Operate and recover the Chatwoot connector

Use this runbook to locate a Chatwoot connector failure before changing state. It starts with process, readiness, lease, storage, and provider evidence, then moves to Action, webhook, and central Event delivery.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/operations/health-observability-and-recovery.md

When a provider mutation has an uncertain outcome, stop new related mutations. Read durable Action and provider state before any retry or replacement Action.

Signal order

Read signals in this order:

Order	Surface	Primary question answered
1	`GET /health`	Is the host HTTP process responding
2	`GET /ready`	Can this replica accept routed Actions now
3	`GET /metrics`	Is pressure, rejection, or lease recovery changing
4	Connector registry	Is the version, route, replica status, and lease authoritative
5	Durable Action state	Was work admitted, dispatched, completed, or left uncertain
6	Chatwoot provider reads	What state exists in the configured account
7	Webhook response and local Event state	Was inbound work suppressed, persisted, or queued
8	Central Event ingress	Was a locally durable Event acknowledged upstream

Do not use a later signal to overwrite an earlier fact. A live process can be not ready, and a ready host cannot establish the outcome of an earlier provider mutation.

Liveness is process-only

`GET /health` always returns a static HTTP `200` body while the host router can serve:

```
JSON
{
  "status": "ok",
  "protocol": "AIP",
  "version": "1.0",
  "service": "aip-connector-host"
}
```

This route does not probe PostgreSQL, Chatwoot, the lifecycle lease, gateway trust, admission, webhook replay state, or central Event delivery.

Use it for process and listener liveness only.

Readiness is a live composite probe

`GET /ready` performs a bounded durable-storage check and the connector health probe. It also reads the local drain flag and current lease expiry.

HTTP `200` requires all four conditions:

- the replica is not draining;
- its lifecycle lease is still valid;
- durable runtime storage is readable and writable;
- authenticated Chatwoot account health is ready.

Any false condition returns HTTP `503`. The body has this source-owned shape:

```
JSON
{
  "status": "ready",
  "lease_valid": true,
  "draining": false,
  "runtime": {
    "durability": "durable_shared",
    "ready": true,
    "detail": "durable runtime storage is readable and writable",
    "recovery": {
      "recovered_at": "2026-01-01T00:00:00Z",
      "pending_approvals": 0,
      "recoverable_transactions": 0,
      "queued_results": 0,
      "delegation_results": 0,
      "callback_deliveries": 0
    }
  },
  "connector": {
    "ready": true,
    "detail": "Chatwoot account API is reachable"
  }
}
```

`status` changes to `not_ready` on failure. Values shown above are an illustrative healthy response; inspect the returned values rather than matching the example timestamp or counts.

The `runtime.recovery` object is a bounded startup-recovery summary. It is not a current queue, approval, transaction, callback, webhook, or outbox backlog.

Chatwoot product health

The connector health probe performs:

```
TEXT
GET /api/v1/accounts/{configured_account_id}
api_access_token: <configured token>
```

The common host bounds this probe by the configured health timeout, which defaults to 2,000 ms.

A successful result establishes provider reachability and token acceptance for that account route at that moment. It does not exercise:

- all 153 capabilities;
- approval or idempotency;

- provider mutation payloads;
- webhook authentication;
- message delivery;
- central Event publication.

Host metrics

`GET /metrics` renders all eight Prometheus families on every scrape:

Metric	Type	Operational meaning
`aip_connector_host_ready`	Gauge	Cached connector and storage flags combined with live lease and drain state
`aip_connector_host_storage_ready`	Gauge	Last stored durable-storage probe result
`aip_connector_host_in_flight`	Gauge	Actions currently executing in this process
`aip_connector_host_requests_total`	Counter	Native message requests received
`aip_connector_host_rejected_total`	Counter	Instrumented validation, trust, route, readiness, capacity, credential, approval, or gateway rejections
`aip_connector_host_heartbeat_failures_total`	Counter	Failed lease renewal or recovery attempts
`aip_connector_host_lease_recovery_attempts_total`	Counter	Re-registration attempts after lease expiry
`aip_connector_host_lease_recoveries_total`	Counter	Re-registrations that applied a new lease

The metric families have no application labels and reset when the process restarts. Use bounded scrape-target metadata to identify the replica, version, zone, and connector type.

