Production Observability for Gemini Enterprise Agents

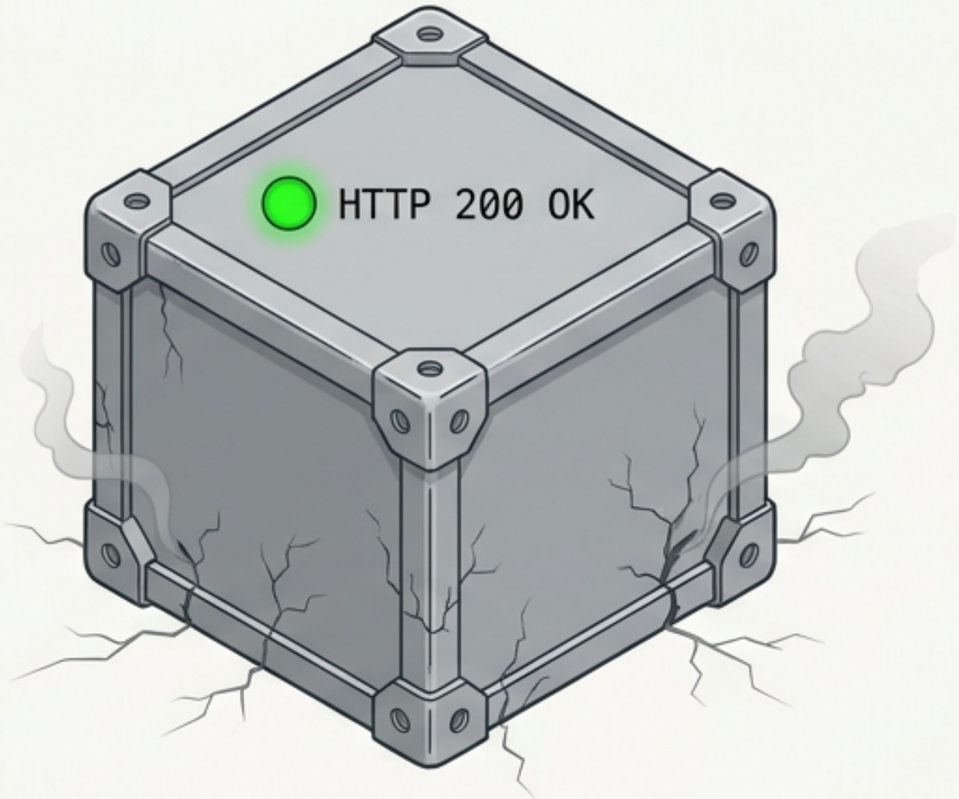
The Diagnostic Playbook for the Four-Layer Agent Stack

Target: SREs | DevOps | Enterprise AI Engineers

A Secured Pipeline Is Not Always a Healthy System

Chapter 5 made the invoice pipeline defensible. But standard endpoint metrics cannot tell you **if the system is actually working**. An agent workflow is a chain—and an HTTP metric only tells you if the outer request succeeded.

The Blind Spot



The Four Unanswerable Questions

Metrics and Health

Which specific **agent**, **model call**, or **tool** made this request slow?

Traces and Pathways

Did this failure originate in **infrastructure**, a **tool**, **policy enforcement**, or **agent reasoning**?

Logs and Facts

Are **latency** and **error rates** trending in the wrong direction?

Evaluations and Quality

Is **answer quality** drifting even when the **runtime** looks perfectly healthy?

The Agent Observability Stack

Layer	Purpose	Google Cloud Tool	Practical Rule
Layer 1: Metrics	Capture operational trends	`Cloud Monitoring`	Detect trend changes & service health.
Layer 2: Traces	Show execution pathways	`Cloud Trace`	Debug one specific request pipeline.
Layer 3: Logs	Capture durable event detail	`Cloud Logging`	Preserve exact business facts.
Layer 4: Evaluations	Capture cognitive agent quality	`Agent Platform Evals`	Catch answer-quality regressions.

Enabling Telemetry Without Inventing an API

Code Anatomy

```
GOOGLE_CLOUD_AGENT_ENGINE_ENABLE_TELEMETRY=true

OTEL_SEMCONV_STABILITY_OPT_IN=gen_ai_latest_experimental

OTEL_INSTRUMENTATION_GENAI_CAPTURE_MESSAGE_CONTENT=EVENT_ONLY
```

Enables agent traces and logs on Agent Runtime.

