1I AUTOMATION REFERENCE ARCHITECTURE

Plan → Approve → Execute → Verify

AI-Assisted Infrastructure Change Automation with Human Approval Gates

Pattern	**Reference Scenario**
Plan → Approve → Execute → Verify	AI-Assisted Infrastructure Change Automation
Core Stack	**Execution Model**
LangGraph + Policy Guardrail Engine + Change Record Store	Plan validation → Human approval → Simulator execution → Production-ready package

Robert Myhre, Independent Researcher

Version 1.1 · June 7, 2026

Reader's Guide

This is a reference architecture, not a turnkey implementation. This document describes an established pattern for AI-assisted infrastructure change automation with non-bypassable human approval. It defines the logical structure, component responsibilities, technical implementation options, the plan validation model, the policy guardrail engine, failure modes, and governance requirements. It is not a deployable product. Production use requires adaptation to the target organization's infrastructure environment, data classification policy, operational workflow, security model, SLA requirements, and staffing model. That adaptation work is expected and necessary. The Production Adaptation Checklist in Section 11 identifies the decisions that must be made before implementation design is finalized.

This document is written for three audiences. Different sections will be most relevant depending on how you are engaging with this architecture.

Audience	Most Relevant Sections
Business and operations reviewers evaluating whether the pattern fits a business problem	Section 1 (Pattern Definition and Fit Criteria), Section 2.5 (Human Touchpoints), Section 10 (Differentiation and Limitations), Section 11 (Production Adaptation Checklist)
Architecture and engineering teams assessing implementation feasibility	Section 2 (Logical Architecture), Section 3 (Technical Architecture), Section 4 (Policy Guardrail Engine), Section 5 (Rollback Architecture), Section 6 (Failure Modes), Section 8 (Implementation Options)
Security, compliance, and governance reviewers	Section 7 (Security and Governance), Section 9 (Auditability and Explainability), Section 11 (Production Adaptation Checklist)
Technical evaluators assessing the architecture against specific use cases	Full document. Start with Section 1 to confirm pattern fit, then Section 6 for failure modes and Section 10 for limitations before proceeding.

1. Pattern Definition

1.1 What This Pattern Solves

The Plan → Approve → Execute → Verify pattern addresses IT and infrastructure operations workflows where configuration changes must be planned, reviewed, executed, and verified with full auditability and a defined rollback path. The automation value is in three specific areas: interpreting natural language change requests into precise executable plans, enforcing policy constraints consistently before human review, and providing symmetric pre/post verification that confirms the change achieved its intended effect.

Without AI assistance, these workflows depend on human engineers to translate requirements into commands, manually check for conflicts, write rollback procedures separately from the change plan, and run pre/post checks inconsistently. The AI layer makes the translation, conflict checking, rollback generation, and verification symmetric and auditable by design.

What is architecturally new in this pattern: Five concepts appear here that read-only classification or routing patterns do not require. First, intent interpretation producing a machine-executable change specification — not a routing decision, but a structured artifact that becomes the execution payload. Second, a policy guardrail engine, including authorization scope, that evaluates the proposed change before human review. Third, a non-bypassable human approval gate where the approver sees the exact commands that will execute before authorizing. Fourth, a Change Package as a first-class architectural artifact — versioned and immutable, traveling through every phase and persisting in a Change Record Store. Fifth, a rollback plan generated at planning time, not at failure time.

1.2 Fit Criteria

Criterion	Indicator	Poor Fit Signal
Repeatable change type	The change follows a pattern that can be specified as a template: VLAN additions, interface configuration, routing protocol changes, ACL updates, security group changes, patch deployment sequences.	One-off novel changes requiring open-ended engineering judgment at execution time.
Policy constraints are codifiable	Organizational rules constraining what changes are permitted can be expressed as deterministic checks: ID ranges, maintenance windows, blast radius thresholds, authorization scope.	Rules that require human judgment to interpret on a case-by-case basis. These belong in the human approval step, not the policy engine.
Verification is measurable	Success can be defined as a set of observable post-change states.	Changes whose success is subjective or unmeasurable at the time of execution.

	Changes whose success criteria are objective and observable.	
Human approval is required	The organization requires human sign-off before production changes. This pattern is designed for that workflow.	Fully autonomous execution without human review. This pattern is not an autonomous execution system.
Rollback is defined	A reversible set of commands exists that can undo the change. The rollback path is known before execution begins.	Irreversible changes or changes where the rollback procedure is not determinable at planning time.
Simulator fidelity is adequate	A simulation or staging environment is available or buildable that is close enough to production that simulator verification results are meaningful predictors of production behavior.	No simulation environment available and direct production testing is the only option.

1.3 Vertical Applicability

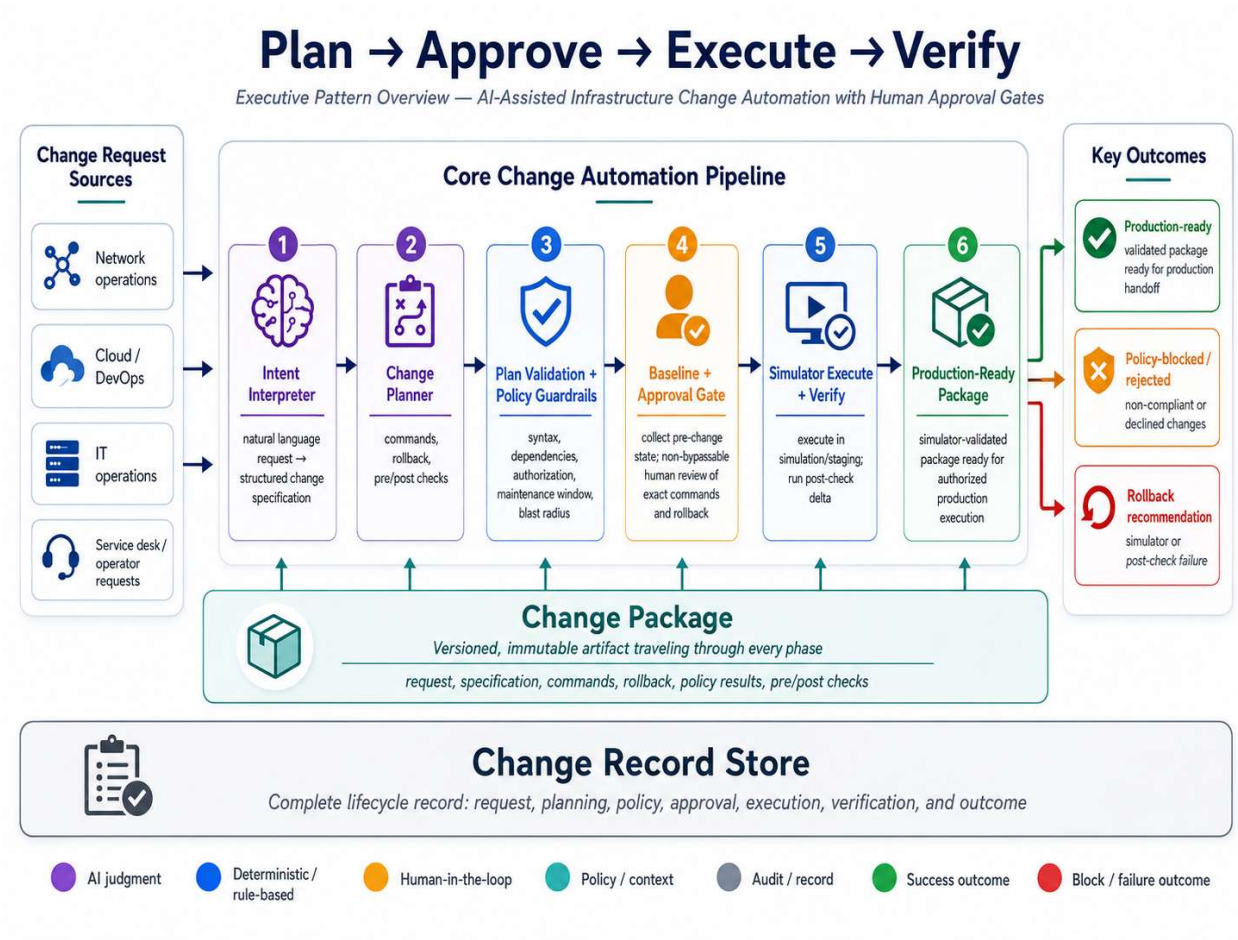
Vertical	Example Process	Adaptation Notes
Network Operations	VLAN provisioning, routing changes, ACL updates, interface configuration	The anchor scenario. Policy engine enforces VLAN ranges, maintenance windows, blast radius, and authorization scope.
Cloud / DevOps	Infrastructure-as-code change validation, Terraform plan approval, security group updates	Swap 'device commands' for 'Terraform plan'; pre/post checks become state drift checks against cloud provider APIs.
IT Operations	Server configuration changes, patch deployment with pre/post health checks, firewall rule updates	Swap 'network device' for 'server' or 'firewall'; execution targets become Ansible playbooks or configuration management tooling.
Healthcare IT	Network segmentation changes for compliance, medical device VLAN isolation, clinical system configuration	Compliance audit trail becomes primary value proposition. Policy engine enforces segmentation policy. Data classification review required.
Financial Services	Trading network changes, firewall policy updates, disaster recovery failover validation	Change freeze windows and CAB integration become prominent policy engine features. Stronger audit retention requirements.
Telecommunications	Core network configuration changes, customer circuit provisioning, VPN tunnel updates	Direct fit. Telco network operations workflows map closely to the infrastructure change automation scenario.

