StackGen

SEPTEMBER 2026

WHITE PAPER

The Autonomous Operations Factory

The operating model that completes the AI software factory

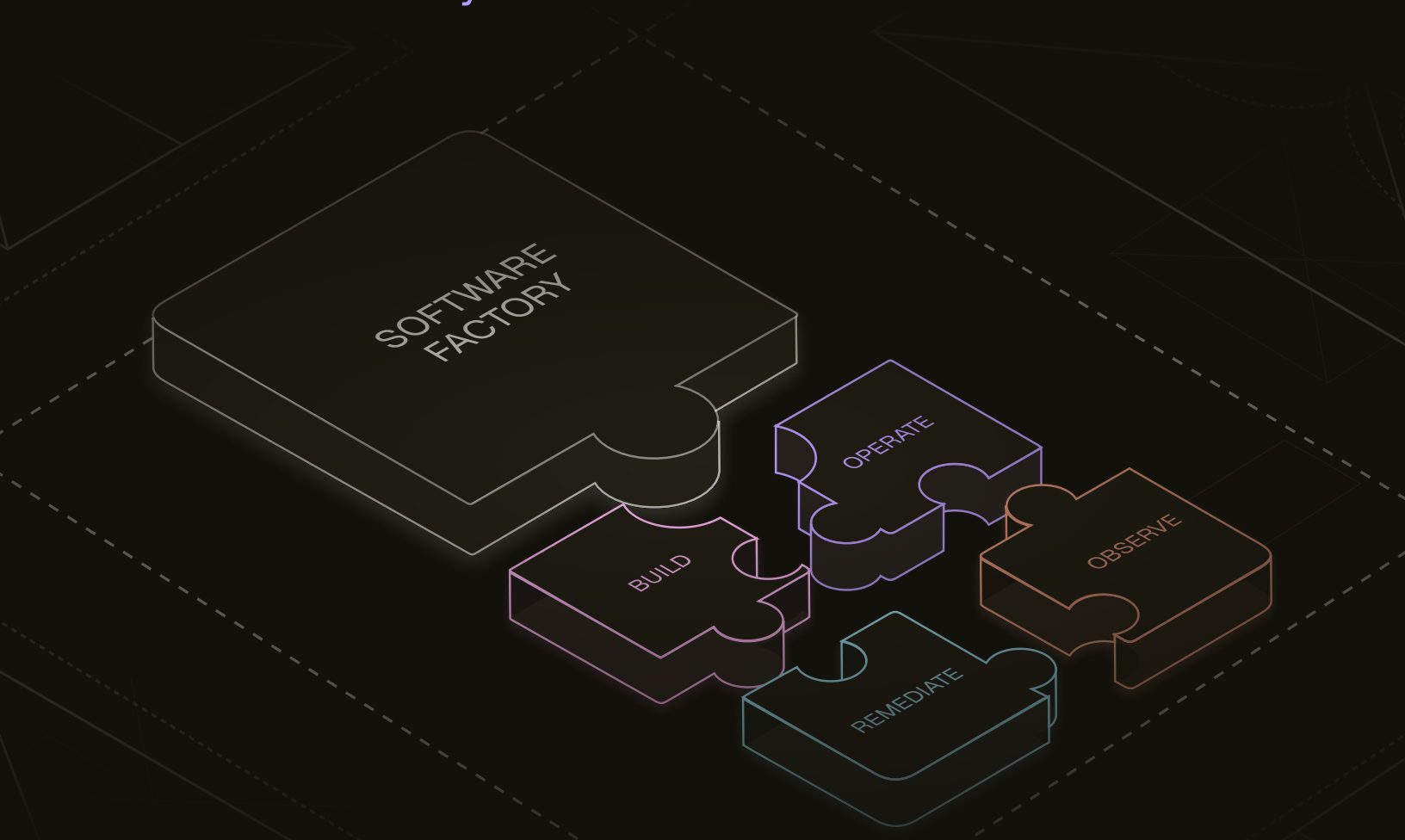


Table of Contents

1. Why an operations factory is needed 3

2. Why more agents is not the answer 5

3. What an Autonomous Operations Factory is 6

4. How it works 9

5. Where to start 14

6. How StackGen is building the Autonomous Operations Factory 16

7. Conclusion 17

Executive summary

AI has solved half of the software problem.
Enterprises invested heavily in AI for writing, reviewing and testing code, and it worked. Large-scale studies at Google and Microsoft found double-digit gains in change throughput per engineer.[1][2]

