International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

A Durable, Python-Native Orchestration Architecture for Enterprise Automation

Mr. Aryan Singh[1], Mr. Sachin Rathod[2], Ms. Pradnya Suryawanshi[3]

[1]Senior AI Engineer, IT, Arcelor mittal digital consulting
[2,3]Python Developer, IT, Arcelor mittal digital consulting

Abstract

Legacy robotic process automation (RPA) platforms couple process governance to proprietary, canvas-based runtimes that are difficult to version, test, and integrate with modern software delivery pipelines. This paper specifies a Python-native orchestration architecture that reproduces the governance role of a traditional RPA control tower — durable execution, human-in-the-loop task management, role-based access control, and audit logging — while building on open-source durable-execution primitives (Temporal, Prefect) rather than a bespoke runtime. The architecture is decomposed into four layers: presentation, orchestration and execution, intelligence and observability, and data and infrastructure. We describe the core execution engine's approach to checkpointing and deterministic replay, a decorator-based task abstraction layer for wrapping existing bots and APIs as orchestrated units, and a FastAPI/PostgreSQL human-in-the-loop queue with SLA-based escalation. We further outline the security, compliance, and deployment model required for enterprise adoption, and position the architecture relative to legacy RPA suites and modern data-orchestration frameworks. The intended outcome is a lower-risk, code-first path to enterprise automation for engineering-led organizations already invested in Python tooling.

Keywords: workflow orchestration, durable execution, robotic process automation, human-in-the-loop systems, Python SDK, enterprise governance, observability

Executive Summary

Enterprise automation has outgrown the platforms that built the category. Robotic process automation (RPA) suites such as UiPath, Automation Anywhere, and WorkFusion were designed a decade ago around record-and-replay bots and heavyweight, often Java-based, orchestration consoles. They remain dominant, but engineering organizations increasingly report the same friction: steep learning curves, weak integration between development environments and the orchestration console, and licensing models built for enterprise procurement cycles rather than iterative, code-first delivery.

This paper proposes an architecture for a Python-native orchestration platform that borrows the operational role of a traditional RPA "control tower" — scheduling, recovery, human-in-the-loop task management, and governance — while adopting the engineering practices of modern workflow-orchestration systems such as Temporal and Prefect. The result is a system that a mid-sized engineering team can operate, extend, and audit, without the multi-year integration overhead associated with legacy RPA suites.

The architecture is presented in four layers — presentation, orchestration and execution, intelligence and observability, and data and infrastructure — each mapped to specific, currently available open-source

components. Two reference diagrams accompany the paper: a system architecture view and a process-execution flow view. The paper closes with security, compliance, and deployment considerations relevant to enterprise evaluation.

1. Industry Context

Analyst estimates of the RPA software market for 2025–2026 vary considerably by methodology, ranging from roughly USD 5–7 billion to USD 28 billion depending on scope, with compound annual growth rates cited between 18% and 41% through the early 2030s. Despite the variance in absolute figures, every major forecast agrees on direction: software-led automation continues to grow faster than services, and North America accounts for the largest regional share.

A parallel and faster-moving category has emerged alongside traditional RPA: Python-native workflow orchestration, represented by Apache Airflow, Prefect, Dagster, and Temporal. These tools were built for data and application engineers rather than RPA developers, and their consolidation is accelerating — Prefect's acquisition of Dagster Labs in mid-2026 is one recent signal of the category's maturity and continued investment. This paper's architecture sits at the intersection of the two categories: RPA-style process governance, delivered through the tooling conventions of modern orchestration engines.

2. Design Principles

- Durable by default — every workflow step is checkpointed; a crash, deployment, or restart resumes execution rather than restarting a process from the beginning.
- Code-first, not canvas-first — processes are defined as versioned Python code, reviewed and deployed through standard CI/CD, addressing the most common complaint about legacy RPA consoles.
- Human-in-the-loop as a first-class citizen — manual approval and exception-handling steps are modeled with the same rigor as automated steps, not bolted on afterward.
- Composable over monolithic — the platform orchestrates existing bots, scripts, and APIs rather than requiring a full rewrite of an organization's automation estate.
- Enterprise governance from day one — RBAC, audit logging, and secrets management are architectural requirements, not later add-ons.

