

Getting Started With Observra

Your AI agent is running. It calls a model, uses tools, makes decisions, and usually behaves as expected. Then something changes: a session costs 10 times more than usual, the agent calls the same tool repeatedly, or a user reports unexpected behavior without a reliable record of what happened.

Observra provides that record. It is an open-source Python SDK for Agent Telemetry and Observability, sponsored by Exabeam and licensed under Apache 2.0. Observra works with the agent framework you already use, including Google Agent Development Kit (ADK), Claude Agent SDK, OpenAI Agents SDK, LangChain and LangGraph, and Pydantic AI.

As your agent runs, Observra captures model calls, tool calls, costs, and errors. You do not need to rewrite the agent or add logging statements throughout your code. Observra uses the callbacks and instrumentation built into each supported framework, converts the telemetry into a consistent event format, and routes it to the destination you choose. Destinations include a local file, webhook, OpenTelemetry collector, or security information and event management (SIEM) system.

The result is one coherent behavioral record across supported frameworks. You can reconstruct what the agent did, understand what it cost, and determine whether its behavior was expected. This guide shows you how to get Observra running for the first time.

Before You Start

You need Python 3.10 or later and an existing agent built with one of the five supported frameworks. Observra is not an agent-building framework. It adds telemetry and observability to agents you have already created.

Add Observra to Your Project

Install the base package first. The base package includes three backends: a local JSON Lines (JSONL) file, a generic webhook, and a fan-out MultiBackend that writes to several destinations at the same time.

```
pip install observra
```

Next, install the extra for your agent framework:

```
pip install observra[adk]          # Google ADK
```

```
pip install observra[claude]       # Claude Agent SDK
```

```
pip install observra[openai-agents] # OpenAI Agents SDK
```

```
pip install observra[langchain]    # LangChain and LangGraph
```

```
pip install observra[pydantic-ai]  # Pydantic AI
```

Additional extras are available for OpenTelemetry, encryption, and Exabeam. You can install several extras together or use [all] to install everything. You do not need to select every destination or option at the start. If you later decide to export telemetry to an OpenTelemetry collector, encrypt stored data, or send events directly to an Exabeam SIEM, install the corresponding extra without changing the integration you already built.

Connect Observra in Three Steps

The five supported frameworks differ internally, but the Observra integration follows the same pattern: initialize Observra when the application starts, create the adapter for your framework, and attach the adapter where the framework expects plugins, callbacks, or instrumentation. Your agent logic stays unchanged.

For a Google ADK agent, use the following setup:

```
import observra

from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService

# Start telemetry and create the adapter for Google ADK.
observra.initialize(backend="jsonl", path="telemetry.jsonl")
plugin = observra.create_plugin("adk")

# Attach the plugin to the existing Runner. This is the only change.
runner = Runner(
    agent=root_agent,          # Existing agent, unchanged
    app_name="my-agent",
    session_service=InMemorySessionService(),
    plugins=[plugin],
)
```

After you attach the plugin, Observra writes the agent LLM calls, tool activity, and cost data to `telemetry.jsonl`. It writes each event as a separate line, without requiring logging statements inside the agent.

The other four frameworks use the same `initialize()` and `create_plugin("<framework>")` pattern. Their internal keys are `claude`, `openai`, `langchain`, and `pydantic_ai`. Only the attachment point changes. One framework may use a plugin list, another may register a callback, and another may expose an instrumentation hook. Review the quickstart for your framework before adapting the Google ADK example:

- [Google ADK quickstart](#)
- [Claude Agent SDK quickstart](#)
- [OpenAI Agents SDK quickstart](#)
- [LangChain and LangGraph quickstart](#)
- [Pydantic AI quickstart](#)

Review the Telemetry

After a run, open `telemetry.jsonl`. A single model call looks similar to this example:

```
{
  "timestamp": 1718115781.882,
  "event_type": "model_response",
  "framework": "adk",
  "agent_name": "research-agent",
  "model_name": "gemini-2.0-flash",
  "session_id": "s-a1f6c2e3",
  "library_version": "1.0.3",
  "data": {
    "input_tokens": 1240,
    "output_tokens": 387,
    "cost_usd": 0.0019,
    "action": "call_llm",
    "result": "success"
  }
}
```

This consistent structure is a core Observra differentiator. An event from a Google ADK agent and an event from a LangGraph chain use the same shape. Observra calls this structure its Common Information Model (CIM). Because Observra normalizes the data, dashboards, alerts, queries, and downstream analytics do not need separate logic for each agent framework.

Observra also applies several controls automatically after telemetry starts flowing:

- **Cost tracking:** Uses a model-specific pricing catalog to track session cost, which helps teams identify unexpected spending.
- **Default redaction:** Redacts secrets and personal data from events by default.
- **Prompt-injection signal:** Uses a lightweight heuristic to check the event stream for signs of prompt injection.
- **Non-blocking pipeline:** Prevents a slow backend or sudden traffic spike from adding latency to the agent. If the pipeline falls behind, Observra drops the oldest events instead of stalling the application.

Choose Where the Data Goes

A local JSONL file is often the fastest way to start, especially during development or in an air-gapped environment. As the agent moves toward production, route the same normalized telemetry to systems your team already uses. Every backend receives the same event object, so changing destinations is a configuration decision rather than an application rewrite.

A webhook can send events to a SIEM ingestion endpoint. MultiBackend can send the same events to several destinations at once. For example, you can retain a local file for debugging while exporting telemetry through OpenTelemetry for shared dashboards and downstream analysis.

The following example sends spans and logs to an OpenTelemetry collector. From there, the data can flow to Dynatrace, Grafana, Honeycomb, New Relic, or another OpenTelemetry Protocol (OTLP)-compatible platform:

```
from observra.backends.otel import OTelExportBackend
from observra.backends.otel_log import OTelLogBackend
from observra.backends.multi import MultiBackend

span_backend = OTelExportBackend(
    endpoint="https://your-collector/v1/traces",
    headers={"Authorization": "Api-Token ..."},
    service_name="my-agent-svc",
)

log_backend = OTelLogBackend(
    endpoint="https://your-collector/v1/logs",
    headers={"Authorization": "Api-Token ..."},
    service_name="my-agent-svc",
)

backend = MultiBackend([span_backend, log_backend])
```

Because the agent-native telemetry retains a consistent CIM structure while moving through different backends, teams can preserve one behavioral record for debugging, audit, investigation, governance, and downstream behavioral analytics.

Next Steps

After the basic integration works, use these resources to deepen or operationalize the deployment:

- **Event schema guide:** Review the complete event contract.
- **Architecture page:** Understand how telemetry moves from capture through the backend pipeline.
- **Production deployment guide:** Compare backend options and operational tradeoffs beyond a laptop or development environment.
- **Compatibility guide:** Confirm supported framework and Python versions.
- **Sample telemetry stream:** Review representative output before configuring an integration.

About Exabeam

Exabeam is the leader in behavior intelligence for the agentic enterprise. As organizations deploy digital workers and confront machine-speed adversaries, Exabeam delivers flexible, industry-proven solutions for insider threat coverage of humans and agents and faster, more accurate threat detection, investigation, and response (TDIR). Learn more at www.exabeam.com.



Learn more at
www.exabeam.com →

Without limitation, the Exabeam and LogRhythm names and logos, related product, service, and feature names, and related slogans are service marks, trademarks, or registered marks of Exabeam (or its affiliates) in the United States and/or other countries. All other brand names, product names, or trademarks belong to their respective owners.
© 2026 Exabeam, LLC. All rights reserved.