2. Logical Architecture

2.1 The Nine Phases

The architecture is organized as nine sequential phases with two exit paths that branch before production execution. Phases are sequential — no phase begins until its predecessor completes and passes its gate condition. Both exit paths (policy block and post-check failure) terminate at the audit logger with complete records.

Figure 1: Logical Architecture — Plan → Approve → Execute → Verify



Phase	Role
1. Intent Interpreter	Receives the natural language change request. Produces a structured change specification: change type, affected devices, parameters, affected services, and constraints stated in the request. Output is schema-constrained. LLM phase.

2. Change Planner	Expands the change specification into a full change plan and creates the Change Package: device-by-device commands in dependency order, rollback commands for each device, pre-check and post-check specifications, blast radius assessment, and metadata. The Change Package is the versioned immutable artifact that travels through every subsequent phase. LLM phase for command generation; deterministic for dependency and rollback validation.
3. Policy Guardrail Engine	Evaluates the Change Package against four organizational policies: authorization scope, change-type conflict, maintenance window, and blast radius. Returns go/no-go with specific violation details per policy. Deterministic phase. Policy block records the violation in the Change Record Store and terminates.
4. Pre-Check Collector	Executes the pre-check specifications against the simulation or staging environment. Collects baseline state. Stores as the before_snapshot in the Change Package. Deterministic phase.
5. Human Approval Gate	Presents the complete Change Package to a human approver. The approver sees: exact commands per device, all policy check results, pre-check baseline, rollback plan, authorization confirmation, and change identifier. Approval is non-bypassable. The approved Change Package is versioned and locked in the Change Record Store before execution begins.
6. Simulator Execution	Executes the approved Change Package against the simulation environment. Commands issued in dependency order. Each command and response logged to the Change Record Store. Deterministic execution phase.
7. Simulator Post-Check	Runs post-check specifications against the simulation environment. Computes delta against before_snapshot. Evaluates verification criteria. Simulator failure stops execution, updates Change Record Store with failure details, and surfaces rollback recommendation.
8. Production-Ready Package	Simulator post-check passed. The Change Package is marked simulator-validated and production-ready. A production-ready summary is generated showing the commands verified against the simulation environment and ready for authorized execution against production infrastructure.
9. Outcome and Audit	Change Package outcome is written as simulator-validated / production-ready. Complete Change Package is written to the Change Record Store. All phase-by-phase state, execution log, post-check results, and approver record are retained.

2.2 The Plan Validation Model

This pattern does not use a confidence scoring model in the same sense as classification patterns. Classification confidence is not the relevant concern — plan correctness and safety are. The plan validation model assesses the change plan on five dimensions before it reaches the human approval gate.

The validation boundary — what plan validation does and does not prove: Plan validation proves four things: the plan is complete (all required fields populated), syntactically valid (command structure is correct for the target system), policy-compliant (passed all policy

checks), and simulator-verifiable (the simulation environment can execute it and post-checks can run against the result). Plan validation does not prove the change is production-safe. It does not verify that proposed parameters avoid conflicts with production-specific configurations or environmental details not present in the simulation. These are operational correctness concerns that require domain expertise and human review. This boundary must be visible to the human approver. The approval interface states explicitly: 'This plan has passed syntactic validation and policy checks. It has not been validated for operational correctness beyond the defined post-check criteria. The approver is responsible for operational review.'

Validation Level	What It Covers	What It Does Not Cover
Syntactic validation (automated)	Command syntax for the target system, parameter value ranges, required keyword presence, and structural validity.	Whether the parameter values are appropriate for the specific production environment.
Structural validation (automated)	Dependency ordering (correct execution sequence), rollback completeness (one rollback command per forward command), specification completeness (all required schema fields populated).	Whether alternative valid orderings exist for specific change types.
Operational validation (human + simulator)	Simulator post-checks verify the intended post-change states are achieved. These are the defined verification criteria, verified against the simulation environment.	Production-specific operational correctness beyond the defined post-check criteria.

Why plan validation replaces confidence scoring: In classification patterns, confidence scoring answers: how certain is the model about its interpretation? In this pattern, the output is not an interpretation — it is a change plan. The relevant questions are not about certainty; they are about correctness, completeness, and safety. A change plan is either syntactically valid or it is not. The rollback plan either covers the forward plan or it does not. These are deterministic checks, not probabilistic assessments. The plan validation model is a structured correctness gate, not a confidence scorer.

2.3 The LLM / Deterministic Boundary

A sophisticated evaluator will ask which parts of the pipeline use AI judgment and which are deterministic. This boundary is architecturally significant: it defines where interpretive flexibility lives and where rule-based certainty is required. The answer must be explicit in the architecture.

Phase	Classification	What This Means
Intent Interpreter	LLM	Interprets natural language change requests into structured specifications. The only place where free-text interpretation occurs.
Change Planner	LLM (command generation) / Deterministic (validation)	Generates device-by-device commands from the specification. Dependency ordering and rollback coverage checks are deterministic.
Plan Validation	Deterministic	Five structured checks: completeness, syntactic correctness, dependency ordering, rollback completeness, rollback ordering.
Policy Guardrail Engine	Deterministic	Four policy checks in defined sequence. Database lookups and arithmetic. No model involvement.
Pre-Check Collector	Deterministic	CLI command execution against simulation environment. Structured output parsing.
Human Approval Gate	Human	Non-bypassable structural INTERRUPT. The approver authorizes specific infrastructure writes, not a routing recommendation.
Simulator Execution	Deterministic	Executes commands as-is from the approved Change Package. No model involvement, no interpretation.
Post-Check and Verification	Deterministic	Applies defined verification criteria. Computes delta against before_snapshot.
Rollback Plan	Deterministic (generated by LLM at plan time)	Generated at plan time by the change planner. Deterministic at execution time — no model involvement in rollback recommendation.

