Examples and reference deployments

Use this page to choose the smallest source artifact that matches your goal. The repository contains short Rust examples, local smoke topologies, scenario harnesses, and qualification environments. These artifact classes answer different qu

SECTION	Build with AIP
TYPE	Guide
PROTOCOL	AIP 1.0
SOURCE	docs/guides/examples-and-reference-deployments.md

Start with a task guide when you want a supported end-to-end learning path. Use an example when you need one focused code pattern. Use a reference topology when you need to inspect component boundaries before designing a deployment.

The inventory targets source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. This authoring pass inspected the source without building an example, starting Docker, contacting a product, or asserting that a retained evidence run exists.

Choose the smallest useful artifact

Goal	Start here	Evidence provided by the artifact itself
Learn one complete local AIP request	Getting-started quickstart	A documented, source-backed procedure
Understand one Rust mapping or framing type	Registered Rust example	Executable source for one bounded concept
Connect an MCP or A2A client	Compatibility-profile guide	A source-backed client procedure and verification boundary
Learn connector-fleet routing	Connector-fleet quickstart	A bounded tutorial using the deterministic fleet topology
Inspect multi-service composition	Reference topology below	Deployment structure, not production qualification
Evaluate a connector artifact	Its qualification harness and retained evidence	Evidence only after the exact harness and artifacts complete successfully

Do not begin with the largest Compose file merely because it contains more components. Larger fixtures introduce credentials, external checkouts, images, state, cleanup, and evidence assumptions that a focused example avoids.

Run a registered Rust example

The `aip` facade crate registers ten examples. Every registered example currently declares the legacy `full` feature aggregate as a Cargo requirement. Run one from the repository root with this pattern:

```sh
cargo run --locked -p aip --example minimal-agent --features full
```

Replace `minimal-agent` with another registered name from the table. The `full` aggregate enables more optional code than most examples use. Treat it as a repository compatibility boundary, not as dependency guidance for a new application.

Example	What the source demonstrates	What it does not do
`minimal-agent`	In-process gateway, test manifest, local handler, and one action result	Start `aipd`, authenticate a transport, or persist state
`sse-streaming`	Encode one in-memory transport event as SSE	Open an HTTP listener or test reconnect behavior
`nats-native`	Build a native NATS subject and envelope headers	Connect to a NATS server or send a request
`mcp-bridge`	Map a local manifest to MCP tools and a tool call back to an action	Start an MCP server or invoke a provider
`mcp-client-bridge`	Discover an MCP peer through an in-memory client transport	Contact a network peer or prove client interoperability
`mcp-server-aipd`	Print canonical endpoint names and an MCP initialize request	Start `aipd` or establish an authenticated session
`mcp-stdio-server`	Encode one newline-delimited MCP request frame	Spawn a child server or manage a session
`mcp-streamable-http`	Encode and decode one MCP SSE event	Perform Streamable HTTP initialization or replay
`mcp-sampling-elicitation`	Construct sampling, elicitation, and response DTOs	Install model, human-input, or completion providers
`chatwoot-dify-crewai`	Compose local mapping types for Chatwoot, Dify, and CrewAI	Call any of the three products or establish a durable workflow

The output of `mcp-server-aipd` includes an illustrative daemon command from the example source. It is not the current deployment recipe. Use the MCP guide for a runnable loopback endpoint and for the separate production introspection controls.

Most examples print JSON or framing text to standard output. Validate the specific field that matters to your adaptation rather than treating process exit alone as behavioral evidence.

Choose a source topology

The multi-service directories have narrower purposes than their names may suggest:

Source path	Intended question	Important boundary
`examples/hermes-agent-aip`	Can the Hermes Agent connector discover and smoke-check two local Hermes gateways?	Builds an adjacent external checkout, uses a hard-coded smoke credential, and is not a production topology
`examples/cal-diy-qualification`	Can one pinned Cal.diy artifact satisfy the isolated qualification procedure?	Requires the exact upstream checkout, local images, Docker, generated secrets, retained run evidence, and the legacy bundled daemon
`examples/restaurant-booking`	How do Cal.diy, Hermes Agent, a coordinator, MCP, approval, and durable state compose in one scenario?	Uses the legacy bundled daemon and local identity fixtures despite the deployment-oriented Compose name
`deploy/connector-fleet`	How do admission, registry, control plane, edge, product-neutral daemons, remote hosts, mTLS, and qualification fit together?	It is a deterministic qualification topology with fixture-specific trust, endpoints, images, and evidence paths
`examples/aipd-docker`	How does the older bundled local E2E assemble its dependencies?	Contains non-public qualification fixtures and is excluded from the public adoption path

None of these directories is a drop-in production deployment. A source path can be useful as architecture evidence while remaining unsuitable as an operational template.

