Pulumi

WHITE PAPER SUMMER 2026

The agentic infrastructure operating model

Why agents mastered code before infrastructure, and how to close the gap.

Agentic infrastructure is an operating model in which AI agents author, deploy, and operate the majority of infrastructure changes, subject to the policies, approval requirements, and audit controls established by the organization.

Product engineers need to ship at AI speed. Infrastructure teams have to keep up, and they're still on the hook for security, compliance, uptime, and cost. The velocity gap between the two is widening, and closing it has become a top priority for infrastructure leaders everywhere. This paper explains why the gap exists, and the operating model that teams from two-person startups to the top-three frontier labs, and everyone in between, use to close it: infrastructure that is self-service for engineers and agents, and self-governing for the organization.

IN THIS DOCUMENT

FOREWORD

A note from our co-founder and CEO

Not long ago, an executive at one of our customers pulled me aside at the end of a meeting. “You and I both know where this is headed,” he said, meaning one hundred percent agentic infrastructure, “even if my team doesn’t know it or believe it yet.” He was describing the present more than the future: over 30% of deployments on the Pulumi platform that humans used to run are now done by agents, most of them the same coding agents your engineers use every day, up from essentially zero at the beginning of 2026.

I have a version of that conversation almost every week now. A CIO who says the same thing from conference stages. An executive at a global automaker who watched an agent debug and fix live infrastructure, then turned to his team and asked why a proof of concept wasn’t already underway. Teams at the AI frontier lab who concluded that AI-scale coding requires AI-scale infrastructure to match. Nobody asks me whether agents will operate infrastructure anymore. They watched what happened to coding and now ask why infrastructure is lagging, and how to close the gap without losing control of what their infrastructure becomes.

This paper is the answer we give them. Operating infrastructure lagged coding for one structural reason, and that reason is fixable. Fix it, and agentic infrastructure stops being a science project and becomes an operating model… a substrate that agents are fluent in, a control plane that governs what they do with it, and a deliberate path from assisted to autonomous operations. The teams that do this will have a massive time-to-market advantage.

We have always seen the cloud a little differently: less a pile of discrete resources, more an evolving operating system to be modeled and built upon in software. That conviction is why Pulumi exists, and why the next operating model is fundamentally software-driven infrastructure.

— Joe Duffy
Founder & CEO, Pulumi

01

The velocity gap

A CIO we work with put the whole situation in four sentences:

> **"My developers are writing more code than ever before, shipping it faster than ever before, and understanding it less than ever before. And that code still runs in the cloud. It still has to be secure, compliant, dependable, and cost-effective. Those requirements don't go away because AI is writing the code."**
>
> CIO, LARGE HEALTHCARE COMPANY

Infrastructure teams are taking on more load than ever. Product engineers, accelerated by AI, need on-demand infrastructure at the pace at which they now ship. And the team is still on the hook for everything it has always owned including security, compliance, uptime, and cost.

Meanwhile, the cloud sits at the center of every product, and AI pushes it further: GPU fleets, inference at scale, data pipelines, per-customer tenancy. That changes what infrastructure is to the business, changing it from a cost center to an innovation budget, one whose returns manifest as velocity across the entire company.

Leaders are pushing their teams hard to use AI to accelerate, and for the most part it's working, for applications and coding. The most aggressive adopters report that agents now write roughly 90% of their code. But the benefit has been slow to show up on the infrastructure side of the house. Agents are hitting the infrastructure wall and struggling to provision, manage, and scale the systems on which their code runs.

Andrej Karpathy captured what that feels like after vibe-coding an app of his own. Building a modern app, he wrote, is "a bit like assembling IKEA furniture." He spent most of his time not in the code editor but in the browser, "moving between tabs and settings and configuring and gluing a monster" of services, API keys, and deployments, none of it accessible to an LLM. AI writes the code; humans still assemble the furniture. The numbers make the point:

The result is a widening velocity gap where applications are shipping at AI speed, but infrastructure is still moving at human speed. That gap is untenable, and, left alone, the infrastructure team becomes the bottleneck for everything the company wants to do. The teams operating at the largest scale have already reached the same conclusion. One of the world's leading AI research organizations determined that the only way to keep up with AI-powered coding is AI-powered infrastructure, and the key to AI-powered infrastructure is that the infrastructure itself needs to be expressed in code the models can understand.

$700B
Hyperscaler capex in 2026, 75% of it AI-related

90%+
Share of code written by agents at the most aggressive adopters

6%
Organizations with complete infrastructure-as-code coverage

02

Why infrastructure lagged, and why it's fixable

Agents got so good at coding that it was natural to assume infrastructure would follow on the same timeline. It didn't. The reason comes down to what the models learned from.

Coding became the first great agentic success because the training signal was overwhelming with billions of lines of real, production-grade code on the public internet, paired with compilers, type checkers, and tests that make every answer verifiable. That tight feedback loop produced a steep rise in the capability of AI agents to write code. Public coding benchmarks went from roughly a third of tasks solved in mid-2024 to the high eighties less than two years later, and the frontier keeps climbing.

Today's models write passable bespoke infrastructure scripts such as YAML, HCL, etc. for the simple cases. The public DSL corpus is real, just far smaller than general-purpose code, skewed toward tutorials and boilerplate, with most production-grade examples locked away in private repos. So agents are competent right up until they aren't. A January 2026 research paper on infrastructure synthesis found that even state-of-the-art models "struggle… often hallucinating resource types and attribute names," and the failures concentrate exactly where production infrastructure lives.

