WSO2CON NORTH AMERICA

May 20 - 22, 2026 | Austin, Texas, USA

Evolving Your Enterprise Architecture for the Agentic Era

Chintana Wilamuna

Vice President, Sales Engineering

Asanka Abeyweera

Associate Director & Architect

Agenda

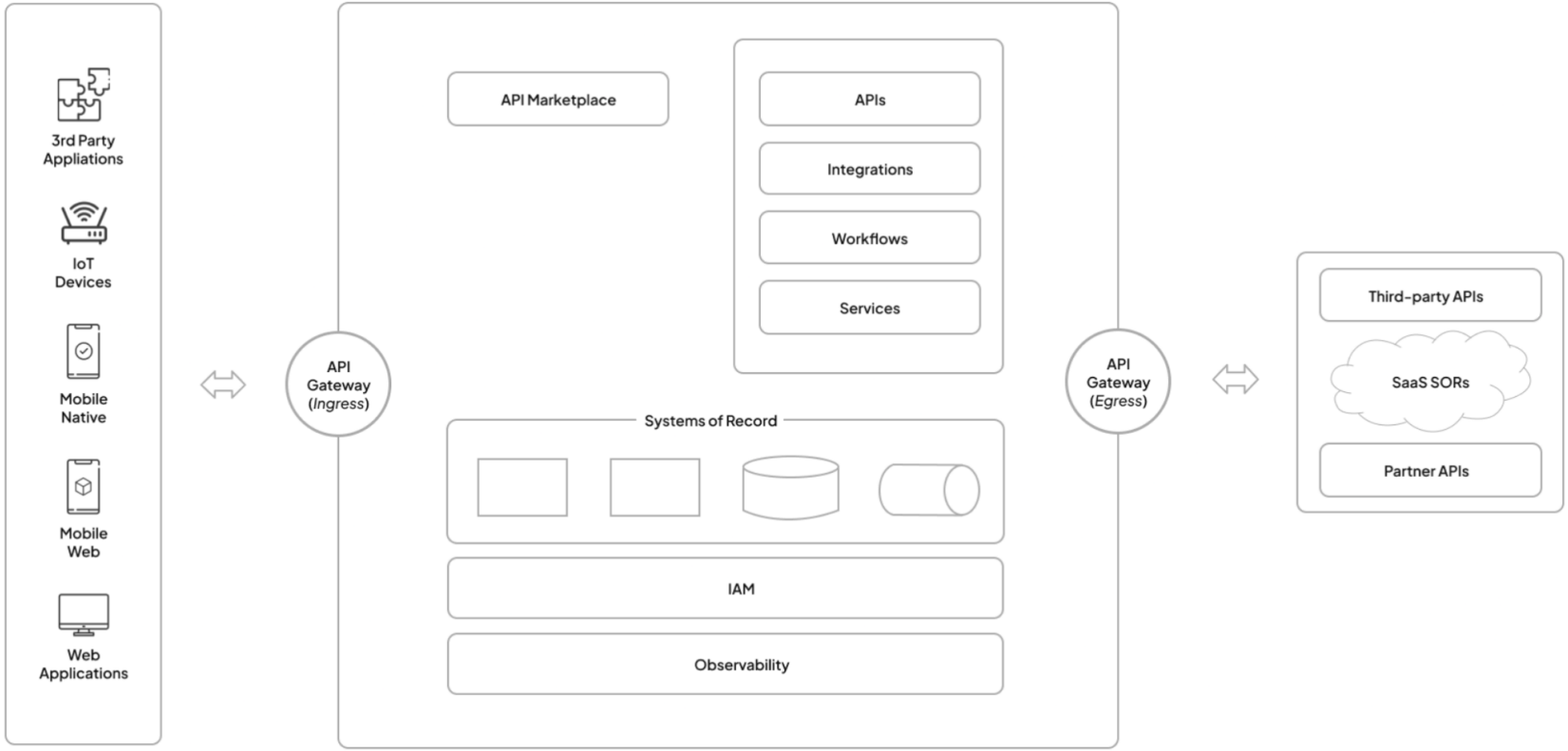
- The shift from app centric to agentic
- Where conventional architectures fall short
- Five things to rethink - identity, traffic, observability, QA, governance
- A minimum viable agentic architecture
- Patterns for incremental adoption
- Live demo
- Q&A

What is an agent?

