

Observability for Production AI Agents: Architecture, Deployment Profiles and Cost Behaviour

Saathvikeya Simha Banala

Abstract

Autonomous software agents built on large language models failed in ways that conventional application monitoring did not detect. An agent returned a successful response code while producing an incorrect answer, or consumed a large token budget in a retry loop that no latency or error dashboard registered. This study examined how production teams instrumented such systems during the first half of 2026 and what the resulting telemetry cost. Vendor documentation, platform pricing schedules and published practitioner accounts were reviewed, and a layered reference architecture was synthesised from the patterns that recurred across them. Three deployment profiles were then derived and costed against three traffic volumes using a stated set of assumptions. The analysis found that the billing unit a platform adopted influenced the final invoice more than its headline price, because a single agent request generated twenty to fifty spans rather than one. Sampling and redaction performed at an intermediate collector tier, rather than at the storage backend, emerged as the decisive architectural control over both cost and privacy exposure.

Copyright © 2026 International Journals of Multidisciplinary Research Academy. All rights reserved.

Keywords:

Agent observability;
OpenTelemetry;
Distributed tracing;
Site reliability engineering;
Telemetry cost management.

Author correspondence:

Saathvikeya Simha Banala,
Student, Alpharetta, Georgia, USA
Email: saathvikeya42@gmail.com

1. Introduction

Monitoring practice rests on an assumption that has held for most of the history of web services: when something goes wrong, the system says so. A request fails, a queue backs up, a latency percentile climbs, and the dashboard turns a colour that brings somebody to a keyboard. Software agents built on large language models break that assumption quietly. An agent asked a question may plan an approach, call a model several times, reach for two or three tools, retrieve documents from a vector store and hand off to another agent, and it will usually return a well-formed answer at the end of all that. Whether the answer was correct, whether the plan made sense, and what the whole sequence cost are questions the response code does not address.

Three failure modes recur in practitioner reports and motivate this study. The first is silent incorrectness: the agent returns a confident and entirely wrong answer with no exception anywhere in the trace. The second is unbounded iteration, in which a tool call fails in a way the agent treats as retryable and the loop consumes tokens until a quota or a timeout stops it. The third is drift. Someone edits a prompt, a model version is swapped underneath the application, or the retrieved corpus shifts, and answer quality declines over weeks with no deployment to correlate against.

The instrumentation problem itself is close to settled. OpenTelemetry has been adopted widely enough that most agent frameworks and observability vendors now speak it, and its generative-AI semantic conventions define agent-aware span names and token-usage metrics. Cloud providers document how their own agent runtimes emit the same signals. What the existing material does not do is connect the architecture to its consequences. Vendor guides describe what to instrument without pricing it; pricing pages describe rates without describing the telemetry volume an agent actually produces. Teams consequently discover the cost of their design after the first full month of traffic rather than before.

This paper addresses that gap. It contributes, first, a five-layer reference architecture synthesised from the configurations that recurred across the reviewed sources; second, a comparison of nine platforms organised

by billing unit rather than by feature list; third, a cost model applied to three traffic volumes with its assumptions stated openly; fourth, three deployment profiles drawn as complete stacks, showing that they differ only below the instrumentation layer; and fifth, an account of how the same telemetry supports agent-operated incident response. Section 2 describes how the study was conducted. Section 3 presents the architecture, the platform survey, the cost analysis and the deployment profiles. Section 4 draws the argument together.

2. Research Method

This is a synthesis study rather than an experimental one. No new system was built and no controlled benchmark was run. The work proceeded in three stages: assembling a source corpus, deriving an architecture from it, and building a cost model that the architecture could be evaluated against.

2.1. Source corpus

Sources were collected between January and June 2026 and fall into three groups. The first is specification and standards material, principally the OpenTelemetry generative-AI semantic conventions and the accompanying implementation notes. The second is primary vendor documentation: cloud platform observability guides, agent-runtime instrumentation references and published pricing schedules. The third is practitioner writing, meaning engineering accounts from teams that had run agents in production and independent cost analyses. Marketing material that made capability claims without describing a method was recorded but excluded from the analysis. All prices reported below are those listed by vendors in mid-2026 and should be treated as a starting point rather than a quotation, since they change frequently and rarely downward.

2.2. Architecture synthesis

Each source was read for two things: the components it named and the direction of data flow between them. Components were then grouped by the function they performed rather than by the vendor that supplied them, on the reasoning that a self-hosted trace store and a commercial one occupy the same position in a design even though they differ in every other respect. A layer was retained in the reference architecture only where it appeared, in some form, across sources from all three groups. The intermediate collector tier is the clearest example: it is described in the standards material, assumed by the cloud documentation and identified in the practitioner accounts as the component most often omitted on the first attempt.