Most infrastructure sits on the wrong side of that benchmark line:

WHERE INFRASTRUCTURE LIVES	TRAINING SIGNAL	WHAT THAT MEANS FOR AGENTS
Consoles, ClickOps, runbooks, tribal knowledge	Essentially none: the actions leave no artifact to learn from	Invisible to agents entirely
Infrastructure DSLs (YAML, HCL, CloudFormation, ARM)	Real, but orders of magnitude smaller, skewed toward tutorials and common patterns	Competent on the common cases; hallucinates the long tail
General-purpose languages (TypeScript, Python, Go, C#, Java…)	Vast: billions of lines of production code, plus the compilers, types, and tests to verify against	Fluent, and improving with every model release

The training volume gap compounds with two other elements. 1) Verification: general-purpose languages come with compilers, type systems, and unit tests, the loop agents use to catch their own mistakes, while a DSL gives an agent far less to check against before a change reaches your cloud. 2) Transfer: every new model gets better at Python and TypeScript because that's where the training data keeps growing, and those gains carry over to your infrastructure only if it's written in those languages. A DSL doesn't ride the curve.

02 - CONTINUED

You can fight this with better prompts, retrieval, and an MCP server in front of your tooling. It helps at the margin, and every prior-generation vendor now ships one. But nothing beats distribution. There's direct evidence in peer-reviewed benchmarks, where agents that take actions by writing code outperform agents that emit structured config by up to 20%. Code is what agents know best, and it's also how they act best.

Do that, and infrastructure collapses into a coding problem, one agents already solve, with a verification loop of types, tests, and previews they already know how to use. Every gain the frontier labs make in general coding ability becomes a gain in your infrastructure automation, at no cost to you.

Don't teach the model bespoke infrastructure practices. Express infrastructure in languages the model is already fluent in.

It helps to look at the alternatives, which fall into three categories and solves just one part of the puzzle:

APPROACH	EXAMPLES	WHERE AGENTS STAND	OPERATIONALLY COMPLETE?
Static DSLs	Terraform/OpenTofu HCL, CloudFormation, ARM	Competent on common patterns; hallucinates the long tail; doesn't ride the coding curve	Partial (needs bolt-on tooling)
Transpile-to-DSL	AWS CDK, Azure Bicep	Fluent at the code layer, but debugging and operations drop to the single-cloud DSL underneath	No engine of its own; single cloud
Home-grown scripts	Bash, cloud CLIs and SDKs	Very fluent, and that's the danger: every action is an immediate side effect	No plan, no state, no record
Real code + platform	**Pulumi (TypeScript, Python, Go, C#, Java)**	**Fully fluent, with types, tests, and previews to verify against**	**Yes (state, policy, identity, audit, secrets)**

Special-purpose languages are great for well-bounded problems, but modern cloud infrastructure is unbounded. Asking an agent to work in a thinly learned configuration language, often through a wire protocol, is a much weaker position than letting it write infrastructure in the same language it uses to write applications. In this case, what was designed to be ergonomic for humans turns out to also be ergonomic for agents too.

And picking the new side of the generational line doesn't mean throwing away the old estate. Pulumi runs HCL, existing Terraform programs, and YAML as-is, on the same engine and under the same control plane as real languages.

03

Two pillars, one platform

The operating model rests on two pillars. The whole argument of this paper is that you need both, as one system.