Standardizes generative AI attributes using the latest OpenTelemetry semantic conventions.

Warning: Design Decision Required. Enables logging of input prompts and output responses.

Production Policy Callout

In staging, high capture allows inspection of bad extractions. In production, this captures supplier names, addresses, and bank details. Content capture must be sampled, access-controlled, and retention-limited.

Layer 1: The Minimum Production Dashboard

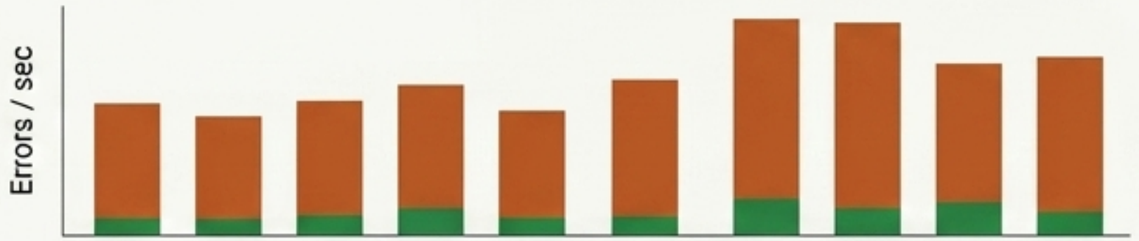
1. Request Volume

Monitors `aiplatform.googleapis.com/reasoning_engine/request_count`.
Catches traffic drops and deploy routing mistakes.

2. Latency Percentiles

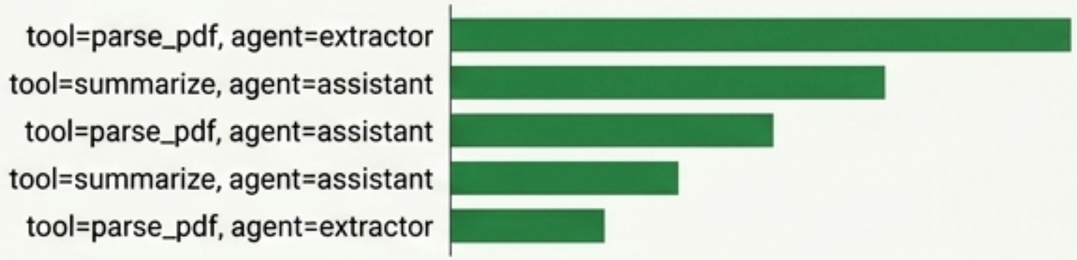
Track `p50`, `p95`, and `p99` by `agent deployment`.
Alert: Never alert on average latency. Agent workloads have long tails—one slow document retrieval can ruin 5% of requests while the average stays fine.

3. Error Rate Ratio



Distinguish 5xx (infrastructure incidents) from `4xx/policy-denials` (product integration incidents).

4. Tool-Call Volume



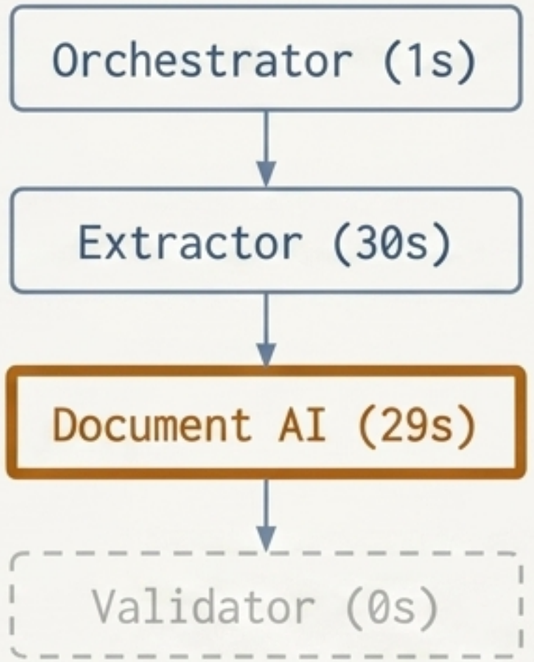
Create log-based metrics for specific functions (e.g., `tool=parse_pdf`, `agent=extractor`).