2.3. Cost model and assumptions

The monthly observability cost of a workload was modelled as the sum of four metered quantities: spans retained after sampling, log bytes ingested, metric bytes ingested, and bytes scanned by analytical queries. Four assumptions were held constant so that the three workloads in Section 3.6 remain comparable. Each request produces approximately twenty-five spans before sampling, which sits in the middle of the twenty to fifty range reported across sources. Tail sampling at the collector retains all errors and slow traces plus ten percent of successful ones. High-volume prompt and response logs are routed to an analytical store rather than held in the default logging bucket. Finally, and importantly, the figures cover observability alone and exclude the cost of the model inference itself, which is normally the larger of the two.

Three limitations follow from this design and should be kept in view when reading the results. Prices were volatile throughout the collection window, so the absolute figures will age faster than the relative comparisons between billing models. Accuracy claims made by vendors of autonomous incident-response tools were recorded as claims and not verified, because no blind test data was available. And the cost model assumes a uniform span count per request, whereas real agent traffic is heavy-tailed, with a small number of long planning runs producing a disproportionate share of the telemetry.

3. Results and Analysis

3.1. A layered reference architecture

The configuration that recurred across the corpus is shown in Figure 1. It reads from top to bottom in five layers. The agent runtime produces telemetry; a single open standard gives that telemetry a consistent shape; a collector tier in the middle is where volume and privacy are controlled; two storage backends serve two different investigative questions, with a third and cheaper tier beneath them for bulky payloads; and a reliability layer on top converts the whole thing into something that protects users. The layers are described in turn below.

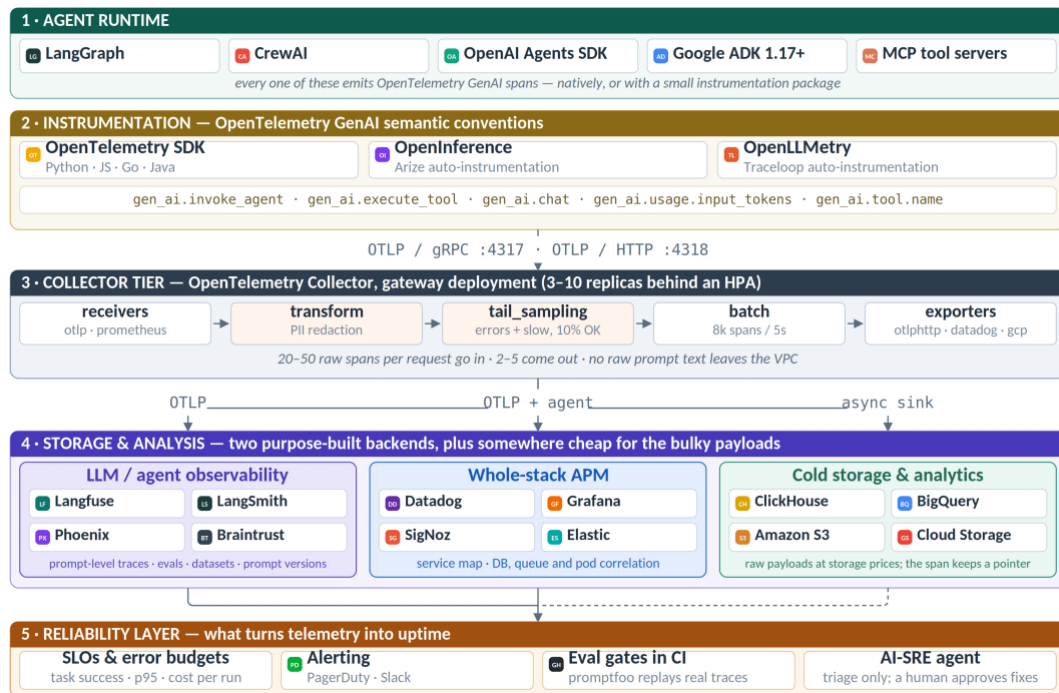


Figure 1. Five-layer reference architecture for agent observability, showing the frameworks, collector pipeline and storage backends that recur across sources

3.2. Instrumentation and the semantic conventions

The conventional triad of metrics, logs and traces survives the move to agents; each signal simply grows an agent-shaped extension. A trace should span an entire request, with child spans for the planning step, each model call and each tool invocation, so that an unusual run opens as one coherent timeline instead of a set of log lines to be correlated by hand. Metrics extend beyond latency and error rate to token usage, cost per run, tool-call success rate and quality scores broken out by model and by agent. Logs carry the prompts and responses themselves, which is where the privacy exposure concentrates, alongside the output of evaluation checks for faithfulness, safety and tool-use accuracy.

The OpenTelemetry generative-AI conventions supply agent-aware span names for agent creation, agent invocation and tool execution, with individual model calls hanging beneath them, together with the duration and token metrics needed for costing. LangChain and LangGraph, CrewAI and the OpenAI Agents software development kit emit these already, either by default or through a small instrumentation package, and agents built with Google's Agent Development Kit from version 1.17 onward export them to cloud observability with very little configuration. One caveat deserves stating plainly: the specification remained officially experimental as of mid-2026. The agent span names have been stable in practice, but occasional attribute changes should be expected.

