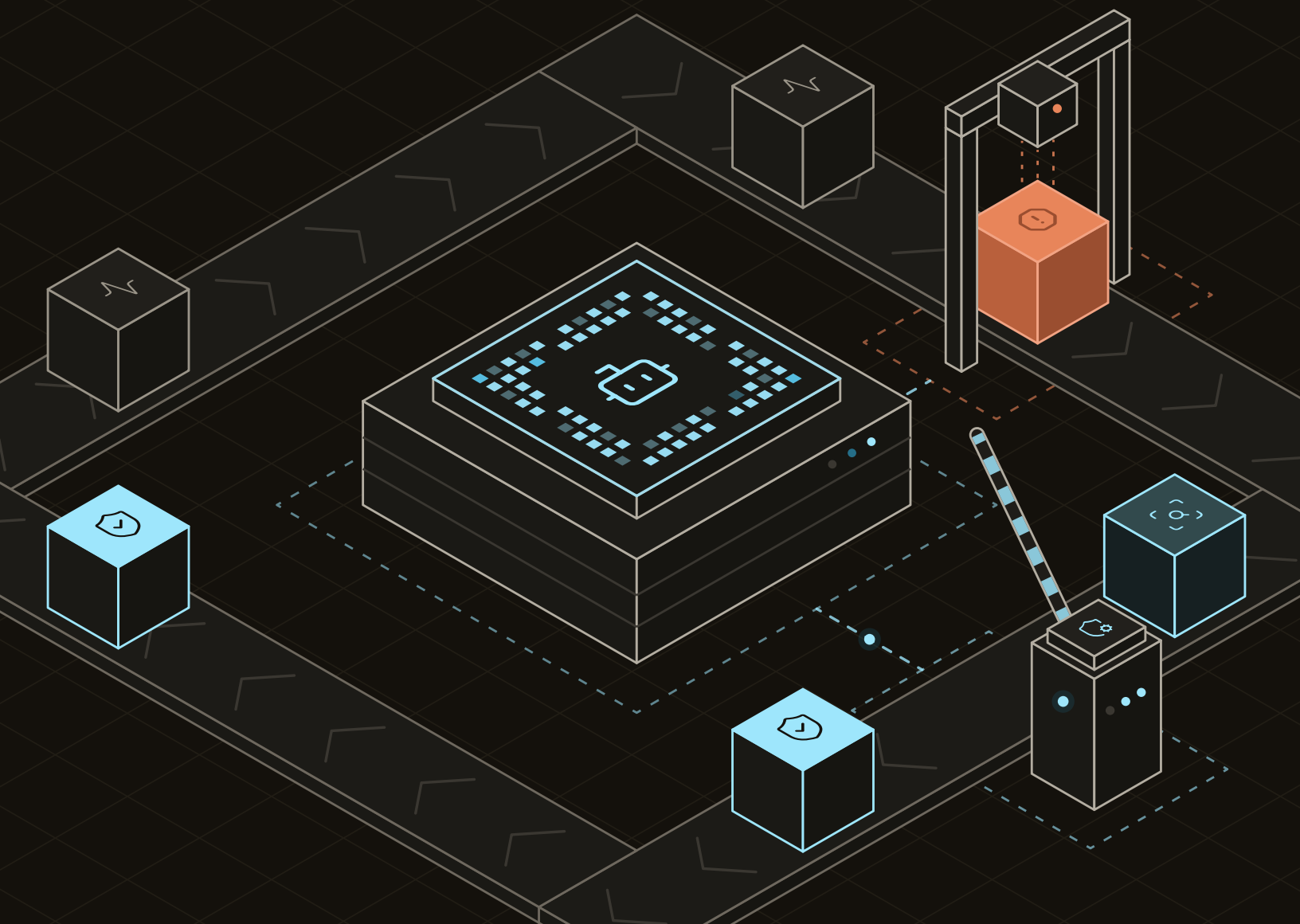
StackGen

E-BOOK

# The AI SRE Buyer's Guide

AI SRE that closes the loop: from root cause to governed remediation, in one system - 2026 Edition



# Table of Contents


Executive Summary 2

1. Why AI SRE, Why Now 3

2. The Three-Stage Operating Model 5

3. Investigation Discipline: Ruling Out Before Ruling In 7

4. Why Agents Need a Knowledge Graph, Not Just RAG 8

5. Governance: What Makes Remediation Safe to Automate 11

6. How to Evaluate AI SRE Platforms 13

7. Making the Decision 14

In Production at Our Customers, and at StackGen 15

Appendix: Evaluation Checklist 16

Sources 17

# Executive Summary

## Why this guide exists

### StackGen's thesis

A root-cause explanation is half a fix. The loop closes when that explanation becomes an approved, audited action inside one system.

