Profiles, transports, and connectors

Use this page to choose the correct integration boundary without creating a second action model or bypassing AIP governance. It is for application, gateway, and connector developers who need to distinguish protocol semantics, wire delivery,

SECTION	Core Concepts
TYPE	Concept
PROTOCOL	AIP 1.0
SOURCE	docs/concepts/profiles-and-connectors.md

AIP has one native semantic model. A transport moves that model, a compatibility profile projects another protocol onto it, and a connector maps an AIP capability to a product. A connector can run inside a trusted process or behind a separately deployed connector host. Placement changes the trust and failure boundary, but it does not change the meaning of the action.

This page describes the reviewed Rust implementation of AIP 1.0 at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. It explains architecture, not production qualification. The maintained public connector set in this documentation is Cal.diy, Hermes Agent, Chatwoot, Dify, CrewAI, and Twenty.

Keep four integration layers separate

The layers compose, but none substitutes for another.

Layer	Question it answers	What it owns	What it does not own
AIP semantics	What work is requested, governed, and observed?	Envelopes, actions, capabilities, identity claims, lifecycle, approvals, transactions, errors, and evidence	A provider API or network connection
Transport binding	How does a native AIP envelope move between peers?	Framing, encoding, connection behavior, delivery metadata, and transport errors	Capability authorization or provider behavior
Compatibility profile	How does another protocol project onto AIP?	Foreign data-transfer objects (DTOs), deterministic mappings, method and error projection, and compatibility metadata	A parallel action store or product credentials
Product connector	How does a product perform the requested capability?	Provider DTOs, authentication at the provider edge, invocation, error normalization, and supported recovery behavior	Global identity, policy, approval, or action lifecycle

Process placement is a separate decision. The same product connector contract can be registered as a trusted local handler or hosted behind the connector fleet boundary. Treating placement as a fifth protocol layer would confuse a deployment choice with wire semantics.

A profile identifier is a declaration

`ProfileId` is a general namespace used across discovery and bindings. A participant manifest advertises supported profile identifiers. A manifest request and its filter can select them, and an individual capability binding can attach profile-specific metadata.

The namespace is broader than “compatibility protocol.” The reviewed source uses profile identifiers for all of these purposes:

- native transport bindings, such as `aip.native.nats.v1`;
- streaming bindings, such as `aip.sse.stream.v1` and `aip.websocket.stream.v1`;
- compatibility mappings, such as `aip.mcp.compat.v1` and `aip.a2a.compat.v1`;
- the generic signed-webhook contract `aip.http.webhook.v1`;
- connector-specific capability bindings.

An advertised identifier says that the publisher declares compatibility with that namespace. It does not prove that an endpoint is reachable, a route has been admitted, the peer is authenticated, the caller is authorized, or a live replica is ready. Those checks belong to transport, identity, registry, and runtime boundaries.

Native transports preserve the envelope

The shared transport abstraction carries a native `Envelope` with normalized metadata. Request/reply and streaming behavior are separate traits, so a transport implementation does not imply support for every interaction pattern.

Binding	Native behavior in the reviewed source	Appropriate use
HTTP	Encodes an envelope as JSON with `application/aip+json`; recommends message, action, and manifest paths; maps protocol error categories to HTTP status classes	Common service ingress and request/reply
NATS	Publishes or requests native envelopes on structured AIP subjects, with optional queue groups and correlation streams	Brokered service routing and workers
SSE	Encodes native envelopes as one-way events, preserves event identifiers for reconnection, and marks results, errors, and final chunks as terminal	Following events or streamed output over HTTP
WebSocket	Encodes envelopes as text frames and provides session queues, correlation subscriptions, ping/pong, and close behavior	Authenticated bidirectional multiplexing

Changing the binding does not change an action ID, capability contract, identity rules, approval state, transaction state, or result meaning. Transport authentication also does not grant capability authority. The gateway must still establish a trusted actor and apply policy before execution.

The transport crates provide reusable framing and state. A concrete server must still expose routes, authenticate connections, apply limits, and connect frames to the gateway. The product-neutral daemon supplies those server boundaries for its configured HTTP, SSE, WebSocket, and NATS surfaces.

Compatibility profiles translate at the edge

A compatibility profile lets a client use an established protocol while AIP remains the internal semantic and lifecycle authority.

The MCP profile owns MCP DTOs and deterministic mappings between tool, resource, task, progress, error, and AIP concepts. It deliberately contains no HTTP server, subprocess, or runtime state. Those responsibilities stay in the MCP server, client, and transport components.

The A2A profile similarly maps Agent Cards, messages, tasks, status updates, streaming, cancellation, and push-notification concepts. The signed-webhook profile is narrower: it defines normalized webhook headers and

verifies an HMAC-SHA256 signature with a timestamp-skew boundary. A product connector still has to interpret the verified provider event.