3.3. The collector tier

The collector sits between the agents and wherever the data finally lands, and it earns its place twice over. The first reason is cost. A request that produces twenty to fifty spans makes indiscriminate retention expensive very quickly, so sampling is applied with intent: every error and every slow trace is kept, while perhaps five to ten percent of the fast, uneventful successes are retained for baseline comparison. The second reason is privacy. Prompts and responses are free text, and free text has a habit of containing things it should not. Personally identifiable information is stripped here, before the data leaves the network boundary. Where regulation demands it, the bulky payloads are pushed into object storage and the span retains only a reference to them.

The ordering of the processors in that pipeline is not cosmetic. Redaction is placed ahead of sampling and batching so that no unredacted payload is ever held in a buffer, and the exporters sit last so that a backend can be added or replaced without touching a line of application code. This is the property that makes the architecture reversible, and it is the strongest practical argument for standardising on an open instrumentation format.

3.4. Platform landscape and billing units

Nine platforms were compared, and Table 1 records them alongside how each one charges. The final column matters more than the feature comparison that usually dominates these decisions. Agent traffic

punishes the wrong billing unit severely. Consider a fifty-step agent run. Priced per span, it costs roughly what a single model call costs. Priced per trace, it can cost fifty times as much. Unit-based plans that tally traces plus observations plus evaluation scores fall between the two, typically consuming eight to fifteen units for a run that a simple completion would settle in one. Flat per-span or per-gigabyte pricing proved the most predictable at volume.

Table 1. Agent-observability platforms and their billing models, approximate, mid-2026

Platform	Billing model	Free tier	Paid entry
OpenTelemetry	Open standard, instrumentation only	Free	Free; a backend is still required
Langfuse	Usage units: traces, spans and scores	50K observations per month, 30-day retention	About \$29 per month; free if self-hosted
LangSmith	Seat licence plus trace overage	Limited	About \$39 per seat per month
Arize Phoenix / AX	Open source; AX charged per gigabyte	Phoenix free when self-hosted	AX about \$3 per gigabyte stored
Braintrust	Spans plus evaluations	1M spans and 10K evaluations per month	Custom
Datadog Agent Observability	Per monitored request	40K spans per month	About \$8 per 10K requests; Pro from about \$160 per month
Logfire (Pydantic)	Flat per span	Generous	About \$2 per million spans
OpenObserve	Flat per gigabyte	Yes	About \$0.50 per gigabyte
SigNoz / Grafana LGTM	Self-hosted; infrastructure only	Free, community edition	Infrastructure cost

Two backends run side by side in the reference architecture, and the reviewed practitioner accounts were consistent about why. A specialist tool such as Langfuse, LangSmith, Arize Phoenix or Braintrust provides prompt-level traces, evaluation runs and dashboards built around agent concepts. A general-purpose application performance monitor such as Datadog, a self-hosted Grafana stack or an OpenTelemetry-native product such as SigNoz ties the agent's behaviour back to the databases, queues and services beneath it. Consolidating both jobs into one product is possible, and teams with longer operational experience generally declined to do so, on the grounds that the seams become visible at the least convenient moment.

3.5. A single-vendor realisation

Where agents already run on a single cloud, the same architecture maps onto native services without assembling five vendors. Figure 2 shows the Google Cloud realisation, with the unit prices attached to the components that meter them. Agents built with the Agent Development Kit and those running on the Gemini Enterprise Agent Platform emit telemetry with little or no configuration; anything custom, including a Model Context Protocol server, is instrumented by hand. Cloud Trace, Cloud Logging and Cloud Monitoring hold the three signals, BigQuery provides inexpensive retention and query capability, and Gemini assists investigation both proactively through Cloud Assist and on demand through natural-language queries over the history.