Compare public connector example coverage

The six maintained connectors do not have symmetric example coverage at the reviewed revision:

Connector	Focused or scenario example	Deterministic fleet product profile	Current interpretation
Cal.diy	Isolated qualification and restaurant-booking scenario	Yes	Strongest dedicated local harness coverage; evidence still belongs to an exact completed run

Connector	Focused or scenario example	Deterministic fleet product profile	Current interpretation
Hermes Agent	Two-node smoke topology and restaurant-booking scenario	Yes	Covers smoke and scenario composition; neither proves a general production deployment
Chatwoot	Local three-product mapping example	Yes	Mapping plus fixture-backed fleet execution, not a live external-product claim
Dify	Local three-product mapping example	Yes	Mapping plus fixture-backed fleet execution, not a live external-product claim
CrewAI	Local three-product mapping example and fleet sidecar	Yes	Mapping and durable fixture-sidecar behavior, not an arbitrary crew qualification
Twenty	No example directory at this revision	No	Connector and host source exist, but this inventory provides no runnable-example or fleet-profile evidence

The product profile under `deploy/connector-fleet` covers five product hosts. It uses controlled upstream fixtures to test connector and fleet mechanics. It does not contact the five public products, and it must not be reported as live product verification.

Interpret the evidence correctly

An artifact becomes stronger evidence only when the corresponding step is actually completed and recorded:

Observed state	Claim it can support	Claim it cannot support
Source file exists	The repository contains the described implementation	The example compiles or runs
Exact example compiles	The selected revision type-checks for that build	Runtime behavior or interoperability
Example exits with checked output	The bounded local behavior occurred once	Conformance, qualification, or production readiness
Scenario harness completes	The recorded scenario passed for exact inputs and artifacts	General connector coverage or external-product compatibility
Qualification harness and evidence pass	The identified artifact passed the documented matrix	Other revisions, topologies, tenants, or products
Live external-product run passes	The exact observed upstream behavior succeeded	Future behavior or production SLOs
Production observation meets a defined window	The named deployment met that bounded operational claim	Universal protocol or connector guarantees

Directory names, comments, fixture labels, and intended commands are not run results. Keep source truth, current checkout truth, retained evidence, and live deployment truth separate.

Adapt an example without importing its assumptions

Before copying code or configuration, record:

1. the exact source revision and example path;
2. whether the example uses `aip-testkit`, an in-memory transport, a local

fixture, or a real external process;

3. which identities, scopes, tenants, credentials, and signing keys are

represented or omitted;

4. whether state is memory-backed, file-backed, PostgreSQL-backed, or owned by

an external product;

5. which side effects occur and how idempotency, approval, retries, and unknown

outcomes are handled;

6. the exact expected output and the evidence artifact that records it;
7. every directory, container, network, volume, and secret created by the run.

Replace test manifests and hard-coded credentials before adapting a sample to an application. Preserve stable capability IDs and source-owned schemas. Do not copy a product-specific connector into the product-neutral daemon process when the intended architecture is a standalone connector host.

Verify a selected artifact

For a Rust example, verify the command exit status and parse its documented output. Then inspect the source again to confirm whether the result came from a real boundary or an in-memory substitute.

For a multi-service topology, verify all of these before making a claim:

- exact source, external checkout, image, and configuration identities;
- readiness of every required component, not only process liveness;
- expected positive behavior and relevant negative controls;
- durable state and restart behavior when the claim depends on recovery;
- an evidence file that records timestamps, inputs, results, and artifact

identities;

- cleanup scoped to the exact Compose project and its generated runtime path.

Do not convert a saved historical result into a current-live claim. Re-run the documented procedure against the intended artifacts when current evidence is required.

Clean up within the artifact boundary

The registered in-memory examples create no deployment state, although Cargo may update its normal build cache. Multi-service harnesses can create untracked secret directories, evidence directories, images, containers, networks, and volumes.

Use the cleanup procedure owned by the selected harness. Inspect its Compose project name and generated paths first. Do not use a global Docker prune or remove unrelated volumes to clean up one example.

Related documentation

- Quickstart
- Connector-fleet quickstart
- Use native AIP
- Use AIP through MCP
- Use AIP through A2A