

● STATUS: OPERATIONAL

Less downtime. More control. Zero surprises.

How a serverless AI platform cuts incident response time from hours to minutes, reduces operating costs, and always keeps a human in command of critical decisions.

TECHNICAL CASE STUDY · AI OPS SERVERLESS PLATFORM

AWS

Terraform

Python 3.12

Step Functions

Bedrock

GitHub Actions

In this ebook

- 01 The problem this project solves
- 02 Benefits for your business
- 03 How it works, from alarm to recovery
- 04 Full architecture
- 05 Security: six layers between the AI and the infrastructure
- 06 Key features
- 07 Technology stack
- 08 Engineering challenges solved
- 09 Cost and efficiency
- 10 The platform in action: a real incident
- 11 Skills demonstrated
- 12 Roadmap and next steps
- 13 About the author

How to read this: this document assumes no prior context. Each section can be read on its own, so feel free to jump straight to the chapter that interests you most.

When something breaks at 3am

Normally, someone has to notice the alarm, dig through logs to figure out what happened, decide on a fix, and apply it manually. This project automates the detection, the diagnosis, and the explanation, and only waits for a person to say "go ahead" before touching anything.

The AI Ops Serverless Platform is an incident-response system built entirely on AWS, combining real-time monitoring with an AI model (Claude, via Amazon Bedrock) to read alarms, write a plain-English explanation of the likely cause, and suggest a fix, without ever deciding on its own what to change in the infrastructure.

The system's core guarantee: the AI never holds the keys. It can only suggest actions from a short, pre-approved list, and the real target always comes from a table defined in Terraform, never from the model. An unfamiliar alarm? The system logs it and waits.

Why this matters

Manual incident response tends to follow the same pattern: 30 minutes for someone to notice, 60 minutes investigating logs, 30 minutes applying the fix, almost 2 hours in total, often outside working hours. This system compresses that cycle into minutes, keeping a human as the final decision-maker at every step that changes infrastructure.

What this means for your business

Behind the technical architecture is a simple question: how does this save a real business time, money, and risk? Here are the numbers and the reasoning.

2–7 min

average time to recover from an incident. Manually, this used to take close to 2 hours.

\$0.60

monthly platform cost at rest, with no incidents. Scales only with actual usage.

100%

of remediation actions require explicit human approval before they run.

24 / 7

monitoring coverage, with no dedicated on-call team needed to watch alarms.

Financial

COST

Every minute of downtime costs money, whether in lost sales, stalled productivity, or customer trust. Cutting the average response time from nearly two hours to a few minutes has a direct impact there. At the same time, the platform itself costs less than a dollar a month when there are no incidents, and scales only with real problem volume, without needing a dedicated team watching alarms after hours.

Performance & operations

SPEED

The team stops losing time digging through logs by hand. The system delivers the likely cause and a suggested fix in under a minute, written in plain language anyone on the team can understand, not just whoever happens to be on call. This cuts alert fatigue and frees engineers for higher-value work.

Security

CONTROL

The AI never has direct access to the infrastructure. Every possible action comes from a list defined in code, reviewed and approved by the team ahead of time, and no fix runs without a person clicking approve. Automation speed is never traded for the control a business needs to keep over its critical systems.

Availability & resilience

CONTINUITY

Monitoring runs 24 hours a day, without depending on anyone being awake or available. The architecture is serverless, which removes any single server whose failure could take down the whole system. And every decision, approval, and execution is logged, building a history that serves both auditing and continuous improvement.

For a business, this means an incident-response system that is cheaper to run, faster to recover, safer to trust, and simpler to audit, without needing an operations team watching it around the clock.

From alarm to recovery

A typical incident moves through seven steps, with exactly one deliberate pause: human approval.

- 01 CloudWatch detects the anomaly** ~60s
An alarm fires based on application metrics (ECS Fargate, Lambda, API Gateway).
- 02 EventBridge routes the event**
The alarm is automatically routed to the right workflow.
- 03 Step Functions orchestrates the response**
A state machine coordinates every subsequent step, with a full execution history.
- 04 Claude analyses the alarm** ~30s
The anomaly_analyser function asks the AI to explain the likely cause and suggest an action, in read-only mode.
- 05 Slack notification and Mission Control approval** 1–5 min
A person reads the explanation and approves or rejects the suggested action on the dashboard.
- 06 Remediation executed** ~30s
Only once approved, the remediator function applies the action, always from the allow-list defined in Terraform.
- ✓ Service recovered**
Total time from alarm to recovery: 2–7 minutes.

