Building secure multi-agent systems on Google Cloud

Authors

Anirudh Kannan

Christine Sizemore

Connor Herriford

Contributors

Alan Blount

Ashley Lin

Kanchana Patlolla

Melanie Lombardi

Mike Davis

Sean Leighton

Shaun Liu

Sita Lakshmi Sangameswaran

Vaibhav Katkade

Design

Tian Deng

Building multi-agent architectures requires strict defense in depth, treating every LLM and tool as a potential vector for a range of security threats, including but not limited to prompt injection or data exfiltration. Google Cloud's Gemini Enterprise Agent Platform features native security controls at various levels, including key capabilities specifically designed for agentic systems. These capabilities help enterprises scale AI and reduce time to production while enhancing security. This paper describes the practical application of these controls by detailing the process of building an example automated warranty claim system.

User journey

Filing a claim with a warranty claim system

User journey

Filing a claim with a warranty claim system

Imagine a customer whose premium smart appliance suddenly stops working. Frustrated, they access your company's support portal and fill out a single, free-text "Describe Your Issue" form. Instead of navigating complex drop-down menus to categorize their problem or waiting 48 hours for an email reply from a triage team, they simply type out their issue and hit submit.

Within seconds of submission, the AI-powered support system reads the description, understands the specific appliance issue, and extracts the provided serial number. Behind the scenes—without the customer having to dig up old receipts or wait for a human agent to manually check databases—the system verifies the warranty status.

Because the warranty is confirmed, the system automatically drafts a replacement order and generates a return shipping label with a premium carrier. To ensure accuracy and prevent automated fraud, this final fulfillment step includes a brief pause in an internal queue for a human support representative to review a concise, pre-validated summary and click "Approve."

Just minutes after submitting their web form, the customer receives an email that a replacement is on the way, turning a frustrating hardware failure into a nearly frictionless, loyalty-building experience.

How does this seamless experience actually work under the hood? That is where a secure multi-agent architecture comes in.

Architecture overview

The Warranty Claim System

Architecture overview
The Warranty Claim System

The Warranty Claim System is built on Gemini Enterprise Agent Platform. The goal of the Warranty Claim System is to automate the processing of customer warranty claims. This goal includes diagnosing customer issues, verifying products against purchase and warranty history, and executing fulfillment steps such as generating shipping labels for replacement parts.

The multi-agent system enforces a security boundary by splitting tasks across three personas:

Agent 1
Case Manager (orchestrator)

- **Runtime**: Agent Runtime with Agent Identity.
- **Purpose**: Serves as the diagnostic lead, that consumes the initial Pub/Sub trigger and categorizes the customer issue. Its primary function is to coordinate the workflow by initiating secure agent-to-agent (A2A) calls to specialized sub-agents.
- **Security boundary**: Strictly denied access to personal information or financial records.

Agent 2
BigQuery Agent (with Data Vault)

- **Runtime**: Agent Runtime with Agent Identity
- **Purpose**: Takes sanitized, deterministic inputs (serial numbers) to look up the free form warranty terms and returns only a simplified status like "Covered" to Agent 1. The agent also looks up specific PII (name and address) required by Agent 3.
- **Security boundary**: Delegates execution to a managed Google Cloud MCP Server for BigQuery. This agent is granted the exclusive IAM permissions required to query BigQuery transaction and warranty tables. The agent also looks up only the specific personal information (name and address) required by Agent 3. This agent is the only point of contact with BigQuery.

Agent 3
Logistics Liaison (executor)

- **Runtime**: Cloud Run. Deploying the Logistics Liaison on Cloud Run demonstrates how easily you can mix managed AI platforms with custom containerized environments. It's worth noting that you could deploy this agent directly on Agent Runtime, which now supports Bring Your Own Container (BYOC) and long-running operations of up to seven days. However, this paper will specifically show you how to bridge Cloud Run with Agent Runtime.
- **Purpose**: Handles the fulfillment phase.
- **Security boundary**: The agent is granted access to the specific personal information (name and address) required for fulfillment, to generate shipping labels through external Model Context Protocol (MCP)-enabled APIs. To mitigate risk, it includes a human-in-the-loop (HITL) trigger that automatically flags anomalous patterns to fraud specialists before any automated fulfillment occurs.

