Run the Chatwoot connector quickstart

In this tutorial, you will start the repository's deterministic Chatwoot path, read its tenant-bound account, and create one conversation after an explicit approval. Both Actions travel from aipctl, through product-neutral aipd, to the stan

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/getting-started/quickstart.md

The provider is a controlled local fixture. You do not need a Chatwoot deployment or credential, and the procedure does not call an external service. This is a learning exercise, not a production deployment or qualification run.

What you will verify

By the end, you will have observed:

- source-derived local images and connector release identities;
- registry admission before the Chatwoot host starts;
- one host bound to one AIP tenant and one configured Chatwoot account;
- a signed, low-risk `cap:chatwoot:account.get` Action;
- a `cap:chatwoot:conversation.create` Action held for approval;
- a signed approval followed by one completed, idempotent result;
- cleanup limited to the named tutorial Compose project.

You will not test a real Chatwoot deployment, webhook ingress, cancellation, rollback, gateway failover, or a production security boundary.

Prerequisites

You need:

- a clean `aip-core` checkout at the reviewed source revision;
- Docker Engine, Docker Compose v2, and BuildKit;
- enough CPU, memory, and disk for the source-owned fleet images;
- TCP port `18443` free on loopback;
- a POSIX-compatible shell, `lsof`, `sed`, and `grep`;
- authority to remove the named tutorial project after reviewing its state.

Run every command from the `aip-core` repository root. Image preparation builds the gateway, control plane, fixtures, and all product hosts, so the first run can take substantially longer than a single native binary.

The Compose project name is fixed as `aip-connector-fleet-qualification`. Do not continue if another investigation or retained qualification run owns that project or its volumes.

1. Verify the source boundary

Set and verify the reviewed revision:

```
SH
export AIP_SOURCE_REVISION=97be86e9efedf07ecf1783b03800f683f107fb04

test "$(git rev-parse HEAD)" = "$AIP_SOURCE_REVISION"
test -z "$(git status --porcelain)"
```

Both checks must exit with status `0`. Use a separate clean checkout if you have local work; do not hide or discard changes to satisfy the preflight.

Confirm that Docker and Compose are available:

```
SH
docker info >/dev/null
docker compose version
```

Confirm that the fixed edge port is unused:

```
SH
if ! command -v lsof >/dev/null; then
  echo "lsof is required for the port preflight" >&2
  exit 1
fi

if lsof -nP -iTCP:18443 -sTCP:LISTEN | grep -q .; then
  echo "Port 18443 is already in use" >&2
  exit 1
fi
```

Expected result: the port check prints nothing and exits with status `0`. Identify an existing listener before changing it. This tutorial does not authorize stopping an unrelated process.

2. Prepare source-derived images

Run the repository preparation script:

```
SH
./deploy/connector-fleet/prepare.sh
test -s deploy/connector-fleet/.env.qualification
```

Expected result: preparation ends with `connector fleet preparation: PASS`. The owner-only environment file records the source digest, image tag, image IDs, connector release ID, and release revision selected by Compose.

Those values identify local fixture artifacts. They are not a production admission package or a Chatwoot credential.

Define a helper that always selects the source-owned file, environment, and product profile:

```
SH
compose() {
  docker compose \
    --profile products \
    --env-file deploy/connector-fleet/.env.qualification \
    -f deploy/connector-fleet/compose.yml \
    "$@"
}
```

Inspect the fixed project before starting it:

```
SH
compose ps --all
```

Stop here if the command reveals resources you need to retain. Do not run cleanup merely because the project already exists.

3. Start the Chatwoot path

Start the Chatwoot host and the dependencies declared by the Compose graph:

```
SH
compose up --detach --wait --wait-timeout 300 chatwoot-acme-a
```

Expected result: the command exits with status `0`. The dependency graph starts durable storage, identity and policy initialization, prerequisite admission jobs, the lifecycle control plane, the TLS edge, two product-neutral gateway replicas, the controlled product upstream, and `chatwoot-acme-a`.

Inspect one-shot and long-running services:

```
SH
compose ps --all
```

The Chatwoot host, both gateways, control plane, edge, PostgreSQL, and product fixture should be running and healthy. Initialization, migration, grant, and admission jobs may be exited with status `0`; that is their successful terminal state.

Do not bypass a failed admission job or start the host outside its declared identity. Preserve the failed job's bounded logs and correct its source-owned input.

4. Read the bound Chatwoot account

Run `aipctl` inside the isolated network with the generated client identity and trust material:

```
SH
compose run --rm --no-deps \
  --entrypoint /usr/local/bin/aipctl \
  -e AIPCTL_NATIVE_PRINCIPAL_ID=service:aipctl:qualification-acme \
  -e AIPCTL_NATIVE_TRUST_DOMAIN=fleet.test \
  -e AIPCTL_NATIVE_SIGNING_SEED_FILE=/fleet-state/secrets/client-acme-signing-seed.hex \
  -e AIPCTL_NATIVE_PEER_DID_FILE=/fleet-state/public/gateway.did \
  -e AIPCTL_NATIVE_TLS_CA_FILE=/caddy-data/caddy/pki/authorities/local/root.crt \
  qualification action call \
  https://aipd.fleet.test:8443 \
  cap:chatwoot:account.get \
  --action-id act_chatwoot_quickstart_account_001 \
  --input '{}'
```

Expected result: the response identifies `act_chatwoot_quickstart_account_001`, names `cap:chatwoot:account.get`, reports a completed result, and contains fixture account ID `42`.

