Knowledge Graphs: Gaining Deeper Insight into Open-Source Data

CS4991 Capstone Report, 2024

Emmitt Sun James
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
gwu8ek@virginia.edu

ABSTRACT
Today, a vast amount of open-source data is generated every single second, presenting a significant challenge in its collection and management. To address this, my team and I implemented a knowledge graph based solution, providing a structured representation of entities and their complex relationships. The project involved implementing many AWS services including Amazon Neptune, a fully managed graph database service. This graph database allowed for efficient querying and analysis, allowing for a better understanding connections within the data. Initial results demonstrated improved data accessibility and enhanced pattern recognition. However, the project still has plenty of room for improvement with future work consisting of adding advanced data visualization features, adding a more advanced searching feature, and optimizing graph performance.

1. INTRODUCTION

Every day, an estimated 2 million blog posts are published on the internet (Puranjay, 2015). These posts, which fall under the umbrella of open-source data, are readily accessible. With all of this data sitting at our fingertips, my team and I were tasked with developing a solution to organize the data in an intuitive and easily navigable way.

In developing this solution, we sought to go beyond traditional relational databases, which can struggle to represent complex relationships in an intuitive manner. As a result, we decided to utilize graph databases to create a knowledge graph, allowing us to connect entites with each other via various relationships. This approach not only enhanced the structure of the data but also improved querying capabilities, allowing analysts to ask more nuanced questions and retrieve insights more efficiently.

2. RELATED WORKS

In analyzing the current data management landscape, Petkova (2024) discusses the advantages of knowledge graphs. One of the primary advantages is its ability to take in data from many different sources, and combine everything into a central access point in a consistent manner. Another advantage that is highlighted is the connection between the data and the real world that knowledge graphs provide. Unlike traditional databases, knowledge graphs provide much richer context, helping to capture inherent complexities of real-world data. The flexibility and efficiency of knowledge graphs is also highlighted, in that a knowledge graph allows for updating schema elements while also storing data in a cost-efficient manner

Upon deciding to move forward with a knowledge graph approach, my team and I had an important decision to make: should we use Amazon Neptune or Neo4j as our underlying managed graph database service. Kumar (2024) discusses the pros and cons of both services, highlighting the factors that might

lead a team to prefer one over the other. Neo4j the more established option in the space has superior community support along with a more stable experience. However, Neptune is better at scale and offers seamless integration with other AWS services. Its seamless integration with other AWS services what ultimately led us to choose Neptune over Neo4j.

In an article on the Neo4j website, a software company known for its extensive graph database management system, Bratanič (2022) presents a detailed breakdown of an information extraction pipeline that takes in text data and adds it to a knowledge graph. The presented pipeline, which includes steps for entity recognition and extraction, relationship extraction, and storing results in a knowledge graph overlaps greatly with my experience working on a knowledge graph based data pipeline.

3. PROJECT DESIGN

In this project, my team and I were tasked with designing a data pipeline and a web application from the ground up. The data pipeline was responsible for ingesting open-source data, processing it, and inserting it into a knowledge graph, a graph-based data structure where information is represented as nodes (entities) and edges (relationships). Meanwhile, the web application would query this knowledge graph and display the data in a user-friendly way. Since we were building this project from scratch, we had a great deal of creative freedom in our design choices, free from the constraints of an existing codebase.

3.1 DATA PIPELINE ARCHITECTURE

The data pipeline was architected entirely using various Amazon Web Services (AWS) services, a cloud computing platform that provides many different services for cloud-based data processing and storage options. Leveraging AWS allowed us to create a scalable and efficient pipeline, streamlining data ingestion and processing, while reducing the complexity of managing separate infrastructures. Furthermore, when the team had to select a managed graph database service to build our knowledge graph on, we decided to go with Amazon Neptune. This choice enabled seamless integration with the rest of our AWS-based pipeline, allowing for easier management and streamlined data flow.

3.2 DATA INGESTION & PROCESSING

The first major step in the data pipeline is data ingestion. Fortunately, an adjacent team had already implemented this functionality, so instead of building it ourselves, we accessed the data through an AWS Simple Storage Service (S3) bucket provided by their team.

Once the data is ingested, it undergoes a multi-step processing phase. Initially, the data is cleaned and standardized to ensure quality and consistency. Next, we performed entity and relationship recognition using DBpedia Spotlight, a natural language processing tool. This step identified and annotated key entities—specifically people, places, and organizations—while enriching these entities with additional properties and connections to other entities. This enrichment process was crucial for building a comprehensive and interconnected knowledge graph that would form the backbone of the web application's querying and visualization capabilities.

3.3 KNOWLEDGE GRAPH

Once processed, the data is transformed to align with a predefined structure, known as an ontology, preparing it for insertion into our knowledge graph. This transformation ensures that the data follows a structured format compatible with our Amazon Neptune graph database. Once transformed, the data is seamlessly loaded into Neptune, enabling the creation of relationships and insights within the knowledge graph that users can interact with through the web application.

We decided to utilize a knowledge graph in this project because of the unique advantages it has over more traditional data storage options. Knowledge graphs offer a more flexible and powerful approach to handling complex data relationships due to their graph-based structure. This structure allows for more natural representation of real-world relationships and can handle the dynamic and interconnected nature of open-source data (Kiff, 2024).

