# Run a CrewAI crew and stream events

Use this guide to execute one admitted CrewAI crew, render its ordered progress, and reach a verified terminal result. The same flow recovers after an application disconnect without starting a second crew run.

| | |
|---|---|
| SECTION | Connectors |
| TYPE | Connector |
| PROTOCOL | AIP 1.0 |
| SOURCE | docs/connectors/crewai/guides/run-a-crew-and-stream-events.md |

A delivery stream does not own execution. Preserve the original AIP Action ID and read durable state before deciding whether any new mutation is necessary.

## Before you begin

Obtain these values from the authorized tenant context:

- one discovered `cap:crewai:<crew_id>` capability;
- the admitted host, sidecar, registry, and crew revisions;
- one immutable input object of at most 1 MiB;
- one unique AIP Action ID and tenant-scoped idempotency key;
- approval authority for the reviewed model, tool, data, and network effects;
- a durable application record for event progress and terminal evidence.

Confirm that the host is ready. Readiness proves authenticated access to a durable sidecar exposing the admitted crew and operations. It does not execute the crew or validate its external providers.

## 1. Discover the exact capability

Use authenticated discovery under the intended tenant. Select the base capability whose binding metadata contains the reviewed crew ID and CrewAI upstream revision.

The run ID has no `:run` suffix:

```
TEXT
cap:crewai:support-primary
```

Stop if discovery contains an unexpected crew, connector instance, artifact, tenant, or operation set. Do not construct a replacement ID from a Python registry key or sidecar address.

## 2. Freeze one execution intent

Allocate the Action ID and key before dispatch:

```
SH
export AIP_CAPABILITY_ID='cap:crewai:support-primary'
export AIP_ACTION_ID='act_crewai_support_001'
export AIP_IDEMPOTENCY_KEY='crewai-support-001-v1'
```

Keep these values with:

- tenant and authenticated principal;
- complete input digest;
- crew and registry revisions;
- model, tool, memory, and knowledge dependency identities;
- expected timeout and cost envelope;
- approval record and policy hash.

Never change input or key while retaining the Action ID. The sidecar hashes all of that material and rejects a conflicting reuse.

## 3. Prepare bounded input

Write only crew kickoff fields to `run-input.json`:

```
JSON
{
  "case_id": "CASE-1042",
  "objective": "Prepare a reviewed incident summary",
  "constraints": {
    "max_findings": 5,
    "include_personal_data": false
  }
}
```

The input permits at most 512 top-level properties and 1 MiB of encoded JSON. It cannot select the crew, Python registry, sidecar, provider origin, or credential.

Validate deployment-owned model and tool inputs before approval. A valid JSON shape does not make tool execution or generated content trustworthy.

## 4. Submit the stable Action

Use the normal native AIP route, not the private sidecar API:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$AIP_CAPABILITY_ID" \
  --action-id "$AIP_ACTION_ID" \
  --idempotency-key "$AIP_IDEMPOTENCY_KEY" \
  --input @run-input.json
```

Expected first state is either pending approval or a terminal policy denial. The connector contract requires tenant-policy approval for run.

Persist the response before continuing. Do not create another Action while the first intent is pending.

## 5. Approve the immutable request

Review the pending Action and confirm:

1. capability, crew, tenant, instance, and artifact are expected;
2. input digest matches `run-input.json`;
3. model, tool, storage, and network authorities are within policy;
4. sensitive input and output handling is defined;

5. timeout, token, provider, and tool budgets are acceptable;
6. the same Action ID and key will resume after approval.

Submit one signed approval decision through the tenant's authorized path. Approval evidence contains the reason, input snapshot, and policy decision.

An approval authorizes this Action. It does not make CrewAI or external tool effects retry-safe.

# 6. Follow durable AIP events

Follow the original Action from a separate consumer:

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

The equivalent native route is:

