Use AIP through MCP

Use this guide to connect an existing MCP client to getaip-server, inspect the projected surface, and call one native AIP capability through a stable facade tool. The same AIP gateway validates and dispatches the resulting action; the MCP l

SECTION	Build with AIP
TYPE	Guide
PROTOCOL	AIP 1.0
SOURCE	docs/guides/use-aip-through-mcp.md

The walkthrough first uses an unauthenticated loopback endpoint for local development. It then shows the separate controls required for a protected network endpoint. It does not qualify a deployment, an identity provider, or an independent MCP client.

The examples target source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. This authoring pass checked the source and command syntax without building the workspace or starting a service.

Choose MCP or native AIP

Use MCP when an editor, agent host, or automation client already implements MCP and needs a compatible view of AIP capabilities. Use native AIP when the client needs direct access to the complete AIP message, lifecycle, identity, and transport contract.

Client need	Preferred path
Discover and call AIP capabilities from an MCP host	MCP profile
Launch one local server process from an editor	MCP over stdio
Connect a network client with resumable server events	MCP over Streamable HTTP
Send native envelopes or use the complete AIP operational API	Native AIP
Depend on an AIP guarantee that the selected MCP version cannot express	Native AIP, or an explicitly documented facade tool

MCP tool metadata alone does not grant AIP approval, transaction, retry, or delivery guarantees. Read the capability contract returned by AIP before depending on those semantics.

Choose a transport and version

The client and server negotiate one protocol version during MCP initialization. The current implementation intentionally exposes this matrix:

MCP transport	2024-11-05	2025-03-26	2025-06-18	2025-11-25
stdio	Yes	Yes	Yes	Yes
Legacy HTTP+SSE	Yes	No	No	No
Streamable HTTP	No	Yes	Yes	Yes

Streamable HTTP uses `GET`, `POST`, and `DELETE` on `/mcp`. The aliases `/mcp/v1` and `/aip/v1/mcp` expose the same handler. Legacy compatibility uses `GET /mcp/legacy/sse` and `POST /mcp/legacy/messages` and is limited to `2024-11-05`.

If the requested version cannot run on the selected transport, the server selects the newest mutually supported executable version. Version-specific methods remain unavailable when the selected version does not define them. For example, task methods are available only with `2025-11-25`.

Prerequisites

For the loopback walkthrough, you need:

- the reviewed source checkout and its locked Rust dependencies;
- `curl` and `jq` for the explicit verification steps;
- local TCP port `18080`, or another unused loopback port used consistently;
- permission to create `.getaip-server-mcp-guide` in the checkout;
- two terminals, one for `getaip-server` and one for the client commands.

Do not expose the development command on a non-loopback interface. It bypasses HTTP authentication intentionally.

1. Start a loopback Streamable HTTP endpoint

From the repository root, start `getaip-server`:

```
SH
cargo run --locked -p getaip-server -- \
  --bind 127.0.0.1:18080 \
  --service-id agent:getaip:server:mcp-guide \
  --storage-dir .getaip-server-mcp-guide \
  --allow-insecure-development
```

Keep this process running. In the second terminal, wait until the daemon reports that the gateway and required workers are ready:

```
SH
curl -fsS http://127.0.0.1:18080/ready \
  | jq -e '.status == "ready"'
```

A successful check prints `true`. The `/ready` result is readiness for this process and configuration. It is not connector qualification or evidence that an external product is reachable.

2. Inspect the projected surface

Initialize a Streamable HTTP session and inspect the MCP peer:

```
SH
cargo run --locked -p getaip-cli -- mcp inspect \
  --url http://127.0.0.1:18080/mcp \
  | jq -e '
      .compatibility.mcp.protocol_version == "2025-11-25"
      and any(.capabilities[]; .name == "aip_capabilities")
      and any(.capabilities[]; .name == "aip_call")
    '
```

The command initializes the client, lists the MCP surface, and prints an AIP manifest reconstructed by the outbound client bridge. A successful check prints `true`.

Capability IDs in this reconstructed manifest use bridge-local IDs derived from the client configuration and MCP tool name. They are not the original native AIP capability IDs. Use `aip_capabilities` or the native manifest when you need stable native identity.