3.4 WEB APPLICATION

The web application includes a frontend (user interface) and a backend, creating a seamless user experience for interacting with the knowledge graph. The frontend allows users to explore and navigate the data intuitively, while the backend acts as a "data shuttle," efficiently retrieving and delivering relevant data from the knowledge graph to the user-facing frontend.

The design process began with rough sketches on a whiteboard to establish a shared vision for the user interface. Once the general design was agreed upon, we transitioned to Figma, a tool that enabled us to create quick mockups with real design elements. This rapid prototyping stage allowed us to experiment with multiple designs, which we then presented to the team for feedback. Each round of feedback led to further refinements in the mockups, ensuring the design evolved to meet our expectations. After finalizing the mockups, we began coding the frontend using TypeScript with the Angular framework and styled it with Tailwind CSS. To ensure the frontend was robust, we utilized the Jest testing framework to develop comprehensive tests, reinforcing the quality and reliability of the application.

While the frontend provides an interface for the users to use, it is unable to function without a backend to receive data from. The backend, written in Java via the Spring Boot framework, handles requests from the frontend via various endpoints for different purposes. Within each endpoint, the backend would query the knowledge graph in a specific way to get the exact data needed. For example, one of our endpoints was to retrieve all entities that were connected to a specific entity, which was specified by an entity ID parameter that could be passed in. These queries were written using Gremlin, a graph query language that streamlined querying the knowledge graph. To ensure the backend was always working properly, we implemented various unit tests using Spring Boot's built-in testing capabilities.

3.5 CI/CD PIPELINES

To ensure efficient and reliable deployment of our application, we implemented continuous integration and continuous deployment (CI/CD) pipelines using GitLab runners. These pipelines automated key stages of our development workflow, including building, testing, and deploying the application. Each code commit triggered the pipeline to run different tasks including linting, testing, and building. In doing so, we could automatically ensure that code quality was maintained, and potential issues were addressed promptly, before being pushed. Additionally, the pipelines streamlined the deployment process, allowing successfully tested builds to transition seamlessly into production environments with minimal manual intervention.

4. RESULTS

While there were still a few features we wanted to implement, my team and I successfully developed an application that achieved its core objectives. The application effectively and automatically ingested open-source data, integrated it into a knowledge graph, and provided users with an intuitive interface to explore and interact with the graph's interconnected data.

One of the highlights of the experience was the opportunity to present our project during the company's all-hands meeting in August. During the presentation, we demonstrated the core features of the application, explored some of the technical innovations, and shared details of our overall internship experience. This not only highlighted our technical achievements but also allowed us to showcase the collaborative effort and learning that went into building the project.

5. CONCLUSION

This project successfully addressed the challenge of managing vast amounts of open-source data through a knowledge graph-based solution built on AWS services. By combining an automated data pipeline and a knowledge graph with an intuitive web application, we created a system that effectively organizes complex relationships within open-source data. The project not only achieved its core objectives of data integration and accessibility but also provided valuable experience in working with modern technologies like AWS, Angular, and Spring Boot. The foundation we established offers a robust platform for future development and demonstrates an effective approach to handling the growing volume of open-source data.

6. FUTURE WORK

A key feature for future development would be the implementation of an interactive graph visualization feature. Such a feature would enable users to visually navigate through entity relationships, offering an intuitive understanding of the knowledge graph's structure. Users could interactively explore connections between entities, zoom into specific clusters of relationships, and discover patterns that might not be apparent otherwise. Amazon Neptune has a feature like this which we played around with, but we were never able to imtegrate it into our own application.

Something else we wished we could have done was to create our own entity extraction service. Although DBpedia Spotlight served our immediate needs for entity recognition and enrichment, developing an in-house solution would offer several advantages. A custom service could be tailored to our needs, improving the accuracy and relevance of extracted information. Additionally, this would eliminate a dependency on an external service, enhancing system reliability and providing greater control over the data pipeline as a whole.

REFERENCES

Bratanič, T. (2022, March 28). From Text to a Knowledge Graph: The Information Extraction Pipeline. *Neo4j*. https://neo4j.com/blog/text-to-knowledge-graph-information-extraction-pipeline/

Kiff, L. (2024, February 16). Exploring the Revolution: Graph Databases in Modern Data Management. *Tom Sawyer Software*. https://blog.tomsawyer.com/knowledge-graph-vs-graph-databases

Kumar, A. (2024, August 4). Neo4j vs. Amazon Neptune: Graph Databases in Data Engineering. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2024/08/neo4j-vs-amazon-neptune/

Petkova, G. (2024, January 26). Knowledge Graphs: Redefining Data Management for the Modern Enterprise. *Ontotext*. https://www.ontotext.com/blog/knowledge-graphs-redefining-data-management-for-the-modern-enterprise/

Singh, P. (2015). 2 Million Blog Posts Are Written Every Day, Here's How You Can Stand Out. *MarketingProfs*. https://www.marketingprofs.com/articles/2015/27698/2-million-blog-posts-are-written-every-day-heres-how-you-can-stand-out