QORTARA CLOUD GOVERNANCE

Policy Enforcement

Policy Enforcement

Policy enforcement is the runtime decision point that sits between an agent and a sensitive action. Before the agent reads a record, calls a tool, or writes to a system, Qortara evaluates the action against your active policies and returns a decision. If the decision is deny, the action does not run.

This is different from an observability tool that logs what happened after the fact. A denied action has no side effects: it never reached the resource, never called the API, never sent the message.

> **Availability.** Dispatch-path enforcement in the open-source Qortara Governance sidecar is shipped and self-serve today (`pip install qortara-governance-langchain`, Apache-2.0). The hosted Cedar policy decision point (PDP) in Qortara Cloud Governance is in pre-launch. Both share the same allow/deny/verify decision model described here; the difference is where evaluation runs and what additional context is available. Endpoints and figures that reference the hosted control plane describe the pre-launch design, not a running service.

Two enforcement surfaces

Surface	Where it runs	Status
Sidecar dispatch-path enforcement	In your own process, on the agent's tool-call path	Shipped, self-serve
Cloud policy decision point (Cedar PDP)	Hosted control plane, called over the network	Pre-launch (designed to)

The sidecar intercepts tool calls as the agent dispatches them and applies policy locally. This is the path that is available today. The hosted PDP is designed to centralize policy across tenants and fleets and to add cross-organization context (such as trust attestations) that a single local process cannot see on its own.

The decision model

Every evaluation resolves to one of three decisions.

Decision	Meaning

`allow`	The action may proceed, and a governance record is written.
`deny`	The action is blocked, and the reason is recorded.
`verify`	The action needs additional approval or evidence before it can proceed (a human or workflow handoff).

Deny takes precedence. If any matching policy denies the action and its conditions are met, the result is deny, even if other policies would allow it.

What "governance at the execution layer" means

Your agent code asks for a decision before performing a governed action. With the hosted PDP, that is a call to the policy evaluation endpoint; with the sidecar, it happens inline on the dispatch path. Either way the action only runs after an allow.

```text
Your Agent                    Qortara                    Target Resource
    |                            |                            |
    |--- evaluate(action) ------>|                            |
    |    {agent, action,         |                            |
    |     resource}              |                            |
    |                            |--- match active policies   |
    |                            |--- record governance event |
    |<-- decision: allow/deny ---|                            |
    |                            |                            |
    |  (if allowed)              |                            |
    |------------------------------------------ API call ----->|
    |                            |                            |
    |  (if denied)               |                            |
    |  [action never happens]    |                            |
```

The request and response cycle

The hosted PDP takes an evaluation request describing who is acting, what they want to do, and against which resource.

Request:

```json
{
  "context": {
    "agent_did": "did:qor:agent:abc123",
    "action": "read",
    "resource": "customers/12345/pii",
    "tool": "crm-api",
    "timestamp": "2026-04-10T12:00:00Z",
    "metadata": {
```

```
      "session_id": "sess_789",
      "user_request": "Look up customer profile"
    }
  }
}
```

Response:

```json
{
  "decision": "deny",
  "reason": "Policy 'restrict-pii-access' denies read on customers/*/pii without capability 'pii-approved'",
  "policy_id": "pol_789ghi",
  "policy_name": "restrict-pii-access",
  "audit_event_id": "evt_abc123",
  "framework_context": ["soc2", "gdpr"]
}
```

The `reason` is human-readable so an engineer (or an auditor) can see why an action was blocked without reading policy source. The `framework_context` field notes which compliance frameworks the decision is relevant to, which feeds compliance evidence.

How evaluation proceeds conceptually

1. **Context parsing.** The agent identity, action, resource, and any metadata are extracted from the request.
2. **Policy matching.** Active policies for your tenant are matched against the context. Matching is by action type and resource pattern; patterns are glob-style, so `customers/*/pii` matches `customers/12345/pii`.
3. **Condition evaluation.** Each matching policy's conditions are checked. Conditions can reference agent capabilities, time windows, resource attributes, or custom metadata.
4. **Decision.** If any matching policy denies and its conditions hold, the result is deny. If all matching policies allow (or none match), the result is allow. A policy may also resolve to verify when it requires a handoff.
5. **Recording.** Every evaluation, allowed or denied, produces a governance event that is written to a tamper-evident audit record. This is the material your auditors review.
6. **Compliance mapping.** The evaluation is tagged with the frameworks it is relevant to, based on the action and resource.

Types of policies

| Policy type | What it controls | Example |

| --- | --- | --- |
Data access	Which agents can read or write which resources	An agent cannot read customer PII without the `pii-approved` capability.
Tool use	Which tools or APIs an agent can invoke	No agent may call the `delete-account` API without the `admin` capability.
Inter-agent communication	Which agents can talk to each other	An agent in one organization cannot message agents in another without a sufficient trust score.
Resource limits	Rate and volume caps on agent actions	No agent may perform more than 100 write operations per hour.
Time-based	Restrict actions to certain windows	Data exports are allowed only during business hours.

Recommended pattern

Wrap sensitive tool calls with a policy evaluation step. Treat a deny as terminal: do not retry the same action hoping for a different answer, and do not fall back to an ungoverned path. Treat a verify as a handoff to a human approver or an approval workflow, then re-evaluate once the condition is satisfied.

Fail closed. If the decision point is unreachable, the safe default is to block the governed action rather than let it through ungoverned. The sidecar is designed so a denied result can never surface as a permissive one.

What gets recorded

Every evaluation produces a governance event that includes:

- the decision (allow, deny, or verify);
- a human-readable reason;
- which policy matched and which condition triggered;
- the agent context (who acted and what they were attempting);
- a content hash linked into a tamper-evident chain;
- the compliance frameworks the evaluation is evidence for.

These records are the raw material for compliance evidence, for streaming into a SIEM through webhooks, and for incident investigation.

Limits and caveats

- Policy enforcement governs the actions your agent routes through Qortara. An action that bypasses the governed path is not evaluated; route sensitive calls through the sidecar or the PDP deliberately.
- The hosted PDP and its endpoints are pre-launch. Build against the decision model now, but do not assume a live hosted instance until Cloud Governance is available.
- A policy is only as good as its conditions. Enforcement applies your rules faithfully; it does not infer intent beyond what your policies express.

Related concepts

- [Saga verification](/docs/concepts/saga-verification) confirms that an allowed action produced the expected outcome.
- [Compliance evidence](/docs/concepts/compliance-evidence) turns these records into audit-ready reports.
- [Webhooks and events](/docs/concepts/webhooks-events) stream decisions into your operations tooling.

Product: Qortara Cloud Governance
Source owner: Qortara
Last reviewed: 2026-06-10
Verified against: Qortara pre-launch docs