Run the Cal.diy connector quickstart

In this tutorial, you will start the repository's deterministic Cal.diy fleet path and route one signed cap:caldiy:profile.get action from aipctl, through product-neutral aipd, to the standalone aip-host-cal-diy process.

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

The provider is a controlled local fixture. You do not need a Cal.diy account or credential, and the procedure does not call an external Cal.diy 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 Cal.diy host starts;
- a healthy account-specific host registered for one tenant;
- one signed, low-risk Cal.diy profile read with a completed result;
- the same durable action state through a second `aipd` replica;
- cleanup limited to the named tutorial Compose project.

You will not test a provider mutation, approval, webhook, restart, failover, external service, or 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 and `lsof`.

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

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` and the generated environment file contains the exact local source, image, and connector release identities used by Compose.

The file contains fixture identities, not a production admission package or a Cal.diy 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 Cal.diy path

Start the Cal.diy host and the dependencies declared by the Compose graph:

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

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

Inspect one-shot and long-running services:

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

The Cal.diy host, both gateways, control plane, edge, PostgreSQL, and product fixture should be running and healthy. Initialization 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 logs and correct the source-owned input.

4. Call the Cal.diy profile capability

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:cal_diy:profile.get \
  --action-id act_cal_diy_quickstart_001 \
  --input '{}'
```

Expected result: the response identifies `act_cal_diy_quickstart_001`, names `cap:cal_diy:profile.get`, and reports a completed action result. Its output is the deterministic profile returned by the controlled upstream.

This proves the configured local path through signature verification, tenant routing, standalone host execution, and result persistence. It does not prove that a real Cal.diy deployment accepts the same credential or payload.

5. Read the durable result through the second gateway

Query the stored action through `aipd-b` without repeating the provider call:

```
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_cal_diy_quickstart_001 \
  --include-result \
  --include-receipts
```

Expected result: the lifecycle view contains the same action and capability, reports a terminal completed result, and includes the stored output. This demonstrates shared durable lookup; it is not a gateway failover test.

6. Verify the tutorial outcome

Require the named product host and both gateway replicas to remain running:

```
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
```

Repeat a non-mutating status lookup through the first gateway:

```
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.fleet.test:8443 \
  act_cal_diy_quickstart_001 \
  --include-result >/dev/null
```

All commands must exit with status `0`. You have now completed the tutorial's single learning outcome: one signed Cal.diy read crossed the admitted standalone-host boundary and remained durably observable.

Diagnose before cleanup

If any step fails, capture bounded state first:

```
SH
compose ps --all
compose logs --no-color --tail 200 \
  aipd aipd-b control-plane cal-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 that job's bounded log; do not bypass the package or trust policy
Cal.diy host is unhealthy	Check its declared dependencies, identity, database, and provider fixture
Signed action is rejected	Keep the generated principal, key, peer DID, CA, and tenant unchanged
Second lookup fails	Preserve both gateway and PostgreSQL logs before changing shared state

This tutorial does not authorize regenerating identities, deleting the database, weakening TLS, or replacing admission with manual host startup as a repair.

Stop and remove the tutorial project

Only after confirming that the fixed project contains no evidence or state you must retain, remove its containers, network, and volumes:

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

The first command permanently deletes the named project's PostgreSQL state, generated keys, and retained evidence. It does not prune unrelated containers, images, volumes, or build cache. Any broader cleanup requires a separate scope decision.

What to do next

- Read the Cal.diy connector overview for supported families and safety boundaries.
- Use the fleet quickstart to understand the cross-connector learning path.
- Follow Deploy the connector fleet only after production admission material and identities exist.
- Use connector fleet qualification for failure, restart, isolation, and evidence claims.
- Read the historical isolated-live report before interpreting external Cal.diy evidence.