The Action supplies no account identifier. The host inserts its configured account ID into `GET /api/v1/accounts/42` and adds the connector-owned API token. The result therefore demonstrates the local account binding without exposing the token or allowing the caller to choose another account.

5. Request a conversation

Create a private temporary directory for the tutorial responses:

```
SH
EVIDENCE_DIR=$(mktemp -d "${TMPDIR:-/tmp}/aip-chatwoot-quickstart.XXXXXX")
chmod 700 "$EVIDENCE_DIR"
printf 'Tutorial evidence: %s\n' "$EVIDENCE_DIR"
```

Keep one Action ID, one idempotency key, and one exact input for the entire mutation:

```
SH
ACTION_ID=act_chatwoot_quickstart_conversation_001
IDEMPOTENCY_KEY=chatwoot-quickstart-conversation-v1
INPUT='{"body":{"source_id":"quickstart-source","inbox_id":1,"contact_id":1}}'
PENDING="$EVIDENCE_DIR/conversation-pending.json"
```

Submit the conversation request with the source-owned Chatwoot caller identity:

```
SH
compose run --rm --no-deps \
  --entrypoint /usr/local/bin/aipctl \
  -e AIPCTL_NATIVE_PRINCIPAL_ID=service:aipctl:qualification-chatwoot \
  -e AIPCTL_NATIVE_TRUST_DOMAIN=fleet.test \
  -e AIPCTL_NATIVE_SIGNING_SEED_FILE=/fleet-state/secrets/client-chatwoot-acme-signing-seed.hex \
  -e AIPCTL_NATIVE_PEER_DID_FILE=/fleet-state/public/gateway.did \
  -e AIPCTL_NATIVE_TLS_CA_FILE=/caddy-data/caddy/pki/authorities/local/root.crt \
  qualification action call \
  https://aipd.fleet.test:8443 \
  cap:chatwoot:conversation.create \
  --action-id "$ACTION_ID" \
  --idempotency-key "$IDEMPOTENCY_KEY" \
  --input "$INPUT" >"$PENDING"

grep -q '"status": "pending_approval"' "$PENDING"
```

Expected result: the command exits successfully, but the Action status is `pending_approval`. No provider request is authorized at this point.

Do not change the input, Action ID, or idempotency key after this step. A Chatwoot mutation is not retry-safe, and the connector does not implement rollback or provider cancellation.

6. Approve the mutation

Extract and validate the approval identity and policy hash returned for this exact Action:

```
SH
approval_id=$(sed -n \
  's/^[[:space:]]*"approval_id": "\([^"]*\)",\{0,1\}$/\1/p' \
  "$PENDING" | head -n 1)
policy_hash=$(sed -n \
  's/^[[:space:]]*"policy_hash": "\([^"]*\)",\{0,1\}$/\1/p' \
  "$PENDING" | head -n 1)

case "$approval_id" in
  appr_[A-Za-z0-9_]*) ;;
  *) echo "Unexpected approval ID" >&2; exit 1 ;;
esac

test "${#policy_hash}" -eq 64
printf '%s' "$policy_hash" | grep -Eq '^[0-9a-f]{64}$'
```

Review `$PENDING` before approving. It identifies the exact Action, input snapshot, policy, and requested side effect. Stop if any value differs from the conversation request above.

Create one signed approval decision:

```
SH
decided_at=$(date -u +%Y-%m-%dT%H:%M:%SZ)
decision=$(printf \
  '{"approval_id":"%s","decision":"approved","approver":{"id":"human:qualification-
approver","kind":"human"},"decided_at":"%s","reason":"Approved for the Chatwoot quickstart fixture","c
onstraints":[],"evidence":[],"decision_id":"decision:%s:approved","policy_hash":"%s","authority_path":
[]}' \
  "$approval_id" "$decided_at" "$approval_id" "$policy_hash")

compose run --rm --no-deps \
  --entrypoint /usr/local/bin/aipctl \
  -e AIPCTL_NATIVE_PRINCIPAL_ID=human:qualification-approver \
  -e AIPCTL_NATIVE_TRUST_DOMAIN=fleet.test \
  -e AIPCTL_NATIVE_SIGNING_SEED_FILE=/fleet-state/secrets/approval-acme-signing-seed.hex \
  -e AIPCTL_NATIVE_PEER_DID_FILE=/fleet-state/public/gateway.did \
  -e AIPCTL_NATIVE_TLS_CA_FILE=/caddy-data/caddy/pki/authorities/local/root.crt \
  qualification approval decide \
  https://aipd.fleet.test:8443 \
  "$decision" >"$EVIDENCE_DIR/approval-decision.json"

grep -q '"kind": "aip.approval.granted"' \
  "$EVIDENCE_DIR/approval-decision.json"
grep -q '"kind": "aip.action.resumed_result"' \
  "$EVIDENCE_DIR/approval-decision.json"
```

Expected result: the signed decision grants the pending approval and resumes the same Action. Approval authorizes this fixture mutation; it does not make the operation retry-safe.

