# Deploy the Chatwoot connector

Use this guide to deploy one immutable aip-host-chatwoot instance for one Chatwoot account. The result is an admitted, durable, observable route that can be replaced or rolled back without changing central aipd.

| | |
|---|---|
| SECTION | Connectors |
| TYPE | Connector |
| PROTOCOL | AIP 1.0 |
| SOURCE | docs/connectors/chatwoot/operations/deployment.md |

This is a production design procedure. The repository Compose stack is useful qualification evidence, but it is not a production deployment template.

## Deployment boundary

One logical connector instance owns:

- one admitted connector type and immutable version;
- one tenant;
- one Chatwoot account;
- one provider origin and API token;
- one durable host-state boundary;
- one or more separately identified replicas, each with a fixed native AIP

endpoint;

- an optional Chatwoot webhook ingress and central Event route.

The process serves native AIP on a configurable bind address, defaulting to `0.0.0.0:8081`. Its advertised endpoint must be an absolute HTTPS URL ending exactly in `/aip/v1/messages`. Explicit loopback development can opt into HTTP; production cannot.

## Topology and ownership

| Connection | Direction | Owner | Required protection |
|---|---|---|---|
| Gateway to host | Inbound native AIP | AIP platform | TLS, signed envelopes, trusted gateway identity |
| Lifecycle service to host | Inbound control | AIP platform | Fixed endpoint, DID verification, sequence and replay fencing |
| Host to Chatwoot | Outbound provider API | Connector operator | HTTPS origin, owner-only API token, no authenticated redirects |
| Host to PostgreSQL | Durable runtime state | Connector operator | Dedicated least-privilege URL file and migrations |
| Chatwoot to host | Optional webhook | Provider and connector operator | TLS, raw-body HMAC, freshness, replay state |
| Host to central Event ingress | Optional signed publication | AIP platform | Fixed endpoint, host signer, durable outbox |

Keep the host port private behind the selected TLS edge. Expose only the native endpoint and, when enabled, `POST /webhooks/chatwoot`.

Every replica of one logical instance uses a unique replica ID. Replicas share the admitted version, instance identity, external account, and durable state needed for leases, replay fencing, and Event publication.

# 1. Freeze the release unit

Before provisioning, record:

| Release evidence | Required value |
| --- | --- |
| AIP source revision | Exact commit |
| Chatwoot connector image | Immutable OCI digest |
| Workspace version | Version embedded in the artifact |
| Connector type and version | Exact admitted IDs |
| Upstream Chatwoot baseline | `8818d276b954ac4f84cffd8915c99f40e43804ed` |
| Capability surface | Three composites plus the effective operation allowlist |
| Configuration revision | Positive monotonic value |
| Admission package | Signed package bound to artifact and manifest |
| Trust policy | Exact policy used to admit the version |

Do not deploy from a mutable image tag alone. If an operations allowlist changes, the immutable manifest changes. Advance the connector version and its admission material before routing work to it.

The source fleet preparation script derives a source digest, image tag, image ID, release ID, and release revision. That is a useful reference pattern, not evidence for an image that was built elsewhere.

# 2. Bind the external account

Set one stable Chatwoot account in:

```
TEXT
AIP_CHATWOOT_ACCOUNT_ID
```

Bind the same account boundary in the connector registry through:

```
TEXT
AIP_CONNECTOR_HOST_EXTERNAL_ACCOUNT_ID
AIP_CONNECTOR_HOST_EXTERNAL_ACCOUNT_SYSTEM=chatwoot
```

The reviewed binary inserts `AIP_CHATWOOT_ACCOUNT_ID` into provider routes but does not compare it with the registry external-account ID. Enforce equality in deployment policy and admission review.

Use a separate logical instance for another Chatwoot account. Do not let an Action select `account_id`.

# 3. Provision durable state

Give the host an owner-only PostgreSQL URL file through:

```
TEXT
AIP_CONNECTOR_HOST_DATABASE_URL_FILE
```

The host runtime uses durable state for Action handling, lifecycle recovery, profile state, webhook replay fencing, local Events, and the central Event outbox. Provision and migrate it before a replica registers.

All replicas of one logical instance need a coherent storage authority. Process-local or independently provisioned state cannot provide shared replay, lease, or outbox behavior.

Back up and restore this state as one documented recovery unit. Never clear replay or outbox records merely to make readiness green.

