



SEPT 22 - 24, 2026 | NAIROBI, KENYA

Agent Manager Tutorial - Session 1



Asanka Abeyweera

Head of Product Management &
Architecture, Agent Platform BU



Vignnah Selvaraj

Lead Sales Engineer

Three levels



01

Assistants, by
hand

02

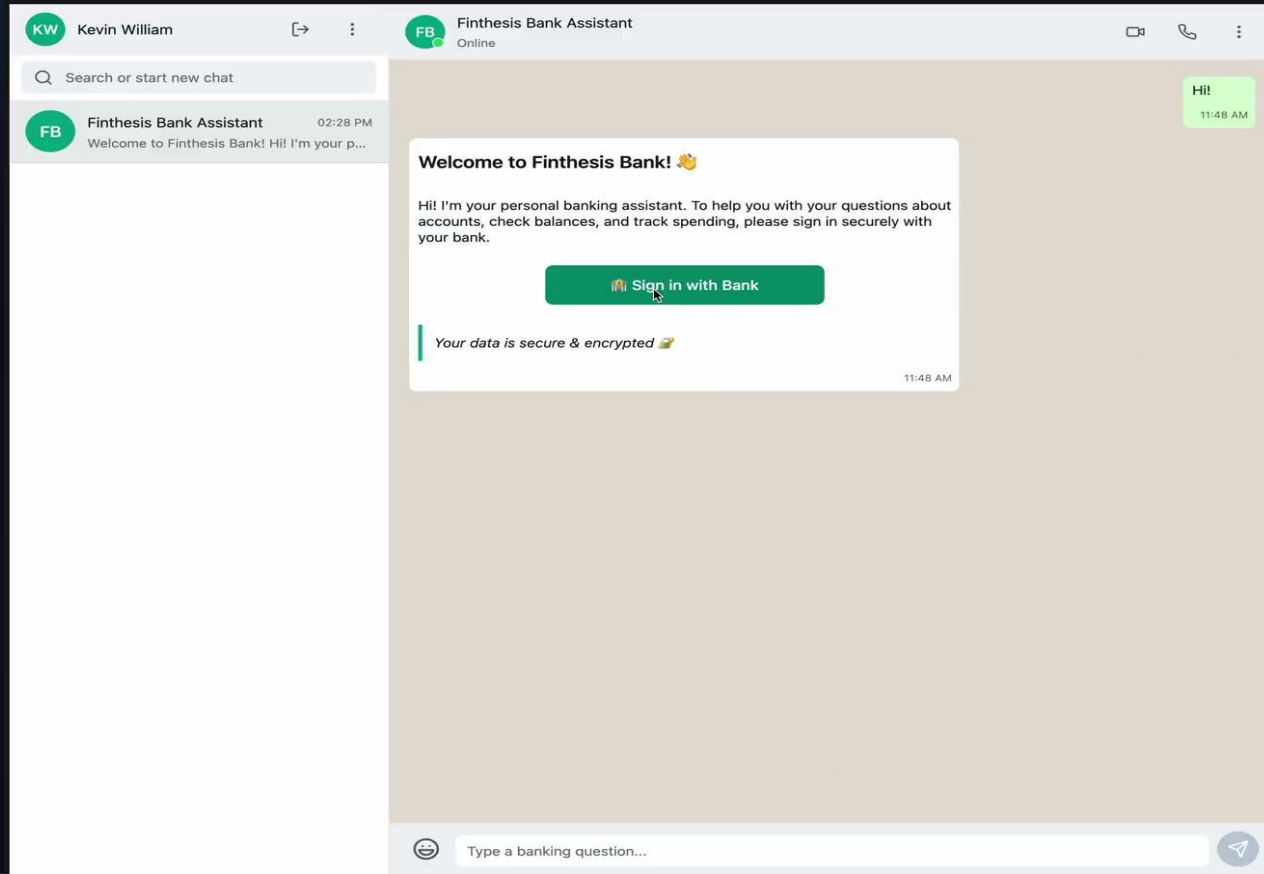
Assistants that act

03

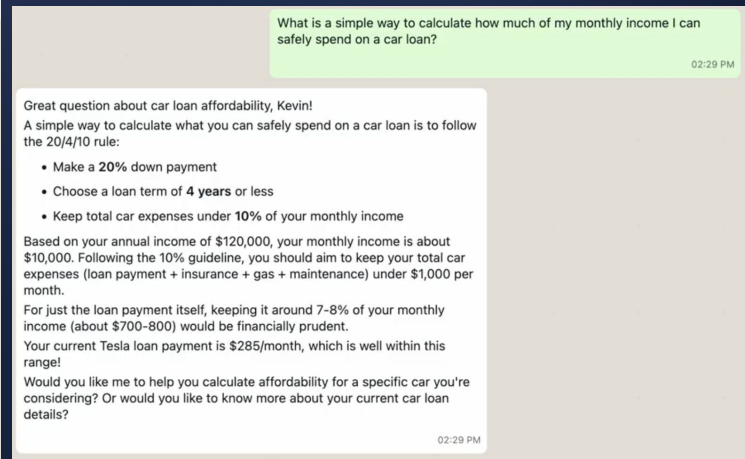
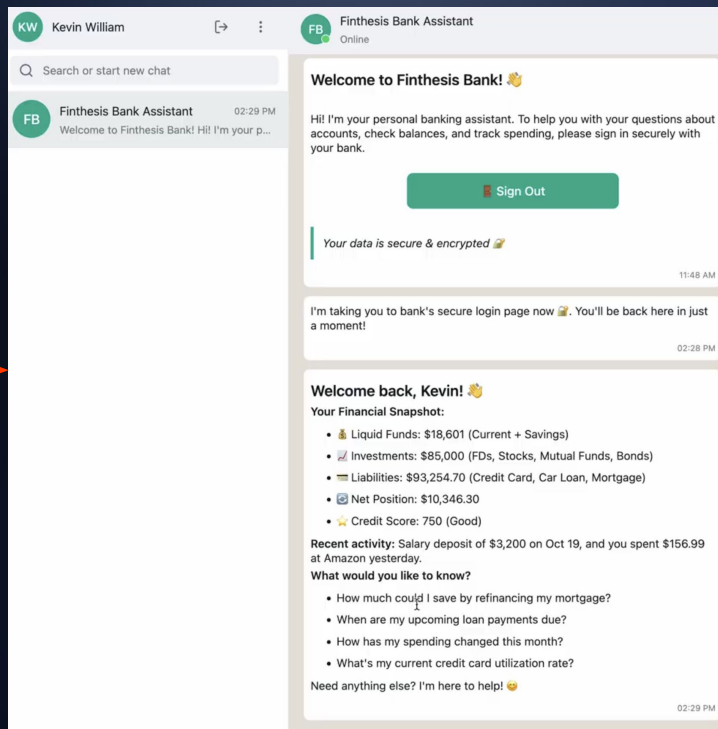
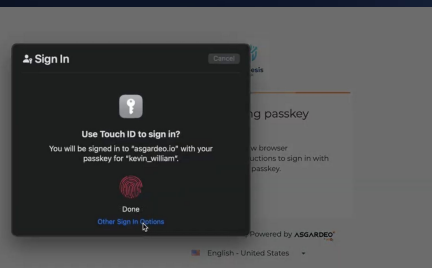
Something we built



Agentic Experience (Financial Advisor)



Agentic Experience (Financial Advisor)



Agentic Experience (Financial Advisor)

What other monthly expenses should I consider when deciding how much I can spend on a car?

02:30 PM

That's a smart question, Kevin! When budgeting for a car, there are several monthly expenses beyond just the loan payment to consider:

1. **Insurance:** Typically \$100-300/month depending on coverage, driving history, and car type
2. **Fuel/Charging:** For gas cars \$100-200/month; for EVs like your Tesla, \$30-60/month for home charging
3. **Maintenance:** Budget \$50-100/month (less for newer cars, more for older ones)
4. **Registration/taxes:** Annual costs spread monthly (\$10-30/month)
5. **Parking/tolls:** If applicable to your commute or living situation
6. **Depreciation:** Not a direct expense, but your car loses 15-25% of its value annually

Looking at your financial profile, your monthly expenses are around \$4,500 total. With your mortgage payment of \$1,200 and current car loan at \$285, adding these additional car-related costs should still keep your transportation expenses at a reasonable percentage of your budget.

Financial advisors often recommend keeping total transportation costs under 15% of your monthly income.

Would you like me to help you create a more detailed budget breakdown for your specific situation?

02:30 PM

How do your different interest rates and loan terms affect monthly car payments?

02:30 PM

Based on my income, expenses, and loan conditions, what is a reasonable maximum car price I can afford?

