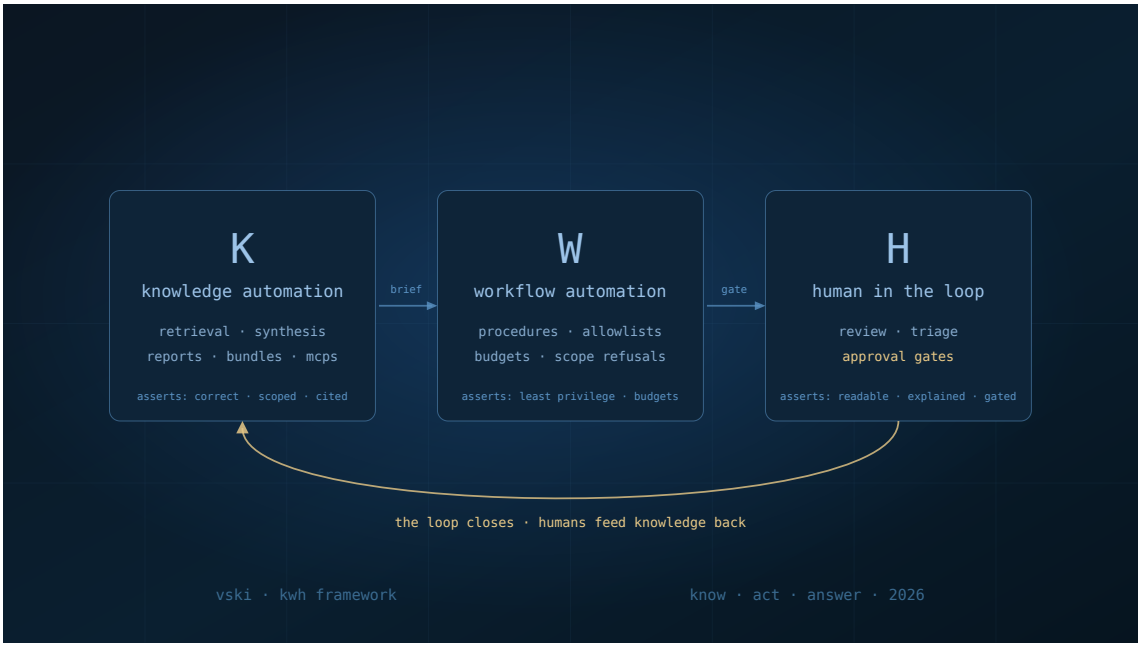
Anton Nesterov · an@vski.sh · October 31, 2026

# KWH Architecture: How to Develop Quality AI Agents

Agents · Workflows · Architecture

*Agents fail in boring ways: wrong facts, wrong actions, humans out of the loop. KWH splits every agent into three co-dependent lanes, Knowledge automation, Workflow automation, and Human In The Loop, and gives each lane its own build rules, its own evals, and its own failure signature. With a worked incident example in diagrams and a case for developing autonomous systems as infrastructure as code.*

---




Agents fail in boring ways. The 2026 incident roundups still open with the classics: a coding agent that deleted a production database for about two dollars of model cost, a shopping agent that placed an order nobody approved. Gartner expects **over 40% of agentic AI projects to be canceled by the end of 2027**, and when the postmortems dig in, the finding is consistent: the failure is almost never the model. It is the tangle. Retrieval, action, and interface get built as one blob, so nobody specified what good looks like, nobody can say which third broke, and nobody notices until the damage has a dollar sign on it.

KWH is how we untangle it. Every agent we build, from a script that watches a pipeline to a production fleet, is split into three co-dependent lanes: **Knowledge automation**, **Workflow automation**, and **Human In The Loop**. Each lane is developed, documented, and evaluated

on its own terms, and each has its own failure signature. The lanes are the whole framework: no library, no subscription, nothing to install. The rest is the operating procedure: create, evaluate, debug, deploy.
*Read tech article on KWH in our docs*

## The three lanes

A workflow, whether a single agent or a multi-agent system, has all three lanes present. That is the first rule, and it holds even for toys. A bot that only answers questions still has an interface and still needs a reviewer somewhere; a bot that only executes still reads state. So: split them, name them, and hold each to its own standard.

### Knowledge automation

The K lane is everything the agent knows: knowledge agents, MCPs, reports, and all knowledge work. The agent retrieves, analyzes, and synthesizes information, and produces a structured output: findings, reports, bundles. The output is a **contract**, not prose: named fields, evidence behind every claim, a confidence number, sources a reader can open.
Objectives in this lane assert on four things:

1. **Retrieval correctness.** The right rows and documents, not the nearest ones.
2. **Scope discipline.** The question that was asked, not the neighborhood of it.
3. **Evidence.** Every claim traceable to something a human can open.
4. **The output contract.** Same fields, same shape, every time. Drift is a defect.

