Qualify the Dify connector

Use this page to decide whether one exact Dify connector artifact supports a bounded release claim. Keep source tests, image checks, controlled fixtures, and actual Dify observations as separate evidence classes.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/dify/qualification/README.md

Current exact-artifact result: not established by this page. The reviewed source defines qualification gates, but this documentation review did not run them.

No retained isolated-live Dify result for the reviewed standalone host was found in the pinned source tree.

Use evidence terms consistently

Term	Required evidence
Implemented	Behavior and contract exist in identified source
Conformant	Exact artifact passed applicable conformance cases with retained results
Controlled-qualified	Exact artifact passed a named deterministic topology and failure matrix
Isolated-live verified	Exact artifact produced reviewed observations against identified Dify artifacts
Production-observed	Deployment records show behavior under a stated production boundary

A test function proves that a gate exists. It does not prove that the gate ran or passed for the artifact under review.

Bind one immutable identity set

Every result must bind identities that can change its meaning:

Identity class	Required values
AIP source	Commit, tree, dirty-source snapshot when applicable, and status
Connector artifact	Immutable image ID, component, version, labels, entrypoint, and architecture
Contract	Manifest digest, descriptor hashes, operation asset, schemas, and implementation support
Dify upstream	Commit, tree or image, migrations, feature configuration, and API mode
Harness	Source digest, runner image, configuration digest, and selected cases
Runtime	Engine, databases, trust policy, descriptors, credentials, and network topology
Run	UTC interval, unique run ID, evaluator, and evidence-root digest

The reviewed AIP commit is `97be86e9efedf07ecf1783b03800f683f107fb04`, with Git tree `2d83a382a312c601ecbbf5ed03f8af2b0218bdae`.

The reviewed Dify source commit is `f8d47616c15d0959f604f6a5e2e1d32d3108991b`, with Git tree `253bd2797b388544d8343d9cb8cd62185abce795`.

These identities do not identify a built image or completed run. A mutable tag, branch, filename, or documentation date is not artifact identity.

Evaluate four independent gates

Gate	Source-owned entry point	What it can prove	What it cannot prove
Q1: connector contracts	Focused tests in `aip-connector-dify`	Deterministic connector semantics through Rust boundaries	Container, fleet, or external Dify behavior
Q2: standalone image	`verify-product-images.sh`	Image identity, user, entrypoint, labels, inspect, history, and SBOM	Provider compatibility, signature, vulnerability policy, or admission
Q3: controlled product fleet	Product qualification and fleet verification scripts	Admission, selected operations, policy, isolation, failure closure, and recovery against fixtures	Actual Dify compatibility or complete catalogue behavior
Q4: isolated live	Reviewed campaign manifest and harness	Selected behavior against identified Dify artifacts	Arbitrary deployments, every capability, scale, or production readiness

Record every gate required by the claim. A deterministic fixture cannot substitute for actual Dify, and a provider observation cannot substitute for host admission or failure recovery.

Review focused connector evidence

The pinned Dify connector crate defines 18 focused tests. Together they cover:

- completeness, uniqueness, governance, and mode projection of 79 operations;
- distinction between stop commands and cancellable runs;
- all 46 Knowledge capabilities in a manifest;
- canonical path construction and bounded query encoding;
- request-size and query-value limits before dispatch;
- scoped Knowledge mutation, key, path, body, and idempotency projection;
- multipart upload fields and decoded-file bounds;
- bounded base64 projection for Knowledge binary output;
- fail-closed mutation behavior without an idempotency key;
- typed retry and uncertainty mapping;
- incremental native SSE publication;
- mode-specific provider stop and cancelled terminal state;
- cleanup after a malformed admitted stream;
- durable task recovery across restart;
- unconfirmed cancellation without durable task identity;
- completion-family routing and request-field preservation;
- CRLF parsing, missing event rejection, and frame bounds.

Retain the exact test selection, build identity, exit status, timestamps, structured results, and redacted diagnostics. Source definitions alone are implementation evidence.

These tests do not establish Dify version compatibility, container packaging, fleet admission, database recovery, or live provider behavior.

Review the standalone image gate

The image verifier builds `aip-host-dify` with the common host Dockerfile. It requires:

- an immutable SHA-256 image ID;
- runtime user `10001:10001`;
- entrypoint `/usr/local/bin/aip-connector-host`;
- exact source revision, version, and component labels;
- a successful read-only, capability-dropped `--help` invocation;
- retained image inspect and full history;
- an SPDX JSON SBOM.

Review Dify-specific output rather than relying on a shared summary. An SBOM is inventory, not a signature, vulnerability decision, license decision, or provider qualification.

Review the controlled product-fleet gate

The controlled fixture configures one chat app and one Knowledge credential. It admits exactly 74 Dify capabilities:

```
TEXT
28 chat-app capabilities + 46 Knowledge capabilities
```

The selected positive cases are:

Case	Capability	Required evidence
App read	`cap:dify:qualification-app:parameters.get`	Completed signed Action and provider GET
Knowledge read	`cap:dify:knowledge:qualification-kb:dataset.list`	Completed signed Action with bounded query
Knowledge mutation	`cap:dify:knowledge:qualification-kb:dataset.create`	Pending approval, separate decision, resumed completion, stable key

The provider fixture audit requires authorization on Dify requests and an idempotency marker on the dataset mutation. The fleet also checks the admitted capability count and an independently owned migrated runtime database.

