `agentidentitycredentials.googleapis.com`), where resources are named `authProviders/` instead of `connectors/`. Both run side by side during migration.

Credential boundary matrix (5-way comparison)

Credential pattern	Backing service (status)	Rotation & lifecycle	Audit trail & confused-deputy resistance	Primary operational trap
Agent's own cloud / peer identity (`SPIFFE`)	**Agent Identity** (`GA` , `principal://agents.global.org-...`)	Google-minted X.509 cert + mTLS / DPoP-bound tokens auto-rotated every ~24h	Cloud Audit Logs record exact Agent SPIFFE Principal (`principalSubject`). Limited confused-deputy resistance across multiple end users; pair with PAB, UAP, and HITL	Recreating an agent mints a new resource ID and SPIFFE principal unless baseline roles sit on the project `principalSet`
End-user delegated SaaS access (`OAuth 3LO`)	**Auth Manager** (`GA` , 3LO `authProviders/`)	One-time user consent (`adk_request_credential` / `consent_nonce` + `continue_uri`); refresh tokens vaulted and rotated automatically	Dual attribution in audit logs (Agent SPIFFE Principal + consenting End User). **Highest** confused-deputy resistance because downstream SaaS enforces per-user ACLs	Requires an interactive front end for the OAuth redirect and registered callback URL; single-scope cap on providers like GitHub/Microsoft
Machine-to-machine external OAuth (`OAuth 2LO`)	**Auth Manager** (`GA` , 2LO `authProviders/`)	Client ID and secret vaulted in Google-managed storage; short-lived access tokens minted and refreshed automatically	Logs agent access to `authProviders/` (`roles/agentidentity.user`), while destination sees one shared app client. **Low** confused-deputy resistance	Using 2LO on multi-user SaaS data forces the prompt to filter per-user records (classic confused deputy)
Third-party tool API keys	**Auth Manager** (`GA` , API-key `authProviders/` , backed by Secret Manager)	Injected into `Authorization` / `X-Goog-Api-Key` headers at the proxy/tool boundary outside LLM context; rotate key value externally	Logs `authProviders/` access; destination sees static bearer key with zero caller identity (**lowest** confused-deputy resistance)	No automatic upstream key rotation unless paired with an external rotation workflow
Infrastructure secrets (DB passwords, webhook HMACs, TLS certs)	**Secret Manager** (`GA`)	Scheduled rotation via Pub/Sub notifications + version pinning, or live runtime lookup via ADK `SecretManagerClient`	`secretmanager.googleapis.com` Admin Activity + Data Access Audit Logs (`roles/secretmanager.secretAccessor`)	Deploy-time secret env vars (`env_vars={"API_KEY": {"secret": "SECRET_ID", "version": "latest"}}`) work only for **same-project** secrets (also needing the Agent Platform service agent) and require a restart on rotation unless fetched at runtime

- **External token exchange (RFC 8693):** outside the five vault/identity boundaries above, **Workload Identity Federation (WIF)** swaps external OIDC/SAML or Kubernetes tokens for short-lived Google credentials via the Security Token Service (used by Agent Gateway cross-cloud early preview to exchange Kubernetes workload credentials for Agent Identity credentials).

Top 3 documented credential misconfigurations

1. **Confused deputy via shared 2LO or service account on multi-user SaaS:** answering per-user Jira, Drive, or Salesforce queries with one shared 2LO client or broad service account lets prompt injection read or mutate another user's records. Fix: enforce **Auth Manager 3LO** for user-scoped SaaS access and pair shared writes with UAP + **HITL** confirmation.
2. **Downloadable service account JSON keys:** baking exported `.json` keys into images, `.env` files, or developer laptops creates non-expiring, replayable credentials. Fix: use keyless **Agent Identity** (or WIF / attached SAs) and enforce Org Policy `constraints/iam.disableServiceAccountKeyCreation` .
3. **Plaintext secrets in environment variables, prompts, or `session.state` :** leaks raw keys into deployment metadata (which CMEK does not encrypt), Cloud Trace spans, and session logs, raising Security Command Center finding `SECRETS_IN_ENVIRONMENT_VARIABLES` . Fix: store infrastructure secrets in **Secret Manager** and inject tool credentials via **Auth Manager**.

State and memory hygiene: what must never enter `session.state` or Memory Bank

Forbidden data class	Why storing it in state or memory fails	Required Google Cloud control
OAuth 3LO / 2LO tokens	Writes live bearer/refresh tokens into session history, traces, and extracted memories where indirect prompt injection can exfiltrate them	Keep tokens exclusively in the **Auth Manager** vault and inject headers at the tool/gateway boundary
Static API keys, DB passwords, SA JSON keys	Persists static credentials across checkpoints and audit logs; deployment env plaintext triggers SCC `SECRETS_IN_ENVIRONMENT_VARIABLES`	Fetch at runtime from **Secret Manager** (`SecretManagerClient`) or an **Auth Manager** API-key provider
Unredacted PII / PHI / PCI	Raw identifiers become immutable session events and get distilled into cross-session **Memory Bank** facts	Redact or deterministically tokenize via **SDP** (`CryptoDeterministicConfig` surrogates) inline with **Model Armor** or ADK `before_model_callback` / `after_tool_callback`
Large binary blobs (PDFs, images, audio)	Exceeds state-store caps (such as Firestore's 1 MiB document limit), bloats LLM context, and chokes **Agent Gateway** `CONTENT_AUTHZ` payload buffers when base64-encoded inside A2A `Part` messages	Store versioned blobs in **Cloud Storage** via ADK `GcsArtifactService` and pass `gs://` URIs across A2A

DECISION RULE.
Per-user consent → 3LO through the auth manager. Static third-party tool key → Auth Manager API-key provider or Secret Manager. Google Cloud API → the agent's own SPIFFE identity with least-privilege IAM. Large files → `GcsArtifactService` (`gs://` URIs). Never put secrets, raw PII, or multi-MB blobs in code, prompts, plaintext env vars, or `session.state` .

Identity propagation, hop by hop

Each hop checks a different credential against a different policy. Access to the data succeeds only when every hop allows it.

Hop	Credential presented	Validator	Policy applied
User → Gemini Enterprise (or custom front end)	User sign-in session (Cloud Identity or federated IdP)	Gemini Enterprise / the front end's identity check	Whether this user may use the app and agent
Front end → agent on Agent Runtime	Caller identity plus the user's context in the request	Agent Runtime	IAM permission to query the agent (e.g., `roles/aiplatform.user` on it); ingress gateway policies if Client-to-Agent is enabled
Agent → Agent Gateway (egress)	Agent Identity certificate-bound credential (mTLS + DPoP) and the tool credential injected from the auth manager	`REQUEST_AUTHZ` extension (IAP evaluates headers and identity), then `CONTENT_AUTHZ` extensions (Model Armor, Semantic Governance read the payload)	UAP (`roles/iap.egressor` / `egressViaIAP`) with CEL on tool and destination attributes; PAB on egress; Model Armor + Semantic Governance
Gateway → tool or MCP server	The destination's credential: user 3LO token, 2LO token, API key, or the agent's Google token	The MCP server or API (e.g., IAP in front of a Cloud Run MCP server; Google remote MCP endpoint)	Destination IAM (`roles/iap.httpsResourceAccessor` for an IAP-protected Cloud Run server; `roles/mcp.toolUser` for Google remote MCP) and OAuth scopes
Tool or MCP server → data source	The user's delegated token, the agent's identity, or the MCP server's own service account	BigQuery, Cloud SQL, Cloud Storage or the SaaS API	Source IAM and ACLs (row/column rules, document sharing), VPC-SC perimeter

```
User —sign-in—► Gemini Enterprise —query (user ctx)—► Agent (Agent Runtime)
Agent —"need Jira token for U"—► Auth Manager (3LO consent —► vaulted token)
Agent —tools/call + Agent Identity (mTLS+DPoP) + token—► Agent Gateway
  ├— REQUEST_AUTHZ (IAP: UAP, PAB)
  └— CONTENT_AUTHZ (Model Armor, Semantic Governance)
Gateway —allowed call + user token—► MCP server —► Jira (enforces U's ACLs)
Jira —U's records—► MCP server —► Gateway (Model Armor) —► Agent —► User
```

EXAM TRAP.
If the MCP server queries the data source with its own broad service account, the user's identity stops at the server and the source can no longer filter by user. Propagate the user token (3LO) to the last hop when results must be user-specific.

