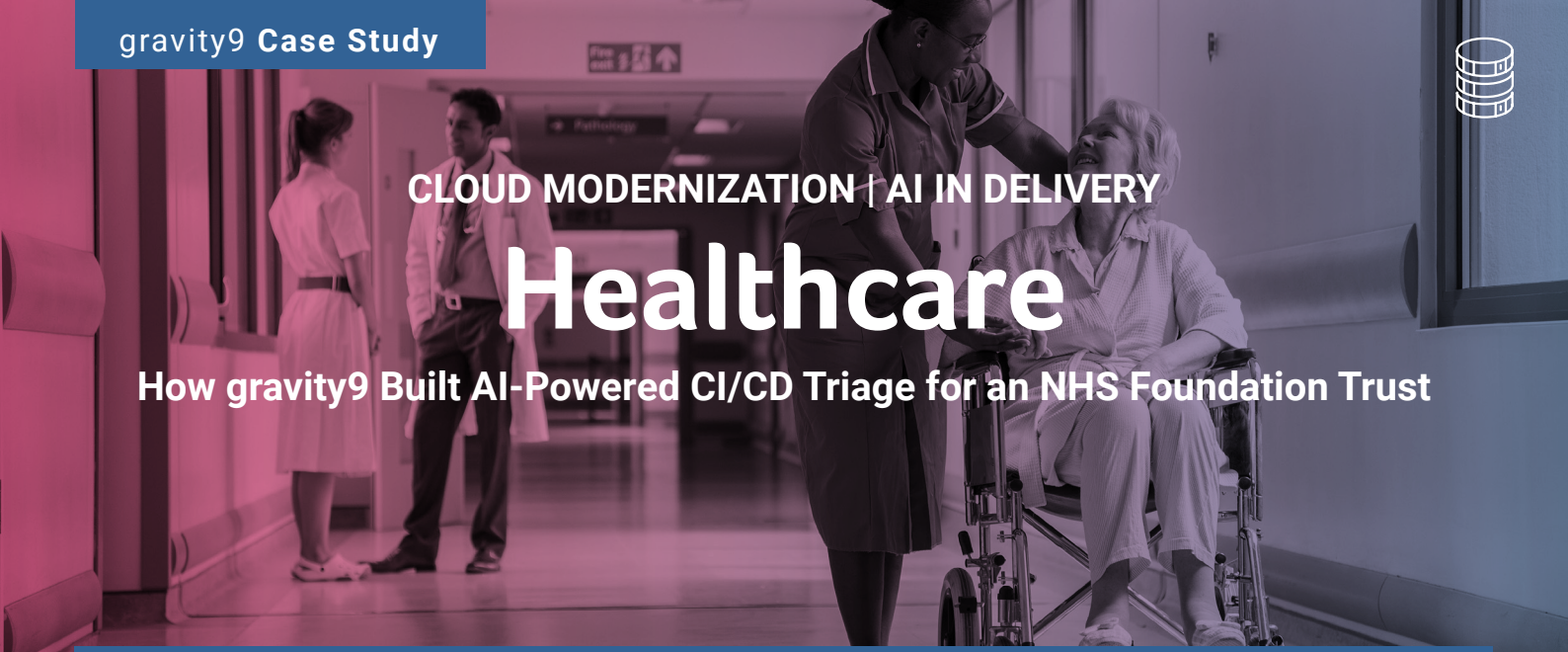


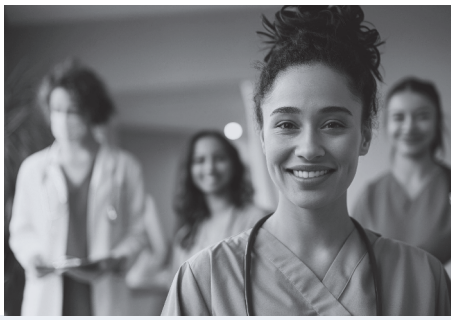
CLOUD MODERNIZATION | AI IN DELIVERY

Healthcare

How gravity9 Built AI-Powered CI/CD Triage for an NHS Foundation Trust

Our client is one of the United Kingdom's largest and most respected NHS Foundation Trusts, running two major teaching hospitals in London plus community health services across South London. gravity9 has been building TAP, its Technology Adoption Platform, with the Trust across multiple phases over 3.5 years.

