CrewAI connector

The CrewAI connector exposes deployment-admitted crews as governed AIP capabilities. Use it when a tenant must run, observe, cancel, train, test, query, or reset an identified crew through a durable boundary.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/crewai/README.md

The connector is not a generic Python execution gateway. Crew factories, operation policy, sidecar identity, credentials, storage, and provider dependencies remain deployment-owned.

Decide whether it fits

Requirement	Fit
Run one pre-admitted CrewAI crew	Yes, through its base capability
Observe durable status and replay ordered events	Yes
Run a bounded input batch or replay from a CrewAI task	Yes, when explicitly allowed
Train or test a crew	Yes, under high or medium risk controls
Query crew knowledge without executing the task graph	Yes
Reset an explicit memory domain	Yes, as a high-risk mutation
Load an arbitrary module or select a crew from Action input	No
Request cancellation for every streamed mutation	The contract advertises it, but provider stop is not guaranteed
Share one sidecar journal across active replicas	No
Use AIP transactions, reconciliation, or rollback	No
Prove compatibility with any CrewAI installation	No; claims bind one exact artifact and upstream revision

Choose another boundary when callers need dynamic code loading, unrestricted Python objects, active-active access to one journal, or automatic rollback.

Understand the two-component boundary

The production integration has two separately identified components:

1. `aip-host-crewai`, a Rust process that owns AIP identity, admission,

routing, policy contracts, durable host state, and sidecar authentication;

2. `aip-crewai-sidecar`, a Python process that loads approved crew factories,

executes CrewAI, journals jobs, and publishes bounded results and events.

The Rust host never imports CrewAI. The Python sidecar never accepts a signed AIP envelope or chooses tenant routing.

Boundary	Owner
Tenant discovery, approval, Action lifecycle, and route	AIP platform
Crew descriptors and advertised operation policy	Rust connector instance
Python registry and crew construction	Sidecar deployment
Model, tool, memory, knowledge, and provider configuration	Deployment-owned crew code
Job journal, event replay, and training artifacts	One sidecar state volume

The host and sidecar policies must agree. Health fails when the sidecar omits a configured crew or an operation admitted by a host descriptor.

Admit crew identities explicitly

Each Rust descriptor has a stable ID, name, optional description, and explicit operation allowlist. IDs must be unique and valid inside an AIP capability ID.

The sidecar registry uses `module:attribute` syntax. The attribute must be a non-empty mapping or a zero-argument callable returning one.

Registry values are zero-argument factories or cloneable Crew instances. A fresh factory result or deep clone is used for each run to avoid cross-request mutation of CrewAI objects.

The registry can contain additional crews, but every host-admitted crew must be present. A caller cannot choose a Python import path.

Read capability IDs

The `run` operation retains the historical base ID. Every other allowed operation uses a suffix.

```
TEXT
cap:crewai:<crew_id>
cap:crewai:<crew_id>:status
cap:crewai:<crew_id>:events
cap:crewai:<crew_id>:cancel
cap:crewai:<crew_id>:batch_run
cap:crewai:<crew_id>:replay
cap:crewai:<crew_id>:train
cap:crewai:<crew_id>:test
cap:crewai:<crew_id>:knowledge_query
cap:crewai:<crew_id>:memory_reset
```

A crew publishes only its allowed operations. Legacy descriptors without an allowlist receive `run`, `status`, `events`, and `cancel`.

There is no universal manifest count. A fully enabled crew publishes ten capabilities; a legacy-policy crew publishes four.

Compare the ten operations

Operation	Kind	Risk	Mutation	Streaming	Advertised cancellation
`run`	Agent	Medium	Yes	Yes	Yes
`status`	Tool	Low	No	No	No
`events`	Tool	Low	No	Yes	No
`cancel`	Tool	Medium	Yes	No	No
`batch_run`	Agent	Medium	Yes	Yes	Yes
`replay`	Agent	Medium	Yes	Yes	Yes
`train`	Workflow	High	Yes	Yes	Yes
`test`	Workflow	Medium	Yes	Yes	Yes
`knowledge_query`	Tool	Low	No	No	No
`memory_reset`	Tool	High	Yes	No	No

The published contract marks all five streamed mutations as cancellable. Replay, training, and testing still execute synchronous CrewAI methods in worker threads, so sidecar cancellation does not prove that provider work stopped.