A typical SRE team gets around 400 alerts a day. Maybe 10 are actionable. The rest still gets opened, evaluated, and dismissed one at a time. Toil consumes more than 60% of most SRE workloads before anyone touches a real problem. [1]

Much of the AI SRE market has focused on investigation: reasoning across telemetry to find and explain the root cause. That's genuine, hard-won progress. But per StackGen's State of Reliability 2026 report, 48% of incidents resolve through an infrastructure or deployment action, a rollback, a config revert, a scaling change, not a conversation. [1]

This guide lays out what closing that loop requires: an operating model that carries context from infrastructure inventory through to remediation, reasoning grounded in a live knowledge graph rather than similarity search alone, and governance enforced at runtime rather than layered on after the fact. It draws on StackGen's State of Reliability 2026 report, 178,000+ production incidents across 360+ companies, and how Aiden for SRE runs in production today.

**400/10** [1]

Alerts a day vs. how many are typically actionable

**60%** [1]

Of SRE workload is toil, not incident judgment

**$18.95B** [2,3]

Size of the AIOps market in 2026

# 1. Why AI SRE, Why Now

Every new microservice adds fifteen to twenty new alert rules. Almost nobody removes the old ones. The result is dashboards nobody fully trusts, on-call rotations that burn people out, and root cause analysis that turns into a multi-day archaeology project across a half-dozen disconnected tools.

This is an engineering gap, not a staffing one: reasoning across heterogeneous telemetry, rather than pattern-matching against it, only became viable at production quality recently. That's the gap AI SRE exists to close.

## The math finally forces the issue

The 2024 DORA State of DevOps Report found that elite-performing teams recover from incidents roughly 7,200 times faster than low performers. [3] Headcount doesn't close that gap; a learning system that carries context and applies intelligence at the point of failure does.

AI-powered monitoring adoption moved from 42% to 54% between 2024 and 2025 alone. [2] That's not a future curve, it's already happening, which is what makes this a now decision rather than a later one.

> ON-THE-GROUND QUOTE
>
> “My SREs spend more time fighting alert noise than fighting actual incidents. We get 400 alerts a day. Maybe 10 are actionable. The rest are just training the team to ignore everything.”
>
> - SRE Manager, Series C SaaS company, 200-microservice environment [2]

## What breaks, and how often

StackGen's State of Reliability 2026 report groups 178,000+ status page incidents and 1,037 engineering post-mortems across 360+ companies into 16 root causes across 7 themes: [1]

| THEME | SHARE | WHAT IT MEANS FOR TOOLING |
|---|---|---|
| Change (code + config) | 26% | Highest automation value: closest to the deploy pipeline and IaC layer |
| External / vendor dependency | 21% | Investigation alone can't prevent it; needs degradation playbooks |
| Data pipeline | 18% | Chronically under-monitored relative to app-layer observability |
| Infrastructure (network/DNS, hardware) | 13% | Requires infra-layer remediation, not just an alert summary |
| Auth & security | 10% | Cert expiry and federation dependencies are a quiet, recurring killer |
| Capacity & quota | 9% | Needs autoscaling or quota remediation, not just detection |
| AI / model & process | 2% | Small today, growing fast as agentic systems enter production |

Source: StackGen State of Reliability 2026 report.

### The pattern worth noticing

More than half of all incidents (26% change + 13% infrastructure + 9% capacity = 48%) resolve through an infrastructure or deployment action: a rollback, a scaling change, a config revert, or a quota adjustment. [1]

The Cloudflare outage in November 2025 traced to a single ClickHouse permissions change, a config-layer root cause. The AWS us-east-1 incident in October 2025 generated 137 downstream incidents in 24 hours, all tracing to a single upstream vendor dependency. [1]

**48%** [1]

Of incidents resolve via infrastructure/ deployment action

**137** [1]

Downstream incidents from one AWS us-east-1 outage, Oct 2025

### Know the ceiling

Per Lightrun's State of AI-Powered Engineering 2026 report, 44% of AI SRE failures in production incidents stem from missing execution-level data that was never captured in the first place. [4]

The industry has made real progress on the investigation side. What follows is the operating model, the reasoning layer, and the governance that carry a finding through to a fix you can trust.