Zero cost while waiting: Step Functions pauses execution at \$0 while it waits for the human decision, thanks to the waitForTaskToken pattern. No compute runs, and no risk accumulates, while nobody has decided anything yet.

04 · UNDER THE HOOD

Full architecture

Five Lambda functions, each with a single responsibility and minimal permissions. No function holds more power than it needs.

FUNCTION RESPONSIBILITY		MODIFIES INFRASTRUCTURE?
<code>anomaly_analyser</code>	Reads the alarm, consults Claude, validates against defined rules	No (read-only)
<code>remediator</code>	Executes the approved action (e.g. restarting a service)	Yes, only pre-approved actions
<code>runbook_assistant</code>	Answers operational questions by reading Markdown documentation	No (read-only)
<code>cost_reporter</code>	Sends a daily cost summary	No (read-only)
<code>mission_control</code>	Web dashboard for approvals and history	No (interface only)

Why Step Functions, not a single function

A single, isolated Lambda function doesn't work for incident response, for three reasons: Lambda functions have a 15-minute limit, but humans need more time to approve; if something fails midway, there's no way to know at which step; and there's no way to pause without paying for compute while you wait.

Step Functions solves this

- Pauses at zero cost while waiting for human approval
- Each step retries independently if it fails
- Full execution history, retained for 90 days
- Designed for long-running flows with human decision points

Infrastructure as code

Nearly all the infrastructure is managed by 8 independent Terraform modules: IAM, Lambda, Step Functions, API Gateway, EventBridge, ECS, monitoring, and Mission Control, all version-controlled and reproducible with a single terraform apply.

— 05 · AI GOVERNANCE

Six layers between the AI and the infrastructure

The design principle is simple: the AI suggests, a person decides, and code, never the model, determines what can be changed.

- 1 The AI never chooses the target**
Every action's possible targets always come from a table defined in Terraform, never from model-generated text.
- 2 Allow-list of actions**
Only a short, pre-approved set of actions can ever run; everything else is denied by default.
- 3 Mandatory human approval**
No action runs without an explicit approval click from a person in Mission Control.
- 4 Redundant validation**
The remediation function independently re-validates the allow-list, with no shortcuts inherited from earlier steps.
- 5 Safe by default**
An unfamiliar or unexpected alarm never results in an action, only a log entry and a wait.
- 6 Full auditability**
Every decision, from alarm to approval to execution, is logged in CloudWatch and the Step Functions history.

06 · WHAT THE PLATFORM DOES

Key features

AI-powered incident analysis

Claude reads alarms and explains the likely cause in plain language, no technical jargon.

Mandatory human approval

Nothing runs until a person clicks approve on the dashboard.

Deterministic remediation

Actions always come from a list defined in Terraform, never invented by the model.

Zero-cost waiting

Step Functions pauses at no cost while it waits for the human decision.

Full audit trail

Every decision is logged in CloudWatch and the Step Functions history.

Serverless & cost-efficient

About \$0.60/month at rest, scaling only with real incident volume.

Runbook assistant (RAG)

Answers operational questions by reading the team's own Markdown documentation.

Mission Control dashboard

Web interface for approvals, incident history, and Q&A.

07 · TOOLS USED

Technology stack

LAYER TECHNOLOGY	
Compute	AWS Lambda, ECS Fargate, API Gateway
Orchestration	Step Functions, EventBridge
Monitoring	CloudWatch (Logs, Metrics, Alarms)
Artificial Intelligence	Amazon Bedrock (Claude Sonnet 4.5)
Infrastructure	Terraform, 8 modules
Storage	S3, Secrets Manager

Language	Python 3.12
CI/CD	GitHub Actions
Runbook format	Markdown with RAG retrieval

Continuous integration pipeline

Every push to main or develop automatically runs: Python unit tests (pytest), Terraform format validation, Docker image build and push to ECR, Terraform apply, and a forced ECS deployment.

08 · REAL PROBLEMS, REAL SOLUTIONS

Engineering challenges solved

CHALLENGE SOLUTION	
Waiting on a person without paying for compute	waitForTaskToken
Keeping AI-driven remediation safe	Deterministic list defined in Terraform; the model never names targets
Avoiding accidental infrastructure changes	The Lambda independently re-validates the allow-list
Covering alarms 24 hours a day without an on-call team	Detection and analysis fully automated; the person only approves or rejects
Explaining incidents to non-engineers	Plain-language summaries generated by Claude, posted to Slack
Sharing operational knowledge	RAG over the team's Markdown runbooks, stored in S3

09 · EFFICIENCY

Cost and efficiency

Serverless architecture means paying only for what you actually use.