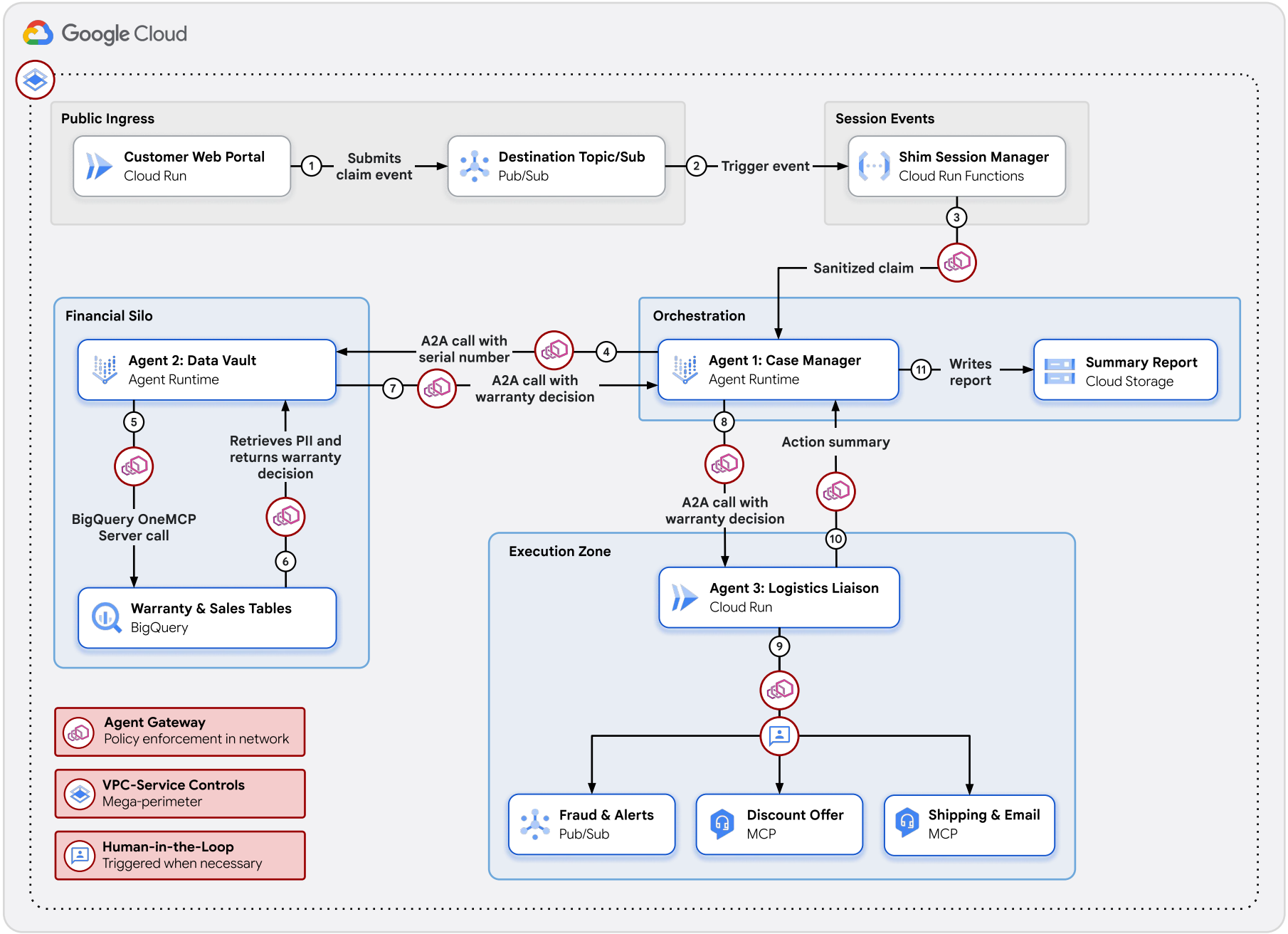

Figure 1. Warranty Claim System architecture

The Warranty Claim System architecture in Figure 1 depicts a sequence of events that can be broken into three phases. Each phase further describes the components of the architecture.

Phase 1

This phase handles the initial user interaction, ensuring the prompt is sanitized by Model Armor before it ever reaches any agent. The architectural components and sequence of actions in this phase are as follows:

1. **Claims submission**: A customer submits a warranty-claim event through the Customer Web Portal (running on Cloud Run). The web portal publishes this event data to a Pub/Sub destination topic.
2. **Event trigger**: The Pub/Sub subscription acts as an event trigger, invoking a Cloud Run Function (the "Shim Session Manager").
    a. Session initialization: The Shim Session Manager takes the raw payload and sends it to Agent Gateway. Agent Gateway establishes a secure session and prepares to route the traffic into Gemini Enterprise Agent Platform Runtime (Agent Runtime).
3. **Input sanitization**: Before routing to the Case Manager Agent, Agent Gateway passes the raw user prompt through Model Armor (integrated with Sensitive Data Protection). Model Armor performs a critical inline security scan, filtering for prompt injections, jailbreaks, and masking personal informationPII, passing only the sanitized claim data to the Case Manager Agent.

Phase 2

In this phase the Case Manager Agent delegates data lookup to the Data Vault Agent. By configuring strict ingress controls on the underlying runtimes, the agents are restricted from communicating peer-to-peer. Instead, all lateral agentic movement is deliberately routed through the Agent Gateway. The architectural components and sequence of actions in this phase are as follows:

4. **Categorize and delegate**: The Case Manager Agent receives the sanitized claim. If the claim is evaluated as legitimate, the Case Manager Agent must then verify purchase history and warranty coverage; it initiates a secure agent-to-agent (A2A) call to the Data Vault Agent.
5. **BigQuery lookup**: The Data Vault agent utilizes its exclusive IAM permissions to access sensitive data, and it queries data from the BigQuery Warranty & Sales tables. The Data Vault agent performs checks on serial numbers, purchase dates, etc. through the Google Cloud BigQuery MCP server.
6. **Status distillation**: The Data Vault Agent processes the raw data and distills it into a simple status (e.g., "Covered", "Not Covered", "Expired"). During this evaluation, the AI also flags any anomalies—for example mismatched details or suspicious claim histories—with a status of "Suspicious".
7. **Returns result**: The Data Vault Agent returns the distilled status, the anomaly evaluation, and only the specific personal information required for fulfillment back to the Case Manager Agent..