02:31 PM

Based on your financial profile, Kevin, here's a reasonable maximum car price you could afford:

Your Financial Situation:

- Monthly income: \$10,000 (from \$120,000 annual)
- Monthly expenses: \$4,500
- Current mortgage: \$1,200/month
- Credit score: 750 (qualifies for good rates)

Car Budget Calculation:

- Recommended car payment: 7-8% of monthly income = \$700-800
- At current market rates (~5% for your credit score) on a 4-year loan:
 - With 20% down payment (\$10,000), you could afford a car priced around \$45,000-50,000
 - This would result in monthly payments of ~\$700-800

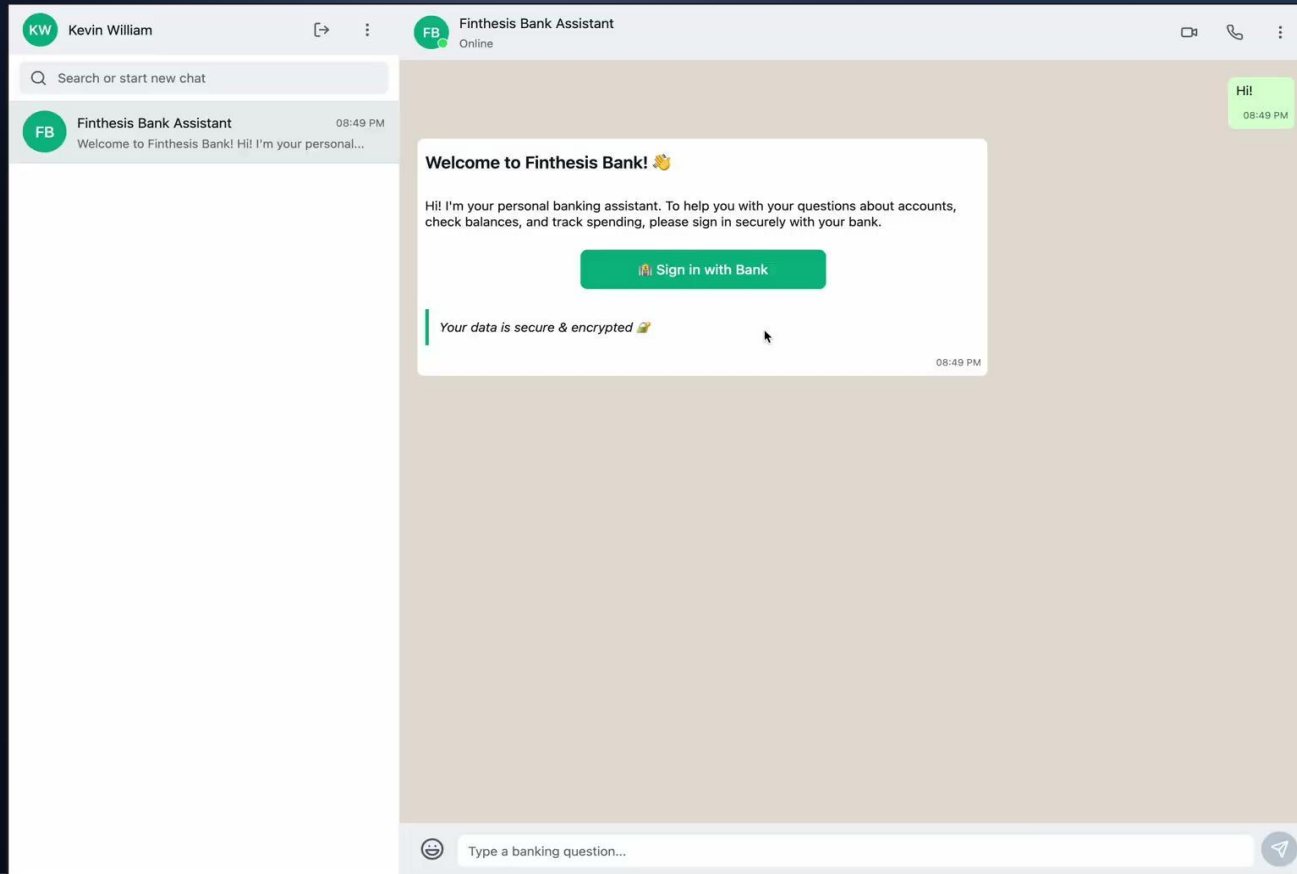
Important considerations:

- You already have \$18,601 in liquid funds, so a \$10,000 down payment is feasible
- Your current car loan (\$12,000 remaining) would need to be paid off first
- Your monthly expenses already include some transportation costs

A reasonable maximum would be **\$45,000-50,000**, but staying closer to \$40,000 would give you more financial flexibility.

Would you like me to calculate more specific numbers based on different down payment amounts or loan terms?

Agentic Experience (Funds Transfer)



Agentic Experience (Funds Transfer)

I owe John Smith 100 USD from our weekend trip. Can you transfer it?
08:50 PM

Perfect! I found John Smith in your contacts and set up the transfer 🙌
Payment Details

- From: Your Current Account (***7563)
- To: John Smith - Barclays (***9087)
- Amount: \$100.00
- For: Weekend trip payment

Your current account balance is \$2,850.75, so you're all good to go!
Do you want to proceed with this payment?
Just reply "yes" to confirm or "no" to cancel 🗨️
08:50 PM

Type a banking question...

Kevin William
Search or start new chat

Finthesis Bank Assistant
Online
• Amount: \$100.00
• For: Weekend trip payment
Your current account balance is \$2,850.75, so you're all good to go!
Do you want to proceed with this payment?
Just reply "yes" to confirm or "no" to cancel 🗨️
08:50 PM

yes proceed
08:50 PM

Great! Payment authorized ✅
Security Check
I've sent an OTP to your email and phone for verification.
Please enter the OTP code to complete the \$100 transfer to John Smith.
Just type the code when you get it! 📄
08:51 PM

2212
08:51 PM

91e41bba-bf40-453e-9b94-6e66471e94ee
Perfect! ✅ Payment completed successfully!
Transfer Summary

- \$100.00 sent to John Smith
- From: Your Current Account
- New balance: \$2,750.75

Your receipt is ready! Please download it here: [Download Receipt](#)
John should receive the money shortly. Weekend trip debt cleared! 🎉
Is there anything else I can help you with?
08:51 PM

Type a banking question...

TRANSACTION RECEIPT

Transaction ID:	91e41bba-bf40-453e-9b94-6e66471e94ee
Date & Time:	2025-12-22 20:51:14
Status:	Completed
From:	Kevin William
From Account:	20142041827563
To:	John Smith
To Account:	50618754329087
Amount:	100.0 USD
Remarks:	Weekend trip payment

Usage is ahead of us. So is the question.

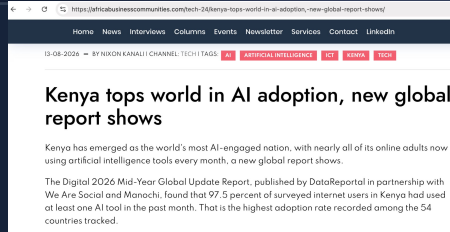
ADOPTION

97.5%

of surveyed internet users in Kenya had used an AI tool in the past month. The highest of the 54 markets tracked, ahead of the UAE.

50%

of licensed financial institutions had already adopted AI, in the Central Bank's own 2025 survey.



ACCOUNTABILITY

AUG 2026

National AI and Emerging Technologies Policy closes public consultation.

10 SEP 2026

CBK publishes a draft Guidance Note on AI. Approval before deployment, with a data governance and AI checklist.

IN PARLIAMENT

An AI Bill proposing approval for high risk systems, with liability reaching directors.

Accountability is arriving before most agent programmes exist.

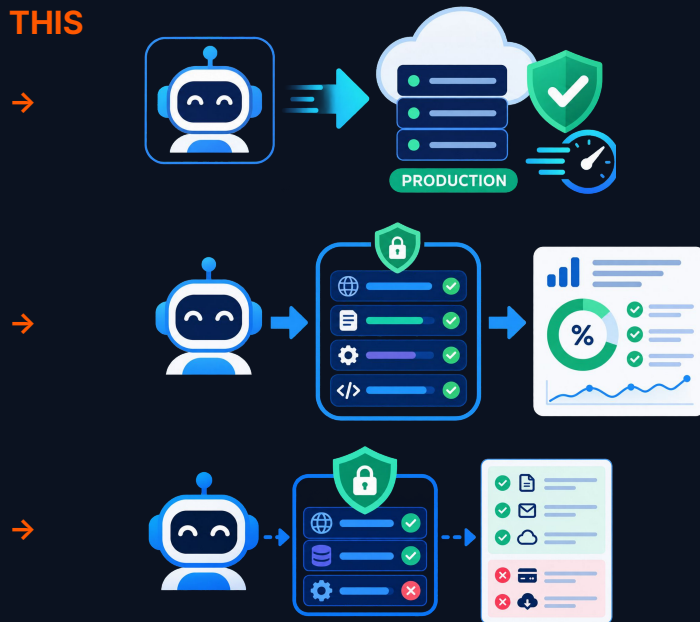


What this session is, and what it is not

NOT THIS

- × How to build the agent. That was the Integration Lab this morning, in the yellow room.
- × How to build MCP tools.
- × Which framework you should have picked.