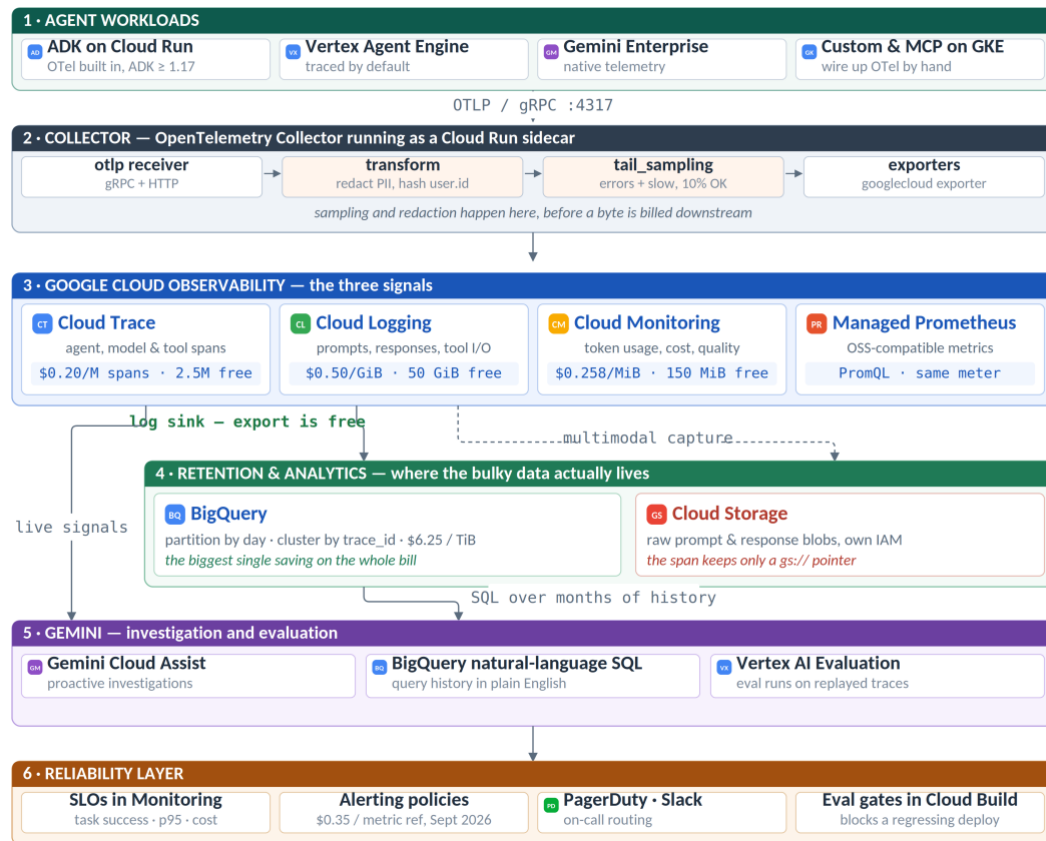


Figure 2. The same architecture built entirely from Google Cloud services, with the unit prices attached to each hop

Two of the arrows in that diagram are really about money. The log sink from Cloud Logging into BigQuery is free to operate, and it is what allows high-volume prompt and response data to be kept at storage prices rather than at logging-retention prices; this was the single largest saving identified in the analysis, and the practitioner sources suggest it is commonly discovered late. The dashed arrow on the right is the one that catches teams out. If full multimodal capture is enabled, the prompts and responses themselves are written to a Cloud Storage bucket and only a pointer remains on the span, which means a fourth location holds sensitive data under its own retention and access rules. Table 2 records the individual unit prices.

Table 2. Google Cloud observability and BigQuery unit pricing, approximate, mid-2026

Component	Approximate rate (2026)	Free allowance
Cloud Logging	~\$0.50 per GiB ingested	50 GiB per project per month, 30-day retention
Cloud Monitoring	~\$0.258 per MiB of metrics	150 MiB per account per month
Cloud Trace	~\$0.20 per million spans	2.5 million spans per month
Alerting (from September 2026)	~\$0.35 per metric reference	None
Log sink to BigQuery	Export is free	BigQuery storage is charged separately
BigQuery (on-demand)	~\$6.25 per TiB scanned	10 GB storage and 1 TiB of queries per month
New-account credit	\$300 over 90 days	None

3.6. Cost behaviour across three workloads

Abstract rates are hard to reason about, so the model described in Section 2.3 was applied to three workloads of increasing size. Table 3 reports the result. The figures cover observability only; inference spend is excluded and is usually the larger number.

Table 3. Illustrative monthly observability cost by workload, inference spend excluded

Line item	Small (10K req/day)	Mid (100K req/day)	Large (1M req/day)
Spans per month, after sampling	~0.75M	~7.5M	~75M
Cloud Trace	\$0 (within free tier)	~\$1	~\$15
Cloud Logging	\$0 (within free tier)	\$0-10	~\$40-95

Line item	Small (10K req/day)	Mid (100K req/day)	Large (1M req/day)
Cloud Monitoring	\$0-20	\$20-150	\$150-800+
BigQuery (storage and scans)	\$0-15	\$20-120	\$100-600
Alerting (from September 2026)	\$0-5	\$5-30	\$30-100
Estimated total per month	~\$10-60	~\$100-450	~\$500-2,500

The shape of the table is more informative than any single cell. Cost is close to negligible at ten thousand requests a day and remains modest at a hundred thousand, but the metric and analytical lines grow fastest as volume rises, which is the opposite of what most teams anticipate when they budget for trace storage. Agents emit something like ten to fifty times the telemetry of an ordinary service, and this is the mechanism behind the forty to two hundred percent increase in observability spend that several practitioner accounts reported after agents entered production.

