Webhooks And Events

Webhooks And Events

Qortara emits a structured event for every governance action so your security and observability systems can correlate agent activity with the workflows you already run. Events are delivered to webhook endpoints you configure, signed so you can verify them, and retried if delivery fails.

> **Availability.** Webhook delivery is a Qortara Cloud Governance capability and is in pre-launch. The event catalog, payload format, signature scheme, and retry behavior below describe the hosted design. Build your receiver against this contract now; do not assume a live hosted instance until Cloud Governance is available.

Why stream governance events

A governance decision is most useful when it reaches the place your responders already watch. Routing denies, verification requests, and incidents into your SIEM or paging tool means agent governance becomes part of your existing operations rather than a separate console someone has to remember to check. Streaming the full event flow also gives you a real-time copy of governance activity for your own retention and analysis.

Event catalog

Each governed action produces one of the following event types.

Event type	Description	Typical volume
`policy_evaluation`	Every policy decision (allow or deny) with full context	100 to 10,000 per day per agent
`audit_event`	Signed audit record, one-to-one with `policy_evaluation`	Same as above
`compliance.scan_complete`	Results from a compliance framework scan	1 to 100 per day
`trust.score_updated`	An agent's trust score changed (a new signal was recorded)	Per signal
`trust.attestation_issued`	A cross-org trust attestation was issued	Per lookup
`saga.step_verified`	A saga step was verified (verified or mismatch)	Per verification

`saga.state_mismatch`	A saga step produced an unexpected outcome	Event-driven
`incident.created`	A new incident was raised (for example from a saga mismatch)	Event-driven
`incident.deadline_warning`	A reporting deadline is approaching	Event-driven
`budget_cap_alert`	A budget threshold was crossed (50, 80, or 100 percent)	Event-driven
`rate_limit_exceeded`	A tenant rate limit was hit	Event-driven
`tenant.provisioned`	A new tenant was created	Low volume
`plugin.submitted`	A plugin was registered for review	Low volume

Payload format

Every delivery wraps the event in a standard envelope. The outer fields are stable across event types; the `data` object varies by `event_type`.

```json
{
  "event_id": "evt_abc123def456",
  "event_type": "policy_evaluation",
  "timestamp": "2026-04-10T12:00:00.123Z",
  "tenant_id": "qor:your-tenant-id",
  "schema_version": "1.0",
  "data": {
    "agent_did": "did:qor:agent:abc123",
    "action": "read",
    "resource": "customers/12345/pii",
    "decision": "deny",
    "reason": "Policy 'restrict-pii-access' denies read on customers/*/pii",
    "policy_id": "pol_789ghi",
    "framework_context": ["soc2", "gdpr"]
  },
  "merkle_hash": "a1b2c3d4e5f6..."
}
```

The `event_id` is stable for a given event and is the field you use for deduplication. The `merkle_hash` ties the event to the tamper-evident audit record.

HMAC signature verification

Every delivery includes an `X-Qortara-Signature` header containing an HMAC-SHA256 signature over the raw request body. Verify it before trusting the payload so that only deliveries signed with your webhook secret are accepted.

Verify against the raw bytes of the body, not a re-serialized copy, and use a constant-time comparison.

```python
import hashlib
import hmac

def verify_webhook(payload: bytes, signature: str, secret: str) -> bool:
    """Verify a Qortara webhook HMAC-SHA256 signature."""
    expected = hmac.new(
        secret.encode("utf-8"),
        payload,
        hashlib.sha256,
    ).hexdigest()
    return hmac.compare_digest(f"sha256={expected}", signature)

# In your webhook handler:
payload = request.body  # raw bytes
signature = request.headers["X-Qortara-Signature"]
webhook_secret = os.environ["QORTARA_WEBHOOK_SECRET"]

if not verify_webhook(payload, signature, webhook_secret):
    return Response(status_code=401)

event = json.loads(payload)
# Process the event.
```

Your webhook secret is set when you create the endpoint. Store it the way you store any other credential; it is the shared secret between Qortara and your receiver.

Retry and backoff

If your endpoint returns a non-2xx status, the delivery is retried with increasing delay.

Attempt	Delay before retry	Elapsed by this attempt
1st retry	10 seconds	about 10 seconds
2nd retry	60 seconds	about 70 seconds
3rd retry	300 seconds (5 minutes)	about 6 minutes

After three failed retries the delivery is marked failed. Failed deliveries are visible in the endpoint's delivery history.

Delivery is at-least-once. Under rare network conditions you may receive the same event more than once, so make your handler idempotent and deduplicate on `event_id`.

Configuring an endpoint

```bash
curl -X POST https://api.qortara.com/v1/portal/webhooks \
  -H "$AUTH_HEADER" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "My SIEM",
    "url": "https://your-endpoint.com/qortara-events",
    "auth_type": "custom_header",
    "auth_header_name": "Authorization",
    "auth_header_value": "Bearer your-secret",
    "event_types": ["policy_evaluation", "compliance.scan_complete",
"budget_cap_alert"],
    "active": true
  }'
```

Subscribe only to the event types your receiver needs. Subscribing to everything is convenient for a catch-all SIEM but adds volume you then have to filter.

Common integration patterns

Destination	Auth approach	Suggested event types
Splunk HEC	Custom header with a Splunk HEC token	All events
Datadog Logs	Custom header with a Datadog API key	`policy_evaluation`, `audit_event`
PagerDuty	PagerDuty Events API	`saga.state_mismatch`, `incident.created`
Slack	Slack incoming webhook URL	`budget_cap_alert`, `compliance.scan_complete`
Custom backend	Custom header with your own auth	Whichever events your system needs

Limits and caveats

- Webhook delivery is hosted Cloud Governance and is pre-launch.
- Always verify the signature before acting on a payload, and verify against the raw body.
- Because delivery is at-least-once, a non-idempotent handler can double-process an event; deduplicate on `event_id`.
- A 2xx response acknowledges receipt, not successful downstream processing. If you need durable handling, acknowledge quickly and process asynchronously from your own queue.

Related concepts

- [Policy enforcement](/docs/concepts/policy-enforcement) generates the `policy_evaluation` events.

- [Saga verification](/docs/concepts/saga-verification) generates verification and mismatch events.
- [Compliance evidence](/docs/concepts/compliance-evidence) is built from the same governance activity these events carry.

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