SCENARIO	MONTHLY COST
At rest (no traffic, no alarms)	~\$0.60
Sample load (10 incidents/day)	~\$3.00
With the demo app running 24 hours a day	~\$9.60

The demo application (ECS Fargate) exists only to show the system running live. Removing it brings the cost down to under \$1/month.

— 10 · A REAL INCIDENT, START TO FINISH

The platform in action

To make all of this concrete, here is a real run of the demo environment, from a healthy service to an incident resolved with human approval.

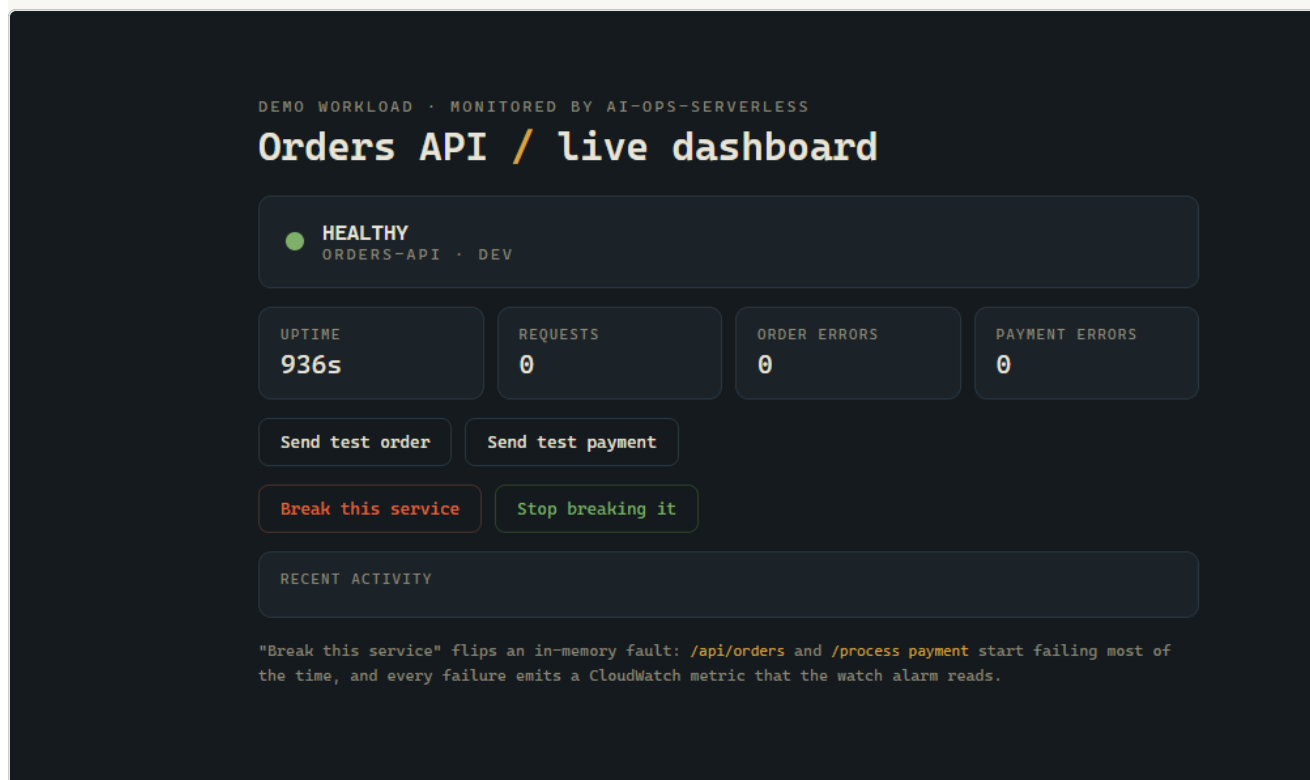


FIG. 1 Orders API healthy, continuously monitored by the platform. This is the normal state CloudWatch observes at every moment.

