Hermes operator and delegation capability

Use the endpoint-qualified operator capability when Hermes must plan a governed objective, call admitted AIP tools, pause for approval, resume the same provider run, or reconcile cancellation across a process restart.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/hermes-agent/capabilities/operator-and-delegation.md

This lifecycle adds a durable AIP binding around a volatile Hermes structured run. It does not make Hermes run storage durable. The binding prevents an unknown external start or approval command from being replayed automatically.

The same connector can also own first-class AIP delegation. That route is disabled by default and uses a different entry contract: the child Action keeps its real capability ID, while the delegate principal selects the Hermes endpoint.

Operation contract

Replace `` with the normalized endpoint ID present in AIP discovery.

Field	Published value
Capability	`cap:hermes_agent:<endpoint_id>:operator`
Provider request	`POST /v1/runs` with endpoint Bearer authentication
Kind and stability	Agent; stable
Risk and approval	High; tenant-policy AIP approval required
Idempotency	Required; capability scope; 24-hour TTL; ReturnOriginalResult
Side effects	`read`, `write`, `send_message`, `external_network`, `code_execution`
Data	Confidential; contains PII; redaction required
Execution	Sync, async, streaming, and cancellation supported
Expected completion	Streaming; `async_expected` is true
Expected latency and timeout	30 seconds and 15 minutes
Transaction	None
Compensation	Rollback not supported; approval required

The approval policy lasts 15 minutes and requires a reason, input snapshot, and policy-decision evidence. The contract also declares a 10-second maximum queue-delay hint and a `99.9%` availability target. These are declarations, not observed performance or qualification evidence.

Choose direct operator work or delegation

Need	Entry	Capability carried by the Action	Result authority
Ask Hermes to plan and execute a bounded objective	Invoke `operator` with `objective`	Hermes `operator` capability	Governed operator binding and provider terminal result
Continue a source operator Action after a provider approval prompt	Invoke `operator` with `resume` in a new Action	Hermes `operator` capability	Stored source binding plus provider continuation
Delegate one exact native AIP child Action to Hermes	Send a `DelegationRequest` to `agent:hermes_operator:<endpoint_id>`	The child's actual target capability	Native child Action queue and lifecycle result

`candidate_capabilities` in a direct objective is a model hint. It does not add an admitted capability, authorize a call, bypass approval, or replace schema validation. The AIP MCP server remains the execution boundary for downstream actions.

First-class delegation is stricter. Hermes receives one exact `aip_call` argument object and instructions to invoke it once without rewriting it. The connector ignores Hermes final prose as the delegated business result.

Submit a direct objective

The input schema selects exactly one branch: `objective` or `resume`. Do not send both.

An objective request accepts:

Field	Constraint	Purpose
`objective`	Required non-empty string	The bounded outcome for Hermes to pursue
`context`	Any JSON value	Application context treated as data, not authority
`candidate_capabilities`	At most 256 non-empty strings	Hints about relevant admitted capabilities
`constraints`	Object	Task-specific limits that do not override AIP policy

A bounded input can look like this:

```
JSON
{
  "objective": "Read the open support case and prepare a factual summary.",
  "context": {
    "case_id": "case-example-2030"
  },
  "candidate_capabilities": [
    "cap:support:case.get"
  ],
  "constraints": {
    "allow_mutations": false
  }
}
```

Set the AIP Action ID, idempotency key, approval decision, identity, deadline, and delegation chain on the Action itself. Do not place them inside this input.

The connector serializes a provider payload with kind `aip_operator_objective`. It assigns the stable Hermes session ID `aip-operator-` and adds deployment-owned system instructions. Those instructions require server-assigned downstream Action IDs, prohibit reuse of the enclosing operator Action ID, and reject fabricated results or approvals.

The serialized provider input defaults to at most 1 MiB. An optional model is selected only by deployment policy. Client input cannot choose an endpoint URL, provider credential, system instruction, or model route for the operator.

Understand the durable binding

Before the connector calls Hermes, it creates one profile-state record under `aip.connector.hermes_agent.operator_runs.v1`, keyed by the source AIP Action ID.

