Architecture overview

Use this page to understand which component owns each decision and state transition in the reviewed AIP implementation. It is for developers, connector developers, operators, and security reviewers who need a complete component map before c

SECTION	Architecture and Security
TYPE	Architecture
PROTOCOL	AIP 1.0
SOURCE	docs/architecture/overview.md

The architecture describes version `2.0.0` of the Rust workspace at source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. AIP 1.0 defines interoperable message and lifecycle meaning; it does not require this Rust crate layout, process topology, database, or orchestration model.

Why ownership boundaries matter

An AIP action can cross transport decoding, authentication, policy, durable lifecycle state, tenant routing, a connector process, and an external product. Treating that path as one undifferentiated service obscures who may establish identity, where a secret may exist, which store survives a restart, and which component can safely retry an uncertain operation.

The reviewed implementation keeps those concerns separate. The semantic core does not depend on networking or products. Compatibility profiles do not own native action state. The target fleet composition keeps the six maintained product adapters and their provider credentials outside the primary daemon. Connector lifecycle control cannot admit a new artifact, and the registry does not execute an image or call a provider.

The six-boundary model

Boundary	Primary responsibility	Authority it does not gain
Semantic contract	Native IDs, envelopes, message and domain objects, and pure validation	Network identity, persistence, product behavior, or deployment policy
Ingress and projection	Native transport framing plus bounded MCP, A2A, and webhook mappings	Permission to invoke a capability or redefine native lifecycle meaning
Central execution	Gateway authentication and policy composition plus runtime lifecycle, queues, approvals, transactions, and observation	Raw provider credentials or external product truth
Connector fleet data path	Tenant catalog, immutable route assignment, signed remote dispatch, host execution, and event return	Connector release admission or arbitrary target selection by payload
Fleet release and lifecycle control	Signed admission, registry administration, host leases, and platform-neutral orchestration	Client action execution, provider mutation, or secret resolution
Persistence and evidence	Runtime records, registry rows, host-local durable state, provider records, and retained artifact evidence	A broader claim than the identity, topology, and procedure of each record

These are ownership boundaries, not mandatory process counts. A small trusted deployment can use a local module and file-backed state. A clustered connector fleet uses separate central, registry, lifecycle, host, and provider boundaries.

Component map

The solid arrows below show action and result flow. Dashed arrows show release, deployment, or host-lifecycle control that is intentionally outside the action path.

```
MERMAID
flowchart LR
    N["Native AIP client"] --> T["Native transport edge"]
    F["MCP or A2A client"] --> P["Compatibility profile"]
    T --> G["Gateway"]
    P --> G
    G --> R["Durable runtime"]
    R --> L["Trusted local module"]
    R --> D["Tenant catalog and remote handler"]
    D --> H["Standalone connector host"]
    H --> X["External product"]
    R --> S["Runtime stores"]
    D --> Q["Connector registry"]
    H -.->|register, heartbeat, drain| C["Lifecycle control plane"]
    C -.->|lease and status| Q
    A["Signed connector admission"] -.->|atomic catalog change| Q
    O["External orchestrator"] -.->|exact admitted process| H
```

The text equivalent is:

1. A native transport decodes an AIP envelope, or a compatibility profile maps

a foreign request into native AIP.

2. The gateway establishes the authenticated actor, resolves trusted identity

and tenant context, validates the message boundary, and applies policy.

3. The runtime owns action governance and durable lifecycle state.
4. A trusted local capability executes through a startup-admitted local module.

A fleet capability instead uses the tenant catalog and shared remote handler.

5. The remote handler persists an action-specific route before dispatching a

signed native request to the assigned connector host.

6. The host verifies the pinned route and its lease, resolves provider

credentials inside its process, and invokes the external product.

7. Results, chunks, callbacks, or events return to the central lifecycle owner;

authorized readers inspect durable state independently of the original connection.

8. Separately, signed admission changes catalog state, an external orchestrator

starts exact admitted replicas, and the lifecycle service manages host registration, heartbeat, drain, and offline transitions.

Follow one action through the component owners

Enter through one semantic path

Native HTTP, NATS, SSE, and WebSocket bindings carry AIP data with different delivery behavior. MCP and A2A clients enter through compatibility mappings. Both routes converge on native capabilities, actions, results, and lifecycle state rather than creating parallel business semantics.

Transport authentication establishes a peer or actor only under the configured edge policy. Payload sender and tenant fields remain claims until trusted resolvers bind them to deployment-owned identity, membership, and credential evidence.

Govern and persist centrally

The gateway composes validation, replay protection, identity resolution, authorization, and dispatch. The runtime owns the durable action record and coordinates schema checks, capability support, idempotency, approval, transaction state, cancellation, callbacks, events, receipts, retention, and recovery.

The product-neutral daemon installs fleet discovery and remote execution as one pair: it cannot publish a remote capability without a handler that can route it. Trusted local modules are frozen into startup composition. Long-tail product connectors use the remote fleet boundary and do not add product code to the central daemon.

Select local or fleet execution

A local handler runs inside the trusted central process and receives runtime-established execution context. This path has fewer distributed dependencies but shares the central process, resource, fault, and credential boundary.

A fleet action uses verified tenant context to resolve an enabled capability binding and an eligible connector instance and replica. The resulting route assignment pins the version, endpoint, peer identity, manifest digest, catalog and policy revisions, credential revision, capacity reservation, and fence token before the provider side effect. Retry, cancellation, and reconciliation reuse that assignment rather than silently selecting another product account or host.

Execute at the product boundary

Each standalone host serves one logical connector instance and one immutable connector version. It verifies signed central ingress, recipient and route coordinates, lease and health state, credential revision, approval evidence, and local capacity before executing its compiled product adapter.

