CrewAI run lifecycle and replay capabilities

These six operations start CrewAI work or inspect and control its durable sidecar job. Use one stable AIP Action ID and idempotency key for each start, then target that Action ID from separate status, event, or cancel Actions.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/crewai/capabilities/run-lifecycle-and-replay.md

Task replay is a new CrewAI execution from a CrewAI task ID. It is not replay of an AIP envelope or event cursor.

Compare the six operations

Operation	Input focus	Result focus	Stream	Advertised cancel
`run`	Crew kickoff input object	Serialized Crew output	Yes	Yes
`status`	Original run Action ID	Durable job record	No	No
`events`	Original run Action ID and cursor	Ordered journal slice or stream	Yes	No
`cancel`	Original run Action ID and optional reason	Updated durable job record	No	No
`batch_run`	Bounded array of kickoff inputs	Ordered result array and count	Yes	Yes
`replay`	CrewAI task ID and optional inputs	Serialized replay output	Yes	Yes

`run`, `batch_run`, and `replay` are medium-risk Agent mutations. `cancel` is a medium-risk Tool mutation. `status` and `events` are low-risk Tool reads.

Preserve the two Action identities

A startable operation uses its own AIP Action ID as the sidecar job ID. The sidecar stores that value before scheduling CrewAI work.

Control operations are separate AIP Actions. Their `run_action_id` input names the original job:

```
TEXT
act_run_001       run or batch_run Action and durable sidecar job
act_status_001    status Action targeting act_run_001
act_events_001    events Action targeting act_run_001
act_cancel_001    cancel Action targeting act_run_001
```

Keep the original ID after a timeout, disconnect, restart, or unknown outcome. Creating another start Action changes the durable identity and can duplicate model or tool effects.

Run one crew

The base capability has no `:run` suffix:

```
TEXT
cap:crewai:<crew_id>
```

Its input is an arbitrary JSON object with at most 512 top-level properties. The encoded input may not exceed 1 MiB.

The sidecar enforces the byte limit after receiving the request. The pinned Rust connector has no serialized-request size preflight.

```
JSON
{
  "case_id": "CASE-1042",
  "priority": "high",
  "facts": {
    "region": "emea"
  }
}
```

The sidecar creates a fresh Crew instance, enables CrewAI streaming, and calls `akickoff(inputs=...)` when available. It falls back to `kickoff_async(inputs=...)` at the frozen revision.

The completed output is a JSON object. The published schema identifies these CrewAI fields and permits additional serialized fields:

Field	Shape
`raw`	String result
`json_dict`	Object or `null`
`pydantic`	Object or `null`
`tasks_output`	Array of task outputs
`token_usage`	Usage object

The sidecar converts other supported CrewAI values into JSON-safe data and bounds the complete output to 4 MiB.

Read durable status

Invoke `cap:crewai::status` with:

```
JSON
{
  "run_action_id": "act_run_001"
}
```

The identifier must contain 1 through 256 bytes, no control characters, and no slash. It must identify a job for the same configured crew.

The result contains:

```
JSON
{
  "action_id": "act_run_001",
  "crew_id": "support-primary",
  "operation": "run",
  "status": "completed",
  "event_count": 4,
  "output": {}
}
```

`status` can be `running`, `completed`, `failed`, or `cancelled`. `output` is present only after successful completion and can otherwise be `null`.

Status is retry-safe and does not require an idempotency key. It does not prove that every external tool or model effect is reconciled.

Replay durable events

Invoke `cap:crewai::events` with the original run ID and an optional cursor:

```
JSON
{
  "run_action_id": "act_run_001",
  "cursor": 2
}
```

The published cursor range is `0` through `100000`. The sidecar also requires the cursor to be no greater than the current event count. An omitted cursor starts at zero.

A blocking read returns:

```
JSON
{
  "action_id": "act_run_001",
  "cursor": 2,
  "next_cursor": 4,
  "status": "completed",
  "events": [
    {
      "event": "chunk",
      "sequence": 2,
      "data": {}
    },
    {
      "event": "completed",
      "sequence": 3,
      "data": {}
    }
  ]
}
```

The stored sequence is zero-based and monotonic. Persist `next_cursor` only after processing the returned events successfully.

Streaming event replay starts at the supplied provider cursor. The connector validates every sequence and rebases emitted AIP chunk sequence to zero for the new reader.

Cancelling the read detaches that reader. It does not cancel the target job.

Request cancellation

Invoke `cap:crewai::cancel` with:

```
JSON
{
  "run_action_id": "act_run_001",
  "reason": "Operator stopped the governed run"
}
```

The published schema and sidecar model limit the optional reason to 2,048 characters. The pinned Rust connector forwards the string without its own control-character check.

The cancel Action requires approval and its own idempotency key.

The reviewed sidecar accepts cancellation for any running job. It closes a streaming result when possible, cancels the asyncio task, records a terminal `cancelled` event, and persists the updated record.

