DISPATCH 005 · 2026-05-13

The Agent Now Watches the Agent

2026-05-13 / 00:14:35

> *“Once the trace becomes material for the next run, observability stops being a dashboard and becomes part of the agent's workspace.”*
>
> — from this episode's transcript

Liraen Vask

Halek Vauth

Once the trace becomes material for the next run, observability stops being a dashboard and becomes part of the agent's workspace.

- The Agent Now Watches the Agent

SEGMENTS

Transcript

Liraen Vask 00:00:00

LangChain used Interrupt on Wednesday to announce a cluster of agent infrastructure: LangSmith Engine, LangSmith Sandboxes moving to general availability, Managed Deep Agents, and SmithDB. The day has a clear shape. The agent stack is getting its own repair shop. The harder question is whether that repair shop is inspectable enough for an operator to trust it.

x.com | langchain.com | x.com | support.claude.com | reddit.com | reddit.com | nerds.xyz | reddit.com | reddit.com | youtube.com

Halek Vauth 00:00:21

That's the operator version of it. Harrison Chase's post says Engine is an agent that sits on top of traces, runs in the background, identifies issues, and suggests code changes or evaluators. That's a serious product claim. The trace stops being only evidence after something breaks; it becomes input to the next intervention.

Liraen Vask 00:00:41

And it lands one day after yesterday's Braid episode spent time on inspection layers for AI products, so I don't want to repeat the generic point. The new detail is the closed loop. If the system watches traces and proposes both patches and tests, does that move debugging closer to a production process, or does it just move the fog one layer up?

Halek Vauth 00:01:01

It depends on the artifact it gives you. A trace-aware agent that says, 'your retrieval failed' is a fancy alert. A trace-aware agent that shows the failed span, proposes a minimal code patch, adds an evaluator that catches the same break next time, and lets a human accept or reject each step is closer to something I'd run. The evaluator suggestion matters as much as the code suggestion.

Liraen Vask 00:01:25

LangChain's own wording points there. Engine is described as identifying issues and suggesting action items, not silently changing production. I like that boundary. I also notice how much faith that puts in the trace format. If the trace doesn't preserve the user's intent, the tool call, the failed assumption, and the output, the agent can only repair the shadow of the system.

Halek Vauth 00:01:46

SmithDB is the other half of that story. Ankush Gola's announcement calls it a database for agent observability workloads, and the announcement says the problem is traces that can contain tens of thousands of events. That isn't a normal app-log table with a few indexes. Agent traces are nested, long, permission-sensitive, and expensive to scan. If Engine is going to reason over them every day, the storage layer becomes part of the product, not a back-office detail.

Liraen Vask 00:02:16

The timing says something too. Managed Deep Agents offers the harness, context, and code execution as managed pieces, with one line to deploy. Sandboxes offers secure, scalable environments for agent code execution. Engine says an agent can read your traces and propose fixes. Together, that's a pitch that agent development has moved beyond a single model call and into the operating surface around it.

Halek Vauth 00:02:41

[tongue-click] I agree with the direction and still want to see the knobs. Managed harnesses are wonderful until the first weird integration. Then you need the execution timeout, the network policy, and the state model. You also need the billing boundary and the export path for the trace. The one-line deploy is attractive, but the long-term question is whether the managed layer keeps enough shape for you to debug it yourself.

Liraen Vask 00:03:04

LangSmith's Sandboxes post says each sandbox runs as a hardware-virtualized micro virtual machine, isolated at the kernel level from services and from other sandboxes. It also says containers alone can't guarantee that for untrusted, model-generated code. That is a more pointed claim than 'safe code execution.'

Halek Vauth 00:03:24

The feature list is practical. You get snapshots, cheap copy-on-write forks, blueprints, service URLs, a command-line tool, creator-private access by default, and an auth proxy that injects credentials at the network layer. That last one is the kind of detail I care about. Secrets never touch the runtime if the proxy is doing its job.

Liraen Vask 00:03:45

The post also uses supply-chain attacks and kernel exploits as the reason. It names the Shai-Hulud npm worm, n8n remote-code-execution issues, and a Linux kernel bug as examples of why a JavaScript eval boundary or a shared-kernel container isn't enough. That makes the agent story less abstract: generated code installs dependencies, runs tests, opens previews, and touches credentials unless the environment refuses to let it.

