Connector fleet architecture

Use this page to understand how the reviewed AIP implementation can add, isolate, and scale connector processes without adding product code or provider secrets to the central aipd binary. It is primarily for developers designing the fleet b

SECTION	Architecture and Security
TYPE	Architecture
PROTOCOL	AIP 1.0
SOURCE	docs/architecture/connector-fleet.md

This architecture describes version `1.0.0` of the Rust workspace at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. It is an implementation topology, not a process topology required by the AIP 1.0 protocol.

Why the fleet is a separate boundary

A bundled connector shares the central daemon's release, resource, fault, and credential boundary. That can be useful for a small trusted local module, but it becomes difficult to manage when connector versions, product accounts, rollout schedules, and failure domains grow independently.

The fleet moves product adapters into standalone connector hosts. The primary `aipd` package depends on product-neutral connector contracts, registry access, and remote dispatch, but not on the six maintained product adapters. Adding or scaling an already admitted connector therefore changes catalog, process, and lease state rather than the `aipd` executable.

This separation provides three practical properties:

- a product adapter and its secrets stay inside its connector-host process;
- a connector release can be admitted, started, drained, or replaced without

changing the central binary;

- central routing can apply tenant, capability, version, health, topology, and

capacity constraints before a provider request leaves the platform.

It does not remove distributed-systems work. The fleet introduces durable routing, signed peer identity, leases, host-local recovery, release admission, and external process orchestration as explicit responsibilities.

Six planes with different authority

The implementation separates the connector fleet into six cooperating planes. The solid arrows in the diagram carry discovery, action, result, or event data. Dashed arrows carry release, deployment, or lifecycle control.

```
MERMAID
flowchart LR
    C["AIP client"] --> D["Product-neutral aipd"]
    D -->|catalog and route| R["Connector registry data plane"]
    D -->|signed action| H["Standalone connector host"]
    H -->|provider request| P["External product"]
    H -->|result, callback, or event| D
    A["Admission authority"] -.->|publish admitted state| R
    O["External orchestrator"] -.->|start exact replica| H
```

```
    H -.->|register and renew lease| L["Lifecycle control plane"]
    L -.->|update lifecycle state| R
```

The text equivalent is:

1. The central daemon discovers tenant-visible capabilities through the

registry data plane.

2. For a fleet action, the daemon resolves and persists one route, then sends a

signed native AIP request to the selected standalone host.

3. The host verifies the route and central peer, uses its process-owned provider

credentials, and calls the external product.

4. The result returns on the action path. Stream callbacks and normalized

provider events return through authenticated central ingress.

5. Separately, admission publishes trusted catalog state, an external

orchestrator starts only declared replicas, and the lifecycle service registers, renews, drains, or offlines those replicas.

Plane	Owns	Deliberately does not own
Central data plane	Tenant-scoped discovery, action governance, fair local admission, route resolution, signed dispatch, callbacks, and event ingress	Product implementation code, provider secrets, catalog administration, or process creation
Connector-host data plane	One admitted connector instance and version, provider execution, local durable recovery, webhook normalization, and signed responses	Selection of another tenant, instance, version, or replica
Registry data plane	Indexed catalog reads, leases, durable action assignments, capacity state, and settlement	Provider credentials, connector execution, or artifact startup
Lifecycle control plane	Signed registration, heartbeat, drain, and offline transitions through a restricted registry role	Catalog migration, release admission, client execution, or provider mutation
Release admission	Verification and atomic publication of one immutable release package and its independently trusted evidence	Process startup, host lifecycle, client execution, or provider mutation
External orchestration	Reconciliation of exact admitted process identities and artifact digests on a deployment platform	Registry administrator credentials or invention of connector identities

The central daemon installs its capability catalog and catch-all remote handler as one pair. It cannot advertise a remote capability without installing the execution path that can resolve and dispatch it.

Keep the fleet identity chain explicit

The word *connector* is too broad to identify a safe route. The fleet uses a chain of identities, each answering a different question.