The most recent phase carried the same delivery pressure as any long-running NHS engagement: a small team, a wide platform surface to keep tested, and pull requests and CI failures that needed fast, reliable triage without slowing the sprint rhythm down. Rather than treating AI as a productivity add-on for individual developers, gravity9 built AI tooling directly into the delivery pipeline itself, so it could act as a full participant in the loop between a failing build and a resolved, tracked issue.



Utilized Technology Stack

Cloud Platform: Microsoft Azure (App Services, Azure B2C, Blob Storage)

Frontend: React (TypeScript), Material UI (MUI)

Backend: Java (Spring)

Database: MongoDB

Authentication: Azure B2C (ROPC flow, MSAL token injection)

CI/CD & Infrastructure: Azure DevOps Pipelines, Terraform (Infrastructure-as-Code)

Test Automation: Playwright end-to-end test suite, plus API coverage

AI Test Generation & Repair: Claude Code

AI Code Review: CodeRabbit, integrated with Azure DevOps

Agent-to-Tool Integration: Model Context Protocol (MCP), connecting Claude Code to Azure DevOps, Jira, and Playwright

Review of Challenges

Delivering additional phases of TAP came with pressure spread across the whole team, not one discipline. Requirements needed to be refined into something unambiguous and testable before development could pick them up efficiently — a step that easily becomes a bottleneck when it relies entirely on manual back-and-forth.

On the development side, a small team had to keep a wide, actively evolving platform moving without accumulating technical debt, while every change still needed a careful human review before merge — a step that adds up fast against tight sprint timelines.

QA carried its own version of the same pressure: continuous end-to-end coverage for new feature work, constant hardening of existing tests against flakiness, and — the most acute pain point — CI triage. When a build failed, someone had to notice, read the logs, reproduce the failure locally, check whether it was already tracked in Jira, and write up a new ticket if not, a manual sequence that cost real time and attention every single day.

Across all three disciplines, the common thread was the same: valuable human judgment was being spent on repetitive, well-defined work that AI tooling was well suited to absorb.

Our Solution

AI tooling was adopted across the full delivery lifecycle on TAP, not as an isolated experiment for one team:

Product. Claude was used to help refine requirements into clearer, more testable specifications, and — connected to Jira via MCP — to create the resulting tickets directly, removing a manual handoff step between refinement and a tracked backlog item.

Development. Claude Code used the same Jira MCP integration to pull requirements straight from tickets, developed against them, and then used the Azure DevOps MCP integration to open the resulting pull requests — keeping the whole path from requirement to PR inside one tool-connected flow.

QA. This is where the approach reached its fullest maturity, closing the loop all the way from a code change to a verified, tracked outcome — and it's where the rest of this case study focuses in depth. QA used the same Jira MCP integration to pull requirements that needed test coverage, the Playwright MCP integration to help build and run the tests themselves, and the Azure DevOps MCP integration to open pull requests and trigger pipelines with those tests. Three capabilities were built on top of this foundation:

Test generation and repair. Claude Code was used to build new end-to-end coverage from scratch against real feature work, and to harden existing tests against flakiness and race conditions.

AI-driven code review. CodeRabbit was integrated with Azure DevOps to review every pull request automatically — line-by-line feedback, bug detection, security scanning, and a change summary, with a suggested fix and a ready-made prompt for an AI agent wherever it found a problem. Claude Code, connected through the same integration, pulls those suggestions, evaluates them, and applies the accepted fixes directly, closing the loop without a human copying a review comment between tools.

A self-triggering triage workflow. A single command, /triage-failure, defined in a markdown file checked into the repository, runs end to end with no arguments: it always finds the latest Azure DevOps build on its own, so no one needs to supply a build ID or any configuration.

Our Approach

Requirements refinement and coding acceleration were adopted as day-to-day working habits for Product and Development, woven into existing tools rather than built out as a standalone workflow. QA is where the team went a step further, engineering a repeatable, self-triggering process on top of Claude Code rather than relying on ad hoc prompting. The rest of this section walks through that workflow in detail.

