Scalable Real-Time Graph Data Processing with Docker, Neo4j, Kubernetes, and Kafka

Sampada Nemade, Arizona State University, snemade1@asu.edu,

Abstract

Applications in social networks, transportation systems, and urban planning are made possible by graph-based data processing, which is essential for evaluating densely interconnected datasets. This project shows how to use Docker, Neo4j, Kubernetes, and Kafka to create a scalable pipeline for processing the NYC Yellow Taxi Trip dataset. In phase 1 of the project, the main goal was to build a Neo4j containerized environment to apply graph algorithms such as PageRank and Breadth-First Search (BFS). Phase 2 of the project included scaling of the pipeline using Kafka for real-time data streaming and Kubernetes for orchestration. The project provides insights into scalability, performance, and real-world use cases, demonstrating the smooth integration of various technologies for effective graph analytics

1. Introduction

Graph databases have transformed data analysis by providing efficient ways to handle interconnected datasets. Unlike traditional relational databases, graph databases such as Neo4j model relationships as first-class entities, enabling intuitive queries and faster traversal.

1.1. Neo4j

Neo4j is a specialized graph database that is optimized for storing, managing, and analyzing data with complex interrelationships, ensuring high performance in handling such interconnected datasets.

Unlike traditional relational databases, Neo4j uses a property graph model, representing data as nodes, relationships, and properties, which mirrors real-world connections and simplifies complex queries.[1][2] This approach allows Neo4j to provide blazingly fast performance, often surpassing relational databases by up to 1000x in traversal-based operations [1].

Its scalability is supported by advanced features like sharding and clustering, enabling organizations to handle vast datasets while maintaining high query speeds [1]. Neo4j integrates seamlessly with modern tools such as Apache Kafka, enabling real-time data processing and analytics [1][2].

1.2. Docker

Docker is an open-source platform designed to simplify the development, deployment, and operation of applications by packaging them into standardized units called containers. Containers are lightweight and portable, encapsulating the application code along with its dependencies, runtime, system tools, and libraries. This ensures consistency across different environments, making Docker ideal for solving the common issue of "it works on my machine" during software development .[3][4]

1.3. Minikube and Kubernetes

Minikube is a tool designed to quickly set up a local Kubernetes cluster on various operating systems. It allows users to run Kubernetes clusters in lightweight environments, such as virtual machines, containers, or even on bare-metal systems, making it an ideal tool for learning and development purposes [5][6].

1.4. Helm Charts

Helm Charts are packages for managing Kubernetes applications. A Helm Chart encapsulates the information necessary to deploy a Kubernetes application, including configuration settings, resource templates, and dependencies. These charts simplify the process of defining, installing, and upgrading even the most complex Kubernetes applications. [8]

1.5. Apache Kafka

Apache Kafka is a widely used open-source platform designed for distributed event streaming. It facilitates seamless data integration, supports streaming analytics, and enables the creation of high-performance data pipelines with minimal latency and high throughput. [7]. Kafka functions as a messaging system that decouples producers (data sources) and consumers (applications or systems) through a publish-subscribe model, ensuring fault-tolerant and durable data storage in streams known as topics [7].

2. Methodology

The project used containerization, orchestration, and real-time data streaming tools in a stepwise manner to create a scalable and effective graph data processing pipeline. This made it easier to integrate technologies like Docker, Neo4j, Kubernetes, and Kafka and guaranteed modular development.

2.1. Phase 1: Containerization with Docker and Neo4j

a. Environment Setup:

- A Docker container was created with Ubuntu as the base image. It included Neo4j 5.5.0, Python 3, and the Neo4j GDS plugin for running graph algorithms. This setup ensured a repeatable and isolated development environment[9] [10].

b. Dataset Preparation:

- The NYC Yellow Taxi dataset was cleaned and filtered to focus on trips within the Bronx, leveraging pandas for data preprocessing. Nodes represented pickup (PULocationID) and dropoff (DOLocationID) locations, while relationships captured trip metadata.

c. Graph Algorithms:

- PageRank: Evaluated the significance of nodes based on their connectivity, inspired by Google's original algorithm[15].
- BFS: Traveled from a source node to target nodes to identify shortest paths, reflecting geographical relationships between locations.[14]

d. Validation:

- Automated tests verified the data loading process and algorithm outputs, ensuring correct node and edge counts in the graph.