Provider secrets remain inside the connector-host boundary. The connector maps trusted AIP context to provider requests, normalizes results and typed failures, and retains provider operation evidence needed for reconciliation. The external product remains authoritative for its own committed state.

Keep control paths outside action execution

Three control paths prepare or maintain the fleet without becoming an action handler:

Control path	Input and decision	Output	Explicit exclusion
Connector admission	A signed package, exact artifact and manifest digests, mandatory independently trusted evidence, limits, freshness, and revocation policy	Atomic registry change plus durable journal result	Does not start an image or call a provider
Deployment orchestration	Verified admission package, signed desired intent, exact observed replica snapshot, generation, and rollout bounds	Deterministic signed start, restart, drain, or stop plan for an external executor	Does not receive registry administrator credentials or invent replica identities
Host lifecycle	Signed and replay-fenced registration, heartbeat, drain, and offline requests from an admitted replica	Monotonic health sequence, bounded lease, and routing status	Does not migrate the registry or admit a connector release

This separation limits authority. A compromised host cannot self-admit a new version. A lifecycle service cannot change the signed release package. An orchestrator can run only the pre-provisioned identities and immutable artifact digests present in a verified plan.

Place state with the component that can interpret it

State owner	Durable facts	Recovery boundary
Central runtime stores	Actions, sessions, approvals, transactions, queues, events, chunks, callbacks, receipts, replay claims, profile state, and retention metadata	Central workers resume lifecycle work under fenced leases and idempotency state
Connector registry	Types, versions, capability definitions, instances, bindings, policies, replicas, health and leases, assignments, settlements, catalog revision, and admission journal	New routing and host lifecycle resume from indexed durable rows
Connector-host runtime	Host-side action state, provider checkpoints, webhook replay, event outbox, and connector-owned profile state	The same immutable host identity recovers local work before declaring readiness
External product	Product objects, provider operation status, and committed side effects	Reconciliation queries provider truth after an ambiguous response
Release and evidence stores	Exact images or artifacts, signatures, software bill of materials, provenance, scan results, and conformance reports	Admission re-verifies identity, digest, validity, and revocation before publication

The implementation offers in-memory, file-backed, and PostgreSQL-backed central runtime stores. Production connector-host bootstrap requires PostgreSQL-backed runtime state. The protocol object alone does not choose or qualify a storage backend.

Failure ownership follows state ownership

Failure	Component that detects or records it first	Safe architectural response
Client disconnect	Transport or profile edge	Read the authorized durable lifecycle instead of inferring failure from the connection
Central process restart	Runtime supervisor and selected store	Recover fenced queued, approval, callback, reconciliation, and retention work from durable state
Registry or lifecycle outage	Remote handler, fleet supervisor, or connector host	Stop new unsafe routing or lease renewal; preserve existing assignments and evidence
Connector-host loss	Lease maintenance and central remote dispatch	Mark the replica unavailable, preserve the pinned action route, and reconcile uncertain provider state
Provider timeout after a mutation	Connector and runtime transaction state	Record outcome unknown and reconcile; do not choose a new replica or repeat blindly
Stale release or deployment control	Admission or orchestration verifier	Reject by signature, digest, revision, generation, snapshot, expiry, or revocation fence

Readiness, reachability, authorization, and provider outcome are independent. A healthy central daemon does not prove that a connector lease or external product is healthy. A ready host does not authorize a tenant action.

Trust and data cross specific boundaries

Boundary	Trusted fact established there	Data kept out
Client edge to gateway	Authenticated actor and transport context	Self-asserted authority from payload fields
Gateway to runtime	Verified tenant, resolved identity, credential handle, and policy context	Raw provider credentials
Runtime to registry	Tenant-scoped capability and route request	Caller-selected instance, replica, endpoint, or secret
Central daemon to connector host	Signed envelope pinned to assigned peer, tenant, instance, replica, version, and manifest	Registry administrator authority
Connector host to provider	Product-authenticated request using host-owned secret material	Secret values in AIP messages, errors, metrics, or central storage
Release authority to admission	Signed package and separately trusted evidence statements	Mutable image tags or self-asserted scan and conformance success

No single signature establishes every row. Transport identity, tenant membership, capability authorization, artifact admission, route identity, provider credentials, and outcome evidence have separate owners.

Design choices and trade-offs

- **A small semantic core supports several implementations.** Keeping async,

transport, cryptography, storage, and products outside the core improves reuse, but requires explicit composition at the gateway and deployment edge.

- **Durable lifecycle replaces connection-based inference.** Recovery and

authorized observation survive disconnects, but require stores, leases, retention, and operational supervision.

- **Profiles preserve client compatibility.** MCP and A2A mappings reduce

integration cost, but expose only their defined projection of native AIP.

- **Standalone hosts isolate product code and secrets.** Independent rollout

and failure containment add registry, signed peer, lease, storage, and orchestration responsibilities.

- **Immutable routing favors safety over opportunistic failover.** Reusing one

action assignment preserves account and evidence identity, but an uncertain mutation may require reconciliation instead of immediate rerouting.

What this architecture does not imply

- AIP 1.0 does not require Rust, `getaip-server`, PostgreSQL, NATS, or one particular

process count.

- A workspace dependency or implemented code path does not prove that a

deployment enables, configures, conforms to, or has qualified that path.

- A compatibility profile is not an alternative native lifecycle.
- A connector manifest or ready process cannot self-authorize a tenant route.
- A signed action, admitted artifact, durable record, or receipt does not by

itself prove the external product outcome.

- A control-plane component does not become a provider action executor merely

because both use the connector registry.

Related pages

- How AIP works
- Connector fleet architecture
- Gateway
- Runtime
- Security model