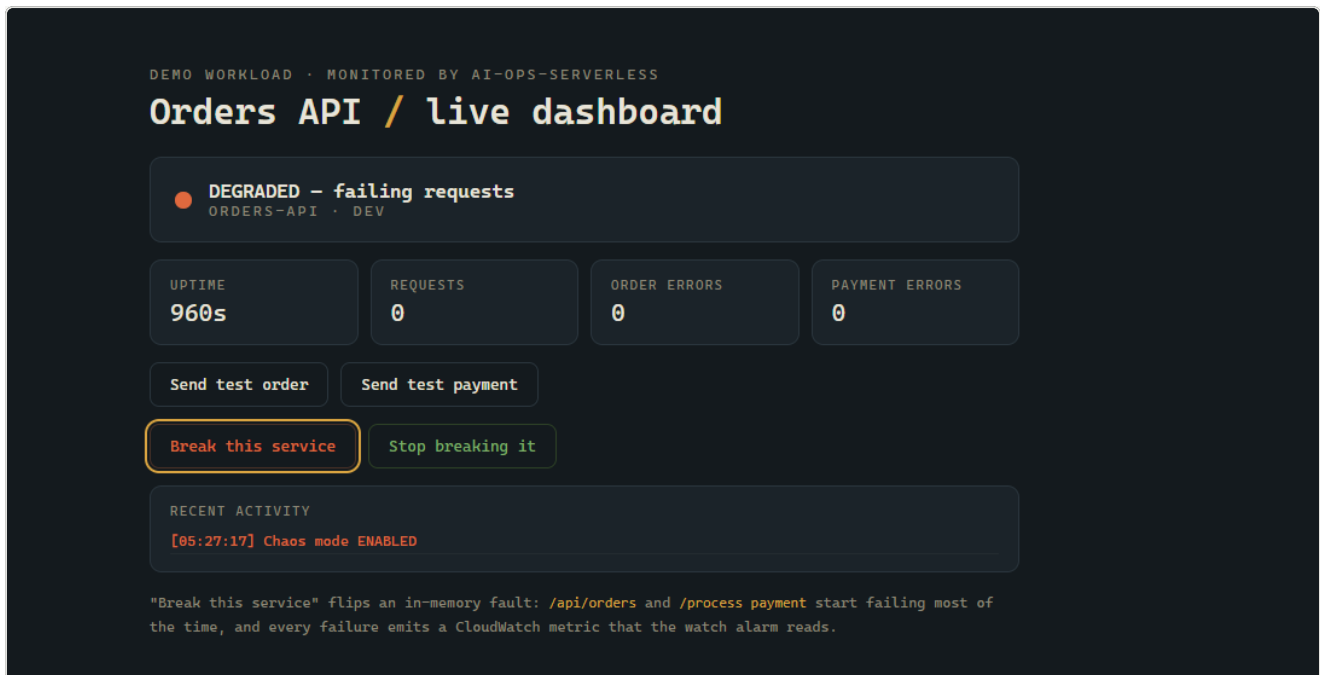


FIG. 2 The service starts failing and errors build up past the acceptable limit. This is the trigger that fires the CloudWatch alarm.

In under a minute, the alarm reaches Step Functions, which triggers the automatic analysis. Claude reads the alarm data and writes a clear summary of what is happening, with no one needing to open a single log manually.