Phase 3

In the final phase, the Case Manager Agent orchestrates the execution of real-world actions through the Logistics Liaison Agent. The Logistics Liaison is a purpose-built deterministic service running on Cloud Run, focused solely on the mechanics of fulfillment. This phase includes HITL safeguards and comprehensive logging. The architectural components and sequence of actions in this phase are as follows:

8. **Logistics handover**: The Case Manager orchestrates the handover, passing the warranty status, fulfillment customer information, and any AI-generated anomaly flags to the Logistics Liaison.
9. **Action selection**: The Logistics Liaison Agent receives the Case Manager's directive and applies deterministic logic to route the execution:
    a. **Branch A (“Covered”)**: If the status is “Covered”, and no anomalies are flagged, the Logistics Liaison Agent triggers the Shipping & Email MCP tools to generate a premium return shipping label and draft a confirmation email, placing the action into an internal queue for a final HITL "Approve" click.
    b. **Branch B (“Expired”)**: If the warranty is "Expired", the Logistics Liaison Agent triggers the Discount Offer MCP tool to automatically send a discount offer on a replacement unit.
    c. **Branch C (“Suspicious”)**: If the payload from the upstream agents includes a "Suspicious" flag, the Logistics Liaison Agent triggers deterministic code, bypasses automated fulfillment and routes the claim to a Fraud & Alerts queue for immediate human review.
10. **Action summary return**: After the real-world action is completed (or flagged), the Logistics Liaison Agent returns an action summary back to the Case Manager Agent (e.g., "Replacement unit order ID generated: [ID], awaiting HITL approval").
11. **Summary report**: Upon completion of its orchestration tasks, the Case Manager agent compiles a summary report. This report encapsulates the key findings (e.g., issue category, warranty status "Covered") and actions initiated. This summary is then written by the Case Manager agent to a Cloud Storage bucket for business review, longer-term record-keeping, or potential analysis.

Note

As the Case Manager Agent completes the workflow cycle, system interactions, A2A communications, and fulfillment actions generate logs that are captured in Cloud Logging. Additionally, specific interactions with Google Cloud services, such as data access in BigQuery, are also recorded in Cloud Audit Logs.

Building and deploying a secure agent-based architecture

Building and deploying a secure agent-based architecture

To bring the user journey to life as a secure, agent-based architecture, we'll apply the following principles:

- Use secure agent design
- Build with secure frameworks
- Test and verify agent boundaries
- Implement a secure build pipeline
- Deploy to secure and scalable runtimes
- Define authentication and authorization policies

1. Use secure agent design

Security must be a primary consideration from the earliest stages of agent design. Taking a proactive stance is fundamental to building trustworthy AI and aligns with the principles of frameworks like Google's Secure AI Framework (SAIF). Our approach involves mapping AI-specific risks to the workflow. We divide our security strategy into two distinct layers: model controls (securing the underlying LLM's reasoning) and agent controls (securing the system's autonomous actions).

Here is how these SAIF principles are applied across the three agents in our Warranty Claim System

- **Securing the model** - Focuses on defending the underlying foundational model against manipulation, prompt injection, jailbreaking, and sensitive data exposure.
  - **Case Manager Agent (LLM)**: Handles raw, untrusted customer prompts, but only after they are pre-scrubbed by Model Armor. The LLM is programmed to extract intent but must not pass the raw conversational text laterally to other models.
  - **Data Vault Agent (LLM)**: Shielded from direct prompt injections as it is designed to accept only a sanitized, deterministic variable (e.g., a 12-digit serial number) from the orchestrator. To prevent exposure of personal information, the LLM is instructed to return only a simplified status (e.g., "Covered"), rather than raw database rows.
  - **Logistics Liaison (LLM)**: Receives strictly the minimum data required for fulfillment (e.g., Name, Address, and "Covered" status). This strict data minimization prevents the execution LLM from hallucinating unauthorized PII into external shipping labels.
- **Securing the agent** - Focuses on mitigating excessive agency, unauthorized tool access, cascading execution errors, and infrastructure vulnerabilities.
  - **Goal bounding**:
    - **Case Manager Agent**: Its system-level goal is strictly workflow routing. It cannot access the tools to execute real-world actions or query backend databases.
    - **Data Vault Agent**: Its goal is exclusively read-only data retrieval. It lacks the agency or tools to make routing decisions, execute fulfillment, or talk directly to the customer.
    - **Logistics Liaison Agent**: Its goal is strictly fulfillment execution. It is prohibited from querying the core database or overriding the Case Manager's original claim classification.
  - **Access boundaries**:
    - **Case Manager Agent**: Has no direct network access to internal databases or external APIs. Its only permitted network path is sending A2A handoff requests to Agent Gateway.
    - **Data Vault Agent**: Granted exclusive IAM access to the Google Cloud MCP Server for BigQuery. Data is completely blocked from public internet egress through a VPC-Service Controls perimeter.
    - **Logistics Liaison Agent**: The only agent that is permitted to communicate with external shipping and discount APIs. Its

egress is tightly locked down to pre-approved vendor URLs through a Secure Web Proxy (SWP).