2.1. Phase 1: Containerization with Docker and Neo4j

a. Kubernetes Orchestration:

- Kubernetes was used to deploy Neo4j, providing persistent storage and stable network identities. Helm charts streamlined configuration, while Minikube simulated a local cluster environment.[10][11].

b. Kafka Integration:

- Kafka handled real-time data streaming, connecting producers (data sources) to Neo4j. Kafka-Connect transformed JSON messages into Cypher queries, automating data ingestion into the graph database.[11][12][13].

c. Enhanced Algorithms:

- BFS and PageRank were re-implemented on the updated dataset within the distributed pipeline. The outcomes confirmed that the system could manage dynamic data streams in real time.

d. Data Flow:

- In Phase 2, the project transitions into a distributed pipeline leveraging Kubernetes and Kafka to handle real-time data streaming and processing. The flow begins with the data from the NYC Yellow Taxi Trip dataset, which is pre-processed and streamed into the pipeline using Kafka producers. Each record contains information about trips, including pickup and drop-off locations, trip distance, fare, and timestamps.
- Kafka serves as the central messaging system that is efficiently transferring data between producers and downstream consumers. These messages are published to Kafka topics, organized logically to ensure separation of data streams if needed. Kafka's partitioning capabilities allow parallel processing, making the pipeline scalable and suitable for high-throughput applications**.**
- Neo4j, deployed within Kubernetes, acts as the graph database that ingests these messages. The Kafka-Connect Neo4j plugin transforms the incoming data into the query language of Neo4j, to create nodes and relationships in the graph database. For example, pickup and dropoff locations are represented as nodes, while trips between them are encoded as relationships with properties like distance and fare
- Kubernetes orchestrates the entire pipeline by managing the deployment and scaling of Kafka, Neo4j, and other services. Helm charts are used for ensuring persistent storage and consistent service endpoints, while Kafka brokers and zookeeper handle distributed messaging and topic management.
- As data flows through the system, Neo4j processes it with graph algorithms like PageRank and BFS to derive insights. The processed results can be streamed back to Kafka for further integration with visualization tools or analytics dashboards, completing the feedback loop. This architecture enables seamless, real-time graph-based analytics with high scalability and fault tolerance.

3. Results

a. Algorithm Outputs:

- PageRank identified high-demand locations, with top nodes reflecting popular taxi hotspots. BFS provided accurate traversal paths, demonstrating the model's ability to simulate NYC's transportation network.

b. Pipeline Performance:

- Kubernetes improved scalability and fault tolerance compared to the Docker-only setup. Kafka allowed for effective real-time streaming while preserving data consistency.

4. Discussion

This project demonstrates the use of various tools to process and analyze graph-based data at scale. In Phase 1, Docker was used to set up a reliable Neo4j environment, this enabled the implementation of graph algorithms like PageRank and Breadth-First Search (BFS). This also ensured consistency across different setups and simplified the handling of data transformations and schema design.[9][10]

In Phase 2, we used Kubernetes and Kafka to scale the pipeline and enable real-time data streaming. Kubernetes enhanced fault tolerance and resource optimization by orchestrating containerized applications, while Kafka facilitated seamless data flow between producers and Neo4j. The integration of these tools enabled automated and dynamic data processing [11]

The graph algorithms provided valuable insights into the NYC Yellow Taxi dataset, identifying key locations and paths within the transportation network. The project had challenges such as configuring Kubernetes services and optimizing Kafka streams.

Overall, this project highlights the potential of combining containerization, orchestration, and graph analytics for real-world applications. Future enhancements, such as deploying Neo4j in cluster mode and adding more algorithms other than BFS and Page Rank, could further improve its robustness and analytical depth.

5. Conclusion

This project effectively illustrated how to incorporate modern data engineering technologies to create a real-time, scalable graph processing pipeline. While Kubernetes and Kafka provided scalability and fault tolerance for managing dynamic datasets, Docker with Neo4j enabled effective graph analytics. By using graph algorithms like PageRank and BFS, the system demonstrated its capacity to extract valuable information from traffic data. These findings highlight the potential of such pipelines for practical uses, such as network analysis and smart city design.

6. References

1. Neo4j. "What's New in Neo4j 4.0." Neo4j Graph Database.
2. Neo4j. "Neo4j Product Brief." Neo4j Product Brief (PDF).
3. Docker. "Docker Overview." Docker Documentation.
4. Docker. "What Is a Container?" Docker Resources.
5. Minikube. "Minikube Documentation." Minikube Official Site.
6. Minikube. "Kubernetes 101 Tutorial." Minikube Kubernetes Tutorial.
7. Apache Kafka. "Overview of Kafka." Apache Kafka.
8. Helm. "What is Helm?" Helm Official Site.
9. Neo4j. "Graph Data Science Documentation." Neo4j GDS Library.
10. Neo4j Online Community. "Running Neo4j on Kubernetes." Neo4j Kubernetes Guide.
11. Eric-Waters. "Kubernetes and Kafka Integration." Eric-Waters Blog.
12. Neo4j. "Kafka Connect Plugin for Neo4j." Kafka-Connect Neo4j.
13. Neo4j. "Deploying Neo4j on Kubernetes with Helm." Neo4j Helm Charts.
14. Page and Brin. "The Original PageRank Algorithm." Backrub Algorithm.
15. Wikipedia. "PageRank." PageRank Overview.