Connector admission and supply-chain trust

Use this page to understand how the reviewed AIP implementation converts a signed connector release and independently trusted evidence into registry state that can receive traffic. It is primarily for security reviewers; connector developer

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

The architecture describes version `1.0.0` of the Rust workspace at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. This admission pipeline is an implementation security boundary, not a release process required by AIP 1.0.

A release signature is necessary but insufficient

A valid package signature establishes that a trusted release authority authorized this exact canonical package value. It does not independently prove the expected source, an acceptable license set, or conformance. It also does not prove the absence of disallowed vulnerabilities or a current unrevoked state.

The admission boundary separates those decisions. A release authority signs the complete declarative package. Seven evidence roles each sign a statement about the same artifact and manifest. A deployment-owned trust policy decides which signing identities may speak for each role and which outcome is acceptable.

Only the verifier derives the registry's `ArtifactAttestation`. Package callers cannot submit `Passed` registry statuses directly.

Keep five authorities separate

```
MERMAID
flowchart LR
    B["Build and policy systems"] --> E["Seven signed evidence documents"]
    R["Release authority"] --> P["Signed admission package"]
    E --> P
    P --> V["Admission verifier"]
    T["Deployment trust policy"] --> V
    V --> A["Verified package and derived attestation"]
    A -.->|journaled idempotent writes| G["Connector registry control plane"]
    G --> D["Registry data plane"]
    A -.->|exact targets only| O["External orchestration"]
```

The text equivalent is:

1. Build and policy systems produce evidence documents for one exact artifact

and manifest.

2. Each evidence authority signs a canonical statement containing the evidence

kind, subject digests, document digest, validity window, and outcome.

3. The release authority embeds the seven signed documents in one declarative

package and signs the complete package.

4. The admission verifier applies deployment-owned trust roots, bounds, signatures, digests, time windows, subject relationships, and registry invariants.
5. A short-lived administrator claims the verified package in a durable journal and applies its state through restartable idempotent steps.
6. Long-running data-plane and orchestration processes consume admitted identities without receiving catalog-administrator authority.

Authority	May establish	Does not establish
Release authority	Authorization of the exact complete package	Independent scanner, conformance, license, or revocation success
Evidence authority	One role-specific signed observation and outcome	Permission to publish the package or start a process
Admission verifier	Whether all configured trust, subject, freshness, bounds, and compatibility checks pass	Whether an artifact is running or a provider is healthy
Catalog administrator	Durable application or revocation of verified registry state	Client action authorization or provider mutation
Orchestration executor	Whether exact admitted replica processes match signed deployment intent	Artifact admission, tenant binding, or registry administration

Bind the complete release subject

The signed `AdmissionPackage` is a declarative catalog change, not an image or deployment command.

Package group	Bound content
Package identity	Schema, stable package ID, monotonic revision, issue time, and expiration time
Connector type	Stable type ID, name, owner, and initial enabled state
Version declaration	Version ID, type ID, release label, requested admitted or active status, complete manifest, manifest digest, artifact digest, implementation support, supported AIP versions, and SDK requirement
Policy and tenancy	Admission policies, tenant-owned instances, capability bindings, policy revisions, and opaque credential or secret-provider references
Deployment identities	Pre-provisioned replica IDs, peer identities, endpoints, topology, capacity, and initial lifecycle state
Evidence	Exactly one signed document for each mandatory evidence kind

Every instance in the package must use the package's connector type and version. Every replica must refer to one package instance and version, begin `offline`, and have zero active assignments. Every binding must refer to a package instance owned by the same tenant.

The package requests `admitted` or `active` version status. It cannot introduce a `candidate` or already `revoked` version. The canonical digest of the embedded manifest must equal the declared manifest digest before evidence is considered.

Require seven evidence roles

Evidence kind	Required signed outcome	Decision represented
`oci_signature`	`verified`	A detached signature statement is bound to the immutable artifact digest
`sbom`	`verified`	A software bill of materials document is present and digest-bound
`provenance`	`verified`	Build provenance is present and digest-bound
`conformance`	`passed`	A trusted evaluator accepted the exact artifact and manifest under connector conformance policy
`vulnerability`	`passed`	A trusted evaluator accepted the exact artifact and software bill of materials under vulnerability policy
`license`	`passed`	A trusted evaluator accepted the exact artifact and software bill of materials under license policy
`revocation`	`passed`	A trusted evaluator reported an acceptable non-revocation state for the artifact and signer

The verifier requires the exact seven-kind set. A missing kind fails closed, and an extra unsupported kind is rejected.