- **Downstream risk**:
  - **Case Manager Agent**: The primary risk is misclassifying an issue. This is mitigated by the agent's lack of execution tools; a hallucination here only results in a routing error, not a financial loss.
  - **Data Vault Agent**: The primary risk is mass data exfiltration through indirect SQL injection. This risk is mitigated mechanically outside the LLM by the ADK BeforeToolCallback (for example, rejecting non-alphanumeric strings before the tool fires).
  - **Logistics Liaison Agent**: The primary risk is automated financial fraud (for example, generating unauthorized 100% discount codes). This is mitigated by mandating an asynchronous HITL interrupt through the ADK, requiring a human support representative to click "Approve" before the API actually fires.
- **Identity mapping**: At the core, Gemini Enterprise Agent Platform agents have an identity that is provisioned with Agent Identity. Agent identities are primary Google Cloud identities, like users. Although agents can run in service accounts, service accounts can be risky. Service accounts can be over-permissioned, abandoned, and accessed with reusable service account keys. Agents are provisioned with unique, mTLS-protected identities, and with signed, certificate-bound tokens. Agent identities are restricted to their authorized runtime, which can render compromised tokens useless. Anchoring access to the agent's lifecycle enforces surgical least-privilege and eliminates the security debt of residual identities.
  - **Case Manager Agent**: Operates using its own unique Agent Identity to authenticate its A2A requests at the Gateway.
  - **Data Vault Agent**: It uses its Agent Identity for secure A2A communication and to seamlessly authenticate to the Google Cloud MCP Server for BigQuery. It relies on the IAM policies bound to the agent's identity to enforce least-privilege database access.
  - **Logistics Liaison Agent**: Operates under a tightly scoped Workload Identity, granting it just enough privilege to invoke its specific MCP server and write fulfillment receipts to the centralized audit log. We choose a Workload Identity here as a means to showcase the flexibility within the GCP ecosystem to deploy across diverse runtimes.

2. Build with secure frameworks

The Warranty Claim System architecture security is enhanced by using the Agent Development Kit (ADK), running on Agent Runtime. The ADK provides a secure foundation for session management, tool integration, and identity management.

To work with ADK and its agent functionalities, install the `google-adk` package by following the instructions in the documentation. ADK offers several key features to build secure agents. To prevent "session bleeding" and enforce tenant isolation, the system utilizes the ADK's `session_id` and `user_id` primitives. These primitives ensure that data and operations for different sessions and users remain separated.

Furthermore, the ADK enhances security through its native tool-level authentication, which—combined with Agent Identity Auth Manager (Preview)—automates the complex lifecycle of OAuth and API key management. This approach reduces the need for hardcoded secrets and helps enforce the principle of least privilege.

The architecture also leverages ADK's callback mechanisms to ensure data integrity and protect against threats like prompt injection. For instance, Before and After tool callbacks are implemented for inter-agent data verification. These callbacks deterministically verify the data returned by the Data Vault Agent before it is consumed by the Logistics Liaison Agent.

To safeguard the connection to Google's managed MCP Server for BigQuery, a BeforeToolCallback is used within the Data Vault Agent. This

callback is an important step for preventing prompt injection. It validates inputs, for example, by ensuring that only a valid 12-digit alphanumeric serial number is passed to the Google Cloud MCP server:

```
import os
import asyncio
import vertexai
from vertexai.preview import reasoning_engines
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client

def validate_serial_number(tool_input: dict) -> dict:
    """Deterministic inline firewall before sending
payload to the Google Cloud MCP  Server."""
    serial = tool_input.get("serial_number", "")
    if not serial.isalnum() or len(serial) != 12:
        raise ValueError("Security Exception: Invalid
Serial Number Format detected.")
    return tool_input

async def execute_bq_query(serial_number: str) -> str:
    """Executes a query against BigQuery through the
managed MCP server."""
    # 1. Fire the BeforeCallback Security Validation
    validate_serial_number({"serial_number":
serial_number})

    # 2. Connect to the Google Cloud BigQuery MCP Server
    server_params = StdioServerParameters(
        command="npx",
        args=["-y", "@googlecloud/mcp-server-bigquery"]
    )

    # 3. Execute the tool through the standard MCP
protocol
    async with stdio_client(server_params) as (read,
write):
        async with ClientSession(read, write) as
session:
            await session.initialize()
            query = f"SELECT status FROM
`warranty.entitlements` WHERE serial =
'{serial_number}'"

            result = await session.call_tool(
                "bigquery_query",
                arguments={"query": query}
            )
            return result.content[0].text

# 4. Assign the secure tool to the Data Vault Agent
using ADK
entitlement_agent = reasoning_engines.LangchainAgent(
    model="gemini-2.5-pro",
    tools=[execute_bq_query],
)
```

In addition, we use confirmation primitives that facilitate HITL oversight, to enable Long-Running operations. Using confirmation primitives ensures that high-stakes actions are paused for explicit user approval or data verification, and resume upon confirmation.

```
# Configure the Logistics liaison Agent to halt for
human approval before shipping
logistics_agent = adk.Agent(
    name="logistics-liaison",
    tools=[shipping_label_mcp_tool],
    require_human_approval=["generate_shipping_label"]
)
```

In addition, we use confirmation primitives that facilitate HITL oversight, to enable Long-Running operations. Using confirmation primitives ensures that high-stakes actions are paused for explicit user approval or data verification, and resume upon confirmation.

```
# Configure the Logistics liaison Agent to halt for
human approval before shipping
logistics_agent = adk.Agent(
    name="logistics-liaison",
    tools=[shipping_label_mcp_tool],
    require_human_approval=["generate_shipping_label"]
)
```

3. Test and verify agent boundaries

Before an agent is committed to the build pipeline, we must test it to validate both its reasoning accuracy and its security boundaries. Because LLMs are non-deterministic, traditional unit testing isn't enough. We need to implement a methodical approach to agent evaluation, focusing on three core pillars:

