Case Study

Vanguard

Monoliths to Microservices: Improving Code Quality with Graph Technology

INDUSTRY
Financial Services

USE CASE
Application Modernization

GOAL
React quickly to compete effectively, avoid disruption, and comply with regulations

CHALLENGE
Refactoring a massive production code base into microservices

SOLUTION
Used Neo4j to capture and analyze all relationships in legacy code and microservices

RESULTS
- Communicated and improved objective code quality metrics
- Reduced risk, performing impact analysis and enforcing best practices

Vanguard Group faced refactoring a jumble of critical legacy code into microservices, which increases the number of moving parts. By capturing all dependencies in the graph, the team progressively rationalizes their architecture and drives refactoring, collaborating across the business to improve code quality metrics derived using graph analytics.

The Company

Vanguard is one of the world's largest investment management companies, with more than $3.5 trillion under management. It is the largest provider of mutual funds and the second largest provider of exchange-traded funds (ETFs). With its 17,600 employees, Vanguard serves more than 20 million investors in about 170 countries. Its founder, John Bogle, is the creator of index funds.

The Challenge

Financial services companies require cutting-edge computing infrastructure. Refactoring Vanguard's existing monolithic Java-based systems into myriad microservices required visibility into all components and their dependencies. Some of Vanguard's legacy Java archives (jars) have **3 to 4 million lines of code.**

Further, Vanguard faced technical debt. Dead code needed to be pruned and impact assessments were difficult.

"Managing code is critical," said John Lavin, Software Architect, Vanguard. "Often the focus is on feature delivery, time to market and getting things out the door. But if you don't make it easy to manage code, none of that gets done."

The team first tried to visualize jar dependencies in a desktop tool. Attempts to load all the jars crashed the tool, which wasn't designed for Vanguard's scale.

Refactoring jars with a million lines of code into microservices requires managing a lot of moving parts. To tackle this, the team started by managing services in a spreadsheet, compiled over the course of a year.

"We had a lot of different services in a lot of different states. We tried to group things and document dependencies as best we could in the spreadsheet. But it really just wasn't going to work out long-term," said Lavin.

Vanguard needed to model all their services and dependencies in the graph for impact analyses that show how many other services will be impacted if a given service is down. API gateways that use the runtime to capture dependencies would not work for Vanguard, because key services called only at certain times of the year and would be omitted.

Case Study

"Managers go right to the dashboards to see whether their metrics are trending up or down. Graph technology gives us a great way of pulling these metrics up, and keeping our code clean because we're all looking at it."

– John Lavin
Software Architect, Vanguard Group

The Solution

The team needed a flexible solution that could scale to unprecedented levels. "We realized that the management of our modules and services was really a graph problem," said Lavin. Vanguard adopted Neo4j for its scale and flexibility.

The team started with a simple graph data model, adding in jar dependencies automatically when they ran a build. They then added all their existing code artifacts into the graph, along with their dependencies.

Using graph analytics, the team began evaluating their code against best practices and deriving metrics. Next they added in information from their architecture spreadsheet to enrich their schema.

The team built out tools to visualize relationships and enforce best practices such as constraining the number of service-to-service calls to reduce risk. Based on metrics derived from the graph, a code quality scorecard enables collaboration across the business, from management to developers.

The Results

Vanguard gained full visibility into dependencies between existing jars and microservices, enabling them to perform impact analyses and modernize their monolithic systems incrementally.

With all relationships and dependencies captured in Neo4j, the team is effectively addressing technical debt moving forward and demonstrating concrete improvements.

"With simple metrics, and a little bit of math, you can get a lot of module metrics using simple Cypher queries," said Lavin.

Vanguard visualizes the metrics derived from Neo4j in a code quality scorecard that grades each module. The scorecard is shared with the business, along with a spreadsheet-style visualization of the underlying graph. The business sees trends to gauge the progress of the application modernization initiative.

"Managers go right to the dashboards to see whether their metrics are trending up or down.

Graph technology gives us a great way of pulling these metrics up, and keeping our code clean, because we're all looking at it."

Architects use Neo4j not only to manage the current state of all of Vanguard's code but also to progress toward the desired state, maximizing the efforts of developers and promoting reuse.

"When a project team goes into their planning sprint, we provide information about which services to use, where those services are being used, what's partially built out and where developers may add their logic so that it makes sense in our overall architectural scheme," said Lavin.

Questions about Neo4j?

Contact us across the globe:
info@neo4j.com
neo4j.com/contact-us