Identity	Question it answers	Stability
Connector type	Which adapter family implements this product boundary?	Stable across its releases
Connector version	Which admitted artifact and manifest implement it?	Artifact and manifest identity remain immutable; lifecycle status can change
Connector instance	Which tenant-owned product account or installation is configured?	Logical deployment identity
Connector replica	Which pre-provisioned process identity may serve the instance?	Concrete lifecycle and lease identity
Connector host	Which running process enforces that replica's signed, storage, and secret boundary?	One instance and one version per process

Identity	Question it answers	Stability
Route assignment	Which exact replica is pinned to this action under the accepted catalog and policy state?	Durable for the action

A capability binding connects a verified tenant and capability to an eligible instance. A route assignment then connects one action to one replica. Payload data and transport code do not choose the tenant, product account, endpoint, or replica.

Keeping these identities separate allows several versions of one connector type, several tenant-owned instances of a version, and several replicas of an instance to coexist without treating them as interchangeable.

Follow one fleet action

Discover only callable tenant bindings

The tenant-scoped catalog exposes capabilities that have an enabled binding and an implementation admitted for the selected profile. Discovery does not prove that a provider is reachable or authorize a later action by itself. The action still enters normal gateway and runtime governance.

Admit work before selecting a host

The shared remote scheduler applies bounded global and per-tenant concurrency, queue, request-size, and queue-age limits. Tenant weights affect local fairness; they do not grant capability or routing authority.

After local admission, the remote handler asks the registry to resolve the action from runtime-established tenant and capability context. The registry either returns the action's existing assignment or creates one from eligible catalog, policy, lease, topology, and capacity state.

Persist, then dispatch

The route exists durably before the connector can perform a provider side effect. The assignment pins the peer, tenant, instance, replica, immutable version, manifest, policy revisions, credential revision, capacity reservation, and fence state. The dispatcher signs the native AIP request for that assigned peer and carries the route coordinates needed for host verification.

The host rejects a request when the signed sender or pinned coordinates do not match its own identity. It also rejects the request when its lease or readiness is invalid, when it is draining, or when local capacity, credential revision, or approval evidence is not acceptable. Only then does the product adapter receive trusted execution context and access its provider credential.

Preserve the route after uncertainty

Successful, cancelled, and uncertain outcomes settle against the same assignment. A retry or cancellation reuses the action's durable route rather than selecting another account or replica. If a provider may have committed a mutation before a response was lost, reconciliation determines provider truth; the architecture does not promise exactly-once external effects.

Return results, callbacks, and provider events safely

The normal action result returns from the assigned host to the central runtime. A streaming host can send signed chunks through a callback route derived from the same durable assignment. Central ingress checks the callback against the assignment and current replica identity before accepting it.

Provider-originated events take a different route. The host verifies the provider webhook, normalizes the event, stores it in a bounded durable outbox, and publishes it to shared authenticated event ingress. Central AIP owns the protocol-level actor after authenticating the host; provider actor information is retained as data rather than trusted as an AIP sender claim.

An active host lease is required for event publication. A transient central failure can leave an accepted event batch queued for retry, and central event storage deduplicates repeated event identifiers. This path is not a substitute for action-route authorization.

Change scale without changing identity meaning

Fleet growth happens at different layers. Each change has a different safety boundary.

Change	State that changes	State that remains fixed
Add a connector type	Admitted type, version, manifest, implementation claims, and release evidence	Central product-neutral executable and native AIP semantics
Add a tenant account	Instance, capability bindings, policy, and credential revision references	Connector artifact and provider secrets outside the registry
Add capacity	Pre-provisioned replica identity, placement, endpoint, lease, and capacity	Instance ownership and immutable version
Roll out a version	New admitted version, target replicas, binding or rollout state, and deployment generation	Existing action assignments and retained release evidence
Drain capacity	Replica lifecycle state and eventual process observation	Pinned work and its original assignment

The external orchestration contract derives a deterministic, signed sequence of start, restart, drain, and stop operations from a verified admission package, operator intent, and an exact platform observation. It is platform-neutral: the executor maps those operations to its deployment system and checks fresh state before applying them.