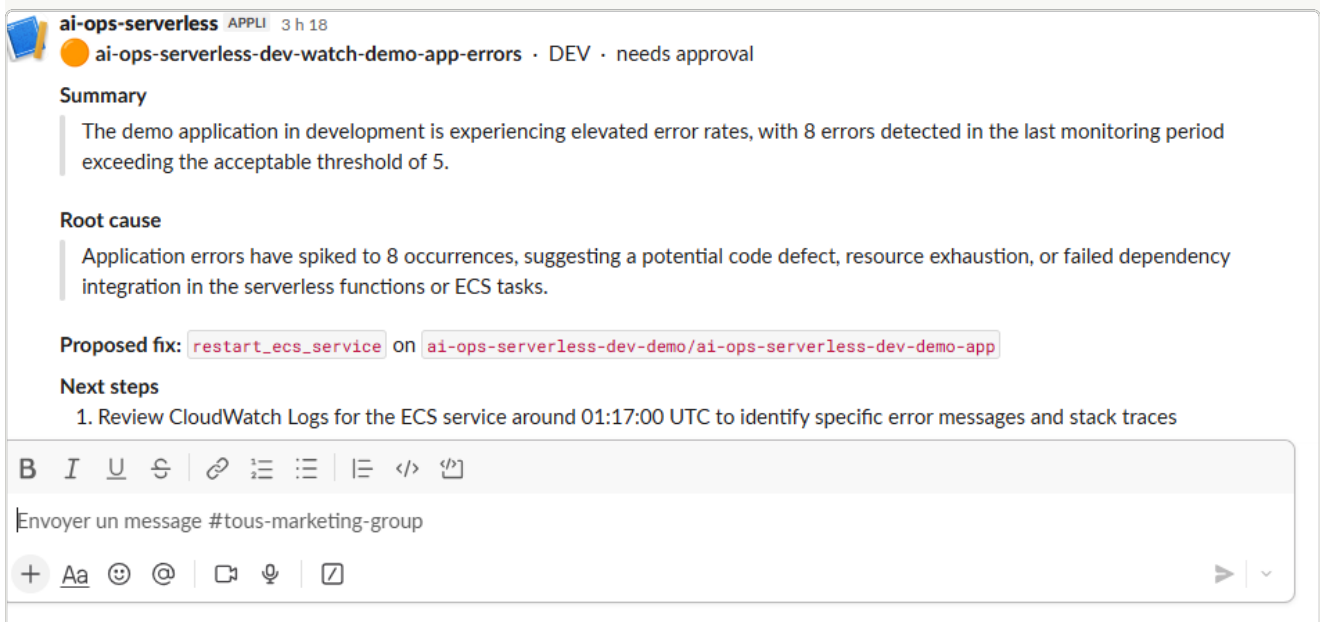


FIG. 3 The explanation lands directly in Slack: the problem summary, likely cause, and proposed action, ready for the team to review.

The suggested action is not executed right away. It waits in Mission Control until a person decides. Note that the platform explicitly states the target was resolved by the system, and not chosen by the AI.

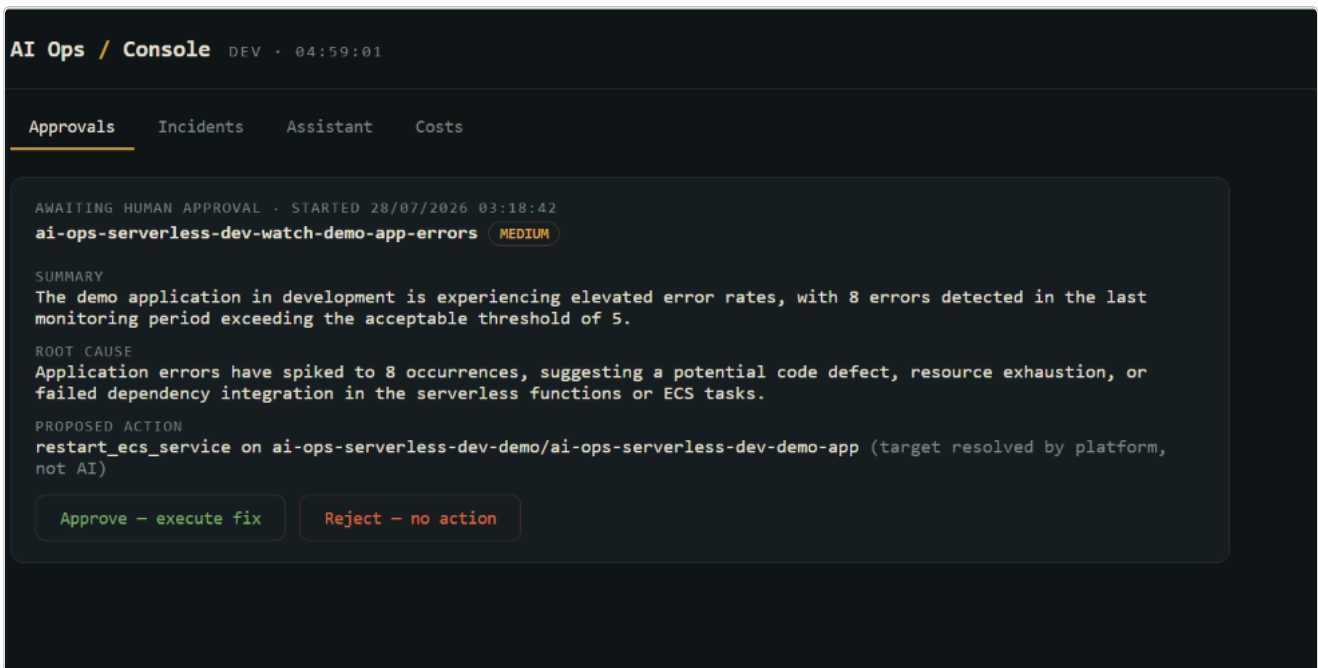


FIG. 4 The Mission Control dashboard shows the incident awaiting approval, with two clear buttons: approve and fix, or reject and do nothing.

Once the fix is approved, everything is logged automatically, with execution IDs, start time, and completion time.