03. Access Boundaries: IAM, Deny & PAB

Policy types compared

Policy	Effect	Granularity	Answers
IAM allow policy	Grants roles	Resource + role (+ condition)	"What *may* this principal do?"
IAM deny policy	Denies specific permissions; overrides allow	Permission-level	"What must it *never* do, whatever its roles?"
Principal Access Boundary (PAB)	Limits which resources a principal is *eligible* to access	Org/folder/project (up to 500 rules per policy, 500 resources per rule; effect `ALLOW` only)	"Where can it *ever* operate?"
Unified Access Policy (gateway)	Allow/deny on agent traffic through Agent Gateway	MCP tool, agent, endpoint, unregistered host/path/method	"Which tools and destinations?"

Least privilege for agents

1. Give each agent its own identity (Agent Identity), not the shared default service agent.
2. Grant specific predefined roles on specific resources. Avoid `Owner` / `Editor`.
3. Put baseline roles on the project `principalSet`; per-agent extras on the `principal://` of that agent.
4. Add deny policies for permissions that must never be used (e.g., IAM admin).
5. Bind a **PAB policy** to the agents' principal set so they can't reach resources outside approved projects, even if someone over-grants.
6. Govern tool-level access at the gateway (UAP) because Google Cloud IAM can't see MCP tool names.

PAB specifics

- Rules list Resource Manager resources. PAB has an `enforcementVersion` and blocks only the permissions supported by that version.
- Binding example: `gcloud iam principal-access-boundary-policies bindings create … --target-principal-set=…` (bound to a principal set, e.g., the agents in a project or org).
- For agents, PAB can also be enforced on egress through Agent Gateway (via IAP).

EXAM TRAP.
PAB doesn't grant anything. A principal needs an allow-policy role **and** must be inside its boundary **and** must not be denied.

Effective access and CEL conditions

- **Effective access** = (resources granted by allow policies ∩ resources eligible under PAB) minus permissions blocked by deny policies.
- **Worked case:** someone grants `roles/bigquery.admin` at the org level to the agents' `principalSet`. A PAB bound to that set that lists only the `analytics-prod` project keeps the agents out of every other project's datasets.
- **IAM conditions** add CEL to an allow binding so the grant applies only in context, e.g., `resource.name.startsWith("projects/_/buckets/agent-reports")` or `request.time < timestamp("2027-01-01T00:00:00Z")` for a time-boxed grant.
- **Gateway CEL** (in UAPs, §05) works on tool and destination attributes that IAM conditions can't see, e.g., allow only when `destination.agent_registry.mcp_server.tool.annotations.read_only_hint == true`.

Identity failure scenarios

Failure	How it happens	Controls that prevent it
Confused deputy	The agent holds one strong credential (2LO token or broad service account) and a user's prompt steers it to read or change records that user can't access	3LO so the source checks the requesting user; UAP deny on `destructive_hint == true`; Semantic Governance intent check; HITL confirmation for writes
Token leakage in prompts, logs or traces	A token or key is placed in the prompt, returned in tool output, or captured in logs and spans, then replayed	Auth manager header injection (credentials never in code or prompts); gateway-side decryption of end-user credentials with Gemini Enterprise; Model Armor SDP filter on prompts and responses; certificate-bound Agent Identity tokens that fail when replayed elsewhere; SCC finding *Agentic Identity Credential Used Outside of Google Cloud*
Over-privileged service account	Every agent in a project runs as the default service agent or one shared SA with `roles/editor`, so one hijacked agent reaches everything	Per-agent Agent Identity; PAB on the agents' `principalSet`; deny policies for admin permissions; per-provider `roles/agentidentity.user`; SCC over-privileged agent detection

04. Agent Registry, Skill Registry & Agent Gateway

Agent Registry

- A central catalog for **agents, MCP servers and tools, endpoints, and standalone skills** (skills: Preview).
- **Automatic registration** covers supported workloads in the governance project, including **Google Cloud remote MCP servers** (`https://<service>.googleapis.com/mcp`, enabled with the service API and requiring `roles/mcp.toolUser` / `mcp.tools.call` plus service IAM roles). **Manual registration** uses `Service` resources for remote, custom (e.g., MCP Toolbox for Databases or third-party SaaS connectors on Cloud Run/GKE), or cross-project components (`gcloud agent-registry mcp-servers list`).
- Governance-project model: single-project or cross-project.
- Agent Gateway **queries the registry before routing**, and policies reference registry identifiers (`urn:agent:` identifiers are for inventory, **not** for IAM bindings).
- Limit: up to 5,000 Agent Registry resources per gateway.
- Purpose: fight **agent sprawl**, fragmented tool access and redundant integrations.

Skill Registry (Preview)

- A secure, private, low-latency repository of **skills** (instructions + code + docs) that agents discover and load on demand.
- Includes a built-in `gcp-skill-registry` skill. Semantic Governance can constrain which skills an agent loads.

Agent Gateway

Path	Direction	Typical controls
Client-to-Agent	Ingress (users/apps → agent)	mTLS termination, IAP access policy, Model Armor screening of ADK `streamQuery`
Agent-to-Anywhere	Egress (agent → tools, agents, LLMs, APIs)	UAPs, PAB, Semantic Governance, Model Armor on MCP `tools/call`, A2A messages and OpenAI-compatible payloads

- Mediates **MCP, A2A, REST, gRPC**, and inspects MCP JSON-RPC to extract tool names for policy.
- Emits OpenTelemetry traces and access logs to Cloud Logging/Trace.
- Works with the auth manager for OAuth and WIF.
- **Deployment models:** Google-managed for Agent Runtime + Gemini Enterprise. GKE, Cloud Run and cross-cloud (self-managed Envoy + WIF) are coming soon / early preview.
- Org governance: custom org policy constraints can restrict which gateways may be associated with agents.

LIMITATIONS.
Gemini Enterprise supports only **Agent-to-Anywhere (egress)**; Client-to-Agent is for Agent Runtime. Ingress governs `query` / `streamQuery` only. All agents in the same project + region bind to the same egress/ingress gateway. SCC Agent Engine Threat Detection isn't available while a gateway is enabled. VPC-SC works with Agent Gateway when the agent uses VPC connectivity (route all egress through your VPC, restricted VIP).