3. System Architecture

The platform is organized into four layers. The presentation layer exposes dashboards, an administrative console, and both an API and a Python SDK for developers. The orchestration and execution layer hosts the durable execution engine, the human task queue, bot and connector runners, and the scheduler. The intelligence and observability layer adds anomaly detection, an optional natural-language process designer, and centralized metrics, logging, and audit capture. The data and infrastructure layer provides the persistence, messaging, secrets management, and container runtime that the other layers depend on.

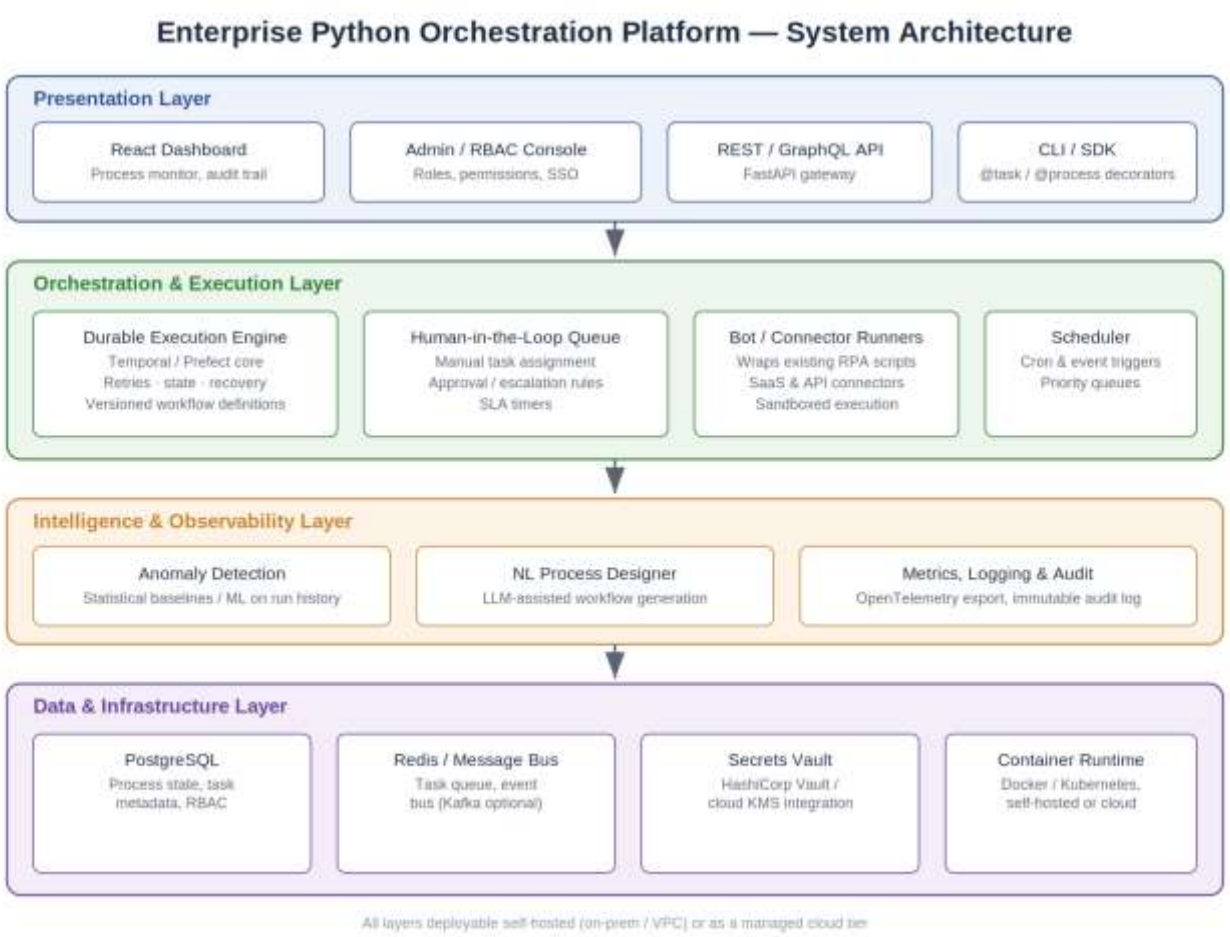

Figure 1. System architecture — presentation, orchestration, intelligence, and infrastructure layers.