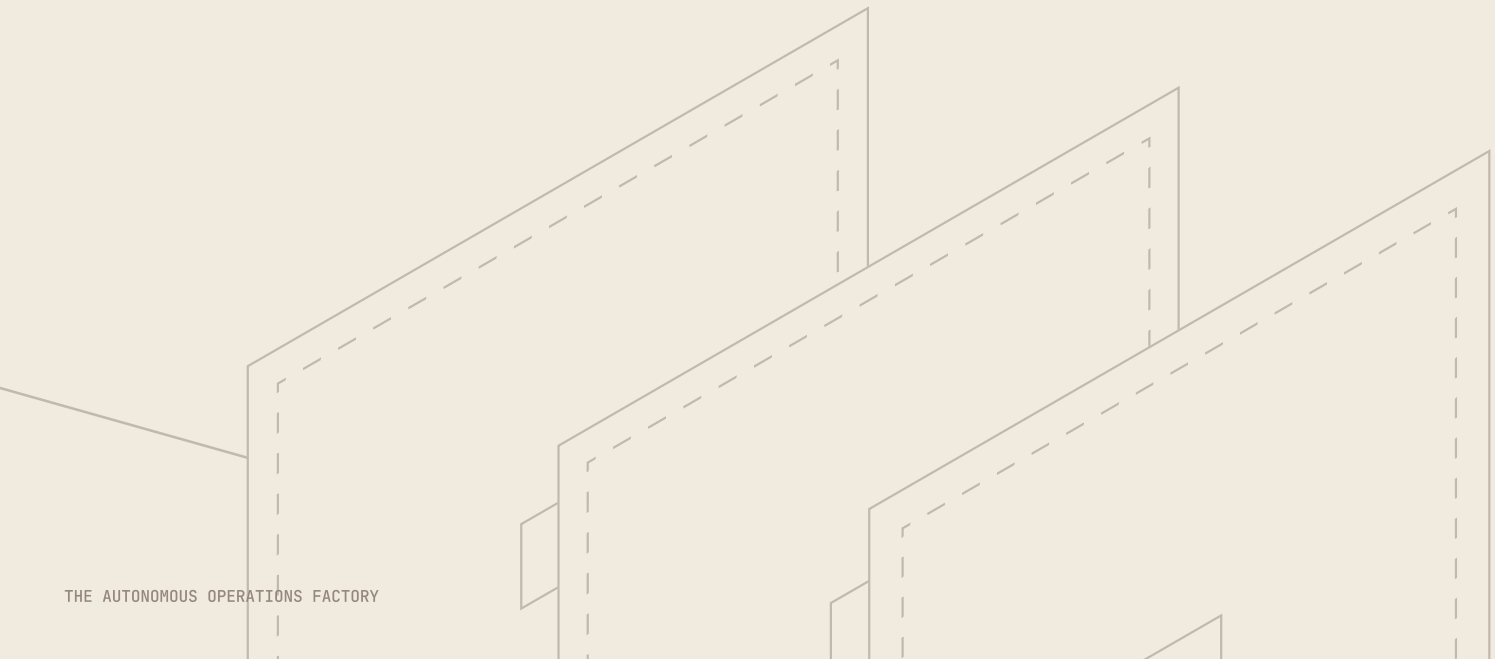
Operations is now at risk of not keeping up.
The half that runs the software has not changed. Four key stages of operations (build, operate, observe and remediate) absorb everything the inner loop produces, at roughly the rate they always did. As coding velocity increases, work accumulates in operations, and risk accumulates with it.

Improving cycle time is now critical, and that means tackling fragmentation.
Cycle time is slow because the four stages run as four disconnected functions, and the elapsed time lives in the handoffs between them.

Adding more point agents will not solve the problem.
The point agent model has five challenges: context is rebuilt for every agent; single-domain agents cannot reach their full potential; nothing holds the objective between runs; approvals and audits become fragmented; and toil moves rather than disappearing.

What operations needs is a factory of its own.
The Autonomous Operations Factory is specialized agents sharing context, working toward the objectives you set, at an autonomy level you control, on the tools you already run.

Start moving toward the factory, one stage and one service at a time.
First, begin measuring your autonomy: the Autonomy Index gives a level per stage, Autonomy Coverage the share of your estate at it. Second, expand that coverage, beginning with the highest-payoff stage on the services where work is waiting, then add stages. Third, build the enablers as you go: one identity for a service across stages, declared intended state, and objectives written as numbers. Part 5 has the detail.



PART 1

Why an operations factory is needed

The bottleneck has moved

Change volume is up, deploy frequency is up, and the blast radius of any single change is larger than three years ago. Operations absorbed all of it with no comparable change of its own.

The gain is measured, not asserted. Google engineers who adopted an AI feature in their IDE submitted 17.5% more change lists per user-month against an 18,000-engineer comparison group.[1] Microsoft engineers adopting command-line coding agents merged 24% more pull requests per engineer per day.[2] Both measure more change arriving faster at the boundary of the outer loop.

The imbalance is now doing damage rather than creating drag. DORA's 2025 research found AI adoption associated with reduced delivery stability.[4] StackGen's State of Reliability 2026 analysis found AI now accounts for roughly 10% of disclosed company incidents, a six-fold rise in three years, and documented nine cases in the past year of an AI agent taking destructive action against a live production system on its own.[5]

It is a queue, and only one lever moves it

Changes arrive at the operations loop at some rate. The loop clears them at some rate. When arrivals outpace clearance, work accumulates: deployments waiting on a window, incidents open, tickets queued. That is the backlog, and it grows for as long as the imbalance persists.

The signature is already visible in delivery telemetry. Across 28.7 million CI workflows, CircleCI found feature-branch activity up a median 15% year on year while main-branch throughput fell a median 7%.[6] Faros AI, across roughly 22,000 developers, found median time to first review up 156.6%.[7] Both measure a jam at code review, upstream of operations and moving toward it.

Three levers could change that balance: slow the arrivals, add people, or clear the work faster. Only the third is open to most.

- **Slow the rate of software change.**
  In practice this means slowing development down.
- **Add capacity.**
  Headcount is not scaling with change volume in any organization we talk to, and even where the money exists, hiring does not track a step change in output.
- **Reduce cycle time.**
  The only lever left. If the operations loop clears work as fast as it arrives, the backlog stops growing however high the arrival rate goes.



FIGURE 1: The outer loop today, and with the factory. The same arrival rate, a different rate of clearance

That makes the case for the operations factory a structural argument. The question is not what tools you are using or the strength of your operations team. It is whether your outer loop can clear work at the rate your inner loop now produces it.

The cause of slow cycle time is fragmentation

The four stages map, in most organizations, to four teams and four toolchains: build to infrastructure, operate to DevOps, observe to observability, remediate to SRE. Each is excellent within its domain. The problem is that none is aware of what the others know. Two ordinary examples.