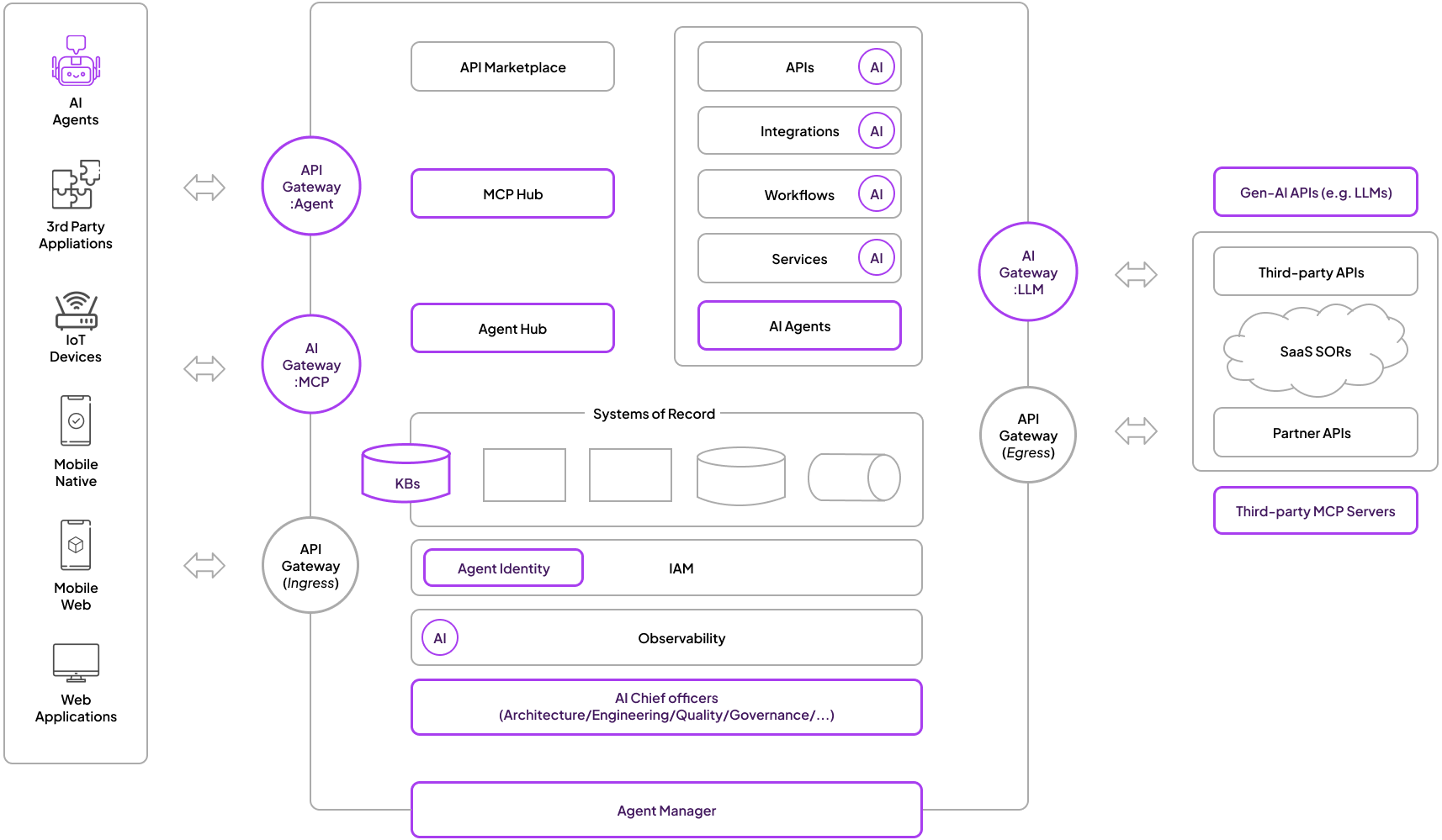
An agent is an **LLM** running in a **loop**, with **tools**, working toward a goal it can **decide** when it has met.

- Loop - the model gets to act multiple times
- Tools - the model can interact with the external world
- Decide - the model controls flow, not the developer (non-deterministic)

Non agentic enterprise architecture



Ref: https://wso2.com/library/blogs/agentic-enterprise-with-wso2/


AI Agents
3rd Party Appliations
IoT Devices
Mobile Native
Mobile Web
Web Applications
API Gateway :Agent
AI Gateway :MCP
API Gateway (Ingress)
API Marketplace
MCP Hub
Agent Hub
APIs
AI
Integrations
AI
Workflows
AI
Services
AI
AI Agents
Systems of Record
KBs
Agent Identity
IAM
AI
Observability
AI Chief officers
(Architecture/Engineering/Quality/Governance/...)
Agent Manager
AI Gateway :LLM
API Gateway (Egress)
Gen-AI APIs (e.g. LLMs)
Third-party APIs
SaaS SORs
Partner APIs
Third-party MCP Servers

