Use AIP through A2A

Use this guide to connect an A2A 1.0 client to getaip-server, discover the local Agent Card, invoke one native AIP capability, and read the projected task. The A2A profile translates JSON-RPC messages into native actions. AIP remains the ow

SECTION	Build with AIP
TYPE	Guide
PROTOCOL	AIP 1.0
SOURCE	docs/guides/use-aip-through-a2a.md

The walkthrough uses one bearer identity on a loopback listener and invokes the built-in health capability. Later sections explain streaming, push notifications, and the current tenant boundary. The walkthrough does not qualify a production deployment or an independent A2A client.

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

Choose the A2A boundary deliberately

Use A2A when an existing agent already implements the A2A task model and needs to call a capability in the local `getaip-server` manifest. Use native AIP when the client needs the complete envelope, verified tenant context, operational read model, or a protocol feature that the A2A projection does not expose.

Client need	Preferred path
Discover local skills through an Agent Card	A2A
Send a user message and follow a projected task	A2A
Invoke a tenant-routed remote connector fleet	Native AIP or the documented MCP facade
Preserve complete AIP identity, approval, transaction, and receipt semantics	Native AIP
Launch a command-based compatibility client	MCP over stdio

An Agent Card is discovery data, not authorization. A visible skill can still require scopes, approval, idempotency, or other AIP policy.

Know the current routes and names

The preferred public routes are:

Purpose	Route
Agent Card	`GET /.well-known/agent-card.json`
A2A 1.0 JSON-RPC	`POST /a2a/v1`

The Agent Card aliases `/.well-known/agent.json` and `/a2a/agent-card` return the same local projection. The JSON-RPC aliases `/a2a` and `/aip/v1/a2a` reach the same handler.

Current operation names are `SendMessage`, `SendStreamingMessage`, `GetTask`, `ListTasks`, `CancelTask`, `SubscribeToTask`, the four task push-notification configuration operations, and `GetExtendedAgentCard`.

Explicit v0.3 method aliases remain accepted for migration. New clients should send current names and should not infer a separate legacy task store.

The `A2A-Version` request header is optional. When present, its value must be `1.0` at the reviewed revision.

Prerequisites

For the loopback walkthrough, you need:

- the reviewed source checkout and its locked Rust dependencies;
- `curl` and `jq`;
- local TCP port `18080`, or another unused loopback port used consistently;
- permission to create local guide state and an owner-only token file;
- two terminals, one for `getaip-server` and one for client commands.

The example token is intentionally local and disposable. Do not reuse it on a shared or public endpoint.

1. Create the local bearer credential

Create one owner-readable token file from the repository root:

```
SH
umask 077
printf '%s\n' 'a2a-guide-token' > .getaip-server-a2a-token
```

`getaip-server` trims a conventional trailing line ending when it reads the secret file. Keep the file out of version control.

2. Start the authenticated A2A edge

Start the daemon in the first terminal:

```
SH
cargo run --locked -p getaip-server -- \
  --bind 127.0.0.1:18080 \
  --service-id agent:getaip:server:a2a-guide \
  --storage-dir .getaip-server-a2a-guide \
  --native-bearer-token-file .getaip-server-a2a-token \
  --native-principal service:a2a:guide \
  --native-principal-scope action:read \
  --native-principal-scope action:write
```

Keep the process running. In the second terminal, verify readiness:

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

A successful check prints `true`. This result covers the local process and its required workers. It does not prove external connectivity, connector qualification, or production readiness.

3. Inspect the local Agent Card

Agent Card discovery is public. Fetch it without the bearer token:

```
SH
curl -fsS http://127.0.0.1:18080/.well-known/agent-card.json \
  | tee .getaip-server-a2a-card.json \
  | jq -e '
      .supportedInterfaces[0].url == "http://127.0.0.1:18080/a2a/v1"
      and .supportedInterfaces[0].protocolVersion == "1.0"
      and .capabilities.streaming == true
      and .capabilities.pushNotifications == false
```

```
      and .securitySchemes.aipBearer.httpAuthSecurityScheme.scheme == "Bearer"
    '
```

A successful check prints `true`. Push notification support is `false` in this walkthrough because no callback signer or credential-encryption key is configured.

The card projects capabilities from the local AIP manifest into A2A skills. It declares content modes, the preferred JSON-RPC interface, and its bearer scheme. It does not expose secrets or grant task access.

