Run a governed Hermes operator lifecycle

Use this guide to take one direct Hermes operator objective from its AIP approval to completion, a provider approval, a failure, or a controlled stop. It keeps every identity and approval decision attached to the work it governs.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/hermes-agent/guides/operator-lifecycle.md

The operator can execute models, tools, messages, code, and external network operations. Never create a replacement Action after an uncertain start, approval, or cancellation. Query and reconcile the recorded identities first.

This guide covers direct objectives and their provider-approval resumes. For first-class AIP delegation, use the operator capability reference instead.

Understand the identity ledger

One objective can acquire several related identifiers.

Identity	Owner	Lifetime and use
Source Action ID	AIP runtime	Stable identity of the original objective and operator binding
Source idempotency key	AIP runtime	Required capability-scoped replay key for that exact objective
AIP approval ID	AIP runtime	Authorizes one high-risk source or resume Action
Approval decision ID	AIP runtime	Makes one approval vote replayable without duplicating it
Hermes `run_id`	Hermes process	Addresses the volatile provider run after start is bound
Hermes `session_id`	Connector and Hermes	Deterministically `aip-operator-<source_action_id>`
Provider approval generation	Connector binding	Distinguishes successive Hermes approval prompts in one run
Resume Action ID	AIP runtime	New identity for one provider approval response
Resume idempotency key	AIP runtime	New required key for that exact resume Action

Persist the complete ledger in application state. Do not use `run_id`, an approval ID, or a resume Action ID in place of the source Action ID.

Separate the two approval loops

The lifecycle has two independent approval layers.

Layer	AIP result	What the reviewer decides	How work continues
Capability approval	`pending_approval` with an AIP `approval_id`	Whether this exact high-risk Action may execute	A verified AIP decision resumes the same queued Action

Layer	AIP result	What the reviewer decides	How work continues
Hermes provider approval	`requires_human` with output status `waiting_for_approval`	Whether pending tool work inside the bound Hermes run may proceed	A new operator resume Action carries `once`, `session`, `always`, or `deny`

An AIP approval does not answer a Hermes prompt. A Hermes choice does not authorize a high-risk AIP Action. Every resume Action passes the first layer before it can change the second.

Before you start

Confirm all of these conditions:

- the endpoint-qualified `operator` capability is admitted and visible to the

request tenant;

- the requester can invoke the capability and read its Action lifecycle;
- a transport-authenticated reviewer who differs from the requester has

tenant-policy approval authority;

- the AIP Action, approval, lifecycle, idempotency, and connector profile-state

backends have the durability claimed by the deployment;

- every downstream capability the objective may use is separately admitted and

governed;

- raw objective, provider event, tool, and result data is handled as

confidential and PII-bearing.

The capability timeout is 15 minutes. The generated AIP approval request also expires after 15 minutes. These are configured contract values, not observed availability or completion guarantees.

1. Allocate the source identity

Generate the source Action ID and idempotency key before submission. Retain them with the endpoint, tenant, requester, and exact objective input.

```
SH
export AIP_CAPABILITY_ID='cap:hermes_agent:primary:operator'
export AIP_SOURCE_ACTION_ID='act_0123456789abcdef0123456789abcdef'
export AIP_SOURCE_KEY='operator-objective-case-example-2030-01'
```

One source Action ID identifies one exact endpoint and input. Reusing it with a changed objective, context, constraint, endpoint, or delegation shape is an idempotency collision.

The connector uses the source ID to create the stable Hermes session ID and to key the durable operator binding. The provider `POST /v1/runs` route is not idempotent; the binding's start claim is the second replay fence.

2. Prepare a bounded objective

Create `operator-objective.json` with one outcome and explicit limits:

```
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,
    "maximum_records": 1
  }
}
```

The input must select exactly one of `objective` or `resume`. A direct objective can also carry arbitrary JSON `context`, up to 256 non-empty capability hints, and an object of constraints.

`candidate_capabilities` is a model hint. It cannot admit a capability, grant authority, bypass approval, widen tenant access, or override a schema. Keep untrusted instructions inside data fields and make deployment policy the authority boundary.