```
TEXT
GET /aip/v1/actions/{action_id}/events?include_chunks=true&follow=true
```

Persist the application consumer position only after applying an event. A closed follow connection is a delivery interruption, not proof that the Action stopped.

Reconnect with the same Action ID and the last confirmed AIP event position. Do not resubmit run to recreate a display stream.

# 7. Decode CrewAI chunks

The sidecar normally emits `started`, zero or more `chunk` records, and one terminal `completed`, `failed`, or `cancelled` record. A factory failure can reach `failed` before `started`.

Every stored record carries a zero-based provider sequence and JSON data.

The connector maps chunks as follows:

| Sidecar event | Channel | AIP chunk kind | Application action |
|---|---|---|---|
| `started` | Any | Progress | Mark the job admitted and executing |
| `chunk` | `tools` | Tool | Update the correlated tool activity |
| `chunk` | `llm` or `messages` | Data | Append structured output data |
| `chunk` | Other or absent | Progress | Update non-terminal progress |
| `completed` | Any | Done | Candidate terminal success |
| `failed` | Any | Error | Preserve available failure details and classify uncertainty |
| `cancelled` | Any | Done | Preserve the sidecar cancellation record |

When event data contains non-empty `content` or `data.chunk`, the connector can also project a text message part. Use structured event data for tool and model identity instead of merging every part into one string.

The Rust connector requires strict provider sequence. It rejects malformed UTF-8, invalid JSON, a missing sequence, gaps, oversized frames, excessive events, and streams that close without a terminal event.

The pinned parser checks its 1 MiB buffer before extracting delimited frames. A coalesced transport chunk can fail that check even when its individual frames are smaller.

## 8. Keep the two cursor domains separate

The application can observe two durable event surfaces:

| Cursor domain | Owner | Purpose |
| --- | --- | --- |
| AIP Action event position | AIP runtime | Resume delivery of Action state and `StreamChunk` records |
| CrewAI job cursor | Sidecar journal | Replay sidecar `started`, `chunk`, and terminal events |

The numbers can differ because AIP records approval, lifecycle, and result events in addition to connector chunks.

Never pass an AIP follow position as the `cursor` input to the CrewAI events capability. Store both with explicit names.

## 9. Replay sidecar events after interruption

Use a new read Action against:

```
TEXT
cap:crewai:support-primary:events
```

Target the original run and the last processed sidecar position:

```
JSON
{
  "run_action_id": "act_crewai_support_001",
  "cursor": 4
}
```

A blocking read returns the requested events and `next_cursor`. A typed stream waits for later events until the target job is terminal.

Advance the saved sidecar cursor to `next_cursor` only after processing the complete slice. Repeating the read is retry-safe.

A cursor beyond the current event count is rejected. Return to the last confirmed cursor instead of guessing a later value.

## 10. Read status independently

Use a separate read Action against:

```
TEXT
cap:crewai:support-primary:status
```

Its input contains the original Action ID:

```
JSON
{
  "run_action_id": "act_crewai_support_001"
}
```

Status reports crew, operation, state, event count, and completed output when available. It is the concise durable view; events explain how that state was reached.

After a disconnect, read both AIP Action state and sidecar status. A mismatch requires reconciliation rather than a replacement run.

## 11. Settle a completed result

Accept success only after retaining:

1. the final AIP `ActionResult` with `completed` status;
2. the sidecar terminal `completed` event for the same original Action ID;
3. the serialized CrewAI output within the expected schema and data policy;
4. required evidence for every tool or external provider effect;
5. the final AIP and sidecar consumer positions.

A Done chunk is not the final AIP result. A text fragment is not proof that tools completed or that output validation passed.

The final output can contain confidential or personal data. Redact general logs and retain raw evidence only in an authorized store.

## 12. Request cancellation when required

Use the explicit cancel capability with a new Action and key:

```
JSON
{
  "run_action_id": "act_crewai_support_001",
  "reason": "Operator stopped the approved run"
}
```

The cancel Action requires its own approval. The sidecar closes the streaming object when supported, cancels its asyncio task, and persists a terminal `cancelled` record.

Observe status and events after the request. Confirmed sidecar cancellation acknowledges that route and record transition. It does not prove provider termination or rollback of earlier model and tool effects.

If streaming validation fails after admission, the connector also attempts remote cancellation. Failure to confirm that attempt becomes an unconfirmed cancellation.

## 13. Recover an uncertain outcome

| Observation | Decision |
| --- | --- |
| AIP follow connection closed | Reconnect to the original Action event stream |
| Sidecar stream closed without terminal event | Read status and replay events from the saved sidecar cursor |
| Run remains `running` | Continue observation or submit a governed cancel Action |
| Run fails after provider work starts | Reconcile every possible model and tool effect, even when the event has no uncertainty field |
| Run is `cancelled` | Verify completed effects separately; no rollback is implied |
| Sidecar restarted during run | Treat the restored failed record as uncertain and do not rerun automatically |
| Original job is absent | Check crew, instance, journal ownership, retention, and Action ID |
| Repeated start conflicts | Restore the exact original input and key; do not change the request |

Create a new run only when the original intent is terminal, its external effects are understood, and policy authorizes another distinct intent.

## Completion checklist

- The capability came from authenticated tenant discovery.
- One Action ID and key represented the complete run intent.
- Approval covered crew, input, models, tools, data, and budgets.
- AIP and sidecar cursor positions were stored separately.

- Every stream sequence was processed once at the application boundary.
- Final AIP result and sidecar terminal state agreed.
- Cancellation claims distinguished stopped execution from prior effects.
- Uncertain effects were reconciled before any new mutation.

## Related documentation

- CrewAI run lifecycle and replay reference
- CrewAI capability index
- Run the CrewAI quickstart
- Use native AIP
- Errors and retry decisions