Connector registry and routing

Use this page to understand how the reviewed AIP connector registry turns an authenticated tenant and capability into one durable execution target. It explains which records participate, how discovery remains revision-consistent, how a new

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

The model describes version `1.0.0` of the Rust workspace at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. These registry and PostgreSQL choices are implementation architecture, not storage requirements imposed by AIP 1.0.

Why routing needs durable registry state

An action target is more than a URL. It combines a verified tenant, a callable capability, a tenant-owned product account, an admitted connector artifact, a live host identity, policy state, credential revision, and reserved capacity. Recomputing that combination after a timeout could select a different account or process after the first provider already committed a side effect.

The registry therefore performs two different jobs:

- it publishes a revisioned, tenant-scoped capability catalog; and
- it atomically creates or returns one persistent route for each action.

The route is evidence as well as dispatch state. It records which destination and policy snapshot governed the first attempt, then remains available after capacity is released.

Separate the contract from the durable backend

The product-neutral registry crate defines five narrow contracts:

Contract	Consumer	Responsibility
Capability catalog	Runtime, discovery, and compatibility profiles	Resolve one definition or return a bounded tenant-scoped page
Action target resolver	Shared remote handler	Create, read, and settle one durable route assignment
Registry reader	Host admission plus callback and event ingress	Read version, instance, and replica identities
Registry administrator	Admission, lifecycle, and operator components	Change catalog state or replica leases through backend-enforced roles
Fleet status provider	Daemon supervision and observation	Expire bounded stale leases and return fixed-size aggregate state

The same crate includes a deterministic in-memory implementation for embedded deployments and conformance work. The fleet backend uses normalized PostgreSQL tables and indexed columns. It does not materialize the fleet as one JSON document and does not store provider credential values.

The PostgreSQL implementation can hold two independently permissioned pools:

- the **control pool** installs schema and changes catalog or admission state;
- the **data pool** serves catalog reads, route resolution, reservations,

replica lifecycle updates, and aggregate status.

A data-plane-only process verifies that an external control plane installed the expected schema, then retains no control pool. Pool names describe authority, not separate databases: both can connect to the same database through different roles and bounds.

Read the normalized entity graph

```
MERMAID
flowchart LR
    T["Connector type"] --> V["Immutable connector version"]
    V --> C["Capability definitions"]
    V --> I["Tenant-owned instance"]
    C --> B["Capability binding"]
    I --> B
    I --> R["Live replica lease"]
    P["Admission policy"] --> B
    B --> A["Durable action assignment"]
    R --> A
```

The text equivalent is:

1. A connector type identifies an implementation family and owner.
2. An admitted version binds that type to one immutable artifact, manifest,

capability set, supported profiles, and supply-chain attestation.

3. A connector instance selects a version for one tenant-owned installation

and points to non-secret configuration and secret-provider revisions.

4. A capability binding makes one tenant and capability eligible to use one

instance under a priority, policy revision, credential revision, and optional quota policy. Gateway authorization still applies to each action.

5. A replica is one pre-provisioned host identity for the instance and version,

with endpoint, signed peer identity, topology, health sequence, lease, and capacity.

6. A route assignment joins the binding and replica to one action and retains

the decision after execution capacity is released.

Know what may change

Record	Fixed identity or content	Controlled mutable state
Connector type	ID, name, and owner	Enabled flag
Connector version	ID, type, artifact, manifest, attestation, capability contracts, and profiles	Lifecycle status can move through admitted, active, or revoked; revoked cannot reactivate
Capability definition	Capability ID, complete contract digest, and schema digest	A conflicting contract under the same ID is rejected
Connector instance	ID, connector type, and tenant	Version, status, secret reference, and configuration only through a newer configuration revision
Capability binding	Tenant, capability, and instance key	Priority, enabled state, credential and quota references only through a newer policy revision
Connector replica	ID, instance, version, endpoint, peer identity, trust domain, transport, and topology identity	Status, lease, capacity, and health only through a newer health revision; active assignments remain registry-owned
Route assignment	Complete action destination and policy snapshot	Reservation and settlement bookkeeping; the assignment record itself remains pinned

Instance and binding changes advance the catalog revision. Replica heartbeat, lease, and capacity churn do not. This keeps a catalog page stable without serializing every host heartbeat through the catalog publication lock.

