SAMPLE ARCHITECTURE REVIEW

# Northstar Desk

Active portfolio workflow

EXECUTIVE VERDICT

Keep automated execution disabled

| REFERENCE | SAMPLE STATUS | ISSUE DATE | CLIENT |
|---|---|---|---|
| HA-AR-SAMPLE-002 | Fictional sample | 4 August 2026 | Northstar Desk |
| **SYSTEM STAGE** | **REVIEW BASIS AGREED** | **REVIEW TYPE** | **SCOPE** |
| Active; automated execution paused | 3 August 2026 | Architecture Review | Architecture and control design only |

PREPARED BY HODL ADVISORY

Architecture Reviews and Blueprints for consequential agent systems. **hodl.org**

This sample is based on a simulated engagement and contains no real-client material.

DECISION FIRST

# The verdict and the Review boundary.

EXECUTIVE VERDICT

## Keep automated execution disabled

Keep automated execution disabled. In the client-confirmed workflow, a timeout retry creates a new send reference, while no stable operation identity connects approval, every send attempt, provider evidence, and final reconciliation. The current status and stop controls also cannot prove what happened at the provider or that all execution work has stopped.

The recommendation model is not the first repair target. Start with one accountable lifecycle owner, a clear operating mandate and limits, one action identity across retries, provider-grounded outcome confirmation, and stop-and-drain behavior that can be demonstrated. The manual path remains consequential because an operator may act on the recommendation.

**REVIEW BOUNDARY**

Conclusions are based on the agreed Review basis and material listed in this report. No source code, production access, provider verification, investment-strategy assessment, or compliance review was performed.

### At a glance

| ITEM | REVIEW CONCLUSION |
|---|---|
| Client | Northstar Desk |
| Reviewed system | One recommendation → approval → provider action → state and reconciliation workflow |
| Current operating state | Recommendations active; automated execution disabled; operator executes manually |
| Immediate decision | Which layer to repair first before restoring automation |
| First control repair | Name the lifecycle owner and restoration authority; define current authority and limits |
| First technical repair | Durable operation identity and enforced action lifecycle across retries |
| Confidence | Strong for the supplied architecture gaps; no independent code or provider verification |

THE CLIENT’S QUESTIONS

# Direct answers before technical detail.

**1. Which layer must be repaired first?**

Repair the external-action lifecycle first. Start by assigning its owner and defining the authority and limits for restoration. The first technical control is one durable operation identity created before the provider boundary and reused across retries. This dependency comes before UI cleanup, new recommendations, or added automation.

**2. Can automatic retries be kept safely?**

Potentially, but not under the current contract. Stable identity reused across retries is necessary, not sufficient. Provider duplicate-request behavior must be verified; attempt and operation states must be durable; repeated and out-of-order responses must be safe; expiry must be rechecked; reconciliation must establish closure; and stopping must prevent queued or running work from crossing the boundary.

**3. What minimum evidence is required before automated execution can be enabled again?**

At minimum: durable operation and approval records; evidence that timeout retry cannot duplicate the operation; tested valid state transitions including partial and unknown outcomes; provider-grounded reconciliation; and a stop or drain record showing that no queued or not-yet-sent work can cross after the stop barrier. These tests must cover interruption, duplicate callback, out-of-order callback, late response, expired approval, partial completion, and restart.