The ready gauge does not run a provider or storage probe. `/ready` and the heartbeat loop update those cached flags. Read the readiness body when the cause matters.

Do not calculate success as requests minus rejections. Response signing, process errors, durable Action state, and provider outcomes are not represented by that subtraction.

Source-visible observability gaps

The reviewed host exposes no public Chatwoot-specific metric for:

- provider request latency or status distribution;
- response-size rejection count;
- webhook authentication or loop-suppression count;
- replay-state entries or CAS contention;
- local Event append count;
- pending central Event outbox batches;
- provider business objects or delivery state.

`ConnectorHostEventPublisher::pending_batches` exists as an internal bounded method, but the Chatwoot HTTP host does not publish it as a route or metric.

Use the webhook response's `central_delivery` value as per-delivery evidence. Use controlled internal diagnostics for current outbox state. Do not infer an empty backlog from a healthy scrape.

Catalogue results can include a provider request ID from `x-request-id` or `x-runtime`. Composite results do not expose a provider request ID. Preserve the distinction during incident correlation.

Heartbeat and lease behavior

The heartbeat loop periodically:

1. probes Chatwoot account health;
2. probes durable storage;
3. records the two readiness flags;
4. reports in-flight Actions;
5. renews the lease or recovers an expired lease.

An unhealthy heartbeat moves the registry replica offline. When storage and provider health recover, the loop can return it to ready under the current lease or re-register after expiry.

Lease recovery keeps one request ID and registration payload across ambiguous control-plane failures. This prevents a lost response from creating a second lifecycle mutation. Definitive admission or configuration rejection clears that pending request before a corrected attempt.

Increasing heartbeat failures with no recovery indicates a control-plane, identity, sequence, admission, or network boundary. It does not by itself prove a Chatwoot provider failure.

First response

1. Contain

Stop new related mutations when:

- provider outcome is uncertain;
- the wrong account or route may be active;
- storage consistency is unknown;
- credentials may be revoked or misbound;
- webhook or outbox state may be lost;
- a rollout changed version or manifest identity.

Drain the replica when it must stop receiving new Actions. Keep the process and state available long enough to inspect pinned work and complete bounded recovery.

2. Capture bounded evidence

Retain:

- host, version, instance, replica, tenant, and external-account identity;
- `/health`, `/ready`, and `/metrics` observations with timestamps;
- registry route, replica status, lease sequence, and expiry;
- configuration and credential revisions without secret bytes;
- Action ID, capability, canonical input hash, approval, and key hash;
- provider status and request ID when exposed;
- webhook delivery ID and response fields when relevant;

- final provider reads used for reconciliation.

Do not retain tokens, signing seeds, webhook secrets, or unrestricted customer payloads in incident notes.

3. Classify the failing boundary

Observation	Likely boundary	Next evidence
`/health` unavailable	Process, listener, edge, or network	Supervisor state and bounded startup error
`/health` is `200` and `/ready` is `503`	Drain, lease, storage, or provider	Read each readiness field
`runtime.ready` is false	Durable PostgreSQL or storage timeout	Database reachability, migration, and ownership
`connector.ready` is false	Provider origin, token, account, TLS, or provider availability	Bounded health detail and credential revision
`lease_valid` is false	Heartbeat, control plane, sequence, or admission	Registry replica and recovery counters
`draining` is true	Planned or interrupted shutdown	Lifecycle owner and in-flight count
In-flight equals configured capacity	Local saturation	Action duration, route pressure, and capacity rejection
Rejections increase	Input, trust, route, readiness, capacity, credential, approval, or gateway	Signed protocol error code
Webhook returns `401`	HMAC, clock, raw bytes, JSON, or connector mapping	Exact header identity and bounded host diagnostic
Webhook returns `503`	Replay state, Event Log, or durable outbox	Durable-state health and retained delivery ID
Webhook returns `durably_queued`	Central Event delivery pending	Fixed endpoint, signer, and internal outbox diagnostics
A mutation returns connector failure	Provider may already have applied it	Durable Action plus authoritative provider reads

Recovery procedures

Storage not ready

1. Keep the replica out of routing.
2. Verify the configured database URL file and least-privilege role.
3. Restore network, database, schema, and storage-health availability.
4. Preserve Action, replay, Event, and outbox tables.
5. Restart only when production Runtime Stores remain one coherent durable

class.