- An observability agent diagnoses an incident without knowing which infrastructure change is pending approval.
- A cloud agent optimizes a resource without knowing the application's release commitments.

Each is locally correct and globally wrong. None is a tool defect. The failure happens in the gap between them, and that gap is where the cycle time goes. The tooling knows what the logs say. It does not know what Terraform changed.

The cross-domain case is where the cost concentrates, and it has been measured. Across 225 services in one large cloud estate over twelve months, for more than 52% of outages the service where the problem surfaced was not the service where the root cause sat.[3] Across twenty large online services, reassigning an incident to a second team multiplied triage time by up to 10.2 times, and 15.2 times at the highest severity.[8] Operations runs as a line, and every handoff is a stop.

PART 2

Why more agents is not the answer

Every one of those four stages now has agents available for it. They are useful, and adopting them is the right first move for most teams. They do not shorten the loop, because they inherit the boundary they were built inside. There are five reasons, and the Autonomous Operations Factory model in Part 4 answers each. The element that does so is named in italics.

1. **Context is rebuilt for every agent, on every task.**
   Every point agent assembles its own view of the estate and discards it when the task ends. That cost is paid again each time and never compounds. An agent reviewing a pull request does not know the current infrastructure state, or what incidents the last similar change caused. Answered by the world model.

2. **A single-domain agent cannot reach its full potential.**
   It can improve the work inside its own stage, and often does. What it cannot see is the handoffs, the waits and the rework either side of it, so end-to-end cycle time barely moves. Answered by the runtime.

3. **Nothing holds the objective between runs.**
   A point agent takes an instruction, completes it and stops. No objective survives a sequence, so a person restates it at every handoff. That person is the integration layer. Answered by the intent.

4. **Approvals and audit become fragmented.**
   Each agent arrives with its own guardrail model. Ten agents means ten approval surfaces, ten audit trails and nowhere to state policy once. Governance gets harder with every agent added. Answered by the factory specification.

5. **The toil moves rather than disappearing.**
   Integration, context plumbing and agent supervision land on the platform team, and headcount was the constraint the exercise started with. Answered by the learning step.

PART 3

What an Autonomous Operations Factory is

The definition

The Autonomous Operations Factory is a governed operating model in which specialized agents share operational context, work toward the objectives you set, and coordinate work across build, operate, observe and remediate, at an autonomy level you control, using your existing tools as the system of execution.

It is an operating model rather than a product. A factory is typically assembled from several elements rather than bought whole, and a smaller operations footprint may need only one of them.

- **It is governed.** Autonomy without policy is not a factory. Autonomy is set at one of five levels, and you choose which.
- **It works toward objectives you set.** The agents are not only bounded by what they may not do. They are pointed at what the organization is trying to achieve, and the factory holds that objective between runs.

Where it sits

The market for operations tooling is organized by stage: infrastructure platforms for build, delivery platforms for operate, observability platforms for observe, incident and SRE tooling for remediate. Each of those four is now growing an agent layer, and each agent inherits the boundary of the category it was built in.

An operations factory is not a fifth category alongside them. It is what makes the four work as one: shared context underneath, one objective across them, one governance record over the top. Your existing platforms stay the system of execution.

This is also what separates it from AIOps. AIOps correlates signals across domains to tell a person what happened. An operations factory coordinates action across domains, so the work is cleared rather than reported.

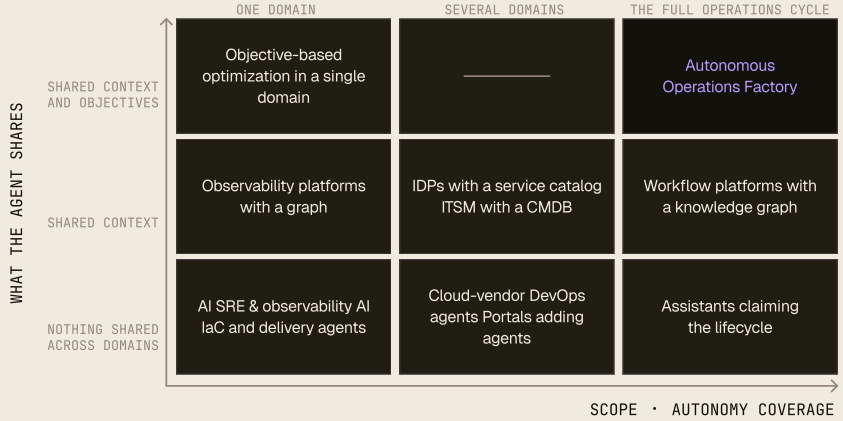


FIGURE 3: The market, by scope and by what the agents share. Scope is what Autonomy Coverage measures in Part 5; what the agents share is what lets the Autonomy Index move. The top right is empty because operations tooling is bought by domain and teams are staffed by domain, so the categories cluster low and left.

The four components

This is the inclusion test: a candidate is only an operations factory if it has all four. Gartner predicts more than 40% of agentic AI projects will be canceled by the end of 2027 on escalating costs, unclear business value or inadequate risk controls.[9] Components three and four are the answer to that. Governance and instrumentation are what stop an agentic program being canceled. The stages can grow, FinOps and security being the obvious additions, but the four components do not.

1. **A production line with defined stages.**
   Work enters at a stage and produces a defined output: a provisioned environment, a verified deployment, a resolved incident.
2. **A shared context that every stage works from.**
   Not integrations between stages. One picture of the system each stage reads before it acts and writes to after it finishes. This is the component almost every current architecture is missing.
3. **Guardrails and inspection at each step.**
   Policy evaluated at every action boundary, before the action executes. Human approval where you require it, and an audit record of every input, decision and action.
4. **Instrumentation that reports what it costs and how well it runs.**
   In operations, how much of the work is genuinely being done autonomously, and at what cost.