AGREED REVIEW BASIS

# The complete client-confirmed description used for this Review.

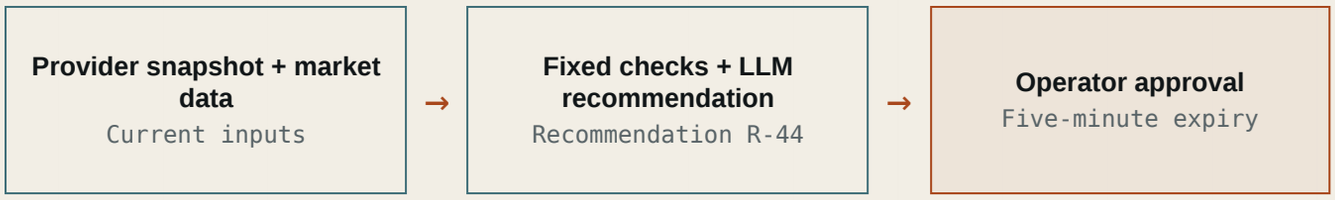
- **Purpose:** decide what must be repaired first before the team adds capabilities or considers restoring automated execution.
- **Reviewed workflow:** provider snapshot and market data feed fixed checks and an LLM recommendation; an operator approves; Celery submits to the provider; callback, polling, position refresh, Postgres, and UI can each influence recorded outcome.
- **Operating limits:** no value, scope, frequency, aggregate, concurrency, or reversibility limits were supplied, and the effect of expiry, revocation, policy change, or a stop on pending work is not defined.
- **Runtime as described:** production uses custom Python services, Celery/Redis, a scheduler, and Postgres. Claude Code helped write code; it is not the production orchestrator. The LLM supplies recommendation text after fixed prechecks.
- **Inputs, approval, and manual action:** proposals do not carry immutable references to the provider snapshot, market observation, fixed checks, policy, model result, current authority, or operating limits. Source authority, effective time, freshness, version, validity, and conflict handling remain incomplete. A recommendation may influence manual provider entry, but the identity, credential authority, and separation of duties for that path are not defined.
- **Events as described:** callbacks may repeat or arrive late. The handler uses provider time when present, but no documented transition rule or test evidence for duplicate and out-of-order events was supplied. Callback, polling, and refresh can each influence closure.
- **Incident description:** the client reported three state or retry incidents in six weeks and believes changed send references contributed to the latest one.
- **Acknowledgement:** Mira Vale confirmed this summary as the basis for the Review on 3 August 2026.
- **Current state:** recommendations remain active; automated execution is disabled; an operator checks the provider portal and performs actions manually.
- **Roles and ownership:** operators approve actions; the operations lead controls the execution flag; engineering owns code and workers; no one currently owns the complete action lifecycle and its closure meaning. No authorizing principal or versioned operating mandate was supplied.
- **Client questions:** which layer to repair first; whether automatic retries can be retained; and what minimum evidence is required before automated execution can return.
- **Identity and retry:** a recommendation ID persists through approval, but each execution attempt creates a new send reference, including a retry after network timeout. A provider order ID is stored only after a response. No single durable operation identity joins approval, attempts, and provider evidence before the first send. The provider reportedly deduplicates the same reference, but same-reference duplicate delivery has not been tested.
- **Status and closure as described:** the UI reads Postgres. “Submitted” is recorded when the API call returns; “complete” may be recorded from a callback or expected position later. Partial completion is not represented consistently. Operators treat the provider portal as final during a dispute, but no formal conflict rule exists and position refresh does not reconstruct the full action lifecycle.
- **Stopping and unresolved work:** disabling execution prevents new queueing, but the team cannot demonstrate that already queued or running tasks cannot still call the provider. There is no recorded drain barrier or stop receipt. Unresolved outcomes have no defined owner, age, escalation point, remaining exposure, or closure rule.
- **Material reviewed:** the client's description of the component workflow, representative incident, state and ownership responsibilities, and the lifecycle clarifications summarized above.

> This is a client-confirmed description of the system and representative incident. It was not independently verified against source code, raw logs, deployment, provider documentation or records, credentials, production access, account data, or portfolio data.

SYSTEM MAP

# The reviewed route and the points where identity, evidence, or authority changes hands.

CLIENT-CONFIRMED ROUTE · HUMAN AUTHORITY · PROVIDER EFFECT · REPORTED INCIDENT PATH




**Celery execution task** — IDENTITY SPLITS AFTER APPROVAL


**Architecture reading:** one approval can fan out into separately identified sends, while returned provider evidence cannot be joined to one durable operation. The incident path is client-reported; this Review did not independently verify its cause.

PRIORITY

01

ARCHITECTURE FINDING

CRITICAL

# The described retry path changes identity at the provider boundary

**What is happening.** A recommendation ID survives approval, but a timeout retry sends the approved action under a new client reference. No stable operation ID is created before the first provider call. The client believes this identity break contributed to the representative incident; that cause was not independently verified.

**Why it matters.** A timeout does not prove the provider rejected the first request. Changing identity removes the provider's reported same-reference deduplication opportunity and can turn one approval into multiple accepted actions.

**Recommended change.** Persist one durable operation identity before the first provider send and ensure it survives interruption or restart. Reuse the provider idempotency reference across retries where the provider contract supports it. Keep every attempt under that operation, and treat timeout as unknown until provider evidence resolves it.

**What to validate.** The engineering lead should run repeated timeout and restart tests. One approval must produce at most one provider operation, and every attempt and returned event must remain linked to the same operation.

