

WHITE PAPER · 2026

The Definitive Guide to AI for DevOps / Agentic DevOps

Everything your engineers need to learn for your organization to reach the next stage of agentic AI.

WRITTEN BY:



Jennifer Riggins

Table of Contents

START HERE

- Executive summary
 - The voices in this guide
 - Introduction: The pivot to production
-

PART 1: WHERE WE ARE TODAY —

- 01 The next level of abstraction
- 02 More code than ever is a problem
- 03 The persistent review problem at AI scale
- 04 Choosing the right thing to build
- 05 Demand for DevOps is only starting

PART 2: WHERE WE NEED NOW —

- 06 Guardrails, not gates
- 07 DevOps drives agentic experience
- 08 What should top your DevOps for AI wish list?
- 09 Proactive, preventative DevOps
- 10 Agentic observability
- 11 Agentic security
- 12 Shift to multi-agent systems
- 13 Multi-agent observability

PART 3: MEASUREMENT THAT MATTER —

- 14 Metrics for AI in DevOps
- 15 Error budgets as guardrails

NEXT STEPS —

Upskilling for agentic DevOps, with KodeKloud



ABOUT THE AUTHOR

Jennifer Riggins

Jennifer Riggins is a tech storyteller and journalist, event and panel host. She bridges the gap between business, culture and technology, with her work grounded in the engineer experience.

Executive Summary

Six takeaways from the research and the two dozen practitioners interviewed for this guide — and where to go deeper on each.

TAKEAWAY 1

The bottleneck has moved.

Writing code was never the constraint — safely getting it to production is. AI velocity is rendering human-paced PR culture obsolete, shifting engineers from individual contributors to orchestrators of agentic systems.

TAKEAWAY 3

Guardrails, not gates.

Intent must leave prompts and checklists and become machine-readable, executable policy inside the delivery workflow. The teams with the strictest guardrails move fastest — the rules are already in the rails.

TAKEAWAY 5

Multi-agent systems are distributed systems.

Trained on post-mortems, runbooks, and live telemetry, agents close the loop between detection and resolution — root cause analysis, intelligent rollbacks, and pre-approved auto-remediation with humans in the loop.

TAKEAWAY 2

AI is an amplifier of everything.

Organizations with golden paths, progressive delivery, and platform engineering compound their gains. Those without them ship instability faster — DORA finds throughput up, but stability and security down.

TAKEAWAY 4

Ops turns proactive.

Trained on post-mortems, runbooks, and live telemetry, agents close the loop between detection and resolution — root cause analysis, intelligent rollbacks, and pre-approved auto-remediation with humans in the loop.

TAKEAWAY 6

You can't improve what you don't measure.

DORA's five, the MTT-x family, AI confidence metrics, and error budgets wired directly into agent behavior— measurement is what turns agentic adoption from a demo into a durable advantage.

WHO IS THIS FOR

Individual contributors planning their upskilling journey — and the engineering leaders responsible for filling the gaps.

TEAM SCALE UPSKILLING → [KODEKLOUD.COM/BUSINESS](https://kodekloud.com/business)

The Voices in This Guide

29 PRACTITIONERS · 24 ORGANIZATIONS

Charity Majors

CTO, Honeycomb

Kelsey Hightower

Distinguished Engineer, ex-Google

Grady Booch

Co-Creator of UML

Dana Lawson

CTO, Netlify

Dave Farley

Pioneer of Continuous Delivery

Michael Privat

Chief Data & Engineering Office,
Avallity

Michael Forrester

DevOps Advocate, KodeKloud

Theo Valmis

Founder, Mneme HQ

Tarak Bach Hamba

Senior Growth Manager,
Alkido Security

Charlotte Fleming

Octopus Deploy

Steve Fenton

Octopus Deploy

Nimesha Jinarajadasa

DevOps & Cloud Consultant,
KodeKloud

Asaf Savich

Komodor

Phanidhar Koganti

Hewlett Packard Enterprise

Jason Davenport

Area Tech Lead, DevEx & AI, Google

Ryan Peirce

Systems Engineer,
Palo Alto Networks

Mayank Agarwal

Co-founder & CTO, Resolve AI ·
co-creator, OpenTelemetry

Rodrigue Schäfer

VP of Tech Foundations,
Delivery Hero

Grace Torany

Software Engineer, Nvidia

Kinjal Vaishnav

Staff Reliability Engineer III, Oracle

Ben Lloyd Pearson

Director of Developer Experience,
LinearB

Laura Tacho

Senior Principal Technologist, AWS

Nana Janashia

Host, TechWorld with Nana

Gergely Orosz

The Pragmatic Engineer

Drew Hoskins

Author, The Product-Minded
Engineer (O'Reilly)

Taylor Bruneaux

Analyst, Dx

Introduction

The pivot to production

The second half of 2026 marks a definitive pivot in the software development lifecycle from *How fast can we write code?* to *How can we safely get code into production?*

Since writing code was never the bottleneck, AI velocity is quickly rendering pull request (PR) culture obsolete. Engineering managers and leaders have to prepare their developers for this move from code creation to agentic supervision and operations. From individual contributor to orchestrator of agentic systems.

As Honeycomb CTO Charity Majors puts it, “AI came for code generation first because it was the easiest problem to solve.” Now it’s time to get down to the real problems that teams have been facing for decades — security, uptime, user centricity, feedback loops — at a scale like never before.

“AI came for code generation first because it was the easiest problem to solve.”

- CHARITY MAJORS · CTO, HONEYCOMB

We’re reaching a point where potentially anyone — including non-technical subject-matter experts — can become a builder, so guardrails, gates, and policies become increasingly impactful on your organization’s ability to safely, securely, and reliably scale. This only serves to create more operational toil than ever. Add on the novel bugs and security threats because of AI, and the DevOps engineer simply can’t keep up.

McKinsey’s tech talent research again found last year that the biggest skills gaps the industry is facing continues to include cloud computing and DevSecOps. Specifically with DevSecOps, the consultancy finds that companies are shifting security left, integrating it within the CI/CD lifecycle, but still struggle to hire. And now, Storyblok finds, one in four enterprises don’t think they can attract and retain the engineering talent needed to keep systems running.

Many enterprises are making short-term OpEx gains via blanket layoffs, but forward-thinking organizations are investing in their future by upskilling software engineers. This not only fills in existing talent gaps, but the new ones opening at AI scale — because DevOps engineers are the stewards of scale.

“If we’ve democratized your ‘hardcore’ skills, and now those are no longer as valuable as they used to be, then you’ve got to go learn some additional skills. A lot of those people are going to get forced to broaden the scope of what their abilities are,” remarks ex-Google’s Kelsey Hightower. For him, the core question to being an engineer hasn’t changed: “‘How can I be better at this?’ And if you aren’t sure, you go find someone that you thought was better at this and learn from them.”

At a time when engineers are building and managing multi-agent systems, everyone needs a better understanding of architecture, guardrails, security, quality, continuous delivery pipelines, platform engineering, intelligent monitoring, root cause analysis, and more.

The International Data Corporation (IDC) argues that success with AI adoption on the developer side hinges on how “effectively AI is embedded into the core of IT operations to enable continuous modernization and deliver measurable business value.” This has 17% of organizations actively rebuilding for this Age of AI, “shifting to a continuous, incremental application maintenance model to drive velocity, agility, and the systematic reduction of technical debt.” Another 30% are at the AI pilot stage in operations.



WHY THIS GUIDE EXISTS

Since 2017, KodeKloud has been looking to help fill in those DevOps and cloud-native gaps with self-guided, hands-on training.