The first component in detail. These are the stages the line runs, the domain each one maps to, and what an agent does inside it.

STAGE	DOMAIN	WHAT ITS AGENTS DO
Build	InfraOps	Discover, codify, provision and reconcile infrastructure; detect and remediate drift
Operate	DevOps	Gate, verify, promote and roll back deployments against live system state
Observe	Observability	Correlate signal, suppress noise, rank by impact and error-budget burn
Remediate	SRE	Triage, investigate, root-cause and resolve incidents
Learning step	Across all four	Prevention rules and calibrated gates from every outcome

The value is not in any one stage but in what the shared context makes possible between them. A deployment gates on infrastructure drift because the drift is in the shared context.

What it is worth

This is what the model should be judged on. Each moves for the same underlying reason: the context an agent needs is already there when it acts.

- **Velocity.**
  Agents read the shared context immediately rather than waiting for a person to assemble it, so the wait between stages collapses. Lead time from merge to production falls and the queue in front of each stage stops growing.
- **Reliability.**
  Operational context reaches the change while it is still being made, so a class of incident is prevented rather than resolved. When one does happen, the same context is already loaded, and investigation starts from what changed rather than from what is alerting.
- **Productivity and cost.**
  Routine decisions resolve without escalation, so the same team absorbs a higher arrival rate without hiring. StackGen runs its own multi-cloud Kubernetes estate across AWS, GCP and Azure with a four-person SRE team. The factory reports what each action costs to run, so autonomy never becomes an unbounded invoice, and the estate itself gets cheaper because an agent that can see release commitments and compliance constraints can right-size the resource nobody dared touch.

PART 4

How it works

This section covers how the Autonomous Operations Factory works, first from a workflow perspective and then from an architecture one. The workflow is the loop: intent becomes a specification, the specification runs, and every outcome feeds back to drive learning. The architecture is the layers underneath it. The loop answers the five failures in Part 2, one each.

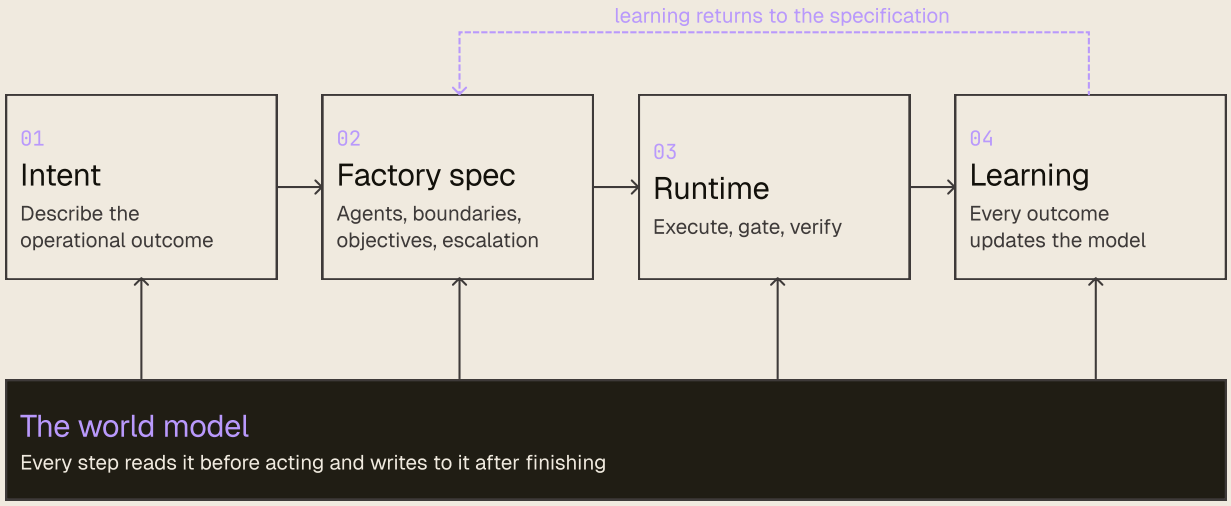


FIGURE 2: The loop. Intent becomes a specification, the specification runs, and every outcome feeds back to drive learning.

1. **Intent holds the objective**
   The model starts with an operational outcome in plain language: move forty services off AWS under the exit mandate; keep this service inside its error budget. Intent names an outcome and leaves the decomposition open, and it can be standing rather than one-off. A copilot takes an instruction and finishes. A factory holds an objective.

2. **The factory specification makes approvals and audit compose**
   Intent becomes a factory specification: a version-controlled document naming which agents to instantiate, their decision boundaries, the context they may read and write, the objectives the factory commits to, and when it escalates. The team approves it before anything runs. This is what makes governance compose rather than multiply. Policy is stated once, for the whole factory, instead of once per agent.

3. **The runtime moves the line, not just the stations**
   Runtime is where the agents work. They come from the four domains set against the stages in Part 3.

   No vendor will have a pre-built agent for every task. Custom agents are expected, and an agent from any source runs inside the same guardrails and audit trail.

   Orchestration owns the rest: assembly, sequencing on durable workflow execution, approvals at each action boundary, verification against the commitment, and holding the objective between runs. Without it there is no factory, only four domains automating in parallel.

4. Learning stops the toil simply moving

Four steps run continuously: execute, observe whether the outcome matched the objective, record the event with confidence attached, and adapt so the next similar situation starts from richer context.

That compounds three ways. Resolution time falls, as a novel incident class goes from a full analysis pass, to a pattern matched in seconds, to a prevention rule that stops it firing at all. Detection moves earlier. And inference cost falls, as familiar work moves to smaller models.