7. Read the completed conversation result

Repeat the same Action with the same input and idempotency key:

```
SH
COMPLETED="$EVIDENCE_DIR/conversation-completed.json"

compose run --rm --no-deps \
  --entrypoint /usr/local/bin/aipctl \
  -e AIPCTL_NATIVE_PRINCIPAL_ID=service:aipctl:qualification-chatwoot \
  -e AIPCTL_NATIVE_TRUST_DOMAIN=fleet.test \
  -e AIPCTL_NATIVE_SIGNING_SEED_FILE=/fleet-state/secrets/client-chatwoot-acme-signing-seed.hex \
  -e AIPCTL_NATIVE_PEER_DID_FILE=/fleet-state/public/gateway.did \
  -e AIPCTL_NATIVE_TLS_CA_FILE=/caddy-data/caddy/pki/authorities/local/root.crt \
  qualification action call \
  https://aipd.fleet.test:8443 \
  cap:chatwoot:conversation.create \
  --action-id "$ACTION_ID" \
  --idempotency-key "$IDEMPOTENCY_KEY" \
  --input "$INPUT" >"$COMPLETED"

grep -q '"status": "completed"' "$COMPLETED"
grep -q '"id": 9001' "$COMPLETED"
grep -q '"status": "open"' "$COMPLETED"
```

Expected result: the completed output contains the controlled conversation with ID `9001` and status `open`. Reusing both identities retrieves the durable result for the same intent. Do not generalize the fixture's fixed values to a real Chatwoot deployment.

If the transport fails after approval and the result remains unknown, do not change the Action ID or key and do not submit a new conversation. Preserve the response files and inspect durable Action state before deciding on recovery.

8. Verify the tutorial outcome

Require the Chatwoot host and both gateway replicas to remain running:

```
SH
test "$(compose ps --status running --services | \
  grep -cx 'chatwoot-acme-a')" -eq 1
test "$(compose ps --status running --services | \
  grep -Ec '^aipd(-b)?$')" -eq 2
```

Verify the three expected state transitions from the retained responses:

```
SH
grep -q '"status": "pending_approval"' "$PENDING"
grep -q '"kind": "aip.approval.granted"' \
  "$EVIDENCE_DIR/approval-decision.json"
grep -q '"status": "completed"' "$COMPLETED"
```

All checks must exit with status `0`. You have now completed one coherent learning path: an admitted standalone host fixed the provider account, served a read, held a mutation for approval, and returned one idempotent conversation result.

Diagnose before cleanup

If a step fails, capture bounded state before changing the project:

```
SH
compose ps --all
compose logs --no-color --tail 200 \
  aipd aipd-b control-plane chatwoot-acme-a product-upstream
```

Symptom	Check
Preparation cannot reach Docker	Repair the daemon, then repeat the source and project checks
Port `18443` is occupied	Identify the owner; do not terminate it without confirming scope
Admission exits nonzero	Read the failed job's bounded log; do not bypass its package or trust policy
Chatwoot host is unhealthy	Check its declared identity, account, database, token file, and fixture
Account read is rejected	Preserve the generated caller, signing key, peer DID, CA, and tenant
Conversation stays unapproved	Verify the returned approval ID and policy hash before deciding
Approval does not resume the Action	Preserve the pending and decision responses; do not create a replacement Action
Completed result is absent	Reuse the exact Action ID, key, and input; treat provider outcome as unknown

This tutorial does not authorize regenerating identities, editing credentials, weakening TLS, deleting the database during diagnosis, or starting the host manually outside registry admission.

Stop and remove the tutorial project

Review the temporary response files and the fixed Compose project first. Retain them if they are needed for an investigation.

When neither contains evidence or state you need, remove only the named project:

```
SH
compose down --volumes --remove-orphans --timeout 30
test -z "$(compose ps --all --quiet)"
```

The first command permanently deletes that project's PostgreSQL state, generated keys, and retained Compose evidence. It does not prune unrelated containers, images, volumes, or build cache.

Delete the separate temporary response directory only after deciding that it is no longer needed:

```
SH
case "$EVIDENCE_DIR" in
  "${TMPDIR:-/tmp}"/aip-chatwoot-quickstart.*)
    rm -rf -- "$EVIDENCE_DIR"
    ;;
  *)
    echo "Refusing unexpected evidence path: $EVIDENCE_DIR" >&2
    exit 1
    ;;
esac
```

What to do next

- Read the Chatwoot connector overview to choose the next

capability family and understand the one-account boundary.

- Use the connector fleet quickstart

for the shorter cross-connector learning path.

- Read Deploy the connector fleet

before replacing fixture identities with deployment-owned identities.

- Use connector fleet qualification

to evaluate failure, restart, isolation, and retained-evidence claims.

- Read conformance and qualification before

describing an artifact as conformant, qualified, or live verified.