As an extension of that work, this book aims to give you an overview of what the modern engineer needs to know from the other side of the proverbial wall between devs and ops. And it discusses what those already sitting in operations need to know to be ready for these agentic changes.

Two-thirds of DevOps teams have already integrated AI into their delivery pipelines, but Thoughtworks finds, only 12% have achieved true AI-driven operations.



2/3

of DevOps teams have
already integrated AI into
their delivery pipelines

THOUGHTWORKS



12%

have achieved true AI-
driven operations

THOUGHTWORKS



1 in 4

enterprises don't think they
can attract and retain the
engineering talent to keep
systems running

STORYBLOK

The next shift toward agentic workflows with multi-agent systems requires even more complex governance. Just like AI has the potential to make software engineering more strategic, it can take DevOps teams from passive dashboard monitoring to proactive or even automated resolution and remediation.

Then, this guide will take you deep into AI for operations. Learn the foundational guardrails needed for agentic ops, and how to apply autonomous agents plus humans-in-the-loop to root cause analysis and auto-remediation, shifting ops from reactive to proactive.

Finally, because you can't improve what you don't measure, we will touch on some of the most important metrics to know how your agentic strategy impacts stability, security, and the bottom line.

This guide to AI for DevOps is something that can help both the individual contributor plan your own upskilling journey, and engineering leadership understand the gaps they need to fill.

Remember, just like DevOps itself, the multi-agent AI system adoption is a journey, not a destination. Let's all support each other in learning together.

PART 01/03

Where we are today

The state of play: what AI velocity is doing to code, to review, and to the teams who keep it all running in production.

- 01 The next level of abstraction
- 02 More code than ever is a problem
- 03 The persistent review problem at AI scale
- 04 Choosing the right thing to build
- 05 Demand for DevOps is only starting

The history of software is defined by constantly rising levels of abstraction, argues Grady Booch, co-creator of the unified modeling language (UML). Each time there is an abstraction, some skills become obsolete and there's demand for new skills. As we enter this third golden age of software engineering, we are abstracting away the tedium of traditional code writing. Moving forward, engineers will deploy agents to automate the "complex and messy" work.

"Move up a level of abstraction, start worrying about systems," Booch recommends.

"If you have the skills of knowing how to manage complexity at scale, if you know as a software engineer how to deal with all of these multiple forces, which are human as well as technical, your job's not going to go away. If anything, there will be even greater demand for what you're doing because those human skills are so rare and delicate."

No doubt, it's a scary time for software engineers. Suddenly this lucrative career path is obstructed by massive uncertainty. Smart organizations want to help their engineers take that leap. It saves you from lengthy and often impossible operations recruitment cycles, and preserves loyalty, consistency, and institutional knowledge. All while ensuring that your security and stability remains intact, tying your engineering investment much closer to your business goals.

As Mitch Ashley, VP practice lead at The Futurum Group, puts it,

"AI is the third platform inflection point after Linux and Kubernetes, and the answer is the same one that worked the last two times: an open foundation, a broad ecosystem, and the people who build the rails."

Bigger certainly isn't better when it comes to code.

As Dave Farley, godfather of continuous delivery, argues, smaller batch size is one of the most critical factors to lowering risk and improving software quality. Yet AI-assisted developers and agents are shipping more code in fewer PRs, making batch size bigger, diverging from engineer best practices.

AI-generated code tends to create more bloat with one study finding an average 18% increase in size of code additions per PR.

"The toil didn't disappear. It moved," observes Michael Privat, chief data and engineering officer at Availity healthcare portal. "We used to spend our time writing code. Now we spend it comprehending code we didn't write. Every line an agent generates is a line a human has to trust, and trust at scale is expensive."

This has verification load and fatigue with AI coding assistants running high. Recent research out of Taiyuan University of Science and Technology finds that human-computer interaction is stilted by:

- a Interface vs. model confusion.
- b Process costs are unspecified.
- c Unclear mode crossovers.

All resulting in more cognitive load than ever, for both your developers and your DevOps engineers.

"The new toil is review fatigue, archaeology on three-week-old generated code nobody fully read, and debugging systems where the original 'intent' lived in a prompt that's long gone," Privat says. "Writing code was never the bottleneck. Understanding it was — and AI just made the gap between the two wider."

People, processes, and code have organized around a collaborative software development practice where all code changes are proposed, reviewed, and validated before being integrated into the main codebase.

“Every engineering mantra we live by — fail fast, ship early, move fast — was written assuming a human closes the loop between action and learning. Strip that human out, and the principle inverts,” he continues.

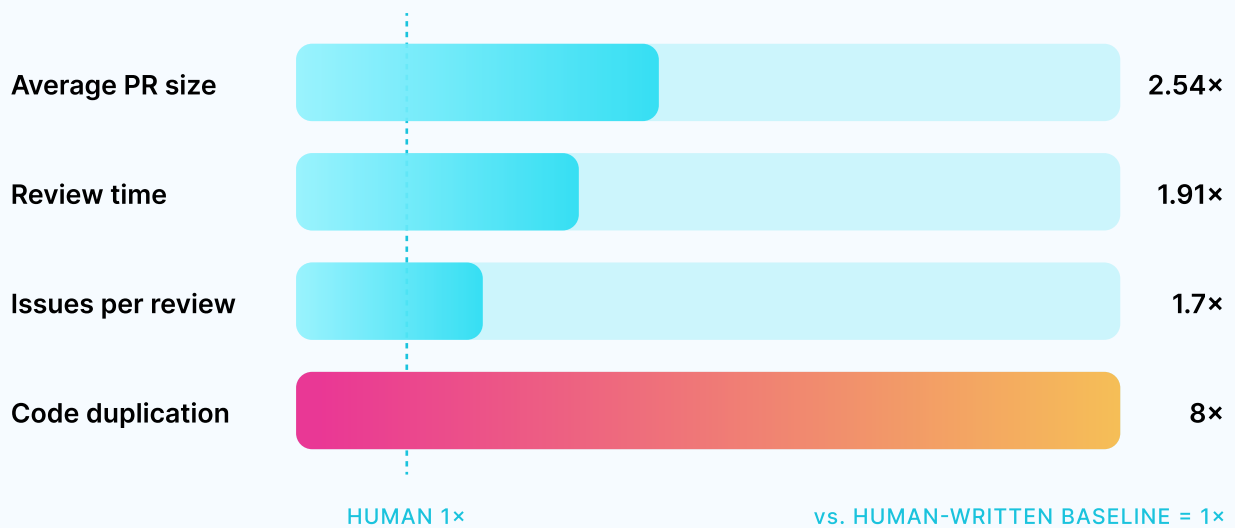
“Agents accelerate decisions without accelerating learning cycles, so instead of failing fast, you fail at scale. The most dangerous moment isn't when an agent writes bad code. It's when an agent writes plausible code and the organization has quietly stopped checking.”

“Every engineering mantra we live by — fail fast, ship early, move fast — was written assuming a human closes the loop between action and learning. Strip that human out, and the principle inverts. Agents accelerate decisions without accelerating learning cycles, so instead of failing fast, you fail at scale.”

MICHAEL PRIVAT, CHIEF DATA AND ENGINEERING OFFICER, AVAILITY

Human reviewing capacity cannot keep up pace with AI-generated code. A recent CodeRabbit report finds that AI-generated PRs include roughly 1.7x more issues per review than human-generated ones, further complicating the review of these larger diffs.