05. Gateway Policies: UAP & Semantic Governance

Agent Policies: 4-layer governance architecture

Agent Policies ("Agent Platform Policies") form the centralized multi-layer policy stack governing **A2A agent delegations**, **MCP tool invocations** (`tools/call`), **dynamic skill loads** (`load_skill`), and **content/data egress** across **Agent Gateway**, **Agent Registry**, and **IAM**:

Policy layer (stage & profile)	Enforcement mechanism	Hard quotas & limits	Primary configuration trap
1. IAM Unified Access Policies (UAP) (`GA`, `REQUEST_AUTHZ`)	Deterministic header-phase RBAC/ABAC via Cloud IAP (`roles/iap.egressor` / `iap.googleapis.com/resources.egressViaIAP` + CEL); default-deny, deny overrides allow	Max **4 custom `AuthzPolicy` resources** per gateway; header-only (cannot inspect HTTP bodies)	Binding a UAP rule to a `global` Agent Registry URI when the gateway/agent resolves a regional registry (or vice versa) causes `403` (`IAP_AUTHZ = DENIED`)
2. Principal Access Boundary (PAB) (`GA`, IAM + gateway IAP)	Identity-centric resource eligibility ceiling bound to an Agent Identity `principalSet`	Max **500 rules per PAB policy**, **500 resources per rule**	Never grants access on its own; requires an explicit IAM/UAP allow binding inside the boundary
3. Semantic Governance Policies (SGP) & Skill Governance (`Preview`, `CONTENT_AUTHZ`)	Intent-aware LLM policy firewall evaluating **Natural Language Constraints (NLCs)** and user-intent alignment across chat history, proposed tool calls, and dynamic skill loads (`govern-agent-skills` intercepting `load_skill`); strict **deny-overrides** (`any DENY ⇒ block`)	Max 1 **`CONTENT_AUTHZ` policy on ingress**; **5,000 chars per NLC**, **1,000 chars** per inspected MCP/tool description, **1,000 chars** custom denial message	Using probabilistic SGP NLCs for static role/tool blocks that belong deterministically in `REQUEST_AUTHZ` IAP CEL, or intercepting raw gRPC health traffic without CEL scoping
4. Model Armor Guardrail Policies (`GA` text / `Preview` features, `CONTENT_AUTHZ`)	Inline full-duplex streaming inspection (`FULL_DUPLEX_STREAMED` Envoy `ext_proc`) for prompt injection, jailbreak, RAI, SDP, and malicious URLs	Max 1 **`CONTENT_AUTHZ` policy on ingress**; screens first **256 URIs** per payload; Org/Folder **Floor Settings** enforce a mandatory baseline	Project admins granted `roles/modelarmor.floorSettingsAdmin` can disable inherited floors unless that role is restricted to central security

Unified Access Policies (IAM v3) & verified CEL attributes

- Bound to the governance/gateway project and enforced by Cloud IAP (`roles/iap.egressor` granting `iap.googleapis.com/resources.egressViaIAP`).
- **Default-deny with deny-overrides:** nothing passes unless an allow rule matches, and an explicit deny rule always overrides an allow rule.
- **Verified UAP CEL attributes:**

- **MCP tool call metadata:** `api.getAttribute('iap.googleapis.com/mcp.toolName', '') == 'get_order'` and `api.getAttribute('iap.googleapis.com/mcp.tool.isReadOnly', false)` (alongside registry tool descriptors `destination.agent_registry.mcp_server.tool.name` and `.annotations.read_only_hint` / `destructive_hint`).
- **Registered Agent & MCP targets:** `destination.agent_registry.agent.name` and `destination.agent_registry.mcp_server.name`.
- **Unregistered external endpoints:** `destination.unregistered.host == 'api.partner.com' && destination.unregistered.path.startsWith('/v1/')` (plus `destination.unregistered.method`).
- **Best practices:** roll out in dry-run first (`iamEnforcementMode: "DRY_RUN"`, then omit the field to enforce; see §08), allowlist explicit tool names (no wildcards), consolidate rules inside CEL expressions so you stay within the **4-** `AuthzPolicy` per-gateway limit, and audit denials regularly.

4-step runtime evaluation order & worked `roles/iap.egressor` allow-list

Every outbound call from an Agent Runtime workload through Agent Gateway executes four sequential checks:

1. **SPIFFE Agent Identity** mints the short-lived X.509 mTLS + DPoP-bound credential for the calling agent.
2. **Principal Access Boundary (PAB)** verifies that the target resource sits inside the principal set's eligible resource ceiling.
3. **Gateway** `REQUEST_AUTHZ` **(Cloud IAP + IAM UAP)** evaluates header-level `roles/iap.egressor` bindings and deterministic CEL predicates.
4. **Gateway** `CONTENT_AUTHZ` **(**`FULL_DUPLEX_STREAMED`**)** streams request and response bodies through **Model Armor** and **Semantic Governance** (a custom Service Extensions callout engine can attach at either `REQUEST_AUTHZ` or `CONTENT_AUTHZ`).

Worked declarative UAP allow-list permitting coordinator `coord-a` to invoke only peer A2A `agent-b` and read tool `get_order` on `mcp-c`:

```
{ "bindings": [{
    "role": "roles/iap.egressor",
    "members": ["principal://agents.global.org-123456.system.id.goog/resources/aiplatform/projects/111/locations/us-central1/reasoningEngines/coord-a"],
    "condition": { "title": "AllowPeerBAndToolCOnly",
      "expression": "destination.agent_registry.agent.name.endsWith('/agents/agent-b') || (destination.agent_registry.mcp_server.name.endsWith('/mcpServers/mcp-c') && api.getAttribute('iap.goo
    } }] }
```

Semantic Governance (Preview)

- **LLM policy engine running in your VPC** that evaluates each proposed tool call and `load_skill` action against two checks: 1. **Alignment with user intent** (enabled by default unless `sgpUserIntentMatching: DISABLED`): does this call genuinely serve what the user asked? 2. **Natural Language Constraints (NLCs)** (up to 5,000 chars each; inspected MCP/tool descriptions up to 1,000 chars): business rules such as `"refunds may not exceed $75 without escalation"`.
- **Verdict:** returns `ALLOW` or `DENY` with an audit `rationale` (replaceable by a custom denial message `⩽ 1,000` chars). When multiple constraints apply, **strict deny-overrides** means any single `DENY` blocks the action.
- **Complements, never replaces, UAP:** Semantic Governance catches **context poisoning / indirect prompt injection** where an allowed tool is called with hijacked intent, while deterministic UAPs and code checks remain mandatory for hard entitlement boundaries.

Need	Primary control
Hard allow/deny by registered agent, MCP tool name, read-only hint, or external host/path	**IAM UAP** (`REQUEST_AUTHZ`, deterministic)
Resource-hierarchy blast-radius cap across org/folder/project	**PAB** (deterministic ceiling)
"Does this allowed tool call or skill load match the user's actual intent or NLC rule?"	**Semantic Governance** (`CONTENT_AUTHZ`, backed by deterministic checks/HITL for hard limits)
Custom deterministic header or payload authorization logic	**Service Extensions** custom authorization engine

MCP risks mapped to controls

Risk	Attack	Controls
Tool poisoning / rug pull	A tool description, or a server update after approval, carries hidden instructions the model follows	Register only vetted servers in Agent Registry; gateway default-deny for unregistered destinations; UAP names each allowed tool (`iap.googleapis.com/mcp.toolName`). Model Armor on the gateway doesn't screen `tools/list`, so review tool metadata (and keep descriptions `⩽ 1,000` chars for SGP) at registration
Over-broad scopes	An auth provider or role grants far more than the tool needs, so any misuse has a large blast radius	Request minimal OAuth scopes per provider; grant `roles/agentidentity.user` only on the providers each agent needs; UAP allow on `iap.googleapis.com/mcp.tool.isReadOnly`; PAB
Indirect prompt injection via tool output	A ticket, email or web page returned by a tool contains instructions that steer the next tool call	Model Armor (`CONTENT_AUTHZ`) screens `tools/call` responses; Semantic Governance blocks calls that don't match the user's request. Streamable HTTP/SSE streams aren't screened
Data exfiltration	The agent is steered to send sensitive results to an attacker-controlled host or tool	Gateway default-deny plus UAP on `destination.unregistered.host` / `.path`; Model Armor SDP on outbound arguments; VPC-SC with egress through your VPC

5-layer defense-in-depth against indirect prompt injection & tainted sessions

Input-only filters fail when an agent ingests untrusted external text mid-turn (inbound emails, scraped pages, RAG chunks, or third-party MCP payloads). Stack five independent controls:

1. **Inline tool & RAG output screening (**`Model Armor`**):** Inspect every inbound tool/RAG payload via **Agent Gateway** `CONTENT_AUTHZ` or `sanitizeModelResponse` / `after_tool_callback` before injecting it into LLM context.
2. **Semantic Governance Policies (**`SGP`**,** `Preview`**):** Enforce **NLCs** (`⩽ 5,000` chars) and user-intent alignment at `CONTENT_AUTHZ` (`sgpEnforcementMode: "DRY_RUN"` → enforce) to block hijacked outbound tool calls when a session ingested untrusted external documents.
3. **Skill Governance (**`govern-agent-skills`**,** `Preview`**):** Intercept dynamic `load_skill` actions so tainted sessions cannot escalate privileges by loading sensitive skills on demand.
4. **Taint-isolated reader subagents:** Delegate untrusted document parsing over A2A to a sandboxed specialist whose **PAB + UAP** strip all mutating and egress tools, returning only schema-validated structured JSON to the coordinator.
5. **Deterministic HITL confirmation:** Require out-of-band human sign-off (`LongRunningFunctionTool` / `RequestInput` / `require_confirmation=True`) on irreversible financial, IAM, or data-deletion actions.