A terminal job is returned unchanged. Replay, train, and test run provider methods in worker threads, so cancelling their asyncio task does not stop the underlying call. The durable record can say `cancelled` while that call continues.

For a streamed mutation, Rust can return `remote_stop_confirmed: true` after the sidecar route succeeds. Interpret that field as a sidecar acknowledgement, not confirmation from CrewAI or its providers.

The `cancel` operation itself does not advertise runtime cancellation support. It is one bounded request that targets another job.

Run a bounded batch

Invoke `cap:crewai::batch_run` with one through 100 input objects:

```
JSON
{
  "inputs": [
    {
      "case_id": "CASE-1042"
    },
    {
      "case_id": "CASE-1043"
    }
  ]
}
```

Each item has at most 512 top-level properties in the published schema. The complete encoded operation input still shares the 1 MiB limit.

The sidecar disables CrewAI streaming for each item. It prefers `akickoff_for_each`, falls back to `kickoff_for_each_async`, then runs `kickoff_for_each` in a worker thread.

It records one durable job for the entire batch, not one journal entry per item. Cancellation cannot prove provider stop when the worker-thread fallback is active.

The completed output has this shape:

```
JSON
{
  "results": [
    {},
    {}
  ],
  "count": 2
}
```

One approval and one idempotency identity cover the complete batch. Review the worst permitted item effect before approval.

Replay from a CrewAI task

Invoke `cap:crewai::replay` with a CrewAI task ID:

```
JSON
{
  "task_id": "task_support_triage",
  "inputs": {
    "case_id": "CASE-1042"
  }
}
```

`task_id` contains 1 through 256 bytes without control characters. `inputs` is optional and, when present, must be an object with at most 512 top-level properties in the published schema.

The sidecar calls `crew.replay(task_id, inputs)` in a worker thread. The operation creates a new durable job under the replay Action ID and can produce new model or tool effects.

Replay advertises cancellation support, but its synchronous CrewAI call runs in a worker thread. Cancellation ends the waiter and records `cancelled`; it does not establish provider termination.

Understand the durable start fence

Every startable operation requires an idempotency key. The key contains at most 512 bytes and no control characters.

Before scheduling CrewAI, the sidecar hashes and persists:

- crew ID;
- Action ID;
- operation;
- complete input;
- optional timeout;
- idempotency key.

An exact repeated Action returns the existing job. Reusing the Action ID with different material returns a conflict.

This fence prevents a previously admitted ID from disappearing after an external effect. It cannot make model, tool, or external-provider behavior transactional.

Interpret events and terminal outcomes

One job can record `started`, `chunk`, `completed`, `failed`, or `cancelled`. Every event contains its name, zero-based sequence, and JSON data.

The connector maps tool-channel chunks to Tool, LLM and message chunks to Data, other chunks to Progress, completion and cancellation to Done, and failure to Error.

A streaming response must contain a terminal event. Invalid UTF-8, malformed JSON, a missing sequence, a gap, an oversized frame, excessive events, an oversized complete response, or early close fails validation.

At this revision, the connector checks the accumulated buffer before extracting delimited frames. One transport chunk above 1 MiB can therefore be rejected even when it contains several individually bounded frames.

If a streamed mutation fails after admission, the connector attempts sidecar cancellation for all five advertised operations. A successful route response does not prove provider stop for worker-thread execution.

Apply approval, retry, and recovery rules

Operation	Approval	Key	Connector retry	Recovery anchor
`run`	Required	Required	No	Original Action ID, then status and events
`status`	No	Optional	Yes	Repeat the same read
`events`	No	Optional	Yes	Resume from last processed cursor
`cancel`	Required	Required	No	Status and events for target job
`batch_run`	Required	Required	No	Original Action ID, then status and events
`replay`	Required	Required	No	Original replay Action ID, then status and events

Do not create a replacement mutation because a client timed out. Read the durable job first and reconcile every uncertain external effect.

Respect bounds and failure decisions

Boundary	Decision
Input exceeds 1 MiB	Reduce the complete JSON input before invocation
Key is absent or invalid	Preserve the intent and supply one stable bounded key
Target run is absent	Verify crew, original Action ID, instance, and retained journal
Cursor is beyond event count	Return to the last confirmed cursor
Batch has zero or over 100 items	Split the reviewed intent into bounded batches
Replay task ID is unknown	Correct the CrewAI task identity; do not substitute an AIP Action ID
Thread-backed job reports `cancelled`	Treat provider completion as potentially continuing and reconcile its effects
Journal is full	Archive eligible terminal records only after the retention window
Restored job is failed with uncertain outcome	Reconcile provider state before any new mutation

Related documentation

- CrewAI capability index
- CrewAI connector overview
- CrewAI connector configuration
- Actions and sessions
- Errors and retry decisions