Qualify the Chatwoot connector

Use this page to decide whether one exact Chatwoot connector artifact supports a bounded release claim. It keeps source-defined tests, standalone-image checks, controlled provider fixtures, and actual Chatwoot observations as separate evide

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/qualification/README.md

Current exact-artifact result: not established by this page. The reviewed source defines qualification gates, but this documentation review did not run them. No retained isolated-live Chatwoot result for the current standalone host was found in the pinned tree.

Use evidence terms consistently

Term	Required evidence
Implemented	The behavior and contract exist in identified source
Conformant	The exact artifact passed applicable conformance scenarios with retained results
Controlled-qualified	The exact artifact passed a named deterministic topology and failure matrix
Isolated-live verified	The exact artifact produced reviewed observations against identified Chatwoot artifacts
Production-observed	Deployment records show the behavior under a stated production boundary

A test function proves that a gate exists. It does not prove that the gate ran or passed for the artifact under review.

Bind the claim to one identity set

Every result must bind the identities that can change its meaning:

Identity class	Required values
AIP source	Commit, tree or complete dirty-source snapshot, and clean or dirty status
Connector artifact	Immutable image ID, component, version, source label, entrypoint, and architecture
Contract	Manifest digest, 153-capability catalogue digest, schemas, and implementation-support map
Chatwoot upstream	Commit `8818d276b954ac4f84cffd8915c99f40e43804ed`, tree or image, migrations, and API configuration
Harness	Source digest, runner image, configuration digest, and selected cases
Runtime	Container engine, databases, trust policy, account, and network topology
Run	UTC interval, unique run ID, evaluator identity, and evidence-root digest

The reviewed AIP commit is `97be86e9efedf07ecf1783b03800f683f107fb04`, with Git tree `2d83a382a312c601ecbbf5ed03f8af2b0218bdae`. These source identities do not identify a built image or a completed run. A mutable tag, branch, or filename is not an artifact identity.

Evaluate four independent gates

Gate	Source-owned entry point	What it can prove	What it cannot prove
Q1: connector contracts	Focused tests in `aip-connector-chatwoot`	Deterministic connector semantics through public Rust boundaries	Container, fleet, or external Chatwoot behavior
Q2: standalone image	`verify-product-images.sh`	Image identity, runtime user, entrypoint, labels, inspect, history, and SBOM	Provider compatibility, signature, vulnerability policy, or admission
Q3: controlled product fleet	`run-product-qualification.sh` and `verify-product-fleet.sh`	Admission, selected operations, policy, isolation, fail-closed behavior, and recovery against fixtures	Actual Chatwoot compatibility or all capabilities
Q4: isolated live	A reviewed campaign manifest and harness	Selected behavior against identified Chatwoot artifacts	Arbitrary deployments, every capability, scale, or production readiness

Record a result for every gate required by the claim. A deterministic fixture cannot substitute for actual Chatwoot, and a provider observation cannot substitute for host admission or failure recovery.

Review focused connector evidence

The pinned connector crate contains nine focused tests. Together they cover:

- outbound message authentication and Action identity propagation;
- webhook signature verification, loop prevention, and deterministic duplicate

reconstruction after connector restart;

- conservative retry contracts for non-idempotent mutations;
- uniqueness and governance for 150 catalogue operations;
- manifest bindings for all three composites and catalogue capabilities;
- mutation path, body, authorization, and idempotency projection;
- bounded multipart upload and allowed-operation filtering;
- credential redaction from connector debug output.

Retain the exact test selection, binary or build identity, process status, timestamps, structured results, and redacted diagnostics. The source functions alone are implementation evidence. They are not an observed pass.

These focused tests do not form a complete frozen-conformance report. A release claim that requires cross-connector conformance must retain the applicable global suite separately.

Review the standalone image gate

The image verifier builds `aip-host-chatwoot` with the common host Dockerfile. It requires an immutable image ID, runtime user `10001:10001`, entrypoint `/usr/local/bin/aip-connector-host`, exact component labels, image inspect, full history, and an SPDX JSON SBOM.

Review the Chatwoot-specific artifacts rather than relying on a shared summary. An SBOM is an inventory. It is not an image signature, vulnerability decision, license decision, provenance attestation, or provider qualification.

Review the controlled product-fleet gate

The controlled fixture publishes 153 Chatwoot capabilities: three AIP composites and 150 upstream-pinned account operations. The selected positive cases require:

- `cap:chatwoot:account.get` and `cap:chatwoot:agent.list` through the gateway;

- approval-governed `cap:chatwoot:conversation.create` with stable Action and

idempotency identities;

- fixture audit entries for account read and conversation creation;
- authorization and idempotency markers at the fixture boundary;
- exclusion of `cap:chatwoot:account.get` from an unrelated tenant catalogue.