- **Pillar 1: Agent Success & Quality (The Output)**: Using an "LLM-as-a-judge" (Pointwise) evaluator, we automatically score agent responses against a golden dataset of historical claims.
  - **Application**: Did the Case Manager correctly classify the hardware issue? Did the Data Vault precisely extract the serial number and return the correct binary status without hallucinating? Did the Logistics Liaison generate a valid JSON payload for the shipping API?
- **Pillar 2: Process & Trajectory (The Logic):** An agent's success depends entirely on its decision-making sequence. We have to parse the chain-of-thought traces to programmatically verify the agents are using the right tools in the right order.
  - **Application**: Did the Case Manager attempt to invoke the Logistics Liaison before verifying the warranty with the Data Vault? Did the Data Vault actually invoke the BigQuery MCP server, or did it try to hallucinate the warranty status from its base training data?
- **Pillar 3: Trust & Safety (The Guardrails)**: We build automated adversarial test suites that aggressively feed the system out-of-bounds inputs to ensure the IAM boundaries and ADK 'BeforeToolCallbacks' hold firm.
  - **Application**: Can the Case Manager be jailbroken through prompt injection? Does the Data Vault successfully reject SQL injection strings disguised as serial numbers? Does the Logistics Liaison's Semantic Guardrail correctly block attempts to generate unauthorized 100% discount codes?

Note

Evaluations can be performed using the Gemini Enterprise Agent Platform evaluation tool.

4. Implement a secure build pipeline

Our architecture hardens the software supply chain by embedding security throughout the CI/CD lifecycle, adhering to the principles outlined in Shifting left on security: Securing software supply chains. We address the distinct security needs of the two CI/CD supply chains for our agents:

- **Agent Runtime (Case Manager and Data Vault Agents)**: Since Google manages the underlying container infrastructure, the agent pipeline focuses on application-level integrity:

- **Secure artifact storage**: Custom-built agent components, container images, or private libraries that are used by the agents are securely stored and versioned in Artifact Registry. These practices ensure a trusted source for all agent-related artifacts. Artifact Analysis can also be enabled on repositories to scan for vulnerabilities.
- **Secure dependencies**: Leverages Assured Open Source Software (Assured OSS) to pull curated Google-secured Python packages, mitigating risks of supply-chain poisoning.
- **Secure staging & promotion**: Authentication is handled through Workload Identity Federation, eliminating the need for static keys. Agent configurations and prompts are staged in a hardened Cloud Storage bucket and seamlessly deployed through the Agent SDK.

- **Cloud Run (Logistics Liaison Agent)**: This agent runs on Cloud Run and requires the following container-focused supply chain controls:
  - **Verifiable builds & scanning**: Cloud Build generates images with SLSA Level 3 provenance, pushing them to Artifact Registry where Artifact Analysis continuously scans for known CVEs.
  - **Admission control**: Binary Authorization acts as the final gatekeeper at deployment, using strict admission criteria policies, including digitally signed attestations to ensure that malicious actors cannot deploy containers that have not met admission criteria policies.
  - **Example admission criteria policy**:
    - Was built by an authorized pipeline.
    - Has a verifiable SLSA provenance record.
    - Is verified free of "Critical" or "High" vulnerabilities by Artifact Analysis.
  - By verifying digitally signed attestations at the moment of deployment, this approach helps to ensure that even if an unauthorized actor gains access to anywhere in the supply chain, including Artifact Registry, they cannot deploy a malicious version of the warranty agents into the system.

5. Deploy to secure and scalable runtimes

The architecture utilizes a hybrid runtime strategy to balance managed simplicity with granular container control. The two runtimes used are:

- **Agent Runtime**: Serves as the primary managed runtime for the Case Manager and Data Vault Agent. It manages infrastructure scaling and platform-level security, allowing the orchestration layer to focus on agent logic.
- **Cloud Run**: Provides the runtime for the Logistics Liaison Agent. To harden this environment, we implement the defense-in-depth patterns described in the Cloud Run security design overview.

A detailed description of the secure runtime environment can be found in the Agent interaction and runtime section.

Deployment methodology: Independent, phased rollouts

To ensure system stability, we recommend that you roll out system agents in phases, rather than deploying the entire multi-agent system. Each agent must be deployed independently through its own CI/CD pipeline into a hardened non-production environment for integration testing. After it is validated against Agent Registry and Model Armor policies, utilize the native traffic-splitting capabilities of Cloud Run and Agent Runtime to execute gradual, canary rollouts (e.g., routing only 5% of customer claims to the new agent version) in the production environment, allowing for real-time anomaly detection before full cutover.

6. Define authentication and authorization policies

Managing access for AI agents requires moving beyond standard service accounts to more granular and secure workload identities. To do this, we leverage the Agent Identity model.

Agents are provisioned with a dedicated `AGENT_IDENTITY`, which is distinct from conventional Google Cloud service accounts. While

dedicated Google Cloud service accounts with fine-grained roles can still t is generally discouraged. The agent identity is directly tied to the agent's lifecycle, and therefore a more secure option for agentic authentication and authorization.

The code sample below demonstrates how to provision an Agent Identity using the ADK. For more information, refer to Use agent identity with Agent Runtime

```
# Example provisioning with ADK
remote_app = client.agent_engines.create(
  agent=app,
  config={
    "display_name": "running-agent-with-identity",
    "identity_type": types.IdentityType.AGENT_IDENTITY,
    "requirements": ["google-cloud-
aiplatform[adk,agent_engines]"],
    "staging_bucket": f"gs://"BUCKET _NAME",
  },
)
```

Finally, Principal Access Boundary (PAB) policies and explicit IAM deny policies are applied to Agent Identity. These controls strictly confine the agent's maximum possible reach to approved projects and resources. By adding restrictions, agents cannot access the broader Google Cloud organization, even if base permissions are misconfigured.