3. Discover and call a stable AIP capability

Ask the stable facade for the native daemon health capability:

```
SH
cargo run --locked -p getaip-cli -- mcp call-tool aip_capabilities \
  --url http://127.0.0.1:18080/mcp \
  --arguments '{
    "capability_id": "cap:aip:server:health",
    "include_schemas": false,
    "include_contracts": true,
    "include_bindings": false
  }' \
  | jq -e '
      .isError == false
      and any(
        .structuredContent.capabilities[];
        .id == "cap:aip:server:health"
      )
    '
```

The returned contract identifies this operation as a read with synchronous execution support. Invoke it through `aip_call` by native capability ID:

```
SH
cargo run --locked -p getaip-cli -- mcp call-tool aip_call \
  --url http://127.0.0.1:18080/mcp \
  --arguments '{
    "capability_id": "cap:aip:server:health",
    "input": {}
  }' \
  | jq -e '
      .isError == false
      and .structuredContent.status == "ok"
      and .structuredContent.protocol == "AIP"
    '
```

A successful call prints `true`. The local manifest also maps this capability to the generated MCP tool name `getaip_server_health`. Treat that name as a projection, not as the stable native identity. The facade pair `aip_capabilities` and `aip_call` remains usable when a tenant connector-fleet catalog replaces the generated per-capability tool list.

4. Use stdio for a command-launched client

Choose stdio when the MCP host owns the `getaip-server` child process. After building the release binary, verify the same projection directly:

```
SH
cargo run --locked -p getaip-cli -- mcp inspect \
  --command ./target/release/getaip-server \
  --arg=--mcp-stdio \
  --arg=--service-id \
  --arg=agent:getaip:server:mcp-stdio \
  --arg=--storage-dir \
  --arg=.getaip-server-mcp-stdio
```

Configure another MCP host with the same command and ordered argument list. Use an absolute executable and state path when the host does not start in the repository directory. MCP host configuration field names vary, but the child process boundary uses these values:

Field	Value
Command	Absolute path to the reviewed `getaip-server` binary
Arguments	`--mcp-stdio`, `--service-id`, one canonical service principal, and deployment-owned storage options
Framing	One JSON-RPC frame per line on standard input and output
HTTP credentials	Not applicable to the stdio boundary

The stdio transport uses one process-owned session. The spawning host is the trust boundary; HTTP bearer tokens and browser Origin checks do not apply. Use `--mcp-principal` and repeated `--mcp-principal-scope` values when the host needs a more specific AIP actor than the deployment default. Restart a host that caches command configuration or initialization state.

5. Check client compatibility

Run the outbound MCP client conformance checks against the loopback endpoint:

```
SH
cargo run --locked -p getaip-cli -- mcp conformance \
  --url http://127.0.0.1:18080/mcp
```

Or check the command boundary:

```
SH
cargo run --locked -p getaip-cli -- mcp conformance \
  --command ./target/release/getaip-server \
  --arg=--mcp-stdio \
  --arg=--service-id \
  --arg=agent:getaip:server:mcp-conformance
```

The command exits unsuccessfully when its client-conformance report contains a failure. A passing report covers the tested client path and negotiated contract. It does not establish production readiness, independent-client interoperability, connector qualification, or live-product behavior.

Understand the projected surface

`aip-profile-mcp` owns MCP data transfer objects and their AIP mappings. It does not own HTTP state, subprocesses, or the AIP runtime. Separate session, server, client, and transport components enforce those boundaries.

The current `getaip-server` composition exposes stable AIP facade tools and, without a fleet catalog, generated tools for local capabilities. Other MCP families are available only when both the negotiated version and installed provider support them:

Surface	Current boundary
Stable AIP facade tools	Advertised by `getaip-server`; calls pass through the native gateway and runtime
Generated capability tools	Derived from the local manifest; disabled when the fleet catalog owns discovery
Resources	Metadata can come from the manifest; content reads require a resource provider
Prompts and completions	Advertised only when their providers supply entries
Roots, sampling, and elicitation	Depend on negotiated client capability and a real peer/provider path
Logging	Supported by the current server composition
Tasks	Advertised only for `2025-11-25` when task support is enabled
Dynamic list notifications	Emitted when the installed projection source changes

Do not infer provider availability from the MCP schema vocabulary alone. An empty list or unsupported-method response can be correct for a deployment that has not installed that provider.

Understand sessions and replay

Streamable HTTP initialization creates or accepts a session ID and returns the selected version in response headers. Later `POST` requests carry that session ID and the negotiated protocol version. `GET /mcp` opens the server stream, and `DELETE /mcp` closes the session and its replay state.

The server binds the session to the authenticated actor and optional verified tenant. Another identity cannot take over the ID. Tool arguments cannot select the actor because the server constructs trusted execution context before it calls the AIP gateway.