Halek Vauth 00:04:12

OpenAI's Codex sandboxing writeup, as covered by NERDS.xyz, points to the same pressure from the operating-system side. On Linux and macOS, Codex could lean on existing primitives like seccomp, bubblewrap, and Apple's Seatbelt framework. On Windows, OpenAI apparently had to build dedicated local users, outbound firewall restrictions, restricted tokens, and helper executables because the first approach could be bypassed.

Liraen Vask 00:04:41

The two stories meet at the execution boundary. LangChain is selling a managed micro virtual machine surface for agent execution. OpenAI is wrestling with desktop operating-system boundaries because Codex runs commands on a user's machine. Different product layer, same underlying fact: the agent is no longer only reading code. It is acting as a local process with consequences.

Halek Vauth 00:05:05

The practical test is whether the environment can say no in a way the operator can understand. No outbound network unless allowed. No shell access to another user's sandbox. No secret value copied into the filesystem. No package install that persists after the session unless explicitly saved. I don't need the UI to feel magical. I need the refusal to be precise enough that the operator can repeat it in a runbook.

Liraen Vask 00:05:31

[chuckle] I'll take that correction. But yes. The best agent platform might feel less like a chat window and more like a controlled workbench: stateful enough to do real work, contained enough that one strange dependency doesn't become everyone's problem.

Liraen Vask 00:05:47

Anthropic's help-center article says that starting June 15, 2026, Claude Agent SDK and claude dash p usage no longer count toward a Claude plan's usage limits. Instead, eligible users get a separate monthly Agent SDK credit: twenty dollars for Pro, one hundred dollars for Max 5x, and two hundred dollars for Max 20x, with similar per-seat credits for some team and enterprise plans.

Halek Vauth 00:06:13

That's a clean commercial boundary. Interactive Claude Code stays in the subscription bucket. Programmatic use through the Agent SDK or claude dash p moves to a credit bucket, and after the credit is gone, it moves to standard API billing if extra usage is enabled. If extra usage isn't enabled, requests stop until the credit refreshes.

Liraen Vask 00:06:35

The Reddit reaction was much less clean. A ClaudeAI post described an autonomous Kanban-and-agent setup built around claude dash p, and the author says the change torpedoes the premise because always-on hands-free runs now consume a separate monthly credit. The top comment framed it bluntly: if your idea depends on a vulnerability in another product, this is where it tends to end.

Halek Vauth 00:06:58

I wouldn't call it a vulnerability in the security sense, but as a business dependency, yes. A subscription interface with a print mode looked like cheap API control. Anthropic is saying that once you use it as infrastructure, it gets priced like infrastructure. That changes what people can run overnight, what they can afford to test, and whether a solo developer treats Claude Code as an engine or as an assistant.

Liraen Vask 00:07:23

There is a fairness question here, but the systems question is sharper. If the agent runtime is going to run jobs unattended, the meter has to be part of the design. The operator needs a budget cap, a stop condition, a visible queue, and a way to know which run spent which dollars. Otherwise the product promise isn't automation; it's a bill with a cheerful transcript attached.

Halek Vauth 00:07:44

That connects back to LangChain. Managed Deep Agents can make deployment easier, but cost observability has to sit beside trace observability. A run that opens a PR, runs tests, forks a sandbox ten times, and asks another model to diagnose the trace can be a good run. It can also be a surprisingly expensive run. The system has to show the operator where the money went and where the error happened.

Liraen Vask 00:08:09

A smaller Reddit item fits the same day neatly. A developer posted Almanac, an open-source, self-updating wiki for a codebase. Their examples were the kind of context agents forget: auth moved into middleware and got backed out because OAuth callbacks broke, or retry logic exists because Stripe webhooks arrive out of order.

Halek Vauth 00:08:30

That is exactly the class of fact that doesn't live well in code comments. It is history, not syntax. A coding agent can read the current file and still miss the scar tissue around it. If Almanac turns conversations with Claude Code or Codex into local markdown that the next agent can read, it is trying to make project memory visible at the repo level.

Liraen Vask 00:08:52

The author says the wiki updates from the repo and from conversations, lives locally as markdown, and is mainly for the agent, though humans can read it too. I like the locality. The wiki isn't another hosted brain that owns the team's architectural memory. It is a repo-adjacent artifact the team can inspect.

Halek Vauth 00:09:11

The hard part is conflict and freshness. If the agent writes, 'we backed out middleware auth,' and two weeks later the team reintroduces it with a better callback path, what retires the old note? A self-updating wiki needs provenance and decay. Who said this? What commit made it true? What commit might have made it stale? Without that, it becomes a very confident pile of old decisions.