A host serves exactly one logical instance and one immutable version. Scaling therefore creates additional declared replica processes; it does not turn one host into a multi-tenant product router.

Keep lifecycle authority narrow

Connector hosts register, heartbeat, drain, and go offline through a standalone signed lifecycle service. That service uses a restricted registry role and must not retain catalog-administration credentials. Schema migration and release admission remain operator-controlled actions outside the long-running service.

The lease establishes current routing eligibility, not release trust. Admission establishes release trust, not current host health. Orchestration establishes which exact process should run, not whether a client action is authorized.

This separation prevents one authority from silently acquiring another:

- a running host cannot admit a replacement artifact;
- the lifecycle service cannot create a tenant binding or change a release;
- the central daemon cannot start arbitrary images;
- an orchestrator cannot mint unregistered replica identities;
- a client cannot choose a product account through request metadata.

Isolate state and secrets by failure domain

Boundary	Durable or sensitive state	Effect of failure
Central runtime	Action lifecycle, approvals, transactions, callbacks, events, replay, and recovery records	Authorized lifecycle observation and recovery may pause, but connector identity does not change
Connector registry	Catalog state, bindings, replicas, leases, assignments, capacity, settlement, and admission journal	New discovery or routing may stop; existing assignments remain the identity for recovery
Connector host	Provider credential access, local action checkpoints, webhook replay, event outbox, and provider evidence	One instance and version lose execution capacity; other hosts need not share its secret or fault
External product	Product objects and committed side effects	Provider truth may require reconciliation after an ambiguous response
Release and deployment systems	Signed packages, evidence, desired generations, platform observations, and execution journal	New rollout work stops without granting action-path authority

The registry stores opaque credential and secret-provider references, not provider credential values. The host deployment resolves the referenced secret inside the host boundary. Rotation can advance accepted revisions and drain old work without exposing the secret to the central daemon.

Contain failures instead of silently rerouting them

Failure	Containment behavior
No eligible instance or replica	Reject new routing; do not infer a target from client data
Scheduler or capacity limit reached	Queue within configured bounds or fail explicitly; do not bypass tenant fairness
Registry unavailable	Stop unsafe new selection and preserve already durable assignments for recovery
Lease expires or host drains	Remove the replica from new assignments while allowing policy-governed pinned work to finish
Host disappears after a provider call	Preserve the route and reconcile the uncertain outcome instead of selecting a new provider account
Lifecycle service unavailable	Stop lifecycle renewal or change; do not convert the central daemon into a lifecycle administrator
Admission or orchestration evidence is stale	Reject the release or plan and require a fresh verified decision

Readiness is local to a component. A ready `aipd` does not prove registry, connector-host, or provider readiness. A ready host does not prove tenant authorization, and a healthy provider connection does not establish artifact admission.

Trade-offs and non-implications

- **Isolation adds coordination.** Separate hosts reduce product-code and

credential blast radius but require signed identities, durable routing, leases, host storage, and process supervision.

- **Pinned routing favors evidence integrity.** It prevents silent account or

replica changes but can require reconciliation rather than immediate failover after an uncertain mutation.

- **Normalized state supports bounded lookup.** Indexed records allow the

registry to resolve a large fleet without materializing one fleet document, but catalog publication and schema ownership become explicit operations.

- **Platform-neutral orchestration limits authority.** The contract can be

implemented by different schedulers, but each executor must provide its own platform observation, journal, and secret injection.

This implementation does not imply that every AIP deployment needs a connector fleet, PostgreSQL, separate processes, or an external orchestrator. Source code for a fleet path does not prove that a deployment enabled or qualified it. The architecture also makes no portable throughput, availability, provider-health, or exactly-once guarantee.

Related pages

- Connector registry and routing
- Connector admission and supply-chain trust
- Deploy the connector fleet
- Operate the connector host lifecycle
- Orchestrate and roll back connector hosts