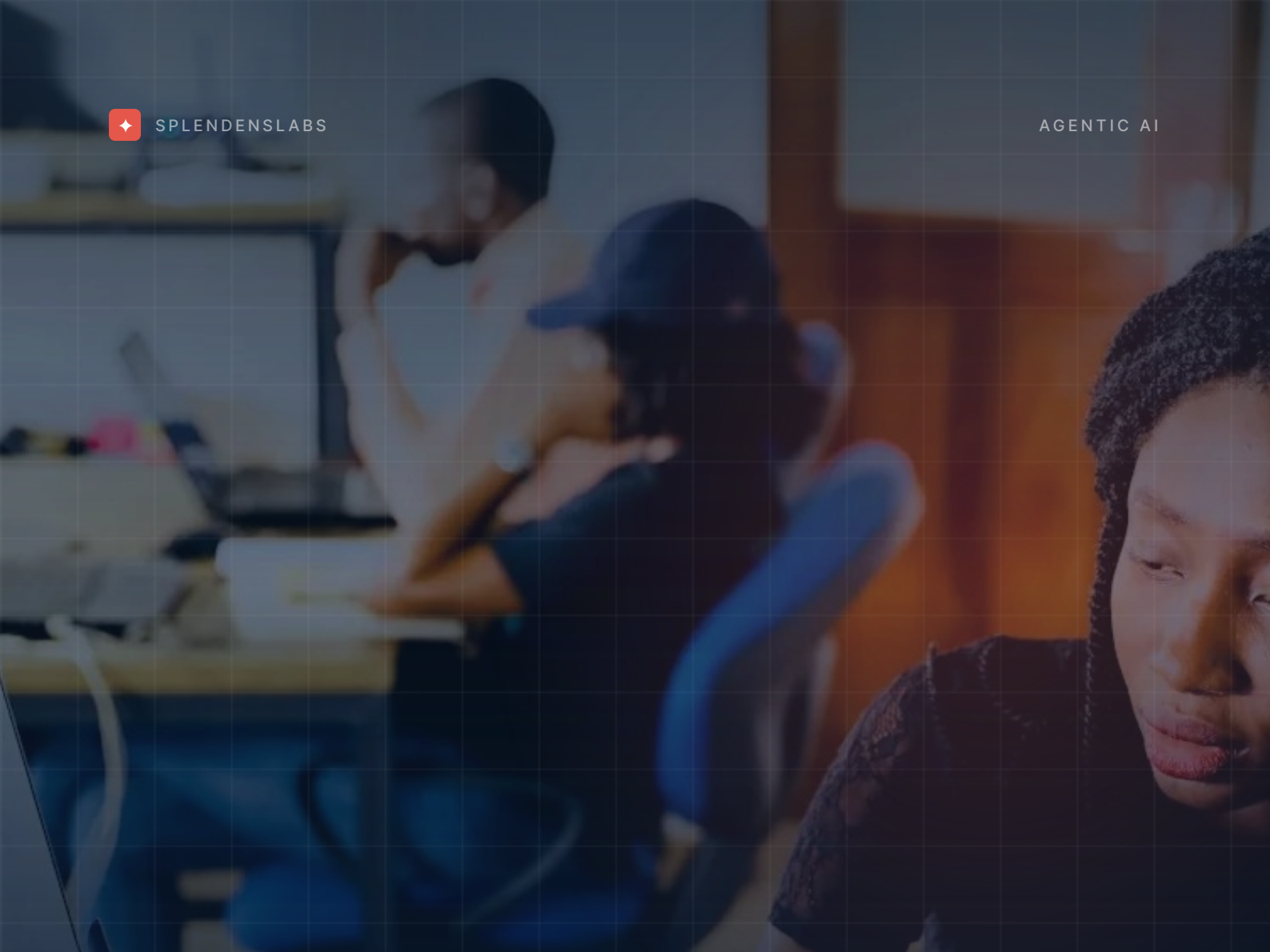


✦ REFERENCE ARCHITECTURE · AGENTIC AI

Agentic Workflow Architecture

How to orchestrate planner, executor and reviewer agents on an explicit state machine, with policy checks, human approval, failure recovery and full traceability.