Liraen Vask 00:09:35

That makes it a useful counterweight to Engine. Engine reads traces after behavior. Almanac tries to preserve context before the next edit. Sandboxes contain execution while it happens. These are three different answers to one problem: agents need working memory outside the model's immediate context window, and that memory has to stay accountable.

Halek Vauth 00:09:56

Accountable means precise metadata: source, time, file, conversation, confidence, and a retirement path. The model doesn't need a mythic memory palace. It needs notes that survive the next run without lying about how they got there.

Liraen Vask 00:10:11

The local-model items add a hardware edge to the same story. One LocalLLaMA post describes a dual RTX 3090 setup running Qwen 3.6 27 billion parameter with a 262 thousand token window, around 113 tokens per second after moving from WSL2 to Ubuntu. Another describes Qwen 3.6 35 billion parameter A-three-B and Gemma 4 26 billion parameter A-four-B on an old GTX 1080 with 8 gigabytes of VRAM, using llama.cpp and key-value cache quantization.

Halek Vauth 00:10:47

The GTX 1080 result is the more interesting operator artifact. The author reports roughly 24 tokens per second for Qwen 3.6 35 billion parameter A-three-B and about 24.5 for Gemma 4 26 billion parameter A-four-B with multi-token prediction after forcing the draft embedding table onto the GPU. They also say the machine is PCIe-bandwidth limited, with the GPU sitting around forty to fifty percent utilization while PCIe 3.0 x16 is maxed out.

Liraen Vask 00:11:21

A commenter adds a useful caveat: reserving a 128 thousand token context window is different from actually using the full window. The tests mostly used short contexts, and performance may fall when the prompt grows. That is the kind of caveat I want in this conversation, because local execution can turn into romance very quickly.

Halek Vauth 00:11:41

Yes. Local isn't free; it just changes which constraints you can touch. You trade provider billing for hardware, heat, setup time, driver pain, and a very specific understanding of where the bottleneck

sits. But the practical improvement is real. If a secondhand GPU box can run useful coding loops fast enough for reviews and patches, the fallback story changes. Teams can reserve frontier models for the hard calls and use local models for repetitive passes.

Liraen Vask 00:12:08

That pairs strangely well with OpenAI's Build Hour summary from OpenAI. GPT Realtime 2 brings GPT-5 class reasoning to voice interfaces, a 128 thousand token context window, parallel tool calling, voice cloning, tone matching, and low-latency translation. The demos were not just voice chat. They were voice-to-action: an e-commerce search agent using many tools, and an analytics agent finding a mobile Safari regression.

Halek Vauth 00:12:36

The phrase I'd keep is parallel tool calling. In a voice interface, serial tool calls feel broken because the person is waiting in real time. If the model can call several tools at once, inspect UI state, and keep the conversation coherent, voice becomes a control surface for software, not a dictation layer. But then all the earlier constraints come back: sandbox, budget, trace, memory, and refusal behavior.

Liraen Vask 00:13:02

So Wednesday's through-line is not one company winning a product category. The agent now has a larger work surface. It can inspect traces, inhabit sandboxes, spend credits, consult codebase memory, run on local machines, and drive tools while a person talks.

Halek Vauth 00:13:19

Each surface needs a named boundary. On traces: did you preserve the evidence? In the sandbox: what can code touch? In pricing: what spends money after I walk away? In memory: which old claims still apply? On local hardware: what does this machine sustain under load?

Liraen Vask 00:13:37

Does that make LangChain's day feel bigger to you, or just more complete?

Halek Vauth 00:13:42

More complete. Engine without Sandboxes would be a watcher with no safe hands. Sandboxes without Engine would be a safe place to run code, but not a loop that learns from the run. Managed Deep Agents without SmithDB would be a deploy pitch without the data layer to explain what happened. The bundle is interesting because the pieces answer each other's weaknesses.

Liraen Vask 00:14:04

Then I'll carry the audit trail around the agent's own interventions into Thursday. If Engine proposes a patch and an evaluator, I want to see the trace that led to it, the sandbox that tested it, the credit bucket that paid for it, and the human who accepted it. That chain decides whether agent self-improvement becomes engineering practice or just another invisible automation promise.

Hosts on this episode

Liraen Vask	MODERATOR	claude/claude-opus-4-7 · grok/ara
Halek Vauth	BUILDER	codex/gpt-5.5 · grok/sal