And it's simply harder to review code that you didn't create. (And arguably less fun.) Engineers are feeling increased “verification fatigue,” where their mental models change from direct creator to orchestrator of AI agents. Telemetry from Faros AI — which actually argues that it's more like a 154% jump in average pull request (PR) size — finds that AI-generated code has 91% longer review times.



According to Faros AI, review time has increased by up to 90% with humans acting as quality gates around AI.

While lines of code is a bad metric at the individual productivity level, it's a great measure of your codebase's complexity, which explains why it's harder to maintain code written or deployed by agents.

As DX's Taylor Bruneaux writes, code doesn't age, it decays. This code rot, where code quality degrades even a month in, is accelerating in the face of AI-assisted code. This includes increasing duplication by 8x, which creates hidden technical debt that can cost companies millions.

"When code becomes harder to understand, reason about, and change, it introduces cognitive and emotional friction for developers," Bruneaux writes. "That friction compounds, slowing throughput, eroding confidence, and weakening organizational adaptability."

This all compounds to more weight on the DevOps and site reliability engineering (SRE) teams' shoulders, as they are the stewards of quality and security.

Software development teams need more feedback faster as the cognitive load has gotten a lot worse in the face of AI.



“Devs don’t need AI to write their code, they need it to get out of the damn queue.”

TARAK BACH HAMBA · SENIOR
GROWTH MANAGER, AIKIDO
SECURITY

“Devs don’t need AI to write their code, they need it to get out of the damn queue. Most of the frustration I see isn’t about building, it’s about waiting. Waiting for infra, approvals, or some process that exists ‘just because,’” said Tarak Bach Hamba, senior growth manager at Aikido Security. “That’s where AI could help, navigating policy bottlenecks, pre-validating checklists, catching config and security issues before review and unblocking safe deployments.”

Just don’t go rushing to have AI as your single code reviewer. As Quotient CEO Lizzie Matusov warns, “That might sound efficient, but research shows it weakens the responsibility developers feel toward code quality and social accountability.”

For AI across the software developer lifecycle, every engineer must maintain a sense of ownership over their code.

Now, with the deluge of AI-assisted and AI-generated code, development work has shifted from writing to reviewing code. Review time has increased by up to 90% with humans acting as quality gates around AI.

Developers are also waiting around for code to be deployed to production. These bottlenecks are most likely networking, security, and/or legal’s manual approvals, and are all ripe for AI automated guardrails.

Generative AI is famously very persuasive. A lot of the time it’s almost accurate that only the most senior developer may notice why it’s not 100% correct. In fact, AI-coauthored PRs show 1.7 times more issues compared with human-only PRs.

A recent study out of the University of Windsor finds developers in “debugging hell,” with 73% of AI samples containing vulnerabilities — despite developers reporting that they feel more confident in the security of their code.

Daily AI users ship 60% more PRs than non-users.

73%

of AI code samples contain vulnerabilities — while developers report feeling more confident in their security

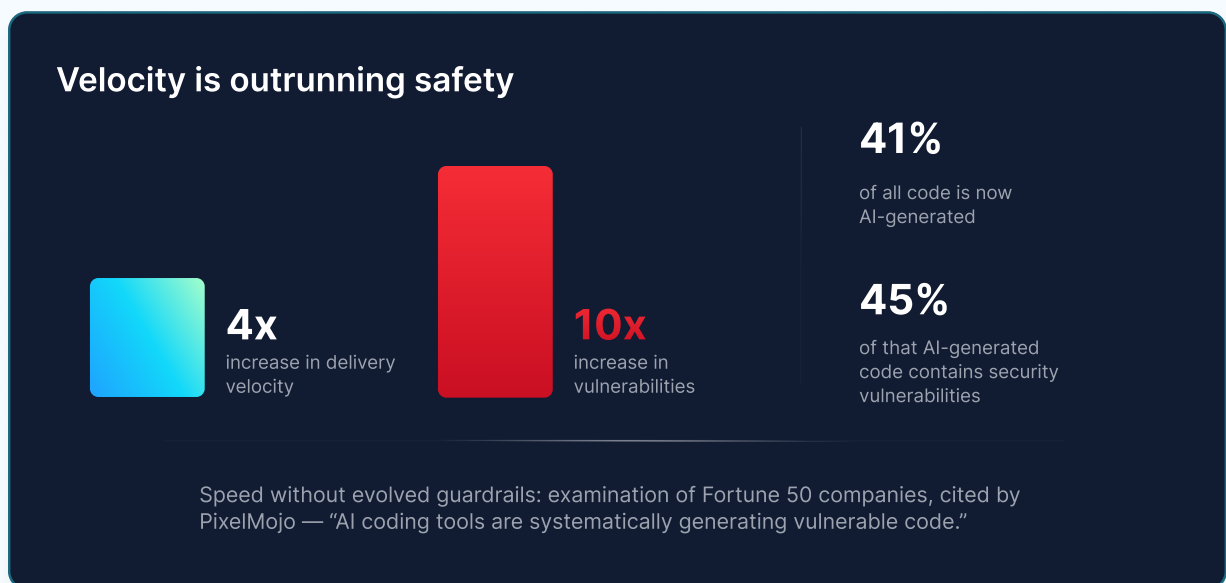
UNIVERSITY OF WINDSOR

+60%

more PRs shipped by daily AI users than non-users— all of it landing in someone's review queue

DX

Examination of Fortune 50 companies finds that, when companies saw 4x increase in velocity, they saw 10x vulnerabilities. Mostly that's tied to 41% of all code being AI-generated, of which 45% contains security vulnerabilities. As PixelMojo puts it, vibe coding is triggering a technical debt crisis, and, "Unlike traditional security debt that accumulates gradually through neglect, AI coding tools are systematically generating vulnerable code."



A recent GitClear Research Report also found that "Development teams are firefighting recent AI-generated problems rather than addressing genuine technical debt," which just slows everything down.

Expanding AI infrastructure makes it difficult for DevOps and SRE teams to bridge siloed data and tools, often taking them farther from progress with cloud migrations and continuous delivery pipeline automation.

It's no wonder that ops teams are as burnt out as ever, with alert fatigue leaving things missed. Add to this there has always been a hiring pipeline problem where there were never enough trained folks on the ops side.

The value of knowing what to build is becoming much more important than the technical skills of how to build it. All while blindly trusting the AI to accomplish what you tell it to do increases overall risk.

If you agree with the Outcome Engineering (O16g) manifesto, it was never about the code. Signees of this new movement believe that agents remove the constraints on human capacity. "Creation is now limited only by the cost of compute," the O16g manifesto reads. The backlog is dead, as everything built now must be grounded in understanding human intent and impact on customers: "If the outcome is worth the tokens, it gets built."

Which means every engineer, no matter where you sit in the technical organization, must become closer to the business objectives and closer to the customer.

Platform engineering is one way to create translation overlap between developer work and business outcomes. Golden paths for developers also create the easiest and safest paths for agents to achieve anything.

If, as Nana Janashia, host of TechWorld with Nana, puts it, the historical role of coding is a means to an end, with that goal being to support user needs, then the speed of AI should bring the average developer closer to their users. This is already shown in how AI enables rapid prototyping, tightening developer-user feedback loops.

Pragmatic Engineer's Gergely Orosz says that AI is increasing the demand for product-minded engineers, the developers who understand business goals, know why they are building a feature, and care about its user impact.

This mindset shift must now extend into operations, which, in the face of ever-tightening budgets, can no longer afford to maintain its distance from business. They must instead show how they are the enabler of AI adoption at enterprise scale.