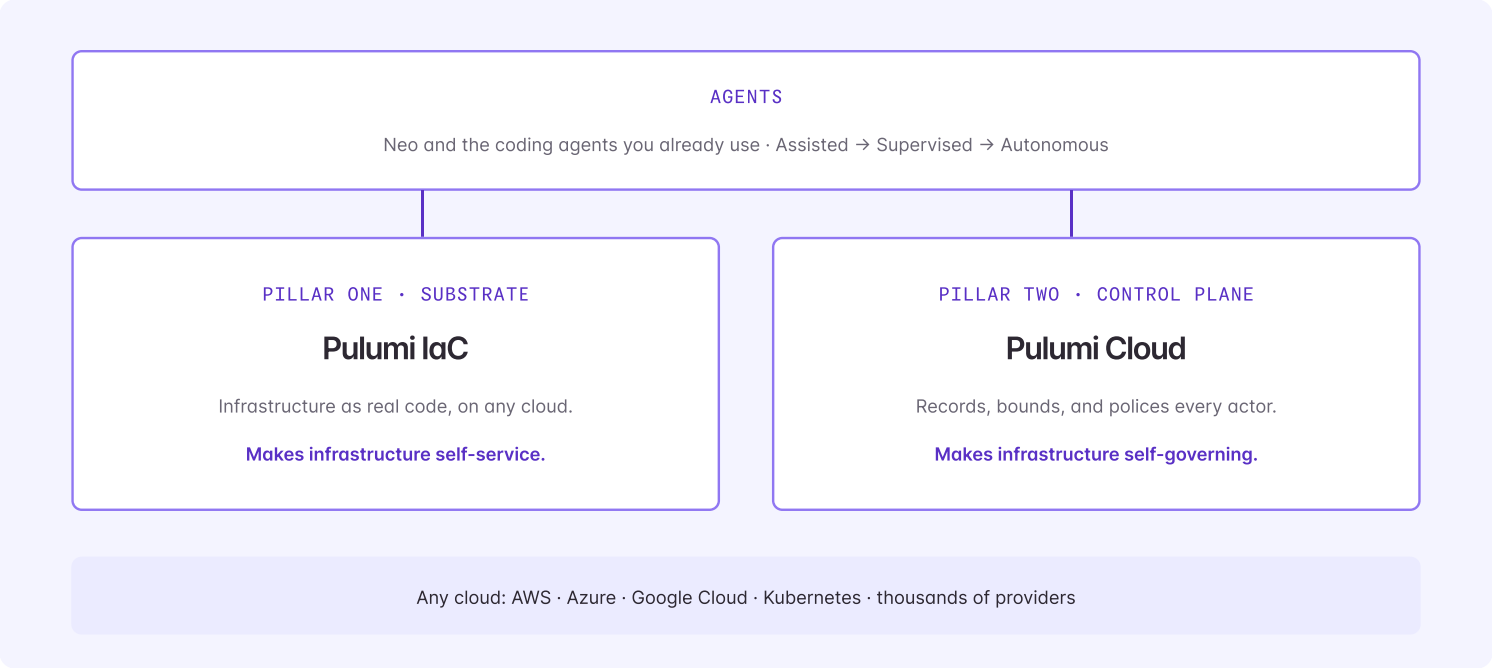

Figure 1. The operating model: agents ride on two pillars, the substrate and the control plane, spanning every cloud.

The first pillar is the substrate: Pulumi IaC. Infrastructure expressed as real code is what makes the agentic future possible in the first place. It puts infrastructure in-distribution, where agents are fluent and where every improvement in the underlying models makes your automation better.

We spent fifty years building software engineering as a practice, and learned how to build bigger things out of smaller things leveraging components, packages, versioning, type systems, tests, and code review. Infrastructure mostly missed out. Config files don't compose, so teams share by copy-paste, and every pasted copy is one more place for drift to creep in and a security issue to hide, a long tail someone eventually has to chase down.

The first pillar places infrastructure atop all that accumulated engineering. Define infrastructure in a real language and the whole toolbox applies. Tried and true engineering for infrastructure. And, not coincidentally, the discipline agents are best at.

Pulumi IaC is that substrate: infrastructure defined in TypeScript, Python, Go, C#, or Java, on an open-source engine that also runs YAML and HCL, across every major cloud and thousands of providers, including the AI-native stack, where GPU training fleets and unusual cross-cloud architectures don't fit anyone's templates.

Code also scales with complexity. Static VM fleets can survive on config files but still benefit from repeatable deployment pipelines. Modern multi-cloud, multi-service apps and distributed systems need 1,000s of resources, and real languages and engineering tame the complexity. AI-native workloads, with enormous scale, dynamism, and exotic cross-cloud GPU architectures benefit greatly. The harder the workload, the more code helps.

The benefits compound as teams grow. A platform team defines a golden pattern once as a typed, versioned, tested package, and every engineer and every agent builds on it everywhere. Pulumi IDP provides this self-service for developers and scientists on golden paths, so the vetted building blocks are the easy path for everyone. And because the entire engine is embeddable, infrastructure becomes a capability of your own software. It's how Wiz programmatically drives more than a million cloud resources with hundreds of thousands of updates a day.

03 - CONTINUED

Two pillars, one platform

The second pillar is the control plane: Pulumi Cloud. It governs what the substrate enables by recording every change, bounding every actor, and enforcing policy at the speed agents operate.

Code is only half the story. The other half is operational.