Knowledge work looks harmless, which is why it rots first. A report with a wrong number is not a writing bug; it is a missing eval. The practice here is settled enough to copy: evaluate the whole **trajectory** (what was pulled, in what order), each **step**, and the **final output** against a golden dataset, and turn every production failure into a new test case. We do not ship a K lane without a probe set.

### Workflow automation

The W lane is where the agent stops describing and starts acting: actionable intelligence. This lane is the home of **execution procedures**: every actionable item the system can perform is defined here as a named procedure with preconditions, parameters, guards, and a journal entry. If the system can restart a service, that is a W procedure. If it can spend money, that is a W procedure with a gate in front of it. The procedures are the substance of the agent; everything else feeds them or audits them.
The lane carries its own security rules:

1. **Least-privilege tool allowlists.** The agent gets exactly the tools its procedures need, enforced at the server layer, never in the prompt. Every extra tool widens what an injection can reach.

2. **Call and turn budgets.** Hard ceilings on tool calls, turns, spend, and wall clock, per procedure. A loop that cannot stop is an outage with a progress bar.
3. **Scope refusals.** Anything outside declared scope is refused by rule, and the refusal is an event somebody sees.

And it carries its own fallback: an autonomous action that cannot proceed safely **degrades to a human decision** instead of failing silently or acting outside its scope. Degrading is not failure; it is the designed outcome for the cases the rules do not cover.
The 2026 security record explains the strictness. The MCPTox benchmark threw 1,312 malicious test cases at 45 live MCP servers and hit attack success rates up to 72.8%, and in not one case was the poisoned tool executed directly: a tool the operator had already approved carried the attack. Allowlists are necessary and not sufficient. Budgets, enforcement at the server layer, and audited journals are what make them hold.

### Human In The Loop

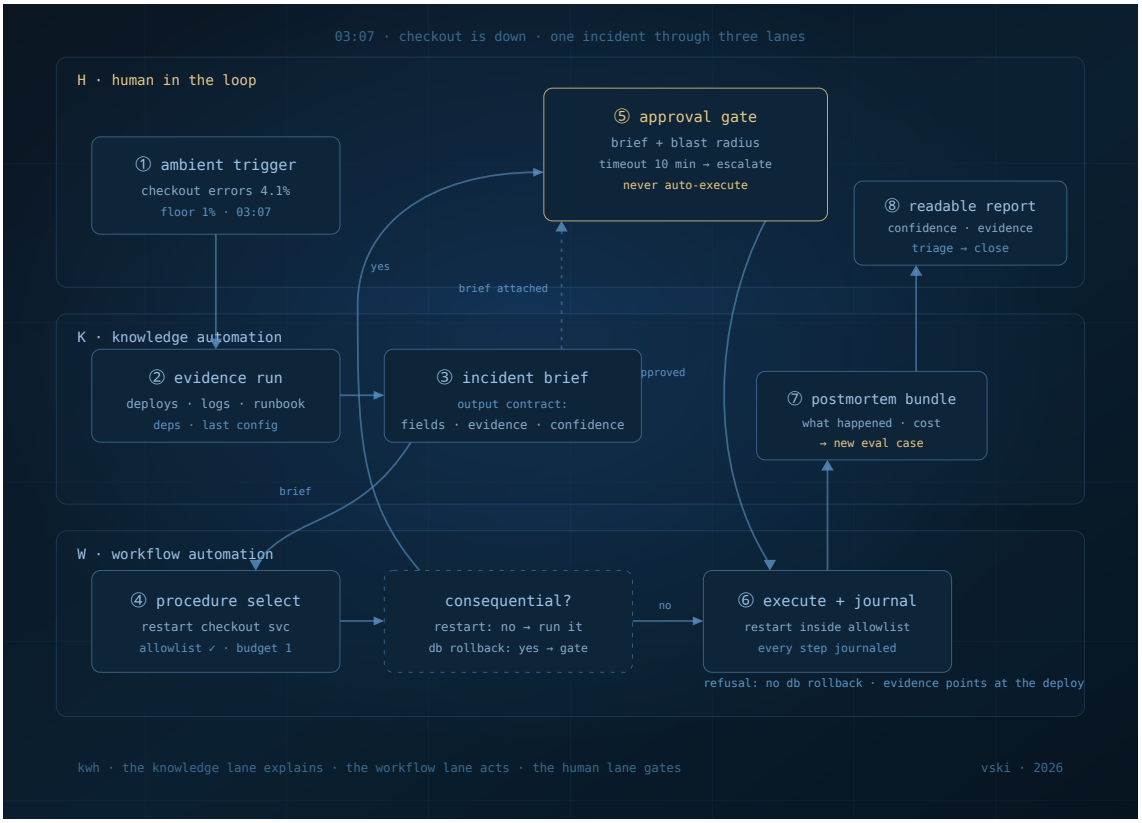
The H lane is the interface to the only component that can be held accountable: people. Human review, triage, and ambient triggers live here, and so do the properties that make the other two lanes safe to run:

- Outputs are shaped for human consumption: readable reports with confidence and evidence, not JSON dumps.
- Refusals explain themselves: “I will not roll back the database, because the evidence points at the deploy, not the schema.”
- Approval gates stand in front of consequential actions, and what the approver sees is the K lane’s brief plus the W lane’s blast radius, not a button with a label.

The pattern set is named and proven: gates at the level of the individual tool call, with pause and resume; escalation when a timeout expires, because “nobody answered” is not an approval; reviewer views that show what the action touches. The oversight layer is also where the analysts put the knife: Gartner’s cancellation forecast lists escalating costs, unclear business value, and inadequate risk controls, not model quality, and the EU AI Act writes human oversight into law for high-risk systems. Budget for the human, or the project joins the statistic.

## The example: checkout is down

The setting is deliberately small: one store, one on-call engineer, one agent built on a local ~8B model. The incident runs through all three lanes, and every step leaves an artifact the next step reads.




The run, step by step:

1. **The ambient trigger (H).** Checkout errors cross 4.1% against a 1% floor at 03 . The trigger belongs to the human lane because its job is to decide when a human might be needed, not to explain anything.
2. **The evidence run (K).** The agent pulls the last deploys, the error logs, the runbook, and dependency status. No actions yet.
3. **The incident brief (K).** The output contract: what changed, what broke, candidate causes, each with evidence and confidence. Prose is allowed inside fields; the fields are not negotiable.
4. **Procedure select (W).** Given the brief, the agent picks from its allowlist: restart the checkout service, budget one restart. Not consequential, so it runs. A database rollback would have been consequential, and consequential does not run on its own.
5. **The approval gate (H), had it been needed.** For the rollback branch: the human sees the brief, the proposed action, and the blast radius. Ten minutes of silence escalates to the on-call lead. “Nobody answered” is not an approval.
6. **Execute and journal (W).** The restart runs inside the allowlist, budget spent one of one, every step journaled.
7. **The postmortem bundle (K).** What happened, what it cost, and one new eval case for the probe set, because turning production failures into tests is how the K lane compounds.
8. **The readable report (H).** Confidence, evidence, triage notes. Close.

One incident, three lanes, one loop. The trigger, the gate, and the report are H. The evidence run, the brief, and the postmortem are K. The procedure, the allowlist, the budget, and the journal are W. No step skipped a lane. And the refusal in the W lane (“no db rollback, the evidence points at the deploy”) is doing H lane work in advance: it explains itself before any human had to ask.

## The guideline

KWH works on a greenfield build and as a forensic pass over an existing agent: write down the three lanes, and whatever you cannot write down is what is missing.

**Create.** Contracts before code. Write the K output contract first: fields, evidence rules, confidence. Write the W procedure list before wiring a single tool: name, preconditions, budget, fallback, gate or no gate. Sketch the H surface before the first demo: who sees what, which actions gate, what the refusal says. If those three documents do not exist, the agent is not specified; it is a vibe with an API key.

**Evaluate.** Never one score; three scorecards, run in CI on every change to any lane. K: probe sets for retrieval correctness, citation checks, contract conformance. W: dry runs in a twin environment, tool-call trajectories against golden sequences, budget conformance, and an injection red team (prebuilt suites exist for exactly this). H: the human test: can a teammate who has never seen the agent triage an output in two minutes; approval latency; time-to-degrade. Every lane’s production failures become next week’s eval cases.

**Debug.** The lanes give you the fault domain, which is most of debugging. Wrong or missing fact: K. Wrong action, blown budget, silent failure: W. Human surprised, rubber-stamped gate, ignored report: H. Bugs that seem to span two lanes are interface bugs: the contract drifted and W reads fields K no longer writes, or W journaled evidence the gate never displayed. Multi-agent systems amplify their own errors: Berkeley’s MAST study read 1,600+ execution traces across seven multi-agent frameworks and measured amplification up to 17.2x for independent agents versus 4.4x with centralized coordination. Pin the fault to a lane before you add another agent to fix it.