The Anti-Pattern: Guessing from the P99 Chart

The Misleading Symptom

Top-level metric shows 31 seconds before returning “unable to validate”. A log search just shows a timeout.

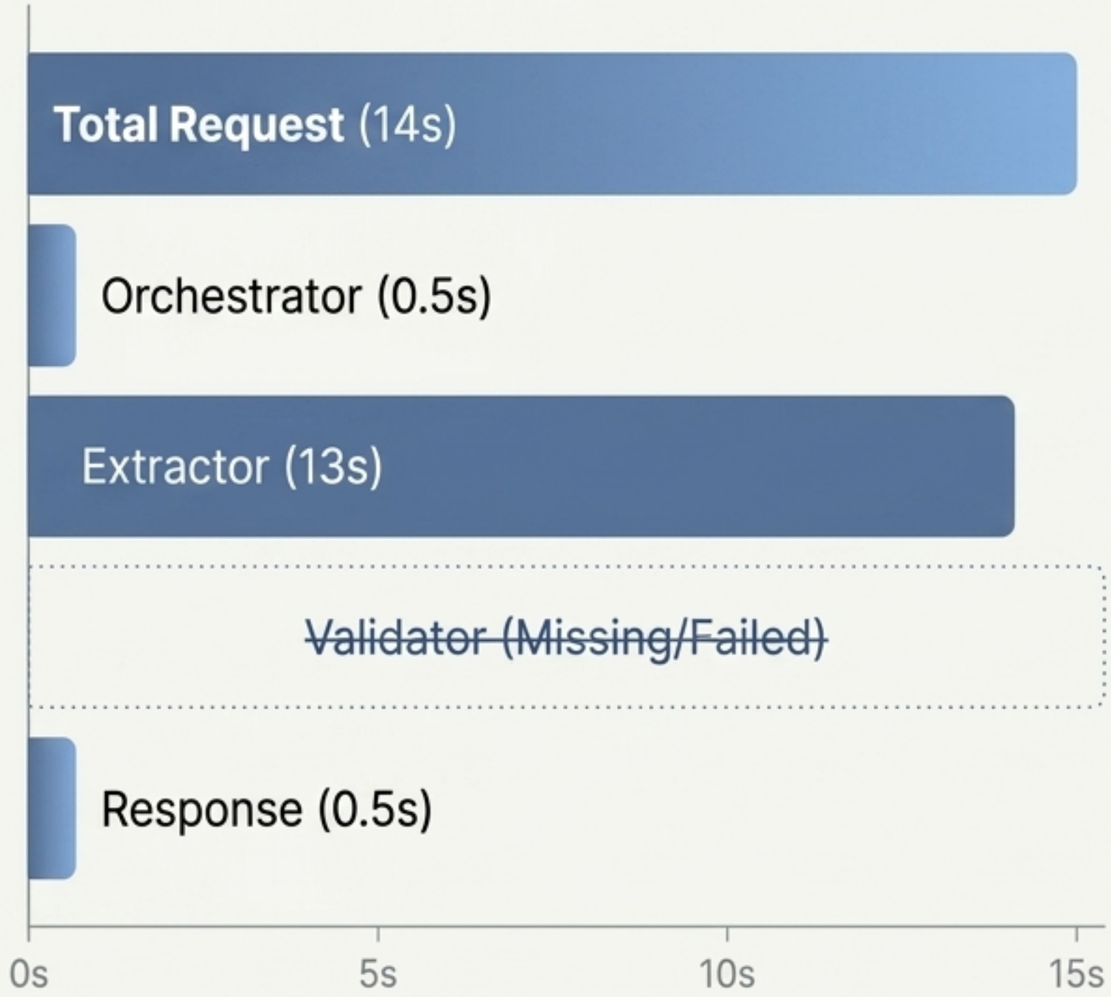
The Truthful Trace (DAG)



Key Insight: The trace proves the bottleneck is a tool, not model reasoning. Crucially, it proves the Validator never ran, meaning the “unable to validate” error shown to the user is a lie.

