Upgrade a connector to the frozen SDK

Use this guide to move an existing Rust connector from legacy connector or direct ActionHandler execution to the FrozenConnector boundary. It is for connector developers who need to preserve published capability behavior while adopting trus

SECTION	Connectors
TYPE	Guide
PROTOCOL	AIP 1.0
SOURCE	docs/guides/upgrade-connector-sdk.md

This migration does not change AIP 1.0 wire semantics. It also does not create a standalone host, admit an artifact into the connector registry, or qualify a provider deployment. Those are separate steps after the SDK migration passes.

The procedure reflects source revision `97be86e9efedf07ecf1783b03800f683f107fb04`.

Use this procedure for an existing connector

Apply this migration when a connector already implements some combination of `Connector`, `CapabilityProviderConnector`, `OutboundConnector`, `InboundConnector`, or `ActionHandler`, but production execution does not yet go through `FrozenConnectorHandler`.

Do not use it to:

- design a new provider surface;
- move an already frozen connector from a bundled daemon to the fleet;
- change capability IDs, schemas, or side-effect contracts as part of an SDK

cleanup;

- claim new streaming, cancellation, retry, transaction, approval, or

credential behavior without implementing and testing it.

If the provider boundary is still undefined, start with Build a connector.

Prepare the migration boundary

You need:

- the exact source and provider revisions for the current connector;
- permission to inspect its deployment configuration and secret flow;
- current deterministic, conformance, and live-provider evidence;
- an isolated account or provider stub for regression tests;
- authority to change connector registration and roll it back;
- retained copies of the previous manifest, schemas, and artifact identity.

Do not expose secret values while collecting the baseline. Record secret kinds, opaque references, scope, and rotation behavior instead.

The highest-risk input is the current execution path. Trace it from gateway or module registration to the provider call. Identify whether any path:

- accepts actor, tenant, credential, approval, or transaction authority from an

action payload;

- invokes `ActionHandler::handle` without an execution context;
- drops idempotency, deadline, cancellation, or redaction state;
- writes to a second lifecycle or idempotency store;
- converts a provider timeout into a retryable success path;
- serializes credentials or full provider payloads in an error.

Block migration rollout until each such path is removed, isolated, or recorded as an explicit compatibility-only boundary.

Capture a contract baseline

Serialize the current manifest under the exact configuration class you intend to migrate. Retain the canonical manifest digest and a stable inventory of:

Baseline	Compare after migration
Connector and provider identity	Same connector family and account boundary
Capability IDs and kinds	No missing, renamed, or newly callable operation
Input and output schemas	Same accepted and returned data shapes
Side effects and risk	No weakened mutation or data classification
Idempotency and retry	Same key requirement, scope, collision, and safety
Completion behavior	Same synchronous, asynchronous, or streaming claim
Cancellation	Same provider-reaching behavior
Approval	Same policy and retained evidence
Transactions	Same plan, commit, reconciliation, and compensation behavior
Credentials	Same required handle and scopes without secret material
Ingress	Same signature, account, timestamp, replay, and durability boundary

A byte-identical manifest is a useful migration target when the existing contract is accurate. If the baseline overclaims behavior, correct the contract as a separately reviewed change and record why its digest changed.

Map legacy paths to frozen operations

Keep `Connector` as the base discovery and health contract. Keep `CapabilityProviderConnector` when the gateway, tests, or compatibility code needs the capability list independently.

Move production execution into these frozen operations:

Existing behavior	Frozen destination
Normal provider invocation	`invoke_typed`
Dry run or transaction plan	`plan_typed`
Prepared mutation commit	`commit_typed`

Existing behavior	Frozen destination
Governed compensation	`compensate_typed`
Provider-reaching cancellation	`cancel_typed`
Unknown-outcome status recovery	`reconcile_typed`
Terminal result or provider event emission	`emit_typed`
Provider event ingestion under trusted authority	`ingest_typed`

Every optional operation already returns `connector.operation_unsupported`. Leave that default in place until the provider path exists. A successful no-op is not an acceptable replacement.