The shift

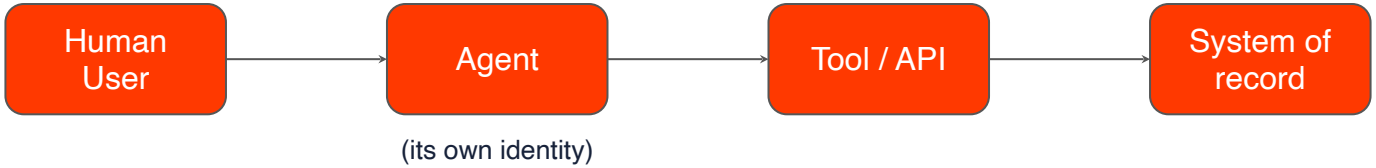
System of record -> System of record + systems of context

- Context graphs, knowledge bases, embeddings
- Tools and policies become runtime deps, not just config
- Models are probabilistic and operate on unstructured data
- Deterministic and probabilistic workloads now live in the same call path

Where the current architecture breaks

Layer	Built for	Breaks because
API Gateway	Users with API keys	Agents act on behalf of users, with budgets and content controls
Identity	Human or service account	Need delegation: human -> agent -> tool
Workflows	Static graphs	Agents build the graph at runtime
Observability	Deterministic traces	Reasoning chain, tool decisions, token cost
QA	Unit tests, fixed outputs	Outputs/behaviors are non-deterministic
Governance	“Can this be called?”	“Should this be called *now*, by *this* agent for *this* user?”

Rethinking identity: the caller is an agent



We must answer per call:

- Who is the human principal?
- Which agent is acting?
- With what scoped permissions, for what task, for how long?
- Can the downstream tool see and enforce all three?

New primitives: agent identity, scoped delegation tokens, on-behalf-of (OBO) flows, policy-bound-credentials

Rethinking traffic: API Gateway -> AI Gateway

API Gateways gives us: routing, schema validation, auth, rate limits, caching …

An AI Gateway adds, sitting alongside it:

- Model routing & fallback (cost / latency / quality)
- Token-aware rate limits and quotas
- Prompt and response inspection and redaction
- Guardrails and content policy at the edge
- Egress control - which models, which providers, which regions

Same shape as the API gateway but different surface

Rethinking integrations: tools and the MCP hub

Agents need to discover and call tools safely

MCP (Model Context Protocol) and similar standards gives us:

- A common interface to expose existing APIs as tools
- Tool discovery, scopes and versioning
- A central place to govern which agents can call which tools

Treat the tool hub like an API portal - but the consumers are agents, not humans

Without this, every team rebuilds tool wiring, every team writes its own policy and nothing is auditable

Observability: beyond logs and metrics

A traditional trace request -> service A -> service B -> response

An agent trace: prompt -> model -> reasoning -> tool A -> reasoning -> tool B -> answer

We need to capture:

- Prompts, model versions, parameters, system messages
- Every tool call - arguments, results, errors
- Token usage and cost per step
- Latency at each hop
- The decision - which path was taken and why

Tracing a decision graph, not a call chain

Rethinking QA: tests aren’t enough

Deterministic software: same input -> same output -> unit test passes or fails

Gen AI: same input -> different output -> no single correct answer

We need evaluations, not just tests:

- Golden datasets with expected behaviors, not exact strings
- LLM-as-judge for qualitative properties
- Regression suites for prompts, models, and agent graphs
- Production replay and A/B comparison