## 2. The Three-Stage Operating Model

AI SRE platforms map to a consistent operational pattern that large infrastructure environments share, regardless of tooling: three stages. [5] The friction between them is where MTTR climbs, and engineers burn out, and closing that friction all the way through to a remediation step, not just a finding, is what the three-stage model is designed to do.

**STAGE 1**

### Infrastructure Inventory Review

Understanding what assets exist, how they relate, and which configuration items are relevant to an active incident. The answers usually already exist in a CMDB like ServiceNow. The problem here is access speed, not data availability.

**STAGE 2**

### Cost, Utilization, and Provisioning Analysis

Identifying over-provisioned, at-risk, or underutilized infrastructure, and surfacing capacity and provisioning decisions before they become incidents. Tools like Grafana provide visibility, but turning utilization data into a provisioning or cost recommendation usually depends on the one or two engineers on a team who know the stack well enough.

**STAGE 3**

### Team Incident Investigation, Through to Remediation

Triaging alerts, investigating root causes, and handing off context between engineers without losing information, and where a fix is approved, carrying that finding into an actual infrastructure change rather than ending at a summary. This is where most of an SRE team's time actually goes, and where a platform's value compounds if it can close the loop.

### Why the stages have to be connected

When an alert reaches Stage 3, having Stage 1 and Stage 2 context already attached is what separates a shared query tool from a shared investigation platform. [5]

## Alert triage is the core of Stage 3

SRE teams in enterprise environments routinely report MTTRs of four hours or more. [5]

DEDUPLICATION AND ENRICHMENT HAPPEN BEFORE TRIAGE




Triage categories only work if what reaches them is already reduced to distinct, correlated incidents, not 400 raw alerts. Before an alert lands in Act Now, Work Today, False Positive, or Needs Review, it survives two steps:

- **Deduplication:** collapsing the dozen alerts that all fired from the same underlying event into one incident.
- **Enrichment:** attaching the infrastructure and deployment context that turns "CPU alert on host X" into "CPU alert tied to the deploy that shipped 40 minutes ago."

Teams running this in production report a 30–50% reduction in alert-to-page volume from deduplication alone. [11]

| TRIAGE CATEGORY | WHAT IT MEANS |
|---|---|
| Act Now | Critical, immediate response required |
| Work Today | Important, address within the business day |
| False Positive | Evaluated as noise, no action needed |
| Needs Review | Ambiguous, flagged for human judgment |

# 3. Investigation Discipline: Ruling Out Before Ruling In

The core idea worth borrowing is that most AI RCA failures isn't a knowledge problem but a sequencing problem. The model gets the first plausible story, usually "must be the recent deploy," and writes an RCA-shaped paragraph before it's actually ruled anything out.

A longer prompt doesn't fix this. What fixes it is forcing the investigation through stages in a fixed order, and refusing to let the narrative get ahead of what the evidence actually supports.

Five steps, done in sequence:

- Pin down what broke and for whom, on which signal, and when it started.
- Rule out the cheap stuff first, with small falsifying queries, so the investigation isn't wasted disproving something obvious later.
- Keep two or three competing theories alive at once instead of collapsing to one story too early.
- Separate coincidence from correlation from mechanism when grading a surviving claim, since only a mechanism actually explains why a fix would work.
- Be willing to end on "we don't know yet, here's what to check next" instead of a confident wrong answer.

This maps onto two failure modes worth naming directly.

| | |
|---|---|
| Same alert, different root cause | A CPU spike that's sometimes a bad deploy, sometimes a noisy neighbor, sometimes a cert expiring and triggering retry storms, gets handled by nailing down identity and onset, which specific entity actually breached, and exactly when, before the change timeline is even consulted. The common mistake runs backwards: diving into deploy history and telemetry fan-out before establishing what actually failed. |
| Same root cause, different alerts | A stale downstream cache quietly producing five unrelated-looking symptoms is the flip side. It requires keeping candidate mechanisms alive across the graph instead of closing the investigation the moment the first symptom clears up. |

An empty branch, "checked this, found nothing," is a successful step, not a wasted one. That's what stops an agent from inventing a plausible-sounding cause just because the template expects one.