06. Content Safety: Model Armor, SDP & Safety Filters

Model Armor

Filter	Detects	Notes
Responsible AI	Hate, harassment, sexually explicit, dangerous (CSAM always on)	Confidence: `HIGH`, `MEDIUM_AND_ABOVE`, `LOW_AND_ABOVE`. The floor settings console defaults to Medium and above when left blank
Prompt injection & jailbreak	Attempts to override instructions, in prompts and in responses	Inputs under 3 words return `NO_MATCH_FOUND`. In a floor setting update that omits the level, it defaults to `LOW_AND_ABOVE`
Sensitive Data Protection	**Basic:** six fixed infoTypes (credit card, US SSN, financial account, US ITIN, Google Cloud credentials, Google Cloud API key), inspect only, no SDP templates. **Advanced:** an SDP **inspect** template plus an optional **de-identify** template	Redaction needs advanced mode. Only infoTypes named in *both* templates are de-identified. SDP templates in another project need `roles/dlp.user` and `roles/dlp.reader` for the Model Armor service agent. Uses SDP match likelihood, not the confidence enum
Malicious URL detection	Phishing/malware links	Only the first 256 URLs in a payload are screened
Document screening	Text in PDF, CSV, TXT, Word, PowerPoint and Excel files	Files over 4 MB are skipped; files under 69 bytes are rejected. Direct API or the Gemini Enterprise integration only; other integrations scan text
Image screening (**Preview**)	Text in images (OCR); visual content through the advanced SDP filter	JPEG, PNG, BMP; one image per request, 4 MB max; `us` and `eu` multi-regions only

- **Built-in limits:** each prompt or response is screened as a single turn with no conversation memory; Base64, hex, URL-encoded or encrypted text is not decoded; audio and video are not supported. Multi-turn or encoded attacks need other layers (Semantic Governance, HITL).
- **Templates** hold filter configs. Enforcement is **Inspect only** or **Inspect and block**.
- **Floor settings** set a minimum baseline at org, folder or project level that every template must meet.
- Best practice: separate **input** and **output** templates; start at High/Medium; run inspect-only to tune; then block.
- Model-agnostic and cloud-agnostic; findings appear in **Security Command Center**.
- Regional endpoints from a VPC need a PSC endpoint. Document (PDF) screening is via the API, not the gateway integration.

On Agent Gateway:

- Templates must be in the **same region** as the gateway.
- Ingress roles `roles/modelarmor.calloutUser` + `roles/modelarmor.user` for the Reasoning Engine service agent. Egress: the same roles for the Service Extensions service agent.
- **Redaction** requires an advanced SDP de-identify template **and** `INSPECT_AND_BLOCK`. If multiple detectors fire, the whole payload is blocked. Logs are redacted either way.

- Ingress screens ADK `streamQuery`. Egress screens A2A SendMessage/AgentCard, MCP `tools/call`, `prompts/get`, tool-execution errors, and OpenAI-compatible LLM payloads. It does **not** screen `tools/list`, `resources/*`, notifications or MCP Streamable HTTP/SSE streams.

Model Armor Floor Settings vs. Templates

Floor settings solve for: dozens of teams writing their own templates, where one team quietly switching off a detector reopens a hole the security team closed.

Dimension	Floor settings	Templates
Purpose	Minimum baseline (*template conformance*) and, at project level, *inline enforcement* on Gemini and Google Cloud MCP servers	The filter set and enforcement type for one workload or integration point
Who manages	Central security team	Application or platform team
IAM role	`roles/modelarmor.floorSettingsAdmin` (Model Armor Floor Setting Admin)	`roles/modelarmor.admin` on the project that holds the template
Scope and resource	`organizations/ORG_ID`, `folders/FOLDER_ID` or `projects/PROJECT_ID`, always at `locations/global/floorSetting`	A regional resource, `projects/PROJECT_ID/locations/LOCATION/templates/TEMPLATE_ID`
How it is set	Console at project level only (inherit, custom or disable). Org and folder need the API, gcloud or Terraform `google_model_armor_floorsetting`	Console, gcloud, REST or client libraries
Inheritance	Org applies to everything below; folder to its projects; project to itself. On conflict the **lower level wins** (project over folder)	Not inherited; referenced explicitly
Enforced by	The Model Armor API at template create/update; inline on Gemini `generateContent` and Google Cloud MCP traffic	Callers of `sanitize*`, Agent Gateway, Gemini Enterprise, Agent Runtime, Service Extensions, Apigee
Enforcement type	Set per integrated service (Agent Platform, Google MCP server)	`enforcement_type` in `template_metadata`
Effect takes hold	A few minutes after a change	On next request

When a template is weaker than the floor:

- **New or edited template:** the create or update call fails; you can't save a template less strict than the effective floor.
- **Template that predates the floor (or a raised floor):** it keeps working, and Security Command Center (Premium or Enterprise tier) raises a **high-severity finding** naming the floor, the template and the gap (missing filter, or confidence below the minimum).
- **SDP is not checked:** floor settings don't validate a template's Sensitive Data Protection config, so an org-wide redaction rule can't be forced through conformance.

> **EXAM TRAP.**
> "Lower level wins" cuts both ways. A project admin holding `roles/modelarmor.floorSettingsAdmin` can choose **Disable** and drop the inherited floor for that project (API flag `enableFloorSetting Enforcement=false`). Grant the role at org or folder level to the central team only, and restrict it at project level.

Inline protection for Gemini and Google Cloud MCP servers

Project floor settings can screen traffic with no code change by picking *integrated services*:

Integrated service	What is screened	Key behaviors
Agent Platform (Gemini)	Gemini API `generateContent` prompts and responses in the project, including calls through the Google Gen AI SDK or Firebase AI Logic	Grant `roles/modelarmor.user` to `service-PROJECT_NUMBER@gcp-sa-aiplatform.iam.gserviceaccount.com`. A request can name its own `promptTemplateName` / `responseTemplateName` instead. No documents or images. De-identified text is not returned to Gemini; an SDP match under `INSPECT_AND_BLOCK` becomes a block. Floor settings use the Stable filter version
Google MCP server (Preview)	`tools/call`, `prompts/get` and tool-execution errors on supported Google and Google Cloud remote MCP servers	Unsupported servers aren't covered. Uses filter version v1, which you can't override. If the agent and the MCP server sit in different projects, each project's floor runs, so Model Armor is called twice
Template creation & update	Template conformance only	Rejects weaker templates; no traffic screening

- **Fail-open on Gemini:** if Model Armor is unavailable in the region, unreachable, or errors, Gemini processes the request unscreened. Configuration errors (permissions, quota) are still reported under `INSPECT_AND_BLOCK`. For guaranteed screening, also screen in the app or at the gateway.
- **Prompt-injection tip for MCP:** skip the prompt injection and jailbreak filter on MCP floors unless the tool traffic carries natural language.

```
gcloud model-armor floorsettings update \
  --full-uri='projects/PROJECT_ID/locations/global/floorSetting' \
  --enable-floor-setting-enforcement=TRUE \
  --add-integrated-services=GOOGLE_MCP_SERVER \
  --google-mcp-server-enforcement-type=INSPECT_ONLY \
  --enable-google-mcp-server-cloud-logging
```

Rollout: inspect only, then block

Mode	Detection	Traffic	Use for
`INSPECT_ONLY`	Logged to Cloud Logging	Passes unchanged	Measuring false positives, monitoring new attack patterns, compliance evidence
`INSPECT_AND_BLOCK`	Logged	The caller (the enforcement point) blocks the prompt or response; redaction applies where the integration supports it	Production protection

1. **Enable logging first:** `INSPECT_ONLY` produces nothing useful without Cloud Logging.
2. **Start conservative:** separate input and output templates; RAI at `HIGH`; prompt injection at `HIGH` or `MEDIUM_AND_ABOVE` (`HIGH` for Gemini Enterprise to avoid false positives).
3. **Replay known traffic:** run known-good and known-bad prompts and record the false-positive baseline.
4. **Read the verdicts in logs:** check `SanitizeOperationLogEntry` records; a model refusal in the response body is not a Model Armor block.
5. **Switch to block:** move to `INSPECT_AND_BLOCK`; lower a threshold only for the one category that is under-protected.