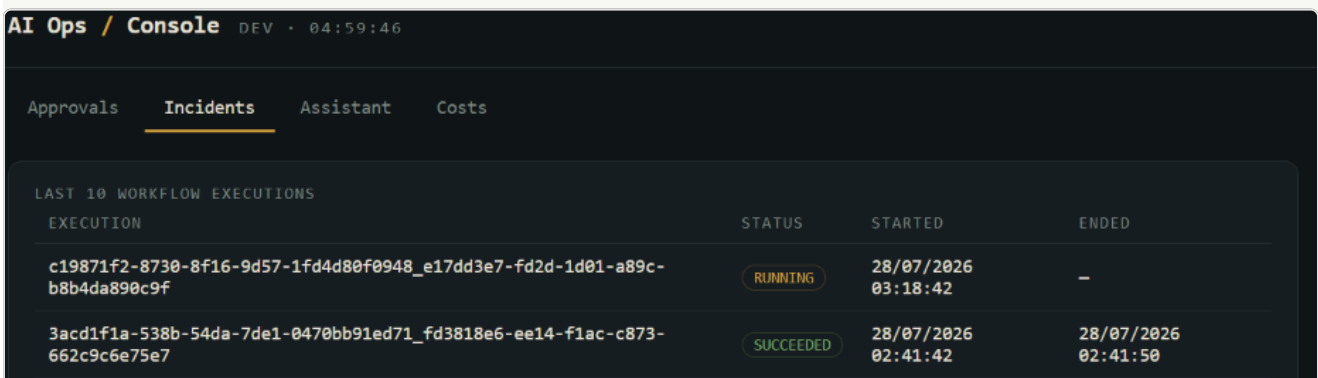


FIG. 5 History of recent workflow executions, with status, start time, and end time for each incident processed.