PRIORITY

02

ARCHITECTURE FINDING

CRITICAL

# The described status model cannot establish provider-confirmed closure

**What is happening.** “Submitted” is recorded when the API call returns. “Complete” may come from a callback or from seeing the expected position later. Partial completion is inconsistent, and the provider portal wins only informally during disputes.

**Why it matters.** The UI may display closure while external state is partial, duplicated, delayed, contradictory, or unknown. Operators cannot distinguish a local projection from a verified provider outcome.

**Recommended change.** Define one operation lifecycle that can represent rejected, failed, delayed, conflicting, unknown, provisional, reversed, and partial outcomes. Tie every transition to dated provider evidence. Treat Postgres and the UI as local views with visible freshness and reconciliation state, and keep disagreements visible until resolved.

**What to validate.** A named lifecycle owner must define what evidence is authoritative and own every unresolved outcome, including its age, next review, remaining exposure, and closure condition. Duplicate, delayed, out-of-order, partial, contradictory, and reversed events must never create false completion.

PRIORITY

03

ARCHITECTURE FINDING

HIGH

# No enforced callback transition contract was demonstrated

**What is happening.** Callbacks may repeat or arrive late, while callback, polling, and position refresh can each influence closure. No tested transition rule was supplied to show that an older event cannot overwrite newer state.

**Why it matters.** Even after stable send identity is introduced, event reordering can corrupt the internal projection or reopen or close an operation incorrectly.

**Recommended change.** Define allowed transitions, terminal-state behavior, event identity and ordering, duplicate handling, and rules that prevent weaker evidence from replacing stronger evidence. Record rejected transitions instead of silently applying them.

**What to validate.** The engineering lead should run a fixed event-sequence suite covering duplicate, late, and out-of-order delivery. Every sequence must preserve the correct final state and retain rejected events for review.

PRIORITY

04

ARCHITECTURE FINDING

HIGH

# The execution switch does not prove the workflow has stopped

**What is happening.** Turning off execution prevents new queueing, but already queued or running tasks may still reach the provider. The team checks the queue dashboard, waits, and may stop workers; there is no drain barrier or durable stop record.

**Why it matters.** Operators may believe the system is stopped while an in-flight worker can still cross the action boundary.

**Recommended change.** Define pause, drain, stop, and emergency stop separately for recommendations, approvals, manual entry, queued work, running work, provider calls, and remaining external effects. Every provider send must verify that execution remains currently authorized after the relevant stop boundary. Add a drain barrier, record every outstanding operation, and test failures that could disable both execution and the mechanisms intended to stop it.

**What to validate.** Operations should own stop intent and engineering should own enforcement. After the stop barrier, no queued or not-yet-sent work may cross the provider boundary. Any request already sent, or whose provider outcome is uncertain, must remain explicitly unresolved until reconciled. The stop record must distinguish drained, cancelled, in-flight, and unresolved work, including failures that could disable both execution and stopping.

PRIORITY

05

ARCHITECTURE FINDING

HIGH

# No end-to-end lifecycle owner was identified

**What is happening.** Operations owns the execution flag, engineering owns code and workers, and operators approve actions. No one owns the meaning and closure of the complete action lifecycle.

**Why it matters.** Local component fixes can continue to move symptoms because no one owns the cross-component invariants: one approval, one operation, valid transitions, provider-grounded closure, and safe stopping.

**Recommended change.** Assign one accountable lifecycle owner. That owner should define the operating mandate and limits, the one-approval/one-operation rule, the state model, authoritative evidence, unresolved-work escalation, stopping rules, and restoration gates. Component owners still implement their parts, but cannot redefine closure locally.

**What to validate.** The owner is not yet assigned. A named owner and the person authorized to restore execution must approve the lifecycle contract, test evidence, and staged limits before automation returns.

ROLES AND OPERATING BOUNDARY

# Keep decision authority explicit while the system is repaired or prototyped.

## AI and platform boundary

Nothing in the reviewed workflow makes the LLM the first repair target. The reported duplicate-action path occurs after human approval, inside the custom execution and retry lifecycle.

The Review also does not conclude that Celery is inherently wrong or that an agent framework is required. The requirement is a durable, testable lifecycle and authority contract. The team may implement that contract with its current substrate or choose another one through a separate design decision.