A profile adapter should therefore perform three bounded steps:

1. validate the foreign request under that protocol's rules;
2. create or address the corresponding native AIP operation;
3. project the native result, status, stream, resource, or error back to the

foreign protocol.

The native action remains authoritative throughout the second step. A profile must not create a separate approval engine, transaction store, or connector credential path merely because the foreign protocol names those concepts differently.

A connector implements product behavior

The base connector contract identifies a connector, discovers its manifest, maps failures, and reports health. Additional traits make supported behavior explicit: capability discovery, outbound invocation, channel ingress and egress, external-event ingestion, escalation, and typed transaction or recovery operations.

The production typed boundary, `FrozenConnector`, receives a complete trusted execution context. It can implement normal invocation, planning, commit, compensation, cancellation, reconciliation, emission, and ingestion. Every optional operation fails as unsupported unless the connector overrides it. That default prevents an advertised capability from silently acquiring safety properties that its provider does not implement.

A product connector owns only its product edge. Typical responsibilities are:

- converting a capability input into provider requests;
- loading credential material from connector-owned process storage;
- pinning the correct provider account from trusted execution context;
- normalizing provider responses and failures into AIP results;
- retaining provider operation references needed for reconciliation;
- verifying product webhooks before mapping them to AIP envelopes;
- declaring implementation support that matches the admitted artifact.

The connector does not trust caller-supplied tenant or credential values. It also does not decide the global approval policy, rewrite the AIP lifecycle, or prove its own qualification merely by publishing a manifest.

Choose local or remote placement deliberately

The reviewed implementation supports two materially different placements.

Placement	Execution path	Trust and operations boundary
Trusted local module	The gateway discovers the connector manifest, admits callable capabilities with local handlers, and invokes the connector with runtime-established context	Connector code, credentials, failures, and resource use share the gateway process boundary
Remote connector fleet	The central runtime resolves a tenant-scoped route, sends a signed native AIP action to a registered host, and verifies the signed response	Provider credentials and connector code stay in a separately leased, observable, and drainable host process

The typed local path wraps each admitted non-resource capability in a handler. The handler rejects calls without trusted execution context and selects the connector operation from the action transaction mode. Local placement reduces network and control-plane machinery, but it also increases the blast radius of connector faults and may

place provider secrets in the central process.

The remote path adds registry, route, identity, lease, capacity, and callback boundaries. That machinery supports independent rollout and isolation, but it also requires durable control-plane state and another authenticated hop. Remote placement is not inherently more qualified; its exact artifact and topology still need evidence.

Follow the complete request path

A native client enters at the transport boundary. An MCP or A2A client first passes through its compatibility adapter. Both paths converge on the same central action lifecycle before choosing local or remote execution.

```
MERMAID
flowchart LR
    F["Foreign-protocol client"] --> P["Compatibility profile"]
    N["Native AIP client"] --> T["Native transport"]
    T --> G["Central gateway: authenticate and enrich identity"]
    P --> G
    G --> R["Runtime: validate, authorize, approve, and persist"]
    R --> D{"Connector placement"}
    D -->|"Local"| L["Trusted local handler"]
    D -->|"Fleet"| A["Tenant binding and durable route assignment"]
    A --> H["Signed native AIP request to connector host"]
    H --> V["Host verifies gateway, route, lease, revision, and capacity"]
    L --> C["Product connector"]
    V --> C
    C --> X["Provider API or event surface"]
    X --> C
    C --> O["AIP result, chunk, or event"]
    O --> R
    R --> T
```

In text, the flow is:

1. A native binding decodes an AIP envelope, or a compatibility adapter maps a

foreign request into the native model.

2. The central gateway authenticates the transport actor and resolves trusted

tenant and credential context.

3. The runtime validates the capability and schema, evaluates policy and

approval, establishes transaction and idempotency state, and persists the action lifecycle.

4. A local capability selects its admitted local handler. A fleet capability

instead selects an enabled tenant binding and persists an action-scoped route assignment before an external side effect.

5. The remote dispatcher signs the envelope for the exact assigned host. The

host verifies the gateway, recipient, tenant, instance, replica, version, manifest, capability, credential revision, lease, and local capacity.

6. The product connector invokes the provider under the trusted context. Its

typed result or failure returns to the central runtime, which remains the client-facing lifecycle owner.

This flow is an ownership map, not a guarantee that every deployment enables every stage. Durable stores, authenticators, policies, admitted artifacts, bindings, hosts, and provider credentials must all be configured independently.

Read fleet identities from stable to ephemeral

Remote routing uses several records so that a product name, tenant account, running process, and one action are not collapsed into a single “connector.”