FORMAT	LEVEL	READ TIME	UPDATED
Reference Architecture	Advanced	22 min	Apr 2026

OVERVIEW

What this reference architecture covers

An agent that loops on a prompt until it decides it is done works in a demo and becomes unpredictable in production. This reference architecture treats an agentic workflow as a durable state machine: every step is a node, every transition is explicit, every side effect passes a policy check, and every run can be paused, resumed, replayed and explained.

KEY TAKEAWAYS

- Model the workflow as an explicit graph of states and transitions; let the LLM choose among allowed edges, not invent them.
- Separate planning, execution and review into distinct roles with distinct prompts, tools and permissions.
- Every tool call with a side effect goes through a policy layer that can allow, deny or escalate to a human.
- Checkpoint state after every node so runs survive crashes, deploys and multi-day human approvals.
- Bound everything: steps, tokens, wall-clock time, retries and spend per run.
- Trace each run as a tree of spans so any decision can be reconstructed after the fact.

WHO IT'S FOR

- Platform and AI engineering leads
- Engineers building multi-step agents
- Security, risk and compliance reviewers
- Product owners automating business processes

HOW TO USE IT

Sections 01–02 define the architecture and orchestration pattern. Use 03–05 when designing guardrails and recovery, and the checklist in 06 as a launch gate for each new workflow.