Eval driven development becomes the primary quality gate

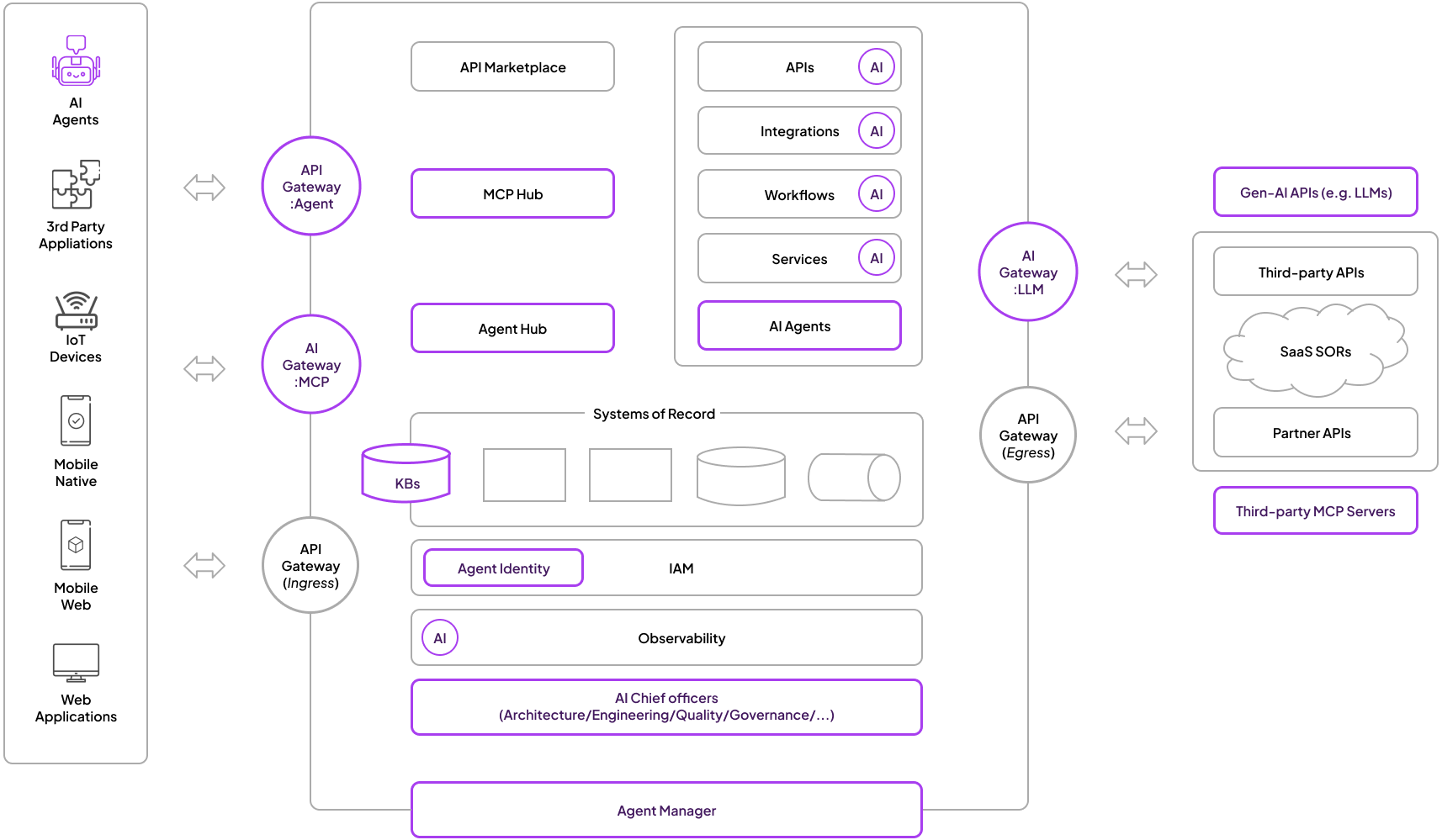
Governance as architecture

Governance is not a policy document, it’s a control surface in the platform

Pushed into the platform layer (not duplicated in each app):

- Guardrails - input/output filters, PII redaction, jailbreak detection
- Cost controls - per-agent, per-tenant, per-task budgets
- Policy enforcement - which models, which tools, which data
- Audit - every action, attributable to an agent and a human
- Kill switches and circuit breakers

If it’s not in the platform, every team will reinvent it inconsistently


AI Agents
3rd Party Appliations
IoT Devices
Mobile Native
Mobile Web
Web Applications
API Gateway :Agent
AI Gateway :MCP
API Gateway (Ingress)
API Marketplace
MCP Hub
Agent Hub
APIs
AI
Integrations
AI
Workflows
AI
Services
AI
AI Agents
Systems of Record
KBs
Agent Identity
IAM
AI
Observability
AI Chief officers
(Architecture/Engineering/Quality/Governance/...)
Agent Manager
AI Gateway :LLM
API Gateway (Egress)
Gen-AI APIs (e.g. LLMs)
Third-party APIs
SaaS SORs
Partner APIs
Third-party MCP Servers

Incremental adoption: don’t rearchitect on day 1

Stage 1 - Wrap, don’t replace

- Use an AI gateway (tools & LLM calls) - Governance controls
- Introduce agent identity
- Observability

Stage 2 - Centralize controls

- Move guardrails and cost controls out of apps to the platform
- Create an evaluation pipeline alongside existing CI

Stage 3 - Make agents first class

- Agent registry, lifecycle and governance
- Multi-agent workflows with policy-bound governance

Go-live checklist

- Treat agents as first-class principals, not API consumers
- Put the AI Gateway next to the API Gateway, not instead of it
- Standardize tool interface (MCP) so agents don’t get custom wiring code
- Trace reasoning, not just requests
- Build an evaluation pipeline before scaling agents
- Put guardrails, cost controls, and policy in the platform - not the app
- Pick a deployment posture that lets you run in hybrid infra