# 4. Why Agents Need a Knowledge Graph, Not Just RAG

The alert fires at 2:07 AM. An on-call engineer with eight months of tenure doesn't have the institutional knowledge that this failure pattern is almost always caused by a downstream cache going stale, because that only happened once before, eighteen months ago. Replace the engineer with an AI agent, and the problem doesn't go away unless the agent has a knowledge graph to reason with. [6]

## Knowledge graph, service graph, or RAG: not the same thing

A knowledge graph is a structured representation of entities and the typed, directional relationships between them, built for multi-hop reasoning. A service graph maps runtime interactions observed from telemetry: excellent for "what's talking to what right now," but limited to observed traffic. A RAG system retrieves text chunks by semantic similarity, useful for documentation, but it doesn't understand that service A depends on database B, only that both are mentioned nearby. A knowledge graph is deterministic; a RAG retrieval is probabilistic. [6]

## Two failure modes without one

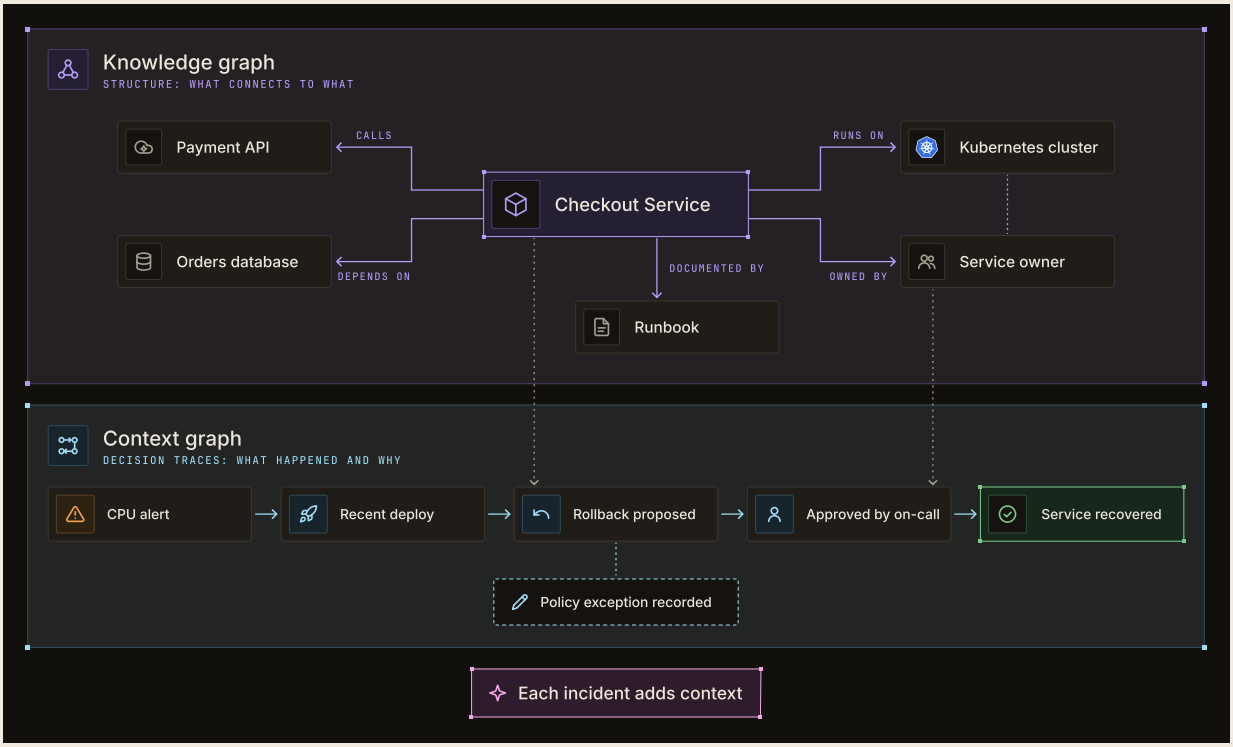
Hallucinated confidence: the agent retrieves loosely relevant documents and synthesizes an authoritative-sounding, confidently wrong root cause. Paralysis by uncertainty: a more carefully guardrailed agent escalates to a human every time, adding zero value over the pager it was meant to replace. [6]

### What a DevOps Knowledge Graph Models

Five entity categories connected by typed relationships: infrastructure, software delivery, operational, human/organizational, and policy/knowledge. The power is in the edges: a failing service connects to its deploying pipeline, the team that owns it, the SLO it's violating, and past incidents with similar characteristics. [6] A graph built by the same system that provisions and governs the infrastructure has a head start here, since it doesn't have to re-infer topology from telemetry alone, which matters most in the moment a finding needs to become a change.