Publish one catalog revision at a time

Catalog administrators take an exclusive database-scoped publication lock, validate all referenced records, apply one transaction, increment the monotonic catalog revision, and commit. A failed transaction does not publish a partial view.

Catalog readers take the shared form of that lock while reading the revision and materializing a result. A tenant-scoped read returns a capability only when all of these conditions hold:

- the tenant has an enabled binding for the capability;
- the bound instance belongs to the same tenant and is enabled;
- the selected version is active and supply-chain qualified;
- the connector type is enabled; and
- the version implements the capability and, when requested, the profile.

Internal administrative reads can explicitly request active unbound definitions. An ordinary read with neither verified tenant context nor that internal permission returns no capabilities.

Keep pagination tied to the revision

Catalog pages are ordered by capability ID and bounded by both item count and serialized bytes. The page carries the revision used for the complete read. A continuation cursor combines that revision with the last capability ID.

If a later request presents the cursor after the catalog revision changes, the registry returns `StaleCursor`. The caller restarts at the first page instead of combining definitions from two publication snapshots. Search text, exact capability, profile, tenant, page-size, definition-size, and total-page-size limits are evaluated server-side.

Resolve a new action in one transaction

The route request contains only an action ID, capability ID, verified tenant ID, and deployment-owned topology preference. It contains no instance, replica, endpoint, credential value, or peer key.

The PostgreSQL resolver follows this sequence:

1. Validate that the tenant ID is present.

2. Serialize concurrent decisions for the same action with an action-scoped advisory lock.
3. Return an existing assignment when its tenant and capability match. Reject the request as a conflict when those coordinates differ.
4. For a new action, take a revision-consistent catalog view and establish that at least one enabled tenant binding exists.
5. Select an eligible replica under a row lock, skipping rows currently locked by other routing transactions and retrying contention only within a bounded limit.
6. Build and reserve the binding's hard admission scopes, increment the selected replica's active assignment count, create a fresh fence token, and insert the complete route.
7. Commit the capacity reservation and assignment atomically, then return the route to the remote dispatcher.

No provider side effect occurs inside this transaction. The remote handler dispatches only after the committed assignment is returned.

Filter before ordering

A replica is eligible for a new assignment only when the complete chain is valid:

- the tenant binding and instance are enabled and belong to the same tenant;
- the active, qualified version implements the requested capability and belongs to an enabled connector type;
- the replica serves that instance and version, reports `ready`, has an unexpired lease, and has unused capacity;
- its circuit is closed; and
- it satisfies the required capacity class and the configured cross-region constraint.

The eligible rows are ordered by:

1. lower binding priority;
2. a preferred-region match;
3. a preferred-zone match;
4. fewer active assignments;
5. an action-dependent hash of the replica ID; and
6. stable instance and replica ID tie-breakers.

Region and zone are preferences for a new route. Capacity class is a filter. Cross-region selection is allowed by default, but a deployment can forbid it. None of these preferences changes an existing route.

Treat the assignment as an immutable execution contract

The persisted `RouteAssignment` groups its fields by purpose:

Field group	Fields	Decision retained
Work identity	`action_id`, `capability_id`, `tenant_id`	Which verified work contract was routed
Logical target	`instance_id`, `version_id`, `manifest_digest`	Which product account and admitted implementation own the action
Concrete peer	`replica_id`, `endpoint`, `peer_principal_id`, `peer_principal_kind`, `peer_did`, `trust_domain`, `transport_profile`, `topology`	Which signed process and failure domain received it
Catalog and policy	`catalog_revision`, `binding_policy_revision`, `credential_revision_ref`, `quota_policy_ref`	Which catalog, binding, credential, and quota decision applied
Capacity and fencing	`replica_health_revision`, `fence_token`, `admission`	Which health observation and reserved policy scopes protect execution
Evidence time	`assigned_at`	When the route was created

The remote dispatcher carries the route coordinates needed by the host in signed request metadata and authenticates the assigned peer. The complete assignment remains registry state; not every internal reservation field becomes an AIP message field.

