Qualify the CrewAI connector

Use this procedure to evaluate one exact CrewAI host and sidecar pair. The result applies only to the recorded artifacts, configuration, topology, CrewAI package set, crew code, and executed cases.

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

Source inspection is a prerequisite, not a qualification result. Record every case as `PASS`, `FAIL`, or `NOT RUN`; never convert missing evidence into a passing claim.

Define the qualification claim

State one bounded claim before execution:

> The identified CrewAI connector artifacts preserve AIP identity, durable job admission, ordered events, fail-closed routing, and recovery for the executed crew and failure cases in the recorded topology.

Do not expand this claim to external model quality, arbitrary tools, every CrewAI feature, active-active sidecar state, provider exactly-once behavior, or unexecuted operations.

Freeze every identity

Record these values in the run manifest:

Identity	Required value
AIP source	Full commit and clean source-tree digest
Rust host	Image digest, OCI labels, SBOM, provenance, signature, and binary version
Python sidecar	Image digest, package version, lockfile digest, SBOM, provenance, and signature
CrewAI	Installed `crewai`, `crewai-core`, and `crewai-cli` versions
Method source	Exact upstream source revision used for contract verification
Crew code	Registry module, crew factory source digest, and admitted descriptor digest
State	Fresh volume identity, filesystem, mount policy, and retention settings
Fleet	Type, version, instance, replica, tenant, assignment, zone, and artifact digest
Credentials	Non-secret provider, revision, scope, and rotation-policy references
Runner	Operating system, architecture, Docker engine, Compose, Rust, Python, and `uv` versions
Time	Run ID, UTC start and completion, and reviewer identity

Use immutable image digests during execution. A rebuilt image requires a new qualification even when source labels are unchanged.

Prepare a clean evidence directory

Create a new owner-only directory for each run. The pinned fleet verifier rejects an existing evidence directory and records diagnostics on both success and failure.

Prepare these evidence groups:

```
TEXT
crewai-qualification/<run-id>/
■■■ identity/
■■■ source-tests/
■■■ sidecar-tests/
■■■ images/
■■■ runtime-evidence/
■■■ failure-matrix.jsonl
■■■ registry/
■■■ state/
■■■ logs/
■■■ summary.json
■■■ limitations.md
```

Store secret-bearing logs under restricted access. Publish only redacted derivatives and cryptographic digests.

Gate 1: verify source and contracts

At the reviewed AIP revision, verify:

- the operation enum contains exactly ten unique operations;
- seven operations create durable sidecar jobs;
- `status`, `events`, and `cancel` are control operations;
- every descriptor allowlist maps to a unique capability ID;
- idempotency is required for all operations except status and events;
- streaming and cancellation flags match the implementation paths;
- the sidecar and Rust operation sets are identical;
- the pinned CrewAI method signatures still match adapter calls;
- public configuration and capability assets match generated values.

Fail this gate on any drift. Do not continue with a documentation or manifest override that hides code disagreement.

Gate 2: run Rust connector tests

The package test gate is:

```
SH
cargo test --locked -p aip-connector-crewai
```

Retain command, environment, exit status, and complete output. Confirm the six ordinary tests cover:

1. stable Action identity across run and cancel;
2. one frozen job route and replay headers for seven startable operations;
3. streaming contract advertisement;
4. unique ten-operation catalogue and idempotency metadata;
5. required SSE ordering metadata;
6. preservation of cancellation reason.

One exact-sidecar test is ignored by the ordinary package run. Record it as `NOT RUN` until the qualification sidecar is supplied and the ignored test is executed deliberately.

Mock-sidecar tests establish Rust mapping behavior. They do not establish real CrewAI execution or journal durability.

Gate 3: run sidecar runtime tests

Resolve the frozen sidecar environment with its test extra, then execute the pinned test directory:

```
SH
uv sync --frozen --extra test
uv run pytest
```

The eleven async cases cover:

- idempotent ordered streaming;
- bounded durable stream checkpoints;
- durable admission for every product operation;
- idempotency before provider admission;
- stream closure and terminal cancellation;
- terminal replay and restart uncertainty;
- exclusive journal ownership;
- persistence before task scheduling;
- retained-run and event fail-closed bounds;
- authenticated HTTP with durable state;
- rejection of unsafe non-loopback startup.

Retain the resolved package inventory and lockfile digest with the output. Passing unit cases do not qualify a built container image.

Gate 4: verify both images

Apply the pinned image verification policy to the host and sidecar:

Check	Host expectation	Sidecar expectation
Runtime user	`10001:10001`	`10001:10001`
Entrypoint	`/usr/local/bin/aip-connector-host`	`aip-crewai-sidecar`
Component label	`aip-host-crewai`	`aip-crewai-sidecar`
Source label	Exact AIP revision	Exact AIP revision
Version label	Exact release version	Exact release version
Runtime probe	`--help` in hardened container	Import installed sidecar package
Retained evidence	Inspect, full history, and SPDX SBOM	Inspect, full history, and SPDX SBOM

Also record the three installed CrewAI distribution versions. The default artifact should resolve the complete package set to `1.15.5` at this revision.