The workflow behind /triage-failure runs six steps:

1. Build discovery, pulling the latest Azure DevOps build automatically.

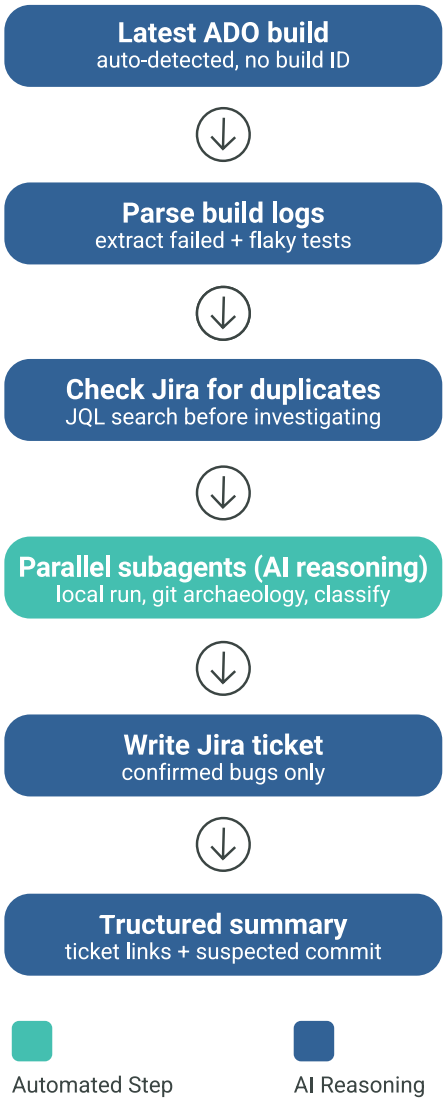
2. Log parsing, extracting every failed and flaky test from the shard logs along with its error and retry pattern. Tests that passed on retry are set aside as flaky; tests that failed every attempt move forward for investigation.

3. Jira deduplication, searching Jira via JQL for each failing test before any investigation starts, so the same failure never produces two tickets.

4. Parallel investigation, one subagent per failing test, launched concurrently. Each subagent reruns the test locally and headless, does git archaeology on the relevant spec and page-object files to find a likely cause, and classifies the result as a reproducible bug, a possible flake, or a CI environment issue. The local rerun is a hard gate: nothing reaches Jira until it completes.

5. Jira write, for confirmed bugs only — commenting on an existing ticket with the local run result and build link, or creating a new one.

6. Structured output, a summary with status, local run result, clickable ticket links, and the suspected commit per test.



None of this holds together without a shared reference the AI and the team both read from. CLAUDE.md acts as that living specification, holding the Jira ticket schema, authentication patterns, test helper conventions, and the triage workflow itself, so the workflow references one source of truth rather than duplicating it. Tool access is deliberately scoped through .claude/settings.local.json, allowlisting only the specific Azure DevOps, Playwright, and Atlassian actions the agent needs.

Subsequent Outcomes

On the Product and Development side, refining requirements with Claude and accelerating coding with Claude Code made day-to-day delivery smoother — clearer specifications reaching the team with less rework, and a small team able to keep pace with a wide, actively changing platform.

The QA workflow produced the most measurable results. The team estimates the manual sequence /triage-failure replaces — scanning CI logs, reproducing failures locally, checking Jira for duplicates, writing up tickets by hand — at roughly 10 minutes of effort per failing test, multiplied by however many subagents run in parallel on a given build. Eighteen commits in the project history show the AI-reviews, AI-fixes, human-approves cycle with CodeRabbit and Claude Code working end to end on real pull requests.

The team is still early in using this workflow and continues to refine it, with auto-triggering from Azure DevOps webhooks and a Slack summary step both under consideration next. What started as AI assistance for individual developers is now a repeatable, self-triggering part of the delivery pipeline itself.

Client Feedback

In gravity9, we found a development partner who goes beyond a transactional relationship. The culture of their team is to be curious and driven to solve problems. They have been quick to try to understand the vision behind the project and have put themselves in a position to be a valued collaborator. We've found the team to be easy to work with, efficient and diligent. They take their work seriously. rather than buil

> "A development partner who goes beyond a transactional relationship."

Visit our Insights page for more articles about emerging technology trends, interviews, and more!