Reuse the route for retry and cancellation

Resolving an already assigned action never runs new target selection. If its previous reservation was released, the registry tries to reserve the pinned admission scopes and the same replica again. That reacquisition accepts a `ready` or `draining` replica only while its lease, version, type, capacity, and circuit state remain valid.

If the pinned replica is unavailable, resolution fails. It does not create a replacement route. This prevents a retry from moving a possibly committed mutation to another provider account or host.

Cancellation reads the same assignment and sends the cancellation to its assigned replica. Reconciliation also uses the retained route to identify the provider boundary whose outcome is uncertain.

Settle capacity without deleting evidence

Settlement accepts the exact assignment and one of four outcomes: `completed`, `outcome_unknown`, `cancelled`, or `lease_expired`.

The registry locks the action, compares the complete stored assignment with the caller-provided value, and rejects a mismatch as `FenceLost`. If the reservation is still active, one transaction:

- releases every admission counter reserved for the attempt;
- marks the route unreserved and records its last settlement;
- decrements the replica's active assignment count; and
- updates the replica circuit state.

Repeated settlement after the reservation is already released is idempotent. The assignment is retained.

A completed outcome resets consecutive unknown failures. An `outcome_unknown` settlement increments the counter when circuit policy is enabled and opens the replica circuit after the configured threshold. A cancelled or lease-expired settlement releases capacity without pretending that the provider succeeded.

Lease maintenance can mark a bounded number of expired replicas offline and release their fenced reservations. It does not contact connector endpoints and does not rewrite the destinations recorded in existing assignments.

Preserve authority and failure boundaries

Boundary	Safe behavior
No verified tenant context	Return no tenant catalog and refuse to derive routing authority from payload fields
Missing binding	Return `BindingUnavailable`; do not search another tenant or unbound implementation
No eligible host	Return `ReplicaUnavailable`; do not route to a draining or expired replica as new work
Admission limit reached	Return `CapacityExceeded` with bounded retry guidance; do not overbook the counter
Changed catalog during pagination	Return `StaleCursor`; restart the read from the first page
Assignment mismatch or stale fence	Return conflict or `FenceLost`; do not settle another reservation
Registry storage failure	Stop the state transition; do not infer that a route, reservation, or settlement committed

The registry stores host endpoints and signed peer identities because routing needs them. It stores opaque credential and secret-provider references because rotation and route evidence need revision identity. It never stores the provider credential value and never lets transport data select a target.

Trade-offs and non-implications

- **Revision locks favor coherent discovery.** A page observes one catalog

publication, while replica heartbeats remain independent of that lock.

- **Per-action serialization favors stable identity.** Concurrent duplicate

requests converge on one assignment but coordinate through one action lock.

- **Row locking favors bounded contention.** `SKIP LOCKED` avoids waiting on a

busy replica row, while bounded retries can still return no capacity during a short contention window.

- **Pinned assignments favor safe recovery.** They prevent silent failover but

can make an unavailable replica a reconciliation problem instead of an immediate rerouting opportunity.

- **Normalized records favor indexed selection.** They avoid loading the whole

fleet, at the cost of explicit schema, transaction, and migration ownership.

The registry does not execute connector code, start an artifact, resolve a provider secret, authorize a client, or prove a provider outcome. The reviewed source contains a scale qualification contract, but its thresholds and results are evidence for a particular environment rather than portable performance or availability promises.

Related pages

- Connector fleet architecture
- Connector admission and supply-chain trust
- Connector fleet HTTP API
- Deploy the connector fleet
- Error reference