Observe and recover AIP work

Use this guide when an AIP action is delayed, interrupted, or has an uncertain external outcome. You will establish whether the process can make progress, read the durable action and delivery records, choose a recovery action, and verify th

SECTION	Deploy and Operate
TYPE	Guide
PROTOCOL	AIP 1.0
SOURCE	docs/guides/observe-and-recover.md

This procedure applies to the product-neutral `getaip-server` runtime and its optional connector fleet at source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. Product-specific failures still require the runbook for the selected Cal.diy, Hermes Agent, Chatwoot, Dify, CrewAI, or Twenty connector.

Preserve the incident boundary

Before changing configuration or restarting a process, record:

- the action, capability, session, correlation, approval, transaction, and

delegation identifiers that are available;

- the authenticated principal and tenant used for the operational read;
- the original idempotency key and any provider operation reference;
- the first observed failure time and the last known successful state update;
- the gateway, connector type, instance, version, and replica involved, when

the action used the fleet;

- the current readiness response and a metrics snapshot.

Pause new mutations for the affected tenant or capability when the blast radius is not yet known. Keep reads and evidence collection available.

Do not submit a replacement action, edit a lease, delete an outbox record, or change an action's route assignment to make the incident disappear. Logs, traces, and metrics explain behavior; durable protocol and runtime records own the current action state.

1. Check liveness and readiness

Query the deployment through its controlled observability boundary:

```sh
export AIP_BASE_URL=https://aip.example.com

curl --fail --silent --show-error "$AIP_BASE_URL/health"
curl --fail --silent --show-error "$AIP_BASE_URL/ready"
curl --fail --silent --show-error "$AIP_BASE_URL/metrics" >aip-metrics.prom
```

`/health` establishes only that the HTTP process is alive. `/ready` returns HTTP `200` when the core daemon is ready and `503` otherwise. Interpret the readiness body by owner:

Readiness area	What to inspect	Decision
`gateway`	`ready`, storage, and local connector checks	Restore storage read/write access or a required local connector before accepting traffic
`supervisor`	`runtime_worker_running`, `worker_cycles`, `last_worker_error`, and `last_worker_success_at`	A live HTTP process with a stopped or failing worker cannot recover queued work
NATS	`nats_required` and `nats_listener_running` together	A stopped listener blocks readiness only when NATS is required
`modules`	Every module marked `required`	Resolve the required module failure; optional modules do not control top-level readiness
`fleet`	Worker state, consecutive failures, last error, last success, and summary	Diagnose registry maintenance and route supply separately from core readiness

Gateway readiness performs a runtime-storage read/write check and probes registered local connectors. Remote fleet replicas are not local gateway connectors. The optional `fleet` snapshot is included in the response, but it does not participate in the top-level `ready` calculation at this revision. An HTTP `200` can therefore coexist with no ready remote replica for a particular tenant capability.

Do not interpret `nats_listener_running: false` as a fault when `nats_required` is false. Do not restart solely because one counter has stopped; compare the worker state, last error, last success time, and the durable work described below.

2. Read the durable action view

Use an identity authorized for the same tenant and action owner. Prefer the client token file rather than placing a bearer token in process arguments:

```
SH
export AIP_TOKEN_FILE=/run/secrets/aip-native-token

getaip \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action list "$AIP_BASE_URL" \
  --state queued \
  --limit 100
```

Change the state or identity filters only when they narrow the incident. Once you have the action ID, request the complete operational view:

```
SH
export ACTION_ID=act_example

getaip \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action status "$AIP_BASE_URL" "$ACTION_ID" \
  --include-result \
  --include-chunks \
  --include-receipts
```

Read the fields in this order:

1. confirm `action_id`, `capability_id`, and the tenant-visible ownership;
2. compare the stable `state` with the backend `queued_state`;
3. inspect `result_status`, `approval_id`, `transaction_id`, and `delegation_id`;
4. compare `started_at`, `updated_at`, and `completed_at`;
5. inspect the redacted `retry` and `lease` metadata;
6. use the result, chunks, receipts, and typed links only after the identity and

state match the incident.

An empty result does not mean the action was never accepted. A queue record, approval, transaction, event, or callback may still own durable progress.