All three reduce cycle time at the same arrival rate, which is the only lever available. Without this step, the work an agent saves is replaced by the work of supervising it.

What the loop runs on

The world model. In reinforcement learning, a world model is learned from experience and run forward to predict what an action will do.[10] An operations world model works the same way from a different starting point. You cannot learn a production estate by exploration, because the cost of a wrong action is an outage. So it begins with what the systems of record already hold and updates from every real outcome after that. Today that prediction is historical: which deviations have preceded which failures, and with what confidence. The direction of travel is toward testing a proposed action against the model before it runs, and we expect that to become how teams establish trust in an agent's judgment.

It holds nine data types: alerts and incidents; metrics, logs and traces; deployments and changes; service dependencies; cloud and Kubernetes resources; app code and infrastructure; policies, desired state and drift; pipelines and workflows; and tickets, runbooks and postmortems. A root-cause agent queries prior incidents with similar signals before forming a hypothesis, then writes the confirmed cause back. The set is not fixed. It widens as more systems of record are connected, and every addition extends what an agent can settle without asking a person.

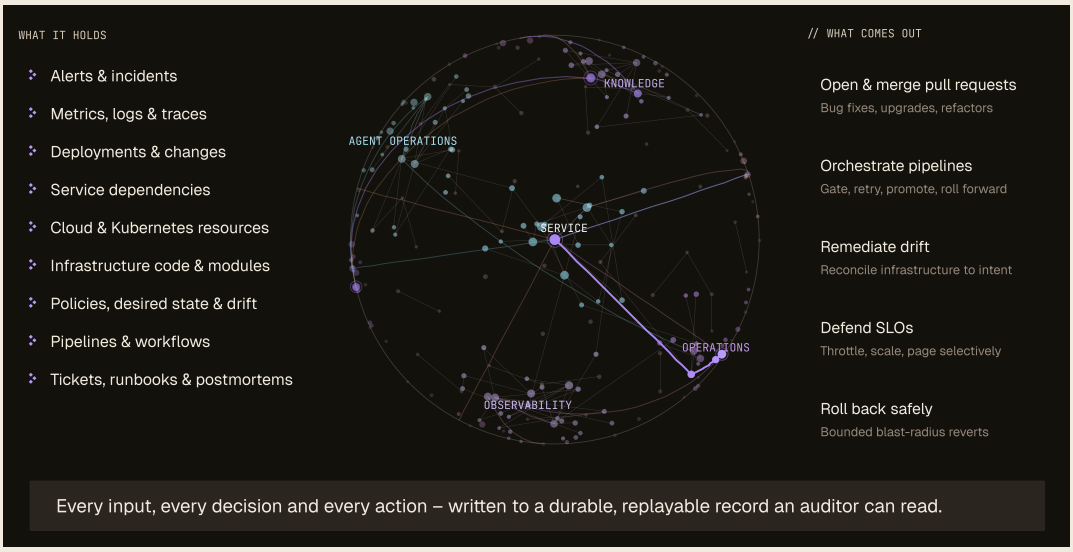


FIGURE 4: The world model. What it holds, how agents reach it, and what comes out of it.

Every fact carries its evidence: where it came from, how confident the system is, when it was true, and who asserted it. Agents reach it through one governed query service rather than the graph itself, which is what lets an agent from any source inherit the same access rules.

Holding a graph is not the same as resolving anything. On the OpenRCA benchmark, agents handed the raw telemetry and its schema still reached a best accuracy of 11.34% on root cause.[11] What closes that gap is modeled state with evidence attached.

The agent operating system. The layer that decides what an agent may do, what it may spend, who it acts as, and what it leaves on the record. Eight things belong here.

ELEMENT	WHAT IT DOES
Skills and workflows	The reusable units an agent runs. Author once, version them, and every agent in every play calls the same tested step.
Policy as code	What each agent may touch, in which environment, at which autonomy rung, declared in version control rather than configured in a console.
Guardrails and approvals	Scope every agent by service, environment and change class, and put a named human on the changes you decide need one.
Identity and access	Every agent holds its own scoped credential and shows up in your logs as itself. No shared key, no borrowed human account.
Tokenomics	A cost ceiling per run, per agent and per team, enforced before the spend, so autonomy never turns into an unbounded invoice.
Audit and evidence	Every decision, the objective version behind it and both agents' positions, written to a signed record you can hand an auditor.
Integrations	Your pipeline, ticketing and monitoring stay the system of execution. Agents act through them; the world model reads from them.
Model routing	Every step goes to the model that suits it, balanced across quality, speed and cost, without anyone hard-coding a model into a workflow.

Bringing these workflows and elements together, this is how an Autonomous Operations Factory is constructed across five layers.



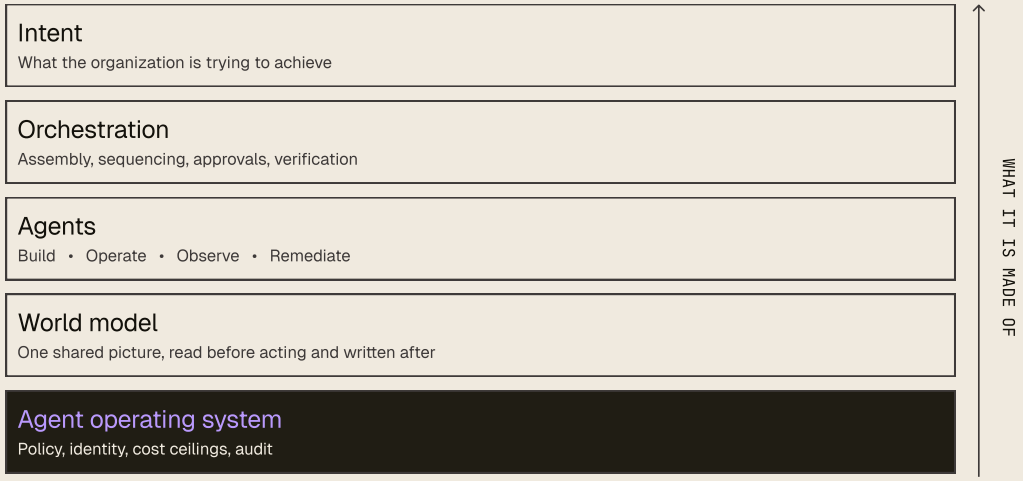


FIGURE 5: The layers. What the factory is made of, where Figure 2 is how work moves through it.