Binding field	Purpose
`action_id`, `request_fingerprint`	Detect exact replay and changed-input reuse
`endpoint_id`, `run_id`, `session_id`	Correlate AIP state to one endpoint and provider run
`start_claim`	Fence the non-idempotent external start
`delegation_id`, `parent_action_id`	Correlate a first-class delegated child
`status`, `cancel_requested`	Retain lifecycle and cancellation intent
`last_event`, `next_sequence`	Retain event progress for AIP publication
`approval_generation`, `approval_commands`	Fence provider approval continuations
`pending_approval`	Retain the redacted current approval event
`output`, `error`	Replay the terminal AIP result without starting Hermes again
`created_at`, `updated_at`	Record durable state transitions

“Durable” means durable in the configured AIP profile-state backend. It does not describe the provider's in-memory run API. A deployment that uses an ephemeral state backend cannot claim restart recovery merely because this record type exists.

The standalone Hermes host wires its prepared runtime profile-state store into the connector. It also supplies the runtime approval store, Action queue, and lifecycle store needed for delegated result verification.

Follow the operator states

State	Meaning	Replay behavior
`starting`	AIP owns the source Action, but no provider `run_id` is bound	One exclusive start claim may call Hermes
`running`	A provider `run_id` is durably bound	Re-entry observes the existing run
`waiting_for_approval`	A provider approval event is retained	Return `RequiresHuman`; wait for a new resume Action
`cancelling`	Cancellation intent is durable and stop reconciliation is active	Do not start or approve new work
`completed`	Provider completion is retained	Return the stored completed result
`failed`	A definitive provider or connector failure is retained	Return the stored failed result
`cancelled`	Provider terminal cancellation is retained	Return the stored cancelled result
`outcome_unknown`	An external mutation may have happened without durable confirmation	Return failure and never replay automatically

`completed`, `failed`, `cancelled`, and `outcome_unknown` are terminal binding states. A waiting approval is not terminal provider state, but it produces an AIP `RequiresHuman` result for the current invocation.

Read an operator result

Every operator result exposes the source binding rather than an unstructured provider body:

```
JSON
{
  "endpoint_id": "primary",
  "run_id": "run_0123456789abcdef0123456789abcdef",
  "session_id": "aip-operator-act_0123456789abcdef0123456789abcdef",
  "source_action_id": "act_0123456789abcdef0123456789abcdef",
  "delegation_id": null,
```

```
  "parent_action_id": null,
  "status": "completed",
  "last_event": "run.completed",
  "pending_approval": null,
  "provider": {
    "event": "run.completed",
    "output": "The requested case was read and summarized."
  }
}
```

The output schema requires `endpoint_id`, `session_id`, `source_action_id`, and `status`. The connector also emits the nullable correlation, approval, and provider fields shown above.

A direct objective maps terminal binding state to Completed, Failed, Cancelled, or RequiresHuman. A resume invocation returns that source outcome under the new resume Action ID, while `source_action_id` continues to identify the original binding. For first-class delegation, the native child Action result replaces the provider operator output as the business result.

Fence provider start

Hermes `POST /v1/runs` is not provider-idempotent. The connector protects it with a compare-and-set claim that contains a unique claim ID, connector instance ID, acquisition time, and expiry. The default claim window is 60 seconds.

Concurrent invocations with the same Action and fingerprint observe one binding. Only the claim winner sends the provider start. Other invocations wait for a bound `run_id` or a terminal record.

The outcomes are deliberately asymmetric:

Start outcome	Binding result
Provider `4xx`	Definitive `failed`; the run was rejected before acceptance
Success with a non-empty `run_id`	Bind the ID, clear the claim, and observe that run
Success without `run_id`	`outcome_unknown`
Transport failure or provider `5xx`	`outcome_unknown`
Claim owner disappears or expires before binding	`outcome_unknown`

Another process never steals an unbound external-start claim and sends a second start. It marks the source outcome unknown instead. Reusing the source Action ID with a different endpoint, input, or delegation request produces an idempotency collision.

The required AIP idempotency key remains important for runtime-level replay. The connector's second fence is keyed by Action ID. The provider does not enforce the forwarded idempotency header on its run-start route.

Observe events and recover by polling

Immediately after binding a new run, the connector attempts the provider SSE stream. It records every event name and next AIP sequence in profile state and publishes typed AIP stream chunks when trusted execution supplied a stream.

Provider event	AIP stream kind	Binding effect
`message.delta`	Data	Advance sequence
`reasoning.available`	Thought	Advance sequence
`tool.started`, `tool.completed`	Tool	Advance sequence
`approval.request`	PendingApproval	Increment approval generation and wait
`run.failed`	Error	Retain terminal failure

Provider event	AIP stream kind	Binding effect
`run.completed`, `run.cancelled`	Done	Retain the corresponding terminal state
Other event	Progress	Retain name and advance sequence

The default limit is 4096 events, and one SSE frame is capped at 1 MiB. The operator observation defaults to 15 minutes and uses the trusted AIP deadline when one is shorter.