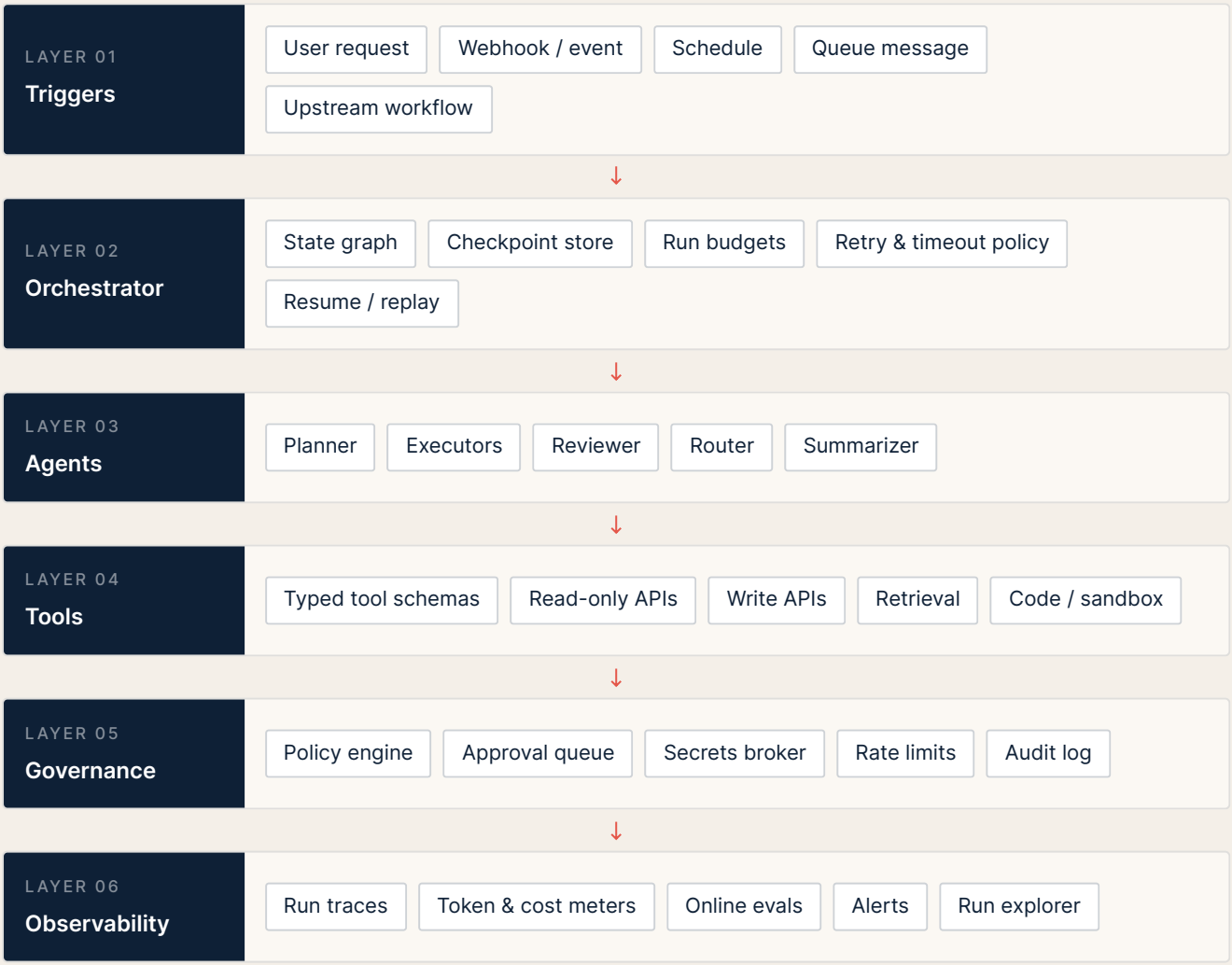
CONTENTS

01 The reference architecture
02 Planner, executor, reviewer
03 Policy checks and tool permissions
04 Approvals without blocking the system
05 Failure recovery
06 Tracing, evaluation and launch readiness

01 / ARCHITECTURE

The reference architecture

An agentic workflow is orchestration code with language models at the decision points. Six layers keep reasoning, action and control separate so each can be tested and changed on its own.



The orchestrator owns control flow and state; agents only propose. An agent can say “call `refund_order` with these arguments” — the orchestrator decides whether that transition is legal, the policy engine decides whether it is allowed, and only then does the tool execute.

Free-running agent vs governed workflow

FREE-RUNNING LOOP	STATE-MACHINE WORKFLOW
Model decides when to stop	Explicit terminal states and budgets
Any tool callable at any point	Tools scoped per node
State lives only in the context window	State checkpointed outside the model
A crash loses the run	Runs resume from the last checkpoint
Hard to explain why it acted	Every transition traced and auditable

✦ 02 / ORCHESTRATION

Planner, executor, reviewer

Splitting one large agent into three roles makes each prompt smaller, each toolset narrower and each failure easier to localize.

PLAN

Planner

Turns the goal into a short ordered plan of typed steps. Has read-only tools at most and never executes side effects.

ACT

Executor

Performs one plan step at a time with only the tools that step needs. Returns structured results, not prose.

CHECK

Reviewer

Validates outputs against the goal, rubric and policy. Can approve, request a retry, re-plan or escalate to a human.

01 Intake & validate goal → 02 Plan steps → 03 Execute next step → 04 Review result → 05 Approve or re-plan → 06 Finalize & report

Designing the state graph

ELEMENT	WHAT IT HOLDS	DESIGN RULE
State object	Goal, plan, step results, approvals, budgets used	Typed schema; versioned; small enough to checkpoint every node

ELEMENT	WHAT IT HOLDS	DESIGN RULE
Nodes	One agent role or deterministic function	Single responsibility; idempotent where possible
Edges	Allowed transitions between nodes	Conditional edges chosen from a fixed set, never free text
Interrupts	Points where the run pauses for input	Persist state, release compute, resume by run ID
Terminal states	Succeeded, failed, cancelled, escalated	Every path reaches one within the run budget

✦ DESIGN NOTE

Prefer deterministic nodes

If a step can be done with ordinary code — parsing, lookups, validation, formatting — make it a plain function node. Reserve model calls for decisions that genuinely need judgment. The graph gets cheaper, faster and easier to test.

✦ 03 / GOVERNANCE

Policy checks and tool permissions

The model is not a security boundary. Authorization has to live in code that the model cannot talk its way past.

Classify every tool by risk

TIER	EXAMPLES	DEFAULT POLICY
Read	Search, fetch record, list orders	Allow, scoped to the caller's permissions
Reversible write	Draft email, create ticket, add note	Allow with limits; log and make undoable
Irreversible write	Send email, issue refund, update ledger	Policy check; thresholds; often human approval
Privileged	Change permissions, delete data, deploy	Human approval always; separate credentials

The policy check on every action

01 **Validate the arguments**

Check the proposed call against its JSON schema and business rules — amount limits, allowed recipients, valid IDs.

02 **Authorize as the principal**

Evaluate the action as the human or service the run acts for, never as a shared super-user.

03 **Apply policy rules**

A policy engine (rules or policy-as-code) returns allow, deny or require-approval with a reason.