3.1 Core Engine

The durable execution engine is not built from scratch. It is built on an existing open-source primitive — Temporal's Python SDK or Prefect's orchestration core — both of which already solve retry semantics, state persistence, and failure recovery to a standard that would otherwise take a dedicated team well over a year to reach. This decision is the single largest determinant of whether the architecture is viable on a realistic engineering timeline.

Mechanically, each workflow run is represented as an append-only event history rather than as mutable process state. Every activity invocation, timer, and signal is recorded as an event; on worker restart or crash, the engine replays this history against the workflow's deterministic code path to reconstruct in-memory state before resuming — the same event-sourcing pattern Temporal uses internally. Individual activities (bot invocations, API calls) run outside this deterministic boundary and are retried independently under a configurable policy (initial interval, backoff coefficient, and maximum attempts), so a transient downstream failure does not require replaying the entire workflow. Workflow definitions are versioned explicitly; the engine pins an in-flight run to the code version it started under so that a mid-flight deployment cannot silently change the semantics of a running process.

3.2 Task Abstraction Layer

A thin Python SDK, using decorator syntax modeled on Prefect's @flow and @task pattern, allows an engineering team to wrap existing RPA scripts, internal APIs, or SaaS connectors as orchestrated units

without rewriting the underlying automation. This is the primary integration point for enterprises with existing automation investments.

Each task is declared with a typed input/output contract, expressed as a Pydantic model or equivalent JSON Schema, which is validated at registration time and again at invocation, catching interface drift before it reaches production rather than at runtime. Existing RPA bots are wrapped as connectors that shell out to, or communicate over gRPC with, the underlying bot runner, so no bot logic needs to be rewritten to participate in orchestration. Task-level configuration — timeout, retry policy, idempotency key, and required scopes for secrets access — is declared alongside the task definition and reviewed through the same pull-request process as any other code change, giving process changes the same change-management trail as application code.

3.3 Human-in-the-Loop Queue

A dedicated task-assignment and approval module, built on FastAPI and PostgreSQL, manages hand-offs between automated steps and human reviewers, including SLA timers and escalation rules. This directly addresses a capability that generic data-orchestration tools such as Airflow and Dagster do not provide natively.

Work items enter the queue as signals against the parent workflow rather than as independent database records, which keeps the assignment, approval, and any subsequent automated step causally linked in a single execution history. Assignment supports both direct routing to a named reviewer and pooled routing to a role or team, with claim locking to prevent two reviewers from acting on the same item concurrently. SLA timers are implemented as durable workflow timers rather than external cron jobs, so an escalation fires reliably even if the queue service itself restarts. Reviewer decisions are captured with the reviewer's identity, timestamp, and rationale, and are replayed into the audit log described in Section 5.1.

4. Process Execution Flow

Figure 2 traces a single task from trigger to completion. A trigger — a schedule, an inbound event, or an API call — loads a versioned process definition into the execution engine. The engine dispatches the next step either to an automated bot or connector, or to the human task queue. Regardless of path, the result updates persisted state and the audit log. A failed step is retried or recovered automatically by the durable execution engine rather than failing the entire process; a successful step advances the process definition or closes the run, with the dashboard updated in real time.