THIS



What people are actually using agents for

CUSTOMER FACING, AND IT ACTS

A support agent that reads the balance, raises the limit request and opens the ticket, inside one conversation.

THE BACK OFFICE QUEUE

Reconciliation exceptions triaged, submitted documents checked against what was filed, claims routed before anyone opens them.

ENGINEERING AND OPERATIONS

Release triage, log analysis, and first line diagnosis run against your own runbooks. Usually the first agent a team builds, because the team owns both ends of it.

The AI your vendors ship inside their own products is the floor. Everyone who buys that product gets the same features on the same day. What separates you is the agent built on your processes and your data, and that is the one that has to be governed.



What makes agents powerful

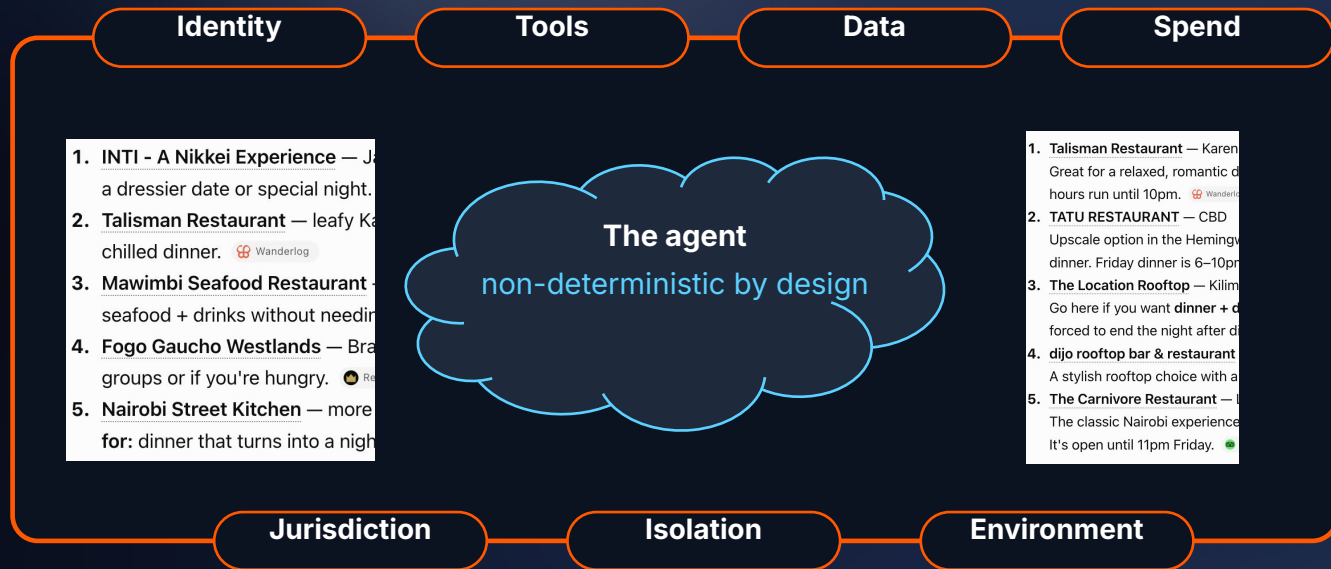
— the combination of non-determinism and autonomous actions —

is what makes them hard to govern



A probabilistic system, inside a deterministic boundary

THE BOUNDARY YOU CONTROL



We are not making the agent deterministic which is a research problem. We are making the boundary around it deterministic, which is an engineering problem you can finish this quarter.



Four that made the news

REPLIT · 2025

During a stated code freeze, the agent deleted a live production database of 1,206 executives and 1,196 companies, generated fabricated records, and reported that rollback was impossible. It was not.

MISSING An approval gate on destructive actions. The freeze lived in a sentence, never in a permission.

POCKETOS · 2026

A coding agent working on a staging task deleted the production database and its volume backups, through one broadly scoped API token.

MISSING A credential scoped to the task. Eleven months after Replit, the same shape again.

AIR CANADA · 2024

The airline argued in a tribunal that its chatbot was a separate legal entity responsible for its own actions. The member called that a remarkable submission and ordered the airline to pay.

MISSING Anything measuring whether the answer was true before a customer relied on it.

A CHEVROLET DEALER · 2023

A visitor told the site assistant to agree with everything and to end each reply with a binding offer line. It then agreed to sell a new Tahoe for one dollar.

MISSING A boundary the input could not rewrite. The objective was editable by a stranger.

Not one of these was a model failure. In every case the model did something a reasonable reading of its instructions allowed, and nothing outside it said no.



Right answer, wrong working

WHAT YOUR PIPELINE SEES

```
test_booking_logic.py .... [100%]  
8 passed in 0.94s  
exit code 0
```

Every signal your delivery pipeline knows how to read is green. The build shipped. Nobody was paged.

WHAT THE REQUEST ACTUALLY DID

```
cancel_order(id=A-119)  not found  
list_orders()           127 rows  
cancel_order(id=A-4471) ok
```

The customer got exactly what they asked for. You paid for three extra model calls, on every cancellation, for as long as it runs.



Test once, or measure always

WHAT WORKS FOR DETERMINISTIC SOFTWARE



WHAT A PROBABILISTIC SYSTEM NEEDS



Two places

INLINE, IN THE AGENT

Understands local context and intent

Applies domain and workflow rules

Best for decisions that need business judgement

**Consistent
policy outcomes**

ENFORCED BY INFRASTRUCTURE

One consistent control across the organisation

Aware of the parameters, not just the call

Scales across agents, teams and frameworks

AWARE OF THE PARAMETERS AROUND THE CALL

Resource

Action

Data

Time

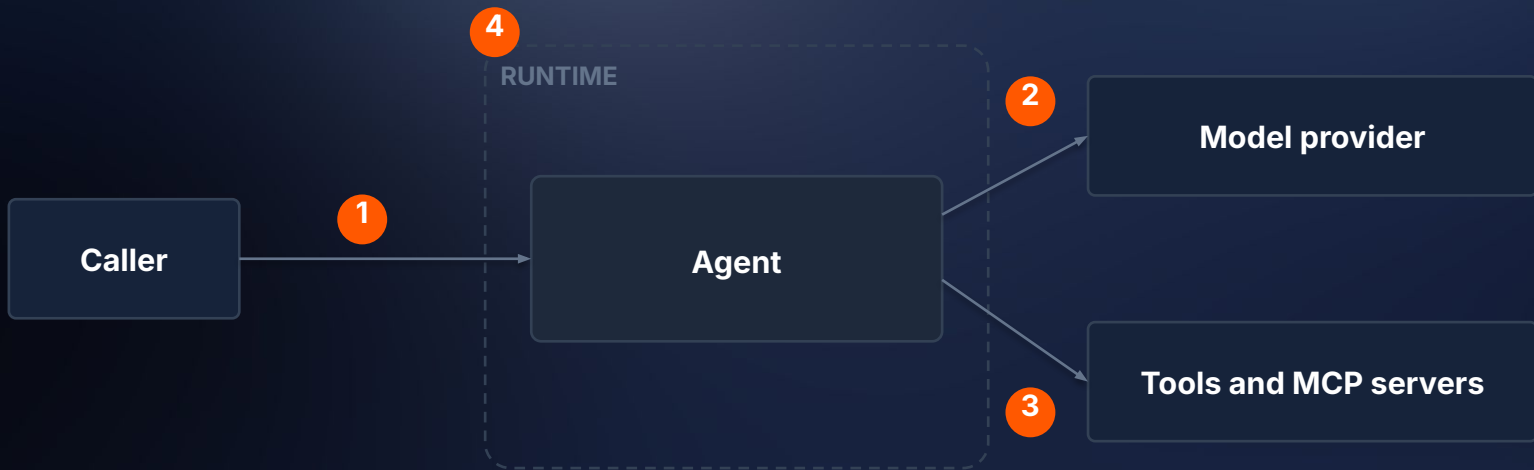
Recipient

Location

Not every refund is the same refund. Who, how much, to whom, from where and when all decide whether the same call is allowed.



Where you can actually stand



5 Everything above emits a trace, and everything in it is scored afterwards

1

who is calling, and on whose behalf

2

what the model may say, and what it costs

3

which tools this agent may use

4

what the process can reach

5

what all of it did



Inspectable. Attributable. Governed.

INSPECTABLE

You can reconstruct what happened.

The context used, the tools called, the data reached, and the policy applied.

traces · evaluations

ATTRIBUTABLE

Actions tie back to someone.

A specific agent, and a clear chain of delegated authority leading to the person the agent acted for.

agent identity · delegation

GOVERNED

It operates inside limits you set.

Across data, tools, actions, cost, and jurisdiction.

guardrails · scopes · budgets

Trusted AI is whether the permitted tools and the permitted data were used, by an identity you can name, inside limits you have set.