Shared failure cases cover graceful host restart, `SIGKILL` and expired-lease fail-closed behavior, gateway failover, PostgreSQL outage, route recovery, and final ready leases.

This gate does not exercise a real Dify instance, app alias execution, streaming, cancellation, binary data, uploads, indexing, pipelines, audio, annotations, or most operation suffixes.

Label its result controlled product-fleet qualification, not live Dify compatibility.

Define isolated-live qualification

An isolated-live campaign needs a dedicated Dify deployment, disposable apps and Knowledge data, separate requester and approver identities, scoped keys, and explicit cleanup.

Select cases by unique behavior:

1. health for every app and Knowledge credential;
2. exact manifest projection for all configured modes;
3. one safe app read and one safe Knowledge read;

4. blocking completion, chat, workflow, and agent-family aliases as configured;
5. incremental SSE with terminal result and ordered chunks;
6. confirmed cancellation after task-ID capture;
7. cancellation before task-ID capture with preserved uncertainty;
8. binary preview, audio, and document archive envelopes;
9. bounded multipart app and Knowledge uploads;
10. async document indexing with provider batch reconciliation;
11. segment update with embedded failure detection;
12. multi-document metadata partial-commit handling;
13. pipeline and node runs through the frozen buffered contract;
14. authentication, provider rejection, rate limit, malformed SSE, and bounds;
15. restart, task-store recovery, lease fencing, and storage outage.

Record provider resources before and after every mutation. One successful HTTP exchange cannot establish provider effect, Action result, stream terminal state, and cleanup together.

Account for every admitted capability

Generate one coverage row per capability in the admitted manifest. The row count depends on descriptors and modes; 74 is only the controlled fixture.

Field	Required value
Capability ID	Exact admitted identifier
Descriptor binding	App or Knowledge ID, mode, and credential revision
Contract evidence	Route, schemas, risk, approval, key, retry, and response mapping
Deterministic evidence	Test and result artifact, or explicit gap
Controlled-fleet evidence	Case and artifact, or `not_run`
Isolated-live evidence	Provider identity, case, artifact, or `not_run`
Failure evidence	Auth, validation, rejection, uncertainty, bounds, and recovery cases
Side-effect control	Approval, key, provider-state check, and cleanup where applicable
Review decision	Accepted scope, limitation, owner, reviewer, and time

Every row needs contract evidence. Live mutation of every writable capability is unsafe and unnecessary by default.

Select reversible live cases by distinct transport and state behavior. Publish all unobserved capabilities as explicit gaps.

Retain a complete evidence set

A qualification record should contain:

1. immutable source, image, manifest, schema, Dify, harness, topology, and run

identities;

2. a gate manifest naming every case and invariant;
3. raw exits, structured results, and redacted diagnostics;
4. admission, catalogue, implementation-support, route, and lease snapshots;

5. image inspect, history, SBOM, and separate supply-chain decisions;
6. provider request, task, workflow, document, and batch correlations;
7. durable Action, approval, idempotency, stream, and cancellation evidence;
8. failure and recovery ordering for process, lease, storage, and provider;
9. a generated admitted-capability coverage matrix;
10. checksums for every artifact and one evidence-root checksum;
11. verified cleanup, residual-resource inventory, and retained exceptions;
12. a summary stating scope, result, exclusions, reviewer, and UTC time.

A summary is not a pass by itself. Review every referenced artifact and ensure later recovery did not hide an earlier failed invariant.

Assign an unambiguous result

Result	Use when
`passed`	Every required invariant passed for the exact identity set and gaps match the claim
`failed`	An invariant failed, identity mismatched, evidence leaked, or cleanup failed
`blocked`	A required dependency or authority was unavailable before evaluation
`not_run`	The gate was not executed for this identity set
`historical`	Retained evidence belongs to another source, artifact, topology, or standard

Do not collapse `blocked`, `not_run`, or `historical` into `passed`. One gate can pass while a broader release claim remains unestablished.

Publish only the supported claim

Acceptable wording binds result and exclusions:

> At the recorded time, the identified AIP and Dify connector artifacts passed
>
> the listed deterministic and controlled-fleet cases in the retained
>
> topology. The coverage matrix identifies unobserved capabilities and
>
> excludes live-provider and production properties.

Do not publish `fully compatible`, `all capabilities live verified`, `exactly once`, or `production-ready` unless the retained campaign defines and proves each phrase.

Reviewer checklist

- [ ] Source, image, manifest, Dify, harness, topology, and run identities are

immutable and mutually consistent.

- [ ] Required gates are explicit and fixture results are not labeled live.
- [ ] Coverage rows match the exact admitted manifest, not a universal count.
- [ ] Focused, image, and controlled-fleet artifacts are retained and redacted.
- [ ] Mutation evidence binds approval, key, provider effect, and uncertainty.
- [ ] Stream evidence binds ordered chunks, terminal state, and cleanup.

- [ ] Durable task evidence covers restart and missing-identity uncertainty.
- [ ] Image inventory is not presented as signature, scan, or provenance.
- [ ] Cleanup and residual resources are independently recorded.
- [ ] Final wording states time, scope, result, gaps, and exclusions.

Related documentation

- Conformance and qualification
- Connector fleet qualification
- Live-product qualification
- Dify capability index
- Release artifacts and verification