4. Send one health message

Create `.getaip-server-a2a-send.json`:

```
JSON
{
  "jsonrpc": "2.0",
  "id": "guide-send-1",
  "method": "SendMessage",
  "params": {
    "message": {
      "messageId": "message-a2a-health-1",
      "taskId": "task-a2a-health-1",
      "contextId": "context-a2a-guide",
      "role": "ROLE_USER",
      "parts": [
        {
          "data": {},
          "mediaType": "application/json"
        }
      ]
    },
    "metadata": {
      "aip": {
        "capability_id": "cap:aip:server:health",
        "input": {}
      }
    }
  }
}
```

The current send mapping requires a non-empty `messageId`, the `ROLE_USER` role, at least one part, and `metadata.aip.capability_id`. The profile maps `messageId` to the native idempotency key `a2a-message:message-a2a-health-1`. A skill label alone is not stable routing authority.

Send the request and retain the response:

```
SH
A2A_TOKEN="$(sed -n '1p' .getaip-server-a2a-token)"
curl -fsS \
  -H "Authorization: Bearer ${A2A_TOKEN}" \
  -H 'A2A-Version: 1.0' \
  -H 'Content-Type: application/json' \
  --data-binary @.getaip-server-a2a-send.json \
  http://127.0.0.1:18080/a2a/v1 \
  | tee .getaip-server-a2a-send-response.json \
  | jq -e '
      .error == null
      and .result.task.id == "task-a2a-health-1"
      and .result.task.status.state == "TASK_STATE_COMPLETED"
      and any(
        .result.task.artifacts[]?.parts[]?;
        .text == "getaip-server is healthy"
      )
    '
```

A successful check prints `true`. The response is an A2A task projection. Its metadata retains the native action identity needed for later lifecycle reads.

When `metadata.aip.input` is absent, the profile derives input from the A2A message and its first text part. Supply explicit input when the native capability expects a structured contract.

5. Read the task again

Extract the returned task ID and issue `GetTask`:

```
SH
A2A_TOKEN="$(sed -n '1p' .getaip-server-a2a-token)"
TASK_ID="$(jq -er '.result.task.id' .getaip-server-a2a-send-response.json)"
curl -fsS \
  -H "Authorization: Bearer ${A2A_TOKEN}" \
  -H 'A2A-Version: 1.0' \
  -H 'Content-Type: application/json' \
  --data-binary "$(
    jq -nc --arg id "${TASK_ID}" '{
      jsonrpc: "2.0",
      id: "guide-get-1",
      method: "GetTask",
      params: {id: $id, historyLength: 5}
    }'
  )" \
  http://127.0.0.1:18080/a2a/v1 \
  | jq -e --arg id "${TASK_ID}" '
      .error == null
      and .result.id == $id
      and .result.status.state == "TASK_STATE_COMPLETED"
    '
```

A successful check prints `true`. `historyLength` keeps the newest requested history entries while preserving their native sequence order.

Choose synchronous, asynchronous, or streaming work

The send configuration changes response behavior, not the underlying AIP lifecycle:

A2A operation or option	Current behavior
`SendMessage`	Dispatches through the gateway and returns a completed, submitted, or input-required task according to the native outcome
`SendMessage` with `configuration.returnImmediately: true`	Requests native asynchronous mode and returns the durable task handle
`SendStreamingMessage`	Forces asynchronous mode and emits task status and artifact updates as SSE until a terminal state
`SubscribeToTask`	Opens a new SSE view for an existing nonterminal task
`GetTask`	Reads one native status projection and optionally limits history
`ListTasks`	Lists visible A2A-bound actions with filters and page sizes from 1 through 100
`CancelTask`	Persists native cancellation intent unless the task is already terminal

The streaming handler polls durable native action status and retained chunks. Its SSE events do not contain event IDs, and it does not implement `Last-Event-ID`. Reconnect a nonterminal task with `SubscribeToTask` and the task ID. Read a terminal task with `GetTask`; terminal-task subscription is rejected.

A transport disconnect does not prove that a mutation failed. Read the task or native action before retrying the same logical work, and reuse the original `messageId` when idempotency should return the original result.

Understand identity and tenant limits

Every A2A JSON-RPC operation requires the configured native bearer token. The daemon compares the token in constant time, binds the configured principal and scopes, and constructs trusted gateway context. JSON-RPC metadata cannot choose the authenticated actor. Public Agent Card discovery remains unauthenticated.