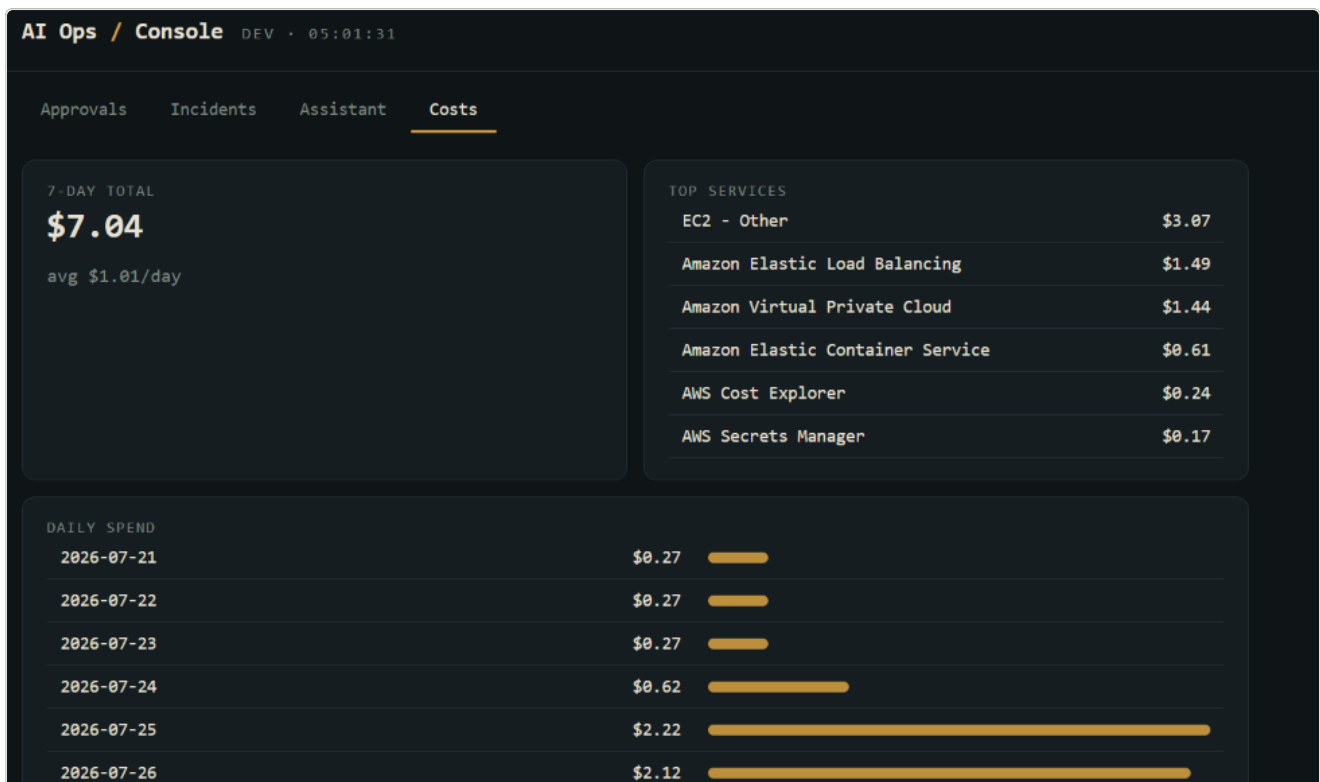


FIG. 6 The costs tab shows real spend over the last 7 days and which AWS services weigh most on the bill, in full transparency.

Beyond incident response, the platform also answers day-to-day operational questions by reading the team's own documentation.