Where the human stays

Two defaults sit underneath, whatever the level. Access is read-only until write is explicitly granted, per resource. Remediation arrives as a pull request, not a silent change, unless a team grants otherwise.

Autonomy is a setting, not a maturity score. It runs on five levels, and different parts of an estate will sit at different levels. The autonomy spectrum below is derived from StackGen's previously published infrastructure autonomy levels.[12]

LEVEL	WHAT THE FACTORY DOES	HOW THE DOMAINS COORDINATE
L5 Full autonomy	The factory improves itself. People set targets, ratify policy and audit outcomes.	A learning in any domain becomes policy for every domain.
L4 Supervised autonomy	The factory runs as one system toward declared outcomes. People intervene by exception.	Machine to machine, through shared context and one governance layer.
L3 Intent-directed	Automation runs inside bounded domain cells. People issue intent and approve at gates.	Within a domain only. People are the glue between domains.
L2 Assisted	People author the definitions. Machines validate, execute and flag deviation.	Tool by tool. People coordinate across teams and tools.
L1 Manual	Everything hand-built: scripts, CLI, tickets, runbooks.	None. Meetings, tickets and tribal knowledge.

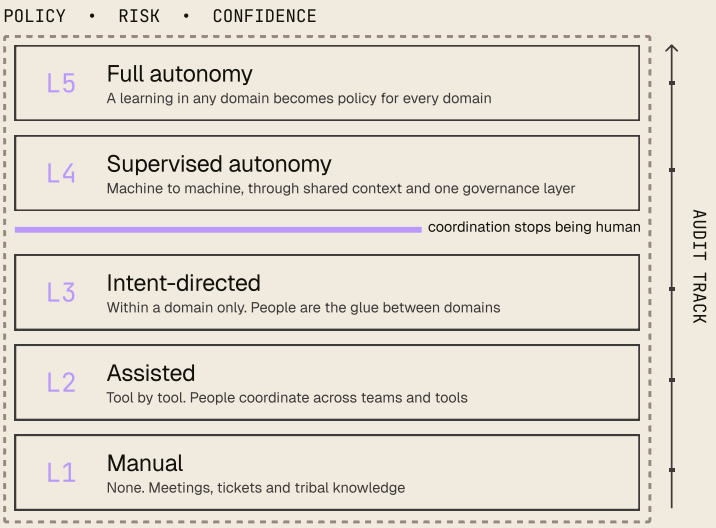


FIGURE 6: The autonomy ladder. Five levels within a stage, and the boundary where coordination stops being human. Figure 3 is the other axis: how much of the estate is covered

L3 is the natural ceiling for a single-domain tool: an agent can be excellent inside its own cell and still leave a person carrying context to the next one, which is why adding more of them does not shorten the loop. L4 is the first level at which a learning in one domain reaches another without a person relaying it, and that is a property of the system the agents run inside rather than of any agent. A separate guide sets out the five levels within each domain.

A worked example of multiple agent coordination

One intent, crossing four domains, with the shared context underneath it. The times are elapsed, and no person is paged at any point in the sequence.

TIME	DOMAIN	WHAT HAPPENS
02:47	InfraOps	An infrastructure watch agent detects a Terraform state deviation in an autoscaling group and scores the drift risk high, having correlated it with three prior incidents of the same class.
02:51	Observability	Memory pressure appears on dependent services. A change-correlation agent links the two. A P1 fires. No human is paged.
02:52	SRE	The root-cause agent traces it to the autoscaling deviation.
02:54	InfraOps	A remediation agent generates corrective infrastructure as code, which passes the policy and compliance gates.
02:55	DevOps	A deploy agent ships it through the validated pipeline behind a canary.
03:04	SRE	A verification agent confirms recovery, closes the incident and drafts the post-mortem.

Seventeen minutes, four domains, no handoff and no page. Without the shared context the drift is never checked, and root cause takes an hour across three tools.

PART 5

Where to start

Measure the autonomy of your operations factory

A factory that cannot report on itself is not instrumented, and the industry does not yet have the measures it needs. Two are wanted. The Autonomy Index is the share of operational decisions resolved by agents without human escalation, weighted by risk and complexity. Autonomy Coverage is the share of the four stages running on agent-executed workflows at all. Both should count every agent acting on the estate.

The first read needs no instrumentation. You can score a stage from how work moves today: who decides, who relays, and where it waits. The enablers in the third section make that measurement precise and continuous.

Neither is the point on its own. A high Autonomy Index over low Autonomy Coverage is a pilot rather than a factory. Together they are the leading indicators for improving velocity, reliability and productivity. Read them by line of business, team and agent type.

Higher autonomy improves the operational measures, which roll up to the business measures above them. For example, autonomous systems with full context can not only reduce time to restore through faster root cause analysis, but can also reduce change failure rate upstream by understanding the development practices that lead to incidents.