Five controls reduced that bill materially in the reviewed deployments. Sampling at the collector, keeping errors and slow traces while discarding most routine successes, has the largest effect and is the cheapest to implement. Routing noisy logs into an analytical store trades a per-gigabyte logging charge for far cheaper storage, and the export itself costs nothing. Watching metric cardinality matters because high-cardinality labels such as a user or trace identifier consume the per-megabyte monitoring meter very quickly. Partitioning and clustering analytical tables determines whether a given query costs cents or dollars, since the difference is whether the engine must scan the whole table. Finally, a hard token quota per tenant is inexpensive insurance against an agent that enters a loop and runs up inference and telemetry charges simultaneously.

3.7. Three deployment profiles

The recommendation that follows from the analysis is not a single stack, because the right answer depends on where a team already is. Three profiles were drawn out in full. The point worth noticing across Figures 3, 4 and 5 is that the top of each diagram is identical: the same agent frameworks, the same OpenTelemetry instrumentation, the same collector. Only what sits below changes, and with it the cost model. That sameness is the practical return on standardising the instrumentation layer, because it means a team can move between these profiles, or run a hybrid, without altering how its agents are instrumented.

The first profile suits a small team or a startup, and is shown in Figure 3. Everything is self-hosted: Langfuse or Arize Phoenix for the agent traces and evaluations, SigNoz or OpenObserve for the infrastructure view. Licensing cost is effectively zero. The bill arrives instead as compute, storage and engineering attention, which is usually the right trade when the team is small and traffic is modest, and the wrong one the moment either of those changes.

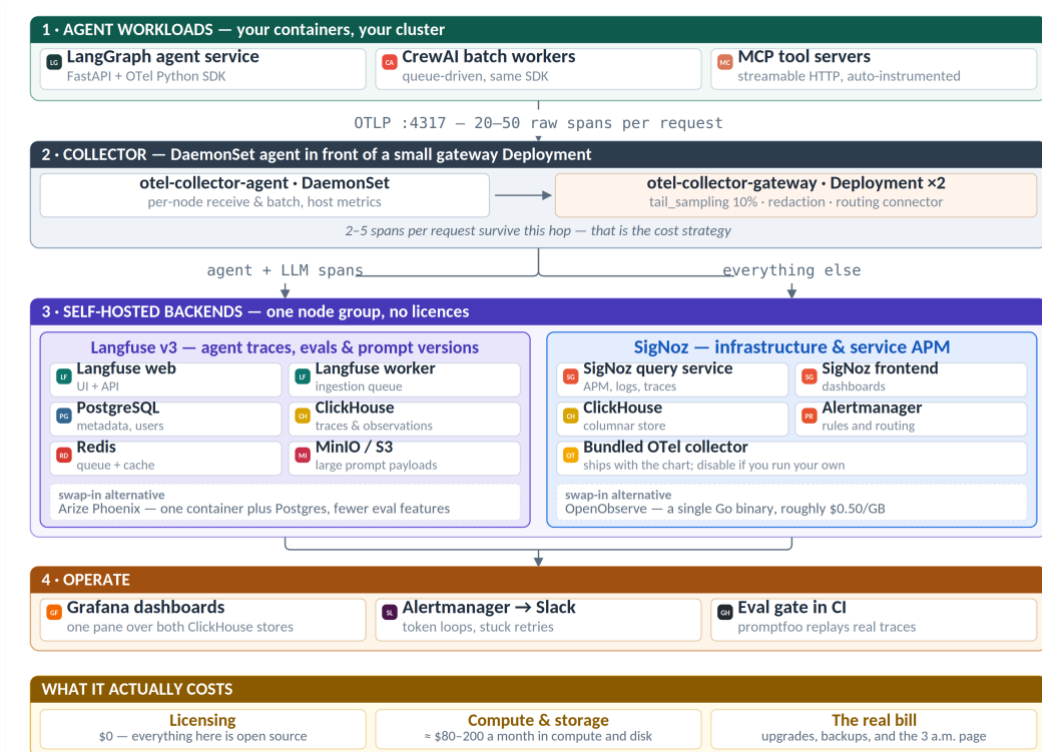


Figure 3. Lean or startup profile: the self-hosted stack component by component, with near-zero licensing cost

The second profile applies where the framework decision has already been made. Figure 4 shows a LangGraph application traced natively by LangSmith, which gives the deepest framework integration and a full run tree rather than a flat span list, while the same application also exports OpenTelemetry data to whichever application performance monitor the organisation already runs. Billing is per seat, so the cost grows with headcount rather than with traffic. That is an unusual property in this category and it cuts both ways, depending on the shape of the team.