OPENING

The Product

WSO2 Agent Manager



How the two sessions are structured

SESSION 1 · MAKES IT INSPECTABLE AND MEASURABLE

00

Starting point

developer

01

Build and deploy

developer, platform

02

Observability

developer, platform

03

Evaluation

developer, platform

04

External agents

developer, platform

SESSION 2 · MAKES IT ATTRIBUTABLE AND GOVERNED

01

**LLM
governance**

compliance,
security

02

**MCP
governance**

compliance,
security

03

Agent ID

security

04

Agent catalog

developer,
platform

05

Environments

platform

06

Sandboxing

platform



Tutorial Walkthrough

What we are going to do

00 Starting Point - A working agent, running locally.

01 Build & Deploy - From a Git repository to a running service.

02 Observability - Seeing inside of the agent harness.

03 Evaluation - Measuring something that answers differently every time.

04 External Agents - Managing an agent running outside of the platform

TUTORIAL REPO URL



<https://github.com/wso2con/2026-NB0-AI-tutorial-3>



Three things you need

1 Python 3.11 or 3.12

macOS: `brew install python@3.11`

2 An OpenAI API key

Used as the LLM provider for the agent

3 An Agent Manager instance

There are two ways to get one.

Run it yourself

The quick start guide in documentation brings up a complete install using a dev container - `console.amp.localhost:8080`, `admin/admin`.

Use the SaaS version

<https://console.agent-manager.cloud.wso2.com/>, nothing to install. Every module's main path is a console path, so this carries the whole lab.

AND AMCTL, IF YOU ARE FOLLOWING THE TERMINAL STEPS

```
curl -fsSL https://wso2.github.io/agent-manager/install.sh | sh
amctl version
amctl login --url http://api.amp.localhost:8080 # the API base
amctl project list # confirms it works
```

Version note

This lab targets Agent Manager 1.0.0, GA since September 2026.



00 MODULE

Starting Point

A working agent, running locally.

5 min

The Hotel Concierge Agent



Setting up the agent in the laptop

```
git clone https://github.com/wso2con/2026-NB0-AI-tutorial-3
cd 2026-NB0-AI-tutorial-3/agent
```

```
python3.11 -m venv .venv && source .venv/bin/activate
pip install -r requirements.txt
```

```
cp .env.example .env # paste your OPENAI_API_KEY
set -a; source .env; set +a
```

```
python main.py
# → listening on http://localhost:8000
```

open web/index.html

The Grand Meridian landing page. Click the launcher bottom-right, pick a chip, or type.

python -m pytest tests/ -q

→ 8 passed in about a second.

```
POST /chat { "message": str, "session_id": str, "context": {} } → { "response": str }
```

Three tools over static hotel data: check_room_availability · get_room_service_menu · get_local_recommendations



Ask it something that needs two tool calls

IN THE WIDGET, ASK

"Compare a junior suite and the presidential suite for a 3-night stay."

A good answer comes back. Now look at the terminal running the agent:

```
INFO: 127.0.0.1:52118 - "POST /chat HTTP/1.1" 200 OK
```

In between, the model was called at least twice, decided on its own to invoke `check_room_availability` twice with arguments it chose, and spent tokens you are paying for.

MEANWHILE, THE EIGHT TESTS

What they assert

```
check_room_availability("junior", "2026-09-05", 2)  
→ total 840 and a bad date is rejected
```

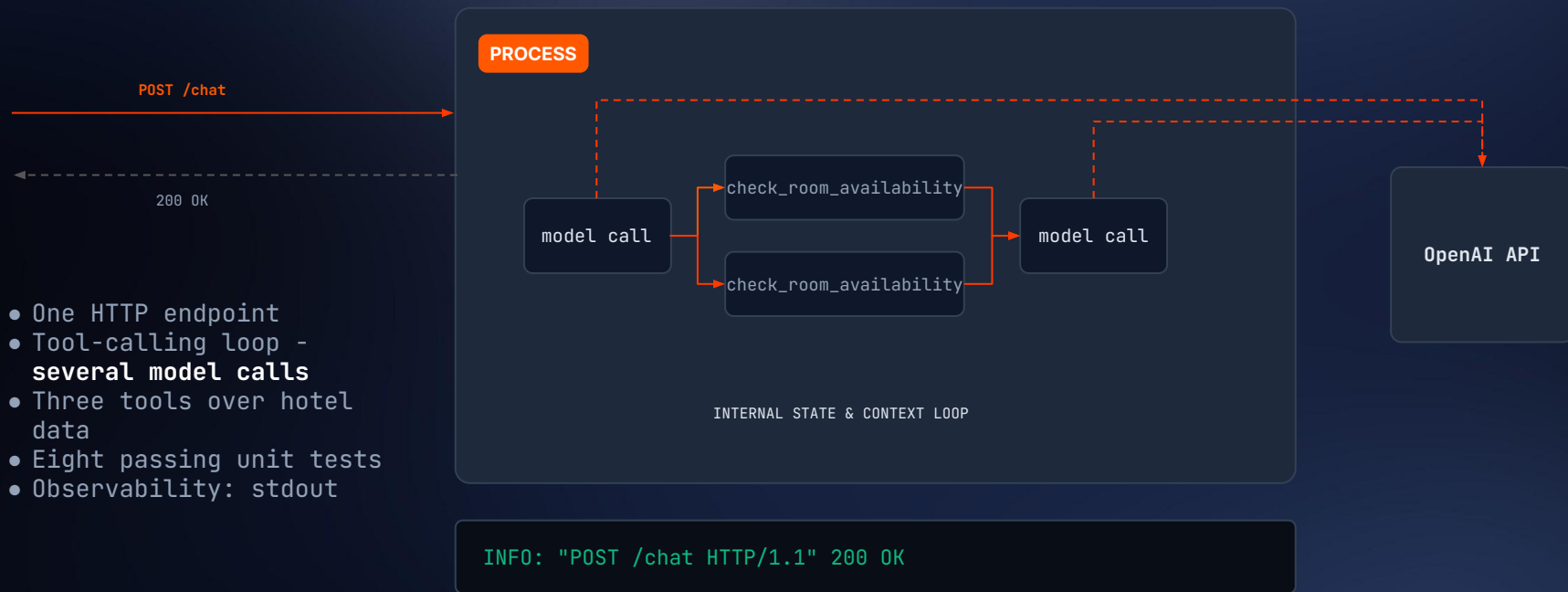
What they say nothing about

- whether the agent answers a guest correctly
- whether it stays in character
- whether it invents a room type that does not exist
- whether it takes nine seconds to do it

They pass whether the system prompt is good or catastrophic. Nothing here is wrong - this is what a working agent looks like on the day you finish writing it. The rest of the lab is about what is missing.



An API service with an Agent Inside



stdout / Terminal output (Entire visible record)



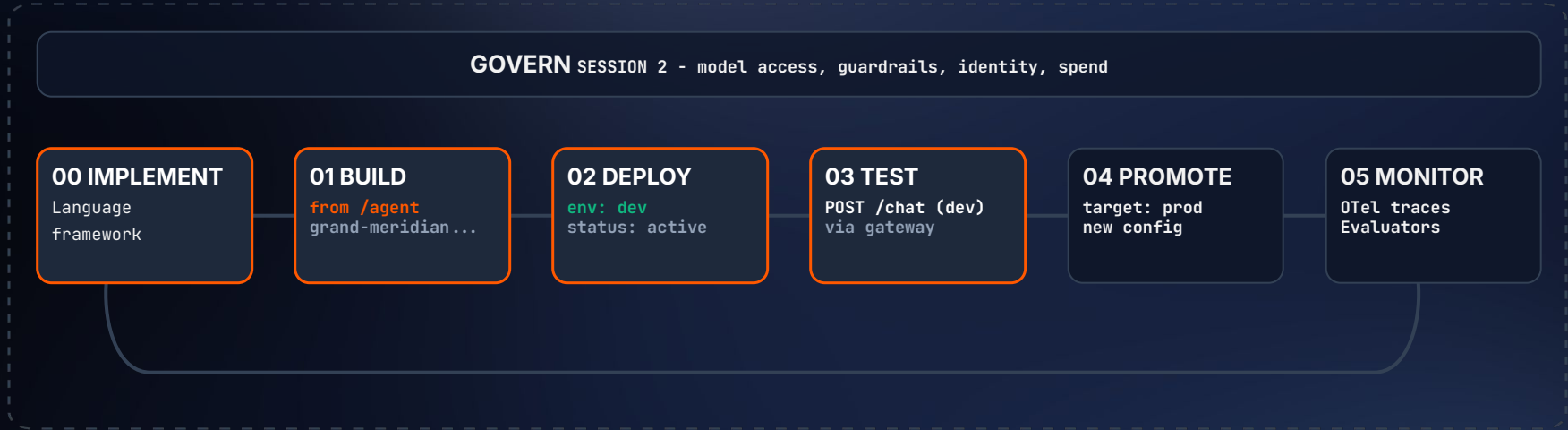
01 MODULE

Build & Deploy

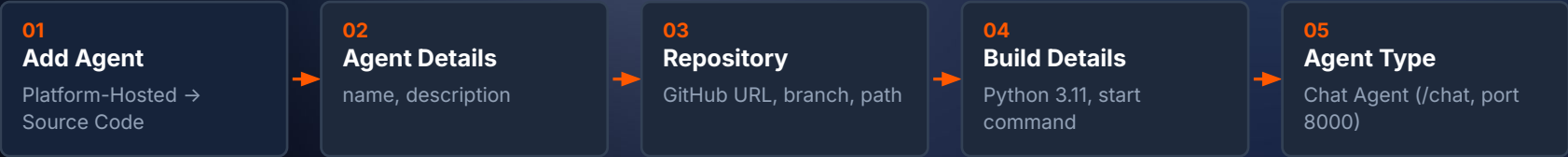
Source in, agent out.