**Deploy.** As declaration, not click-ops.

## Agents are infrastructure. Declare them.

The recommendation is blunt: develop autonomous systems the way serious teams develop infrastructure, as code, in a repository, through review. Agent configuration is infrastructure configuration with the stakes raised: a misconfigured cloud resource exposes a port, and a misconfigured agent policy hands an autonomous system the run of the environment. The industry has already named the practice (agent infrastructure as code) and shipped the pieces: sandboxed runtimes governed by declarative policy (NVIDIA’s OpenShell, launched at GTC 2026, wraps coding agents in kernel-level isolation governed by plain YAML: which filesystems, which user, which endpoints, whether enforcement is on), static scanning

of policy files, pinned and signed tool references, secrets detection, drift alarms, and gates in CI before anything merges.
Declared, in version control, is:

- the model version and the prompts;
- the K contracts and their probe sets;
- the W procedures, allowlists, and budgets;
- the H gates, views, and escalation rules;
- the dashboards and the infrastructure underneath.

What that buys is the boring list, and the boring list is the point. Review: a diff, not a meeting. Reproducibility: the eval ran against exactly this declaration, so what you tested is what you deployed. Audit: the manifest answers “what exactly is in my agent?” Rollback: the previous declaration. Agents change weekly; the declaration is the only stable artifact.

## One shape, any stack

Framework churn is a 2026 condition. The comparisons review six frameworks worth serious consideration; the community threads compare twenty. KWH does not compete with any of them. LangGraph, CrewAI, the OpenAI Agents SDK, plain MCP servers on a laptop: all of them carry the same three lanes, and the lanes always produce the same three artifact sets: contracts and corpora for K, procedures and policies for W, gates and views for H. That is the standardization, and it is why we run one review checklist across everything: every agent in the company has the same shape, so onboarding, review, procurement, and audit all ask the same three questions.
The shape is also what makes small local models trustworthy in production. An ~8B model with a K contract is bounded in what it can claim; with a W allowlist and budget, bounded in what it can touch; with an H gate, bounded in what it can break. Trust comes from the fences, not the parameter count. Build the lanes, declare them, gate what matters, and quality stops being a hope and becomes a checklist.

## Sources


- ARMO, “MCP Prompt Injection: The Attack Uses the Tools You Approved”, August 2026 (MCPTox: 1,312 malicious test cases across 45 live MCP servers; success rates up to 72.8%; the poisoned tool was never the one that executed): https://www.armosec.io/blog/mcp-prompt-injection
- Panther, “How to Secure an MCP Server” (the 72.8% attack success rate and rug-pull patterns, from the same benchmark): https://panther.com/blog/how-to-secure-an-mcp-server
- OWASP, “MCP Tool Poisoning” (the attack class the W lane fences): https://community.owasp.org/attacks/MCP_Tool_Poisoning

- AIRIA, “How to Apply Least Privilege to MCP Tools and AI Agent Permissions” (every extra tool widens what an injection can reach): https://airia.com/blog/how-to-apply-least-privilege-to-mcp-tools-and-ai-agent-permissions
- Cycles, “The State of AI Agent Incidents (2026)”, April 2026 (MAST failure modes over 1,600+ traces and seven frameworks; 17.2x vs 4.4x error amplification; the layered controls: budget gates, scope isolation, audit trails): https://runcycles.io/blog/state-of-ai-agent-incidents-2026
- Codiant, “Why AI Agent Projects Fail in 2026: 7 Mistakes”, October 2026 (Gartner: over 40% of agentic AI projects canceled by end of 2027; causes in workflow design, data, and governance, not model quality): https://codiantech.com/blog/why-ai-agent-projects-fail-2026
- DeepEval, “AI Agent Evaluation” (trajectory, step, and final-output evaluation against golden datasets): https://deepeval.com/guides/guides-ai-agent-evaluation
- Scalekit, “Human-in-the-Loop Tool Calling: Approval Gates for AI”, May 2026 (invocation-level gates, pause and resume, scope escalation): https://www.scalekit.com/blog/human-in-the-loop-tool-calling
- Moring, “Human in the Loop AI Agents: Designing Approval Gates”, October 2026 (which actions to gate, what reviewers see, timeouts, rejection reasons): https://www.moring.ai/blogs/human-in-the-loop-why-approval-gates-are-the-feature
- Cycode, “The Rise of Agent Infrastructure as Code”, March 2026 (declarative agent policies in version control; OpenShell at GTC 2026; scanning, pinning, drift checks, CI gates): https://cycode.com/blog/agent-infrastructure-as-code
- Splunk, “Top Agent Frameworks 2026”, September 2026 (framework churn: six frameworks worth serious consideration): https://www.splunk.com/en_us/blog/learn/agent-frameworks.html