Two persistence boundaries must remain distinct:

- MCP session snapshots use the configured runtime profile store, which can be

PostgreSQL, durable local storage, or memory;

- Streamable HTTP SSE replay retains at most 1,024 events per session and is

file-backed only when `--storage-dir` is set.

PostgreSQL runtime storage alone does not persist the SSE replay log. A client can send `Last-Event-ID` on `GET /mcp` to replay retained events before the live stream continues. This is bounded reconnect support, not an unlimited AIP event archive.

Protect a production HTTP endpoint

The local bypass is unavailable on a public listener. A production MCP endpoint uses protected-resource metadata and RFC 7662 token introspection. The following command shows the MCP-specific controls in context; complete storage, edge, observability, backup, and rollout design belongs in the production deployment guide.

```
SH
cargo run --locked -p getaip-server -- \
  --bind 0.0.0.0:18080 \
  --public-base-url https://aip.example.com \
  --service-id agent:getaip:server:production \
  --postgres-url-file /run/secrets/getaip-server-postgres-url \
  --storage-dir /var/lib/getaip-server \
  --mcp-resource https://aip.example.com/mcp \
  --mcp-authorization-server https://identity.example.com \
  --mcp-scope aip.invoke \
  --mcp-required-scope aip.invoke \
  --mcp-introspection-url https://identity.example.com/oauth2/introspect \
  --mcp-introspection-issuer https://identity.example.com \
  --mcp-introspection-client-id getaip-server \
  --mcp-introspection-client-secret-file /run/secrets/mcp-introspection-secret \
  --mcp-allowed-origin https://app.example.com
```

Terminate TLS at a trusted edge or deployment platform and route the public HTTPS origin to the private listener. The configured public base URL must use HTTPS for a non-loopback bind.

Before exposing the route, verify these controls:

- the resource identifier matches the token audience and published endpoint;
- the introspection issuer is one of the advertised authorization servers;
- the token is active, unexpired, and contains every required scope;
- the token subject parses as a canonical AIP principal;
- any tenant used for routing comes from the verified token, not tool input;
- every browser Origin is explicitly allowed; loopback origins remain the only

implicit browser exception;

- the database URL and introspection client secret are readable only from

their deployment-owned secret files.

Pass an access token to `getaip` with `--bearer-token` or the `GETAIP_MCP_BEARER_TOKEN` environment variable. Prefer a secret-injection mechanism that does not persist the token in shell history. Static `--mcp-bearer-token` authentication on the server is restricted to explicit insecure development and is not a

production alternative to introspection.

Resolve common failures

Symptom	Likely boundary	Safe next check
`401 Unauthorized`	HTTP authentication	Confirm the endpoint is not using the loopback bypass, then check introspection activity, issuer, audience, expiry, and required scopes
Browser request is rejected before authentication	Origin policy	Compare the exact request Origin with repeated `--mcp-allowed-origin` values
Session ID is not found	Ownership or deletion	Reinitialize with the same authenticated identity; do not reuse an ID after `DELETE`
Protocol version is rejected	Transport/version mismatch	Select a version marked for the chosen transport in the matrix above
`aip_capabilities` omits a connector capability	Catalog visibility	Check verified tenant context, fleet admission, catalog revision, filters, and pagination cursor
Generated tool disappeared after fleet enablement	Discovery mode changed	Use `aip_capabilities` and `aip_call` with the stable native capability ID
Resource, prompt, or completion list is empty	Provider is absent	Verify that the deployment installed and populated the matching provider
Reconnect cannot replay an older event	Replay boundary	Check `--storage-dir`, the 1,024-event bound, session ownership, and `Last-Event-ID`
Call completed but an AIP lifecycle view is missing	Client used only the tool result	Query the appropriate stable facade lifecycle tool or use native AIP observation APIs

Do not retry a mutating call merely because the MCP connection ended. First look up the AIP action using its stable action or idempotency identity. A lost response can leave the execution outcome unknown.

Stop and clean up the local walkthrough

Stop `getaip-server` with `Control-C`. Inspect the two local state directories before removing only the walkthrough data:

```sh
du -sh .getaip-server-mcp-guide .getaip-server-mcp-stdio 2>/dev/null || true
rm -rf -- .getaip-server-mcp-guide .getaip-server-mcp-stdio
```

Do not apply this cleanup command to a production state path. It removes local runtime state and any file-backed MCP replay retained under those directories.

Related documentation

- Use native AIP
- Use the Rust SDK
- Profiles and connectors
- Actions and sessions
- Identity and trust