The `FrozenConnectorHandler` chooses the typed operation from the action transaction mode:

Transaction mode	Called operation
`dry_run` or `plan`	`plan_typed`
`commit`	`commit_typed`
`compensate`	`compensate_typed`
`reconcile`	`reconcile_typed`
`execute`, `rollback_not_supported`, or no transaction	`invoke_typed`

Contextual cancellation uses `cancel_typed`. Reconciliation also requires durable transaction context and a provider operation ID; the adapter rejects a request that lacks either.

Move authority into trusted execution context

Each typed operation receives an `ActionExecutionContext`. It is a non-serializable runtime value containing:

- the transport-authenticated actor and verified tenant;
- an opaque credential handle;
- an absolute deadline and cooperative cancellation token;
- an owned idempotency reservation;
- verified approval evidence;
- transaction state and a durable provider-operation checkpoint publisher;
- a provider-effect checkpoint publisher;
- the incremental stream publisher;
- trusted trace identifiers and the redaction policy.

Use these fields for authorization, account scope, credentials, lifecycle, and output handling. Do not reconstruct them from `Action.identity`, input, metadata, model output, or legacy process globals.

`ConnectorContext::from_execution` is a narrow migration helper. It projects only the tenant ID, authenticated principal ID, authentication issuer, and trace ID. It does not preserve the credential handle, deadline, idempotency reservation, approval evidence, transaction state, cancellation token, checkpoint publishers, stream publisher, or redaction policy.

Use that projection only for an audited legacy helper that needs those four non-secret values. Do not pass it through an old invocation path and treat the result as frozen execution.

Declare implementation support per capability

Implement `FrozenConnector::implementation_support` from the admitted operation, not from a connector-wide constant. A connector whose configuration enables only part of the provider surface should return support only for that part.

Set:

- `invocation` when the capability has a typed provider path;
- `cancellation` only when cancellation reaches the provider operation;
- `streaming` only when chunks use the supplied stream publisher;
- `retry` only when classification and downstream idempotency match the

contract;

- `transaction` only when plan and commit exist;
- `reconciliation` only when uncertain outcomes can be resolved;
- `compensation` only for a separately governed compensation action;
- `approval` only when verified evidence and resume behavior are enforced;
- `credentials` only when a deployment credential handle is resolved.

Manifest admission compares this map with each capability contract. It reports `implementation.missing` when a required callable capability has no implementation claim and `implementation.claim_unsupported` when the contract advertises behavior that the support map does not implement.

Do not solve a mismatch by setting every support flag to `true`. Correct the implementation or narrow the contract.

Convert errors and secrets

Return `ConnectorFailure` from typed operations. Preserve the provider request ID, provider operation reference, remote status, retry delay, and unknown-outcome decision when available. Keep the message and structured details bounded and redacted.

Audit each legacy error mapping:

Legacy behavior	Migration decision
String-only provider error	Add a stable namespaced code, category, source, and operation
Every timeout marked retryable	Split pre-dispatch failure from possible provider commit
Mutation timeout retried	Mark uncertain and add reconciliation or fail closed
Local cancellation reported as provider cancellation	Keep outcome conservative until the provider confirms
Full response included in diagnostics	Retain bounded redacted fields and correlation IDs only
Unsupported operation returns success	Restore the explicit unsupported failure

Wrap in-memory credential bytes in `ConnectorSecret` and expose them only at the provider request boundary. Keep opaque credential references in protocol and registry data. Verify that debug output, manifests, action state, errors, receipts, logs, traces, and evidence cannot serialize the raw value.

Switch to the frozen adapter

For an embedded gateway, replace the legacy outbound registration path with `register_frozen_connector`. The frozen registration path discovers the manifest, creates one `FrozenConnectorHandler` per non-resource capability, admits the manifest with those handlers, and registers the connector.

The adapter provides two fail-closed checks:

1. `ActionHandler::handle` without trusted context returns an authorization

error.

2. A handler bound to one capability rejects an action for a different

capability.

Do not register the connector itself as the production `ActionHandler` after cutover. A connector may temporarily retain older trait implementations for tests or known compatibility callers, but those paths should not receive new production traffic.

For a standalone connector host, the common host constructs the same per-capability frozen handlers from the discovered manifest. The host also verifies the configured gateway and pinned route before creating trusted execution context.