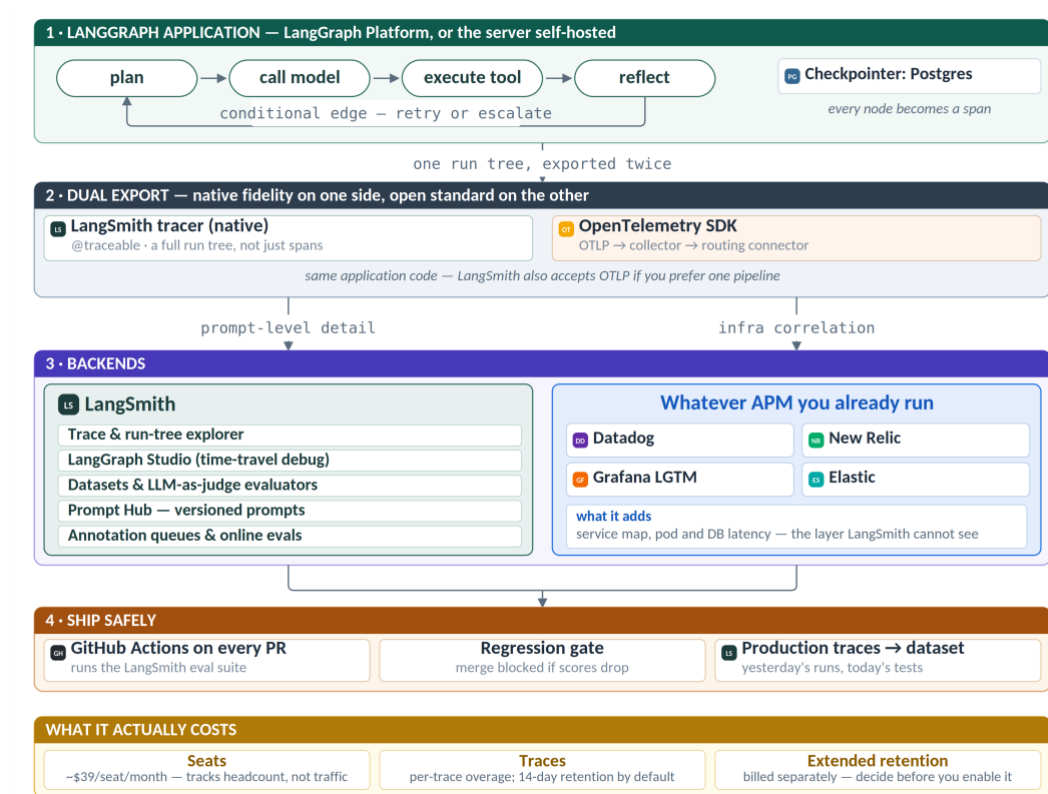


Figure 4. LangGraph profile: native LangSmith tracing alongside an OpenTelemetry export into an existing application performance monitor

The third profile consolidates everything into a single commercial platform, as in Figure 5. The destination is less interesting than the guardrails placed in front of it. Because billing is per monitored request, aggressive tail sampling, retention filters, metric cardinality caps and hard token quotas have to be configured on the first day rather than after an invoice has made the argument. At a million requests a day and twenty-five spans each, an unsampled deployment generates seven hundred and fifty million spans a month, which is the figure that terminates most pilots.

