Run the Dify connector quickstart

In this tutorial, you will start the repository's deterministic Dify path and route two signed reads from aipctl, through product-neutral aipd, to the standalone aip-host-dify process.

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

The provider is a controlled local fixture. The procedure does not require or contact an external Dify deployment. It 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 image and connector-release identities;
- registry admission before the Dify host starts;
- one `chat` app and one separately credentialed Knowledge boundary;
- signed `parameters.get` and `dataset.list` Actions;
- the completed app Action through the second gateway's durable status route;
- cleanup limited to the named tutorial Compose project.

You will not test app invocation, mutation, approval, streaming, cancellation, provider failure, restart, gateway failover, or a real Dify deployment.

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`, and `grep`;
- authority to remove the named tutorial project after reviewing its state.

Run every command from the `aip-core` repository root. The products profile builds all product hosts even though this tutorial starts only the Dify path.

The Compose project is `aip-connector-fleet-qualification`. Stop before startup if another investigation or retained run owns that project or its volumes.

1. Verify source and port ownership

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)"
```

Use a separate clean checkout if either check fails. Do not hide or discard local changes to satisfy the tutorial.

Confirm the required tools and the fixed edge port:

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

command -v lsof >/dev/null
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. Identify an existing listener before changing it; this tutorial does not authorize stopping another process.

2. Prepare the controlled artifacts

Run the source-owned 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 local source digest, image IDs, release ID, and release revision selected by Compose.

Define one helper that always selects the source-owned profile:

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

compose ps --all
```

Stop if the project contains state that must be retained. Do not remove it merely because its name already exists.

3. Start the Dify path

Start the Dify host and its declared dependency graph:

```
SH
compose up --detach --wait --wait-timeout 300 dify-acme-a
compose ps --all
```

Expected result: `dify-acme-a`, both gateways, the control plane, edge, PostgreSQL, and the product fixture are healthy. Initialization, migration, grant, and admission jobs may be exited with status `0`.

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

4. Read application parameters

Route one signed read through the generated client identity:

```
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:dify:qualification-app:parameters.get \
  --action-id act_dify_quickstart_parameters_001 \
  --input '{}'
```

Expected result: the signed response names the Action and capability, reports a completed result, and contains the deterministic parameter payload.

This capability uses the application-specific fixture key. It does not expose the key or let the caller choose a different app.

5. Read the Knowledge catalogue

Use a second Action for the separately credentialed Knowledge API:

```
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:dify:knowledge:qualification-kb:dataset.list \
  --action-id act_dify_quickstart_datasets_001 \
  --input '{"query":{"limit":1}}'
```

Expected result: the response reports a completed Action and contains the fixture dataset listing. The selected Knowledge credential is independent of the application key used in the previous step.

6. Verify durable Action state

Read the first Action through the second gateway without repeating its provider request:

```
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_dify_quickstart_parameters_001 \
  --include-result
```

Expected result: the lifecycle view contains the same terminal Action and stored output. This demonstrates shared durable lookup, not gateway failover.

Diagnose before cleanup

Capture bounded state before changing the project:

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

Symptom	First check
Preparation cannot reach Docker	Repair the daemon, then repeat 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 policy
Dify host is unhealthy	Check descriptor files, key files, database, and fixture reachability
App read is unauthorized	Verify the app-key file path without printing its value
Knowledge read is unauthorized	Verify the separate Knowledge-key file path
Second lookup fails	Preserve both gateway and PostgreSQL logs before changing shared state

The tutorial does not authorize regenerating identities, weakening TLS, printing credentials, deleting the database, or bypassing admission.

Stop and remove the tutorial project

After confirming that the fixed project contains no state or evidence that must be retained, remove its containers, network, and volumes:

```
SH
compose down --volumes --remove-orphans
```

Review `compose ps --all` before this command. Volume removal is intentionally destructive and belongs only to the named disposable tutorial project.

Related documentation

- Dify connector overview
- AIP quickstart
- Deploy the connector fleet
- Errors and retry decisions
- Connector upstream baselines