The boundary is the answer to the most important evaluator question: When a stakeholder asks 'what is the AI actually doing in this system,' the answer is: two things. First, it interprets natural language change requests into structured specifications. Second, it generates executable commands from those specifications. Everything else — policy checking, pre/post verification, execution, rollback — is deterministic and auditable without model involvement. That boundary is what makes the architecture trustworthy enough to put in front of a change advisory board.

2.4 The Change Package and Change Record Store

The Change Package is the central architectural artifact of this pattern. It is created at the end of the change planner phase, versioned and immutable from that point forward, and travels with the change through every subsequent phase. Every phase reads from or writes to the Change Package. The Change Record Store is the persistence layer that retains all Change Packages and their phase-by-phase state history.

Change Package Field	Contents
change_id	UUID generated at creation time. Immutable. Primary key for Change Record Store queries.
schema_version	Change Package schema version. Enables forward compatibility as the schema evolves.
request_text	Original natural language change request. Verbatim. Immutable.
change_specification	Structured specification from intent interpreter: change type, affected devices, parameters, requestor identity.
forward_plan	Device-by-device command list in dependency order. Per-device: device name, commands, estimated duration.
rollback_plan	Device-by-device rollback commands in reverse dependency order. Generated at plan time. Immutable after approval.
precheck_spec	Pre-check command specifications: commands to run, parsing rules, baseline fields to capture.
postcheck_spec	Post-check command specifications: commands to run, parsing rules, success criteria, delta comparison rules.
policy_results	Results of all four policy checks with timestamps. Immutable after policy phase.
before_snapshot	Pre-check baseline collected in Phase 4. Immutable after collection.
approval_record	Approver identity, decision, reason if rejected, timestamp. Immutable after decision.
execution_log	Per-command execution records appended during execution phases: device, command, response, timestamp, success flag.
postcheck_results	Post-check outputs, delta against before_snapshot, verification criterion results, overall pass/fail.
outcome	Final change outcome: simulator-validated/production-ready, failed, or rejected. Timestamp.

The Change Record Store retains every Change Package indefinitely. Change Packages are queryable by change_id, requestor identity, affected device, date range, and outcome. This is a primary compliance artifact that a change advisory board or auditor reviews. It is not a log — it is a structured record of every decision made during the change lifecycle.

2.5 Human Touchpoints

Touchpoint	Description
Human Approval Gate	The central human touchpoint and the non-bypassable gate between planning and execution. The approver reviews the complete Change Package: original change request, interpreted specification, full device-by-device command plan, all four policy check results, pre-check baseline snapshot, rollback plan, estimated blast radius, and change_id. The approver

	approves (Change Package locked and proceeds to simulator execution) or rejects (rejection reason recorded, workflow terminates). No architectural path bypasses this gate.
Rollback Recommendation Review	If simulator or production post-check verification fails, the system presents the rollback plan to a human reviewer — it does not execute rollback autonomously in v1. The reviewer sees what failed in the post-check delta, rollback commands per device in copy-ready format, estimated rollback blast radius, and the change_id.
Policy Rule and Authorization Maintenance	The policy engine rules — assignment databases, maintenance window schedules, blast radius thresholds, and authorization scope tables — are human-maintained configuration. Change management and security teams own these rules. Changes do not require model retraining or pipeline modification.
Change Record Store Audit Review	Post-change audit review of any Change Package by change_id. Accessible to operations management, change management, security, and compliance stakeholders. Shows the complete change lifecycle: request, specification, plan, all policy results, pre-check, approval decision with approver identity, execution log, post-check results, and outcome.

3. Technical Architecture

The technical architecture maps each logical component to a specific technology choice. Technology selections reflect the reference implementation stack. Organizations adopting this pattern should evaluate each choice against their existing infrastructure, licensing constraints, and operational expertise. See Section 8 (Implementation Options) for alternatives.

Figure 2: Technical Architecture — Plan → Approve → Execute → Verify

3.1 Component Technology Mapping

Logical Component	Technology	Rationale
Orchestration	LangGraph (Python)	State management across nine phases, conditional routing at the policy gate and post-check gate, and human approval INTERRUPT support.

LLM (Intent + Planning)	Local model serving (Ollama) or cloud API	Structured output enforcement. Intent interpretation and change planning are the only two LLM phases. All other phases are deterministic. See Section 8.2 for hosting trade-offs.
Change Specification Schema	Pydantic v2 (Python)	Schema validation of LLM output at the interpret and plan node boundaries. Ensures structured output compliance before downstream processing.
Change Package	Python dataclass + JSON serialization	Versioned immutable artifact created at plan time. Travels through every phase. Stored in Change Record Store.
Change Record Store	SQLite (reference) / PostgreSQL (production)	Persistent store for all Change Packages. Queryable by change_id, requestor, device, date, and outcome. A primary compliance artifact. Distinct from the operational audit log.
Policy Engine	Python rule functions + database	Four checkers: authorization scope, change-type conflict, maintenance window, blast radius. Deterministic, auditable, human-configurable without pipeline modification.
Pre/Post Check Execution	CLI execution library (e.g., Netmiko, NAPALM) or infrastructure API	Command execution against simulation environment. Same library used for pre-check, simulator execution, and post-check.
Simulator Execution	Simulation or staging environment API	Change Package execution against virtual or staged topology. Simulator fidelity must be sufficient for post-check results to be meaningful production predictors.
Production Execution	CLI execution library / RESTCONF / infrastructure API	Real device or infrastructure execution with per-command logging. Follows the same pipeline with production credentials substituted for simulation credentials with organization-specific controls, credential handling, change windows, and ITSM integration.
Human Approval UI	Web interface (framework-independent)	Six required elements: (1) commands per device in monospace blocks, (2) configuration delta view, (3) all policy results, (4) pre-check baseline table, (5) rollback plan per device, (6) execution target label and change_id.
Rollback Recommendation UI	Web interface (framework-independent)	Post-check failure details, rollback commands per device in copy-ready monospace blocks, change_id reference.
Audit Logger	LangGraph checkpointer + persistent store	Phase-by-phase execution state. Separate from Change Record Store: the audit log is operational telemetry; the Change Record Store is the compliance record.

API Surface	FastAPI (Python)	Change request submission, status query, approval interface, Change Record Store query endpoint.
Container Management	Docker Compose or equivalent	Full stack: LLM serving, API, database, and observability in coordinated deployment.

3.2 Processing Node Map

Node	Function
interpret_node	Receives change request text. Invokes LLM with structured output schema. Produces change specification including requestor identity. Schema validation at boundary. LLM phase.
plan_node	Expands change specification into full Change Package: device-by-device commands, rollback commands, pre/post check specs, blast radius estimate. Validates dependency ordering and rollback completeness. LLM phase for command generation; deterministic for validation.
validate_node	Runs plan validation: specification completeness, syntactic correctness, dependency ordering, rollback completeness, rollback ordering. Deterministic. Failed validation returns to plan_node (max one retry).
policy_node	Executes four policy checkers in order: (1) authorization scope, (2) change-type conflict, (3) maintenance window, (4) blast radius. Returns go/no-go per check. Any block writes violation to Change Package and routes to audit_node.
precheck_node	Executes pre-check commands against the simulation environment. Collects before_snapshot. Deterministic. Pipeline does not proceed to approval with an incomplete baseline.
approval_node	INTERRUPT. Presents full Change Package to approver via the approval interface. Requires all six elements. Awaits approve/reject. Rejection recorded in Change Package and routes to audit_node.
sim_execute_node	Executes forward plan from Change Package against the simulation environment. Per-command response logged to Change Package execution_log. Deterministic.
sim_postcheck_node	Executes post-check commands against simulation environment. Computes delta against before_snapshot. Evaluates verification criteria. Failure routes to rollback_recommend_node.
production_ready_node	Simulator post-check passed. Marks the Change Package as production-ready. Generates production-ready summary. In the reference scenario, this is the terminal simulator success state — production execution is deferred, not simulated.
rollback_recommend_node	Writes failure state to Change Package. Presents rollback recommendation: failure delta, rollback commands per device, change_id.
outcome_node	Writes success record to Change Package: outcome, post-check delta, completion timestamp, approver identity, total duration.

audit_node	Terminal for all paths. Writes final Change Package state to Change Record Store. All paths terminate here: policy block, rejection, sim failure, and success.