## Context graphs: the decision layer on top

A knowledge graph captures structure, what connects to what. A context graph captures decision traces: what happened in this specific case, which policy exception was granted, who approved it, and why. That's what turns a static graph into something that compounds, each incident adds a trace, and the system gets better at prediction the more incidents it handles. [6]




## The toil math

A 2025 SolarWinds report found that AI-assisted incident response saves an average of 4.87 hours per incident, with leading implementations achieving MTTR reductions of 30 to 70%. [7]

Nearly 70% of SREs report on-call stress as a direct cause of burnout, per the Catchpoint SRE Report 2025. [8]

**4.87 hrs** [7]

Avg. time saved per incident with AI-assisted response

**30–70%** [7]

MTTR reduction range for leading implementations

**70%** [8]

Of SREs cite on-call stress as a direct cause of burnout

### An honest caveat

AI has shifted toil, not always eliminated it. The Catchpoint 2025 SRE Report recorded that toil actually rose to 30% in 2025, the first increase in five years, as teams wrestled with model tuning and "AI babysitting." [8]

## Always On Agents

Most AI agents today work in a single sitting: you give a goal, the agent calls some tools, and it either finishes or the context window runs out, and the session ends. An always-on agent is different; it can keep making progress on a goal across multiple sessions, possibly over hours, days, or weeks, by persisting its state outside any single conversation and picking up where it left off rather than starting cold each time. Three things typically define it:

| | |
|---|---|
| PERSISTENCE | The agent's state (what it's done, what it's found, what's still pending) survives beyond one session, stored somewhere durable rather than living only in the context window. |
| RECOVERY | If a session ends or fails partway through, the next session can resume rather than restart from scratch. |
| IDENTITY ACROSS TIME | The agent behaves like it has continuity, "I already checked X" rather than re-discovering X every time it wakes up. |

For AI SRE specifically, this matters in two situations that look different on the surface but come down to the same problem: the agent needs more room than a single session gives it.

- **The first is temporal:** an incident spans a shift change, or a slow-burning issue takes hours to fully diagnose.
- **The second is volumetric:** the investigation itself generates more than a context window can hold, including sprawling log dumps, long trace chains, a growing list of ruled-out hypotheses, and dozens of tool calls against metrics and dashboards.

In both cases, without persistence, the agent either restarts fresh and has to reconstruct everything a previous session already figured out, or it hits the context ceiling mid-investigation and starts silently dropping earlier findings. With it, the agent externalizes what it's learned, including findings, ruled-out causes, and pending checks, so a new session (or a fresh context window within the same session) can pick the investigation back up where it left off, the way a human on-call engineer hands off context in a standup, except automatically and regardless of whether the trigger was a clock or a payload size.