Note

While the Warranty Agent architecture does not use user-delegation, this is an important construct. User-delegation is an authorization model where the agent acts directly on behalf of the end-user rather than relying on its own Agent Identity or service account. This is facilitated by Agent Identity Auth Manager (Preview) which works by acquiring the human user's authentication credentials (such as an OAuth token), securing them inside a vault, and passing them downstream to any tools, APIs, or databases the agent interacts with. This not only ensures the agent is bound by the exact same IAM permissions and access control lists (ACLs) as the user, but when used in conjunction with the gateway, ensures that the OAuth tokens are injected at the gateway rather than ever touching the agent

Agent interactions and runtime

Agent interactions and runtime

With the build pipelines secured and the containers deployed, the focus shifts from code integrity to interaction security. Key runtime threats include prompt injection, unauthorized lateral movement between agents, and data exfiltration. We employ a multi-layered approach to mitigate these risks.

Network isolation and egress controls

To physically contain the agents and control their network traffic, we implement several layers of network controls. The following are examples of some of the controls applied to the Warranty Claim system:

- **Macro perimeter**: VPC Service Controls (VPC-Service Controls) wraps the entire warranty ecosystem so that even if an agent's credentials are hijacked, sensitive data cannot be exfiltrated to unauthorized external environments.
- **Internal gatekeeper**: Cloud Next Generation Firewall (NGFW) applies Layer 7 inspection to agent-to-agent (A2A) and MCP communications, ensuring a compromised Case Manager Agent cannot execute SQL injections against the database.
- **Egress filtering**: For the Logistics Liaison Agent calling external shipping carriers, a Secure Web Proxy (SWP) restricts outbound connections exclusively to pre-approved vendor URLs, blocking any attempts to contact malicious command-and-control servers.

Note

For centralized authentication on ingress requests from the user interface, deploy Identity-Aware Proxy (IAP) and Cloud Armor in front of your load balancers.

For a deeper overview of agentic network security and additional controls, refer to Multi-agent private networking patterns in Google Cloud.

Mitigating ‘shadow AI’ with Agent Registry

As enterprises scale from isolated prototypes to multi-agent meshes, they face additional risks from the unauthorized use of AI.

When the Case Manager and Data Vault Agents are deployed to Agent Runtime, they are automatically cataloged within Agent Registry. This creates a centralized, tightly governed enterprise directory. Instead of hardcoding external shipping APIs or internal databases directly into the agent's logic, developers register these as managed endpoints and MCP servers. Agent Registry ensures every agent and tool is visible, version-controlled, and bound by strict project-level IAM boundaries.

Inline context and IAM enforcement: Agent Gateway x Model Armor integration (Private Preview)

Above the baseline network layer, Agent Gateway (Private Preview) serves as the core routing layer for Agent Runtime. It secures **Client-to-Agent (ingress)** requests, agent-to-agent communications, and **agent-to-anywhere (egress)** traffic.

The following diagram illustrates the inline security lifecycle of a request, detailing the step-by-step authorization and inspection process when the Data Vault agent initiates a call to the Case Manager:

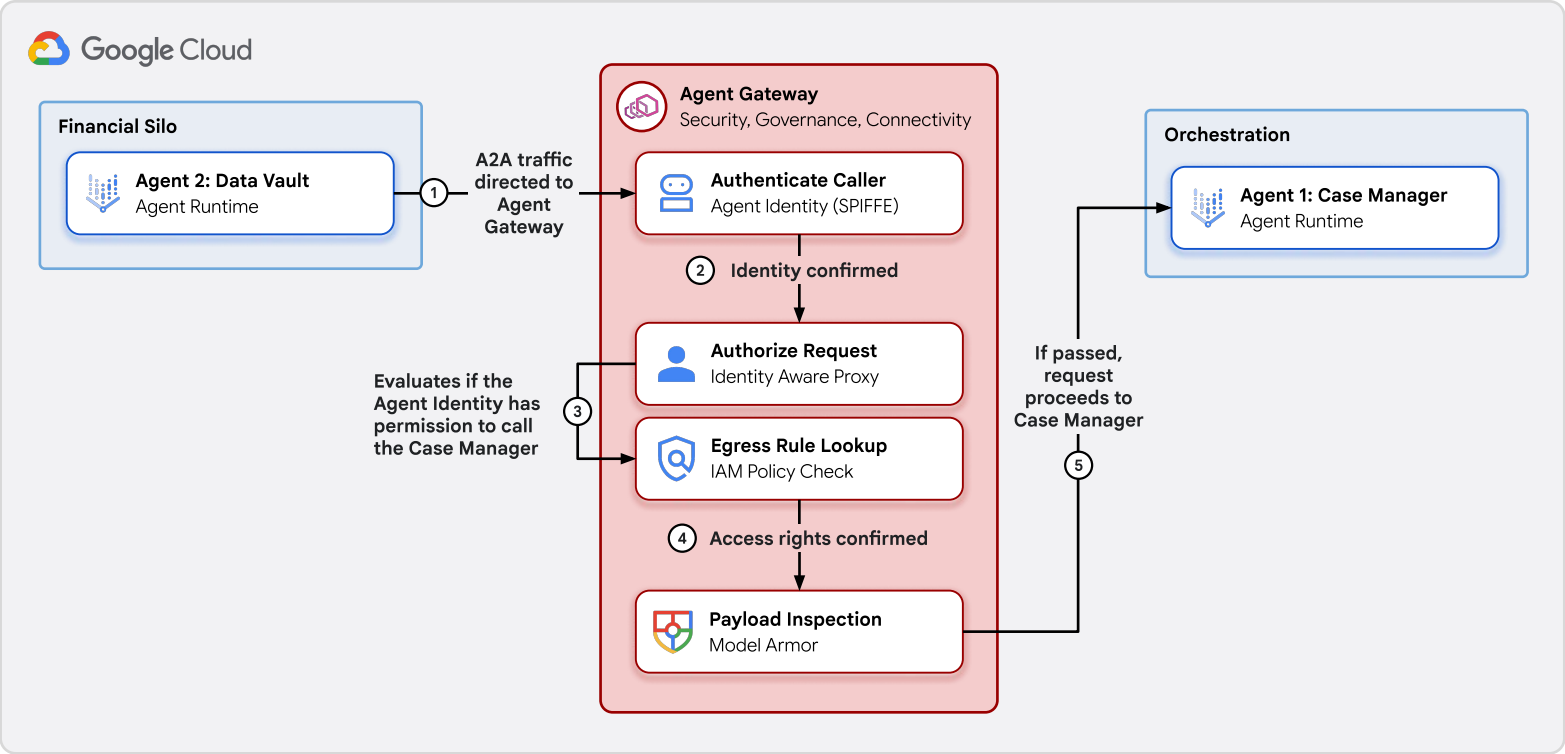

Figure 2. Agent Gateway request lifecycle and policy enforcement

As illustrated in Figure 2, this inline security workflow breaks down into the following sequence of events:

1. **Traffic interception**: The Agent Gateway intercepts the call from the Data Vault agent.
2. **Identity Authentication**: The Agent Gateway first authenticates the calling agent. It verifies the identity of the Data Vault Agent using its presented Agent Identity (e.g., its unique SPIFFE ID).
3. **Access Management & policy enforcement**: The Agent Gateway uses IAP to enforce IAM policies. This IAM policy check (the allowed egress rule) evaluates if the authenticated Agent Identity has the necessary permissions to call the Case Manager Agent.
4. **Model Armor invocation**: With Model Armor enabled on Agent Gateway, agent prompts are subject to deeper analysis. This includes checking for prompt injections, sensitive data leakage (PII, etc.), malicious content (e.g., unsafe URLs), and policy violations based on custom templates.
5. **Inspection & decision**: Based on the final policy evaluation, the request is either allowed or denied. If the IAM policies (egress rules) and Model Armor verdicts permit the interaction, Agent Gateway forwards the request to the Case Manager agent.

Agent Gateway orchestrates a portfolio of robust services to provide content-based and conventional access permissions-based inline inspection. Through Agent Gateway agents are subject to the following inline inspection services:

- **Zero trust routing through IAP**: Access is enforced between the warranty agents through Agent Gateway, which utilizes the Identity-Aware Proxy (IAP) to enforce IAM allow policies dynamically on the network traffic itself, rather than solely relying on IAM allow and deny policies attached directly to an agent's underlying service account. For example, when the Case Manager orchestrator attempts an agent-to-agent (A2A) handoff to the Logistics Liaison, or when the Data Vault agent queries the BigQuery MCP server, that egress traffic must route through the Gateway. IAP intercepts the request and cryptographically verifies the calling agent's identity against the allow policy before permitting the connection, ensuring strict zero-trust communication across the entire warranty ecosystem.
- **Centralized discovery**: Agent Gateway integrates with Agent Registry, utilizing it as the central source of truth. Any tool or agent attempting to communicate that is not explicitly registered and authorized by the IAP policy is automatically blocked.
- **Real-time content sanitization**: Crucially, Agent Gateway integrates directly with Model Armor. By enabling Sensitive Data Protection (SDP), administrators gain granular control over payload sanitization.

Model Armor doesn't just scrub final responses sent back to the human user; it actively inspects and sanitizes all egress payloads sent to downstream agents or external MCPs. In the warranty system, it neutralizes lateral prompt injections and ensures the Data Vault Agent cannot inadvertently leak personal information to the Logistics Liaison Agent.

Protecting agent state and memory

Deploying to Agent Runtime means developers do not have to write custom, potentially insecure database wrappers to maintain conversation history. Agents running on Cloud Run like the Logistics Liaison Agent can also use these services through API calls.

- **Short-Term Context**: When the Orchestrator and Data Vault Agents are deployed to Agent Runtime the Agent Platform Sessions service handles immediate conversational context automatically, securely mapped to the user's session ID.
- **Long-Term Persistence**: Long-term knowledge is securely routed to the Agent Platform Memory Bank. To prevent data poisoning and unauthorized access, the Memory Bank enforces strict isolation using IAM conditions and `memoryScope` attributes. Furthermore, it can manage data life cycles through customization, Time-to-Live (TTL) policies and mandates that all unstructured inputs pass through Model Armor before being written to disk.

The double-guardrail approach

Standard IAM prevents unauthorized API calls, but it cannot prevent an LLM from hallucinating or acting on non-compliant topics or actions. Safely scaling an enterprise agent platform requires shifting from abstract governance concepts to concrete runtime controls. This is achieved using two distinct policy layers enforced during agent interactions:

- **Controlling Access (IAM Boundaries)**
  - **Enforcement**: Applied using IAM allow policies on agent identities at Agent Gateway
  - **Function**: When the Case Manager attempts an agent-to-agent (A2A) handoff to the Logistics Liaison, Agent Gateway intercepts the request. It inspects the Case Manager's unique machine identity (SPIFFE ID) and verifies it against Identity Aware Proxy (IAP). In this way Agent Gateway uses IAM to deterministically control which downstream agents, foundation models, or external Model Context Protocol (MCP) servers the Case Manager is technically permitted to invoke; blocking unauthorized lateral movement before the payload is ever processed.