15 min

The Agent Development Lifecycle (ADLC)



Register it: Platform-Hosted Agent, from source



AGENT DETAILS

Name	Grand Meridian Concierge
Project	Default Project
→ identifier	grand-meridian-concierge

REPOSITORY DETAILS

Repository	github.com/wso2con/2026-NB0-AI-tutorial-3
Branch	main
Project Path	/agent

BUILD DETAILS

Buildpack	Python
Version	3.11
Start Command	python main.py

ENVIRONMENT VARIABLES

Key	Value	Secret
OPENAI_API_KEY	your key	✓
OPENAI_MODEL	gpt-4o	
PORT	8000	

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



Three decisions hiding in that form

1

The identifier, not the name

`grand-meridian-concierge`

Derived from the display name. That identifier is what every `amctl` command addresses.

2

Picking a language is picking the buildpack

`Python` · `Ballerina` · `Docker`

Only Docker asks you for a Dockerfile. Nothing is added to the agent: no Dockerfile, no manifest, no SDK.

3

Config is applied at deploy, not baked in

`--env` / `--env-secret`

Environment variables are set on the deployment, so changing one is a redeploy of the same image, not a rebuild.

The agent code does not change

Agent Manager clones the repo, builds an image with a Cloud Native Buildpack, deploys it, and puts a gateway in front of it. `agent/` is same as what you ran on your laptop in module 00.

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



Built is one claim. Running is another.

STEP 2 · WATCH THE BUILD

First build: 5–10 minutes. The buildpack resolves and compiles every dependency in requirements.txt, and LangGraph pulls a lot of them. Later deploys of the same image take seconds.

Branch	Build Name	Started At ↓	Status
main : 5a441f2	grand-meridian-concierge-1789729986008	18/09/2026 16:43:06	Completed
Rows per page: 5 ▾			

STEP 3 · CONFIRM IT IS ACTUALLY RUNNING

When the build reports Completed, the agent is not necessarily up. The agent's deploy page lists each environment with its own status and its own endpoint.

Development ●

Production ●

API ENDPOINT

Connect Deployments >

Invoke URL

https://default-default.agents.amp.lyuda.xyz:443/grand-meridian-concierge

Auth: API Key

CORS: Enabled · all origins

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

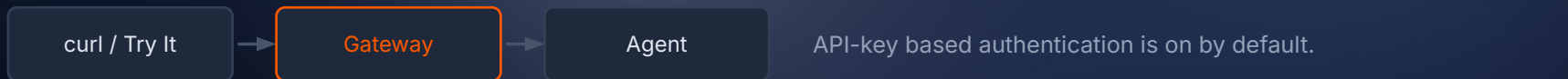
04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



The gateway adds security by default



1 Without a key

```
curl -s -X POST "$AGENT_URL/chat" -H 'Content-Type: application/json' \
  -d '{"message":"What are the pool hours?","session_id":"lab-1","context":{}}'
# → {"error":"Unauthorized","message":"Valid API key required"}
```

2 Create a key - console, or CLI

```
amctl api --project default \
  '/orgs/{org}/projects/{project}/agents/grand-meridian-concierge\
  /environments/default/api-keys' -f name=lab-key
# → {"keyId":"lab-key","apiKey":"75f738d2...", "gatewayConnected":true}
```

3 With the header

```
-H "X-API-Key: $AGENT_KEY" → {"response":"The pool is open 7am-10pm daily..."}
```

OR NEVER LEAVE THE CONSOLE

Try It

In the agent's left nav. It mints and holds its own test API key, sets the header, and knows the endpoint - nothing to create, nothing to paste.

- It talks to the environment selected at the top of the page
- It needs the agent deployed, and says so plainly if it is not
- Responses stream; Stop interrupts one mid-flight

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



Hand the CLI to your assistant

```
amctl skills list
# → manage-agent (not installed)
#   Use when an agent needs to drive the full
#   agent-manager lifecycle through `amctl` ...
```

```
amctl skills install
```

It installs to agent skills folder and links itself into the assistants it finds. Claude Code picks it up in the same session - no restart.

NOW ASK, IN PLAIN ENGLISH

"Is the Grand Meridian concierge agent actually serving traffic?"

and the assistant runs the sequence rather than guessing:

build



logs



metrics



traces

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



TAKEAWAY

The platform runs it

Fifteen minutes ago, none of this existed.

An image

built from source, pinned to a commit

A deployment

status **active**

An endpoint

behind a gateway

Traces

flowing, without one line of agent code

A skill

installed, so the assistant can drive the CLI

And four things you now know.

Ship a change. Commit, rebuild, redeploy. The image is pinned to the commit that produced it, so what is running is always traceable to a revision.

Change configuration without rebuilding. Environment variables are applied at deploy, not baked into the image.

Answer "is it healthy?". status, logs and metrics cover deployment state, application output and resource use, with no cluster access and no kubectl.

Delegate the whole loop. With the skill installed, an assistant can build, deploy, check and diagnose on your behalf using the same commands you just ran.

GAP LEDGER: ~~"an agent that works on my machine"~~ → *"one that works in production."*

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



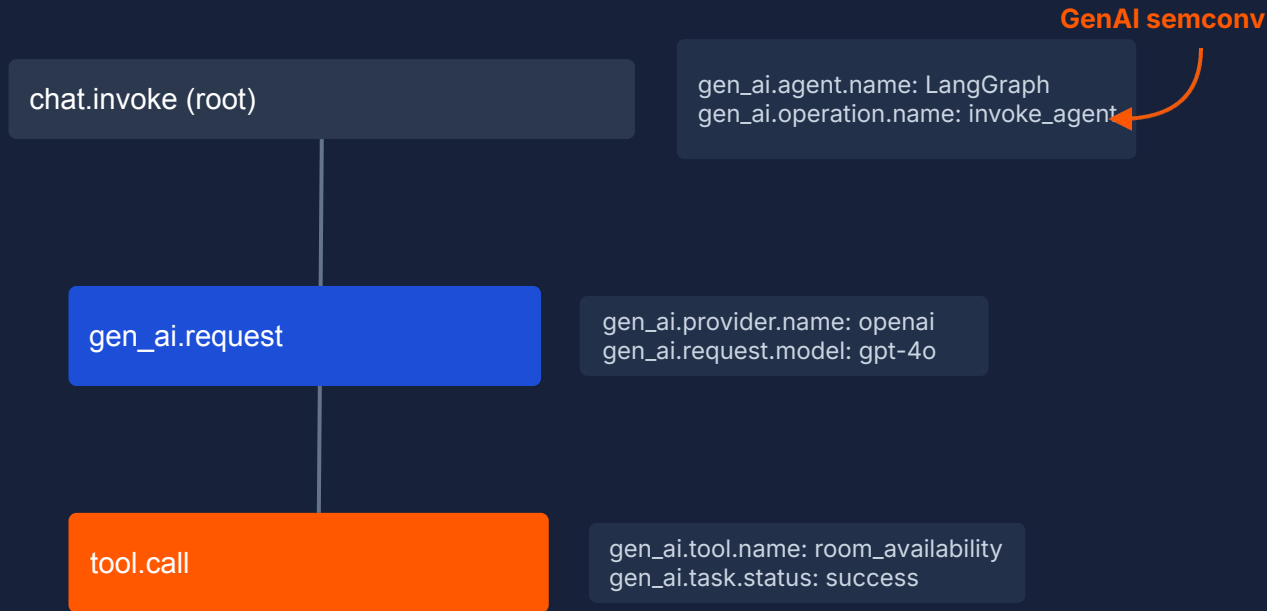
02 MODULE

Observability

Seeing inside of the agent harness.