Layer 2: How to Read an Agent Trace DAG

Stylized Trace Timeline (Waterfall Chart)



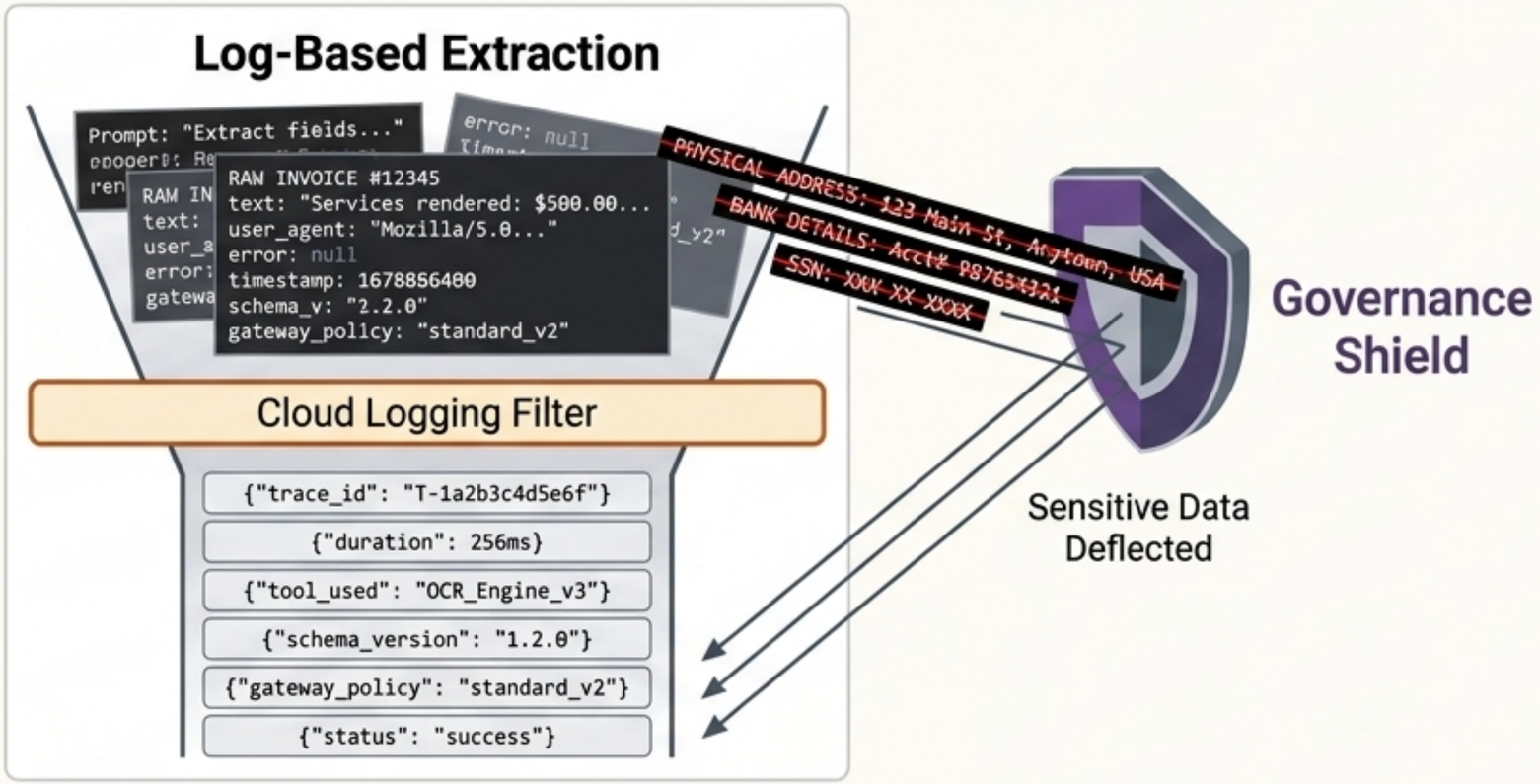
How to Analyze the Trace DAG

1. **Confirm**: Check top-level status and total duration.
   Look at the parent span. Is the overall status OK or Error? How long did the entire request take?)
2. **Isolate**: Identify the longest child span and its status (Tool vs. Model).
   Pinpoint the bottleneck. Is it a slow tool call, or did the model spend a long time reasoning?)
3. **Verify**: Check whether expected downstream spans are missing (did the Validator run?).
   Look for gaps in the timeline. If a step like 'Validator' or 'Response' is absent, it indicates a failure.
4. **Compare**: Separate model reasoning spans from external tool execution spans.
   Differentiate between internal processing (model) and external calls (tools). Tool calls are often the culprit for latency.
5. **Connect**: Copy the trace ID into incident notes.
   Use the trace ID for cross-referencing with logs, metrics, and for future evaluation.

