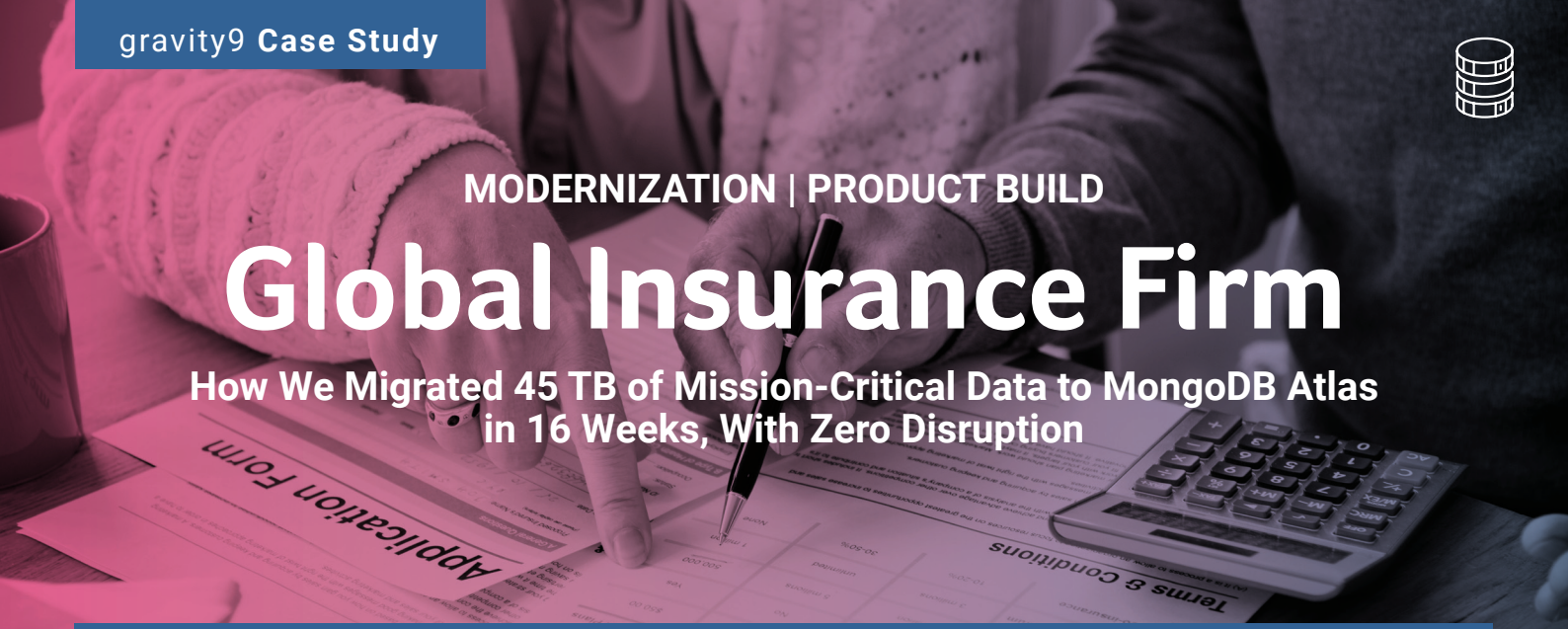


MODERNIZATION | PRODUCT BUILD

Global Insurance Firm

How We Migrated 45 TB of Mission-Critical Data to MongoDB Atlas in 16 Weeks, With Zero Disruption

Our client is one of the largest health insurers and wellness companies in the US, serving close to 15 million medical members through a nationwide network of health plans and services. It employs more than 67,000 people and generates upward of $129 billion in annual revenue, numbers that show how much depends on its technology staying up, fast, and secure.

At the center sat a 45 TB on-premises MongoDB cluster supporting a wide range of business-critical applications. As part of a broader cloud modernization push, the company decided to move this database infrastructure to MongoDB Atlas, seeking better scalability, stronger availability, and less manual infrastructure management.

The complexity lay in what depended on that cluster: thirty three applications, twenty five in Java, five in .NET, and three customized off-the-shelf systems, all connected directly to the on-premises database via LDAP authentication. Touch the database, and every application feels it. The client needed a migration approach that minimized application code changes, allowed controlled rollouts, and kept a rollback path open at every step. They also needed an automated way to validate application behavior and measure latency before and after the move, rather than take it on faith.

Utilized Technology Stack

Database: MongoDB Atlas
Migration: Mongo Sync
Backend: Java, Spring Boot
Configuration Management: HashiCorp Vault
Performance Testing: Apache JMeter
Regression Testing: Gatling
Utility Development: Java
Authentication: LDAP (source), Atlas Authentication

Review of Challenges

Moving 45 TB of live production data with minimal downtime is hard enough on its own. This project layered several more challenges on top of it. The team had to replace an existing LDAP-based connection method with a new SCARM configuration while keeping credential management secure. They had to validate a wide range of business APIs to confirm nothing broke in the switch, and measure MongoDB query latency on both sides of the migration to prove performance held steady or improved.