Test the migration

Run the old and new paths against the same controlled provider behavior before switching registration. Compare:

- canonical manifest and schema digests;
- successful outputs and terminal statuses;
- provider requests and idempotency keys;
- errors, retry delays, and unknown-outcome flags;
- cancellation and streaming behavior;
- approval and transaction state transitions;
- webhook acceptance, replay rejection, and durable event publication;
- redacted logs, traces, receipts, and retained evidence.

Add focused negative tests for:

- context-free handler invocation;
- a capability sent to the wrong frozen handler;
- contract and support-map disagreement;
- expired deadline and cooperative cancellation;
- missing approval or credential context;
- duplicate idempotency keys with changed input;
- commit interruption before and after a provider operation checkpoint;
- restart with pending or uncertain work;
- credential rotation while a route pins the previous revision.

Implement a `ConnectorConformanceDriver` and run every scenario implied by the manifest and support map. Record an unsupported feature as not applicable, not passed. Retain the exact source revision, artifact identity, topology, procedure, time, and result for any conformance or qualification claim.

Cut over one bounded route

Use a reversible rollout:

1. freeze the approved baseline manifest and evidence;

2. deploy the migrated artifact without changing capability bindings;
3. select one non-production tenant or local registration path;
4. switch that path from the legacy handler to the frozen adapter;
5. verify discovery before allowing an action;
6. run a non-destructive or explicitly approved action;
7. exercise one failure and one restart path;
8. compare provider requests, runtime state, and retained evidence;
9. expand only after the migration result is accepted.

Do not send the same mutation through old and new handlers as a comparison. Use provider fixtures, read-only operations, or distinct idempotency and account boundaries.

Hand the connector to fleet packaging

A connector that passes the frozen SDK migration is ready for a separate deployment conversion, not yet for fleet traffic. The next work item is to:

- compose a standalone host around the connector;
- build and identify an immutable host artifact;
- generate the manifest and implementation support from that artifact;
- collect the seven admission evidence families;
- create connector type, version, instance, replica, policy, and tenant binding

records;

- verify and apply a signed admission package;
- qualify the exact artifact through an admitted route.

Follow the connector-building procedure linked earlier for that complete boundary. Keep bundled-to-fleet deployment changes outside the SDK migration diff so execution regressions and topology regressions remain distinguishable.

Verify the upgraded connector

The SDK migration is complete only when:

- published capability behavior is unchanged or every correction is separately

approved;

- each callable capability has an accurate implementation support record;
- production registration creates `FrozenConnectorHandler` instances;
- context-free execution fails closed;
- all authority and lifecycle decisions use trusted execution context;
- failures preserve retry and uncertain-outcome semantics;
- secret bytes cannot escape the connector-to-provider request boundary;
- applicable conformance scenarios pass for the exact migrated artifact;
- the old production handler path receives no new traffic;

- a rollback artifact and state-preservation procedure are available.

Source presence establishes implementation only. Report conformance, qualification, live verification, and production readiness as separate evidence classes.

Decide migration failures

Failure	Decision
Manifest digest changes unexpectedly	Stop and identify the changed field before routing
A contract claim lacks support	Implement it or narrow the contract in a separate reviewed change
Frozen handler returns authorization for normal traffic	Fix gateway registration or trusted-context construction; do not bypass the adapter
Legacy and frozen outputs differ	Compare provider request, context, and error mapping before expanding
Cancellation creates an uncertain mutation	Preserve state and reconcile instead of retrying
A compatibility caller still needs the old trait	Keep the trait behind an explicit boundary while production stays frozen
Conformance scenario is absent	Add the scenario or mark a source-backed non-applicable reason
Live-provider behavior changed	Stop rollout and re-baseline the provider revision

Roll back without restoring a bypass

Drain the migrated route and stop new assignments before restoring the previously accepted artifact or local registration. Preserve runtime state, idempotency records, provider operation references, callbacks, evidence, and logs needed to finish or reconcile in-flight work.

Restore the old path only in the environment where its documented trust limitations were already accepted. Do not route high-risk traffic through a context-free handler as an emergency shortcut. Correct the migration under a new immutable artifact identity, repeat the regression and conformance work, and cut over through the same bounded route.

Related documentation

- Capabilities
- Actions and sessions
- Identity and trust
- Profiles and connectors