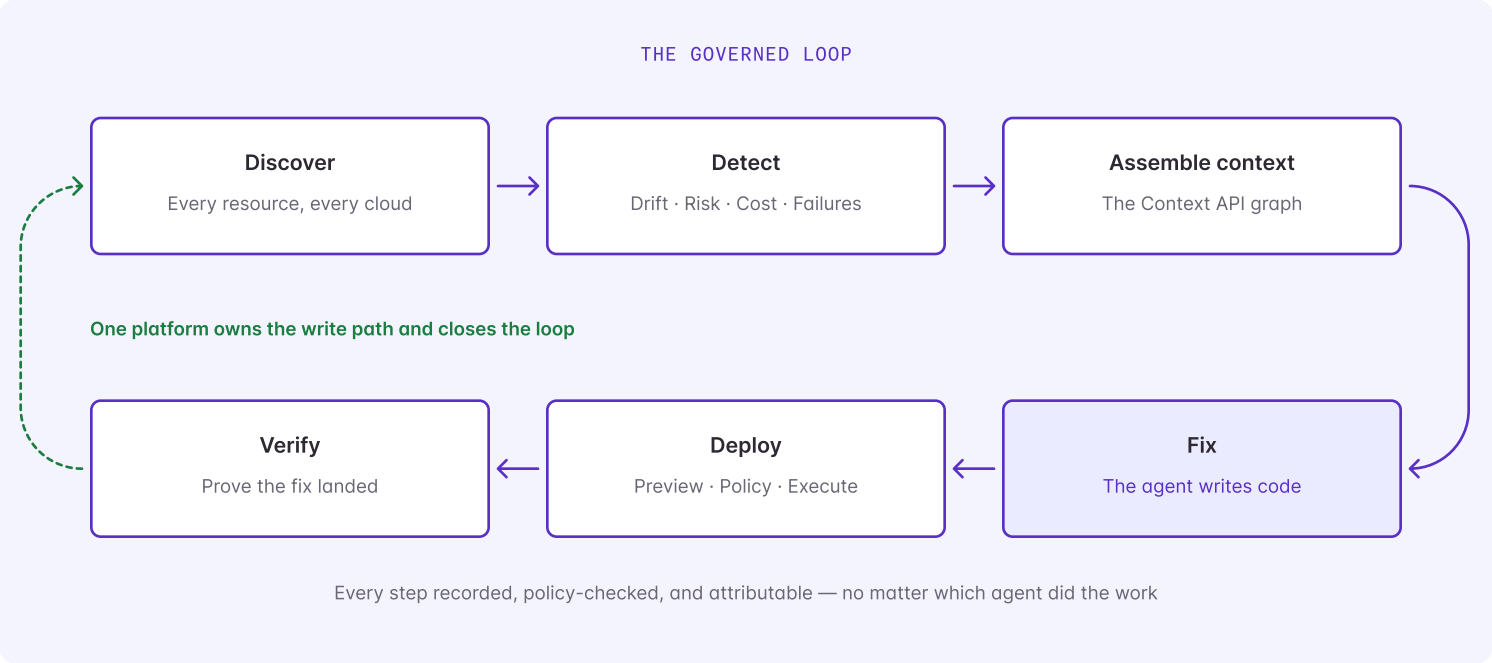

Figure 2. The governed loop: discover, detect, assemble context, fix, deploy, verify — closed inside one platform.

When agents handle authoring, it is effectively free, and the bottleneck shifts to verification and governance. One agent making one change with a human watching is manageable. Teams of agents making thousands of changes in parallel, across every cloud you run, are not... unless something above them can see, constrain, and reconcile everything they do. Slowing agents back down to human review speed defeats the purpose. The job is a control plane that lets a small platform team stay in control of a large fleet without becoming the bottleneck again.

> **"The infra team acts as groundskeepers, not gatekeepers — promoters for the entire org."**
>
> SUPABASE, WHICH RUNS SIXTEEN REGIONS AND EIGHTY THOUSAND RESOURCES WITH ONE SMALL PLATFORM TEAM

03 - CONTINUED

One principle governs the design: **governance must live in the platform, not in the prompt.** You can't instruct your way to safety. A better system prompt can't guarantee an IAM change isn't too broad, or the creation of an orphaned 2 a.m. resource that nobody can attribute. Pulumi Cloud protects against this no matter which agent acted or how cleverly the request was made.

A system of record.

One source of truth for state and change including full history, drift detection, and a complete record of every action and every actor, human or agent. "Which agent changed this, when, and why?" is a query, not an investigation.

Pulumi Cloud

Identity and secrets built for agents.

Agents never hold standing keys. Credentials are issued just-in-time, scoped to the task, and expire on their own, and secrets never land in a prompt or a log. The vaults you already run plug in underneath.

Pulumi ESC

Guardrails at machine speed.

Security, compliance, cost, and architecture rules are evaluated automatically on every plan, regardless of who or what produced it. The same guardrails that govern a human's pull request govern an agent's plan.

Pulumi Policies

Visibility across everything.

When agents create faster than humans can track, visibility is the first thing to break. One view of the entire estate, including whatever an agent created an hour ago, with cost attribution, orphan detection, and remediation.

Pulumi Discovery

Semantic understanding of the whole estate.

Governance and agents draw on the same deep context: every resource, how it's configured, how it connects to everything else, who deployed it and when and why, the code that defines it, what it costs, and whether it's compliant and healthy. It is the difference between an agent that guesses about your environment and one that knows it.

Context API

Governed execution.

Agentic work needs rails with automation, orchestration, and reactive execution of infrastructure tasks, so the work agents do happen inside the system (recorded, policied, repeatable), not beside it.

Pulumi Deployments

Hand agents a fluent substrate with no control plane and you've armed a fleet to change production at machine speed with no brakes. Stand up a control plane on a substrate agents can't write to and they'll default to routing around it creating shadow infrastructure your records can no longer see. Enablement and governance have to be the same system so the thing that authors a change must also be the thing that records, previews, and polices it.

> **Without the substrate, you wouldn't need the control plane; without the control plane, the substrate would be unsafe.**

03 - CONTINUED

The two reinforce each other, and neither stands alone. Without the substrate, you wouldn't need the control plane. Agents can't reliably author infrastructure beyond stringing together bespoke CLI and API invocations, so there's nothing to govern. Without the control plane, the substrate would be unsafe with a fleet of agents fluently changing production with no record, no bounds, and no brakes. Pulumi is deliberately both, in one platform.