- **Controlling Intent (Semantic Boundaries)**
  - **Enforcement**: Applied dynamically through Semantic Governance Policies (SGP) (Preview) and custom classifiers.
  - **Function**: Agents non-deterministically process meaning and generate actions. Semantic content aware guardrails must extend to the business logic layer. Even if the technical guardrails authorize the network connection between the Case Manager Agent and the Logistics Liaison Agent, the semantic guardrail acts as an intent-aware firewall inspecting the actual payload in real-time. While operational controls prevent the Logistics Liaison from querying an HR database, semantic controls ensure the transferred prompt cannot trick the Liaison into offering an unauthorized 100% discount code.

Observability and threat detection

Observability and threat detection

After the Warranty Claim System agents are built, deployed, and secured behind network perimeters, the final requirement is continuous oversight. Because generative AI is non-deterministic, developers must monitor not just if the code executed, but why the agent made its specific reasoning decisions.

Agent observability: decoding the chain-of-thought

To debug complex multi-agent workflows—like understanding why the Case Manager Agent categorized an issue a certain way—visibility into the agent's reasoning is critical:

- Cloud Trace: Send the ADK's default telemetry to Cloud Trace to visualize the agent's "chain-of-thought." This provides a waterfall view linking the agent's internal reasoning directly to its tool executions.
- Cloud Logging & Agent Identity Logs: These logs help to establish behavioral baselines. The Agent Identity Logs allow security teams to cryptographically audit when an agent acquired credentials to act autonomously versus when it utilized user-delegated tokens.

Threat detection & virtual red-teaming

Traditional application security scans code, but it cannot understand a model's intent. To govern dynamic AI, we leverage Google Cloud's native threat detection suite:

- Agent Platform anomaly detection: Ingests OpenTelemetry traces natively emitted by the ADK to provide asynchronous, reasoning-based oversight without introducing latency to live customer chats.
- Security Command Center (SCC) - AI Protection (AIP): Automatically discovers AI data flows within your project. Most importantly, security teams use AIP to execute continuous Virtual Red-Teaming - running automated, AI-driven adversarial simulations against the deployed agents to ensure the IAM boundaries hold firm under active exploit attempts.
- SCC - Event Threat Detection (ETD): Analyzes Cloud Logging in real-time to detect runtime threats specifically targeting your Agent Runtime workloads or Agent Identities. You can also view agent specific findings under the Security tab in Agent Platform.

Next steps: the crawl, walk, run approach

Next steps: the crawl, walk, run approach

Securing an enterprise AI application is a journey. Rather than trying to implement every Google Cloud security product on day one, we recommend a phased approach, like the following:

Phase 1: Crawl (hosting & baseline access)

In the initial phase, your focus is on getting your agent out of your IDE and into a hosted development environment.

- **Initialize Agent Runtime**: Deploy your code using the Python SDK ReasoningEngine.create().
- **Secure Baseline Deployment**: Do not use broad standard service accounts, instead use the deployment flags to immediately provision a unique, SPIFFE-backed Agent Identity (IdentityType.AGENT_IDENTITY). This establishes a cryptographic, least-privilege boundary from day one.

Phase 2: Walk (tool integration & content safety)

After your agent is functional, lock down who it is and what it can say.

- **Secure Google Cloud MCP access**: Grant your Agent Identity strictly scoped IAM permissions to the Google Cloud MCP Server for BigQuery. This eliminates custom API wrappers and allows IAM to cleanly enforce least-privilege database access.
- **Enable Model Armor**: Route the agent's inputs and outputs through Model Armor to automatically detect and redact PII or block jailbreak attempts.

Phase 3: Run (enterprise-grade lockdown)

For production systems that handle sensitive financial or healthcare data, engage your security team to implement the infrastructure perimeter.

- **Verifiable supply chain**: Automate deployments using Cloud Build to generate SLSA Level 3 provenance, and use Binary Authorization to ensure that only cryptographically signed, vulnerability-free containers are allowed to run.
- **Inline policy enforcement (Agent Gateway)**: Route all agent traffic through Agent Gateway. Acting as a centralized Identity-Aware Proxy (IAP), Agent Gateway verifies Agent Identities before permitting any agent-to-agent (A2A) handoffs or outbound MCP queries. It also natively integrates with Model Armor and sanitizes all the interactions.
- **Network isolation**: Wrap your project in a VPC Service Controls (VPC-Service Controls) perimeter to prevent data exfiltration.
- **Defeat shadow AI**: Catalog all agents and external APIs in Agent Registry to provide your CISO with complete visibility into the multi-agent mesh.

Securing a multi-agent system does not have to mean sacrificing development agility. By integrating Gemini Enterprise Agent Platform security capabilities such as Agent Identity, Agent Registry with Policies, Agent Gateway and Model Armor, directly into the architecture, security transforms from a deployment blocker into a foundational enabler.

The "crawl, walk, run" approach allows engineering teams to begin experimenting with multi-agent workflows today, safe in the knowledge that there is a clear, scalable path to enterprise-grade compliance tomorrow.

As companies evolve from isolated, read-only chatbots to autonomous agents, the risks inherently increase. However, by adopting a defense-in-depth strategy grounded in Google's Secure AI Framework (SAIF), combining infrastructure boundaries with intent-aware semantic guardrails, organizations can systematically neutralize those risks. This secure-by-design foundation ultimately empowers your enterprise to innovate fearlessly, automating complex business processes and delivering frictionless, next-generation experiences to your users.