Each `EvidenceStatement` binds:

- its evidence schema and kind;
- the exact artifact and manifest digests from the package;
- the canonical digest of the embedded evidence document;
- a bounded human- or service-readable signer identity;
- issue and expiration times; and
- the role-specific outcome.

The embedded evidence document is JSON retained for the release pipeline. The admission verifier checks its canonical size and digest but does not reinterpret the document's domain-specific contents. Trust in `passed` therefore comes from the configured role-specific signer and its signed outcome, not from an untrusted field inside the document.

Apply deployment-owned trust policy

`AdmissionTrustPolicy` supplies package signer roots, a non-empty signer-root set for every evidence kind, and hard limits for package size, evidence size, instance count, replica count, binding count, and clock skew. It can also require one connector owner.

Distinct signing identities are required by default. When that option is enabled, the package signer and all seven evidence signers must be eight different DIDs. This prevents one broad release key from silently asserting scanner, conformance, license, and revocation outcomes.

Disabling distinct-signer enforcement does not remove role-specific trust checks. Each evidence signer still has to belong to the root set configured for that exact evidence kind.

All signatures use Ed25519 `did:key` identities over canonical JSON. The package and evidence statement must have a valid time window: expiration is after issuance and the current time, and issuance cannot exceed the configured clock lead. A signature over stale or not-yet-valid evidence is insufficient.

At this revision, the package signature covers the `package` payload. The outer `signer_did` selects the trusted verification key, while the outer `signer_identity` is bounded and included in the journaled wrapper digest but is not part of the signed payload.

Security decisions must use the trusted DID; the package-level display identity is audit metadata. In contrast, each evidence statement includes its display identity inside the signed statement.

Verify before any catalog write

`verify_admission_package` is a pure verification boundary. It completes the following work before returning a `VerifiedAdmissionPackage`:

1. Validate that trust roots and every configured bound are present and valid.
2. Canonicalize the complete signed wrapper and enforce the package byte limit.
3. Verify the package signer against release roots and verify its signature over

the package payload.

4. Validate package schema, ID, non-zero revision, validity window, optional

owner, object counts, type-version relationship, requested status, and manifest digest.

5. Validate instance, replica, and binding references and their initial state.
6. For each evidence kind, verify the expected subject, trusted role signer,

optional signer separation, identity, time window, document size and digest, statement signature, and exact required outcome.

7. Derive the artifact attestation and connector version from verified data.
8. Re-run registry version admission over manifest bounds, capability

definitions, implementation-support claims, current AIP compatibility, SDK compatibility, attestation statuses, and the canonical schema-bundle digest.

9. Digest the complete signed wrapper for journal and replay fencing.

No mutable registry method is available in this function. Verification failure therefore cannot leave a partial catalog change.

Derive, do not accept, the attestation

The derived `ArtifactAttestation` records:

- the artifact and canonical manifest-schema bundle digests;
- software bill of materials, provenance, conformance, vulnerability, and

license document digests;

- the detached-signature evidence reference and package-level display identity;
- the connector owner, supported AIP versions, and SDK version requirement; and
- passed conformance, vulnerability, license, and revocation statuses.

The verifier invokes registry version validation before returning. Attestation validation is reused during registry admission or activation, host registration, and authenticated connector callback or event ingress.

At the reviewed revision, the declared AIP set must contain the current protocol version, and the SDK requirement must include the registry crate version. Every mandatory policy status must be `passed`, and the schema bundle must match the manifest's input and output schemas.

An attestation is a verified record of supplied evidence. It is not a claim that every possible security property was tested or that an external provider will behave correctly.

Apply through a restartable journal

Verification produces an in-memory trusted value. Application separately uses a short-lived process that holds both registry-administrator and admission journal authority.

Before the first catalog write, `apply_verified_package` records and claims this secret-free operation identity:

Field	Purpose
`package_id`, `revision`, `package_digest`	Fence one exact signed package revision
`connector_type_id`, `version_id`	Identify the catalog subject
`signer_identity`	Retain the bounded package-level display identity for audit correlation
`claim_id`, `claim_expires_at`	Fence the current application worker with a renewable five-minute claim
`state`, `last_error`, `updated_at`	Record progress, bounded failure detail, and audit time

The claim result is one of:

- `New` when no operation existed;
- `Resume` when the same digest can safely continue after a failed or expired

attempt; or

- `AlreadyApplied` when the exact operation completed earlier.