A trace that caused a customer incident must become an offline evaluation case.

Layer 3: Logs for **Durable Business Facts**

Logs are not a substitute for traces. They are the durable event stream answering questions a trace DAG cannot: Which schema version ran? Which gateway policy applied?



The Golden Rule: Log identifiers that let authorized responders join to the system of record—**never** log the sensitive record itself.

Structured **Event Logging** in **Practice**

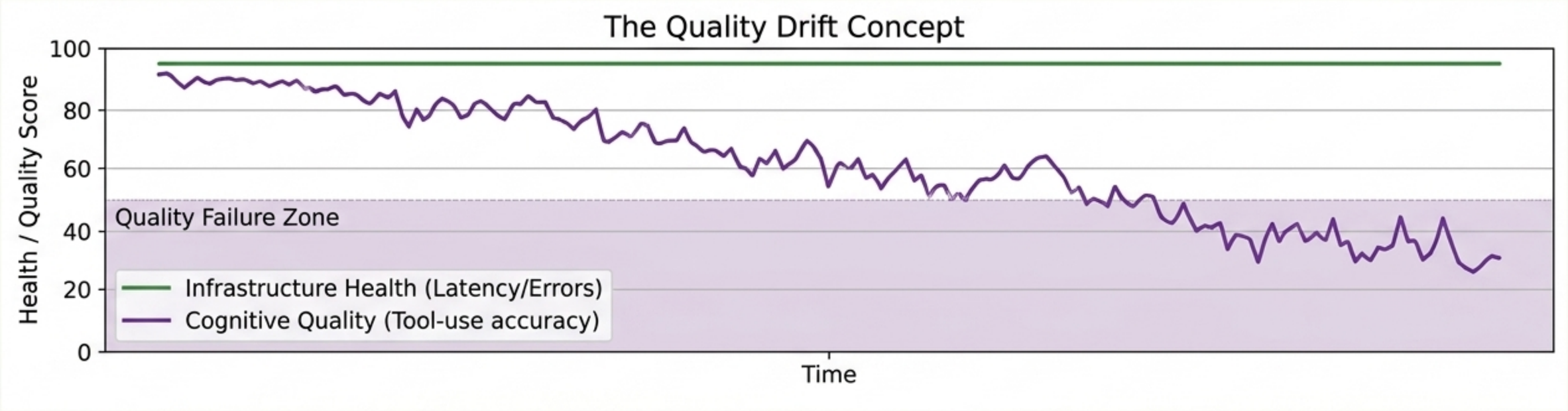
✓ **What to Log** (Operational Facts)	⚠ **What to Drop** (Governance Risks)

`trace_id: "a1b2c3d4..."`
`agent: "extractor"`
`schema_version: "v2.1"`
`supplier_class: "strategic"`
`gateway_decision: "ALLOW"`
	~~Raw invoice numbers~~
	~~Supplier bank account details~~
	~~Raw physical addresses~~
	~~Uploaded filenames~~

Unbounded values (like raw filenames) will destroy log-based metric cardinality, slowing dashboards and ballooning your Cloud bill.

The Synthesis: The Danger of the Fast Wrong Answer

Your invoice pipeline p95 latency is stable. 5xx errors are zero. But customer support reports the agent is approving invoices without checking purchase orders.



The Insight:
An agent can have perfect metrics but be fundamentally broken. Operational Health + Cognitive Quality = True Production Readiness.