Author of O'Reilly's *The Product-Minded Engineer* Drew Hoskins gives his advice for mid- and senior-level engineers who want to be more product-minded:

- 01 Ask why a lot.
- 02 Switch your viewpoint, going from the system to user level and back again.
- 03 Simulate and sequence user interactions. Write scenario tests.
- 04 Spend time on user support.

The rise of platform engineering reminds us of the business benefits of treating internal users like customers.

05 Demand for DevOps is only starting

“Regardless of what an AI tool is promised to be capable of delivering, every engineering leader needs to remember that AI will not reduce the need for DevOps. In my view it does the opposite,” argues Gustavo Melki Portaluri, head of product engineering at Storyblok. “Technical changes will be far easier and faster to implement, but the consequences of getting those changes wrong are not eliminated.”

Acting with speed but without operational judgment is a recipe for risk, he continues, “so I encourage DevOps teams to slow down at the review stage.”

This just makes the role of the DevOps engineer an even more strategic one. AI forces more people to work cross-functionally and cross-organizationally, which, Melki Portaluri says, drives an increase in people — even potentially those domain-expert citizen developers — generating infrastructure and deployment changes, without the appropriate technical understanding.

“Regardless of what an AI tool is promised to be capable of delivering, every engineering leader needs to remember that AI will not reduce the need for DevOps. In my view it does the opposite. Technical changes will be far easier and faster to implement, but the consequences of getting those changes wrong are not eliminated.”

GUSTAVO MELKI PORTALURI, HEAD OF PRODUCT ENGINEERING,
STORYBLOK.

“DevOps is becoming somewhat of a comprehensive immune system for the engineering org. More so than ever before, able to define the guardrails for what is safe to ship, and ensure these parameters are enforced,” he says.

“DevOps acting as an organization’s technical guardian obviously isn’t new, but AI-assisted development is making it a far more urgent and visible problem. DevOps leaders need to figure out what those guardrails are as a priority, as not having total ownership of this process will bring vulnerability.”

And ops continues to benefit from a healthy doze of professional skepticism.

“AI will never abstract away the understanding of what’s going on underneath,” Melki Portaluri promises. No matter what your level of experience, “Every member of the DevOps team needs the skills to say: What will failure look like here? What permissions does this actually require?”

Keep an up-to-date checklist of these questions that are relevant to your team, he advises, that you review every time you’re considering making your ops a little more agentic.

Closing the ops hiring gap from within?

KodeKloud for Business upskills whole engineering teams with hands-on DevOps, Cloud, and AI labs.

[KODEKLOUD.COM/BUSINESS](https://kodekloud.com/business) →

PART 02/03

What we need now

The playbook: Guardrails encoded in the pipeline, agentic experience, proactive operations, observability, security and the shift to multi-agent systems.

- 06 Guardrails, not gates
- 07 DevOps drives agentic experience
- 08 What should top your DevOps for AI wish list?
- 09 Proactive, preventative DevOps
- 10 Agentic observability
- 11 Agentic security
- 12 Shift to multi-agent systems
- 13 Multi-agent observability

“If AI agents are just junior developers, then you give them the same guardrails that you would give any junior developer, and then you can just let them go at speed,” Michael Forrester, DevOps advocate at KodeKloud, says. “They can go faster, as long as they're within the confines of the golden paths.”

AI is an amplifier of everything, finds the latest [DORA report on AI-assisted engineering](#).



“If AI agents are just junior developers, then you give them the same guardrails that you would give any junior developer, and then you can just let them go at speed,”

MICHAEL FORRESTER · DEVOPS ADVOCATE, KODEKLOUD

If your organization already has the established engineering best practices in place, like progressive delivery and platform engineering, then you are likely to be even more successful with AI, unlocking those elusive, massive productivity gains dominating headlines. But, if your systems are not prepared to handle this exponential increase in volume of code, then downstream chaos ensues, the DORA team warns, including:

THE DORA WARNING · WHAT UNPREPARED SYSTEMS GET

01 Increased delivery instability:

While AI adoption has improved software delivery throughput, it is also associated with an increase in instability and insecurity. Operations and QA simply cannot keep up.

02 Verification and coordination costs:

An increase in change volume without evolved guardrails, roles, and golden paths via internal developer platforms lead to higher verification and coordination costs. TicketOps risks returning.

03 Systemic bottlenecks:

More than 80% of developers think they're experiencing productivity gains, but often teams do not. DORA points to processes and systems as the cause, as integration and deployment pipelines struggle to keep up.

04 Need for control systems:

Even when AI dramatically accelerates development, the people and processes are struggling to match that pace. This requires fast feedback loops and architectures that enable independent action. AI can help here.

05 Downstream chaos:

Without a strategic focus on the underlying organizational system, like internal platforms and clear workflows, AI creates localized pockets of productivity that fail to scale.

A big part of the problem is that the majority of enterprises have, year after year, millions after millions, failed to modernize. In fact, the IDC finds that 9 in 10 orgs still take an intermittent approach to modernization — of which 12% follow “If it isn’t broken, don’t touch it” — all which leads to more security vulnerabilities, increased costs and budget overruns, and disruption to business.

A graphic showing the number '9 in 10' in a large, bold, purple font. Above the text is a horizontal bar with a blue-to-green gradient.

organizations still take an intermittent approach to modernization

IDC

A graphic showing the percentage '12%' in a large, bold, purple font. Above the text is a horizontal bar with a blue-to-green gradient.

follow “if it isn’t broken, don’t touch it”

IDC

“The answer to going fast is still the same,” Forrester says. “You need to have the proper guardrails in place in order for you to go fast, and the proper guardrails don't have to be onerous.”

If there are any holes in your CI/CD pipeline, he continues, they will be exploited at the speed of AI. That’s why, he argues, now everyone needs to know the fundamentals of CI/CD pipelines, test-driven development, security, compliance, governance, auditing, logging, and observability.

Also, having a healthy data ecosystem that AI can access, working in small batches, and robust version control practices, continue to be what distinguishes high-performing organizations from the rest.

“As teams move from code generation to autonomous agents, the question is no longer whether agents can write code,” says Theo Valmis, founder Mneme HQ architectural governance for AI-assisted development. “It's whether they can consistently make engineering decisions that remain aligned with architectural intent.”

At the speed of AI, guardrails can no longer remain in prompts, checklists, or coding standards, They need to be encoded in a way that is executable within the development workflow.

“The future of engineering guardrails is making that intent machine-readable, enforceable, and continuously available wherever development occurs,” he continues. “As organizations deploy more AI coding agents, they need mechanisms that preserve architecture, engineering standards, security practices, operational reliability, compliance requirements, institutional knowledge, and cross-team consistency. Otherwise, development may accelerate while systems become increasingly fragmented, difficult to maintain, and disconnected from the intent of the engineering organization and ultimately the business objectives.”

This doesn’t obliterate the role of human operators. Simply knowing what’s good or not, what’s secure or not, remains essential.

“A lot of people have said they have lost the ability to understand if something is secure or not. What use are you at that point?” Hightower warns that “this is the danger of any system that removes people’s understanding, because when it’s time to understand, you won’t be able to.”

For Netlify's CTO Dana Lawson, when everyone can become a citizen developer, an understanding of agentic experience is what distinguishes an engineer. This has engineers playing "shepherds of production" that "make sure that what goes in and what goes out is well-understood. If we do agentic experiences right, those agents should be event-driven activities where the signal is pushed to a developer versus pulled."

Now more than ever, success or failure of agentic adoption comes down to a comprehensive understanding of complex systems, the route to production, and business context.