04 **Inject credentials late**

A secrets broker attaches short-lived, narrowly scoped credentials at execution time; the model never sees them.

05 **Record the decision**

Write the proposed call, policy result, approver and outcome to an append-only audit log.

SECURITY

Treat tool output as untrusted input

Web pages, emails and documents returned by tools can contain instructions aimed at the agent. Keep retrieved content in clearly delimited data fields, never let it change the tool allowlist, and require policy checks regardless of what the content says.

04 / HUMAN IN THE LOOP

Approvals without blocking the system

Human review is a workflow state, not a chat message. Design it so a run can wait hours or days without holding resources or losing context.

01 Policy requires approval → 02 Checkpoint & pause run → 03 Notify approver with context → 04 Approve, edit or reject → 05 Resume from checkpoint

01

Show the decision, not the transcript

Present the proposed action, its arguments, the evidence used and the policy reason. Reviewers should decide in under a minute.

02

Allow edits

Let the approver correct arguments before approving; record the diff as feedback for evals and prompt tuning.

03

Timeouts and escalation

Every approval has an expiry and a fallback owner. An unanswered request must not stall a run forever.

04

Graduated autonomy

Start with approval on every write; relax to sampled or threshold-based review only once evals and audit history justify it.

✦ 05 / RELIABILITY

Failure recovery

Agents fail in more ways than ordinary services: malformed tool calls, loops, hallucinated IDs, provider outages. Plan the recovery path for each.

FAILURE	DETECTION	RECOVERY
Malformed tool call	Schema validation fails	Return the validation error to the agent; retry up to a small fixed limit
Transient tool / API error	5xx, timeout, rate limit	Retry with exponential backoff and jitter; idempotency keys on writes
Model provider outage	Errors or latency above threshold	Fail over to a secondary model or pause and resume later
Loop or no progress	Same node or tool repeated; step budget hit	Stop, mark as escalated, route to a human with the trace
Reviewer rejects repeatedly	Retry count on a step exceeded	Re-plan once; then escalate rather than loop
Partial completion	Crash between side effects	Resume from checkpoint; compensating actions for completed writes

Run budgets to set on every workflow

- [ ] Maximum steps (node executions) per run
- [ ] Maximum tokens and maximum spend per run
- [ ] Wall-clock timeout per node and per run, excluding time paused for approval
- [ ] Maximum retries per tool and per plan step
- [ ] Idempotency key on every write tool so retries cannot double-apply
- [ ] Defined compensating action (undo) for each irreversible tool where one exists

✦ 06 / OBSERVABILITY

Tracing, evaluation and launch readiness

You cannot debug an agent from its final answer. Capture the full trajectory and evaluate it, not just the output.

TRACE

Run as a span tree

One root span per run; child spans per node, model call and tool call, with inputs, outputs, tokens and latency.

EVAL

Trajectory evals

Score whether the right tools were called in a sensible order, not only whether the final result was correct.

OPS

Run health

Track success, escalation and failure rates, steps per run, cost per run and approval wait time.

DESIGN

- [ ] State schema documented and versioned
- [ ] Every node has a defined input, output and owner
- [ ] All paths reach a terminal state within budget

GOVERNANCE

- [ ] Every tool assigned a risk tier
- [ ] Policy check enforced in code before every write

- [ ] Approval flow tested end-to-end including timeout

RELIABILITY

- [ ] Checkpoints after every node; resume tested after a forced crash
- [ ] Idempotency keys on all write tools
- [ ] Provider failover or pause path in place

OPERATIONS

- [ ] Full trace for every run, PII redacted
- [ ] Trajectory eval suite runs in CI
- [ ] Alerts on failure, escalation and spend anomalies

> **Let the model decide what to do next. Let the system decide what it is allowed to do.**
>
> SPLENDENSLABS ENGINEERING PRINCIPLE

GET IN TOUCH

Let's Talk

We design and run governed agentic workflows for operations, support and back-office teams. Bring us a process you want automated and the rules it has to respect.

- Workflow and state-graph design
- Policy, approval and audit layers
- Trajectory evals and production tracing

BOOK AN AI STRATEGY SESSION →

EMAIL
hello@splendenslabs.com

WEB
splendenslabs.com

OFFICE
Sona Incubation Foundation, Salem