3. Submit the source Action asynchronously

Submit the objective with its stable identity and required key:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$AIP_CAPABILITY_ID" \
  --action-id "$AIP_SOURCE_ACTION_ID" \
  --idempotency-key "$AIP_SOURCE_KEY" \
  --mode async \
  --input @operator-objective.json
```

Without a prior verified decision, the high-risk Action does not reach Hermes. The expected response is `pending_approval`, with essential evidence shaped like this:

```
JSON
{
  "action_id": "act_0123456789abcdef0123456789abcdef",
  "status": "pending_approval",
  "output": {
    "requires_human_approval": true,
    "approval_id": "appr_0123456789abcdef0123456789abcdef",
    "capability_id": "cap:hermes_agent:primary:operator"
  },
  "error": {
    "code": "policy.human_approval_required"
  }
}
```

Save the exact `approval_id`. Do not retry the Action to make approval appear; the original Action is already queued behind that record.

4. Review the AIP approval record

Read the generated record with the reviewer's authenticated identity:

```
SH
export AIP_SOURCE_APPROVAL_ID='appr_0123456789abcdef0123456789abcdef'

cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_APPROVER_TOKEN_FILE" \
  approval get "$AIP_URL" "$AIP_SOURCE_APPROVAL_ID" \
  --include-action-status \
  --include-receipts
```

Verify all of these fields before deciding:

1. `action_id` equals the retained source Action ID;
2. `capability_id` equals the endpoint-qualified operator capability;
3. requester, subject, operator, tenant, and expiry match the intended work;
4. the input-snapshot evidence hash covers the submitted objective;

5. the immutable `policy_hash` is present and unchanged;
6. reason, risk, side effects, constraints, and evidence support the decision;
7. the authenticated reviewer is admitted by the required authority policy.

Do not copy raw input into logs merely to review it. Use the bounded approval view and an explicitly authorized evidence export when sensitive detail is required.

5. Decide the AIP approval

Create `source-approval-decision.json`. Copy the exact approval and policy hash from the record, use a current timestamp, and allocate a stable decision ID:

```
JSON
{
  "approval_id": "appr_0123456789abcdef0123456789abcdef",
  "decision": "approved",
  "approver": {
    "id": "human:operator-reviewer",
    "kind": "human"
  },
  "decided_at": "2030-01-15T10:00:00Z",
  "reason": "Read-only objective and constraints match the reviewed case.",
  "decision_id": "decision_operator_source_001",
  "policy_hash": "replace_with_exact_policy_hash"
}
```

The `approver` must match the transport-authenticated principal. The runtime resolves authority itself and records the authority path. The request already contains its input-snapshot and policy-decision evidence; a non-empty decision reason satisfies the remaining operator requirement.

Submit the decision:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_APPROVER_TOKEN_FILE" \
  approval decide "$AIP_URL" @source-approval-decision.json
```

An approved terminal decision leases and resumes the same queued source Action. Do not submit the objective again and do not attach the decision to a replacement Action. The decision request can remain open while the resumed operator executes; observe the Action from a separate consumer.

6. Follow the source Action

Follow events and chunks under the source Action ID:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action events "$AIP_URL" \
  "$AIP_SOURCE_ACTION_ID" \
  --include-chunks \
  --follow
```

The connector publishes provider events under the source Action ID and keeps one sequence across later resumes.

Provider event	AIP chunk kind	Application action
`message.delta`	Data	Append visible text from the structured delta
`reasoning.available`	Thought	Retain only under the deployment's reasoning policy
`tool.started`, `tool.completed`	Tool	Correlate tool progress; do not infer the external effect
`approval.request`	PendingApproval	Stop presenting progress as autonomous; review the prompt

Provider event	AIP chunk kind	Application action
`run.failed`	Error	Preserve the typed failure and uncertain effects
`run.completed`, `run.cancelled`	Done	Wait for the corresponding Action result
Any other event	Progress	Preserve the event name without inventing semantics

Initial observation tries the provider SSE stream. A reconnect, source replay, or connector restart polls the bound `run_id` instead of resuming that provider stream. AIP chunks already persisted remain readable; unobserved provider frames are not reconstructed.

7. Classify the first operator result

Read the source status with its result and recorded chunks:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action status "$AIP_URL" \
  "$AIP_SOURCE_ACTION_ID" \
  --include-result \
  --include-receipts \
  --include-chunks
```

Use both the AIP status and operator output.

AIP result	Operator output status	Decision
`completed`	`completed`	Accept only after required downstream effects are independently confirmed
`failed`	`failed`	Inspect the typed error and retain partial effects
`cancelled`	`cancelled`	Treat prior effects as potentially committed
`requires_human`	`waiting_for_approval`	Review the provider prompt and create a new resume Action
`pending_approval`	No provider run yet	Complete the AIP capability approval; do not create a provider resume
`failed`	`outcome_unknown`	Freeze replay and begin manual reconciliation

Accumulated text, a Done chunk, model prose, or a closed connection is not a terminal business decision by itself.