# 5. Governance: What Makes Remediation Safe to Automate

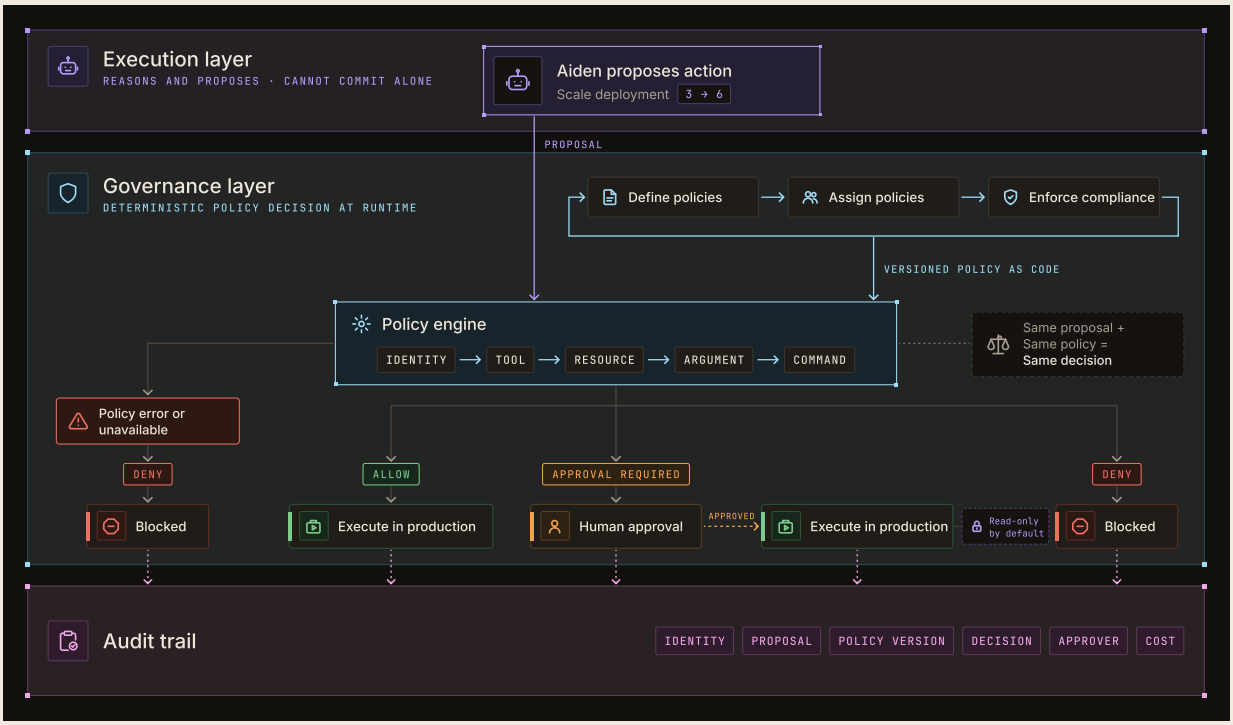
Carrying a finding through to an automated fix only works if the platform can be trusted with the action, not just the analysis. Can you trust AI for SRE in a regulated environment? Yes, but only if governance is enforced at runtime, not bolted on afterward. An agent operating on a regulated production infrastructure has to be architecturally incapable of taking an action that violates policy, a change window, or an approval rule. [9]

## Five capabilities, together, at runtime

- Policy enforcement at execution time: a governance layer sits between intent and execution.
- Complete, queryable audit trails tied to identity: reconstructable months later.
- Human approval workflows built into the runtime, not a Slack-bot approximation.
- Cost visibility per action.
- Security guardrails enforced by default, not granted by default: production integrations start read-only, and destructive actions (deleting a pod, scaling a deployment, modifying a config or secret) require explicit write permissions from an admin before an agent can take them. [12]

## Execution and governance as separate layers

The execution layer proposes changes and runs them against live infrastructure; it doesn't get to commit anything on its own. A separate governance layer deterministically evaluates every proposal against policy before anything touches production; the same proposal against the same policy always produces the same decision. [9]



## Policy checks run at five levels

Policy-as-code enforcement is a real runtime decision point, not a review step bolted on after the fact. The pattern showing up across agentic tooling right now is a dedicated policy engine sitting in front of every tool call, so the agent proposes an action and something else decides whether it runs. The model doesn't get a vote here. It can reason and suggest, but a separate engine evaluates the call before it touches anything, and answers only allow or deny.

Where this gets applied matters more than whether it's applied. It isn't just "can this agent act at all," it's granular down to five checkpoints, each its own decision:

- **Identity:** is this agent, acting as this role, permitted to act at all?
- **Tool:** is this specific call in scope for that role?
- **Resource:** is the target one this role can touch?
- **Argument:** a policy can inspect what's actually in the tool call and reject it if the values exceed a limit.
- **Command:** is the specific command being issued one that's permitted, even if the broader tool call would otherwise be allowed?

In one documented case, a command-level check caught the actual command being issued and blocked it, even though the agent's broader tool access technically allowed calling that tool in the first place. That's the granularity a policy layer needs.

Trust and policy intersect in a way that's easy to get wrong. Trust should change which policy tier applies, not remove the check. And the system needs to fail closed: if the policy engine is unreachable, or evaluation errors out, the default has to be deny.

This is what lets policy scale over time. Policies live as versioned code, GitOps, not tribal knowledge in someone's head. They get reviewed, tested, and rolled back through the same git workflow as everything else, and teams can own their own policy domains and ship changes without redeploying the agent itself.