The journal state is `applying` while a valid claim owns the work and `failed` when a resumable step records an error. It becomes `applied` after every step completes or `revoked` after the exact applied version is disabled for new traffic.

The journal rejects a stale revision, a different digest or catalog subject under the same package revision, or a new revision while an earlier revision is unfinished. It also rejects a live competing claim or resumption of a revoked operation.

Apply ordered idempotent steps

The application order is:

1. admission policies;
2. connector type;
3. immutable connector version and capability definitions;
4. connector instances;
5. pre-provisioned offline replicas; and
6. tenant capability bindings.

The process renews its claim before every step. Each registry write validates its references and is idempotent for identical content. Conflicting immutable content or a stale monotonic revision is rejected.

The complete package is deliberately not one database transaction. If a process fails after an intermediate step, the journal records a bounded failure and a later worker resumes the exact package and digest. Earlier identical writes become no-ops; later steps continue in the same order. The operation becomes `applied` only after every step completes.

This multi-step design allows bounded restartable work without giving the central daemon, connector hosts, lifecycle service, or external orchestrator catalog-administrator credentials.

Revoke the exact version

Revocation accepts only an `applied` journal operation. It first moves the exact connector version to irrevocable `revoked` status, which stops new traffic, and then marks the package operation `revoked` with a bounded reason. Repeating an already completed revoke returns the recorded state.

The connector type remains enabled so another independently admitted version of the same type can continue serving. A type-wide emergency stop is a separate registry-administrator action.

Revocation does not erase the package, attestation, route assignments, or journal record. It also does not determine whether an action already in progress committed a provider side effect; that remains a lifecycle and reconciliation decision.

Interpret each evidence level precisely

Observed state	What it supports	What it does not support
Signed package	A trusted release key authorized exact package content	Any independent policy outcome
Complete signed evidence set	Trusted role signers bound required outcomes to the exact artifact, manifest, documents, and validity windows	Correctness beyond the supplied evaluator scopes
`VerifiedAdmissionPackage`	The current verifier accepted trust, subject, bounds, compatibility, and registry invariants	A durable registry write
Journal state `applied`	Every ordered registry step completed for the exact package digest	A running or ready connector host
Active version and ready replica	Routing may consider an otherwise eligible target	Tenant action authorization or provider success
Journal state `revoked`	New traffic is disabled for that exact version	Cancellation or rollback of an already committed provider effect

Source presence, a local image digest, a generated software bill of materials, or an unsigned test report does not reach these evidence levels by itself.

Fail closed at the responsible boundary

Failure	Decision
Unknown package or evidence signer	Reject as a signature failure
Missing, extra, stale, mismatched, oversized, or rejected evidence	Reject as an evidence failure
Wrong schema, relationship, digest, status, owner, count, or compatibility	Reject as invalid admission input
Same revision with different signed content	Reject as an operation conflict
Another unexpired claim owns the operation	Stop and let that claim complete or expire
Idempotent catalog step fails	Record bounded failure and resume the exact digest; do not skip ahead
Applied artifact becomes unacceptable	Revoke the exact version and stop new routing

Never replace failed evidence by editing its signed outcome or registry attestation. Produce a new evidence statement and a new signed package revision under the configured trust policy.

Trade-offs and non-implications

- **Role-specific signatures reduce self-assertion.** They require management

of several trust roots, signer lifecycles, and evidence expiration windows.

- **Canonical digests strengthen subject binding.** Formatting-equivalent JSON

converges on one representation, while a change to canonical content produces a different package identity.

- **Restartable writes favor recoverability.** They expose intermediate durable

catalog state to administrators, so routing eligibility still depends on the complete active type, version, binding, replica, lease, and policy chain.

- **Version-scoped revocation preserves independent releases.** Emergency

containment across every version requires a separate type-wide decision.

Admission does not build or execute an image, start a host, resolve provider credentials, grant a client permission, prove connector readiness, qualify a deployment, or observe an external product. A signed and applied package is a prerequisite for those later boundaries, not a substitute for them.

Related pages

- Build a connector
- Deploy the connector fleet
- Connector registry and routing
- Orchestrate and roll back connector hosts
- Release artifacts and verification