Some legacy application screens read directly from MongoDB Views with no API layer in front of them, which meant the usual testing tools could not reach them. And because this was never meant to be a one-off effort, whatever testing tools got built needed to work again on the client's next migration.

Zero downtime was the standard everyone worked toward, but zero was never guaranteed, so a safe, fast rollback mattered just as much as the migration itself. Every application needed to reach MongoDB the same way regardless of which cluster it pointed to, and every regression result needed to be trustworthy enough to stand as evidence in front of stakeholders who would ask hard questions about risk.

Our Solution

gravity9 built six connected pieces of work to get this migration done, and proven.

1. Large-scale database migration

The full 45 TB on-premises MongoDB cluster moved to MongoDB Atlas using Mongo Sync, with continuous synchronization between source and destination that allowed a controlled cutover instead of one high-risk cut.

2. A common Java MongoDB connection library

Built once and shared across all 25 Java applications rather than rewritten inside each one, the library centralizes connection management and pulls configuration dynamically from HashiCorp Vault. An application moves between the on-premises cluster and Atlas through a configuration change alone, so if something went wrong, rollback was a setting change, not a redeployment.

3. An automated regression testing framework

Built on Gatling, the framework sends configured requests, captures the responses, and compares them against expected results, flagging every mismatch in a detailed report, replacing what used to be repetitive manual checking.

4. MongoDB performance benchmarking

JMeter scripts covered every application API and ran against both the on-premises cluster and MongoDB Atlas, producing side-by-side comparisons of response time, database latency, and throughput, turning a claim of improved performance into measurable evidence.

5. An automated payload generator

A Java-based utility queries MongoDB directly, pulls production-like records, and builds matching request and expected-response files automatically, feeding both the regression and performance suites without manual upkeep.

6. A MongoDB View validation framework

Certain legacy application screens depended on MongoDB Views with no API in front of them. gravity9 built a lightweight Spring Boot application to query those views directly, measure execution latency, and compare results between on-premises and Atlas, closing the one gap the rest of the solution could not reach.

Our Approach

gravity9 split the engagement into four workstreams, run in parallel: infrastructure migration, application connectivity, functional validation, and performance benchmarking. Rather than treat 25 Java applications as 25 separate problems, the team built one abstraction layer for MongoDB connectivity and let every application inherit it. That single decision cut implementation effort sharply and put configuration and rollback control in one place instead of twenty-five.

Confidence mattered as much as code. Every migration decision was backed by an automated framework that generated production-like requests directly from the client's own data and compared results across both environments, so the case for moving forward was built on evidence the client could see, not a promise to take on trust.

Subsequent Outcomes

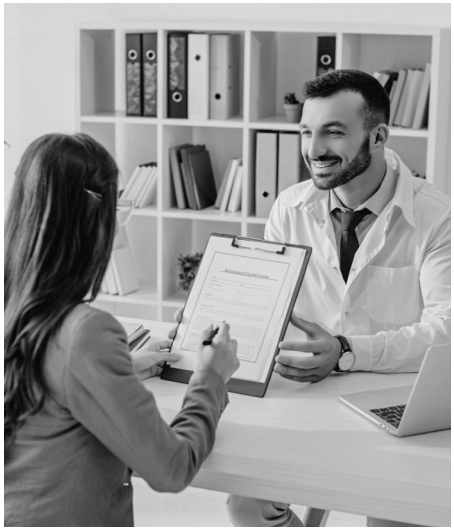
The engagement closed with the full 45 TB environment running on MongoDB Atlas and all 25 Java applications migrated through a single reusable connection library, backed by centralized, Vault-secured configuration and a rollback path built for speed. Automated regression testing, performance benchmarking, and a dedicated framework for validating MongoDB Views gave the client measurable proof, not reassurance, that functionality and performance held steady throughout.

The tools built for this migration did not retire with the project. The connection library, the regression and performance frameworks, and the payload generator are all reusable for the client's next database migration and for ongoing regression testing, extending the value of this 16-week engagement well beyond its original scope.

Client Feedback

The client's senior leadership expressed strong satisfaction with the migration following cutover, reporting a stable environment more than 24 hours in with no cutover-related issues and praising the team's dedication, coordination, and technical execution across a complex, multi-partner effort. The response called it a 'fantastic job all around' and an 'outstanding and very smooth migration,' pointing to the cutover as a major win for the broader modernization program and the culmination of plans dating back to 2022. Together, the feedback reflected a high degree of confidence and appreciation for the team's delivery.

> Rather than treat 25 Java applications as 25 separate problems, gravity9 built one abstraction layer for MongoDB connectivity and let every application inherit it.



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