Run the connector fleet quickstart

In this tutorial, you will start the repository's deterministic connector-fleet topology and route one signed AIP action from aipctl, through a product-neutral aipd gateway, to a standalone Cal.diy connector host. The provider is a controll

SECTION	Start Here
TYPE	Getting started
PROTOCOL	AIP 1.0
SOURCE	docs/getting-started/connector-fleet-quickstart.md

This is a learning path for the fleet process boundary. It is not a production deployment or a qualification run. The reviewed source is commit `97be86e9efedf07ecf1783b03800f683f107fb04`.

What you will observe

By the end of the tutorial, you will have observed:

- source-derived container image and connector release identities;
- registry admission before a connector host starts;
- two product-neutral gateway replicas and one lifecycle control plane;
- one healthy Cal.diy connector replica for `tenant-acme`;
- a signed call to `cap:cal_diy:profile.get` with status `completed`;
- durable action lookup through either gateway replica;
- bounded removal of only the named tutorial stack and its volumes.

The reference Compose file also defines deterministic hosts for Hermes Agent, Chatwoot, Dify, and CrewAI. This tutorial builds those images but starts only the Cal.diy product path. Twenty is one of the six maintained connectors, but it has no service in this deterministic fixture and follows its own deployment and qualification path.

Prerequisites

You need:

- a clean checkout at the reviewed source revision;
- Docker Engine with Docker Compose v2 and BuildKit;
- enough local CPU, memory, and disk for a source-based multi-image build;
- TCP port `18443` free on the loopback interface;
- a POSIX-compatible shell and `lsof` for the port preflight.

Run every command from the `aip-core` repository root. The first preparation can take substantially longer than the native quickstart because it builds the gateway, control plane, fixtures, and five product-host images.

The Compose project has the fixed name `aip-connector-fleet-qualification`. Do not use this tutorial while another run is using that project or while its volumes contain evidence you need to retain.

1. Verify the source and Docker boundary

Confirm the exact source revision, a clean worktree, and an available Docker daemon:

```
SH
export AIP_SOURCE_REVISION=97be86e9efedf07ecf1783b03800f683f107fb04

test "$(git rev-parse HEAD)" = "$AIP_SOURCE_REVISION"
test -z "$(git status --porcelain)"
docker info >/dev/null
docker compose version
```

All four checks must exit with status `0`. If the worktree is not clean, use a separate clean checkout instead of hiding changes: the preparation script derives image and release identities from the complete workspace content.

Check that the fixed host 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 command prints nothing and exits with status `0`.

2. Prepare source-derived images

Build the exact images and write their immutable image IDs to the generated Compose environment:

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

Expected result: the script ends with a line beginning `connector fleet preparation: PASS`. The generated file contains source, image, and release metadata for this workspace. At the reviewed revision it does not contain product credentials.

Define a short shell helper for the remaining commands:

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

Inspect the fixed Compose project before changing it:

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

If this lists containers from another investigation or qualification run, stop here and preserve that run first. This tutorial deliberately does not rename the project because the source-owned volume and evidence tooling use its fixed identity.

3. Start one product path

Start the Cal.diy host and its declared dependencies, and wait for running services to become healthy:

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

Expected result: the command exits with status `0`. Compose runs short-lived initializers and admission jobs before it reports the long-running services as healthy. The dependency graph starts:

- PostgreSQL and registry initialization;
- deterministic identity, policy, secret, and product fixtures;
- the connector lifecycle control plane;
- the TLS edge;
- `aipd` and `aipd-b` against shared durable state;
- the controlled product upstream;
- `cal-acme-a` as a standalone connector host.

The setup graph also creates qualification-only admission artifacts. They are internal test infrastructure, not additional public product connectors, and their host services are not part of this tutorial outcome.

Inspect both completed setup jobs and running services:

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

`cal-acme-a`, `aipd`, `aipd-b`, `control-plane`, `edge`, `postgres`, and `product-upstream` should be running and healthy. Initialization and admission services may be exited with status `0` because they are one-shot jobs.