Agent experience, which she defines as a combination of user experience and developer experience that focuses on "designing systems where humans and AI agents collaborate seamlessly to build, test, and deploy software," is grounded in context and intent across the software delivery lifecycle. "It is the practice of designing where humans and agents collaborate seamlessly — and it isn't just about making API calls agent-friendly."

This can include rebuilding or restructuring systems to safely onboard and serve both the new citizen developers and the traditional software engineers. And by better serving that new audience, her teams have found that they create a better developer experience, overcoming institutional and domain knowledge boundaries.

“Every human assumption we removed made the platform better for everyone.”

- DANA LAWSON · CTO, NETLIFY

"When we made the agent error messages clearer, structured the build output for machines, and removed unnecessary friction, our developers also benefited from this," Lawson reflects. "Every human assumption we removed made the platform better for everyone."

The continuous delivery best practices wishlist seems never-ending. Fortunately, if you present that wishlist as a direct enabler of AI, you just might get the budget to finally fulfill it. Even if you do get these infrastructure and process upgrades funded, you don't likely have the teams to deliver on these goals all at once. So, in the speed of AI adoption, what should be your first priorities?

Octopus Deploy's Charlotte Fleming points to test automation as a commonly underinvested capability, "so teams build pipelines but don't usually build the underlying infrastructure."

While agreeing test coverage is often lacking, if you ask her colleague Steve Fenton, priorities inevitably vary by individual team constraints.

"There's usually a major problem at the heart of the socio-technical system that's caused by deployment issues and undetected production issues," he says. "Every time they deploy a software version, it triggers a network of pain and the team is the last to find out."

This inevitably makes it look like your DevOps teams don't know what they are doing, he warns, which just has business add in even more blockers and hurdles, which inevitably increases batch size and makes the next deployment incident likelier.

But, Fenton offers two best practices that usually reduce the risk of this happening:

PRACTICE 01 · DEPLOYMENT AUTOMATION

"Deployment automation removes all the mistakes, timing issues, and lack of consistency around deployments that cause far-reaching incidents. No more mis-configuration or accidentally skipped steps. Just reliable deployments every time," Fenton explains.

PRACTICE 02 · OBSERVABILITY

"Observability means the team can be the first to know when there's a problem. Creating a fast path for high-signal information to reach the team so they can act quickly and calmly to resolve an emerging problem is vital."

The next step for operations should close the loop between detection and resolution, with the possibility of allowing AI agents — in pre-defined circumstances — to auto-remediate.

For KodeKloud's Nimesha Jinarajadasa, this includes self-optimizing CI/CD pipelines, like:

- Detecting flaky tests.
- Suggesting pipeline improvements.
- Optimizing build times.
- Predicting deployment risks.

“Every incident contains signals that can prevent the next one,” says Komodor’s Asaf Savich of preventative reliability engineering. “By capturing and structuring these signals, AI systems can learn patterns of instability and use them to harden infrastructure against future outages, performance degradation, and other issues.”

To kick off your agentic ops, train ops agents on historical data like post-mortems, outages, runbooks, and resource utilization, plus a deep understanding of your systems, your dependencies, and your AI governance policies. In addition to your own context, AI can recognize anomalous patterns across your metrics, logs, and events. For this, an AI Ops foundation of Prometheus, Grafana, and Python, as well as on AI Ops logging and alerting at scale is powerful. Then add on time-series models to further support time-sensitive, context-aware alerts, which again feed into self-healing autonomous IT.

And uptime isn’t a binary either. More and more, ops teams need to be able to recognize gray failures which are harder-to-detect degradations — like high latency — which can cost users as well as increase your cloud bill. These become a death by a thousand paper cuts over time, and they often have simple remediation, even literally turning something off and on again.

“Remediation doesn’t have to be very sophisticated,” says Hewlett Packard Enterprise (HPE)’s Phanidhar Koganti, “because business continuity is the primary goal. Remediation is different from permanently fixing the issue.” It usually buys you time to uncover a permanent fix.

Training in automated remediation with Python is a powerful tool in most ops’ kits.

“When a build fails, an agent can read the logs, because we wouldn't update them, diagnose the root cause, and submit a fix all without a human in the loop,” Netlify's Lawson explains. “This isn't someone's future, this is happening every day, and this will happen in your workforces as well. There will be a demand that comes in and says there's your common task, turn it to a software factory, make it go.”

But in order to do that, you have to have the right limitations and guardrails in place. For Netlify, that includes agent sandboxed environments with full platform access, where, she says, “Agents don't just write the code, they're participating in the enterprise infrastructure lifecycle, they're generating the tests, detecting fault, analyzing build blocks, proposing fixes, opening PRs, CI/CD, just becomes a continuous feedback loop.”

Lawson offers updates that need to happen to enable agentic autonomy:

FROM APIs
→ **TO CAPABILITIES**

While software architecture has always exposed endpoints via APIs (e.g. POST, GET, PATCH), agent-native systems expose intent-level operations through capabilities that express what you want to accomplish (e.g. `create_a_site`, `deploy-repository`.)

FROM REQUEST-RESPONSE
→ **TO EVENT-DRIVEN**

Moving away from request and wait, agents subscribe to events, observe system behavior, and, when permitted, act autonomously.

FROM MACHINE-READABLE
→ **TO AGENT-LEGIBLE**

Shifting away from individuals understanding certain “mysterious” segments of architecture, an agentic platform creates what Lawson calls blueprints that finally reveal architectural complexity — for agents and humans both to understand a system before they make changes.

“This development cycle becomes a continuous human-agent loop, so no longer is it a handoff,” to operations teams, Lawson says. “It is everybody participating at the same time, and the human stays in the loop by providing judgment, taste, direction, but the balloon loop moves at machine speed. This isn't replacing engineers, it's amplifying everybody's ability to be a builder.”

At Netlify, the team has deployed an observability agent. When a build fails and that agent reads the logs, it can not only diagnose the fault, but it can submit a fix and update those logs.

But, at least for now, it's not autonomous everything, she says, with the agent experience remaining grounded in the humans driving it. For the Netlify team, that looks like:

- 01 Each agent only executes within its specified sandbox.
- 02 Human-in-the-loop by default, focusing on the meaningful, intent-driven part of the engineer's job.
- 03 Each agent action is logged so it is auditable and able to be instantly rolled back.

“If you can't explain what the agent did, why would you trust it in production?”

- DANA LAWSON · CTO, NETLIFY

“If you can't explain what the agent did, why would you trust it in production?” Lawson says. “Engineering evolves from implementing every feature to really ensure that the systems and guardrails architecture is tight and solid.”

10 Agentic observability

When code fails in production, Outcome Engineering argues that you don't just roll back, but debug the decision to understand why the hypothesis was wrong. Which isn't simple. This requires a mix of agent observability — into the internal behavior of agents — and agentic observability — where other agents do the observing, root cause analysis, and even remediation.

KEY TERM · AGENT OBSERVABILITY

Visibility into the internal behavior, reasoning, and actions of your agents.

KEY TERM · AGENTIC OBSERVABILITY

Observability done by agents — observing, root cause analysis, even remediation.

“To me, observability hasn't really changed, other than the fact that we now have to manage non-determinism in addition to the rest of the stack,” reflects Jason Davenport, area tech lead for developer experience and AI at Google.

That means that now, our agent observability has to include managing “things that may not behave the same way every time. How do we leverage the things that we know, take software engineering principles really to the best practice side and the extreme to manage agents and manage things that happen with agents?”

Because both agent and agentic observability take what we study as “unknown unknowns” to the next level.