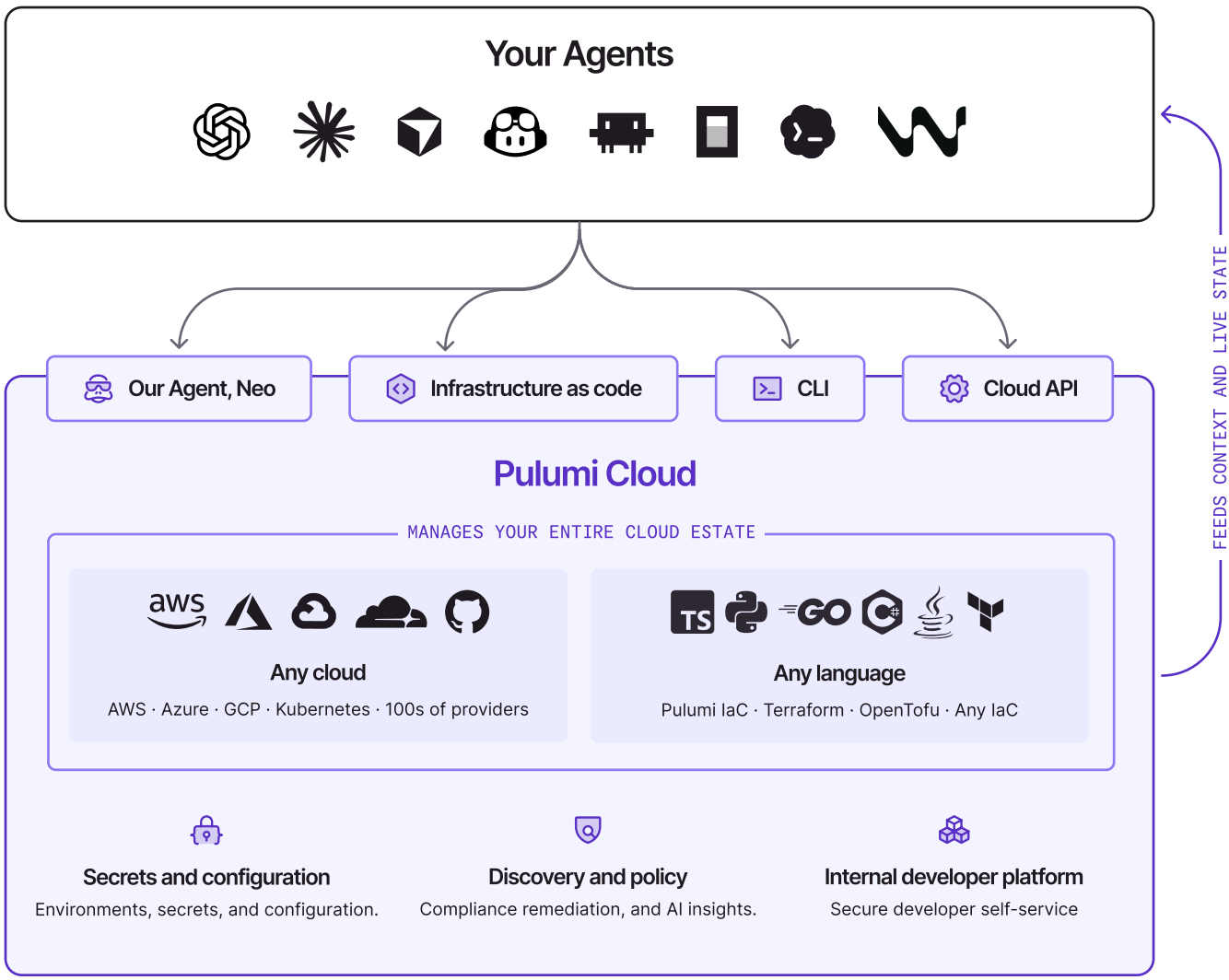

Figure 3. The agentic infrastructure platform: Pulumi IaC at the center; Neo, ESC, Discovery, and IDP around it; Context API, Policy, and Deployments cut across.

04

The road to autonomy

Nobody with an existing estate goes straight to autonomous infrastructure. Start with human-in-the-loop and expand as trust builds. As models improve and agents accumulate a track record within your guardrails, the risk/reward balance shifts, and you delegate more. The control plane is how that trust gets earned, measurably.

STAGE	WHAT AGENTS DO	WHAT HUMANS DO	TYPICAL WORKLOADS
Assisted	Author changes; every plan reviewed	Approve every change	Production deploys, new infrastructure
Supervised	Execute within policy; notify on action	Watch notifications; step in on exceptions	Compliance fixes, scaling within limits
Autonomous	Self-manage within tight bounds	Set intent and bounds	Cost optimization, anomaly response, sandbox cleanup

The platform is built so the smallest possible commitment unlocks each step: 1) an agent can sign up and act in seconds, 2) graduate to authoring full infrastructure as code with previews, and 3) eventually take on asynchronous, autonomous work under the full control plane.

Pulumi Neo is a showcase vertical agent for infrastructure. But Neo isn't the only agent Pulumi assumes you will run and meets you where you're at by supporting the industry's best agents, whether that's Codex, GitHub Copilot, Claude Code, Devin, or any agent that supports standard modern protocols. Pulumi's CLI, REST APIs, and platform overall has been designed with agents in mind. Neo is also decomposed into public skills. InfraBench, our infrastructure analog of the industry standard SWE-Bench coding benchmarks, keeps us honest and Pulumi-equipped agents measurably outperform raw ones, and continues to compound over time.

This approach allows full delegation of infrastructure tasks. For instance, "upgrade every Kubernetes cluster we run, across every cloud and account." The agent discovers the topology, proposes a wave-based rollout, understands nuances like canaries versus blue/green, and executes with approval gates. Canary fails? Stop. Canary succeeds? Carry the learnings forward. That's the direction the platform is built toward, and it's only possible because the engine, the resource graph, policy, execution, and the agent operate as one system.