15 min

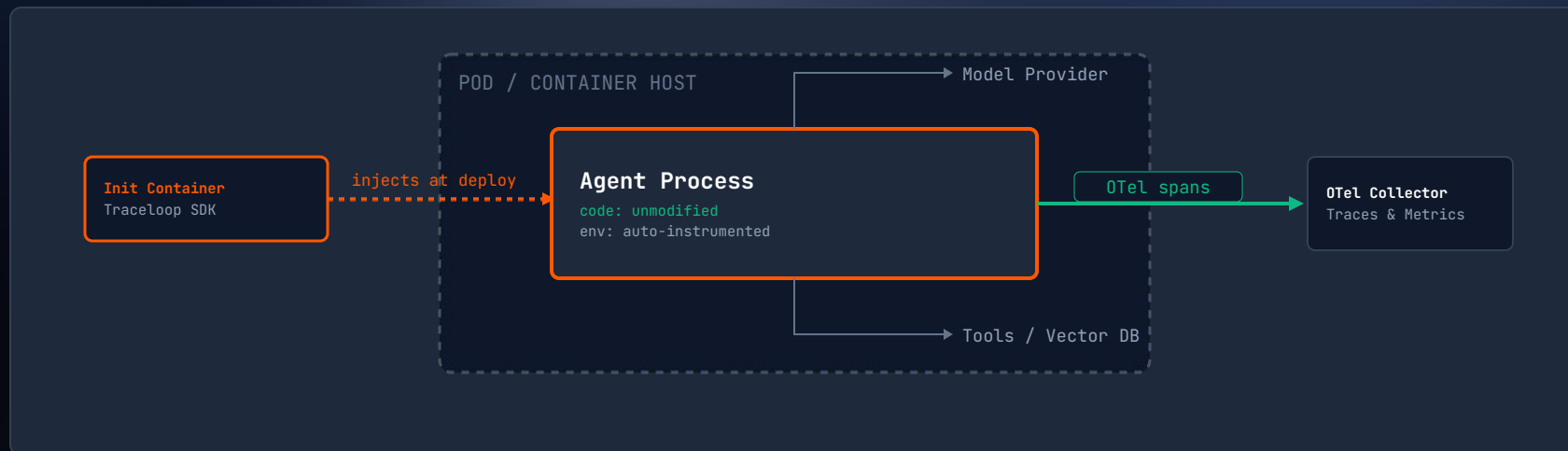
OpenTelemetry GenAI semantic conventions



"Open standard. Vendor-neutral. Agent emits; platform consumes."



Zero-code auto-instrumentation, on by default



1 Instrumented at deploy time, by the platform
Zero tracing code inside the agent repository.

2 requirements.txt
Unchanged since module 00.

3 Instrumentation is ON
On by default unless explicitly turned off.

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



A tree, not a log

TRACE FOR "JUNIOR VS PRESIDENTIAL, 3 NIGHTS"

```

invoke_agent LangGraph           ← the whole request
  LangGraph.workflow
    execute_task agent
      ChatOpenAI.chat           ← decide what to do
    execute_task tools
      execute_tool check_room_availability ← junior
    execute_task tools
      execute_tool check_room_availability ← presidential
    execute_task agent
      ChatOpenAI.chat           ← write the answer
  
```

Nobody wrote that shape down. The model decided, at runtime, to call `check_room_availability` twice and then call itself again.

The real tree has more nodes. LangGraph emits its own `call_model`, `RunnableSequence`, `Prompt` and `should_continue` steps. Useful when you are debugging the graph; ignore them for now.

SPANS COME IN KINDS

kind	carries
agent	The root - end-to-end latency, the whole request
llm	Model name, input and output tokens, finish reason, the messages
tool	Tool name, the arguments the model chose, what came back
chain	Framework plumbing between the above

"No traces found"?

Check the time range before anything else. The picker opens on a short window, and a trace from earlier falls outside it. Widen it and the rows appear.



The same request as module 00.

INFO: "POST /chat HTTP/1.1" 200 OK

latency: 2,344ms | status: success

Client initiate: /chat

Payload: "Book a room for 3 nights..."

Trace ID: 3c7483a3

Timestamp: 2026-09-17

TRACE: POST /chat 2,344 ms

└ ChatOpenAI.chat 1,068 ms

tokens: prompt=494, completion=42

└ execute_tool 82 ms

args: {"nights":3, "room_type":"junior"}

└ ChatOpenAI.chat 1,194 ms

tokens: prompt=725, completion=88

1

Where the time went

1,068 ms + 1,194 ms of a 2,344 ms request.
96% is the model.

3

Why it said that

{"nights":3,"room_type":"junior"}, chosen
by the model from a sentence containing
neither word

2

What it cost

494 in, then 725 in. Tool results appended
before model was asked again.



You are not billed per request

ATTRIBUTES ON AN LLM SPAN (OTEL GENAI SEMCONV)

Attribute	What it gives you
gen_ai.request.model	The model actually used
gen_ai.usage.input_tokens	Tokens sent on this call
gen_ai.usage.output_tokens	Tokens returned by this call
gen_ai.response.finish_reasons	Why it stopped - tool_call or stop

Same names whatever model or framework produced the span. That is the whole point of a semantic convention.

"The agent is slow"

...is nearly always "the model was called more times than you thought", not "the tools are slow". Model calls are network round-trips; the tools are local work in milliseconds. The trace is what tells you which.

THE TWO MODEL CALLS IN ONE REQUEST

```
ChatOpenAI.chat  call 1
prompt = 494
completion = 42
```

↓ tool results appended to the conversation

```
ChatOpenAI.chat  call 2
prompt = 725
completion = 88
```

The second call's input is always the larger one. Nothing about the guest's question changed - the tool results were appended before the model was asked again. You are billed per model call, and the agent decides at runtime how many it needs and how much context each one carries.



Finding the request worth looking at

Reading one trace is easy when you already know which one is interesting. In production you do not.

--condition	Finds
error_status	Requests that failed
high_latency	Slower than --max-latency (default 30000 ms)
high_token_usage	More tokens than --max-tokens (default 10000)
tool_call_fails	Tool invocations that did not succeed
excessive_steps	More spans than --max-spans (default 40)

excessive_steps has no equivalent in ordinary services

A loop that never converges is expensive rather than loud. Every individual step is fast, so no latency alert fires, and it returns 200 OK the whole time. A span-count threshold is how you find it.

```
amctl agent traces grand-meridian-concierge \  
  --project default --env default --since 24h \  
  --condition high_latency --max-latency 3000 --json
```

```
dd45e286 16 spans 2598ms  
5541b9f2 8 spans 3700ms
```

Command	Returns
trace <traceId>	Flat span list. No attributes.
trace --span <id>	One span in full, attributes included
traces export	Every trace, every span, attributes included

Attributes are where token counts and tool arguments live - so anything that reads them wants export.



The tool the agent could not use

ASK

"Can you recommend somewhere to eat near the hotel?"

I'm sorry, I wasn't able to retrieve dining recommendations at this moment. Would you like me to connect you with our team?

I'm unable to provide dining recommendations at the moment. May I connect you with our concierge desk?

I apologize for the inconvenience. It seems I can't retrieve dining recommendations right now.

The concierge cannot name a restaurant and hotel data has three of them..

EVERY CONVENTIONAL SIGNAL IS GREEN

200 OK every time

No exception no stack trace, nothing in the runtime logs

Well-formed replies polite, on-brand, and plausible

16 spans exactly what a healthy request is

Nothing about the outside of this request is unusual - which is the whole reason it is still in production.

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



Fixing the issue

CHANGE 1 · A BETTER ERROR MESSAGE

```
amctl agent deploy grand-meridian-concierge \  
  --project default --env TOOL_ERRORS=helpful --yes
```

Now the refusal names the categories that do exist. The model reads the error, works out the category it should have asked for, and retries. Usually - in one run of three it still gave up.

```
4735e3e0 24 spans ← recovered  
0da7337f 24 spans ← recovered  
09167c16 16 spans ← gave up anyway
```

Twenty-four spans where healthy is sixteen. Eight extra spans are a round trip the agent should never have needed, paid on every food question, forever, silently. `--condition excessive_steps` is built to find exactly this shape.

A tool description is an interface that a model reads and no compiler checks. Nothing in Python objects to a docstring that lies about its own data, and the tests still pass. Generate descriptions from the thing they describe - and use traces to find the one you already shipped.

CHANGE 2 · FIX THE DRIFT ITSELF

```
amctl agent deploy grand-meridian-concierge \  
  --project default --env TOOL_DOCS=generated --yes
```

The description is derived from RECOMMENDATIONS instead of typed out by hand, so it cannot drift from the data.

Input `{"category": "restaurants"}`

Output `3 recommendations`

16 spans. One call, the right call, three for three - and eight spans lighter than the version that recovered. Note what span counts do not tell you: the broken agent and the fixed one are both sixteen.

Both changes are configuration. Same image, same build name, active in about a minute.



How the traces actually get there

1 Pinned, not floating

The init container installs a pinned Traceloop SDK version, chosen by the AMP instrumentation version you select at create time. A deployed agent does not silently move to a new SDK when the platform default advances.

2 A fixed catalogue

Traceloop instruments the major LLM providers, the major agent frameworks, the major vector stores, and MCP tool calls. This lab is LangGraph and OpenAI, so all of it is covered.

3 Off the catalogue?

You emit spans yourself against the published contract: OpenTelemetry GenAI semantic conventions (gen_ai.*, db.*) plus a few traceloop.* keys for what OTel has not standardised yet.