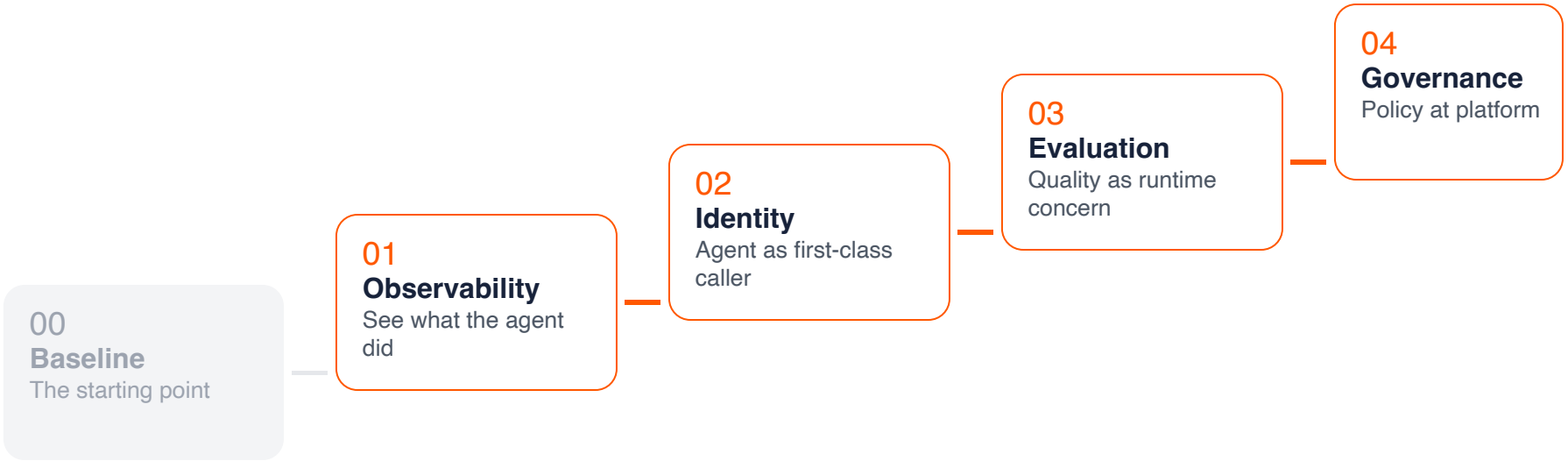
Demo

WSO2

Agents don't plug in.
They challenge your architecture.

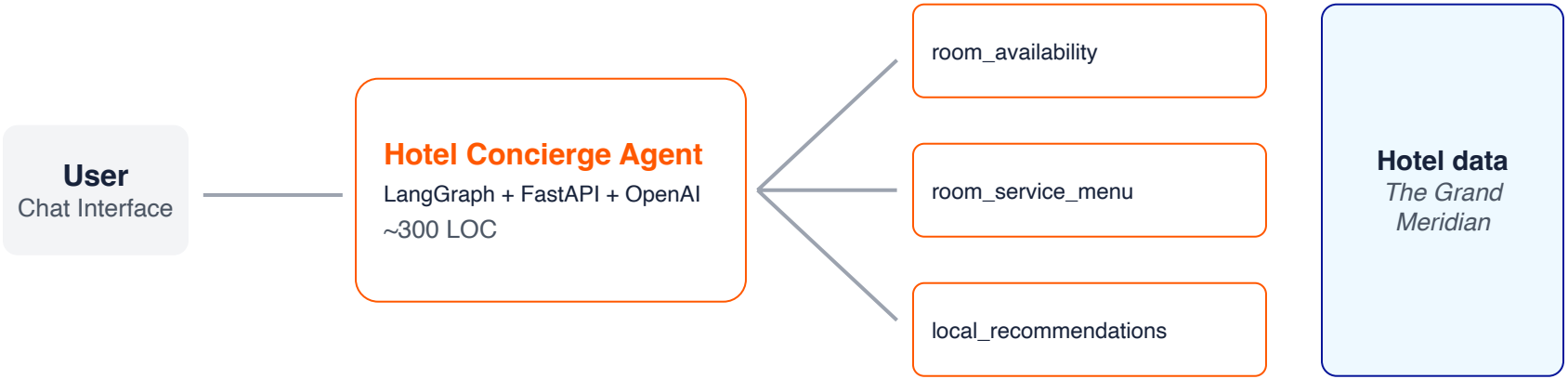
- Request-response APIs behind a gateway
- Static workflows
- Identity bound to a human user
- Deterministic test suites as the quality bar
- Governance retrofitted at the application layer

Tutorial Plan



"We solve one concern at a time. Same agent. Platform evolve around it."