> **ANTI-PATTERN.**
> Turning on `LOW_AND_ABOVE` across all RAI categories on day one. The docs flag it as high false-positive risk for general content; reserve it for categories where a miss costs more than a wrong block.

Model Armor logging, traces and findings

Signal	Default	Contents	Notes
Processing	Stateless	Prompts and responses handled in memory and discarded	Nothing is retained unless you enable logging
Template logging	Off	`log_template_operations` (template CRUD); `log_sanitize_operations` (full prompt and response text plus verdicts)	Sanitize logs hold raw user data; route them to an access-controlled sink. Anyone with `roles/logging.privateLogViewer` can read them
Floor setting logging	Console: on for Agent Platform, off for MCP	`--enable-vertex-ai-cloud-logging`, `--enable-google-mcp-server-cloud-logging`	Detection logs land under `modelarmor.googleapis.com/detection`. With residency needs, create a regional log sink *before* enabling
Audit logs	Admin Activity on	Template and floor setting changes	See §08
Cloud Trace (Preview)	When tracing is set up for the agent	Parent span `apply_guardrail "Google Cloud Model Armor"` with `Request Path` and `Response Path` children; attributes include `gen_ai.security.decision.type` (`modify` means redacted), `gen_ai.security.policy.id`, `gcp.modelarmor.violations`, `gcp.modelarmor.sdp.info_types`	Viewed from Agent Registry (agent, Traces tab) or Trace explorer; needs `roles/cloudtrace.user` and `roles/logging.viewer`. Separates guardrail latency and blocks from model errors
SCC findings	Premium or Enterprise tier	High-severity floor setting violations	Complements the AI Protection view in §08

Model Armor configuration patterns

Pattern	Floor settings	Template	Outcome
Org baseline, team redaction	Org: prompt injection and malicious URL enabled, RAI minimums	Payments agent: input and output templates with advanced SDP (inspect + de-identify), `INSPECT_AND_BLOCK`, bound to Agent Gateway in the gateway's region	No team can drop below the baseline; card numbers are masked on gateway traffic
Staged tightening	Staging project: custom floor with the tighter thresholds, Agent Platform service set to Inspect only (inline inspection is configured per project, not per folder)	Existing templates unchanged	Logged would-be blocks show false-positive rates before production moves to block
Zero-code MCP guard	Project: Google MCP server service, malicious URI and RAI dangerous at `MEDIUM_AND_ABOVE`, logging on	None required	Calls to supported Google Cloud MCP servers are screened even if a developer attaches no template
Gemini Enterprise assistant	Optional	Template bound to the Gemini Enterprise app	Violating turns are blocked; SDP hits block rather than mask, since this integration doesn't de-identify

Sensitive Data Protection (formerly DLP)

- infoType detection, inspection and de-identification (masking, tokenization) via templates.
- Use it inside Model Armor (advanced mode) for agent traffic, or directly to scan and de-identify data before it reaches a corpus or memory.

SDP de-identification methods compared

Method	SDP config	Reversible	Referential integrity	Format kept	Key	Best agent use
Replace with infoType	`ReplaceWithInfoTypeConfig`	No	No: every email becomes the same `[EMAIL_ADDRESS]` label	No	None	Free text bound for RAG chunks, logs and prompts where who-is-who doesn't matter
Redaction	`RedactConfig`	No	No	No	None	Dropping a value entirely; the model loses even the hint that something was there
Character masking	`CharacterMaskConfig`	No	No	Length kept; chosen characters stay visible	None	Support agents confirming the last digits of a card or account
Crypto hash	`CryptoHashConfig`	No	Yes	No (fixed-length hashed surrogate)	Yes	Stable join keys when nobody may ever re-identify
Deterministic tokenization	`CryptoDeterministicConfig` (AES-SIV, optional surrogate infoType)	Yes, via `content.reidentify` with the same key	Yes	No; any input accepted	Yes	Multi-turn, multi-tool flows that must keep customer A and customer B apart; an authorized back end re-identifies
Format-preserving encryption	`CryptoReplaceFfxFpeConfig` (FFX mode)	Yes	Yes	Same length and alphabet (2 to 95 characters)	Yes	A tool or legacy schema validates length or character set
Date shifting	`DateShiftConfig`	No	Same shift per context (e.g., per patient), so sequence and duration hold	Still a valid date	Optional, for consistent shifts	Clinical or financial timelines where intervals matter

- **Key choice & KMS separation:** a Cloud KMS-wrapped key is the preferred option (a transient key is discarded per request so tokens cannot be matched across calls). **Exam trap:** never grant the Cloud KMS unwrap / `content.reidentify` permission to the agent's own runtime IAM identity, or a prompt-injected agent can re-identify PII itself; restrict re-identification to a privileged downstream backend tool.
- **FPE, co-reference & date-interval caveats:** `ReplaceWithInfoTypeConfig` collapses distinct people into identical `[PERSON_NAME]` tags (breaking multi-entity co-reference). `CryptoReplaceFfxFpeConfig` is slower, capped at 2 to 95 characters, and **destroys temporal intervals on dates**; use `CryptoDeterministicConfig` for general entity joins and **`DateShiftConfig` scoped to an entity/patient context key** whenever downstream reasoning computes days elapsed between events.
- **Structure:** infoType transformations act on findings detected in free text; record transformations act on named columns of a table or CSV.

De-identify at the data layer

Once raw values are embedded, stored in sessions or copied into telemetry, finding and deleting them is slow and incomplete. Clean data before each storage boundary:

Boundary	Risk if raw data lands	Control	Constraints
Before indexing (RAG Engine, Agent Search, vector stores)	Retrieval can surface embedded PII to anyone allowed to query the corpus	Land files in a quarantine bucket; run an SDP inspection job with a de-identify action (console: *Make a de-identified copy*) that writes to a clean bucket; index only the clean bucket	Output bucket must differ from the input bucket; CSV, TSV, text and image file groups
Gemini Enterprise connectors and uploads	Sensitive entities from SaaS sources appear in answers; sensitive files get uploaded	SDP content policy (`projects/PROJECT_ID/locations/LOCATION/contentPolicies/POLICY_ID`) set on the connector, the assistant or Gemini Notebook Enterprise	Blocks content rather than masking it. Covers third-party, custom MCP and Workspace connectors, not older Cloud Storage, BigQuery or website data stores. Not in the Business edition. The Gemini Enterprise service account needs `roles/dlp.user`; a policy from another project must sit in the same VPC-SC perimeter
Before session and Memory Bank writes	A pasted SSN stays in session events and can be distilled into a long-lived memory	De-identify upstream (Model Armor advanced SDP de-identify on the gateway, or in the front end) so the raw value never becomes an event. ADK `before_model_callback` / `after_tool_callback` can also scrub what the model sees and what tools return	Use deterministic tokens when the agent still has to tell entities apart
Before log and trace export	Prompts, tool arguments and responses copied into spans, logs, prompt archives in Cloud Storage and BigQuery	`ADK_CAPTURE_MESSAGE_CONTENT_IN_SPANS=false` keeps PII out of span attributes. `OTEL_INSTRUMENTATION_GENAI_CAPTURE_MESSAGE_CONTENT=EVENT_ONLY` still records prompts, as log events, so restrict and de-identify those logs and the prompt and response bucket before analysts query them	Model Armor sanitize logs also hold raw text (see the logging table above)

- **Anti-pattern (prompt-only privacy):** a system instruction such as "never repeat card numbers" is not a control. The raw value is still indexed, stored and logged.