The fleet verifier also requires an independently owned Chatwoot runtime database with all five runtime tables. It includes graceful restart, `SIGKILL` and expired-lease fail-closed recovery, dual-gateway failover, shared PostgreSQL outage and recovery, and final ready-lease checks.

This gate exercises three capability IDs against a deterministic provider fixture. It does not exercise the other 150 IDs, webhook ingress, multipart uploads, a real Chatwoot deployment, provider-version drift, or production traffic. Label it controlled product-fleet qualification.

Define an isolated-live gate before running it

The pinned tree does not contain a source-owned isolated-live Chatwoot campaign or a retained current result. A new gate should use a dedicated account, reversible data, independent requester and approver identities, isolated credentials, and explicit cleanup.

Select cases by distinct risk and transport behavior rather than operation count. A bounded campaign should include authenticated account discovery, one read, one approval-governed mutation, same-key replay, one deliberate idempotency collision, authenticated webhook delivery, duplicate delivery, loop prevention, provider rejection, uncertain mutation handling, restart, and storage recovery.

Record the provider resource before and after each mutation. One successful HTTP exchange is insufficient when the provider effect, Action result, local Event, and central Event delivery are separate boundaries.

Account for all 153 capabilities

Maintain one generated or machine-validated coverage row per published capability:

Field	Required value
Capability ID	Exact admitted identifier
Contract evidence	Method, route, schemas, risk, approval, retry, and idempotency mapping
Deterministic evidence	Test and result artifact, or an explicit gap
Controlled-fleet evidence	Case and artifact, or `not_run`
Isolated-live evidence	Provider identity, case, and artifact, or `not_run`
Failure evidence	Auth, validation, rejection, uncertainty, replay, and recovery cases exercised
Side-effect control	Approval, idempotency, provider-state check, and cleanup as applicable
Review decision	Accepted scope, limitation, owner, reviewer, and time

Every row needs contract evidence. Live mutation of all 90 mutation capabilities is not a default requirement and may be unsafe. Select reversible live cases by unique behavior, then publish every unobserved capability as a gap.

Retain a complete evidence set

A qualification record should contain:

1. immutable AIP, image, manifest, schema, Chatwoot, harness, dependency,

topology, and run identities;

2. a gate manifest naming every required case and invariant;
3. raw exits, structured per-case results, and redacted diagnostics;
4. admission, catalogue, implementation-support, route, and lease snapshots;
5. image inspect, history, SBOM, and separate supply-chain decisions;
6. provider request and audit correlations without tokens or personal data;
7. durable Action, approval, idempotency, webhook, Event, and outbox evidence;
8. failure and recovery ordering for process, lease, database, and provider;
9. a generated 153-row capability coverage matrix;
10. checksums for every artifact and one evidence-root checksum;
11. verified cleanup, residual-resource inventory, and retained exceptions;
12. a summary stating scope, result, exclusions, reviewer, and UTC time.

A summary file is not a pass by itself. Review every referenced artifact and ensure later recovery did not hide an earlier failed invariant.

Assign an unambiguous result

Result	Use when
`passed`	Every required invariant passed for the exact identity set, artifacts are complete, and gaps match the claim
`failed`	A required invariant failed, identity mismatched, evidence leaked or was corrupted, or cleanup failed
`blocked`	A required dependency or authority was unavailable before evaluation
`not_run`	The gate was not executed for this identity set
`historical`	A retained result belongs to a different source, artifact, topology, or evidence standard

Do not collapse `blocked`, `not_run`, or `historical` into `passed`. One gate can pass while a broader release claim remains unestablished.

Publish only the supported claim

Acceptable wording binds the result to its evidence:

> At the recorded time, the identified AIP and Chatwoot connector artifacts
>
> passed the listed deterministic and controlled-fleet cases in the retained
>
> topology. The coverage matrix identifies every unobserved capability and
>
> excludes live-provider and production properties.

Do not publish `fully compatible`, `all 153 capabilities live verified`, `exactly once`, or `production-ready` unless the retained campaign defines and proves each phrase.

Reviewer checklist

- [ ] Source, image, manifest, upstream, harness, topology, and run identities

are immutable and mutually consistent.

- [ ] Required gates are explicit and no fixture result is labeled live.

- [ ] The 153-row coverage matrix matches the admitted catalogue.
- [ ] Connector, image, and controlled-fleet results are retained and redacted.
- [ ] Mutation evidence binds approval, idempotency, provider effect, Action state, and uncertainty decisions.
- [ ] Webhook evidence separates authentication, replay, local Event storage, and central publication.
- [ ] Image inventory is not presented as signature, scan, or provenance.
- [ ] Cleanup and residual resources are independently recorded.
- [ ] Final wording states time, scope, result, gaps, and exclusions.

Related documentation

- Conformance and qualification
- Connector fleet qualification
- Live-product qualification
- Chatwoot capability index
- Release artifacts and verification