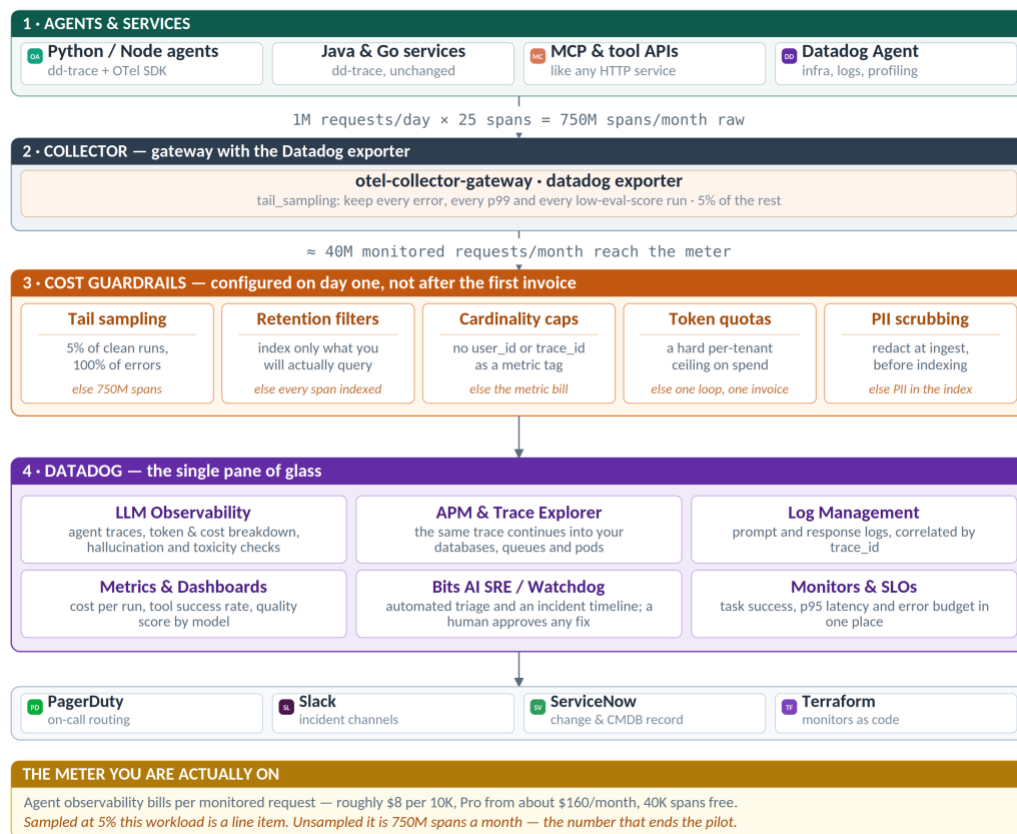


Figure 5. Enterprise profile: consolidation on a single platform behind five cost guardrails configured in advance

3.8. When the operator is also an agent

The reliability layer in Figure 1 includes an agent that triages alerts and drafts incident timelines. By 2026 this had developed into a field of its own, and the change was mostly architectural. Teams working hardest on the problem stopped attempting to build a single capable system that handled the whole incident and instead divided the lifecycle among several narrow agents, each with one job and an explicit hand-off to the next. The result resembles a well-organised on-call rotation more than it resembles a single expert. Table 4 sets out the division that recurred.

Table 4. Division of a single incident across a production agentic system

Agent	Responsibility
Detector	Watches metrics, logs and traces and raises a signal when behaviour departs from baseline. Everything downstream begins here.
Investigator	Takes the anomaly, queries the observability tools, gathers evidence and forms a hypothesis about the cause.
Knowledge	Searches past incidents, postmortems and runbooks for a matching prior occurrence.
Remediator	Executes the fix through a controlled gateway, subject to whatever approval step has been configured.
Verifier	Confirms that metrics returned to baseline and escalates when they did not.
Communicator	Maintains incident channels, status pages and stakeholder updates, and drafts the initial postmortem.
Guardian	Observes the other agents and intercepts hallucinations, policy violations and action loops before they cause harm.

Two design choices proved to matter more than the orchestration framework itself. The first is least privilege: each agent receives an explicit and minimal tool set, such as querying metrics, searching logs, running one named command or posting to a channel, so that a confused investigator cannot restart a cluster. The second is that hand-offs carry structured context rather than prose, so that what the detector observed reaches the investigator as data rather than as a paragraph requiring reinterpretation. Memory is layered accordingly: current incident state, a long-term corpus of past incidents, and a running record of what the agents themselves recently decided and how those decisions turned out.

The better platforms give all of these agents a shared representation to reason against, sometimes described as a production world model. It is a live, machine-readable picture of the environment: service topology and dependencies drawn from traces and the service mesh, recent deployments and configuration changes, current health, trusted runbooks, and the history of what has broken before and how it was repaired. It performs the function of the mental model a senior engineer carries, with the advantages that it does not go stale and that every participant, human or otherwise, is looking at the same one. That shared picture is what allows an agent to reason about blast radius and causation instead of pattern-matching without context.

For teams that would rather buy than build, Table 5 maps the market by origin. One widely cited projection places its growth from roughly twelve billion dollars in 2025 to above fifty billion by 2032. The precise figures deserve the scepticism any forecast deserves, although the direction is not seriously disputed.

Table 5. The agentic site-reliability tool landscape in 2026, grouped by origin

Category	Representative tools	Points worth knowing
Observability platforms adding agentic features	Datadog Bits AI, Dynatrace Davis AI, New Relic AI	The lowest-friction path for existing customers. Datadog shipped the first generally available agent of this kind; Dynatrace builds on a long-standing causal engine. Splunk and Honeycomb offer variants.
Pure-play autonomous incident response	Resolve.ai, Traversal, Sherlocks.ai	Startups pursuing autonomous resolution. Resolve targets roughly 80 percent hands-off operation and others advertise accuracy above 90 percent. These should be treated as claims until a blind test has been seen.
Incident management with added intelligence	incident.io, Rootly, PagerDuty AIOps	Chat-native workflows with intelligence applied to alert grouping, status updates and postmortems. incident.io reports use at Netflix and Etsy.
Open standards and open source	OpenTelemetry, CrewAI, LangGraph, the Grafana stack	The foundation beneath the rest. OpenTelemetry supplies the data layer, CrewAI and LangGraph build custom agent workflows, and Prometheus with Grafana, Tempo and Loki is adding its own machine learning.

