# Architecture documentation

Use this index to choose the architecture page that answers a component, execution, connector-fleet, trust, or dependency question. It is for developers, connector developers, operators, security reviewers, and protocol implementers who nee

| | |
|---|---|
| SECTION | Architecture and Security |
| TYPE | Documentation |
| PROTOCOL | AIP 1.0 |
| SOURCE | docs/architecture/README.md |

These pages describe version `1.0.0` of the Rust workspace at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. They explain how that implementation realizes AIP 1.0; they do not add requirements to the normative wire protocol or prove that a particular deployment is qualified.

## Choose an architecture view

The pages follow this order in documentation navigation:

| Order | Page | Use it to answer |
|---|---|---|
| 1 | Architecture overview | Which layers participate in a request, and where do protocol, profile, runtime, storage, and connector responsibilities separate? |
| 2 | Connector fleet architecture | How do product-neutral daemon processes, lifecycle control, standalone hosts, and external orchestration form one fleet? |
| 3 | Connector registry and routing | How do type, version, instance, replica, binding, lease, and immutable route assignment select an execution target? |
| 4 | Connector admission and supply-chain trust | Which signed package, evidence, identity, digest, revocation, and journal checks permit a connector release to enter the registry? |
| 5 | Gateway | How does ingress become authenticated, policy-governed work dispatched to a local or remote handler? |
| 6 | Runtime | Which component owns durable actions, sessions, approvals, transactions, events, retries, reconciliation, and recovery? |
| 7 | Dependency graph | Which crate layer may depend on which other layer, and where are product and infrastructure dependencies kept out? |
| 8 | Security model | Where are trust established, secrets resolved, signatures and replay checked, tenants isolated, and unsafe outcomes contained? |

Read one page for a bounded question. Follow the complete order when reviewing an architectural change that crosses multiple ownership or trust boundaries.

## Responsibility map

| Boundary | Owns | Does not decide |
|---|---|---|
| Semantic core | Native IDs, envelopes, message and domain types, and pure validation | Transport, async execution, product APIs, persistence, or deployment trust |
| Transports and compatibility profiles | Delivery mechanics and bounded mappings for native AIP, MCP, A2A, and webhooks | Native lifecycle meaning, product behavior, or global authorization policy |
| Gateway | Ingress validation, configured authentication, trusted identity and policy resolution, and dispatch | Durable lifecycle storage or provider-specific API semantics |
| Runtime and storage | Lifecycle state, queues, leases, approvals, transactions, events, callbacks, reconciliation, and retention | Connector selection policy or external product truth |
| Fleet control | Connector catalog, admission, bindings, routing, replica lifecycle, and platform-neutral orchestration plans | Secret material, image execution, or provider mutations |
| Connector host | One admitted product boundary, provider credentials, connector execution, readiness, and event delivery | Central catalog administration or tenant-wide policy definition |

The primary daemon is product-neutral at the reviewed revision. Product adapters execute in separately built connector hosts; the legacy bundled daemon is a migration boundary rather than the target composition.

## Recommended reading paths

| Reader or task | Read in this order | Result |
|---|---|---|
| First implementation review | Architecture overview → Gateway → Runtime → Security model | Follow one request from ingress through durable state and its trust checks |
| Connector-fleet design | Connector fleet architecture → Registry and routing → Connector admission → Security model | Separate release trust, tenant policy, routing, host lifecycle, and orchestration |
| Connector implementation | Architecture overview → Connector fleet architecture → Gateway → Runtime | Place product behavior at the host boundary without changing native semantics |
| Operations and incident review | Connector fleet architecture → Runtime → Security model | Identify the owner of readiness, leases, durable recovery, and containment |
| Crate or feature change | Dependency graph → Architecture overview → the affected owner page | Preserve dependency direction and avoid moving product or infrastructure concerns into the semantic core |
| Protocol implementation | Architecture overview → Gateway → Runtime, then the normative specification | Distinguish required wire meaning from choices made by this Rust implementation |

## Use architecture claims safely

Architecture pages own component responsibilities, flows, trust boundaries, and design trade-offs. They do not own exact HTTP routes, command flags, environment variables, metric names, error inventories, schemas, connector capability catalogs, or qualification results. Use the corresponding reference or connector page for those details.

A diagram or flow describes only the revision and topology named on its page. A deployment can choose different storage, identity, policy, transport, or orchestration providers while preserving the same AIP contract. Verify current support and evidence before treating an optional implementation path as available or production-ready.

Use documentation search for an exact crate, type, route, or trust term. Start with the glossary when two similar fleet or lifecycle terms appear to overlap.

## Related pages

- AIP 1.0 specification
- How AIP works