8. Review the Hermes provider prompt

A provider approval result retains the source binding and a redacted prompt:

```
JSON
{
  "action_id": "act_0123456789abcdef0123456789abcdef",
  "status": "requires_human",
  "output": {
    "endpoint_id": "primary",
    "run_id": "run_0123456789abcdef0123456789abcdef",
    "session_id": "aip-operator-act_0123456789abcdef0123456789abcdef",
    "source_action_id": "act_0123456789abcdef0123456789abcdef",
    "status": "waiting_for_approval",
    "last_event": "approval.request",
    "pending_approval": {
      "event": "approval.request",
      "description": "Review the redacted provider operation"
    }
  }
}
```

Retain the `run_id`, but continue to address the operator binding by `source_action_id`. Review the actual redacted command, tool, pattern, scope, and consequence supplied by the deployment.

Choose the narrowest provider decision:

Choice	Provider meaning	Use
`once`	Allow this pending operation without persisting an allow rule	Default for one reviewed operation
`session`	Allow the matching pattern for this Hermes gateway session	Only after reviewing the session scope
`always`	Persist the matching allow rule in Hermes configuration	Exceptional administrative decision
`deny`	Reject this pending operation	Use when the operation is not authorized or needed

Leave `resolve_all` false unless every currently queued provider prompt in the run has been reviewed. AIP approval of the resume does not review those prompts for you.

9. Prepare a separately identified resume

Allocate a new Action ID and key for one exact provider decision:

```
SH
export AIP_RESUME_ACTION_ID='act_abcdef0123456789abcdef0123456789'
export AIP_RESUME_KEY='operator-resume-case-example-2030-01'
```

Create `operator-resume.json`:

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

Use `deny` instead of `once` to refuse the provider operation. Never reuse the source Action ID for a resume. Never reuse a prior resume ID, key, approval ID, or decision ID for a later provider approval generation.

The resume capability endpoint must be the same endpoint that owns the source binding. A different endpoint fails before it can answer the provider prompt.

10. Approve and observe the resume Action

Submit the resume as another asynchronous operator Action:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$AIP_CAPABILITY_ID" \
  --action-id "$AIP_RESUME_ACTION_ID" \
  --idempotency-key "$AIP_RESUME_KEY" \
  --mode async \
  --input @operator-resume.json
```

This Action first returns its own AIP `pending_approval`. Repeat steps 4 and 5 with the new approval ID, exact new policy hash, and a new decision ID. Review the resume input itself: the decision authorizes sending that exact provider choice to the exact source run.

After AIP approval, the connector records a fenced provider-command entry before calling Hermes. A definitive provider `4xx` rejects the command. A transport failure, provider `5xx`, or lost command owner makes the provider approval outcome unknown and blocks replay.

Continue reading operator chunks from the source Action ID. Read settlement from the current resume Action ID:

```
SH
cargo run --locked -p aipctl -- \
```

```
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action status "$AIP_URL" \
  "$AIP_RESUME_ACTION_ID" \
  --include-result \
  --include-receipts \
  --wait-ms 60000
```

If Hermes raises another provider prompt, allocate another resume Action and repeat this step. Do not mutate or replay the preceding resume.

11. Settle from the latest invocation result

After a resume, the connector deliberately separates stream and result identity.

Evidence	Identity	Meaning
Operator chunks	Original source Action ID	Ordered provider progress across the complete bound run
Current ActionResult `action_id`	Latest resume Action ID	Result of this separately governed invocation
Output `source_action_id`	Original source Action ID	Binding whose provider state produced the result
Output `run_id`	Hermes provider run	Volatile provider correlation only

The earlier source ActionResult remains `requires_human`; the resume does not rewrite it to completed. Do not wait for that old result to change. Accept the latest resume result only when its `output.source_action_id` matches the source ledger and its output status is terminal.

A representative terminal correlation is:

```
JSON
{
  "action_id": "act_abcdef0123456789abcdef0123456789",
  "status": "completed",
  "output": {
    "source_action_id": "act_0123456789abcdef0123456789abcdef",
    "run_id": "run_0123456789abcdef0123456789abcdef",
    "status": "completed",
    "last_event": "run.completed"
  }
}
```

Provider completion proves the governed Hermes run ended. Confirm important downstream AIP Actions and external effects from their own authoritative lifecycle records rather than from operator prose.

12. Recover a disconnect or process restart

On application delivery loss:

1. keep the source and current invocation IDs unchanged;
2. query status for the current invocation before considering any submission;
3. read bounded events and chunks under the source ID from the last native

cursor;

4. read the current AIP approval record if its result is `pending_approval`;
5. read the current provider prompt if its result is `requires_human`;
6. accept only a terminal result whose source correlation matches the ledger.

Closing an `action events --follow` connection does not cancel the AIP Action or Hermes request. Resume the native delivery route; do not reconnect directly to the provider SSE URL.

The operator binding can survive only when the configured profile-state backend is process-durable. Hermes `run_id`, event queue, active task, and provider approval queue are process memory. A provider `404` after restart or retention loss is an evidence gap, not proof that the run never executed.

Do not create a replacement objective when the provider handle is missing. The connector intentionally refuses to repeat an external start whose outcome may already exist.

13. Cancel according to the current phase

Native cancellation has different effects depending on whether an operator Action is active.

Phase	Action to cancel or decide	Provider effect
Waiting for initial AIP approval	Cancel the source Action	No provider run has started
Source Action actively executing	Cancel the source Action	Active connector callback records intent and attempts provider stop
Provider returned `requires_human`	Submit a governed resume with `deny`	Rejects the pending provider operation and returns control to Hermes
Resume Action waiting for AIP approval	Cancel that resume Action	Prevents that provider decision; the source run remains waiting
Resume Action actively executing	Cancel the resume Action	Connector maps its input to the source binding and attempts provider stop
No operator Action is active, but full stop is required	Invoke separately approved `run_stop` with retained `run_id`	Requests provider interruption; binding needs operational reconciliation

Cancel an active Action explicitly:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action cancel "$AIP_URL" \
  "$AIP_RESUME_ACTION_ID" \
  --reason 'The requester withdrew the governed objective'
```

The runtime calls the connector only for an active Action. After an invocation has returned `requires_human`, cancelling that inactive native record does not call Hermes. The public operator input has no standalone stop branch.

For an active invocation, the connector records cancellation intent, maps a resume back to the source binding, sends provider stop after a `run_id` is bound, and polls within the configured grace. The provider binding can settle completed, failed, cancelled, or outcome unknown.

The native cancel response is an AIP cancellation acknowledgement. It does not expose that provider classification and is not proof that Hermes stopped. Neither path rolls back model, tool, message, code, or network effects.

14. Freeze unsafe unknown outcomes

Error or observation	What may have happened	Required decision
`operator_outcome_unknown`	Hermes may have accepted a start before `run_id` was bound	Freeze the source identity; reconcile provider and profile state manually
`approval_outcome_unknown`	Hermes may have applied a resume command	Freeze that resume and source; never resend the choice blindly
`cancellation_outcome_unknown`	Stop did not reach a confirmed provider terminal state	Treat provider state and effects as uncertain
Provider `404` after a bound run	Provider process state may be lost or expired	Preserve the gap; do not infer that no run existed
`operator_start_rejected` on provider `4xx`	Hermes definitively rejected the start	Correct the cause before creating a separately reviewed Action
Initial AIP approval denied or expired	No provider run started for that source Action	Keep the terminal record; do not reuse its identifiers
Resume AIP approval denied or expired	No provider choice was sent by that resume; the source run can remain waiting	Keep both records; create a new reviewed resume only for a still-current prompt

Unknown is a terminal safety classification, not a temporary retry hint. Keep the input fingerprint, endpoint, source and resume IDs, approval records, binding revision, provider diagnostics, and timestamps for reconciliation.

15. Verify the integration

Before accepting the client, use controlled evidence to verify that it:

1. allocates the source Action ID and required key before submission;
2. persists the exact objective and endpoint with that identity;
3. distinguishes AIP `pending_approval` from provider `requires_human`;
4. validates approval ID, policy hash, evidence, authority, and expiry;
5. lets an approved decision resume the same source Action;
6. follows operator chunks under the source Action ID;
7. treats candidate capabilities only as hints;
8. creates a new Action, key, approval, and decision for every provider resume;
9. leaves `resolve_all` false unless all queued prompts were reviewed;
10. reads terminal settlement from the latest resume Action when one exists;
11. preserves `source_action_id`, `run_id`, and current invocation ID separately;
12. recovers delivery without submitting replacement work;
13. cancels only with phase-aware expectations and verifies provider effects;
14. freezes all three unknown-outcome classes without automatic replay;
15. redacts objectives, provider prompts, tool data, results, and diagnostics.

This checklist is source-aligned integration evidence. It is not live provider, latency, storage-durability, cancellation, or production qualification.

Related documentation

- Operator and delegation capability
- Structured runs and approvals
- Use native AIP

- Approvals and policy
- Actions and sessions
- Errors and recovery