The Hotel Concierge Agent

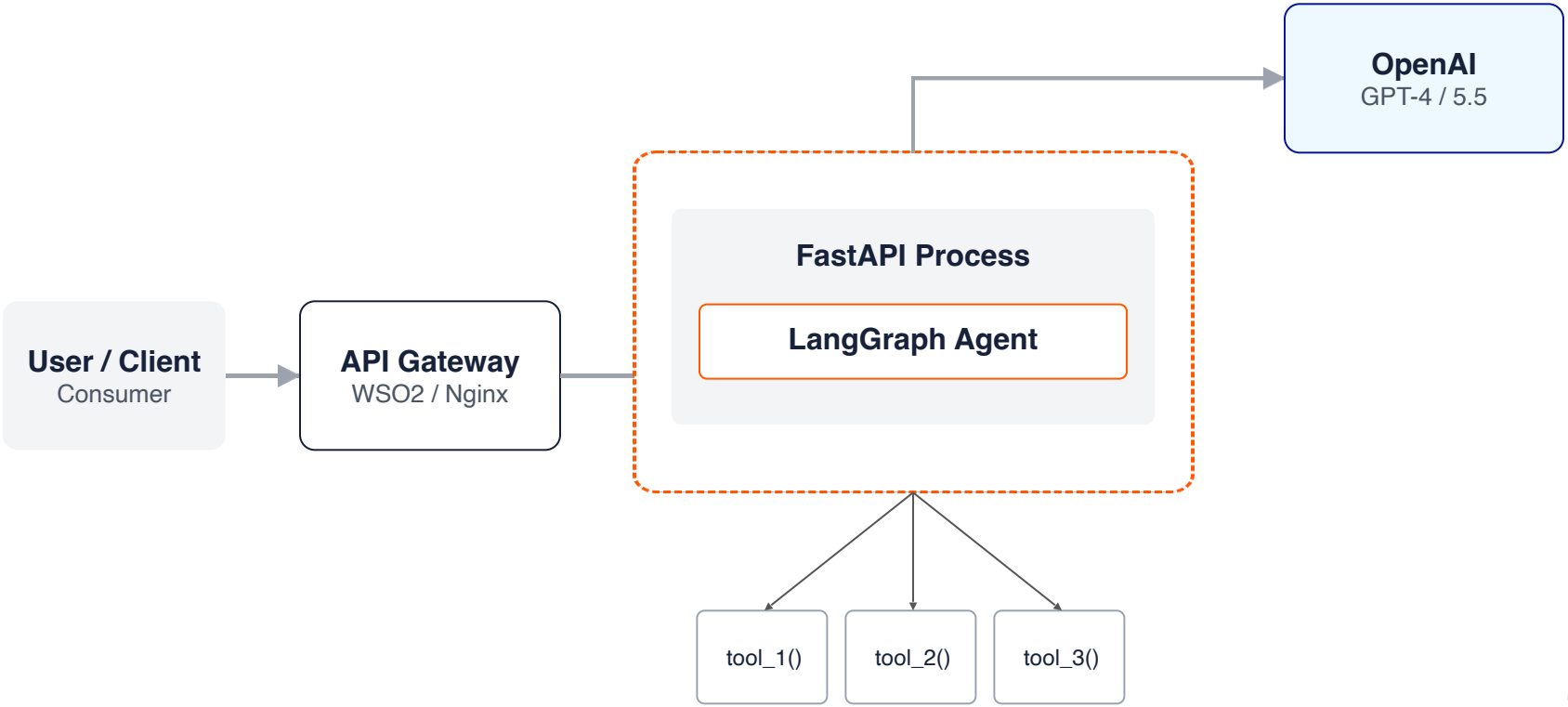


00 - Baseline

A traditional agent deployment

WSO2

An API service with an agent inside



It works. Until you ask the hard questions

1. What did the agent actually do on that call?
2. Who called the booking tool - the user, or the agent?
3. Is the model getting worse?
4. How do we enforce "no PII in prompts" across every agent?
5. When an agent misbehaves, what's the blast radius?

01 - Observability

Logs are not enough

Logs say what happened. Not why.

- **Reasoning steps** aren't captured
- **Tool calls** and arguments aren't structured
- Latency, tokens, and outcomes **not correlated**
- Retries and fallbacks have **no causal chain**

OpenTelemetry GenAI semantic conventions

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

Observability for agents is push-based and standardized.

- **The trace** (not the log line) is the unit of debugging
- **Standardized instrumentation:** emit OTEL GenAI, any compliant backend consumes it
- **Unified substrate:** cost, audit, and debugging all in one place

02 - Identity

The caller is an agent, not a user.