4. Route one signed read

Run `aipctl` inside the isolated fleet network. The command mounts the source-generated client key, gateway DID, and local TLS authority from the Compose volumes:

```
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:cal_diy:profile.get \
  --action-id act_connector_fleet_quickstart_001 \
  --input '{}'
```

Expected result: the response is an AIP action result whose `action_id` is `act_connector_fleet_quickstart_001` and whose status is `completed`. The output contains the deterministic Cal.diy profile returned by the controlled upstream.

This completion demonstrates the configured local path: signed client request, gateway authentication, tenant-scoped capability routing, connector-host execution, and a product-shaped fixture response. It does not demonstrate behavior against a live Cal.diy deployment.

5. Read the action through the second gateway

Query the shared durable action state through `aipd-b` without replaying the provider operation:

```
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 \
  -e AIPCTL_NATIVE_BEARER_TOKEN_FILE=/fleet-state/secrets/native-bearer-token \
```

```
  qualification action status \
  https://aipd-b.fleet.test:8443 \
  act_connector_fleet_quickstart_001 \
  --include-result \
  --include-receipts
```

Expected result: the lifecycle view identifies the same action and capability, reports a terminal state with `result_status` set to `completed`, and includes the stored result. A successful lookup shows that the two gateway replicas use the same durable runtime state; it is not a failover or recovery test.

6. Verify the tutorial outcome

Repeat a non-mutating status lookup and require both the connector host and gateways to remain healthy:

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

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 \
  -e AIPCTL_NATIVE_BEARER_TOKEN_FILE=/fleet-state/secrets/native-bearer-token \
  qualification action status \
  https://aipd.fleet.test:8443 \
  act_connector_fleet_quickstart_001 \
  --include-result >/dev/null
```

All three commands must exit with status `0`. You have now observed admission, registration, signed routing, standalone execution, and shared durable lookup for one low-risk product capability.

Stop and clean up

Before deletion, capture diagnostics if any step failed:

```
SH
compose ps --all
compose logs --no-color --tail 200 aipd aipd-b control-plane cal-acme-a
```

When no retained state from this named project is needed, remove its containers, network, and volumes:

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

The first command deletes the project volumes, including PostgreSQL state, generated keys, and any qualification evidence stored there. It does not prune unrelated containers, images, build cache, or volumes. Keep the images if you intend to repeat the tutorial; remove them only through a separately reviewed cleanup decision.

Resolve common failures

Symptom	Decision
Preparation reports that Docker is unavailable	Start or repair the Docker daemon, then rerun the same source check before preparation
Port `18443` is already in use	Identify the owning process or existing fleet run; do not stop or delete it without confirming ownership
An image build fails	Preserve the build output, confirm disk and tool availability, and rerun `prepare.sh` from the same clean revision
A one-shot admission job exits nonzero	Inspect `compose ps --all` and that job's logs; do not bypass admission or start the host manually
A service does not become healthy	Inspect the named service, its dependency state, and bounded logs before cleanup
The signed action is rejected	Confirm that the command uses the generated principal, key, peer DID, CA, and `tenant-acme` fixture unchanged
The action completes but the second lookup fails	Preserve gateway and PostgreSQL logs; this tutorial does not authorize recreating shared state as a repair

What happened internally

The fixture generated trust material and immutable admission packages, migrated the connector registry, and admitted the Cal.diy connector version before its host process started. The host then registered a tenant-scoped replica and renewed its lease with the lifecycle control plane.

The signed `aipctl` request reached the product-neutral gateway. The gateway selected an eligible Cal.diy replica from the registry and sent the action to the standalone host. That host resolved its fixture credential, called the controlled upstream, and returned a typed result. Both gateways used the same PostgreSQL-backed runtime state, so the second gateway could read the completed action without repeating the product call.

The repository's full product-fleet qualification exercises all five fixture hosts, governed mutations, tenant isolation, durable replay, outages, restarts, and failover. Those checks belong to the qualification documentation and are intentionally outside this quickstart.

Next steps

- Understand connector-fleet architecture
- Adopt the Cal.diy connector
- Deploy a connector fleet
- Run connector-fleet qualification
- Follow the separate Twenty path