4. Policy Guardrail Engine

The policy guardrail engine is the deterministic gate between the plan validator and the human approval gate. It evaluates the proposed change plan against organizational policy and returns a go/no-go decision with specific violation details. Policy check is not human judgment — it is rule application. A change that violates policy does not reach a human approver; it is blocked with a specific violation record and terminated.

4.1 Policy Set

Policy	What It Checks	Failure Behavior
Authorization Scope (runs first)	Checks whether the requestor identity is authorized to request the specified change type on the specified target group. Authorization scope table maps roles to permitted change types and target groups.	Block. Surface: requestor identity, requested scope, authorized scope, the specific boundary exceeded. No other policy check runs if authorization fails.
Change-Type Conflict	Checks the proposed change parameters against the assignment database. Flags if the target ID is already assigned to a different purpose, is in a reserved range, or conflicts on any affected device or system.	Block. Surface: conflicting assignment, range policy, suggested available alternatives.
Maintenance Window	Checks whether any affected device or system is in a change freeze window (no changes permitted) or an active maintenance window (change expected but noted).	Block on change freeze. Advisory flag during active maintenance window: approver sees explicit warning in Change Package.
Blast Radius	Counts affected devices, systems, estimated end-user impact, and Tier-1 service exposure if the change fails mid-execution.	Block if threshold exceeded. Surface: affected device count, service list, threshold.

4.2 Policy Engine Design Principles

- Deterministic and auditable — the same change plan always produces the same policy result given the same policy database state.
- Human-owned configuration — the policy rules and thresholds are configuration, not model parameters. Change management teams update ranges, schedules, and thresholds without touching the pipeline.
- Block before human review — a change that violates policy should not consume human review time. The policy engine runs before the approval gate so that blocked changes never appear in the approval queue.
- Advisory vs. blocking — not all policy conditions are hard blocks. The active maintenance window condition is advisory: the change is permitted but the approver

sees the warning. Hard blocks are reserved for conditions where proceeding would definitely violate organizational policy.

- Extensible without pipeline changes — adding a new policy type means adding a new policy checker function and registering it with the engine. The pipeline, the approval interface, and the audit logger do not change.

5. Rollback Architecture

Rollback is a first-class architectural concern in this pattern, not an error handler. The distinction is significant: an error handler is built when something goes wrong; a rollback plan is built when the change plan is built. By the time a post-check failure occurs, the rollback plan already exists, has been reviewed as part of the human approval package, and is immediately executable.

5.1 Rollback Plan Generation

The change planner generates rollback commands for every forward command at plan time, in the correct reverse dependency order. The rollback plan is stored in the Change Package alongside the forward plan and presented to the human approver. The approver is authorizing both the change and its rollback procedure.

Rollback dependency order — why order matters: The correct rollback order is the inverse of the forward dependency chain, not simply the reverse of the device list. For an infrastructure change with multiple dependent configuration layers, the safe rollback sequence stops dependent traffic first, removes the gateway or intermediary configuration second, and cleans up the base configuration last. Removing base configuration before removing dependencies that reference it leaves orphaned configuration in an indeterminate state. The change planner must generate rollback commands in this safe sequence. The plan validator checks rollback ordering independently from rollback completeness.

5.2 Rollback Trigger Conditions

Trigger	Response
Simulator post-check failure	Execution stops. Rollback commands from Change Package presented to human reviewer. Change Package outcome written as simulator-failed. Production-ready status not granted.
Mid-execution error	Partial execution state recorded in Change Package execution_log. Rollback plan identifies which commands succeeded. Partial rollback guidance presented to reviewer with the specific device and command that failed.
Human-initiated rollback	The approver or a post-change reviewer may initiate rollback at any point after execution. The rollback plan is always available in the Change Record Store via change_id.

v1 boundary: rollback is recommended, not automated. In v1, the system recommends rollback and provides the exact commands — it does not execute rollback autonomously. A system that automatically reverts infrastructure changes without human authorization introduces a risk profile that requires explicit organizational acceptance. A system that provides rollback commands, labeled and ready, and requires a human to execute them is a

more defensible first deployment. Automated rollback is the documented v2 enhancement path.

6. Failure Modes and Mitigations

Failure Mode	Likelihood	Mitigation
LLM generates syntactically invalid commands	Medium	Plan validation syntactic check catches invalid commands. One retry with explicit correction prompt. If second attempt fails, block with validation_failed flag. Log raw LLM output for prompt review.
LLM generates incorrect dependency ordering	Low–Medium	Dependency ordering validation in the validate_node detects and flags out-of-order commands. Explicit ordering rules in the plan prompt and plan validator.
Rollback plan incomplete	Low–Medium	Rollback completeness check verifies one-to-one forward/rollback coverage. Rollback ordering check verifies correct sequence independently. Both block progression if failed.
Requestor not authorized for target group	Low–Medium	Authorization scope check is the first policy check. Block before any other policy check runs. Surface: specific role boundary exceeded.
Policy database stale	Medium	Freshness timestamp checked at policy_node entry. Stale database (over 24 hours) generates advisory warning in Change Package. Operations team owns database currency.
Simulation environment unreachable	Low–Medium	Pre-check and execution nodes handle connection errors explicitly. Connection failure surfaces as a pre-check failure with specific device and error. Does not proceed to approval with incomplete baseline.
Approver anchor bias	Medium	Interface design presents the configuration delta view prominently. Approver sees before/after state before commands. Approval UI must not bury the delta view.
Human approval gate becomes a bottleneck	Medium	Before production, estimate expected approval volume under realistic change distributions. Define staffing and SLA expectations. Policy engine filtering reduces the load on human review by blocking non-compliant changes before they reach the queue.

7. Security and Governance

Security review should occur before implementation design is finalized. This architecture touches sensitive operational data, infrastructure credentials, audit logs, LLM hosting decisions, and potentially cloud inference. The sections below are intended to support early security review, not to substitute for it. Engage security and compliance stakeholders at architecture review stage — late-stage findings may require architectural rework that early engagement would have avoided.

7.1 Data Classification

Change request data flowing through this architecture may contain sensitive operational information: device identifiers, infrastructure topology details, configuration specifics, and asset criticality ratings. The classification of this data determines where the LLM may run and what audit retention applies.

- If change data is classified at a level that prohibits cloud transmission, the LLM must run locally. Cloud API-based inference is not permissible for this data class.
- The Change Package carries change content through every phase. Transit encryption (TLS) and encryption at rest must match the data classification of the changes being processed.
- Change Record Store entries contain the full command log and all approver records. Access controls must match or exceed the classification of the source changes.