Gemini safety filters (model level)

- Non-configurable filters (e.g., CSAM, PII) always apply.
- Configurable harm categories: harassment, hate speech, sexually explicit, dangerous content.
- Thresholds: `BLOCK_LOW_AND_ABOVE`, `BLOCK_MEDIUM_AND_ABOVE`, `BLOCK_ONLY_HIGH`, `BLOCK_NONE`, `OFF`. Method: `SEVERITY` (default) or `PROBABILITY`.
- Per the docs, newer Gemini models default to `OFF` for configurable filters (verify for your model version).

Need	Gemini safety filters	Model Armor
Scope	One Gemini call	Any model, any cloud, plus agent traffic via the gateway
Prompt injection / jailbreak	×	✓
Sensitive data / redaction	× (beyond built-in)	✓ (SDP)
Malicious URLs	×	✓
Central org baseline	×	✓ floor settings

07. Network & Data Protection

VPC Service Controls

- Inside a perimeter, Agent Runtime keeps access to Google APIs, **public internet is blocked**, and Agent Runtime `max_instances` is capped at **`100`** (versus `1,000` outside VPC-SC). For controlled outbound egress, combine a PSC interface + RFC 1918 proxy VM + Cloud NAT.
- **DNS VIP rule (`restricted.googleapis.com` vs. `private.googleapis.com`):** Map Cloud DNS peering for `googleapis.com.` to **`restricted.googleapis.com`** (**`199.36.153.4/30`**) so every API call traverses VPC-SC enforcement. Routing general Google APIs to **`private.googleapis.com`** (**`199.36.153.8/30`**) allows non-VPC-SC APIs and bypasses perimeter enforcement.
- **Developer Knowledge MCP VPC-SC exception:** Google's remote documentation MCP server (`https://developerknowledge.googleapis.com/mcp`, public English docs only) does **not** support the `restricted.googleapis.com` VIP; add a specific host DNS override routing only `developerknowledge.googleapis.com` via **`private.googleapis.com`**.
- Without VPC-SC the Google-managed tenant keeps default internet access, yet **cannot reach your private VPC or on-prem** without a PSC interface.

- Deploys may need **ingress rules** for the Reasoning Engine service agent: pickle deploys need Cloud Storage + Artifact Registry; source deploys need Artifact Registry; bring-your-own-container needs none.
- Agent Identity APIs (`agentidentity.googleapis.com` , `agentidentitycredentials.googleapis.com`) support the `restricted.googleapis.com` VIP, and agent identities can appear in VPC-SC ingress/egress rules (`DRY_RUN` perimeter mode logs simulated denials before enforcement; live violations emit **Policy Denied** audit logs).

Private Service Connect, `AgentConnectivityTemplate` & private model endpoints

- **PSC interface (`PSC-I` + `AgentConnectivityTemplate` , `GA`):** Private zero-public-IP egress from the Google-managed Agent Runtime tenant into your Shared VPC, Cloud Interconnect, or multicloud network via a `networkAttachment` (`/28` minimum subnet; **cannot be changed without recreating the agent**, and PSC has **no transitive VPC peering** without explicit routing). When paired with Agent Gateway (`AGENT_TO_ANYWHERE`) inside VPC-SC, set `vpcEgress: ALL_TRAFFIC` (setting `vpcEgress: PRIVATE_RANGES_ONLY` lets public destinations bypass the customer VPC and violates zero-exfiltration):

```
name: projects/123456/locations/us-central1/agentConnectivityTemplates/strict-vpcsc
accessPath: AGENT_TO_ANYWHERE
egressNetworkConfig:
  networkAttachment: projects/123456/regions/us-central1/networkAttachments/agent-psc-na
  dnsPeeringConfig: { domain: "googleapis.com." } # Resolves to restricted.googleapis.com (199.36.153.4/30)
  vpcEgress: ALL_TRAFFIC
```

- **PSC endpoint:** Private *ingress* from your VPC to regional Google APIs (such as Model Armor or sandboxes).
- **Private model endpoints (`GA`):** When compliance requires strict customer-VPC compute residency and zero public VIP traversal for open-weight models (`Gemma 4` , `Llama`) or partner endpoints, serve them via **Model Garden** `PrivateEndpoint` (PSC NEG) or a private **GKE** GPU/TPU `vLLM` cluster inside the VPC-SC perimeter.

CMEK

- Set `encryption_spec` at creation, using a **single-region key in the same region. Can't be changed after deploy.**
- Grant `roles/cloudkms.cryptoKeyEncrypterDecrypter` to the `gcp-sa-aiplatform-re` service agent.
- Not covered by your key: metadata such as display names, descriptions, service-account emails, method names and env vars, so don't put secrets there.
- Security controls such as CMEK aren't supported for preview models.

Data residency

Endpoint	At-rest data	ML processing
Regional (e.g., `us-central1`)	In region	In region
Jurisdictional multi-region (`us` , `eu`)	In jurisdiction	Within jurisdiction (EU excludes UK/CH)
Global	In chosen location	**No residency guarantee**

DECISION RULE.
For a strict residency requirement, use a regional or jurisdictional endpoint and block the global endpoint with org policy.

08. Audit, Observability & Threat Detection

Cloud Audit Logs for Agent Platform

Log type	Default	Captures	Viewer role
Admin Activity	Always on, can't be disabled	Config/metadata changes (create/delete/patch resources, `setIamPolicy`)	`roles/logging.viewer`
Data Access	**Off: must be enabled**	Admin read, data read/write (e.g., model/endpoint usage)	`roles/logging.privateLogViewer`
Policy Denied	On	Requests denied by a security policy (e.g., VPC-SC)	`roles/logging.viewer`
System Event	Always on	Google-initiated changes	`roles/logging.viewer`

- UAP `DRY_RUN` results land in Cloud Audit Logs, which makes them a safe policy-tuning loop.
- Agent Gateway emits OpenTelemetry traces and access logs (Agent Observability), so you can trace *which tool* was called and *with what verdict*.

Agent Gateway monitoring signals

Signal	Where	What it holds
Gateway request log	Resource `networkservices.googleapis.com/Gateway` , log `gateway_requests`	`httpRequest` (URL, status); `jsonPayload.agentGatewayInfo` with `mcpInfo` (MCP method and tool name) and `agentRegistry Resource` (matched registry entry); `authzPolicyInfo` result (`ALLOWED` / `DENIED`); `serviceExtensionsInfo` (which extension handled the call); `trace` . Dry-run traffic is logged too
IAP decision	`cloudaudit.googleapis.com/data_access` , service `iap.googleapis.com`	`authorizationInfo[].granted` ; `authenticationInfo.principalSubject` (the agent's SPIFFE or `principal://` ID); resolved resource; label `iap.googleapis.com/audited_resource_name` reads `unregisteredResource` for unregistered hosts; `metadata.iam EnforcementMode`
Model Armor	Sanitize and detection logs (`SanitizeOperation LogEntry`); Agent Platform Security tab (Preview)	Matched filter and verdict; total, flagged and blocked interaction counts
Semantic Governance	Log `semantic-governance-policy`	Overall `verdict` ; `evaluations[]` with `toolName` , `verdict` , `rationale` ; `token_usage`
Agent output	`reasoning_engine_stderr` / `reasoning_engine_stdout`	Startup failures and the agent's view of a denied call
Observability dashboard	Agent Gateway page, Observability tab	Attempted authorizations, authorization failures, requests per second, egress logs including 403s and unregistered destinations. Needs the `_Default` log bucket upgraded to Observability Analytics
Cloud Trace (Preview)	Telemetry policy attached to the gateway	Gateway spans joined to agent and MCP spans through W3C `traceparent` ; compare with metric `networkservices.googleapis.com/agentgateway/total_latencies` to split gateway latency from destination latency