3. Interpret the lifecycle state

The stable action view defines these states:

State	Meaning for recovery	Safe next action
`unknown`	The authorized runtime has no matching view	Recheck the endpoint, tenant, owner, and exact ID before considering a new action
`accepted`	The protocol state allows accepted work before queue entry	Inspect events and audit evidence; the reviewed queue projection does not normally produce this state
`queued`	Work is waiting, delayed for retry, or available after a lease	Inspect retry time, lease expiry, worker progress, and last error
`running`	A worker owns the current attempt	Preserve the lease boundary and wait or follow events unless the lease has expired
`streaming`	Ordered chunks exist before the terminal result	Resume from the last processed cursor and continue to wait for a terminal view
`pending_approval`	The same action is parked behind durable governance	Inspect the approval record; do not submit a replacement action
`cancelling`	The protocol state allows cancellation in progress	Treat the external outcome as unresolved until a terminal record exists
`cancelled`	Runtime work ended as cancelled	Verify whether any provider effect preceded cancellation
`completed`	A completed result is durable	Verify receipts, delivery, and provider evidence; do not rerun for a missing callback
`failed`	The action has a terminal failed result	Classify the typed error and provider outcome before deciding on new work
`expired`	Retention or queue policy expired the action	Preserve remaining evidence and decide through the owning policy
`dead_lettered`	Automatic action recovery has stopped	Preserve the original record and require an explicit, reviewed recovery decision

`accepted` and `cancelling` are part of the stable lifecycle contract. The reviewed runtime projection does not currently map its stored queue and result records to those two states. Do not create an alert that assumes every defined state must be observed in this implementation.

4. Follow retry, lease, and event evidence

For a queued or running action, use the read model rather than database rows:

- `retry` identifies attempts, retry policy, the next attempt time, and the

last redacted error when those values exist;

- `lease` identifies the current worker and lease expiry without exposing the

internal fencing operation;

- `updated_at` shows the last durable state transition;
- `queued_state` preserves the backend queue label when it differs from the

stable lifecycle view.

An unexpired lease belongs to its current worker. An expired recoverable lease can be claimed by the runtime worker using a new fencing token. Never extend, release, or transfer a lease with a manual database update. A

stale worker is not allowed to settle an action after losing its lease.

Read action-scoped events with an opaque cursor:

```
SH
getaip \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action events "$AIP_BASE_URL" "$ACTION_ID" \
  --limit 100 \
  --include-chunks
```

Persist the returned cursor after successfully processing the page. Supply it with `--cursor` for the next request, or use `--follow` for the HTTP event stream. Consumers must tolerate replay after reconnect and deduplicate by the event identity or ordered chunk sequence.

5. Resolve approval and transaction branches

When the action exposes `approval_id`, read the approval and its related action state:

```
SH
export APPROVAL_ID=appr_example

getaip \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  approval get "$AIP_BASE_URL" "$APPROVAL_ID" \
  --include-action-status \
  --include-receipts
```

`pending` waits for an authorized decision. `approved` permits the frozen action to continue through the durable transition path. `denied`, `expired`, and `revoked` are terminal approval decisions. Do not copy an approval result into a new action or alter the frozen payload to bypass its policy hash.

When the action exposes `transaction_id`, read the transaction separately:

```
SH
export TRANSACTION_ID=txn_example

getaip \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  transaction get "$AIP_BASE_URL" \
  --transaction-id "$TRANSACTION_ID" \
  --include-result \
  --include-receipts
```

For `outcome_unknown`, preserve the provider operation reference and reconciliation cursor. Allow reconciliation to establish whether the provider accepted the original operation. Do not submit another commit. Compensation is a separate governed action and is available only when the capability contract declares it.

6. Separate execution from delivery

A completed action and its callback delivery have independent durable states. Use an authorized native HTTP client to list `GET /aip/v1/callback-deliveries` by action, status, profile, target, or tenant. Read one delivery at `GET /aip/v1/callback-deliveries/{delivery_id}`.

Inspect the delivery policy, status, attempt history, `next_attempt_at`, `leased_by`, `lease_expires_at`, `last_error`, `dead_letter_reason`, and optional receipt chain. The stable callback states are `pending`, `running`, `delivered`, `failed`, and `dead_lettered`. The reviewed dispatcher returns failed attempts to `pending` while retry budget remains and records exhausted delivery as `dead_lettered`; it does not normally publish `failed` as its final stored state.

Pending delivery or a running delivery with an expired lease is recoverable by the runtime worker. A dead-lettered callback is terminal for automatic delivery, and this revision exposes no public callback replay command. Restore the target first, preserve the record, and escalate through a reviewed delivery-recovery procedure. Never rerun the action merely to recreate its callback.

Remote delegation also uses a durable outbox with `pending`, `leased`, `delivered`, and `dead_lettered` states. At this revision, the action view can expose `delegation_id`, but `getaip-server` has no public HTTP or `getaip` query for the delegation outbox record itself. Use action events, audit evidence, and peer observations; do not edit the outbox database to force a retry.