The current command-line A2A edge maps one static token to one principal. It does not perform OAuth introspection or map different external users to different AIP principals. A trusted edge can protect and rate-limit the route, but requests forwarded with the daemon token still share that one daemon-side identity.

At this revision, A2A JSON-RPC dispatch calls the verified gateway without a `VerifiedTenant`. Treat a request `tenant` field as profile data or a list filter, not as tenant authority. When connector fleet services are enabled, the Agent Card advertises an optional tenant-capability-discovery extension that points to a separate authenticated native endpoint. It does not materialize the remote fleet catalog as Agent Card skills.

Use native AIP or the stable MCP fleet facade when verified tenant routing is a requirement. Do not describe A2A task success as connector-fleet qualification.

Enable push notifications safely

`getaip-server` advertises A2A push notifications only when both controls are present:

- an Ed25519 callback signer from `--callback-signing-seed-file`, its inline

alternative, or the corresponding environment variable;

- a 32-byte encryption key from `GETAIP_SERVER_A2A_PUSH_ENCRYPTION_KEY_HEX` or the

inline `--a2a-push-encryption-key-hex` option.

Prefer a file for the signing seed and secret-manager injection for the encryption-key environment variable. Inline hexadecimal values can persist in shell history or process inspection.

For a non-loopback deployment, the A2A-specific controls fit into a command such as this after the encryption-key environment variable is injected:

```
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:a2a-production \
  --postgres-url-file /run/secrets/getaip-server-postgres-url \
  --native-bearer-token-file /run/secrets/getaip-server-native-token \
  --native-principal service:a2a:trusted-edge \
  --native-principal-scope action:read \
  --native-principal-scope action:write \
  --callback-signing-seed-file /run/secrets/aip-callback-signing-seed \
  --callback-allowed-host callbacks.example.com
```

Terminate TLS at a trusted edge and route the public HTTPS origin to the private listener. The external base URL must be an HTTPS origin without a path, query, fragment, or embedded credentials.

Before accepting a push configuration, verify that:

- the callback URL uses HTTPS and its exact host is allowlisted;
- private and loopback destinations remain blocked unless an owned isolated

network requires an explicit exception;

- stored callback token and authorization credentials use the configured

encryption key;

- the receiver verifies the signed callback and treats deliveries as

idempotent;

- durable profile state, callback delivery state, recovery, backup, and key

rotation are part of the deployment design.

The delivery worker uses a durable cursor and fencing to coordinate recovery. That control suppresses competing recovery work but does not justify an exactly-once provider claim.

Resolve common failures

Symptom	Likely boundary	Safe next check
`401 Unauthorized`	A2A bearer identity	Confirm `--native-bearer-token-file`, the exact `Authorization` header, and token-file permissions
`VERSION_NOT_SUPPORTED`	A2A header	Send `A2A-Version: 1.0` or omit the optional header
`INVALID_PARAMS` for send	Message mapping	Check `messageId`, `ROLE_USER`, at least one part, and `metadata.aip.capability_id`
Agent Card omits a fleet capability	Discovery boundary	Use the advertised tenant discovery extension, native AIP, or the stable MCP facade
`TASK_NOT_FOUND`	Identity, task binding, or retention	Reuse the authenticated principal and exact returned task ID; then inspect native action state
Streaming ended before the result was read	Connection boundary	Call `GetTask`; use `SubscribeToTask` only while the task is nonterminal
Push support is `false`	Callback controls	Configure both the signer and A2A credential-encryption key, then restart the daemon
Push configuration is rejected	Destination or ownership policy	Check task access, HTTPS, the exact host allowlist, private-network policy, and configuration ID conflicts

JSON-RPC application errors normally arrive in a successful HTTP response. Inspect the JSON `error` field even when `curl` reports HTTP success.

Stop and clean up the walkthrough

Stop `getaip-server` with `Control-C`. Inspect and remove only the local guide files:

```sh
du -sh .getaip-server-a2a-guide 2>/dev/null || true
rm -rf -- .getaip-server-a2a-guide
rm -f -- \
  .getaip-server-a2a-token \
  .getaip-server-a2a-card.json \
  .getaip-server-a2a-send.json \
  .getaip-server-a2a-send-response.json
```

Do not apply this cleanup to a production state path. Durable task, push, callback, and recovery records can reside there.

Related documentation

- Use native AIP
- Profiles and connectors
- Actions and sessions
- Identity and trust
- Approvals and policy