- **Tracing setup:** import a `telemetryPolicies` resource with `samplingRate` (the docs suggest 0.01 or 0.001 for busy production traffic) and `parentBasedSampling` to honor upstream sampling. Grant `roles/cloudtrace.agent` to the Compute Engine, Network Security and Agent Gateway service agents, and to the calling agent's identity, or the parent span is missing.
- **Topology:** Agent Registry and the Topology page draw agent and MCP traffic edges with the last hour of latency and error rate (App Topology API). An MCP server registered in Agent Registry is treated as a shared resource and shows no edges. The SCC data on these graphs is Preview.
- **Agent Anomaly Detection** (Preview, allowlist) analyzes OpenTelemetry logs and traces out of band, so it adds no request latency. Built-in detectors target agentic threats such as tool misuse, privilege abuse and cascading failures; findings go to SCC and metrics to Cloud Monitoring.

4-layer dry-run → enforce lifecycle, audit sinks & minimum IAM roles

Policy layer	Shadow / dry-run setting	Switch to enforce	Audit & telemetry sinks	Minimum author & audit IAM roles
IAM UAP (`REQUEST_AUTHZ` via Cloud IAP)	`iamEnforcementMode: "DRY_RUN"` in `AuthzExtension` metadata	**Omit/remove** `iamEnforcement Mode` field (no `"ENFORCE"` enum exists)	Cloud Logging (`networkservices.googleapis.com/Gateway` `IAP_AUTHZ=DENIED`) & Cloud Audit Logs (`iap.googleapis.com` , `proto Payload.metadata.iamEnforcementMode="DRY_RUN"`)	**Author:** `roles/iap.admin` + `roles/networksecurity.admin` **Audit:** `roles/logging.viewer` (`private LogViewer` for Data Access)
Semantic Governance (`Preview` , `CONTENT_AUTHZ`)	`sgpEnforcementMode: "DRY_RUN"` in extension metadata (allows traffic, logs verdicts)	**Omit/remove** `sgpEnforcement Mode` flag (preserve sibling keys on PATCH)	Cloud Logging (`semantic-governance-policy` : `verdict: DENY` + `evaluations[].rationale`) & Cloud Trace	**Author:** `roles/networksecurity.admin` + Agent Platform policy admin **Audit:** `roles/logging.viewer` , `roles/cloudtrace.user`
Model Armor (`CONTENT_AUTHZ`)	`enforcement_type: INSPECT_ONLY` on template or floor setting	Set `enforcement_type: INSPECT_AND_BLOCK`	Cloud Logging (`SanitizeOperationLogEntry` , `modelarmor.googleapis.com/detection`), Cloud Trace (`apply_guardrail` span), Security Command Center (SCC)	**Author:** `roles/modelarmor.admin` (templates) / `roles/modelarmor.floorSettingsAdmin` (floors) **Audit:** `roles/logging.viewer` , `roles/securitycenter.findingsViewer`

Policy layer	Shadow / dry-run setting	Switch to enforce	Audit & telemetry sinks	Minimum author & audit IAM roles
Principal Access Boundary (PAB) (`GA`)	Bind policy to a staging/canary test `principalSet` first and validate audit logs	Bind policy to the production Agent Identity `principalSet`	Cloud Audit Logs (`PERMISSION_DENIED` / Policy Denied)	**Author:** `roles/iam.principalAccessBoundaryAdmin` **Audit:** `roles/logging.privateLogViewer` / `roles/iam.securityReviewer`

```
name: iap-egress-authz
service: iap.googleapis.com
failOpen: false
metadata:
  iapPolicyVersion: "V2"
  iamEnforcementMode: "DRY_RUN"
```

- **Endpoint discovery:** run IAP in dry run, list the hostnames the agent actually reaches from the logs, register and authorize exactly those, then enforce by removing `iamEnforcementMode`.
- **Metadata trap:** a PATCH with `updateMask=metadata` replaces the whole map. Clearing `sgpEnforcementMode` that way also drops `sgpUserIntentMatching: DISABLED`, which turns intent matching back on.
- **Reading a 403 in dry run:** IAP doesn't block in dry run, so a 403 then comes from the gateway's egress proxy or from the destination.

Agent Gateway troubleshooting & exam discriminators

Symptom	Layer	Check	Likely cause and fix
403 "Egress request is not authorized" (`IAP_AUTHZ = DENIED`)	IAP (`REQUEST_AUTHZ`)	IAP audit entry with `granted: false`; `principalSubject`	Missing `roles/iap.egressor` (`egressViaIAP`) grant, **registry location mismatch** (policy bound to `global` Agent Registry while gateway/agent resolves a regional registry, or vice versa), CEL mismatch, or PAB exclusion. Test with a direct `principal://` binding to rule out `principalSet` sync delay
Quota error creating a 5th `AuthzPolicy` on one egress gateway	Gateway policy quota	Network Security API response	Hard limit of **4 custom `AuthzPolicy` resources** per egress gateway; consolidate tool/destination rules into combined CEL expressions inside existing policies
Schema validation error setting `iamEnforcementMode: "ENFORCE"`	Extension metadata	Deployment / PATCH error	Neither `iamEnforcementMode` nor `sgpEnforcementMode` accepts an `"ENFORCE"` enum; **omit the field** to enforce
403 and no IAP entry at all	Gateway, before IAP	Gateway log `authzPolicyInfo` = `DENIED`, `requestUrl`	Hostname not registered exactly (regional and `mtls` variants count as different hosts), or no authorization policy targets the gateway, so the extension never runs
403 with different text	Destination	Agent stderr	The tool rejected the call: fix destination IAM (e.g., `roles/iap.httpsResourceAccessor`, `roles/mcp.toolUser`) or the injected token
Auth error from a third-party tool	Auth manager	Agent logs	Agent lacks `roles/agentidentity.user` on the auth provider, or the user never completed 3LO consent
403s while Agent Runtime starts	Default-deny egress	Gateway logs, status 403	Google Cloud APIs the runtime calls at startup (Agent Platform, Resource Manager, Logging, Trace, Secret Manager hostnames) not registered or allowed: dry-run, register, grant
Startup failure or severe p99 latency spike after enabling Semantic Governance	Semantic Governance (`CONTENT_AUTHZ`)	Gateway latency metrics + `AuthzPolicy` predicate	`AuthzPolicy` intercepted raw **gRPC control/health traffic** without scoping the CEL predicate to MCP/A2A/HTTP tool paths
Multi-MB PDF/image A2A call fails or times out at gateway	`CONTENT_AUTHZ` buffer	Gateway request size / A2A payload	Inline base64 A2A `Part` blobs choke `CONTENT_AUTHZ` streaming inspection; store files in Cloud Storage via ADK **`GcsArtifactService`** and pass `gs://` URIs
"Failed to convert project number to project ID" at startup	Agent identity roles	`reasoning_engine_stderr`	Missing `resourcemanager.projects.get`: grant `roles/browser` alongside the default agent roles
Payload blocked	Model Armor	Sanitize log filter; span `gen_ai.security.decision.type`	Template threshold, or a floor setting stricter than the template
Tool call denied mid-turn	Semantic Governance	`rationale` in the policy log	Constraint too strict, or the call didn't match the user's request (intent matching is on by default); tune in dry run
No semantic verdict	Semantic Governance	Gateway log `serviceExtensionsInfo` lacks the extension	Engine not `ACTIVE`, gateway not bound, request path outside the policy's `when` condition, or a streamed model call (not supported)
TLS handshake failures	Egress connectivity	Agent logs	Destination uses a private CA or self-signed cert: set `tlsConfig` in the agent connectivity template (keep `PUBLICLY_TRUSTED_ROOTS` for Google APIs) and rebind it after changes. BYOC containers must trust the gateway's root CA
Binding fails or false "unregistered destination" denials	Project setup	Error text	`iap.googleapis.com` API disabled
Gateway spans appear as separate root traces	Trace propagation	Trace explorer	`parentBasedSampling` off, caller lacks `roles/cloudtrace.agent`, or the app doesn't propagate `traceparent`
Latency spike	Gateway vs. Destination	`agentgateway/total_latencies` against the span breakdown	Separate guardrail and policy time (Model Armor and Semantic Governance spans) from tool time before tuning

EXAM TRAP.
Logs explain *why* one call was denied; dry run shows *what would break* before you enforce. Enforcing without a dry-run window is a common cause of startup 403s on Agent Runtime.