“For me and for our customers, I think the agentic era is switching away from having manual practitioners and operators doing every task and looking at dashboards, to now understanding how to actually plug in pieces of software that are agents that can augment or take on some of those responsibilities,” says Ryan Peirce, systems engineer at Palo Alto Networks, “or even solve things that maybe the team hasn't thought about as a process ahead of time. And then being able to continually understand and iterate throughout.”

For example, AI-trained models can also be leveraged to catch silent infrastructure failures.

“AI agents can take on increasingly complex operational work that previously required human coordination and expertise, but, unlike traditional software, agents are not static systems,” says Mayank Agarwal, co-founder and CTO of Resolve AI and co-creator of OpenTelemetry. “Their behavior evolves as models are updated, tools are modified, and new data becomes available.”

But, that same adaptability that gives agents value creates its own challenge. AI is dramatically increasing the amount of software, he says, so that production environments are being changed, and we don't always know why agents did what they did. Organizations need ways to evaluate, monitor, and govern agentic behavior and ensure they have agentic explainability in place.

“As more engineers rely on AI-generated code, it becomes increasingly difficult for any individual or team to maintain a complete understanding of every workflow,” which makes troubleshooting exponentially more difficult. Agarwal continues that agents will increasingly be deployed to help “humans understand and operate systems whose complexity increasingly exceeds what can be managed manually.

This has agent observability emerging as the next evolution.

“While traditional observability is focused on understanding the health and performance of software systems, agent observability builds on that to understand the behavior, reasoning, and actions of autonomous systems,” he explains, along with comprehensive audit trails to pinpoint when the agent diverges from the expected behavior. “The goal is not simply to explain what happened after the fact, but to create the confidence required for organizations to safely delegate more responsibility to autonomous systems.”

Enterprise-grade guardrails must determine what these nondeterministic agents can and cannot do, with full audibility into how decisions are made and continuous evaluation frameworks to measure agent performance over time.

The organizations that succeed will be the ones that treat agents as operational systems that require ongoing oversight, measurement, and governance. As agents become more deeply embedded in critical business workflows, the ability to understand and govern their behavior will become just as important as the capabilities they provide.

Unsurprisingly, the first AI agents adopted in the enabling teams of operations and SRE are usually support agents, even internally used by platform teams, as a first line of operational support defense.

The next ops agent adopted tends to be in root cause analysis, which Hewlett Packard Enterprise’s Phanidhar Koganti argues can cut time to root cause at least by half.

At Availability, agents operate inside three layers:

- L1** Capability gates that define what an agent can touch.
- L2** Behavioral guardrails enforced at runtime.
- L3** Audit trails that capture intent, not just outcomes.

“That last one matters most. When something goes wrong, 'What did it do?' is recoverable from logs; 'What was it trying to do?' is not, unless you built for it,” Privat explains. “The payoff is counterintuitive: the teams with the strictest guardrails move fastest, because their engineers don't have to stop and ask permission. The rules are already in the rails. Governance done as process is a brake. Governance done as infrastructure is an accelerant.”

“Governance done as process is a brake.
Governance done as infrastructure is an accelerant.”

MICHAEL PRIVAT · CHIEF DATA & ENGINEERING OFFICER, AVAILITY

11 Agentic security

Perhaps the biggest promise of ROI on your AI comes from integrating it into security and DevSecOps. Gartner pegs the average cost of an outage at \$4,000 per minute.

\$4,000 the average cost of an outage,
per minute

Until now, security has remained very manual and tool centric. IDC's Reimagining IT Operating Models Survey finds that AI in vulnerability management already can realize the following benefits:

Where AI already pays off in vulnerability management



% OF ORGANIZATIONS REALIZING BENEFIT

Benefits organizations already realize from AI in vulnerability management.
Source: IDC, Reimagining IT Operating Models Survey.

Chainguard's whitepaper "Five habits to operationalize AI in the SDLC for engineering leaders" presents these key points regarding the ever more challenging threat landscape and the necessity of structural security:

FIVE HABITS OF STRUCTURAL SECURITY · CHAINGUARD

01 Negative mean time-to-exploit:

The rise of AI models like Mythos means that vulnerabilities are now being identified and weaponized before a Common Vulnerability and Exposure (CVE) is even published. As traditional reactive, patch-based security models collapse, the window for manual intervention disappears.

02 Agentic pipelines as an expanded attack surface:

As AI agents become active participants in the CI/CD pipeline, triggering deployments and updating dependencies, they also become prime targets for credential-harvesting malware. Guardrails must extend to the skills and workflows agents utilize, ensuring every action has verifiable provenance.

03 Compliance as a continuous output:

Compliance can no longer be a periodic, manual activity. SRE teams must integrate automated attestations — like software bill of materials (SBOMs) and supply chain levels for software artifacts (SLSA) provenance — directly into the build process as a byproduct of CI/CD.

04 Standardizing on a secure-by-default foundation:

Engineering leaders are encouraged to transition teams to dependencies and container images built from verified, attributable source code. This shifts the security posture from How fast can we detect threats? to a structural defense where specific categories of attacks are eliminated before they ever reach the production environment.

05 The GPU tax and migration efficiency:

Much like the operational challenges of managing GPU costs, cleaning up legacy technical debt is a major blocker to AI adoption. Instead of manual remediation, teams increasingly use AI tooling to automate the migration of legacy Dockerfiles and high-exposure images. In the face of rising costs, AI can also flag workloads that could be safely moved back on-premise.

AI-enabled workflows demand a new mix of skills and roles.

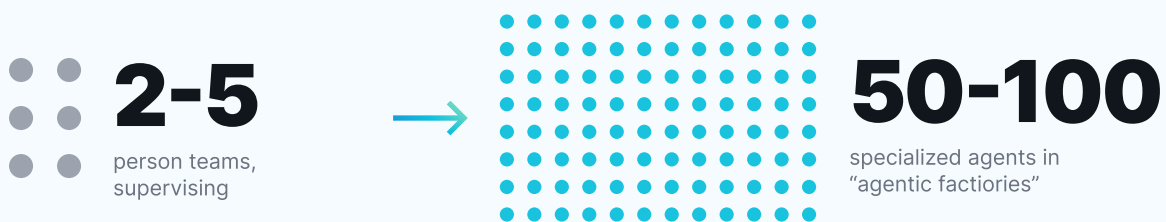
“This is not a one-time transformation,” explains IDC, “but an ongoing shift toward human-AI collaboration, requiring both technical expertise and a deep understanding of business architecture.”

The next stage of this agentic era will see enterprises shifting from deploying individual ops agent assistants that focus on single tasks, to end-to-end workflow automation via AI-first agentic systems.

This shift will soon have organizations reimagining their work and workflow as AI-first, McKinsey predicts, with humans and IT systems “selectively reintroduced in AI-native design.”

An increasingly popular use case is swarms of agents working together to reverse-engineer and modernize legacy systems. All while human operators coordinate and validate — because, at the speed of AI, there’s more software for humans to operate than ever.

The way we work will also continue to evolve, including around team size. The traditional two-pizza team will break off. McKinsey observes that some organizations already have two- to five-person teams supervising agentic factories of 50 to 100 specialized agents. Of course that can only happen with the right agent observability and fine telemetry in place.



And it requires proper context engineering to actively design and structure the information fed into these AI agents and language models. This includes architectural, historical, operational, and organizational data. Delivery Hero’s VP of tech foundations Rodrigue Schäfer gives the example of feeding agents not just on incident logs, but Slack channels and transcripts from bridge calls, to train on real insights into the human-led, team-based decision-making processes.