The recommendation can still influence manual provider entry, so its source inputs, policy, model result, configuration, and tool state should be identifiable at the time of the decision. Untrusted content must not be able to change authority or permissions, execution must not reconstruct an approved action from conversation memory, and any new tool or permission must pass the same restoration gates. No implementation or test evidence for these controls was reviewed.

### Do not automate yet

- provider action submission;
- network-timeout retry across the provider boundary;
- callback-driven completion without enforced transition rules;
- position-based inference of completion where partial or duplicate outcomes are possible;
- re-enabling execution from a dashboard check without a drain and stop record;
- adding further autonomous capabilities while lifecycle invariants remain unowned;
- treating a recommendation as a harmless read-only path when it can influence a manual provider entry.

The recommendation path may continue only within the team's own operating-risk judgment and with manual provider verification. It remains a consequential human-executed path when an operator may act from it, and this Review does not validate its recommendation quality, authority chain, provider-entry control, or portfolio strategy.

DEPENDENCY-ORDERED ROADMAP

# Sequence the work by evidence, not by feature appetite.

STAGES 0–1

### Ownership and authority

**0** **Keep execution disabled**

Lifecycle owner and restoration authority accept the repair sequence.

**1** **Define authority and limits**

One testable contract binds exact approval, current inputs, credentials, limits, and blockers.

STAGES 2–4

### One action lifecycle

**2** **Create one operation identity**

Timeout or restart produces one provider operation per approval.

**3** **Enforce lifecycle transitions**

Duplicate, late, and out-of-order events cannot corrupt state.

**4** **Reconcile against provider evidence**

Disagreement stays visible and correction preserves history.

STAGES 5–6

### Stopping and runtime proof

**5** **Prove stop and drain**

Not-yet-sent work cannot cross after stop; uncertain work stays visible.

**6** **Run replay and failure injection**

Retry, callback, restart, permission, and infrastructure-failure tests pass.

BLOCKED RESTORATION GATE

**7** **Consider narrowly staged automated execution**

Proceed only after all prior gates pass with implemented evidence and the named lifecycle owner and restoration authority approve the limits. Otherwise automated execution stays disabled.

ASSUMPTIONS AND DECISION GATE

# Unknowns stay visible; the next decision stays bounded.

## Assumptions and unknowns

This Review assumes the client-confirmed Review basis and sanitized incident timeline accurately describe the system. It does not establish the provider's actual duplicate-request contract, code transaction boundaries, queue configuration, callback implementation, database constraints, security posture, or financial consequence.

Unknowns include the authorizing principal, operating mandate, value and aggregate limits, operational volume, latency and availability targets, other incident classes, provider event guarantees, jurisdiction and compliance obligations, credential design, separation of duties, material model, prompt, context, and tool states, and whether other action paths bypass the reviewed workflow.

## RECOMMENDED NEXT DECISION

Do not choose a new framework yet. Keep automated execution disabled, name the lifecycle owner and restoration authority, and write the operating mandate, limits, exact approval record, evidence hierarchy, and separation of duties. Then use the lifecycle contract and test suite to decide whether the current Python and Celery system is adequate.

## Restoration evidence checklist

- **Identity:** one approval maps to one durable operation across every attempt.
- **Outcome:** partial, unknown, duplicate, and contradictory outcomes remain representable and reconcile against provider evidence.
- **Stopping:** stop and drain prevent new boundary crossings while already-sent or uncertain work remains visible until reconciled.
- **Authority:** current approval, limits, credentials, and expiry are enforced at the provider boundary.
- **Transitions:** duplicate, delayed, and out-of-order events cannot create false closure.
- **Failure evidence:** restart, retry, concurrent-change, permission, and infrastructure-failure tests pass before restoration.

TERMS USED IN THIS REVIEW

# Plain-language definitions for the decisions and controls discussed in the report.

**Operation identity** — one stable identifier for the approved real-world action across every send attempt and returned event.

**Idempotency reference** — a reference intended to make repeated delivery of the same operation produce at most one external effect, subject to the provider's verified contract.

**Projection** — the application's local view of provider state; useful for UI, but not automatically authoritative.

**Reconciliation** — comparison of the local projection with provider evidence, preserving and resolving differences explicitly.

**Durable stop record** — evidence that queued and not-yet-sent work cannot cross the external action boundary after stopping, while already-sent or uncertain work remains visible until reconciled.

METHOD NOTE

Hodl Advisory uses the open PROOF Standard as a control lens when reviewing consequential agent systems. The Review remains an architecture assessment of the client system, not a separate PROOF conformity assessment.