Which credential does the agent use?

"Book the deluxe suite for the first weekend in June."

close

A. Shared agent credential

Every agent everywhere authenticates as the same principal.

- Audit: "Agent service did it"
- No per-user/conversation attribution
- Rotation requires full redeployment

close

B. Forward the user's token

Hides the agent as the actual decision-maker.

- Audit: "Sarah did it"
- Scope is either too wide or too narrow
- Lacks agent-specific policy enforcement

check_circle

C. Agent identity

Narrowly scoped with a verifiable record of the acting user.

- Audit captures both identities ✓
- Delegated, accountable authority
- First-class security principal

Traditional IdP authenticates users and registers services. It does not natively model C.

OAuth 2.0 + RFC 8693 token exchange

"Two identities. One audit trail. Standard protocol."

Identity for agents is per-agent, scoped, and delegated.

- The agent is a first-class registered **principal** at the IdP
- Tools authorize via OAuth scopes: **least privilege** per agent
- Every user-initiated call is a signed **delegation**

03 - Evaluation

Deterministic tests aren't enough

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"

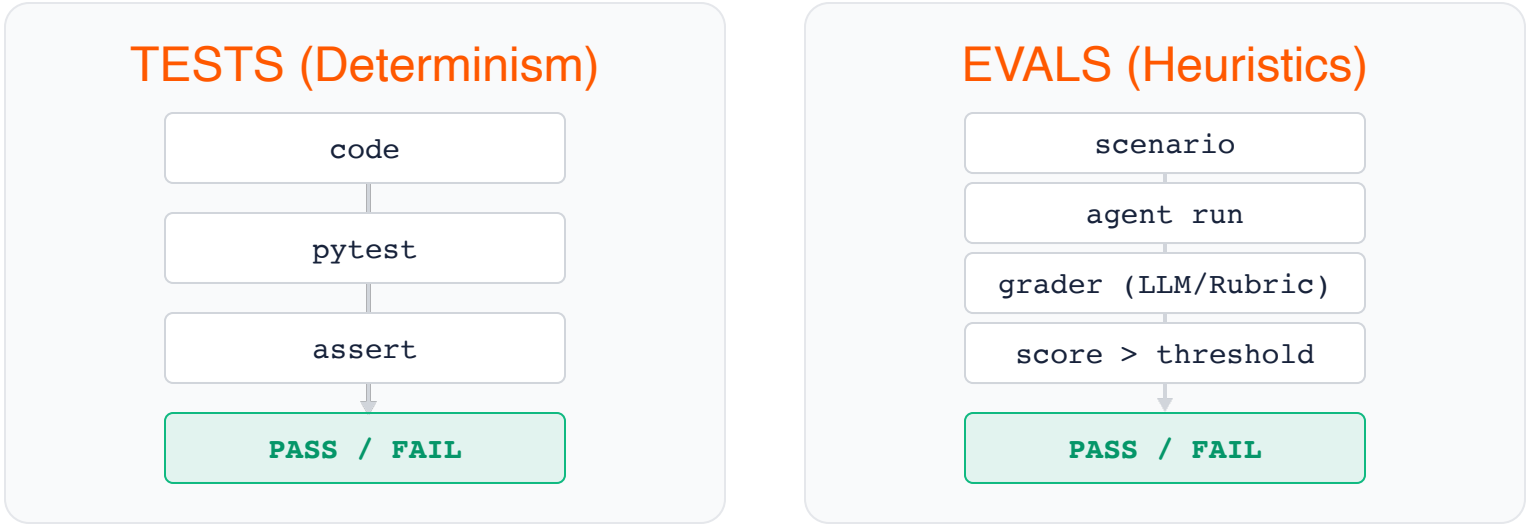
```
test_booking_logic.py .... [100%]
================ 4 passed in 0.05s
================

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.



Both run in CI. Both gate the deploy.

Break it. Catch it. Fix it.

This is how regressions actually happen, and how you actually catch them.

01. Break.

Model auto-upgrades or prompt tuning degrades behavior-but code stays the same, so tests stay green.

02. Catch.

The eval suite runs on schedules. Scores drop below threshold, pointing to exact scenario regressions.