If initial event subscription returns `404`, the HTTP request fails, or the stream decode fails, the connector polls `GET /v1/runs/{run_id}` instead. A replayed invocation and restart recovery also poll rather than trying to resume an already consumed provider stream.

Polling defaults to every 250 ms. It recognizes `queued`, `running`, `waiting_for_approval`, `stopping`, `completed`, `failed`, and `cancelled`. Another state fails closed as unsupported.

Resume a provider approval

An `approval.request` stores the redacted provider event, increments the source binding's approval generation, and returns an AIP `RequiresHuman` result. The operator capability itself remains high risk and AIP-approval governed.

After review, create a new operator Action with a new Action ID and idempotency key. Point it to the source Action:

```
JSON
{
  "resume": {
    "source_action_id": "act_0123456789abcdef0123456789abcdef",
    "choice": "once",
    "resolve_all": false
  }
}
```

The allowed choices are exactly `once`, `session`, `always`, and `deny`. The resume endpoint must match the source binding's endpoint. `resolve_all` applies only to pending provider approvals in that run.

The connector records each resume command inside the source binding before it contacts Hermes. The record contains the resume Action ID, canonical fingerprint, approval generation, decision, claim owner, timestamps, and one of these states:

Command state	Meaning
`claimed`	One connector invocation owns the provider approval call
`applied`	Hermes accepted the decision; exact replay returns stored progress
`rejected`	Hermes returned a definitive `4xx` before applying it
`outcome_unknown`	Hermes may have applied the decision; replay is blocked

The claim uses the same default 60-second fencing window as provider start. The ledger is capped at 256 commands per source run. Reusing a resume Action ID with changed input is rejected. A transport failure or `5xx` marks both the command and source binding outcome unknown.

This command ledger is separate from the tenant-policy AIP approval record that authorizes the resume Action. It is also separate from the provider's own in-memory approval queue.

Reconcile cancellation

Cancellation first writes `cancel_requested` to the durable source binding. It can therefore arrive before Hermes returns a `run_id`. Once the ID is bound, the connector sends `POST /v1/runs/{run_id}/stop` and polls status.

This connector path runs only while the cancelled source or resume Action is active in the AIP runtime. After an operator invocation has returned `RequiresHuman`, that invocation is no longer active. Cancelling its native AIP lifecycle record does not call the connector again and therefore does not stop the waiting Hermes run.

To refuse the pending provider operation, submit a separately governed operator resume with `choice: deny`. To request full interruption while no operator Action is active, use the separately approved `run_stop` capability with the retained `run_id`, then reconcile the operator binding as an operational exception. The public operator input has no standalone stop branch.

The default cancellation grace window is 15 seconds. A racing `completed` or `failed` state remains authoritative. Only provider-confirmed `cancelled` becomes the connector's cancelled operator result. A missing or non-terminal result at the end of the grace window becomes `connector.hermes_agent.cancellation_outcome_unknown`.

The native cancel endpoint returns an AIP `cancelled` response after an active handler's cancellation callback succeeds. That response does not expose the connector's provider reconciliation result and is not proof that Hermes reached `cancelled`. Retain provider or operator-binding evidence separately.

Cancellation does not roll back model calls, tools, messages, or external effects. A provider `stopping` receipt is progress, not terminal evidence.

Route first-class delegation

The delegated entry does not invoke the public `operator` capability with an objective. A native `DelegationRequest` selects a configured endpoint through the delegate principal:

```
TEXT
agent:hermes_operator:<endpoint_id>
```

The delegate must be an Agent principal. The connector checks the endpoint's tenant boundary before it accepts the route.

Delegation policy is fail-closed:

- `delegation_enabled` defaults to `false`;
- enabling it requires explicit capability-ID and scope prefix lists;
- `*` opts into an unrestricted dimension and should receive explicit review;
- the current chain length must be below the configured depth limit;
- durable connector-level approval verification defaults to enabled;
- any approval exemption uses a textual capability prefix and should be as

narrow as one complete capability ID.

Normal AIP authorization, capability contracts, tenant checks, and downstream approval still apply. The connector-level exemption changes only its extra delegation approval check.