Apply execution controls

Surface	Approval	Idempotency key	Retry support
Mutating operation	Required	Required	None
`knowledge_query`	Not required	Required	Safe
`status` and `events`	Not required	Optional	Safe

`knowledge_query` is provider-read-only but creates a durable sidecar job. That job boundary requires a key even though the capability is retry-safe.

The reviewed Rust error mapper nevertheless marks every concrete connector failure `retryable: false`. Treat the contract classification as capability metadata, not permission to retry a failed request automatically.

Keys use tenant scope, input-hash revalidation, and a 24-hour contract TTL. Mutation approval uses tenant policy and a 15-minute decision TTL.

Every operation treats data as confidential, PII-bearing, and redaction-required. Mutations publish read, write, external-network, and code- execution side effects.

The connector publishes no transaction, reconciliation, compensation, or provider rollback support.

Preserve durable job identity

The sidecar uses the AIP Action ID as its job identity. It hashes crew, operation, input, timeout, and idempotency key before scheduling provider work.

Reusing an Action ID with matching material returns the existing record. Reusing it with different material returns a conflict.

The idempotency fence is persisted before CrewAI starts. A crash can therefore leave an uncertain result, but it must not make an admitted Action disappear.

At restart, any restored `running` job becomes a terminal failed record with `uncertain_outcome: true`. The sidecar never silently restarts it.

Only one active sidecar process may own a journal. The file lock and state volume form a process fence, not a distributed active-active coordinator.

Treat streams and cancellation precisely

The connector validates strict event sequence, normalizes sidecar SSE to AIP chunks, and requires a terminal `completed`, `cancelled`, or `failed` event.

The connector bounds each undelimited SSE frame to 1 MiB, the complete response to its configured limit, and one stream to 100,000 events.

All five streamed mutations can call the sidecar cancellation route. Cancelling an event replay only detaches the read-only stream.

For replay, train, and test, the route cancels the asyncio waiter and persists a terminal `cancelled` record while the worker thread can continue. The Rust connector may then report `remote_stop_confirmed: true`; this confirms only the sidecar response, not provider termination.

Respect the principal limits

Boundary	Reviewed value
Sidecar-validated operation input	1 MiB
Sidecar output	4 MiB
Connector response, default	64 MiB
Connector response, maximum	256 MiB
Sidecar event, default	256 KiB
Sidecar events per run, default	1,000
Connector SSE events	100,000
Execution timeout, default	600,000 ms
Execution timeout, maximum	3,600,000 ms

The sidecar also bounds concurrency, retained runs, journal size, checkpoint frequency, query items, batch inputs, and training iterations.

Bind the compatibility baseline

Component	Reviewed identity
AIP source	`97be86e9efedf07ecf1783b03800f683f107fb04`
Connector profile	`aip.connector.crewai.v1`
Sidecar package	`aip-crewai-sidecar` `1.0.0`
Default product sidecar	locked `crewai`, `crewai-cli`, and `crewai-core` `1.15.5`
Optional source-overlay image	CrewAI distribution from source version `1.15.2`, installed without dependencies
CrewAI source	`bfa652a7be8637562cc9b0833f75d927a64552d1`
Python range	`>=3.10,<3.14`

The controlled product Compose path builds the default lockfile-only image. A separate `Dockerfile.upstream` first resolves the same lockfile, then reinstalls only the CrewAI distribution from the pinned source tree without dependencies.

The source-overlay image therefore has a mixed package baseline. Keep default and source-overlay image digests distinct; a source commit or package version alone cannot identify either artifact.

Know what is not guaranteed

The reviewed source does not establish:

- compatibility with arbitrary CrewAI versions, registries, models, or tools;
- sandboxing of deployment-authored Crew code inside the Python process;
- exactly-once effects in models, tools, memory, or external systems;
- cancellation of synchronous replay, training, or evaluation work;
- active-active sidecars over one journal or training directory;
- automatic provider-state reconciliation after uncertain failure;
- production qualification for the current artifact.

Crew factories and their tools can execute deployment-owned code. Isolate the sidecar as a privileged workload and grant only intended provider access.

Choose the next path

Use the concise legacy page for the earlier high-level contract. Read the AIP execution model before integrating Actions, and use the fleet guide for the standalone topology.

Implement retries only after reading the global error rules. The upstream- baseline page explains how framework source identity is bounded.

Related documentation

- Legacy CrewAI connector overview
- How AIP works
- Deploy the connector fleet
- Errors and retry decisions
- Connector upstream baselines