7.2 Access Controls

- The human approval interface must enforce role-based access. Approvers see full command detail; that access must be scoped to their authorization domain.
- The policy engine requires read access to authorization scope tables, assignment databases, maintenance schedules, and CMDB data. These credentials must be scoped to read-only.
- Change Record Store access for compliance review should be segregated from operational access. Audit reviewers should not be able to modify Change Package records.
- The pipeline execution layer requires credentials for the simulation environment and production infrastructure. These credentials must be managed with appropriate rotation and vault integration.

7.3 Human Approval Gates

This architecture contains one non-bypassable human decision point as a structural property: the human approval gate between planning and execution. This is a structural INTERRUPT in the pipeline, not a configuration threshold.

Modifying the human approval gate changes the risk posture. The approval gate is a structural INTERRUPT in the pipeline. Removing it requires modifying the pipeline architecture, not adjusting configuration. An organization may choose to modify this architecture for specific automated change categories. That is a legitimate architectural choice. It is also a change to the risk posture described here. Any modified architecture should be evaluated against its own failure mode profile before production deployment.

7.4 Audit Retention

- At minimum, retain the Change Record Store long enough to support post-incident review of any change made during the preceding period.
- In regulated verticals (healthcare, financial services, critical infrastructure), consult applicable regulatory standards for change and decision record retention requirements.
- Change Record Store entries must be tamper-evident. Write-once or append-only storage is appropriate for the compliance record backing.
- Align audit retention policy with any applicable change management framework requirements (CAB, ITIL, SOC 2, etc.).

8. Implementation Options

8.1 LLM Hosting

The choice of LLM hosting has direct implications for command generation quality, classification SLA, data classification policy compliance, and operational cost. Before committing to a hosting model, validate command generation quality for the target change type vocabulary. The two LLM phases (interpretation and planning) are the only inference-dependent steps — all other phases are deterministic.

Option	When to Choose
Local model serving (e.g., Ollama + open-weight model)	When data classification prohibits cloud transmission, when per-change inference cost at volume is a constraint, or when the organization requires air-gapped operation. Requires GPU infrastructure sized to maintain planning SLA. Benchmark command generation quality against the target change type vocabulary before production commitment.
Cloud API (OpenAI, Anthropic, Azure OpenAI, etc.)	When managed availability, model quality, and reduced infrastructure burden outweigh per-change cost and data residency constraints. Appropriate when change data can be transmitted to the cloud provider under the organization's data agreements.
Private cloud / on-premises GPU cluster	When the organization requires cloud-scale model quality with on-premises data residency. Appropriate for large-scale deployments in regulated verticals.

8.2 Simulation Environment

Option	When to Choose
Network simulation (e.g., Cisco CML, EVE-NG, Containerlab)	When the target production environment is network infrastructure. Simulator fidelity must be sufficient for post-check verification results to be meaningful predictors of production behavior.
Infrastructure-as-code staging (e.g., Terraform plan + apply to a staging workspace)	When the target environment is cloud infrastructure managed through IaC. The staging workspace serves as the simulation tier.
Configuration management dry-run (e.g., Ansible check mode)	When the target environment is server or application configuration. Dry-run mode validates the plan without executing changes.
Shadow environment / canary deployment target	When a representative non-production environment is maintained for change validation. The shadow environment receives changes first; production follows on human authorization.

8.3 Change Record Store

Option	When to Choose
SQLite	Reference implementations, proofs of concept, or low-volume deployments. JSON serialization is identical to PostgreSQL — migration is a deployment concern, not a schema change.
PostgreSQL	Production environments with concurrent access requirements, high change volume, or compliance mandates requiring a durable relational store.
Enterprise CMDB / ITSM integration (e.g., ServiceNow)	When the organization requires Change Package data to be written to an existing ITSM or CMDB system for CAB visibility and change tracking. Integration via API write-back from the audit_node.

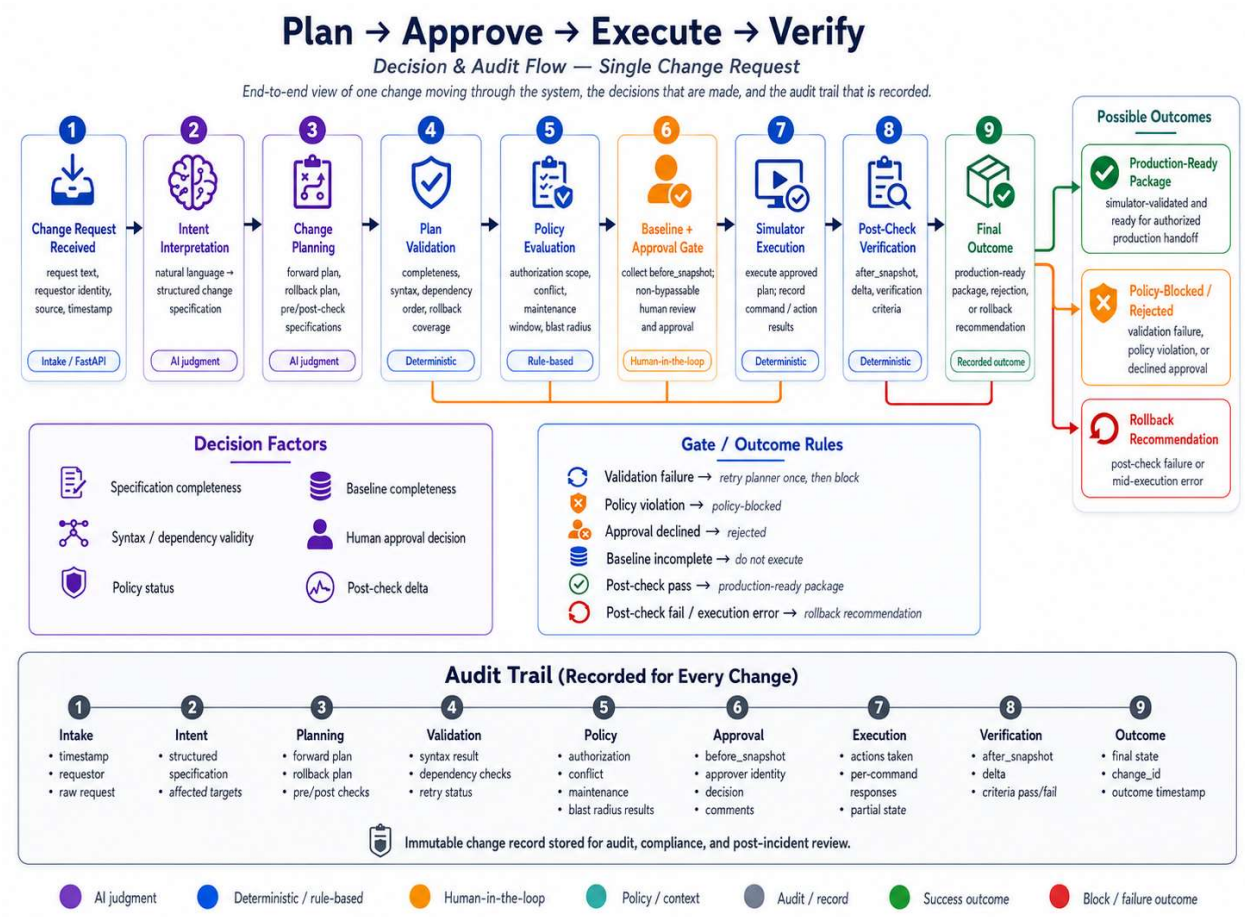
9. Auditability and Explainability

9.1 What the Audit Record Contains

The Change Record Store entry for every processed change includes:

- Original natural language change request, verbatim, as received.
- Change specification output from the intent interpreter: all structured fields.
- Schema validation result at the plan boundary.
- Complete forward plan: all commands per device in dependency order.
- Complete rollback plan: all rollback commands per device in correct reverse dependency order.
- All four policy check results with timestamps: authorization scope, change-type conflict, maintenance window, blast radius.
- Before_snapshot: pre-check baseline from all target devices.
- Approval record: approver identity, decision, reason if rejected, timestamp.
- Execution log: per-command execution records with device, command, response, timestamp, and success flag.
- Post-check results: after_snapshot, delta against before_snapshot, verification criterion results.
- Final outcome: simulator-validated/production-ready, failed, rejected, or policy-blocked.