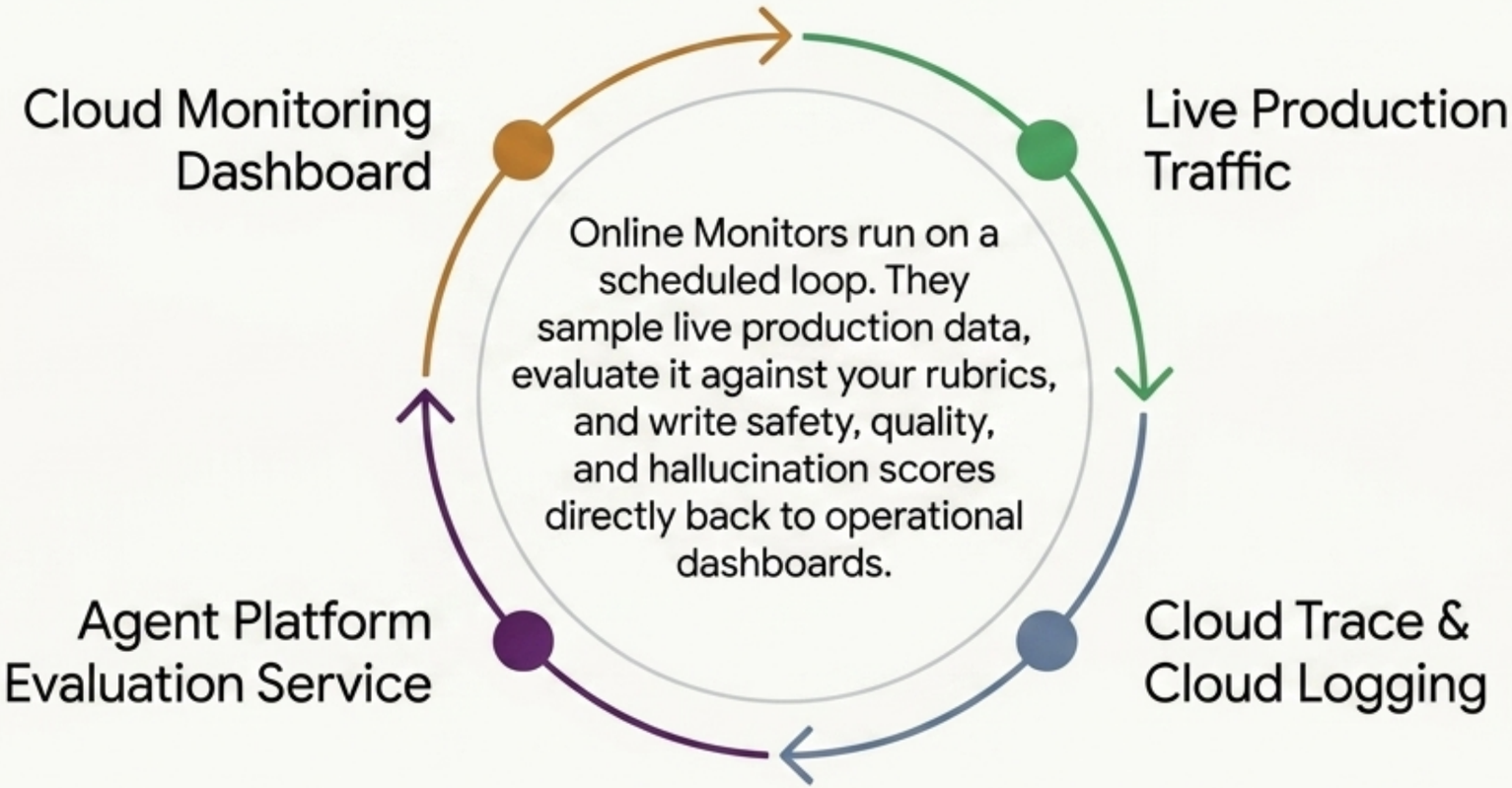
The Transition:
Runtime metrics catch infrastructure failures. Only Evaluation catches quality failures.

Layer 4: The Three Modes of Agent Evaluation

Rapid Evaluation (Local Dev)	Test Case (Offline) (CI/CD Regression)	Online Monitors (Production Drift)
Trigger: Local iteration.	**Trigger:** Pre-deployment / Code merge.	**Trigger:** Continuous scheduling.
Use Case: Checking if an orchestrator prompt orchestrator prompt tweak improved tool-routing on 10 quick examples.	**Use Case:** Running a baseline Running a baseline dataset of 30 edge-cases (rotated scans, missing POs, ambiguity) against predefined metrics.	**Use Case:** Sampling live Cloud Trace and Logging data to track hallucination rates rates and tool-use quality over time.