Record	Stable meaning	Routing role
Connector type	One implementation family and owner	Allows or blocks admission of versions and instances
Connector version	One immutable artifact, manifest, digest, attestation set, and implementation-support map	Defines the exact admitted code and contract
Connector instance	One tenant-owned configured installation on a selected version	Names the logical provider account and secret-provider reference
Connector replica	One live host process for an instance and version	Advertises a control-plane-selected endpoint, peer identity, topology, lease, health, and capacity
Capability binding	One verified tenant and capability mapped to an enabled instance	Applies priority and policy, credential-revision, and quota references
Route assignment	One immutable action-scoped execution target	Pins instance, replica, endpoint, peer, topology, version, digests, policy, credential revision, health, and fencing data

New route selection considers enabled bindings and instances, admitted active versions, ready unexpired replicas, capacity, policy, circuit state, and topology. Once persisted, the assignment is reused for that action's retries, cancellation, and reconciliation. A retry does not silently select a different provider account or host.

Split ownership across the fleet boundary

Component	Owns	Must not be treated as owning
Central gateway and runtime	Client authentication, trusted identity enrichment, policy, approval, idempotency, transaction state, action lifecycle, and original callback	Provider secrets or host process health
Registry and admission data	Connector types and versions, artifact evidence, tenant instances and bindings, replicas, leases, health, capacity, and durable routes	Provider invocation or client identity authentication
Connector control plane	Signed host registration, heartbeat, drain, and offline transitions	Catalog migration, artifact admission, or product execution
Connector host	One logical instance and immutable version, provider credentials, local durability, connector health, lease renewal, and signed native AIP ingress	Global tenant selection or caller-supplied identity
Product connector	Provider mapping, product calls, provider errors, and supported recovery behavior	Central action governance or fleet routing policy

The host serves health, readiness, metrics, its immutable manifest, and native message ingress. Readiness requires a valid lease, a non-draining state, available durable storage, and connector readiness. For new actions, the host also enforces local capacity and the route coordinates pinned by the central assignment.

Return results through the central owner

A normal remote invocation returns a signed `ActionResult` directly to the central dispatcher. The dispatcher verifies that the returned action ID and peer match the assignment before the runtime settles the action.

For a streaming action, the remote dispatcher removes arbitrary client callbacks and supplies only the configured central connector-callback endpoint with signed route metadata. The shared central ingress validates the sender, replica, instance, version, manifest, lease, and durable assignment before it publishes a chunk into runtime state. The central runtime retains the original client callback.

Provider-originated events use a separate fixed connector-event ingress. A host can durably enqueue and sign those events, but event ingress does not make the event an action result. Stream callbacks and provider events have different correlation and authorization rules.

Fleet routing is not delegation

Remote connector execution sends an ordinary action to an implementation selected for a tenant capability. It does not create a parent-child action edge or transfer work to another agent principal under a `DelegationRequest`.

Use delegation when one participant explicitly asks another participant to execute a child action and the graph relationship matters. Use connector fleet routing when the central runtime is selecting an implementation of the same governed action. The two paths may both use signed native AIP envelopes, but their identities, records, and lifecycle meanings remain distinct.

Choose the extension point by responsibility

Need	Correct extension point	Reason
Move native envelopes over a new network mechanism	Transport binding	Framing and delivery change; action meaning does not
Serve MCP, A2A, or another general protocol	Compatibility profile plus its edge server	Foreign methods and objects require deterministic projection into the native model
Invoke or receive events from one product	Product connector	Provider credentials, DTOs, side effects, and errors are product-specific
Isolate connector code, secrets, rollout, or capacity	Remote connector host and fleet records	Process placement and routing need explicit trust and lifecycle controls
Ask another AIP participant to execute a child action	Delegation route	The operation creates a first-class parent-child graph edge

Do not introduce a compatibility profile merely because one provider has unusual endpoints. Do not place a general protocol mapping inside one product connector. Do not introduce a new transport to encode product-specific business semantics.

Understand the trade-offs and limits

- Native bindings preserve AIP concepts directly; compatibility profiles make

existing ecosystems accessible but require careful loss and error mapping.

- Local connectors have fewer moving parts; remote hosts isolate secrets and

failures but add routing, lease, signing, and recovery dependencies.

- Manifest advertisement supports discovery; registry admission and live

readiness provide stronger, still bounded evidence.

- A capability binding expresses tenant policy; an action-scoped route freezes

the exact execution target before side effects.

- A signed remote response authenticates a configured peer under the reviewed

verification path; it does not by itself prove provider-side behavior.

This page does not claim that all transports support all message patterns, that every advertised profile has a live endpoint, or that a connector manifest proves conformance. It also does not claim production readiness for any of the six maintained product connectors. Those conclusions require exact-artifact, topology, provider, and retained-evidence review.

Related concepts

- Read Capabilities and contracts for the semantic unit a

connector implements.

- Read Identity and trust for the actor, tenant, credential, and route authority boundaries.
- Read Delegation before routing work to another AIP participant.
- Use Build a connector for the implementation procedure.
- See Connector registry and routing for the detailed fleet data model and selection algorithm.