Dify connector

The Dify connector exposes configured Dify applications and workspace Knowledge APIs as governed AIP capabilities. Use it when AIP must invoke published apps, manage conversations or files, operate knowledge data, stream workflow events, or

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

The connector is not a generic pass-through. Its source pins provider routes, credentials, request encoding, user placement, response transport, risk, approval, retry, and size limits.

Decide whether it fits

Requirement	Fit
Invoke a configured workflow, completion, chat, advanced-chat, agent-chat, or agent app	Yes, through one app invocation alias
Use frozen application Service API operations	Yes, when the operation supports the configured app mode
Manage workspace knowledge data	Yes, through a separate Knowledge API credential
Stream app or workflow output	Yes, for the app alias and two catalogue SSE routes
Cancel an active app execution	Yes, after task identity is known or supplied safely
Retry provider mutations automatically	No
Use AIP transactions, reconciliation, or rollback	No
Select a Dify URL or API key from Action input	No
Prove compatibility with any Dify deployment	No; qualification belongs to an exact artifact and upstream boundary

Choose another integration boundary when arbitrary provider routes, dynamic credentials from callers, or automatic rollback are required.

Understand the deployment boundary

One standalone `aip-host-dify` process loads deployment-owned descriptors and owner-only key files. It constructs a manifest, receives signed routed Actions, calls one configured Dify origin, and publishes signed AIP results.

The base URL, app set, knowledge set, credentials, connector identity, artifact, manifest, tenant, and registry assignment are outside Action input. The host uses runtime profile state to retain the Action-to-task mapping needed for cancellation across process replacement within one logical instance.

This topology separates four authorities:

Authority	Owns
AIP tenant and policy	Discovery, approval, routing, retry budget, and Action lifecycle
Dify host instance	Static descriptors, bounded keys, task correlation, and provider transport

Authority	Owns
Dify application	Published app behavior, conversations, messages, files, and remote tasks
Dify workspace	Knowledge datasets, documents, chunks, metadata, models, and pipelines

Keep credential domains separate

Application Service API keys and workspace Knowledge API keys are different credential classes.

Credential	Capability prefix	Health probe	Scope
Per-app API key	`cap:dify:<app_id>`	`GET /v1/parameters`	One configured published application
Knowledge API key	`cap:dify:knowledge:<credential_id>`	`GET /v1/datasets?limit=1`	One configured workspace knowledge boundary

Every app should use its own key file. A legacy global app-key file is accepted only when all listed applications intentionally share that key. Knowledge credentials always name their own files.

No capability advertises frozen connector credential-handle routing at this revision. Isolate hosts and key files by tenant and intended provider scope.

Choose an app mode deliberately

Configured mode	Invocation route	Approval on the app alias
`workflow`	`POST /v1/workflows/run`	Required
`completion`	`POST /v1/completion-messages`	Not required by the connector contract
`chat`	`POST /v1/chat-messages`	Not required by the connector contract
`advanced-chat`	`POST /v1/chat-messages`	Required
`agent-chat`	`POST /v1/chat-messages`	Required
`agent`	`POST /v1/chat-messages`	Required

The mode also controls which app-scoped catalogue operations are discoverable. An unsupported mode fails connector construction instead of falling back to a different provider family.

Plain chat and completion aliases still perform external model work and require a tenant-scoped idempotency key. The absence of connector-required approval is not a claim that every deployment should authorize them without policy.

Read capability IDs by domain

The manifest is configuration-dependent:

Shape	Meaning	Example
`cap:dify:<app_id>`	Invoke the configured app in its declared mode	`cap:dify:support-chat`
`cap:dify:<app_id>:<operation>`	Use a supported app Service API operation	`cap:dify:support-chat:conversation.list`
`cap:dify:knowledge:<credential_id>:<operation>`	Use a workspace Knowledge API operation	`cap:dify:knowledge:main:dataset.list`

The frozen catalogue contains 79 provider operations: 33 app-scoped and 46 Knowledge-scoped. One app publishes only the operations compatible with its mode, plus its invocation alias. Each Knowledge credential publishes all 46 Knowledge operations.

There is no universal Dify manifest count. The controlled fixture happens to publish 74 capabilities from one `chat` app and one Knowledge credential. That number is not a general deployment invariant.

Apply the execution controls

Surface	Approval	Idempotency	Retry	Streaming	Cancellation
App invocation alias	Workflow and agent-family aliases	Required	Disabled	Yes	Yes
Catalogue `GET`	No	Optional	Published safe	Only `workflow.event.stream`	Operation-specific
Catalogue mutation	Required	Required	Disabled	`workflow.run_by_id` only	Operation-specific
Knowledge mutation	Required	Required	Disabled	No	No

All keys use tenant scope, input-hash revalidation, and a 24-hour contract TTL. The connector does not publish transaction, reconciliation, or provider rollback support. Approval expires after 15 minutes where the connector requires it.

A transport category or temporary provider response never overrides these contracts. Read durable Action and provider state before replacing an unsafe operation.

Preserve end-user isolation

Every app invocation requires a non-empty `user`. Operations that Dify binds to an end user also require it in the exact query, JSON, or multipart location declared by the catalogue. The connector rejects a conflicting `user` supplied inside another input partition.

Use a stable, non-secret, tenant-scoped external value. Reusing one value across unrelated users can merge Dify conversation identity even when AIP Actions are otherwise isolated.

Knowledge operations do not inject an app end-user value. Their isolation comes from the selected workspace credential and AIP tenant route.

Treat streams and cancellation as durable work

The app alias requests Dify streaming mode and converts bounded SSE events to ordered AIP chunks. It retains the remote task ID when Dify emits it, requires an explicit terminal event, and maps human-input events to `requires_human`.

Each SSE frame is limited to 1 MiB, a stream to 10,000 events, and the complete response to the configured byte bound. Provider close without a terminal event is a failure, not successful completion.

Cancellation uses the retained app ID, task ID, and end-user value. Dropping an HTTP response before task identity is known is only cooperative cancellation; it does not prove that remote work stopped. Preserve the original Action and reconcile provider state before starting replacement work.

Respect request and response bounds

Boundary	Source limit
JSON request body	8 MiB
Decoded upload file	32 MiB
Default response or complete stream	8 MiB
Maximum configurable response	64 MiB
SSE frame	1 MiB
SSE events per stream	10,000

Binary outputs are returned as bounded base64 with content type and byte size. They are not written to a caller-selected filesystem path by the connector.

Know what is not guaranteed

The reviewed source does not establish:

- live compatibility with every Dify edition or deployment;
- automatic provider-state reconciliation after an uncertain mutation;
- exactly-once external effects;
- transactional grouping across app and Knowledge operations;
- provider rollback or AIP compensation;
- durable provider runs after Dify-side retention or restart;
- production qualification for the current artifact.

The source pins upstream commit `f8d47616c15d0959f604f6a5e2e1d32d3108991b`. A pin records the reviewed mapping baseline; it is not an identity for a deployed Dify image.

Choose the next path

For an existing V1 reader, use the concise legacy page for the previous high-level contract. Read the AIP execution model before integrating Actions, use the fleet guide for the standalone topology, and use the global error reference before implementing retries. The upstream-baseline page explains how provider identities are bounded.

Related documentation

- Legacy Dify connector overview
- How AIP works
- Deploy the connector fleet
- Errors and retry decisions
- Connector upstream baselines