Verify signatures and provenance against the admitted trust policy. The pinned image helper records metadata and SBOMs but does not itself prove registry signature policy.

Gate 5: qualify deterministic real CrewAI execution

The pinned qualification registry constructs a real CrewAI `Agent`, `Task`, and sequential `Crew`. It uses a custom `BaseLLM` that emits real CrewAI stream events without network access.

The LLM returns `case triaged` and reports five tokens. This case proves the CrewAI library path, task graph, event bus, stream consumption, terminal output, and Rust-side mapping without depending on a model provider.

Execute and retain these CrewAI native calls through the fleet gateway:

Case	Capability	Required observation
Seed status	`cap:crewai:support-qualification:status`	Preloaded terminal Action is readable
Governed run	`cap:crewai:support-qualification`	Approval and completed real-CrewAI output
Run status	`cap:crewai:support-qualification:status`	New Action identity and completed state
Run events	`cap:crewai:support-qualification:events`	Ordered retained events from cursor zero

The product qualification script runs these among fifteen native calls across five product connectors. Retain the per-operation responses, stderr files, approval decision, operation ledger, and product summary.

This deterministic case does not contact an external model, tool, knowledge store, evaluator, or telemetry service. Record those surfaces as `NOT RUN` unless separate exact-provider cases execute them.

Gate 6: verify fleet identity and isolation

Require:

- one ready CrewAI replica with a live lease;
- exactly ten admitted CrewAI capabilities;
- manifest and artifact digests matching admission;
- one independently owned CrewAI runtime database;
- all five required runtime tables present;
- the correct tenant, account, version, instance, replica, and zone;
- discovery through the central gateway only for the admitted tenant;
- a private authenticated sidecar route.

The pinned five-product topology totals 353 capabilities. That fleet total is context, not a CrewAI invariant.

Gate 7: verify restart and failure behavior

Execute every case with before, fault, after, and recovery evidence.

Case	Required result
Graceful host restart	Replica goes offline, returns ready, and seeded status completes
Host `SIGKILL`	Expired route fails closed, host returns healthy, and status completes
Sidecar graceful restart	Real-run status and events remain readable from durable state
Sidecar `SIGKILL`	Host becomes unroutable, status fails closed, then recovers after sidecar return
Shared PostgreSQL outage	CrewAI request fails closed, storage recovers, lease returns, and status completes
Gateway failover	The surviving gateway routes through the same admitted CrewAI replica

Preserve the sidecar volume across restart cases. A restart using a new empty volume is not a durability test.

The pinned failure matrix reads status for an already completed Action. It does not prove recovery of a running thread-backed train, replay, test, or batch job. Record those cases as `NOT RUN` unless added explicitly.

Gate 8: add operation-specific cases

The public capability catalogue includes more operations than the pinned fleet script exercises end to end. Add bounded cases for:

Operation	Required evidence
`batch_run`	Input count, ordered outputs, fallback path identity, and partial-effect analysis
`replay`	Exact task ID, input override, thread behavior, and retained result
`train`	Protected artifact path, digest, iterations, timeout, and uncertainty handling
`test`	Evaluator identity, iterations, provider calls, and retained evaluation artifact
`knowledge_query`	Query bounds, source identity, result count, scores, and read-only claim
`memory_reset`	Selected domain, approval, before and after evidence, and irreversibility
`cancel`	Async run and each thread-backed operation with provider-stop limitations

Use an external provider only when credentials, budgets, data policy, and retention have explicit approval. Keep deterministic and provider-live results separate.

Gate 9: inspect the evidence bundle

The retained bundle should include:

- source and dependency identities;
- image inspect, history, SBOM, signature, and provenance;
- Rust and Python test outputs;
- admission package and trust decision;
- registry baseline and final state;
- runtime database ownership and migration state;
- Compose process and Docker event timelines;
- per-case native requests and responses;
- approval decisions and operation ledger;
- sidecar state digests before and after restart;
- redacted timestamped logs;
- failure-matrix outcomes;
- final summary and explicit limitations.

Validate that each referenced file exists and calculate a manifest digest for the bundle. Keep raw restricted evidence separate from its publishable index.

Decide the result

Result	Rule
`PASS`	Every required gate passed for the exact identities and every limitation is explicit
`FAIL`	A required case failed, evidence conflicts, identity drifted, or cleanup destroyed required evidence
`NOT RUN`	The case did not execute or lacks sufficient evidence

A qualification can pass with documented out-of-scope cases only when the claim excludes them. It cannot pass by silently omitting a required case.

Publish the exact artifact digests, topology, executed case list, result, evidence-manifest digest, and limitations. Never publish unredacted credentials or governed provider content.

Clean up without losing proof

After the evidence bundle is sealed:

1. revoke qualification credentials and assignments;
2. stop the isolated topology;
3. remove only run-scoped containers, networks, and volumes;
4. retain required journal and artifact snapshots in protected evidence;
5. verify evidence hashes after transfer;
6. record cleanup status separately from the qualification result.

Cleanup failure does not rewrite a passed case. Record it as an operational failure requiring remediation.

Related documentation

- CrewAI connector overview
- CrewAI capability index
- Deploy the CrewAI connector
- CrewAI health, observability, and recovery
- Conformance and qualification