“Technology is also absolutely needed because the more production code we produce with agentic engineering in general, the more we will need to operate it in production. If we stick to the same quality level as before, it would just linearly scale the number of incidents,” Schäfer says. “I’m also very optimistic about agents that can help with triaging, root cause analysis, even executing runbooks in predefined scenarios.”

Enterprises also have to embrace agentic observability, he says, especially in consumer- or business-facing products.

“For example, a customer service agent that's answering customer service requests,” he continues, “we need to figure out, what are we looking at? How do we detect if the agent is drifting in some direction? If we have some sort of regression?”

13 Multi-agent observability

The true magic isn’t agents working on tasks in isolation. It’s whole agentic fleets collaborating toward a common goal. Of course, that makes it all more complex, but also likely more successful.

“The biggest mistake teams make with AI in DevOps is building one giant agent and asking it to do everything, starting from gathering logs, classifying the error, checking git, looking up tickets, deciding what to do, creating a fix, and notifying the human. It's expensive, it's fragile, and when it breaks you have no idea where it went wrong.” says Grace Torany, software engineer at Nvidia.

“In DevOps and cloud operations, that design breaks down quickly because incidents aren’t linear. A real investigation moves across metrics, logs, deploy history, runbooks, ownership data, and sometimes multiple regions or services.”

A single agent looks impressive, she warns, but as context and tool surface expands, it becomes harder to trust and debug multiple agents working together.

Instead, Torany offers the pattern of an agent train, composed of small, purpose-built agents with each in charge of exactly one responsibility. Instead of raw context dumps, she explains that each agent is encoded with plain-text mermaid diagrams of the structure of the system it’s working on, so “it knows the workflow before it reads a single log line. That's how you get AI to reason over complex systems without burning thousands of tokens on context that's mostly irrelevant.”

“The biggest mistake teams make with AI in DevOps is building one giant agent and asking it to do everything, starting from gathering logs, classifying the error, checking git, looking up tickets, deciding what to do, creating a fix, and notifying the human. It's expensive, it's fragile, and when it breaks you have no idea where it went wrong.”

GRACE TORANY · SOFTWARE ENGINEER, NVIDIA

The handoff between the agent train cars is a JSON brief, not leaving the previous agent's intent up for interpretation, she explains. “The train stops at every junction for a human review. The AI never runs ahead of you. That discipline is what separates a clever demo from something you'd trust in a production CI pipeline at 2am.”

Another mistake that Oracle's staff reliability engineer III Kinjal Vaishnav has found is teams treating these complex agentic systems like a singular, smart chatbot with more tools. She writes about this in her new academic paper “Why One Agent Is Never Enough: A Multi-Agent Architecture for Infrastructure Observability.”

When setting up observability of a multi-agent system, her advice echoes Torany's. Start with the workflow, not the agents. In cloud operations, that likely is:

THE INCIDENT WORKFLOW · MAP IT BEFORE YOU AGENT IT

- 01 An alert comes in.
- 02 Someone identifies the impacted service.
- 03 That person will check telemetry, review logs, and recent changes.
- 04 They form a hypothesis.
- 05 Decide whether to remediate or escalate.

“I map each of those steps to a bounded agent role instead of giving one agent every responsibility,” she explains. “In my observability work, that means a supervisor-style agent routes the alert, a telemetry agent gathers metrics and log findings, a reasoning agent compares the evidence against known failure patterns, and an action agent prepares the remediation recommendation.”

Then the most important setup decision becomes how agents share context. Her preference is shared state over free-form agent chat.

“Each agent reads and writes a defined object: alert metadata, telemetry findings, hypotheses, recommended actions, confidence, and audit trail. That makes the system much easier to debug because every handoff is explicit,” Vaishnav explains. “If the final recommendation is wrong, you can inspect whether the routing was bad, the telemetry was incomplete, the reasoning step overreached, or the action step proposed something unsafe.”

Finally, she recommends, for engineers building multi-agent systems in the cloud, start with reliability controls before autonomy.

“Each agent should have the smallest toolset it needs, with typed schemas and validation before anything runs,” Vaishnav says. “A telemetry agent should not have broad production write access. An action agent should not execute risky changes without approval gates.”

Because, maybe even more than human operators, ops agents need very clear guardrails and focus. Always remember, she recommends, that multi-agent systems are still distributed systems, which means they also require routing, contracts, failure handling, tracing and root-cause analysis, and human-approval paths.

“The agents may use LLMs, but the reliability comes from engineering discipline around them,” Vaishnav says. “For DevOps teams, the first production value of these systems is not replacing the on-call engineer. It is reducing the time humans spend gathering evidence.”

In fact, a good multi-agent system, should prepare the full context of an incident before a human operator even opens their laptop:

-  What changed.
-  What signals are abnormal.
-  What dependencies look healthy.
-  What prior incidents look similar.

Then, she continues, “the engineer can review a structured hypothesis instead of starting from a blank dashboard.”

PART 03/03

Measurements that matter

The scoreboard: the metrics that prove AI's impact on stability, security, and the bottom line — and the error budgets that keep agents honest.

14 Metrics for AI in DevOps

15 Error budgets as guardrails

Of course, you can't improve what you don't measure. Even without AI's complexity acceleration, 60% of organizations still struggle to effectively measure their impact on software velocity or stability. When budgets are tight, making sure your DevOps and operations teams know how to measure what they've built and enabled is essential.

60%

of organizations still struggle to effectively measure their impact on software velocity or stability — even before AI's complexity acceleration.

"Devs might feel faster with AI, but the hard data could paint a different picture," Ben Lloyd Pearson, director of developer experience at LinearB, said. "It really all comes down to how much toil AI reduces and how much time it frees up for developers to focus on more strategic work."

One of the biggest struggles for engineering teams of all sizes is measuring the return on investment of AI, which, in tighter economic times, will determine if your team gets budget. And for DevOps and platform engineering teams, that means distancing yourself from being seen as a bottleneck, by measurably impacting AI adoption at scale.

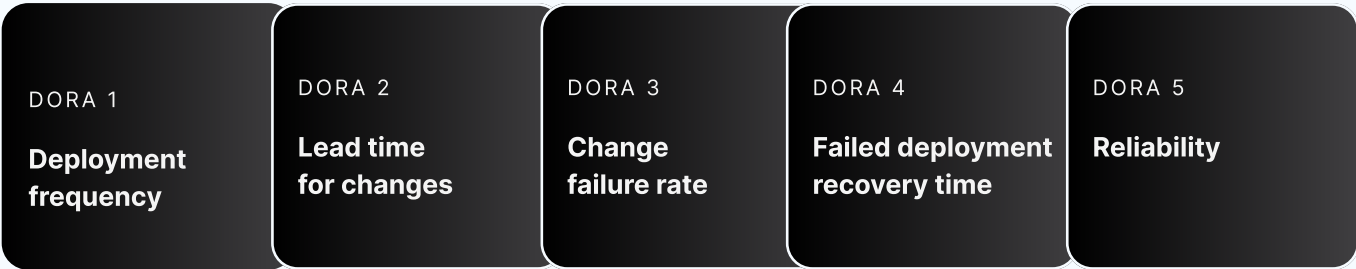
To do this, Laura Tacho, senior principal technologist of developer experience at AWS, says this comes down to first and foremost defining:

- 01 **What engineering performance means for your organization.**
- 02 **The specific metrics to measure the impact of AI on that performance.**

"You need to know what's working with AI, what's not working with AI, and how it's influencing some of those foundational measurements of performance in order to make the right choice of what to do next," Tacho says. "You also need to protect your org from long-term damage, not sacrificing long-term velocity, stability, and maintainability for short-term gains when it comes to AI."