6. Inspect startup recovery and require `runtime.ready: true`.
7. Reconcile every mutation whose dispatch boundary is unknown.

Do not edit leases, replay identities, idempotency records, or outbox entries manually to force readiness.

Provider health or authentication fails

1. Confirm the fixed provider origin and configured account.
2. Identify the credential revision without printing the token.
3. Distinguish TLS, DNS, timeout, `401`, `403`, rate limit, and server failure.
4. Rotate the owner-only token file through a replacement replica when needed.
5. Re-run readiness and the safe `account.get` canary.
6. Reconcile prior mutations before enabling writes.

The token is read at startup. Replacing file bytes does not prove a running process adopted them.

Lease or control-plane recovery fails

1. Compare local instance, replica, version, manifest, artifact, and sequence

with the registry.

2. Verify the fixed control endpoint, trusted DID, TLS, and lifecycle database.
3. Preserve the host's pending recovery request across ambiguous transport

failure.

4. Correct definitive admission or configuration rejection before another

request.

5. Require a valid ready lease and correct route before admitting traffic.

Do not create a replacement replica with the same replica ID while a live lease or pinned Action ownership remains unresolved.

Capacity is exhausted

`connector_host.capacity` is returned before provider dispatch and is marked retryable by the host. Confirm durable Action state before repeating the same correlated request.

Reduce upstream pressure or add an admitted replica before raising `AIP_CONNECTOR_HOST_MAX_IN_FLIGHT`. A higher local limit does not increase provider capacity or preserve latency.

Provider mutation outcome is uncertain

Treat provider-temporary, transport, cancellation, and oversized-response invocation failures as possible provider success.

1. Keep the original Action ID, key, approval, and input.
2. Read durable Action and provider state.
3. Correlate provider request ID or AIP Action content attributes when present.
4. Accept matching state, escalate ambiguity, or create a separately approved

corrective intent.

5. Never replace uncertainty with an uncorrelated retry.

All Chatwoot connector failures currently report `retryable: false`. Capability contracts separately identify safe GET reads and the status composite.

Webhook authentication or replay state fails

Preserve the provider delivery ID, timestamp, signature identity, and protected raw-body evidence. Do not log the secret or full customer payload.

Correct clock, proxy byte preservation, signature format, secret revision, or durable replay state. Replay the same provider delivery only under the signed freshness and deterministic Event rules.

Central Event delivery is pending

A `durably_queued` webhook response means the local outbox owns delivery. Repair the fixed central endpoint, TLS policy, host signer, admitted channel, or central ingress without asking the provider to create a new delivery.

The background worker retries with bounded backoff. Preserve the outbox and Event Log through restart. Confirm central acknowledgement and Event identity before declaring the incident resolved.

Post-recovery verification

Require:

- `/health` is reachable;
- `/ready` returns `200` with a valid lease, no drain, and both probes ready;
- the registry exposes the intended version, replica, endpoint, and account;
- host-ready and storage-ready gauges agree with the latest readiness probe;
- recovery counters stop increasing unexpectedly;
- the safe account-read canary uses the intended route;
- every uncertain mutation has an explicit provider-state decision;
- webhook and central Event delivery are reconciled when enabled;
- no state, secret, or customer data was deleted to obtain recovery.

Monitor through at least one lease renewal and the deployment's normal provider observation window. These checks establish operational recovery, not complete capability qualification.

Related documentation

- Deploy the Chatwoot connector
- Chatwoot connector configuration
- Webhook security, loop prevention, and replay
- Observe and recover
- Metrics reference