Figure 3: Single Event Decision and Audit flow — Plan → Approve → Execute → Verify



9.2 Explainability Model

The architecture produces an explainable change record at every step because:

- LLM output is constrained to a defined schema. There is no free-form narrative to interpret. Every planning decision can be traced to specific field values in the change specification.
- Policy checks are deterministic and individually logged. A reviewer can replicate any policy determination given the same policy database state.
- The human approval record captures approver identity and timestamp alongside the exact Change Package that was reviewed. The approver is accountable for what they saw.
- The execution log is per-command. Every command sent to every device, and every response received, is retained in the Change Package.

LLM reasoning is not in the execution path. This architecture does not use LLM-generated reasoning narratives to explain change decisions. Explainability comes from structured, deterministic records — the change specification, the policy check results, the plan validation output, and the execution log. An LLM-generated explanation of its own planning decision is neither reliable nor auditable. This architecture produces explainability through structure, not through narrative.

9.3 Compliance-Oriented Audit Queries

The Change Record Store supports the following query patterns without requiring LLM involvement:

- For a given change: show the complete lifecycle record — who requested it, who approved it, what commands ran, what verification criteria passed or failed.
- For a given device or system: show all changes targeting it in a date range, with outcomes.
- For a given approver: show all approval decisions made, with the Change Package state visible at the time of each decision.
- For a given policy block: show which policy fired, which rule was violated, and what was surfaced to the requestor.
- For a given time window: show all changes that were policy-blocked versus approved versus rolled back, with outcomes.

10. Differentiation and Limitations

10.1 What This Architecture Does Differently

- Non-bypassable human approval. The approval gate is a structural INTERRUPT, not a configuration threshold. No execution path reaches infrastructure without an approver who has seen the exact commands, the configuration delta, and the rollback plan.
- Rollback generated at plan time, not at failure time. The rollback plan exists before any execution occurs. By the time post-check failure surfaces, the rollback commands are already reviewed, approved, and immediately executable.
- Explicit LLM/deterministic boundary. Two phases use model judgment; seven are deterministic. This boundary is documented at the architecture level, not left to inference. A change advisory board can evaluate exactly where human-equivalent reasoning is involved.
- Policy engine runs before human review. Blocked changes never consume human approval time. The approval queue contains only changes that are policy-compliant, structurally valid, and have a complete rollback plan.
- The Change Package is a first-class compliance artifact and a primary audit record. It is distinct from operational telemetry, versioned and immutable, and queryable by change_id. It is what a compliance auditor reviews, not what an implementation engineer debugs. In most production environments it will integrate with or complement existing systems of record such as ITSM, SIEM, or CAB tooling.
- Honest failure mode documentation. The plan validation boundary is explicit about what it does and does not prove. The human approver sees the validation disclaimer. Production safety is the approver's responsibility, not the pipeline's claim.

10.2 Limitations

- Plan validation does not prove production safety. Syntactic correctness and simulator verification confirm that the plan is structurally sound and passed defined checks. They do not confirm that the change is appropriate for the specific production environment beyond those checks.
- LLM command generation quality is change-type dependent. Complex or rarely-seen change types may require more prompt engineering iteration than common change types. Validate command generation quality against the target change vocabulary before production commitment.
- Simulator fidelity is an implementation prerequisite. The simulation environment must be close enough to production that simulator post-check results are meaningful. A simulator that does not accurately represent production behavior produces false confidence.
- Automated rollback is not included in v1. The rollback plan is provided as immediately executable commands. Human initiation of rollback is required. Organizations that require automated rollback execution must implement that as an explicit v2 enhancement with its own risk assessment.
- Policy engine effectiveness is proportional to policy database quality. Stale authorization tables, stale assignment databases, or missing maintenance window records reduce policy engine reliability. Operations teams must own database currency.

- This architecture plans, approves, executes, and verifies changes. It does not manage downstream incident workflows, escalation, or change calendar coordination. Integration with ITSM systems (ServiceNow, Remedy, etc.) is a documented implementation option, not a built-in capability.
- Human approval can degrade under volume. The non-bypassable approval gate is the architecture's primary safety mechanism, but its effectiveness depends on the approver actually reviewing the configuration delta, commands, policy results, and rollback plan. Under high approval volume, time pressure, or a noisy UI, the gate can become a rubber stamp. This is a systemic risk, not a UI problem. Production deployments must validate staffing capacity, approval SLA, and UI legibility before treating the approval gate as a functioning control. Policy engine filtering reduces approval volume by blocking non-compliant changes before they reach the queue — that filtering must be operational before approval volume becomes meaningful.

11. Production Adaptation Checklist

Before committing to implementation, validate each of the following. These are organizational and architectural decisions that must be made before implementation decisions will hold. Skipping any of these produces a deployment that will require rearchitecting under operational load.

Security review (Item 9 below) should occur before implementation design is finalized, not at deployment.

Item	Validation Question	Owner
Data Classification	What is the classification level of change data flowing through the pipeline? Is cloud LLM inference permissible, or is local hosting required? What classification applies to the Change Record Store?	Security / Compliance
Integration Systems	Which systems will the policy engine connect to (CMDB, assignment databases, maintenance schedules)? What APIs, credentials, and rate limits apply?	Architecture / Operations
Workflow Ownership	Who owns the human approval function? What is the expected approval volume? What are the SLA expectations for approval turnaround? Is the approval gate operationally staffed at projected volume?	Operations / Change Management
Model-Hosting Policy	Is cloud API inference permitted for this change data class? If not, what local GPU infrastructure and model serving platform will be used? Has command generation quality been benchmarked for the target change vocabulary?	Architecture / Security
Human Escalation Policy	What conditions trigger the rollback recommendation beyond the default post-check failure? Who is authorized to initiate rollback execution? Does the rollback policy align with existing incident management procedures?	Operations / Change Management
Audit Retention	What is the required retention period for Change Record Store entries? Does the backing store meet tamper-evidence requirements? Are there applicable regulatory standards for change record retention?	Compliance / Operations
Access Controls	Who has access to the approval interface, the Change Record Store, and the policy engine configuration? Are approver roles	Security / Operations

	mapped to the organization's identity provider? Are pipeline credentials rotation-managed?	
Latency / SLA Needs	What is the acceptable end-to-end change cycle time from request submission to production-ready status? Does the simulation environment execution time fit within this SLA? Has LLM planning latency been benchmarked?	Architecture / Operations
Operational Support Model	Who monitors the pipeline, the simulation environment, and the Change Record Store? What alerting exists for validation failures, policy engine failures, and simulation environment connectivity issues? Does the team have tooling to diagnose Change Packages from audit records?	Operations

12. Component Reference

The following section defines each component in the logical architecture: its role, responsibilities, failure modes, and category. Color coding in the logical architecture diagram corresponds to component categories:

- Purple (AI Judgment) — Intent Interpreter, Change Planner
- Teal (Rule-Based Logic) — Plan Validator, Policy Guardrail Engine, Pre-Check Collector, Simulator Execution, Simulator Post-Check
- Amber (Human In The Loop) — Human Approval Gate, Rollback Recommendation Interface
- Coral (Audit and Record) — Change Record Store, Audit Logger
- Gray (System Artifact) — Change Package

Intent Interpreter *AI Judgment*

Role: First LLM phase. Receives the natural language change request and produces a structured change specification conforming to a defined schema. This is where human intent becomes a machine-executable payload. The only phase where free-text interpretation occurs.