She is the co-author of the [DX AI Measurement Framework](#), which balances AI utilization, AI impact, and AI cost metrics.

Everyone within a technical organization should not only know what DORA metrics are but know where they stand — at the team, division and organizational level — on the key five areas:



AI has developers trying to deploy more frequently but the sheer amount of code risks increasing lead time for change. Most crucially, generative AI and agentic AI should not be decreasing reliability nor increasing time to recover. But, for many companies, the impact of AI is problematic at best. According to DX, a major reason AI adoption becomes problematic is that existing organizational bottlenecks — like meetings, review delays, and continuous integration wait times — swamp individual coding gains.

Other metrics that are important for AI for DevOps — and DevOps in light of AI — include:

MTT-IDENTIFY	Meantime to identify or detect
MTT-CORRELATE	Meantime to correlate, which applies AI ops tooling to analyze logs, metrics, and traces, in order to connect disparate signals into a single, traceable, recommended action.
MTT-INNOCENCE	Meantime to innocence (because it's really not always the favorite culprits of network or storage.)

All these metrics must be measured across the whole stack.

In a full-stack environment, symptoms and downtime rarely occur in the same layer as the cause, which means a symptom of a failure and the cause of a failure are rarely in the same layer, especially at the enterprise scale of complexity. An impact of any change, including one made by an agent, must consider all changes in:



Add to this, enterprises often see multiple failures occurring in parallel, which requires tighter integrations across the full stack, along with AI to filter the important signals from the noise.

The next step will likely be AI agents auto-remediating when appropriate — but only if its actions are explainable to its human operator.

Recent research out of Drexel University finds that, while agentic author pull requests perform well at code changes and documentation, they so far aren't meeting standards for bug fixes and performance. One of the reasons for this is that not-merged pull request sizes tend to involve larger code changes, touch more files, and do not pass the project's CI/CD pipeline validation.

Do not jump to agents acting autonomously in your CI/CD pipeline before you prove them in sandboxed environments. And then continuously monitor and observe. Every member of both dev and ops teams has to become fluent in reliability economics, constantly aware of the cost-benefit tradeoff between autonomous system complexity and the financial and/or reputational costs of AI failure. Before moving agents from prototypes and sandboxes to production, teams must understand the reality of the cost of agent errors against the cost of implementing guardrails and evaluations.

Every engineer must also know how to monitor the confidence metrics of any AI agents they are deploying. These help quantify certainty in data and/or algorithmic outputs. In AI-driven DevOps, AI confidence metrics include:

- | | |
|----|---|
| c1 | Token-level log probabilities: How does an individual aspect of an answer match the preceding context? |
| c2 | System-level retrieval-augmented generation (RAG) scores: How's your context precision and answer relevance? |
| c3 | Semantic cache similarity: Does your AI query closely match historical tasks? |

“Confidence scores are not the same as operational safety; a model saying it is 90% confident does not mean the action is safe to run against a live service.”

KINJAL VAISHNAV · STAFF RELIABILITY ENGINEER III, ORACLE

“Confidence scores are not the same as operational safety; a model saying it is 90% confident does not mean the action is safe to run against a live service.” Vaishnav advises to “invest in the tool layer before obsessing over prompts. In cloud environments, that means least-privilege IAM [identity and access management], region-aware tooling, retries, timeouts, idempotent actions, rollback paths, audit trails, and observability for the agents themselves.”

15 Error budgets as guardrails

Autonomous agents are blatantly faster than human developers and operators. But that means they can introduce higher risks faster too.

Error budgets are no longer just important to establish the acceptable amount of system reliability per service. Agents can now query live service-level objectives (SLO) data to safely control rollouts, rollbacks, and incident remediation, without exciting reliability thresholds.

By integrating agentic workflows with error budget policies, you create a calibrated system of guardrails, which includes:

PAUSE & ESCALATE

Pre-action SRE gates:

Before an agent autonomously executes a state-changing action — like restarting a service or deploying a hot-fix — it queries the error budget headroom. If the error budget is exhausted or burning too quickly, the agent pauses and escalates to a human instead.

AUTO-REVERT

Intelligent rollbacks:

Agents continuously monitor metrics. If a canary deployment spikes error rates and drives the error budget burn rate beyond acceptable levels, the agent automatically triggers a rollback and files an incident ticket.

WITHIN BUDGET

Autonomous remediation:

If the budget is within limits, the agent can test hypotheses, parse logs to find the root cause of 5xx errors, and apply fixes autonomously.

Of course all of these actions are logged and fed back into the overall agentic context setting. These can be both reliability error budgets, and, with the cost of AI rising, enterprises should also set and enforce compute and token budgets too.

DevEx metrics have you doing a similar tradeoff, between flow state, feedback loops, and cognitive load, which are valuable agentic DevOps metrics to consider as well, across both the developer and the operational experience.

For now, it's important to position the human operator as the head of any agentic command center, so don't forget to measure the impact of agentic ops adoption on your DevOps teams. All AI adoption should center on how it can help humans, not the other way around. Good luck!

Upskilling for agentic DevOps

Every skill this guide names — CI/CD guardrails, observability, automated remediation, context engineering, multi-agent architecture — is trainable. KodeKloud teaches them the way engineers actually learn: hands-on, in real browser-based environments, with AI-assisted guidance.

The AI roadmap for DevOps engineers

A step-by-step path built from the KodeKloud AI Labs — from fundamentals to production-grade agentic systems. No data science degree required.

STEP 1-3

Foundations

AI fundamentals · Prompt & Context engineering · Model Context Protocol (MCP)

STEP 4-6

Agents & Context

AI agents for Ops tasks · LangChain & LangGraph Orchestration · RAG on your own docs, configs & logs

STEP 7-9

Production Systems

Workflow automation (n8n) · AI playgrounds · MLOps & agent memory at scale

One platform, the whole wishlist

The operational fundamentals from Part 2 — pipelines, Kubernetes, observability, security — covered by role-based learning paths and certification tracks.

HANDS-ON LABS

Every course runs in real, browser-based environments with instant validation — Docker, Kubernetes, Jenkins, Terraform, Prometheus, Grafana, and Python for automated remediation. No setup.

AI-POWERED LEARNING

An AI Assistant inside every lab, plus a personalized AI Tutor that builds adaptive, hands-on learning paths for any DevOps or cloud topic — in any language.

PLAYGROUNDS

60+ DevOps playgrounds plus cloud and AI sandboxes — preconfigured environments for the safe experimentation this guide keeps recommending before production autonomy.

KODEKLOUD ENGINEER

A production-like simulation platform: real scenario tickets — broken pipelines, failing clusters, misconfigured deployments — with automated scoring. An internship built into the browser.

Built for teams

- Enterprise dashboard & team progress reporting
- License rotation & centralized billing
- Guided onboarding & dedicated account support
- A dedicated DevOps team learning consultant

90%

of companies worldwide will face a shortage of skilled tech professionals by 2026, according to IDC. The teams that up skill from within won't.



KodeKloud

Build the team that ships safely at AI speed.

Hands-on DevOps, cloud, and AI training for your whole engineering organization — with the labs, playgrounds, and AI tutors behind this guide.

www.kodekloud.com/business

Team Demos · Enterprise Plans · Progress Reporting For Engineering Leaders

The Definitive Guide to AI for DevOps / Agentic DevOps · By Jennifer Riggins

© 2026 KodeKloud · Hands-on by default. Fun on purpose.