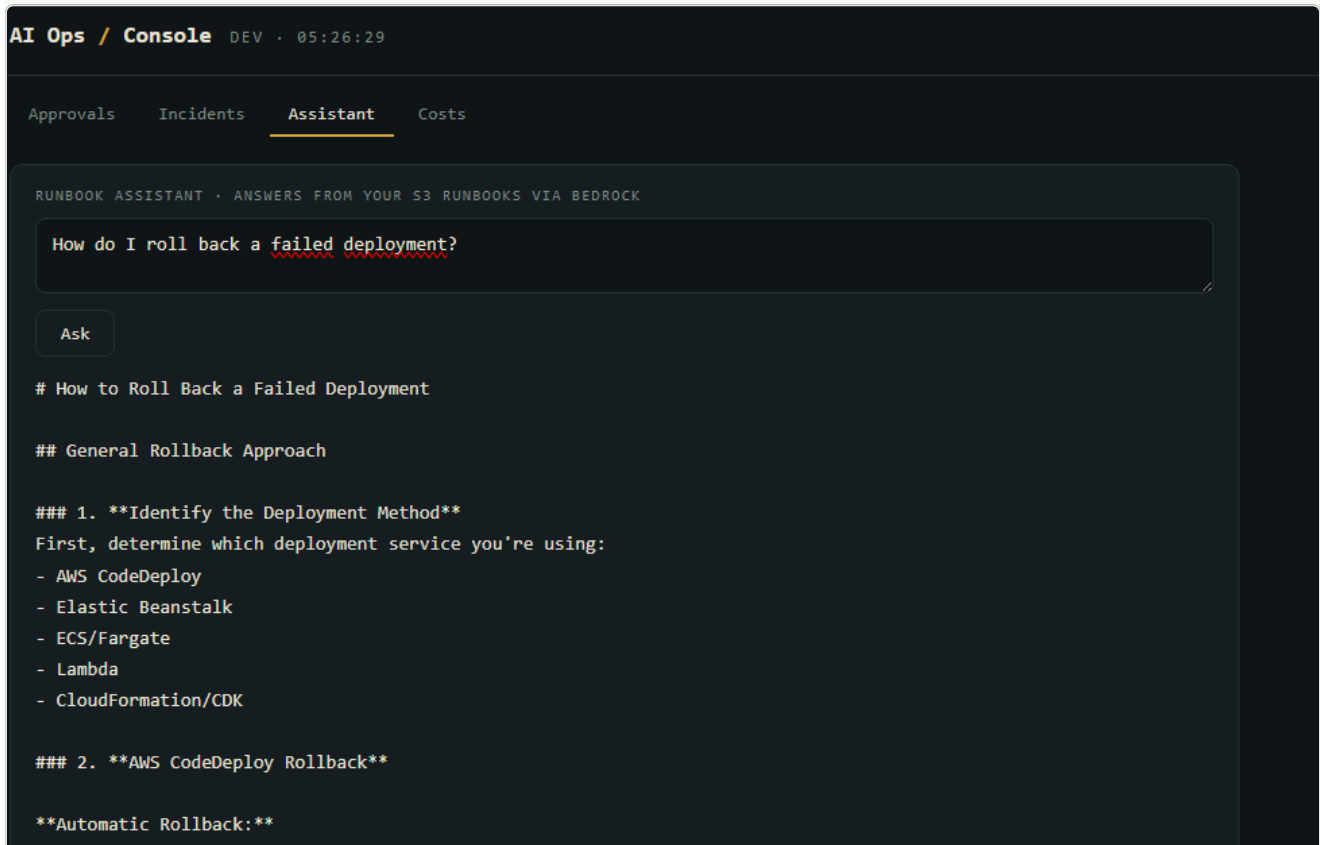


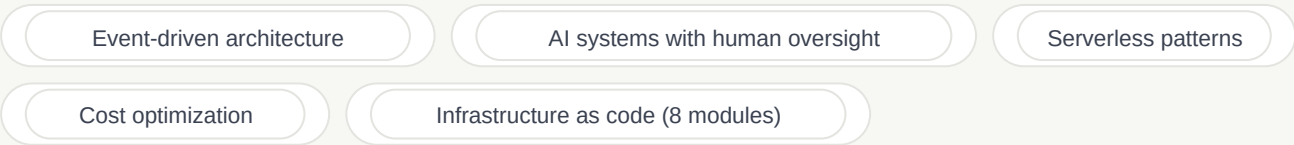
FIG. 7 The runbook assistant answers a real question about how to roll back a failed deployment, based on the documents the team stored in S3.

11 · FOR RECRUITERS AND TECHNICAL TEAMS

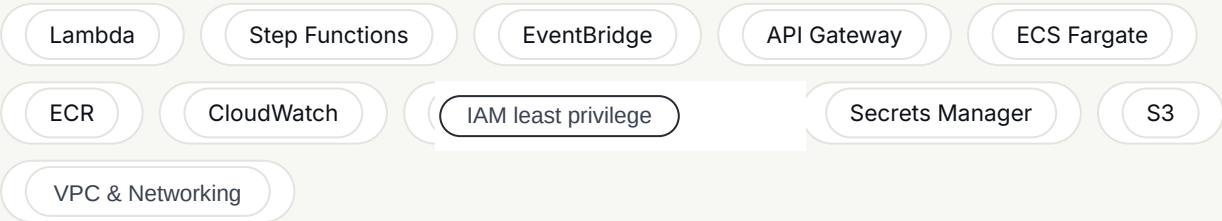
Skills demonstrated

This project isn't a tutorial followed step by step. It's one architecture decision after another, documented and justified.

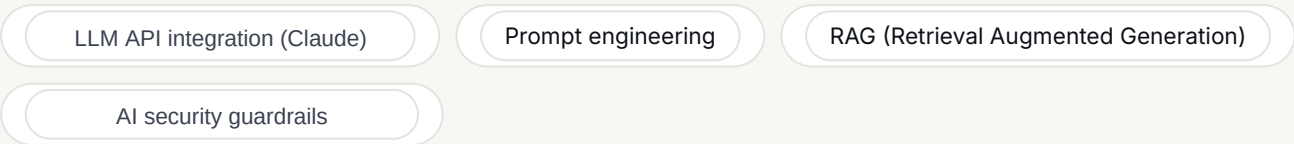
ARCHITECTURE & DESIGN



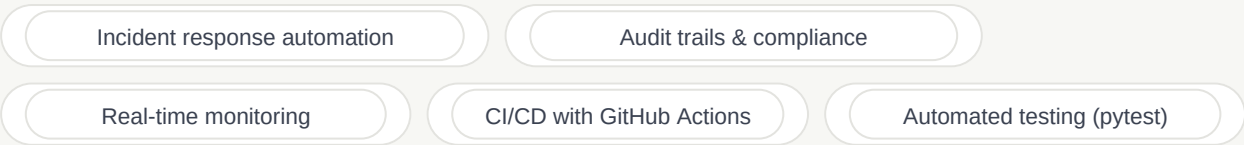
AWS & CLOUD



AI & MACHINE LEARNING



ENGINEERING & OPERATIONS



12 · WHAT'S NEXT

Roadmap

PHASE FOCUS	
Now	Complete Terraform coverage, expand the runbook catalog

Q3 2026	Multi-region support, advanced cost analytics
Q4 2026	Broader remediation (SSM Automation), EKS support
2027	Custom action framework, policy-based approvals

Current state

Working end to end

- Demo app (ECS Fargate) with real alarms
- CloudWatch, EventBridge and Step Functions flow
- Analysis via Amazon Bedrock (Sonnet 4.5)
- Human approval in Mission Control
- Slack notifications
- CI/CD pipeline with GitHub Actions

Next steps

- Structured logging for the custom metrics layer
- Mission Control permissions in Terraform
- Multi-region support
- Broader range of remediation actions

Patricio Lumbe

Cloud Engineer, specialising in AWS and Infrastructure as Code. This project demonstrates real production incident-response patterns: event-driven architecture, secure AI integration, Infrastructure as Code, and human governance over automated systems.

contact@patriciolumbe.com

All code is version-controlled, tested, and deployed via CI/CD. MIT License.