Rethinking Drift: Online Monitors

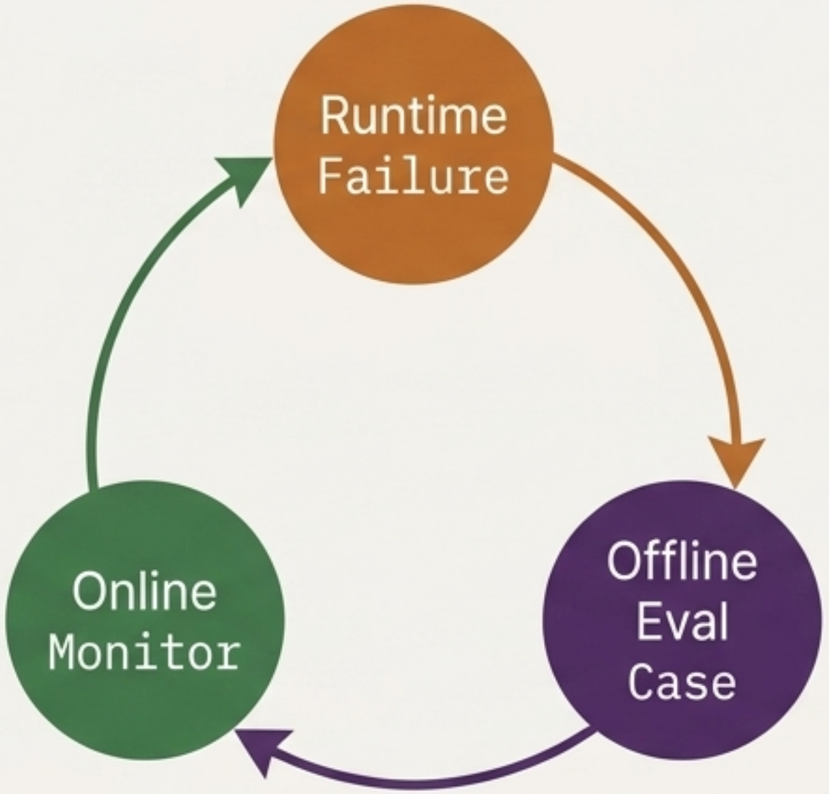
Traditional ML drift asks: “Did the input distribution move?”
Agent drift asks: “Did the agent still complete the task and use the right tools?”



Turn Incidents Into Guardrails

The Rule: Every production incident must leave behind one durable test case.

The Incident-to-Eval Flywheel



Translation Table

Incident Evidence	→	Eval Case Field
`Trace ID`	→	`Source trace`
User request category		`Scenario`
Expected tool path		`Rubric criterion`
Actual tool path		`Failure evidence`
Correct final behavior		`Expected answer`

Diagnosing Failure Clusters

Agent Platform groups evaluation failures into semantic clusters. Treat each cluster as a specific product bug, not a general “model mood”.

Failure Cluster Diagnostic Table	
Failure Cluster (Symptom)	**Likely Fix (Prescription)**
Agent skipped required PO lookup	Orchestrator instruction & tool policy
Tool returned malformed JSON	Tool wrapper/schema validation
Agent invented supplier ID	Grounding & final-answer rubric
Model Armor blocked content	Gateway policy tuning
Extractor timed out on long scans	Tool timeout limits & document preprocessing

The Optimization Order of Operations

Do not optimize prompts before analyzing the failure cluster.

The Quality Flywheel

The Reality Check

If the failure is a bad tool timeout, prompt optimization is theater. You need an infrastructure fix.

The Reality Check

If the failure is poor instruction-following when handling ambiguous supplier names, then prompt optimization is exactly the right tool.

The Observable Agent: Operational Summary

Layer 1: Metrics (Health)

Use aiplatform request counts and percentiles to spot the pain. Never rely on averages.

Layer 2: Traces (Pathways)

Read the DAG to find the exact bottleneck and confirm if tools actually executed.

Layer 3: Logs (Facts)

Capture structured, durable event details without leaking raw PII into metrics.

Layer 4: Evaluations (Quality)

Use Online Monitors and Offline cases to catch the fast wrong answer.

Closing takeaway: You can now scale, provision throughput, and manage costs (Chapter 7) because you can finally see exactly what the agent is doing.