Fallstudie

Vanguard

Von monolithischen Systemen zu Microservices: Mit Graphtechnologie die Codequalität verbessern

BRANCHE
Finanzdienstleistungen

USE CASE
Modernisierung von Anwendungen

ZIEL
Hohe Reaktionsfähigkeit für anhaltende Wettbewerbsfähigkeit, störungsfreien Betrieb und Compliance

HERAUSFORDERUNG
Refactoring einer massiven Produktionscodebasis in Mikroservices

LÖSUNG
Neo4j zur Erfassung und Analyse aller Beziehungen in Legacy-Code und Microservices.

ERGEBNISSE
- Verbesserung und Bereitstellung objektiver Codequalitätsmetriken
- Risikominimierung, Impact-Analysen und Durchsetzen von Best Practices

Der Finanzdienstleister Vanguard Group wollte diversen essenziellen Legacy Code durch Microservices ersetzen. Dabei musste die Vielzahl an Änderungen im Quellcode nachvollziehbar und testbar bleiben. Durch das Erfassen aller Abhängigkeiten im Graphen, rationalisierte das Team schrittweise die Softwarearchitektur und trieb das Refactoring voran. In unternehmensweiter Zusammenarbeit und mit Hilfe von Graph Analytics wurden die Qualitätsmetriken des Codes verbessert.

Das Unternehmen

Vanguard ist mit mehr als 3,5 Bio. Dollar eine der weltweit größten Investmentmanagementgesellschaften. Die Gruppe ist der größte Anbieter von Investmentfonds und der zweitgrößte Anbieter von Exchange Traded Funds (ETFs). 17.600 Mitarbeiter betreuen mehr als 20 Mio. Investoren in rund 170 Ländern. Gründer John Bogle gilt als Pionier im Bereich Indexfonds.

Die Herausforderung

Finanzdienstleister benötigen eine hochmoderne Computerinfrastruktur. Voraussetzung für das Refactoring bestehender, monolithischer Java-basierter Systeme und der Neustrukturierung in Mikroservices ist Transparenz über alle Komponenten und deren Abhängigkeiten. Einige der Legacy Java-Archive (Jars) von Vanguard verfügen über 3 bis 4 Mio. Codezeilen.

Eine Herausforderung war Technical Debt: Dead Code musste gesäubert werden. Folgenabschätzungen gestalten sich schwierig.

„Den Code zu managen ist essenziell," so John Lavin, Softwarearchitekt bei Vanguard. „Oft liegt der Fokus auf der Bereitstellung von neuen Funktionen und der rechtzeitigen Veröffentlichung. Fehlt es dem Code jedoch an Wartung, steckt man fest."

Der Versuch, Jar-Abhängigkeiten in einem Desktop-Tool zu visualisieren, scheiterte, da das bestehende System für die große Datenmenge nicht konzipiert war.

Das Refactoring der Jars mit Millionen von Codezeilen erforderte die Verwaltung zahlreicher Änderungen. Über ein Jahr hinweg trug das Team dazu die Services in einer Tabelle zusammen.

„Wir verfügen über viele verschiedene Services in mehreren Zuständen. Diese mitsamt ihren Abhängigkeiten in einer Tabelle zu gruppieren und zu dokumentieren, war keine langfristige Lösung", erklärt Lavin.

Um eine Impact-Analyse durchzuführen, mussten alle Services und Abhängigkeiten in einem Graph abgebildet werden. API-Gateways, die Abhängigkeiten über die Laufzeit erfassen, waren keine Lösung, da wichtige Services oft nur zu bestimmten Zeiten genutzt werden.

Fallstudie

„Manager gehen direkt zu den Dashboards, um den Zustand ihrer Metriken einzusehen. Die Graphtechnologie bietet uns eine großartige Möglichkeit, diese Metriken aufzurufen, zu prüfen und so unseren Code sauber zu halten."

– John Lavin
Software Architect,
Vanguard Group

Die Lösung

Gefragt war eine flexible hoch skalierbare Lösung. „Das Management unserer Module und Services ist eigentlich eine Aufgabe für Graphen", so Lavin. Vanguard entschied sich für Neo4j aufgrund seiner Skalierbarkeit und Flexibilität.

Das Team begann mit einem einfachen Graph-Datenmodell und fügte mit jedem Build automatisch Jar-Abhängigkeiten hinzu. Anschließend wurden Code-Artefakte samt Abhängigkeiten im Graph modelliert.

Graph Analytics ermöglichte es, Codes hinsichtlich Best Practices zu bewerten und Metriken abzuleiten. Informationen aus der Architektur-Tabelle ergänzten das Modell.

Zur Visualisierung von Beziehungen und der Durchsetzung von Best Practices (z. B. Begrenzung von Service-to-Service-Calls zur Risikominimierung) entwickelte das Team neue Tools. Aus dem Graphen abgeleitete Metriken ermöglichen Code Quality Scorecards für die unternehmensweite Zusammenarbeit.

Die Ergebnisse

Vanguard erhielt einen umfassenden Einblick in die Abhängigkeiten zwischen bestehenden Jars und Mikroservices. Der Finanzdienstleister konnte seine monolithischen Systeme modernisieren und Impact-Analysen durchführen.

Durch die Modellierung aller Beziehungen und Abhängigkeiten in Neo4j kann das Team effektiv technische Schulden beseitigen.

„Mit einfachen Metriken und ein wenig Mathematik erhalten wir über Cypher-Abfragen viele Modulmetriken", erläutert Lavin.

Vanguard visualisiert die von Neo4j abgeleiteten Metriken in einer Code Qualiy Scorecard, die jedes Modul bewertet und unternehmensweit zum Einsatz kommt. Die anschauliche Visualisierung in Tabellenform gibt Aufschluss über Trends und zeigt den Fortschritt der Modernisierungsinitiativen an.

„Manager gehen direkt zu den Dashboards, um den Zustand ihrer Metriken einzusehen. Die Graphtechnologie bietet uns eine großartige Möglichkeit, diese Metriken aufzurufen, zu prüfen und so unseren Code sauber zu halten."

Architekten verwenden Neo4j für die Verwaltung und Wartung des gesamten Vanguard-Codes. Gleichzeitig können sie weiter optimieren, die Arbeit der Entwickler maximal nutzen und Code effektiv wiederverwenden.

„Ein Projektteam im Planning Sprint erhält Auskunft darüber, welche Services zu nutzen sind, wo sie bereits genutzt werden, an welchen noch gearbeitet wird und wo Entwickler die Logik der Anwendungen implementieren können, um eine einheitliche Architektur sicherzustellen," so Lavin.

Fragen zu Neo4j?

Kontakt:
info@neo4j.com
neo4j.com/contact-us