7. Diagnose a fleet route without repinning work

When the action targets a standalone connector host, compare three views:

1. the gateway's `fleet` readiness snapshot and maintenance error;
2. tenant-scoped capability discovery for new route availability;
3. the authorized registry view of the action's durable assignment, together

with action events and provider evidence.

The public `ActionStatus` view does not expose the complete route assignment at this revision. Use a controlled registry diagnostic or retained signed connector request to identify the pinned route. Current capability discovery describes routes for new work and cannot prove where an existing action ran.

Ready-replica and active-assignment metrics describe fleet supply and activity. They do not identify the outcome of one action. An already accepted action can remain pinned to its original connector type, version, instance, and replica while discovery sends new actions elsewhere.

Keep the original replica and its durable state available until every assigned mutation is terminal or reconciled. Do not change the route of an ambiguous action to a healthy replica; that can repeat an external effect. Follow the connector-host lifecycle for drain, lease loss, and replacement.

8. Choose the recovery action

Observation	Recovery	Do not
Runtime worker is down or repeatedly errors	Restore its dependency, then restart the same reviewed process against the same durable store	Start a clean runtime with empty state
Queued action has no live lease	Let the worker claim the existing action after readiness returns	Create a new action ID
Running action has an unexpired lease	Wait, follow events, and inspect the provider timeout boundary	Force a second worker to execute it
Running action has an expired lease	Let durable recovery claim it with a new fenced lease	Rewrite the lease owner or expiry
Retry is scheduled	Preserve the original action and wait until `next_attempt_at`	Bypass backoff with a duplicate submission
Approval is pending	Complete or expire the existing approval through its authority path	Change the frozen input or fabricate approval
Transaction is `outcome_unknown`	Reconcile by provider operation reference and cursor	Retry commit or compensate before outcome is known
Callback remains pending after target recovery	Let the callback worker recover its existing delivery	Rerun the completed action
Action or delivery is dead-lettered	Preserve evidence and require an explicit policy decision	Delete the record to clear an alert
Fleet has no route for new work	Keep the binding disabled or pause affected traffic until a ready route exists	Repin an already accepted ambiguous action

The daemon's worker recovery covers pending approvals, recoverable transactions, queued or running asynchronous actions, running delegations, and recoverable callback deliveries. Recovery reuses the existing durable identities. It does not make an unknown provider outcome safe to execute again.

9. Use metrics for health, not action truth

The reviewed source emits useful health and capacity signals, including:

Signal group	Examples	Use
Core progress	`aip_daemon_ready`, `aip_runtime_worker_up`, `aip_runtime_worker_cycles_total`	Detect a daemon or worker that cannot make progress
Optional NATS	`aip_nats_listener_up`	Correlate required listener state with readiness
Work budgets	`aip_runtime_callback_capacity`, `aip_runtime_callback_available`, `aip_runtime_reconciliation_capacity`, `aip_runtime_reconciliation_available`	Detect exhausted callback or reconciliation concurrency
Remote dispatch	`aip_connector_remote_active`, `aip_connector_remote_queued`, `aip_connector_remote_queued_bytes`	Observe in-process remote scheduling pressure
Fleet	`aip_connector_fleet_ready_replicas`, `aip_connector_fleet_active_assignments`, `aip_connector_fleet_maintenance_failures_total`	Observe route supply, pinned work, and registry maintenance failure

This revision does not emit dedicated gauges for the durable action queue, individual leases, the delegation outbox, or execution checkpoints. Execution-checkpoint observers are process-local notifications and default to a no-op observer; they are not AIP wire records. Use the action, transaction, callback, event, receipt, and audit read models for durable state. The metrics reference owns the complete metric catalog and label contract.

10. Verify recovery

Recovery is complete only when all applicable evidence agrees:

1. `/ready` returns `200`, the runtime worker advances, and the relevant

fleet or local connector view has no blocking error;

2. the original action reaches the expected durable state without a replacement

action ID;

3. approval, transaction, delegation, and callback records agree with the

action;

4. an `outcome_unknown` transaction has a reconciled terminal outcome;
5. stream or event consumption resumes from the stored cursor;
6. provider-side evidence identifies one intended effect;
7. receipts and authorized audit records preserve the incident and recovery

sequence.

If these views disagree, keep the mutation boundary paused and treat the incident as unresolved. Read errors and retry decisions before authorizing any new state-changing action.

Related documentation

- Operator runbooks
- Transactions and compensation
- Production deployment