An observation from teams running this today is to ask what blocks promoting a workload to the next stage, and the answer is almost never model capability. It's a governance gap with missing policy coverage, no agent identity, no scoped credentials. That's good news. It tells you exactly where to invest, and it's all buildable today.

05

This is already happening

Adoption is steep and one-directional. The share of agent-initiated deployments on the Pulumi platform went from essentially zero to roughly 28% in four months, compared to deployments previously run by humans, and the curve keeps bending upward. Our projection is that it approaches 80% within a year.

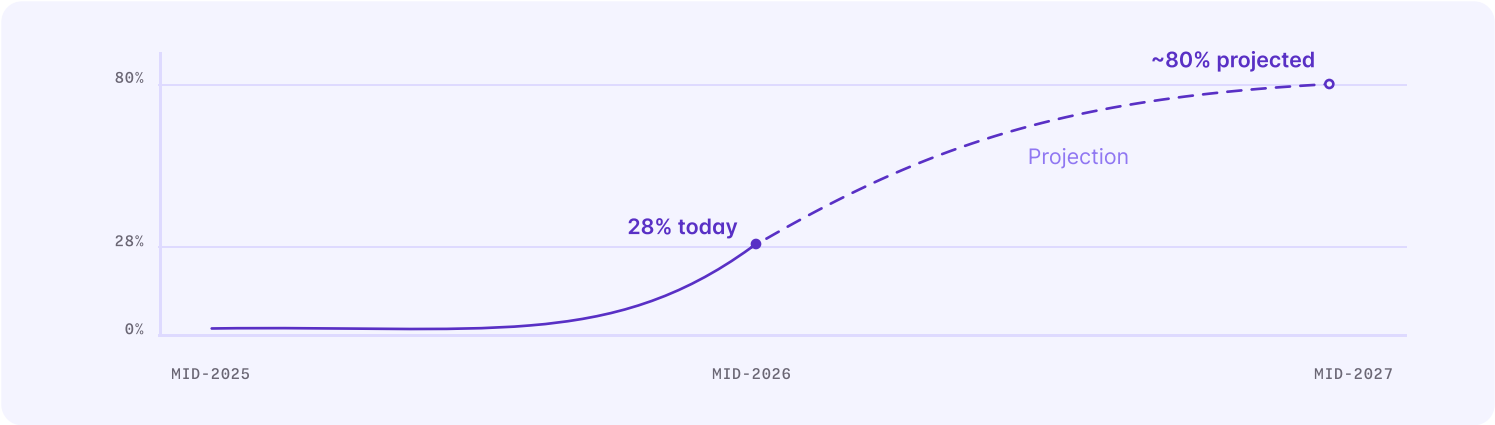

Figure 4. Share of Pulumi deployments previously initiated by humans, now initiated by agents.

The agents doing infrastructure are coding agents. The work is dominated by the same serious software-engineering agents driving the coding flywheel, which is exactly what you'd expect once infrastructure has become a coding problem.

Capability shows up in outcomes. When we compared failed deployments where Neo was engaged versus those without it, the Neo-worked deployments recovered at roughly 85% versus about 24% without over a 24-hour period. That's more than a 3x uplift, and the gap holds across every time window. Agents on the right substrate, inside the control plane, don't just ship changes faster; more of those changes land safely in a correct, applied state.

Figure 5. Deployment recovery within 24 hours, Neo-worked vs. not. Source: Pulumi internal data, 2026; methodology available on request.

And it holds at every scale.

05 - CONTINUED

Over 4,000 companies run on Pulumi, and the pattern extends well beyond AI-native companies. BMW enforces policy as code across 20,000 cloud resources and 11,000 developers. Different scales, different teams, but it's code all the way down.

supabase

Supabase runs **16 regions and 80,000 resources with one small platform team**; every service engineer ships their own infrastructure in TypeScript, inside guardrails.

WIZ

Wiz drives over 1,000,000 cloud resources with hundreds of thousands of daily updates through the Automation API: infrastructure as a programmable capability of their product.

COMPOSTABLE.AI

Compostable AI runs **100% agent-managed infrastructure**, today, with agents driving IaC through skills and structured tooling.

snowflake

Snowflake cut multi-cloud Kubernetes deployments **from weeks to same-day** on one unified programming model. *"When we demonstrated to people that what used to take a week and a half now, with Pulumi, took under a day, they were shocked."* — Raman Hariharan, Director of Cloud Platform Engineering

A top frontier lab grew its infrastructure footprint 982% in twelve months on code-defined infrastructure, accelerating training, inference, and experimentation. They move at the speed of science.