DRIVER	BUSINESS MEASURE	OPERATIONAL MEASURE
Velocity	Value of features shipped; cycle time for a new feature	Lead time, deployment frequency, queue depth per stage
Reliability	Failed customer interactions; customer churn	Change failure rate, time to restore, error-budget burn
Productivity and cost	Operations cost as a share of revenue	Work cleared without escalation; cost per action; development-to-operations staffing ratio

Expand the coverage of your factory

Coverage expands on two axes: which stages run inside the factory, and which services those stages run for. Adoption happens one cell at a time, one stage against one tier of services, rather than everything at once.

- **Start with the highest-payoff stage.** Pick the stage where the cost of today's operations model is easiest to agree on. For many teams that may be remediation if the current toil and business impact of incidents are high, but it is whichever one your own baseline says has the highest immediate opportunity.

- **Start with the highest-payoff services.** Payoff is where work is waiting rather than where the most is at stake: queue depth, change rate, and the share of requests that escalate. The most business-critical service is often the most change-controlled, and that is the slowest place on the estate to prove anything. Remediation on a payment service is a sharper first test than four stages across the whole estate, provided that is where the queue actually sits. Put agents on real production traffic in the cell you pick, set to recommend and prepare rather than execute, and raise the level of autonomy as the agent performance earns it.
- **Add stages, and measure what only the connections can move.** A single-stage agent can improve change failure rate up to a point, and then stops, because the remaining causes sit beyond a boundary it cannot see. Measure it as each new stage joins, so the value of the connection is visible rather than assumed.

Build the key enablers for your factory

Connect the repository, pipeline, ticketing and observability systems, and build the shared context underneath them. Five actions make a strong foundation, and each unlocks something.

- Give a service one identity across all four stages, so an agent can tell that the thing alerting and the thing deployed are the same thing.
- Express policy as code with an append-only record, so an action can be refused before it executes and explained afterward.
- Write objectives as numbers with owners, so an agent has something to work toward and a rule for stopping.
- Declare intended state, so drift is detected rather than inferred.
- Inventory which requests cross domains, so you know where the elapsed time goes before you spend anything.

PART 6

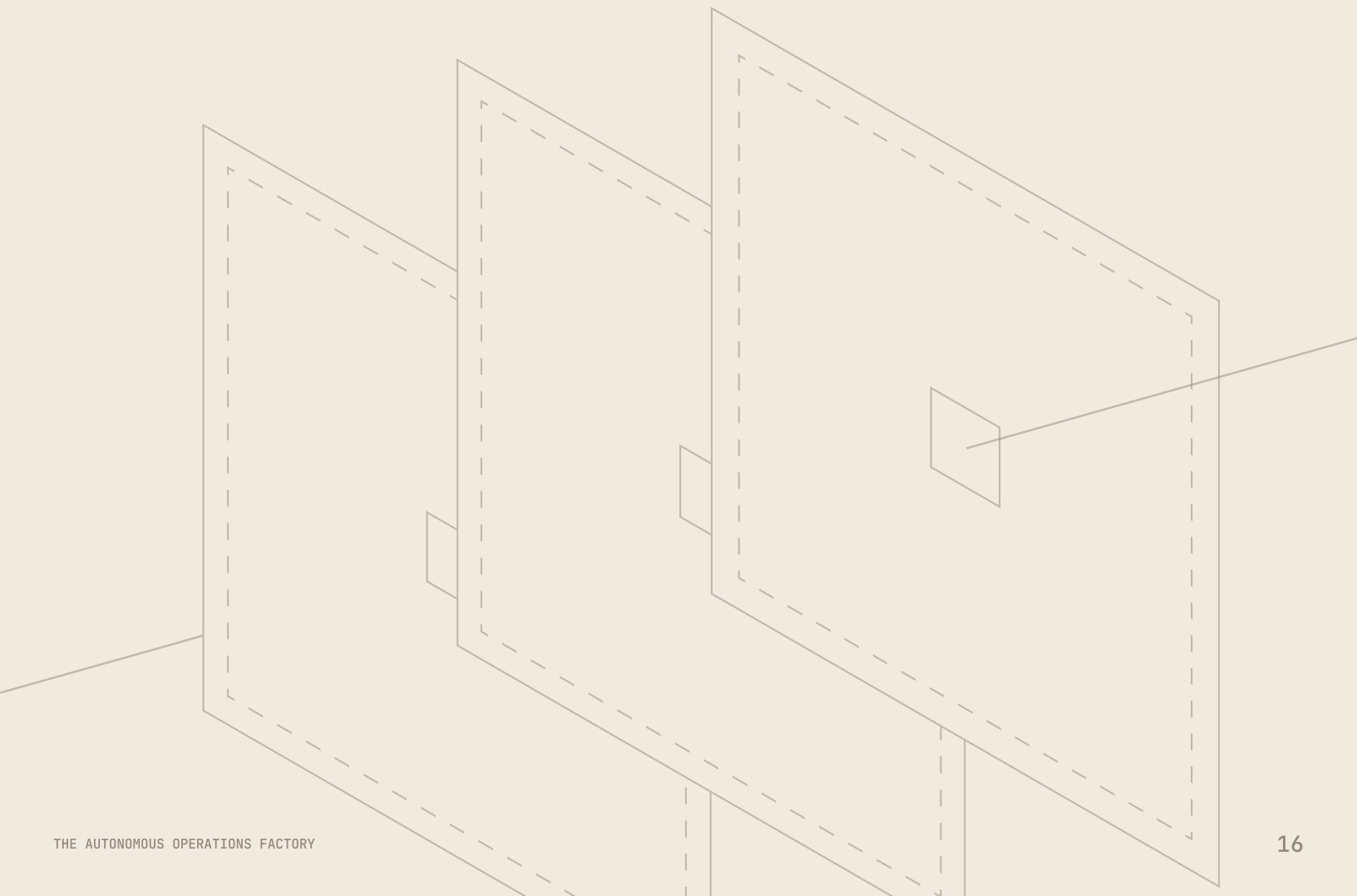
How StackGen is building the Autonomous Operations Factory

Operations needs more coordination than point agents can deliver.

