QORTARA CLOUD GOVERNANCE

Saga Verification

```
# Saga Verification

Saga verification checks that an authorized action actually produced the outcome it was supposed to, not just that the agent was allowed to attempt it. It groups related agent steps into a single governed workflow and records whether each step's result matched expectation.

> **Availability.** Saga verification is a Qortara Cloud Governance capability and is in pre-launch (code-complete, not yet deployed). It is not part of the open-source sidecar. The endpoints and responses below describe the hosted design; build against the model now, but do not assume a live hosted instance until Cloud Governance is available.

## What a saga is here

In general software a saga is a sequence of operations that must either all succeed or be compensated. In Qortara, a saga tracks action-outcome pairs: for each governed step, did the action produce the result the agent expected?

Most governance systems answer only the authorization question: is the agent allowed to do X? Saga verification adds the verification question: did X actually do what it claimed?

## Why it matters

Consider an agent authorized to transfer funds from account A to account B. The policy evaluation returns allow and the transfer runs. Authorization succeeded, but that alone does not tell you whether the balances actually changed correctly. A downstream system could have applied the wrong amount, partially completed, or silently failed. Saga verification compares the expected outcome to the actual one and records the result as durable evidence.

Many agent actions are only meaningful as a sequence. A single allow or deny does not always explain the full workflow outcome; the saga ties the steps together.

## The verification flow

```text
Agent                Qortara                  Target System
```
```

```
  |                      |                                |
  |-- policy eval: "transfer funds" -->|                |
  |<-- allow, saga_id: s1 -------------|                |
  |                      |                                |
  |-- execute transfer ---------------------------------->|
  |<-- result: new_balance 950 -------------------------|
  |                      |                                |
  |-- verify(saga_id: s1, |                               |
  |   expected 950,      |                                |
  |   actual 950) ------->|                               |
  |<-- status: verified --|                               |
```

The three states

Status	Meaning	What happens
`pending_verify`	The action was allowed and executed, but the outcome is not yet confirmed.	The saga stays open; compliance evidence shows it as pending.
`verified`	The actual outcome matches the expected result.	The saga closes; a positive trust signal is recorded.
`mismatch`	The actual outcome does not match the expected result.	The saga is flagged; a negative trust signal is recorded and an incident is raised.

There is no automatic rollback. A mismatch is a detection-and-evidence mechanism: it records the divergence, signals it, and raises an incident for a human or workflow to handle. Qortara does not silently reverse your system's state on your behalf.

Verification API

After the agent executes a governed action, submit the expected and actual outcomes.

```bash
curl -X POST https://api.qortara.com/v1/saga/verify \
  -H "$AUTH_HEADER" \
  -H "Content-Type: application/json" \
  -d '{
    "saga_id": "saga_abc123",
    "step_id": "step_001",
    "expected_outcome": {
      "balance_a": 50,
      "balance_b": 950
    },
    "actual_outcome": {
      "balance_a": 50,
      "balance_b": 950
    }
```

  }'
```

A verified response:

```json
{
  "saga_id": "saga_abc123",
  "step_id": "step_001",
  "status": "verified",
  "match": true,
  "trust_signal": "positive",
  "audit_event_id": "evt_def456"
}
```

A mismatch response:

```json
{
  "saga_id": "saga_abc123",
  "step_id": "step_001",
  "status": "mismatch",
  "match": false,
  "divergence": {
    "field": "balance_b",
    "expected": 950,
    "actual": 940
  },
  "trust_signal": "negative",
  "incident_id": "inc_ghi789",
  "audit_event_id": "evt_jkl012"
}
```

A mismatch does three things:

1. records a negative trust signal, which lowers the agent's trust score;
2. raises an incident for investigation;
3. logs the divergence details into the tamper-evident audit record.

A concrete multi-step example

An agent processes a customer refund across three governed steps under one saga.

Step 1: read the customer order (governed by a data-read policy).
```

```text
evaluate -> allow (saga_id assigned: saga_refund_001)
agent reads order: order_id 123, amount 50.00
verify -> verified (actual matches expected order data)
```

Step 2: process the refund via the payment API (governed by a financial-write policy).

```text
evaluate -> allow (same saga, step 2)
agent calls payment API: refund 50.00 to customer
verify -> verified (refund confirmed by payment API)
```

Step 3: update the CRM record (governed by a crm-write policy).

```text
evaluate -> allow (same saga, step 3)
agent updates CRM: order status set to "refunded"
verify -> mismatch (CRM shows "partially_refunded", not "refunded")
  -> incident raised
  -> agent trust score decreases
  -> SOC 2 CC8.1 evidence: change-management discrepancy flagged
```

The authorization for every step succeeded. Saga verification is what caught that the final state diverged from intent, and it produced the evidence and the incident without altering the CRM itself.

When to use saga verification

Scenario	Verify?	Why
Agent reads public data	No	Low risk; no meaningful outcome to verify.
Agent modifies customer records	Yes	Data integrity matters; confirm the write succeeded.
Agent makes financial transactions	Yes	Always confirm financial outcomes.
Agent calls external APIs	Yes	External systems can fail silently.
Agent generates a report	Optional	Verify if the report content must match an expectation.

Rule of thumb: if an action has side effects that matter for compliance or business integrity, verify it.

Limits and caveats

- Saga verification is hosted Cloud Governance and is pre-launch; it is not in the open-source sidecar.
- Verification compares the outcome you report against the outcome you expected. It is only as accurate as the actual outcome your code supplies; it cannot independently observe a system it is not told about.
- A mismatch raises an incident and a negative signal. It does not roll back, retry, or compensate the underlying action. Compensation logic remains the responsibility of your workflow.

Related concepts

- [Policy enforcement](/docs/concepts/policy-enforcement) authorizes the steps that a saga later verifies.
- [Trust federation](/docs/concepts/trust-federation) consumes the positive and negative signals a saga produces.
- [Webhooks and events](/docs/concepts/webhooks-events) deliver `saga.step_verified` and mismatch events to your tooling.

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