06

Where to start

You don’t need a transformation program. Adoption runs in three stages, mapping exactly onto the pillars. Build the substrate, add the control plane, then let the agents in. Each stage pays for itself before the next one begins:

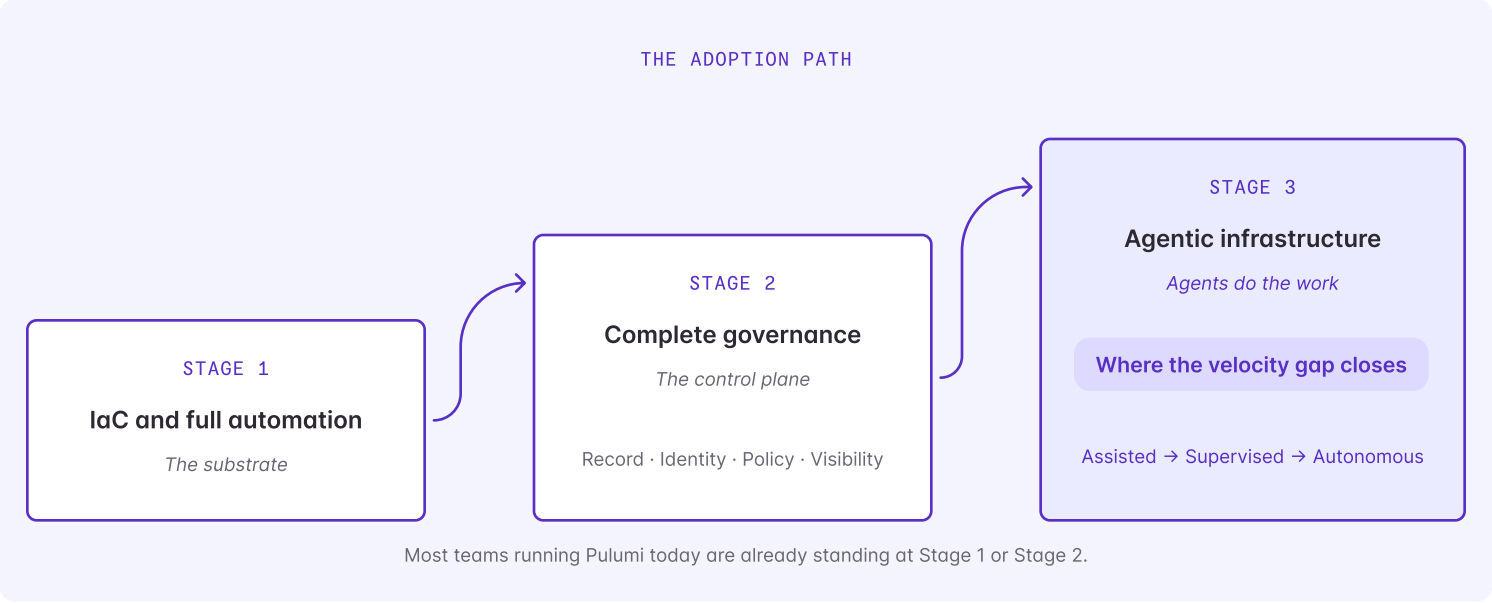

Figure 6. The adoption path: the stages are the pillars, in order, with the agents on top.

Stage 1: IaC and full automation. Get infrastructure into code and deployments automated end-to-end with real languages, reviewable changes, one source of truth. Pulumi runs your existing HCL and Terraform as-is, so this stage starts from wherever you are.

Stage 2: Complete governance. Add the control plane with a system of record, human and agent identity, scoped credentials, policy for every change, and visibility across the whole estate. This is what makes “yes” safe to say.

Stage 3: Agentic infrastructure. Let agents do the work, and widen their autonomy by evidence using the methods outlined in Section 4. This is the stage where the velocity gap actually closes.

If you run Pulumi today, you’re already standing at Stage 1 or 2. Stage 3 isn’t a new platform or a migration. It’s a decision to let agents use the one you have.

06 · CONTINUED

Five moves, in order, to get going this week

01 **Baseline your agent-driven share.**
What fraction of last week's infrastructure changes were agent-initiated? If you can't measure it, that's the finding, and your first gap.

02 **Put one real workload in code.**
Bounded, reversible, owned by a team that wants this. Greenfield or imported, either works. (Stage 1.)

03 **Stand up the guardrails before widening autonomy.**
System of record, agent identity, scoped short-lived credentials, a policy pack, full change history. This is what lets you say yes later. (Stage 2.)

04 **Run agents assisted.**
Preview and approval on every change. Measure against your baseline: recovery rates, time-to-change, drift. (Stage 3.)

05 **Promote autonomy on evidence, per domain.**
Widen the set of changes agents own as the data earns it. You'll be at different stages for different domains at once, and that's exactly right. (Stage 3.)

Wherever you're coming from, the path is seamless. If you have years of Terraform debt, you don't have to throw it away, or even convert it to get started. Pulumi runs HCL and existing Terraform programs as-is, alongside YAML and real languages, with the same engine, previews, policies, and control plane across all of them. Your existing secrets stores plug in as-is too. So adopt the operating model first, and convert incrementally; conversion is a coding task now, the thing agents are good at, so Neo can do the bulk of it when you're ready. Coexist, then converge on your schedule.

Five questions we ask teams to locate themselves honestly

- Hand an agent a service another team owns and ask it to extend it safely. Does it read and build on the real infrastructure, or does it invent resource types because that infrastructure lives in a DSL or a console it never learned? Does it resort to ad-hoc, possibly unsafe CLI commands and direct API invocations?
- How much of your production estate could an agent rebuild in a clean account from your source of truth alone, with no human supplying undocumented steps?
- An agent changed production at 2 a.m. Can you produce the rationale, the plan, the approvals that did (or did not) take place, the policy decision, and the agent's identity in minutes?
- Double the number of agents acting on your infrastructure tomorrow. Does governance scale automatically, or does human review become the bottleneck again?
- For the changes you would not let an agent execute alone today: is the blocker the model's capability, or your inability to bound and prove what it does?