Selecting one of these is a multi-year commitment, and six questions cut through most vendor presentations. Ask for blind-test accuracy figures rather than the three case studies that went well. Establish whether the tool reads the existing telemetry stack or expects re-instrumentation first. Examine the guardrails and the action gateway, specifically whether blast-radius limits, approval flows and an auditable trail can be configured. Determine whether each decision is traceable to reproducible evidence. Confirm the compliance posture and the integrations the workflow actually depends on. And understand the billing unit, the behaviour of inference cost at scale, and the likelihood that a given supplier survives the consolidation that a market growing this quickly normally produces.

4. Conclusion

Observability for agents is not a new discipline so much as an honest extension of an existing one. The instincts carry across unchanged: instrument what matters, keep the signal and discard the noise, and make reliability measurable rather than hoped for. What changes is the definition of a healthy system. A green dashboard means considerably less when an agent can fail politely, returning a success code while getting the answer wrong, so decisions, outcomes and cost have to be watched alongside uptime.

The analysis supports three specific conclusions. First, the billing unit a platform adopts affects the final invoice more than its headline rate does, because a single agent request produces twenty to fifty spans rather than one; per-trace and unit-based pricing models therefore behave very differently from flat per-span pricing under identical traffic. Second, the collector tier is the decisive architectural control, since it is the only place where sampling and redaction can be applied once and take effect across every downstream destination. Third, standardising on an open instrumentation format makes the choice of backend reversible, which is what allows the three deployment profiles in Section 3.7 to share an identical upper half.

The practical recommendation that follows is unglamorous. Instrument once against the open standard, place a collector in the middle where volume and privacy are genuinely controllable, and treat cost as a component of reliability rather than as an accounting concern that arrives later. Whether the operator on call is a person or a rotation of agents, the same discipline applies: begin smaller than seems necessary, keep a human approval step in front of any action that changes production, and install the cost guardrails before the traffic arrives rather than after the invoice does.

Several directions remain open. The cost model here assumes a uniform span count per request and would be improved by a distribution fitted to observed production traffic, particularly in the tail where long planning runs dominate. The accuracy claims made by autonomous incident-response vendors invite an

independent blind evaluation, which the published material does not yet supply. And the generative-AI semantic conventions remain experimental, so a study of how attribute stability affects long-lived dashboards and alerting rules would be useful to anyone committing to them now.

References

1. OpenTelemetry, "Inside the LLM Call: GenAI Observability with OpenTelemetry," *OpenTelemetry Blog*, 2026.
<https://opentelemetry.io/blog/2026/genai-observability/>
2. Google Cloud, "Agent observability," *Cloud Observability Documentation*, 2026.
<https://docs.cloud.google.com/stackdriver/docs/observability/agent-observability>
3. Google Cloud, "Observability in Google Cloud: overview," *Cloud Observability Documentation*, 2026.
<https://docs.cloud.google.com/stackdriver/docs>
4. Google Cloud, "Gemini Enterprise Agent Platform: observability overview," 2026.
<https://docs.cloud.google.com/gemini-enterprise-agent-platform/optimize/observability/overview>
5. Google Cloud, "Instrument generative AI applications," *Cloud Observability Documentation*, 2026.
<https://docs.cloud.google.com/stackdriver/docs/instrumentation/ai-agent-overview>
6. Google Cloud, "Instrument ADK applications with OpenTelemetry," *Cloud Observability Documentation*, 2026.
<https://docs.cloud.google.com/stackdriver/docs/instrumentation/ai-agent-adk>
7. Google Cloud, "Observability: cloud monitoring and logging," *Product documentation*, 2026.
<https://cloud.google.com/products/observability>
8. Google Cloud, "BigQuery pricing," 2026. <https://cloud.google.com/bigquery/pricing>
9. Google Cloud, "Gemini for Google Cloud pricing," 2026. <https://cloud.google.com/products/gemini/pricing>
10. Kablamo Engineering, "Tracing AI agents on Google Cloud with OpenTelemetry and Agent Engine," 2026.
<https://engineering.kablamo.com.au/posts/gcp-otel-adk-agent>
11. groundcover, "AI agent observability guide: telemetry, traces, metrics and evaluations," 2026.
<https://www.groundcover.com/learn/observability/ai-agent-observability>
12. Elastic, "Observability trends for 2026: generative AI and OpenTelemetry reshape the landscape," *Elastic Blog*, 2026.
<https://www.elastic.co/blog/2026-observability-trends-generative-ai-opentelemetry>
13. Egen, "The three biggest AI announcements from Google Cloud Next 2026," 2026. <https://egen.ai/insights/three-biggest-ai-announcements-from-google-cloud-next-2026/>
14. MonitoringCost, "GCP monitoring cost 2026: cloud operations suite pricing explained," 2026.
<https://monitoringcost.com/gcp-monitoring-cost>
15. Costimizer, "BigQuery cost optimization: the complete 2026 pricing guide," 2026.
<https://costimizer.ai/blogs/bigquery-cost-optimization>
16. CloudZero, "Google Vertex AI pricing: complete enterprise guide," 2026. <https://www.cloudzero.com/blog/google-vertex-ai-pricing/>