We are building multiple entry points to the Autonomous Operations Factory. Aiden for InfraOps, DevOps, Observability and SRE are generally available today. Each provides value on its own, and each is built to integrate with the others.

We are building the shared context underneath them. The world model is what turns four products into one factory rather than four tools that happen to be adjacent. The work now is breadth: connecting more sources so that more of the estate is represented.

We are building the governance layer once, for every agent. The agent operating system is in production. Policy, identity, cost ceilings and the audit record apply to agents you built and agents you bought elsewhere on the same terms as ours.



Conclusion

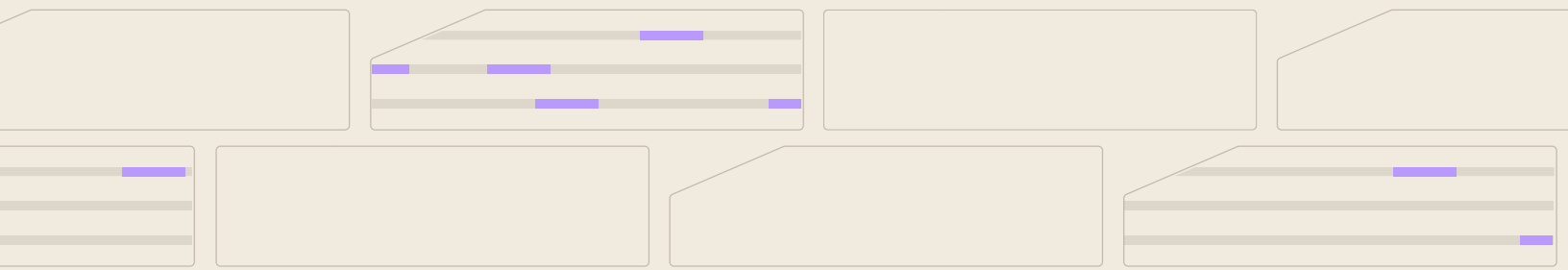
To keep up with the pace of change that the software factory now produces, the industry needs an operations factory. The industry's focus over the last two years on adding intelligence to individual operational tasks as point agents has moved us forward, but the next leap requires that agents share a picture of the system, a common set of objectives, and a governance layer that holds whoever built them. That is the shift from having agents to running a factory. The organizations whose operations will keep pace with development will not be those with the most agents, but those whose agents work from the same context, toward the same objectives, inside guardrails somebody can audit.

About StackGen

StackGen helps companies build autonomous operations factories. Aiden agents for infrastructure, delivery, reliability and observability run on a shared agent operating system that gives every agent common operational context under policy-bound, human-approved guardrails. StackGen's customers include Autodesk, Nielsen, Pepsi, OneTrust and Bancolombia. StackGen was named a Gartner Cool Vendor in AI for IT Operations in 2025, and a Sample Vendor in five 2025 Gartner Hype Cycles covering I&O automation, platform engineering, infrastructure strategy, cloud platform services, and monitoring and observability. StackGen is backed by Thomvest Ventures, Seligman Ventures, WestWave Capital, FireBolt Ventures and Secure Octane, and is headquartered in San Francisco.

Gartner, Cool Vendors in AI for IT Operations, Cameron Haight, Hassan Ennaciri, Chris Saunderson, 2 December 2025, ID G00840513.

GARTNER is a registered trademark and service mark of Gartner, Inc. and/or its affiliates in the U.S. and internationally, COOL VENDORS is a registered trademark and service mark of Gartner, Inc. and/or its affiliates, and HYPE CYCLE is a registered trademark of Gartner, Inc. and/or its affiliates, and all are used herein with permission. All rights reserved. Gartner does not endorse any vendor, product or service depicted in its research publications and does not advise technology users to select only those vendors with the highest ratings or other designation. Gartner research publications consist of the opinions of Gartner's research organization and should not be construed as statements of fact. Gartner disclaims all warranties, expressed or implied, with respect to this research, including any warranties of merchantability or fitness for a particular purpose.



Sources

1. Google, Achieving Productivity Gains with AI-based IDE Features, 27 January 2026.
2. Microsoft, Adoption and Impact of Command-Line AI Coding Agents, 1 July 2026.
3. Wang et al., Fast Outage Analysis of Large-scale Production Clouds with Service Correlation Mining, ICSE 2021 (arXiv:2103.03649). 225 services in one large cloud estate over twelve months.
4. DORA, State of AI-assisted Software Development, 2025, fielded June–July 2025.
5. StackGen, State of Reliability 2026. 109,000+ unplanned incidents across 392 companies.
6. CircleCI, 2026 State of Software Delivery Report, 18 February 2026. 28.7 million CI workflows.
7. Faros AI, The AI Engineering Report 2026, 12 April 2026. Telemetry across ~22,000 developers.
8. Chen et al., An Empirical Investigation of Incident Triage for Online Service Systems, ICSE-SEIP 2019. Twenty large online services.
9. Gartner, Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027, press release, 25 June 2025.
10. Ha and Schmidhuber, Recurrent World Models Facilitate Policy Evolution, NeurIPS 2018, arXiv:1809.01999; earlier preprint published as World Models, arXiv:1803.10122.
11. OpenRCA benchmark, ICLR 2025. 335 real failure cases, 523 million log lines, three production systems.
12. StackGen, Intent-to-Infrastructure, platformengineering.org, June 2025. Five-level spectrum for infrastructure intent.