## 4. Place identity and trust material

Provision the shared connector-host identity fields:

- connector type, version, instance, and unique replica IDs;
- tenant and membership IDs;
- immutable artifact digest;
- public native endpoint;
- trust domain;
- trusted gateway principal and DID;
- fixed control-plane endpoint and DID;
- host signing seed file;
- optional private-PKI CA file;
- credential ID, issuer, scopes, revision, and revision-policy file.

Signing seeds, database URLs, API tokens, webhook secrets, and private CA material belong in bounded owner-controlled files. Environment variables carry paths and non-secret identifiers.

The public endpoint identity, admitted version, manifest digest, and replica signer must agree. Do not reuse a replica signing identity across unrelated instances.

## 5. Configure the provider boundary

Set:

```
TEXT
AIP_CHATWOOT_BASE_URL=https://chatwoot.example.test
AIP_CHATWOOT_ACCOUNT_ID=<stable-account-id>
AIP_CHATWOOT_API_TOKEN_FILE=/run/secrets/chatwoot-api-token
```

The provider URL must be an origin without credentials, path, query, or fragment. HTTPS is required outside explicit loopback development.

The token file must be owner-controlled, non-empty UTF-8, and at most 16 KiB. Give the token only the provider permissions required by the admitted capability surface.

Optional product controls are:

| Setting | Deployment decision |
| --- | --- |
| `AIP_CHATWOOT_OPERATIONS_FILE` | Exact JSON suffix allowlist; omission admits all 150 catalogue operations |
| `AIP_CHATWOOT_MAX_RESPONSE_BYTES` | Bounded response limit from 1 through 134,217,728 bytes |

The three connector composites are not filtered by the operations file. Apply tenant policy separately when one must be unavailable.

# 6. Choose the ingress mode

## Outbound-only

Omit `AIP_CHATWOOT_WEBHOOK_SECRET_FILE`. The process creates no Chatwoot webhook route and does not require a central Event endpoint.

Use this mode when AIP only calls Chatwoot and no provider Event must enter through this host.

## Webhook-enabled

Set:

```
TEXT
AIP_CHATWOOT_WEBHOOK_SECRET_FILE=/run/secrets/chatwoot-webhook-hmac
AIP_CONNECTOR_HOST_EVENT_ENDPOINT=https://aipd.example.test/aip/v1/connector-events
```

The host refuses this mode when the signed Event publisher is unavailable. Route Chatwoot traffic to `POST /webhooks/chatwoot` without altering the three headers or body bytes.

The Event endpoint must pass fixed-host, TLS, private-network, and signer policy. A webhook secret does not replace those central publication controls.

# 7. Admit before starting

Apply the connector admission package and trust policy before the replica starts. Admission must bind:

- connector type and immutable version;
- semantic manifest and capability set;
- artifact digest and provenance;
- supported native and connector profiles;
- external-account and tenant policy;
- public endpoint and lifecycle expectations.

Use separate registry credentials for admission, lifecycle control, and data-plane reads. The host must not receive an administrator credential.

The source qualification graph runs `admit-chatwoot` before `chatwoot-acme-a`. Preserve that dependency in the production orchestrator, even when service names differ.

# 8. Start dependencies in order

Use this order:

1. durable PostgreSQL and required migrations;
2. connector registry and the admitted Chatwoot version;
3. control plane and trusted gateway;
4. TLS edge and fixed internal routes;
5. central callback and Event ingress when configured;
6. the Chatwoot host replica;
7. lifecycle registration and lease renewal;
8. readiness and safe canary checks;
9. production route enablement.

Stop when any earlier authority is unavailable. Do not start the host with a temporary identity, unadmitted version, mutable digest, or substitute account.

The source qualification service also waits for its controlled provider fixture and two healthy gateways. Replace fixture dependencies with real provider and platform readiness ownership; do not copy fixture hostnames into production.

## 9. Gate readiness

Require all of these signals:

| Gate | Established fact |
| --- | --- |
| Process health | Host server and local supervisor are running |
| Durable store | Required runtime and profile-state operations are available |
| Admission | Type, version, instance, replica, artifact, and manifest match |
| Lifecycle lease | Replica is registered, healthy, and not draining |
| Product health | Authenticated account GET succeeded for the configured account |
| Route | Gateway resolves the admitted instance and replica |
| Webhook mode, when enabled | Replay state and central Event publisher are configured |

The product health probe calls:

```
TEXT
GET /api/v1/accounts/{configured_account_id}
```

It proves account-route reachability and token acceptance at that moment. It does not validate every capability, webhook delivery, approval path, or mutation payload.