Responsibilities:

- Parse and interpret the natural language request.
- Infer affected devices or systems from group names or contextual identifiers using topology or inventory configuration.
- Extract or infer all required schema fields, setting ambiguity flags on fields that cannot be determined with confidence.
- Produce schema-valid output enforced via structured output parameter — not prompt instruction alone.
- Retry once with an explicit repair prompt on parse failure or missing required fields.

Not responsible for: Generating execution commands (that is the change planner's role). Policy checking. Device connectivity or reachability.

Key failure modes: Ambiguous device group reference (return schema_error, do not infer); required parameters not stated in the request (return ambiguity flag); JSON parse failure after retry (route to audit_node with interpretation_failed flag).

Change Planner *AI Judgment*

Role: Second and final LLM phase. Expands the validated change specification into a full Change Package: device-by-device commands in forward dependency order, rollback

commands in correct reverse dependency order, pre-check command specifications, and post-check success criteria. After this phase, no further LLM judgment occurs in the pipeline.

Two-phase LLM design: Separating interpretation from planning is a deliberate architectural choice. A single prompt that receives a natural language request and generates commands directly produces commands that are difficult to validate because the interpretation step is invisible. Separating the steps means the intent interpreter's output can be inspected and the change planner's output can be validated against the specification. Two smaller, focused prompts are more reliable and more auditable than one large opaque one.

Rollback generation: The change planner generates rollback commands for every forward command at plan time, in the correct reverse dependency order. The dependency order is specified in the planner prompt and verified by the plan validator. Rollback dependency ordering is checked independently from rollback completeness.

Key failure modes: Syntactically invalid commands (plan validator catches, one retry); incorrect dependency ordering (plan validator's ordering check flags); incomplete rollback coverage (plan validator blocks progression); wrong parameter assignments for device roles (caught by plan validation if parameters are specified in the change specification schema).

Plan Validator *Rule-Based Logic*

Role: Deterministic correctness gate between the change planner and the policy engine. Applies five structured checks to the Change Package before any policy evaluation or human review occurs. A plan that fails validation is returned to the change planner for one retry; if it fails again it is blocked with a validation_failed flag.

Five validation checks:

- Specification completeness — all required schema fields populated.
- Syntactic correctness — command structure valid for the target system; parameter value ranges checked.
- Dependency ordering — commands are in the required execution sequence per device and across devices.
- Rollback completeness — every forward command has a corresponding rollback command.
- Rollback dependency ordering — rollback commands are in the correct reverse dependency order. Checked independently from completeness because a complete but incorrectly ordered rollback is still an unsafe rollback.

Not responsible for: Operational correctness beyond the five defined checks. Policy evaluation. Fixing errors — validation failures are reported to the change planner for retry, not auto-corrected.

Policy Guardrail Engine *Rule-Based Logic*

Role: Deterministic gate between the plan validator and the human approval gate. See Section 4 for full specification. The approval queue contains only changes that have passed all four policy gates.

Key design principle: The policy rules — authorization scope tables, assignment ranges, maintenance window schedules, blast radius thresholds — are human-maintained configuration. Change management and security teams own these rules. Changes to policy do not require model retraining or pipeline modification.

Pre-Check Collector *Rule-Based Logic*

Role: Establishes the before_snapshot — the baseline state of all target devices or systems before any change commands execute. The before_snapshot is the reference point for the post-check delta comparison.

Before_snapshot structure: Stored in the Change Package as a structured dictionary keyed by device or system identifier. Each entry contains parsed output from each pre-check command. The post-check uses the same parsing logic to produce the after_snapshot and computes the delta.

Key failure modes: Simulation environment device unreachable (surfaces as pre-check failure, pipeline does not proceed to approval with incomplete baseline); command output unparseable (flagged with specific device and command); unexpected pre-existing state (advisory warning in Change Package for human approver, not a hard block).

Change Package *System Artifact*

Role: The central architectural artifact of this pattern. Not a processing node — a versioned, immutable record created at the end of the change planner phase that travels through every subsequent phase. See Section 2.4 for full field specification.

Immutability model: Fields are marked immutable at the phase that produces them. The forward_plan and rollback_plan are immutable after human approval. The before_snapshot is immutable after collection. The approval_record is immutable after the approver acts. Execution_log and postcheck_results are append-only. The outcome field is written once at the terminal phase.

Distinction from the operational audit log: The LangGraph checkpointer persists pipeline state at every node transition — that is operational telemetry. The Change Package is a named, versioned, queryable compliance artifact. Both exist; they serve different audiences.

Human Approval Gate *Human In The Loop*

Role: The non-bypassable gate between planning and execution. A LangGraph INTERRUPT point. The approver authorizes a specific set of infrastructure writes — not a routing recommendation. The exact commands that will execute are visible before the approver acts.

Six required approval package elements:

- Commands per device in monospace blocks, in forward execution order.
- Configuration delta view — before/after state per device derived from the before_snapshot and the change specification. This is the element that makes the approval meaningful.
- All four policy check results with pass/advisory/block indicators.
- Pre-check baseline table showing current state before the change.
- Rollback plan per device, expandable, in correct dependency order.
- Execution target label and change_id for reference.

Non-bypassability: There is no architectural path from the plan phase to the execution phase that does not pass through the approval gate. This is a structural property of the pipeline graph, not a configuration option.

Simulator Execution *Rule-Based Logic*

Role: Executes the approved Change Package against the simulation environment. Commands are issued in the forward dependency order defined in the Change Package. Every command and its response is logged to the Change Package execution_log. The execution layer does not interpret, modify, or reorder commands — it executes the approved plan as-is.

Deterministic execution model: The only inputs are the approved Change Package and the simulation environment connections. If a command fails, the failure is logged, execution halts, and the pipeline routes to the rollback recommendation node. There is no on-the-fly command adjustment and no model involvement.

Simulator Post-Check and Production-Ready Node *Rule-Based Logic*

Role: Runs verification criteria against all target systems after simulation execution completes. Computes the after_snapshot and the delta against the before_snapshot. If all criteria pass, the Change Package is marked simulator-validated / production-ready.

The production-ready state: Simulator-validated / production-ready means the change has been validated against the simulation environment and all post-check criteria passed. It is not a claim that the change will succeed on production infrastructure without exception; it is a claim that the change succeeded on the simulation environment running equivalent software and configuration. Production safety is the approver's responsibility.

Rollback Recommendation Interface *Human In The Loop*

Role: Activated when simulator post-check verification fails or a mid-execution error is detected. Presents post-check failure details and the rollback plan from the Change Package

to a human reviewer. In v1, the system recommends rollback and provides copy-ready commands — it does not execute rollback autonomously.

What the reviewer sees: Post-check failure details (which criterion failed, actual vs. expected output); delta comparison; partial execution state if mid-execution error; rollback plan in correct dependency order with copy-to-clipboard per step; change_id reference; estimated rollback blast radius.

Change Record Store *Audit and Record*

Role: Persistent store for all Change Packages. Every completed Change Package — regardless of outcome — is written to the Change Record Store at the audit_node terminal step. The Change Record Store is a primary compliance artifact: queryable by change_id, requestor, device, date range, and outcome. In most production environments it will integrate with or complement existing systems of record. See Section 8.3 for implementation options.