03. Fix.

Tune, pin, or adjust retrieval. Re-run and verify the score recovers. Commit the new baseline.

"Not 'more tests.' A feedback loop you wire into your deploy pipeline and your monitoring."

QA for agents is continuous, rubric-based

- **Rules** catch operational regressions. **Judges** catch semantic ones.
- The eval is the **spec**: write it before the agent
- **Scores** attach to spans and traces

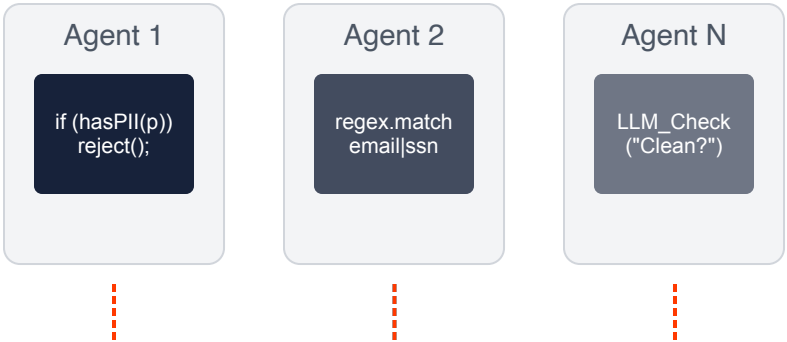
04 - Governance

Policy at the platform, not in agent code.

WSO2

Every agent enforces "no PII in prompts."

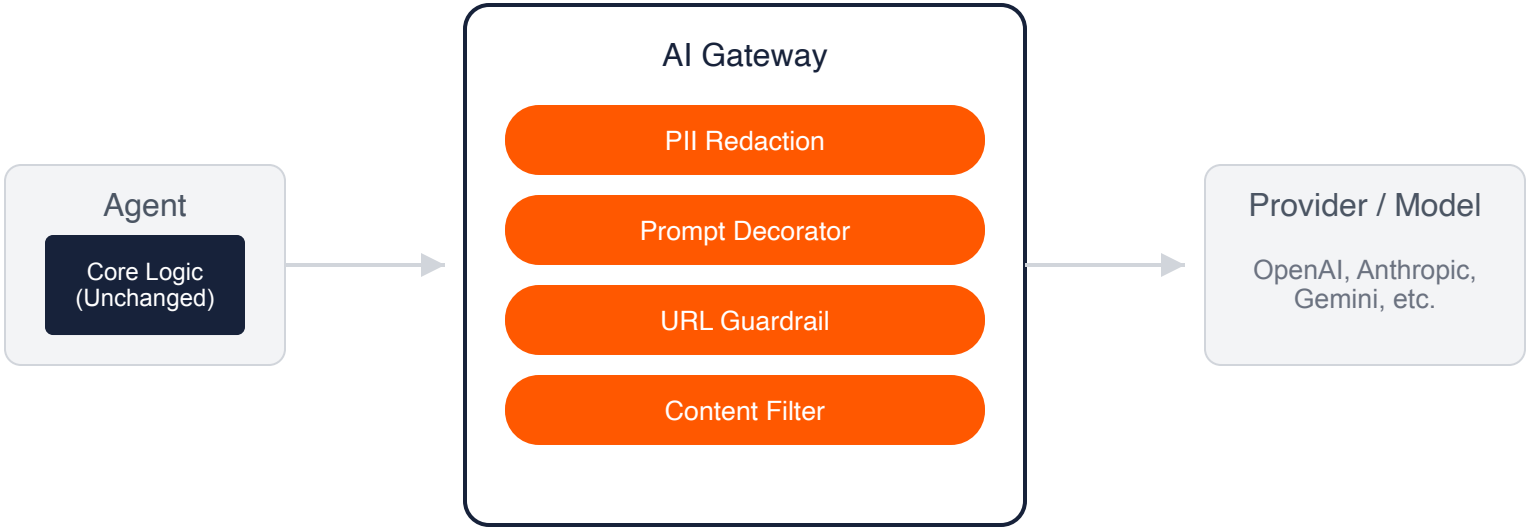
- Policy in code = N implementations, N drift surfaces
- Updating fleet-wide policy = touching every codebase
- Third-party agents you don't own can't be policed
- Auditors want one policy register - not N codebases



Same intent. N implementations. N bugs.

Move the policy to the gateway. Leave the agent alone.

"Same agent code. Policy applied at the platform."



Policy is configuration. Not code.

Governance is platform infrastructure, not application code.

- Policy is **configuration**. Compliance edits a declarative artifact.
- The **AI gateway** is the enforcement point. Nothing else remembers the policy
- **Separation of duties** matches reality: compliance / ops / engineering each own their layer

You don't need to rewrite. Adopt one layer at a time.

01

Instrument with OTEL GenAI

week

02

Use Agent identity with token exchange

sprint

03

Add an eval suite

sprint

04

Move policy to the gateway

quarter

"Pick the layer where today's pain is loudest. Start there."

WSO2CON NORTH AMERICA

May 20 - 22, 2026 | Austin, Texas, USA

Thank You!

Chintana Wilamuna

Vice President, Sales Engineering

Asanka Abeyweera

Associate Director & Architect