### A real result worth knowing

In a 2026 side-by-side test with a large regulated bank, a leading general-purpose coding agent operating directly on the bank's infrastructure skipped a permissions check during an infrastructure change. The same agent, operating through a runtime governance layer, had every action run inside the policy boundary with the full decision trail captured. [9]

# 6. How to Evaluate AI SRE Platforms

One distinction predicts almost everything else you'll need to budget for and own: after the platform finds the root cause, does the fix execute inside this platform, or does it hand off to a separate system you maintain? Neither answer disqualifies a vendor, it just changes what else has to be planned for.

Three broad categories of tooling show up in nearly every enterprise evaluation. [2]

| | |
|---|---|
|  **Observability platforms with an AI layer** | Datadog and Dynatrace lead with deep telemetry collection, and layer AI SRE capability on top of that native data. Best if you're already heavily invested in that ecosystem. |
|  **Incident management platforms with AI** | Tools like Rootly and PagerDuty own the incident and resolution-history layer. Best where the bottleneck is coordination and knowledge retention, not investigation depth. |
|  **Standalone AI SRE platforms** | Tools like Komodor's Klaudia and Lightrun take an integration-first approach, working across an existing observability stack without requiring full data ingestion into a proprietary lake. [2] |

## Five buying criteria

- **Investigation depth:** Does the platform validate against actual execution behavior, or only correlate pre-captured telemetry? The second is faster to demo; the first is what holds up on a root cause you haven't seen before.
- **Automation maturity:** Where does it actually sit between "suggests a fix" and "executes autonomously," and what has to be true (approvals, guardrails, track record) before it moves further along that spectrum?
- **Integration breadth and setup reality:** Not the advertised integration list, the real setup time reported by the last comparable customer, since that's what your own timeline will look like.
- **Governance and auditability:** Can every action be reconstructed months later, with who approved it and why?
- **Total cost of ownership:** Modeled 12 months out at your growth rate, not priced against current incident volume, since that's the number that changes underneath a contract.

# 7. Making the Decision

Quantify the problem in a number your finance team recognizes before you quantify the solution: downtime cost, engineering hours lost to war rooms, and on-call attrition.

## The trust gradient, not a leap

The safe, effective adoption path starts with AI-assisted investigation, expands to human-approved remediation, and only later moves to bounded autonomous execution, once outcomes and audit trails have earned that trust. [2]

## Ask every vendor these questions

- Can it investigate a real incident from our own production data, not a synthetic demo?
- How does it handle a triggering signal never captured in pre-existing telemetry?
- At what specific point does an AI-proposed action require human approval?
- What does the complete audit trail look like for an AI-executed remediation?
- Once a root cause is found, what actually carries out the fix, and does that live inside this platform or in a separate system we'd own?
- How does pricing scale with incident volume over the next 12 months, at our growth rate, not current volume?

**70%** [2]

Alert noise reduction reported by Aiden customers

**45%** [2]

Tooling cost reduction vs. a separate APM + AI SRE stack

**4 min** [2]

MTTR on an OOMKill cascade vs. a 2.5-hour prior average

ABOUT

## In Production at Our Customers, and at StackGen

Aiden for SRE runs in production today at customers, including GreytHR and Corcentric. [10] It also runs on StackGen's own infrastructure: a multi-cloud Kubernetes estate spanning AWS, GCP, and Azure, supporting a growing enterprise customer base that includes SAP NS2, Chamberlain, and Piramal, plus three internal development clusters. Counting those internal clusters, StackGen's own SRE team monitors more environments than its entire customer tenant base. [13]

StackGen doesn't sell Aiden to SRE teams and then run its own production on spreadsheets and tribal knowledge. It's the same platform, on StackGen's own environments, with the same consequences when something breaks. [13]

Since adopting Aiden internally, that team has cut RCA time from 10–15 minutes down to under 2 minutes on a typical alert. In one instance, Aiden surfaced a capacity crisis building in a persistent volume claim two hours before traditional alerting would have fired at all, catching it early enough to resolve before any customer impact. [13]

> “We didn't respond faster. We didn't respond at all, because Aiden found the problem before it became one.”
>
> - Vishwajeet Kumar, Team Lead, StackGen DevOps/SRE [13]

RESULTS

**80%**
Faster RCA
10+ min → under 2 min

**2 hrs**
Earlier Warning
Capacity crisis caught before alerting would trigger

**Zero**
Separate RCA Docs
Shared Chat = built-in audit trail

**4**
SRE Team Size
Managing multi-cloud + growing enterprise customer base

APPENDIX

# Evaluation Checklist

Use this during vendor trials, StackGen included.