Distinction from the LangGraph audit log: The LangGraph checkpointer records pipeline execution mechanics (operational telemetry). The Change Record Store retains the authorization chain, approved commands, pre-change baseline, execution log, and verification results (compliance record). Both exist; they serve different audiences.

Audit Logger *Audit and Record*

Role: Terminal for all execution paths. The audit_node runs at the end of every pipeline path: successful completion, simulator post-check failure, policy block, human rejection, and validation failure. It writes the final Change Package state to the Change Record Store and writes the LangGraph pipeline execution state to the operational checkpointer. No path exits the pipeline without passing through audit_node.

Component Summary

Component	Category	Key Architectural Decision
Intent Interpreter	AI Judgment	LLM. Two-phase design: interpretation separate from command generation to make each step auditable independently.
Change Planner	AI Judgment	LLM — last LLM phase. Rollback generated at plan time. Rollback dependency order is checked by plan validator independently from completeness.
Plan Validator	Rule-Based Logic	Deterministic. Five checks: completeness, syntactic, dependency ordering, rollback completeness, rollback ordering. Rollback order checked independently.
Policy Guardrail Engine	Rule-Based Logic	Deterministic. Four checks in sequence: authorization scope first. Blocked changes never reach a human approver.

Pre-Check Collector	Rule-Based Logic	Deterministic. Establishes before_snapshot from all target devices. Pipeline does not proceed to approval with incomplete baseline.
Change Package	System Artifact	Not a processing node. Immutability model: fields locked at the phase that produces them. Distinct from LangGraph operational telemetry.
Human Approval Gate	Human In The Loop	INTERRUPT. Non-bypassable structural property. Six required elements including configuration delta view. Approver sees before/after state, not just commands.
Simulator Execution	Rule-Based Logic	Deterministic. Executes commands as-is from approved Change Package. No modification, no retry with alternatives. Any failure routes to rollback recommendation.
Sim Post-Check + Production-Ready	Rule-Based Logic	Deterministic. Production-ready ≠ production-executed: deferred, not simulated. Post-check criteria are defined at plan time.
Rollback Recommendation	Human In The Loop	Dependency-ordered sequence shown explicitly. v1: human executes. Automated execution is v2 capability.
Change Record Store	Audit and Record	Distinct from LangGraph audit log. Compliance artifact vs. operational telemetry. See Section 8.3 for implementation options.
Audit Logger	Audit and Record	Terminal for all paths. Writes both Change Record Store and LangGraph checkpointer in one node. No path exits without passing through audit_node.

Appendix A: Demonstration Scenarios

The following scenario types exercise the full range of pattern behavior. Any implementation of this pattern should demonstrate all three scenario types to validate that the architecture handles the complete range of input conditions, including both success and failure paths.

A.1 Clean Change Execution with Symmetric Verification

A complete change request is submitted for a multi-device configuration update with defined pre and post verification criteria. The system interprets the request, generates a device-by-device command plan with rollback commands in the correct dependency order, checks the plan against all four policy gates, collects the pre-change baseline, presents the full Change Package to a human approver, executes against the simulation environment, runs post-checks, and produces a simulator-validated / production-ready Change Package.

Key validation points:

- The intent interpreter produces a structured specification from a natural language request, with all required schema fields populated.
- The change planner generates commands in the correct forward dependency order with distinct command sets for different device roles.
- The rollback plan is generated at plan time — before any execution — and is visible to the approver in the approval interface.
- The approval interface shows all six required elements: commands, configuration delta view, policy results, pre-check baseline, rollback plan, execution target label.
- Post-check verification confirms the change achieved its intended effect by computing the delta against the before_snapshot.
- The completed Change Package in the Change Record Store contains a complete lifecycle record queryable by change_id.

A.2 Policy Block Sequence

Two change requests are submitted that exercise different policy blocks. The first request uses a requestor identity not authorized for the target device group. The second uses a requestor identity authorized for the target group, but the proposed change parameter conflicts with an existing assignment in the policy database.

Key validation points:

- The authorization scope check runs first. The first request is blocked before any other policy check runs. The requestor sees the specific role boundary exceeded.
- The second request passes the authorization scope check and is blocked at the change-type conflict check. Both blocks produce distinct, specific violation records.

- Neither blocked change reaches the human approval interface. The approval queue contains only changes that passed all four policy gates.
- Both blocked Change Packages are written to the Change Record Store with specific policy_results entries documenting the violation.

A.3 Post-Check Failure with Rollback Recommendation

A change request is submitted and passes all validation and policy gates. The human approver approves the change. Execution completes against the simulation environment. Post-check verification fails — one of the defined success criteria is not satisfied after execution.

Key validation points:

- Post-check failure surfaces a specific failure delta: which criterion failed, what the actual post-check output was, and what the expected output was.
- The rollback recommendation interface appears immediately with the rollback plan from the Change Package — the plan that was generated before execution, not constructed at failure time.
- Rollback commands are shown in the correct dependency order, labeled by step, in copy-ready format per device.
- The Change Package outcome is written as simulator-failed. Production-ready status is not granted.
- The complete post-check delta and rollback recommendation are retained in the Change Record Store under the original change_id.

Appendix B: Vertical Adaptation Notes

The Plan → Approve → Execute → Verify pattern applies wherever organizations manage planned changes to infrastructure or systems that require pre-change validation, human approval, and symmetric pre/post verification. The logical architecture is stable across verticals. What changes is the change vocabulary, the execution tooling, the simulation environment, and the policy engine rules.

Vertical	Example Process	Adaptation Notes
Network Operations	VLAN provisioning, routing protocol changes, ACL updates, interface configuration	The reference implementation scenario. Policy engine enforces VLAN ranges, maintenance windows, blast radius, and authorization scope. CLI execution via Netmiko or NAPALM.
Cloud / DevOps	Terraform plan approval, security group updates, IAM policy changes	Swap 'device commands' for 'Terraform plan / apply'; simulation tier becomes a staging workspace. Pre/post checks become cloud provider state queries. Policy engine enforces account scope and change freeze windows.
IT Operations	Server patch deployment, configuration baseline changes, firewall rule updates	Swap 'network device' for 'server' or 'firewall'. Execution tier uses Ansible or configuration management tooling. Pre/post checks become health checks and configuration state queries.
Healthcare IT	Network segmentation changes, medical device VLAN isolation, clinical system configuration	Compliance audit trail becomes the primary value proposition. Policy engine enforces segmentation policy and change freeze windows. Data classification review required before LLM hosting decision.
Financial Services	Trading network changes, firewall policy updates, DR failover validation	Change freeze windows and CAB integration become prominent policy engine features. Stronger audit retention requirements. Event streaming may be appropriate for high-volume audit event transport; durable Change Package retention still requires an auditable system of record.
Telecommunications	Core network configuration changes, customer circuit provisioning, VPN updates	Direct fit. Telco network operations workflows map closely to the reference scenario. Policy engine scope expands to include circuit impact and service catalog lookups.

Adapting for your vertical: The substitution table above identifies the vocabulary, tooling, and simulation environment changes. In every case, the pattern structure holds: natural language change requests flow through nine phases; the policy guardrail engine enforces organizational constraints; the human approval gate is non-bypassable; rollback is generated at plan time; audit records are complete. What varies is the execution tooling, the simulation environment, the policy rule content, and the post-check verification criteria. A production

engagement requires mapping these to the specific operational context, infrastructure environment, and governance requirements of the organization.