For an admitted request, the connector serializes the exact child Action as `aip_call_arguments`. It renames the serialized `id` field to `action_id` and removes `identity`, because the authenticated MCP transport supplies identity. Instructions require Hermes to call `aip_call` exactly once with those arguments and forbid another state-changing tool.

Do not expose recursive `aip_delegate` as a model-facing shortcut. The external AIP runtime owns the delegation graph, depth, cycle, scope, approval, and child Action invariants.

Verify the delegated result

Hermes completion means only that the provider operator run ended. For first-class delegation, the connector then resolves the exact native child Action from the authoritative Action queue and lifecycle store.

The resolver verifies:

1. immutable Action ID, capability, input, mode, idempotency, timeout,

conversation, delegation, federation, callback, observability, compliance, approval, and transaction fields;

2. application memory context, while allowing trusted `_aip` enrichment;
3. a non-expired transport-authenticated principal with `action:write`;
4. the same expected Hermes principal in queue owner, authenticated owner, and

actor context;

5. equality between Action identity and resolved trusted identity;
6. the exact valid tenant partition;
7. agreement between queue and lifecycle results;
8. the expected child Action ID on the terminal result.

If no child execution appears, the resolver waits four poll intervals, bounded to one through five seconds, then fails with `delegated_execution_not_observed`. Once the child is observed, it may wait to the operator resolution deadline for a terminal result.

A terminal queue record without a durable result, disagreement between stores, contract drift, identity mismatch, or tenant mismatch fails closed. Hermes final text never substitutes for the missing native result.

Apply operator policy bounds

Policy field	Default	Valid bound or rule
`delegation_enabled`	`false`	Explicit opt-in
`delegation_requires_approval`	`true`	Durable approval store required when applied
`model`	None	Deployment-owned optional route
`system_instructions`	Governed built-in text	Non-empty; at most 131072 bytes
`timeout_ms`	`900000`	1000–86400000
`poll_interval_ms`	`250`	10–60000
`max_events`	`4096`	1–100000
`max_input_bytes`	`1048576`	1024–16777216
`max_delegation_depth`	`16`	1–128
`start_claim_ttl_ms`	`60000`	1000–300000
`cancel_grace_ms`	`15000`	1000–300000
Capability and scope prefixes	Empty	Must both be explicit when delegation is enabled

The connector validates this policy before an operator run. Empty or oversized prefixes are rejected. Prefix matching is textual `starts_with`, not a semantic capability or scope parser.

Handle failures

Failure	Meaning	Response
`connector.hermes_agent.operator_policy`	Policy denial, changed-input reuse, or a missing source binding	Correct policy or identity; do not retry unchanged
`connector.hermes_agent.operator_state`	Profile-state read, parse, compare-and-set, or contention failure	Protect the state backend and inspect its revision history
`operator_outcome_unknown`	Provider start may have happened before durable binding	Reconcile externally; never auto-start again
`approval_outcome_unknown`	Provider may have accepted a resume command	Inspect source state; never resend that command blindly
`cancellation_outcome_unknown`	Stop did not reach a confirmed terminal state	Treat earlier effects as uncertain and escalate
Delegated integrity or identity error	Native child state disagrees with the requested contract	Fail closed and retain both store records
Delegated execution not observed	Hermes completed without the exact child Action	Reject model prose as evidence

The connector returns terminal binding results idempotently. Temporary provider read failures can fall back to polling, but no failure turns an unknown external mutation into a safe replay.

Verify an integration

For each enabled endpoint, retain evidence that:

1. the operator manifest matches the contract and requires AIP approval;
2. concurrent replay of one source Action creates one provider run;
3. changed input under the same Action ID is rejected;
4. a lost unbound start becomes `outcome_unknown` after process replacement;
5. an approval prompt returns `RequiresHuman`, and exact resume replay sends one

provider command;

6. cancellation is called cancelled only after provider terminal evidence;
7. delegation remains disabled without both explicit allowlists;
8. the model-facing payload contains exact `aip_call` arguments without

identity;

9. the resolver rejects altered contracts, forged owners, tenant drift, and

disagreement between durable stores;

10. the profile-state, approval, Action-queue, and lifecycle backends used by

the artifact have the claimed durability.

Record exact AIP, connector-host, Hermes, policy, endpoint, tenant, credential, and storage identities with timestamps and retained results. Source tests show implemented controls; they are not evidence that a deployed artifact passed them.

Related documentation

- Hermes Agent capabilities
- Structured runs and approvals
- Delegation

- Approvals and policy
- Errors and recovery