## 10. Run a safe canary

Before admitting mutations:

1. read the discovered manifest through the tenant route;
2. verify connector ID, version, profiles, and effective capabilities;
3. call `cap:chatwoot:account.get` with a new read Action ID;
4. confirm the result belongs to the configured account;
5. retain gateway, route, host, and provider request identities;
6. verify no unexpected replica or account served the Action.

When webhook mode is enabled, add a provider-controlled signed delivery. Check that a non-looping delivery becomes one local Event and either reaches central AIP or reports `durably_queued`. Verify a connector-marked loop produces zero Events.

Do not use a customer-visible message, deletion, import, export, merge, macro, or handoff as the first canary.

## 11. Enable traffic gradually

Route a bounded tenant and capability subset first. Observe:

- Action admission and completion;
- provider authentication and request identities;
- host lease and in-flight counts;

- response-size failures;
- uncertain provider outcomes;
- webhook replay, Event persistence, and outbox backlog when enabled.

Expand only after the safe read path, approval path, and recovery ownership are understood. Readiness alone is not a rollout success criterion.

## Replace a replica

For a same-version replacement:

1. provision a new replica ID and signer;
2. start it against the same logical instance and coherent durable state;
3. wait for registration, health, lease, route, and safe canary gates;
4. stop new assignments to the old replica;
5. allow pinned work to settle within the drain boundary;
6. mark the old replica offline and stop it;
7. retain lifecycle and Action evidence.

Configuration that changes the manifest, account, provider origin, capability set, or artifact belongs to a new immutable version or instance. Do not relabel an existing admitted version.

## Roll back safely

Keep the previous artifact, admission record, configuration, and provider compatibility decision available until the new version clears its rollout window.

Rollback means routing new work to a previously admitted compatible version, not overwriting the failed version in place. Preserve:

- current durable Action and Event state;
- idempotency records;
- webhook replay and outbox state;
- provider request correlation;
- pinned work on the replica that owns it.

Drain the regressed replica before removing it. Do not move an ambiguous provider mutation to another replica under a new Action ID.

## Deployment failures

| Failure | Stop condition and response |
| --- | --- |
| Admission mismatch | Do not register; reconcile artifact, manifest, version, and trust policy |
| Account identities differ | Do not route work; correct deployment policy and use one stable account |
| API token rejected | Keep route unavailable; rotate the owner-only file and credential revision |
| Provider origin invalid | Correct the HTTPS origin; do not weaken URL or redirect policy |
| Durable state unavailable | Keep readiness false; restore storage before lifecycle or webhook acceptance |
| Event endpoint missing in webhook mode | Fail startup or return to outbound-only mode intentionally |
| Product health fails | Preserve bounded diagnostics; do not prove health with an unrelated endpoint |
| Canary served by wrong route | Remove traffic and correct registry or gateway identity |
| New version regresses | Stop expansion, drain it, and route new work to the compatible admitted version |

## Deployment evidence

Retain:

- source revision, image digest, SBOM and provenance identity;
- connector type, version, instance, replica, tenant, and external account;
- manifest and operations-allowlist hashes;
- admission package and trust-policy identity;
- configuration revision and credential revision;
- migration and startup outcomes;
- health, lease, route, and canary observations;
- webhook and outbox checks when enabled;
- rollout, drain, rollback, and operator decisions.

Do not retain secret contents or unredacted provider payloads as deployment evidence.

## Completion criteria

Deployment is complete only when:

- the immutable artifact and manifest match admission;
- provider and registry account boundaries agree;
- secrets and signing material are owner-controlled files;
- storage, identity, control, gateway, and TLS dependencies are healthy;
- the replica holds a valid non-draining lease;
- product health and the safe account-read canary pass;
- webhook mode, when enabled, has durable replay and Event delivery ownership;
- rollback artifacts and procedures remain available;
- no mutation is used to substitute for a safe deployment check.

These gates establish a controlled deployment path. They do not establish production qualification for every Chatwoot capability.

## Related documentation

- Chatwoot connector configuration
- Run the Chatwoot connector quickstart
- Production deployment
- Deploy the connector fleet
- Connector host lifecycle