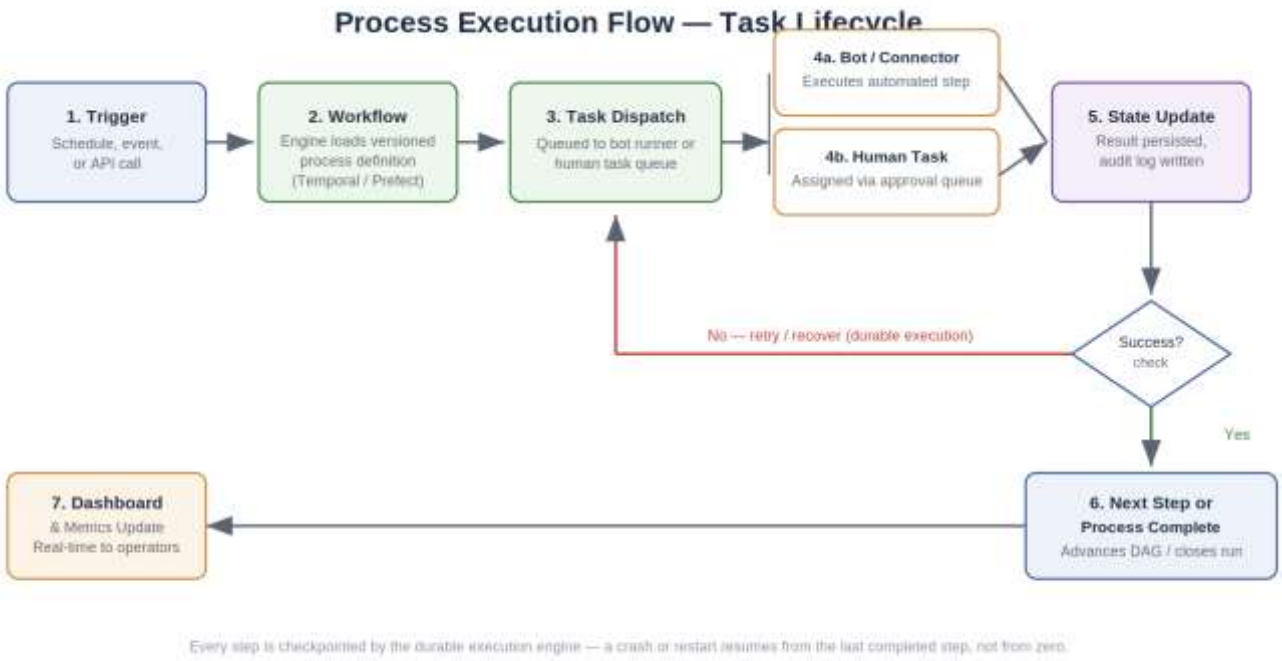

Figure 2. Process execution flow, including automated and human-in-the-loop branches and failure recovery.

5. Enterprise and Industry Considerations

5.1 Security and Compliance

- Secrets (API keys, credentials) are never stored in workflow code or database rows; they are resolved at runtime from a vault (HashiCorp Vault or a cloud provider's KMS) using short-lived, scope-limited leases, and are redacted from logs and workflow history by pattern-matching known secret shapes before persistence.
- All process state transitions and human approvals are written to an append-only audit log, supporting SOC 2 and internal-audit requirements common in regulated industries such as financial services and insurance.
- Role-based access control, enforced through policy-as-code (e.g., Open Policy Agent) rather than hard-coded checks, governs who can view, modify, or approve a given process or task, mirroring the governance model enterprises already expect from RPA platforms.

5.2 Deployment Model

The platform is designed to be deployable self-hosted, inside a customer's own VPC or on-premises environment for regulated customers, or consumed as a managed cloud service for customers who prefer not to operate the infrastructure themselves. This dual-deployment model mirrors how Prefect, Temporal, and n8n structure their own open-core businesses.

5.3 Interoperability

Rather than replacing an enterprise's existing RPA bots outright, the platform is designed to orchestrate them — treating a UiPath or WorkFusion bot as one connector type among several. This lowers adoption risk considerably: an organization can introduce the platform for new processes or for processes underserved by its existing tools, without a disruptive rip-and-replace migration.

6. Comparative Position

Category	Representative products	This architecture's position
Legacy RPA suites	UiPath, Automation Anywhere, WorkFusion	Interoperates with their bots; targets teams underserved by their orchestration console
Data-centric orchestration	Apache Airflow, Dagster	Adds human-in-the-loop and business-process framing they lack
Modern durable orchestration	Temporal, Prefect	Builds on their engines rather than competing with them directly

7. Conclusion

The architecture described in this paper does not attempt to out-build the incumbents of the RPA industry on their own terms. It instead applies the engineering discipline of modern Python orchestration frameworks — durability, versioned code, observability — to the specific governance and human-in-the-loop requirements that enterprise automation teams have always needed from a control tower. For organizations already investing in Python engineering talent, this represents a materially lower-risk path to enterprise-grade automation orchestration than either adopting a legacy RPA suite wholesale or accepting the gaps in general-purpose data-orchestration tools.