Security Command Center: AI Protection

- AI asset inventory, the AI Security dashboard, vulnerability findings (Premium), threat detection, and Model Armor findings.
- **Agent Platform Threat Detection** for Agent Runtime (Preview, Premium). Event Threat Detection rules include:
- *IAM Anomalous Grant to Agentic Identity*
- *Agentic Identity Credential Used Outside of Google Cloud*
- *Sensitive Role Granted by AI Agent*
- Premium capabilities include detection of over-privileged agents and attack paths.

EXAM TRAP.
Audit logs *record*, and SCC *detects and alerts.* A question asking "be notified when an agent's credential is used from outside Google Cloud" points to SCC threat detection, not to enabling Data Access logs.

09. HITL & Governance at Scale

Human-in-the-loop approvals

- ADK **tool confirmation** (Experimental): `FunctionTool(fn, require_confirmation=True)`, or pass a function returning a bool (e.g., confirm only when `amount > 1000`).
- Advanced confirmation returns a structured payload (approve/modify). Remote approvals can arrive via the REST API, which suits async approval UIs.
- ADK graph workflows can pause on a request-input node until a human responds.
- Use HITL for **irreversible, high-blast-radius, regulated, or novel** actions. Use deterministic code for hard limits, and Semantic Governance/Model Armor for scalable screening. See **ResponsibleAI** and **ADK**.

Governing agent sprawl

Lever	Control
Inventory	Agent Registry (auto + manual registration); SCC AI asset inventory
Only approved destinations	Gateway default-deny + registry-backed UAPs
Only approved gateways	Custom org policy constraints on gateway association
Uniform content baseline	Model Armor floor settings at org/folder
Uniform reach	PAB on the org/project agent principal set
Residency	Org policy blocking the global endpoint
Detection	SCC AI Protection; Audit Logs
Lifecycle hygiene	Remove orphaned bindings on deletion; `principalSet` baselines for redeploys

Layered defense, end to end

1. **Identity:** Agent Identity with least-privilege roles + PAB.
2. **Credentials:** auth manager (3LO/2LO/API keys) + Secret Manager.
3. **Gateway:** registry-backed, default-deny UAPs (dry-run → enforce).
4. **Intent:** Semantic Governance on proposed tool calls.
5. **Content:** Model Armor input/output templates + SDP; Gemini safety filters.
6. **Network/data:** VPC-SC, PSC-I, CMEK, regional endpoints.
7. **Human:** tool confirmation for high-stakes actions.
8. **Audit/detect:** Cloud Audit Logs + SCC AI Protection.

10. Glossary

- **Agent Identity:** a per-agent SPIFFE-based principal (`GA`) with certificate-bound, auto-rotated (~24h) X.509 mTLS + DPoP credentials.
- **Auth manager:** Agent Platform's credential vault and OAuth broker (`GA`, `agentidentity.googleapis.com` `authProviders/`, superseding the deprecated Preview `iamconnectors.googleapis.com` API) for 3LO, 2LO, and API-key tool authentication.

- **Agent Policies:** Agent Platform's four-layer governance stack across Agent Gateway, Agent Registry, and IAM: IAM UAP (`REQUEST_AUTHZ`), Principal Access Boundaries (PAB), Semantic Governance (`CONTENT_AUTHZ`), and Model Armor (`CONTENT_AUTHZ`).
- **PAB:** Principal Access Boundary (`GA`), which caps the Resource Manager resources an Agent Identity `principalSet` is eligible to access (up to 500 rules per policy, 500 resources per rule; never grants access on its own).
- **UAP:** Unified Access Policy (`GA`), a default-deny IAM v3 allow/deny policy enforced by Cloud IAP (`roles/iap.egressor` / `egressViaIAP`) on Agent Gateway traffic (up to 4 custom `AuthzPolicy` resources per egress gateway).
- **`roles/iap.egressor` :** the IAM role granted with CEL conditions (`destination.agent_registry.*` , `iap.googleapis.com/mcp.toolName` , `iap.googleapis.com/mcp.tool.isReadOnly`) to authorize outbound gateway calls.
- **Semantic Governance:** an LLM policy firewall (`Preview` , `CONTENT_AUTHZ`) evaluating user-intent alignment and **Natural Language Constraints (NLCs)** (up to 5,000 chars per NLC, 1,000 chars per tool description) under strict deny-overrides semantics.
- **Floor settings:** the org/folder/project Model Armor minimum baseline across templates and inline services.
- **PSC-I:** Private Service Connect interface for private agent egress.
- **Auth provider:** an auth manager resource holding one integration's credentials (3LO, 2LO or API key); agents need `roles/agentidentity.user` on it.
- **Confused deputy:** a privileged component tricked into using its shared authority (such as a 2LO client or broad service account) for a caller who lacks permission.
- **`REQUEST_AUTHZ` / `CONTENT_AUTHZ` :** gateway authorization profiles that inspect headers only (IAP + UAP) or full streamed payloads (`FULL_DUPLEX_STREAMED` : Model Armor, Semantic Governance).
- **`GcsArtifactService` :** ADK's Cloud Storage artifact backend used to keep multi-MB binary files out of `session.state` and A2A inline base64 payloads by passing versioned `gs://` URIs.
- **`SECRETS_IN_ENVIRONMENT_VARIABLES` :** the Security Command Center vulnerability finding triggered when plaintext credentials are placed in deployment environment variables instead of Secret Manager or Auth Manager.
- **Token exchange (RFC 8693):** swapping one security token for another, as Workload Identity Federation's Security Token Service does.
- **Template conformance:** the floor settings check that rejects any Model Armor template weaker than the effective floor at create or update time.
- **Inline enforcement:** project floor settings applied directly to Gemini `generateContent` and Google Cloud MCP traffic, with no template reference in code.
- **`INSPECT_ONLY` / `INSPECT_AND_BLOCK` :** Model Armor enforcement types that log detections only, or log and return a block verdict.
- **Fail-open:** a control that lets traffic through unscreened when it is unavailable, as the Gemini integration does if Model Armor can't be reached.
- **Referential integrity (SDP):** the same input always maps to the same output under one key and context, so joins and entity tracking survive de-identification.
- **Format-preserving encryption (FPE):** reversible tokenization (`CryptoReplaceFfxFpeConfig`) that keeps the input's length and alphabet.
- **Surrogate infoType:** a label prefixed to a deterministic token so `content.reidentify` can find and reverse it.
- **SDP content policy:** a Sensitive Data Protection resource that Gemini Enterprise applies to connectors and uploads to block sensitive content.
- **`iamEnforcementMode` / `sgpEnforcementMode` :** authorization extension metadata keys that put IAP or Semantic Governance in `"DRY_RUN"` (log only); omitting/removing the key activates enforcement.
- **Telemetry policy:** the `telemetryPolicies` resource that turns on Cloud Trace sampling for an Agent Gateway.