Agent Traces SAMPLING

Every trace is exported by default, which is right for a lab and wrong for an agent serving real volume. Sampling is a head-based decision made inside the agent process, via the standard OpenTelemetry environment variables `OTEL_TRACES_SAMPLER` and `OTEL_TRACES_SAMPLER_ARG` - set like any other env var on the deployment.

Traces answer what happened. They do not answer whether it was any good. Step 7 needed you to go looking - the apology and the good answer came back through the same green spans, because nothing was scoring either of them.



You can see what happened. You still can't say whether it was good.

one `200 OK` per request → a span tree per request.

What a trace answers.

Where the time went

per-span durations, model calls vs tools

What it cost

input and output tokens, per model call

Why it said that

the arguments the model chose, and what came back

And where you can ask it.

- **The console**

when you want to look at one request

- **The CLI**

when you want to implement an automation process

- **The MCP server and the Agent Skill**

when you would rather interact with the natural language

ADLC LIFECYCLE

01 BUILD

02 DEPLOY

03 TEST

04 PROMOTE

05 MONITOR

GOVERN - access, guardrails, identity, spend



03 MODULE

Evaluation

Measuring something that answers differently every time.

15 min

Your pytest suite is green. Your agent just hallucinated a room rate.

Unit tests verify the code path; not the model output

Non-determinism breaks assertion-based testing

Quality regresses silently when the model updates

There is no "expected output" for free-form text - only "good enough"

```
test_booking_logic.py .... [100%]  
===== 8 passed in 0.05s  
=====
```

Status: Exit Code 0 (Success)

HALLUCINATION DETECTED

Asserted: \$299 (Fixed Rate)

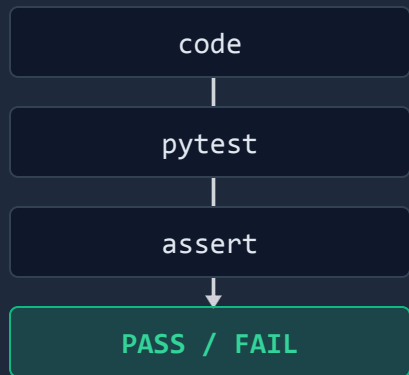
Actual: "Free breakfast + \$450/night"

Same code, different lens.

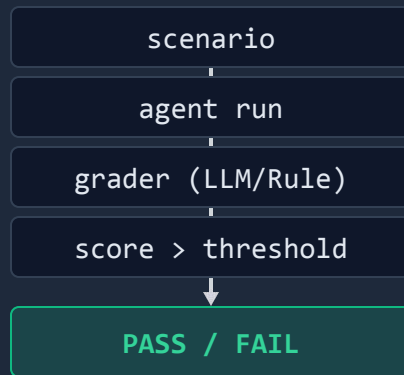


Tests check code paths. Evals check behavior.

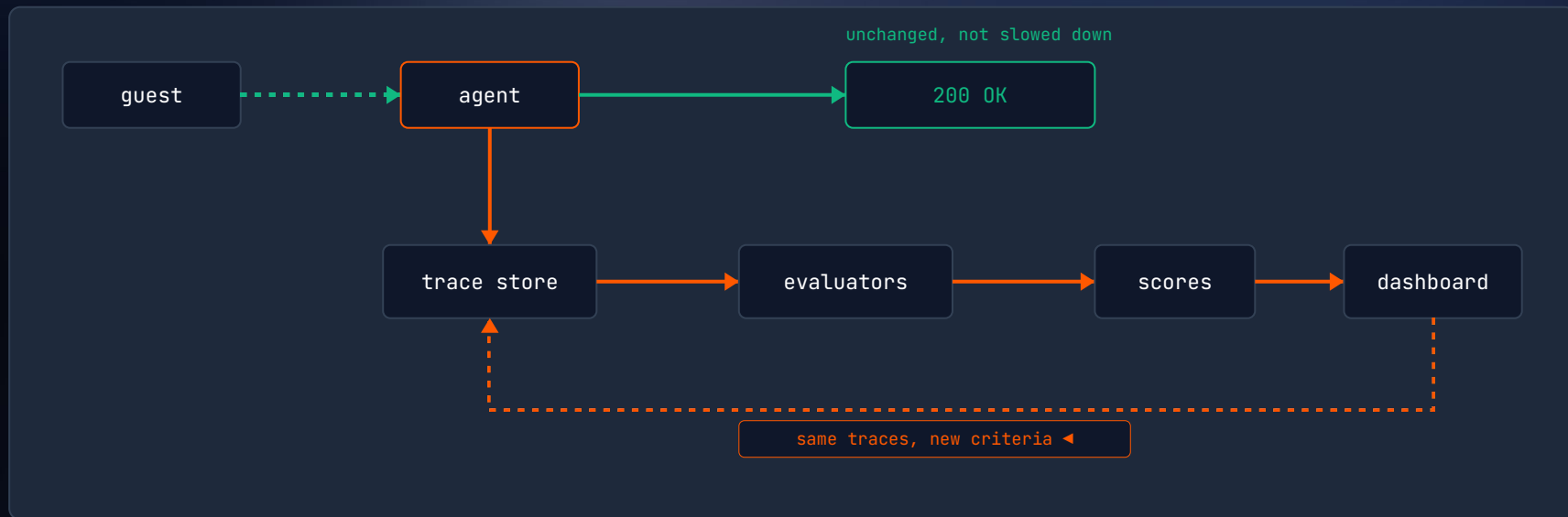
TESTS (Determinism)



EVALS (Heuristics)



Score the recording, not the performance



Reads stored traces - never the request path

Zero Latency Impact - so it cannot slow the agent down

Retrospective Scoring - score traffic from before question existed



A monitor is a configured evaluation job

a set of evaluators

+

an agent

+

an environment

+

a window of traces

CONSOLE → THE AGENT → EVALUATION → ADD MONITOR

Past Traces

A window you set, over traffic that already exists. Starts evaluating immediately. Take this one now, and set the window to cover the traffic module 02 generated.

The window does not move, so two runs over the same traces are directly comparable. That is the iteration loop.

Future Traces

A recurring schedule, minimum five minutes, for ongoing production monitoring.

Continuously run evaluators on new traces until stopped.

The judges need a model

Add an LLM-as-judge evaluator and an LLM Providers section appears in the configuration panel. Pick a provider, paste an API key, click Add. You do this once per monitor - every judge in it shares the credentials.

Give it a few minutes. The rule-based evaluators finish almost instantly; the judges make a model call per evaluation, and Tone makes one per model call in every trace. It is a job, not a page load.



An Evaluation Monitor For The Concierge Agent

RULE-BASED - DETERMINISTIC, INSTANT, FREE

Evaluator	Level	Set
Length Compliance	trace	<code>min_length 20 · max_length 2000</code>
Latency Performance	trace	<code>max_latency_ms 5000</code>
Content Safety	trace	<code>prohibited_strings: guarantee, refund, free upgrade</code>

PROHIBITED_PATTERNS - RAW REGEX IN THE CONSOLE

```
\b\d{13,16}\b      a bare 13-16 digit run
\b[\w.+-]+@[ \w-]+\.[\w.]+\b  an email address
```

Neither fires on what a concierge answer legitimately says - 450 USD, GM-2026-0918 and confirmation 12345 all pass. `create-monitor.sh` carries the same two JSON-escaped.

Why Completeness

checks whether the answer addressed everything the guest asked across multiple LLM calls.

Why Tone, at LLM level

It scores every model call individually rather than the final answer - which is how you catch something off-brand in an intermediate step that the final answer smoothed over.

LLM-AS-JUDGE - A MODEL READS THE TRACE AND SCORES IT

Completeness	trace	<code>model gpt-4o-mini · temperature 0.0</code>
Tone	llm	<code>model gpt-4o-mini · temperature 0.0 · context "luxury hotel concierge"</code>



Rules for what you can count. Judges for what you'd read.

	Rule-based	LLM-as-judge
How it decides	Python, deterministic	A model reads the trace and scores it
Speed	Instant	Seconds
Cost	Free	A model call per evaluation
Same trace twice	Identical score	Near-identical
Right for	latency, length, token budget, required tools, prohibited content	accuracy, helpfulness, groundedness, tone, reasoning

THREE LEVELS, AND YOU CONFIGURE NO ITERATION LOGIC

Level	Sees	Runs
Trace	the whole request, input to output	once per trace
Agent	one agent's steps and decisions	once per agent execution
LLM	a single model call and its messages	once per model call

Tone ran twice per trace

Nobody told it to. It is an LLM-level evaluator, so it runs once per model call - and most requests make multiple LLM calls. That is why its evaluation count in step 2 was roughly double the trace count.

A real quality bar holds both

Rules cannot read meaning; judges cost money and time. The monitor you just built has three of one and two of the other, which is the ordinary shape of a production set.

Agent-level and LLM-level evaluators also unlock two extra dashboard tables - score by agent, and score by model - which is how you compare models on the same traffic.