- [ ] Can the tool investigate a real incident from our own production data, not a synthetic demo environment?
- [ ] How does it handle incidents where the triggering signal was never captured in pre-existing telemetry?
- [ ] What is the model for human oversight, at what specific point does an AI action require approval?
- [ ] What does the complete audit trail look like for an AI-executed remediation in production?
- [ ] How does pricing scale with incident volume over 12 months, modeled against our current growth rate?
- [ ] What integrations are required for our stack, and what was the actual setup time for the last comparable customer?
- [ ] How does the platform capture and resurface institutional knowledge for novel incident types?
- [ ] What happens when the AI service itself goes down: what is the vendor's SLA for the AI layer?
- [ ] Can a junior on-call engineer use this tool effectively for an incident type they have never seen before?
- [ ] Can you prove, in writing, months later, what an agent did and why it was allowed to do it?

REFERENCE

# Sources

Every numbered citation in this guide maps to one of the sources below. Sources 1, 2, 5, 6, 9, 10, 11, 12, and 13 are StackGen's own published research and product data. Sources 3, 4, 7, and 8 are independent third-party research.

## STACKGEN-OWNED SOURCES


[1] StackGen, "The Root Causes Behind 178,000 SRE Incidents: What the Data Shows," Jun 2026.
stackgen.com/blog/sre-root-cause-taxonomy-online-services

[2] StackGen, "The Complete Buyer's Guide to AI SRE Tools in 2026," Apr 2026.
stackgen.com/blog/ai-sre-tools-buyers-guide-2026

[5] StackGen, "How AI SRE Works: A Three-Stage Workflow for Enterprise Infrastructure," Jun 2026.
stackgen.com/blog/how-ai-sre-works-a-three-stage-workflow-for-enterprise-infrastructure

[6] StackGen, "Knowledge Graphs: The Missing Brain Your SRE Agents Desperately Need" (Sanjeev Sharma), May 2026.
stackgen.com/blog/knowledge-graphs-the-missing-brain-your-sre-agents-desperately-need

[9] StackGen, "Can You Trust AI for SRE in Regulated Industries?," Jul 2026.
stackgen.com/blog/can-you-trust-ai-sres-in-regulated-industries

[10] StackGen, "Aiden for SRE Community Edition Is Here: Kill Your 90-Minute War Room for Free," Jun 2026 (customer references: GreytHR, Corcentric).
stackgen.com/blog/aiden-for-sre-community-edition-is-here-kill-your-90-minute-war-room-for-free

[11] StackGen, "How to Remediate Infrastructure Issues with AI SREs," 2026.
stackgen.com/blog/how-to-remediate-infrastructure-issues-with-ai-sres

[12] StackGen, "Aiden for SRE: Enterprise Solution for Multi-Cloud Teams," Jul 2026.
stackgen.com/blog/aiden-for-sre-enterprise-solution-for-multi-cloud-teams

[13] StackGen, "How StackGen's Own SRE Team Runs Aiden on Production" (case study, featuring Vishwajeet Kumar).
stackgen.com/case-studies/how-stackgens-own-sre-team-runs-aiden-on-production

## INDEPENDENT THIRD-PARTY RESEARCH (AS CITED IN THE STACKGEN POSTS ABOVE)


[3] DORA (DevOps Research and Assessment), 2024 State of DevOps Report: elite-vs-low-performer recovery speed. Cited in source [2].
cloud.google.com/devops/state-of-devops

[4] Lightrun, State of AI-Powered Engineering 2026 report: missing execution-level data statistic. Cited in source [2].
lightrun.com/ebooks/state-of-ai-powered-engineering-2026

[7] SolarWinds, 2025 report on AI-assisted incident response: time saved and MTTR reduction range. Cited in source [6].
solarwinds.com/company/newsroom/press-releases/state-of-itsm-2025

[8] Catchpoint, SRE Report 2025: on-call burnout and toil trend statistics. Cited in source [6].
catchpoint.com/asset/2025-sre-report


## ALSO REFERENCED WITHOUT A SPECIFIC STATISTIC:


[n/a] Mordor Intelligence, AIOps Market size and growth forecast (2026–2031). Cited in source [2].
mordorintelligence.com