Most teams can't answer all five, and that's the point. The blocker is almost never the models; the models are exceptionally powerful. It's the operating model around them, and that part you can fix now.

CLOSING

The road ahead

Code went first because it was in-distribution, verifiable, and a primary focus for many major labs. Infrastructure follows the same arc once it lives on the same substrate. Infrastructure was never fundamentally harder, it was just expressed in ways the models couldn't see, and operators couldn't fully govern it. Put it in code, and it becomes a coding problem that improves on its own. Govern it from a single control plane, and you can let the agents do the work. That's how the velocity gap closes and how infrastructure starts to perform like the innovation budget it has become.

If you've read this far, you probably didn't need convincing about where infrastructure is headed. Maybe you championed cloud engineering inside your company years ago, and this is the same bet with the stakes raised. Either way, you may be ahead of your own organization on it. This paper exists, in part, to be forwarded to the team that doesn't believe it yet.

Start today, with the tools you already have

- **Have an engineer point their coding agent at Pulumi and ship something real this afternoon.** A free account is all it takes.
- **Let Neo investigate your last failed deployment,** in read-only mode, and see what it finds.
- Or bring your infrastructure, security, and platform leads to a 30-minute **Agentic Infrastructure Readiness Briefing**: we'll locate you on the autonomy ladder, identify the gaps blocking your next stage, and leave you a concrete plan, whether or not you ever run a line of Pulumi. Reach us at pulumi.com/contact, or talk to your account team.

About Pulumi

Pulumi is software-driven infrastructure: the platform where humans and agents operate the cloud together. Built on an Apache-2.0 open-source core spanning thousands of providers, Pulumi lets teams of humans and agents define infrastructure in real programming languages (TypeScript, Python, Go, C#, Java) and pairs that substrate with a complete control plane: Pulumi Cloud as the system of record, ESC for secrets and identity, Pulumi Policies, Pulumi Discovery, and Neo, the AI infrastructure agent. Over 4,000 companies build on Pulumi, from AI-native startups to global enterprises to top-three frontier labs. Learn more at pulumi.com, or read the founding argument in “The Agentic Infrastructure Era.”

ABOUT - CONTINUED

	WHAT IT IS	ROLE IN THE OPERATING MODEL
Pulumi IaC	Infrastructure as code for any cloud, in any language: TypeScript, Python, Go, .NET, Java, and YAML, on an open-source engine	The substrate: infrastructure agents are fluent in
Pulumi ESC	One interface for all your secrets and configuration, across every vault, with dynamic short-lived credentials	Control plane: every actor on scoped, short-lived credentials
Context API	A data lake with semantic understanding of every resource, configuration, dependency, deployment, cost, and health signal	Control plane: the shared context governance and agents both draw on
Pulumi Policies	Auditing, remediation, and enforcement of security, efficiency, and compliance guardrails on every change	Control plane: rules that hold at machine speed
Pulumi IDP	Self-service infrastructure at scale: golden templates, components, and portals for developers and scientists	Golden paths on the substrate, for humans and agents alike
Pulumi Discovery	See everything, control everything: multi-cloud search from a single pane of glass, with AI-powered insights	Control plane: see and govern everything, across every cloud
Pulumi Deployments	Infrastructure automation, orchestration, and reactive execution of tasks across the whole estate	The rails agentic work runs on, governed end to end
Pulumi Neo	The industry's first AI agent purpose-built for infrastructure, with human-in-the-loop controls and a full audit trail	The agent, riding on both pillars

Sources & notes

Coding-benchmark figures refer to SWE-bench Verified (and SWE-bench Pro for contamination-resistant trends). Infrastructure-synthesis hallucination finding: TerraFormer research paper (Jan 2026). Code-action result: Wang et al., "Executable Code Actions Elicit Better LLM Agents" (CodeAct), ICML 2024, measured across 17 models. Karpathy quote: public remarks on shipping a vibe-coded application.

Training-distribution characterizations are deliberately qualitative. Public infrastructure-DSL code is meaningfully present in model training (models handle common Terraform patterns well) but is orders of magnitude smaller than general-purpose-language corpora and skewed toward tutorials and boilerplate; precise line counts are not publicly auditable and should not be cited as fact.

Pulumi figures (agent-driven deployment share ~28%, ~80% forward projection, Neo recovery rates) are Pulumi internal data as of 2026; forward-looking figures are projections, not demonstrated results, and statements about the road to autonomy describe the platform's direction, not exclusively shipped functionality. Methodology for internal metrics is available on request.

Hyperscaler capital expenditure: ~$700B guided for 2026 across the major cloud providers, roughly 75% of it allocated to AI infrastructure (company guidance, as reported by CNBC and Data Center Dynamics, February 2026). IaC coverage: Firefly, State of IaC (89% of teams use IaC; 6% report full coverage). Agentic coding share: Anthropic has stated publicly that Claude now writes over 90% of its code, with 80%+ of production code merged (as reported May 2026).

Customer figures, quotes, and logos (frontier lab, Supabase, Wiz, Compostable AI, Snowflake, BMW) are drawn from Pulumi's published materials and customer references, as used in Pulumi's go-to-market materials. Neutral marks stand in where logos are not used.

The opening CIO quote and the executive anecdotes in the opening note come from private customer conversations; they are shared anonymized, with identifying details removed and wording lightly paraphrased.