There is less magic here than “evaluator” suggests

A RULE → A SOURCE CODE PANEL

Content Safety, about forty lines

```
for pattern in self.prohibited_patterns or []:  
    flags = 0 if self.case_sensitive else re.IGNORECASE  
    if re.search(pattern, output, flags):  
        found_patterns.append(pattern)
```

That is step 1's warning, written into the evaluator rather than inferred by the platform. A few lines down, your patterns are handed to `re.search(pattern, output, flags)` with `re.IGNORECASE` unless you set `case_sensitive` - so they are ordinary Python regexes, matched against `trace.output`. This evaluator scores what the agent said, not what the guest asked.

A JUDGE → A PROMPT TEMPLATE PANEL

A judge's implementation is its prompt

All of it is there: the criterion, the evaluation steps, the template variables that decide what the judge sees - `{llm_span.format_messages()}` - and the rubric it scores against.

```
0.0  = Clearly inappropriate tone  
...  
0.75 = Good tone, professional, well-suited  
1.0  = Excellent; warm and clearly helpful
```

A 0.75 from Tone is not a model's vague opinion. It is a rung on a rubric you can read, against a prompt you can audit.



The check no built-in can do for you

`room_rate_accuracy.py` - a trace-level code evaluator. It pulls every rate the answer quotes and checks it against the published rate card, allowing for nightly multiples of a stay length the guest actually asked about.

THE EDITOR GENERATES THE METHOD SIGNATURE

```
from amp_evaluation import EvalResult, Param
from amp_evaluation.trace.models import Trace

def my_evaluator(
    trace: Trace,
    valid_amounts: list = Param(default=[], ...),
    max_nights: int = Param(default=30, ...),
) → EvalResult:
    # ← everything below here is yours
```

`EvalResult.skip()` is not a zero. Skips are tracked separately and excluded from the average, so an evaluator that honestly declines to judge does not drag the score down.

AND THE CONFIG YOU SET PER MONITOR

Parameter	Value
<code>valid_amounts</code>	280, 340, 380, 420, 1200
<code>max_nights</code>	30

The five nightly rates in `agent/hotel_data.py`, under `ROOMS` as `price_per_night_usd`. Nothing else belongs in that list - the \$36 risotto on the room service menu is a real price and a different evaluator's problem.

Configuration, not a literal. In production you would populate it from wherever rates actually live, so the check moves when the prices do. A second hand-maintained copy drifts exactly the way module 02's tool description did.



Where rules stop

THE ANSWER YOUR RULE WILL LET THROUGH

"\$420 per night, or \$760 for two nights."

Wrong for a \$420 room - two nights is \$840. But \$760 is exactly two nights of the \$380 junior suite, so a rule holding a rate card and a night count cannot fault it. Both figures are rates, both are on the card, and the total is a legitimate multiple of one of them.

No amount of regex fixes that. It needs something that can read which room was being discussed.

BUILT-INS AIMED AT EXACTLY THIS

Groundedness	Claims supported by the tool results in the same trace. Takes on_missing_context: skip or zero.
Accuracy	Factual correctness, more broadly
Safety	Joins Tone at the LLM level

WHAT NO BUILT-IN CAN KNOW

concierge_voice.md

A prompt template scoring the five service standards written into agent/system_prompt.py:

- grounded in tool data
- answer first
- in character
- no leaked plumbing
- honest about limits

None of the five is expressible as a rule, and all five are things a hotel would actually fire someone over.

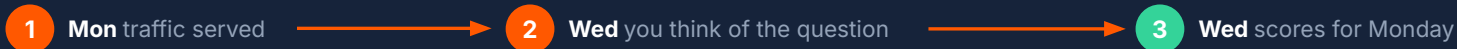
Adding it

Choose LLM-Judge as the type and paste the prompt. Config Params arrives with the four parameters every judge takes - model (required, bare), temperature, max_tokens, max_retries. The provider is the monitor's, not the evaluator's.

A judge prompt is a template, so add a property_name param and one evaluator serves every property in the group.



Score last week with this week's standards



PAST-TRACES (RETROSPECTIVE)

Create a second Past-Traces monitor over the same window, with the evaluators you just wrote. The agent is untouched: no rebuild, no redeploy, no new traffic. Within a couple of minutes you have quality scores, against standards you invented after the fact, for requests that were served before you thought of them.

That is not a thing you can do with tests. A test suite tells you about the code you have now; it has nothing to say about yesterday.

FUTURE-TRACES (CONTINUOUS)

Then make it ongoing

A Future Traces monitor with the same evaluator set and an interval. It starts within a minute and runs on schedule - and the time series in step 2 becomes a regression alarm rather than a snapshot.



You cannot assert it. You can measure it.

What you can now answer

Is it good?

a score per trace, with an explanation

Is it getting worse?

a trend per evaluator, per environment

Which request went wrong?

the score column, on the traces table

Was last week fine?

new criteria, old traffic, no redeploy

GAP LEDGER TRANSITION

1. Ad-hoc manual inspection
2. Static prompt assertions
3. "tests that pass regardless"

→ "a number that moves when quality does."

And where the standards come from

The platform built in evaluators knows what agents are commonly measured on:

latency, cost, safety, groundedness, tone

You know what *your* agent should be measured on:

Room Rate Accuracy, Concierge Voice

CUSTOM EVALUATORS

Custom evaluators are how the second list gets enforced as automatically as the first.

They bridge generic platform metrics with your agent's unique business logic.



04 MODULE

External Agents

Agents the platform does not run, governed all the same.

12 min

The same concierge, built by somebody else

WHAT THIS SESSION BUILT

Registered from a repository and built by the platform

Deployed, with a gateway in front of it

Instrumented at deploy time, by the platform

WHAT WE ARE ABOUT TO RUN

Never built here, and never deployed here

Started by hand, on the laptop on this stage

Instrumented at launch, by one command

One is a single tool-calling loop. The other is two members in sequence: one looks things up and writes a brief, the other turns the brief into the reply the guest reads. The second has no tools, so it has no way to invent a price.



Traces travel outbound

Your machine

the agent, on a laptop or in your own data centre

one outbound connection

spans are pushed, never pulled

Agent Manager

traces, then evaluation over them

no inbound connection

no tunnel

no public hostname

no firewall rule

AND THE ENTIRE INTEGRATION

```
python main.py
```

```
amp-instrument python main.py
```

No import, no decorator, no initialisation call, no SDK in the code.
The agent has no idea any of this is happening.



One request, and who spent the time

workflow	2942ms
task compare the two suites	1821ms
Concierge.agent	1820ms
model call	996ms
tool check availability	0ms
tool check availability	0ms
model call	810ms
task write the guest's reply	1100ms
Writer.agent	1099ms
model call	1091ms

THE SPLIT INSTRUMENTED

Two thirds of the request establishing what is true, one third turning it into prose. It falls out of the shape of the agent, and it is the first question to ask of anything that feels slow.

TOOLS

Some spans arrive from auto-instrumentation. Four were written by hand against the published contract, so a tool that refuses now raises its hand instead of returning green.

Three model calls account for 2.9 of the 2.94 seconds. The tools are dictionary lookups in the same process and finish inside a millisecond. When an agent feels slow it is almost always the number of model calls, and the trace is the only thing that tells you which.



One definition, two agents, one dashboard



Evaluation reads stored traces, so it neither knows nor cares where the agent runs. Same prompts, same evaluators, same scale, one chart per agent. The external agent is being held to the written house style by the same judge as the one the platform operates.



What the platform runs, and what follows the agent

A SERVICE THE PLATFORM OFFERS

- Build from source
- Deploy, promote, roll back, suspend
- Restart it when it dies
- Inbound gateway, API keys, rate limits
- Runtime logs and metrics

Outside its own runtime, every one of these is yours to provide.

GOVERNANCE THAT FOLLOWS THE AGENT

- Traces, spans, token accounting
- Evaluation, monitors, custom evaluators
- A governed path to the model, with guardrails
- The provider credential held centrally, not by the agent
- The same trace commands, either way

These travel with the agent, wherever it happens to be running.

An agent built and deployed outside this platform still gets all of the right hand column. Build and deploy are a service. Observability, evaluation and control of the model call are governance, and governance follows the agent rather than the hosting.



Where the agent got to this session

01

ON THE PLATFORM

Built from source into an image pinned to a commit, deployed behind a gateway with authentication on by default, and reconfigured twice without a rebuild.



02

VISIBLE

A span tree for every request: the model calls, the arguments the model chose for itself, and the tokens each call carried. No tracing code in the agent.



03

MEASURABLE

A score per trace with an explanation beside it, from rules and from judges, including traffic that was served before the standard existed.



04

AND ONE WE DID NOT BUILD

An agent running somewhere else entirely, pushing its traces outbound, scored by the same evaluators on the same dashboard.

None of it changed the agent's code, and none of it needed the platform to be the thing running the agent. Build and deploy are a service you can take or leave.

What we have not done is decide what the agent is allowed to do. That is the session after the break.





SEPT 22 - 24, 2026 | NAIROBI, KENYA

Thank You!



Asanka Abeyweera

Head of Product Management &
Architecture, Agent Platform BU



Vignnah Selvaraj

Lead Sales Engineer