Building a Distributed Data Processing Pipeline with Neo4j, Kafka, and Minikube

Saideepthi Korupolu
Master of Science,Computer Science
Arizona
skorupo1@asu.edu

1. INTRODUCTION

The processing of large-scale document streams in real-time poses a significant challenge for businesses that require timely insights from the data. The existing data processing pipelines may not be capable of handling such a high volume of data and processing it in a distributed and scalable manner. This limits the ability of businesses to gain insights from the data quickly and make informed decisions. The aim of this project is to create a data processing pipeline that is both highly scalable and highly available. It will need to be able to handle a stream of documents as input, perform a range of processing tasks on the data, and then stream it into a distributed Neo4j setup for near real-time processing and analysis. The pipeline must be capable of processing a substantial amount of data and executing graph-based algorithms such as PageRank and Breadth First Search on the data to generate significant insights. Additionally, the pipeline must be able to handle failures gracefully and be highly available to ensure continuous data processing even in case of failures.

2. PROJECT BACKGROUND

In order to undertake this project, I realized that there were certain knowledge areas that were necessary for me to possess. Firstly, I needed to have a basic understanding of Kafka and its functionalities, as this was a critical component for setting up the data pipeline. Secondly, I needed to have knowledge of Minikube and understand the architecture, deployment, and services of Kubernetes, as these tools were vital for setting up the orchestrator for the pipeline. In addition, it was crucial for me to be familiar with YAML configuration, as this is a human-readable data serialization format used for configuration files. Having a good understanding of YAML would allow me to complete the provided YAML files and set up Zookeeper and Kafka for the pipeline.

3. METHODOLOGY

The project is divided into 4 major Steps.

3.1 Step 0: The Network Whisperer

Understanding data flow: Before starting the project, I understood it is important to understand how data will be transmitted from the source to the destination.

Setup the environment: The project required installing the following packages: docker helm,Minikube, Kubernetes, Kafka, Neo4j to set up the environment.

3.2 Setting up the Environment

Orchestrator Setup for the pipeline: Setting up the orchestrator and Kafka for the pipeline involves configuring tools that enable the deployment and scaling of containerized applications, and a distributed streaming platform that allows real-time publishing and subscription to streams of records. This is done to create a data processing pipeline.

Use Minikube as the orchestrator: Minikube is used as the orchestrator to create a local Kubernetes cluster on a machine for development and testing purposes. It provides a simple way to set up the cluster.

Use Kafka to ingest data : Kafka offers a platform for handling real-time data feeds with high throughput and low latency.

Complete the YAML files:Completing the YAML files kafka-setup.yaml and zookeeper-setup.yaml enables the creation of a deployment and service, respectively, for Kafka. A deployment manages a set of identical pods, while a service provides a stable IP address and DNS name for accessing the set of pods.

```
(base) saideepthikorupolu@Saideepthi Phase-2 % kubectl apply -f ./kafka-setup.yaml

service/kafka-service created
deployment.apps/kafka-deployment created
```

Figure 1: Service and deployment creation of kakfa-setup.yaml.

```
(base) saideepthikorupolu@Saideepthi Phase-2 % kubectl get pods
NAME                                  READY   STATUS    RESTARTS      AGE
kafka-deployment-74b7f4d6bb-rt7x7     1/1     Running   2 (38s ago)   2m22s
zookeeper-deployment-799b4bf9b9-gbh2d   1/1     Running   0             19s
```

Figure 2: Output pods after service and deployment creation of kafka and zookeeper yaml files.

3.3 Step 2: Charting the Way Forward

Install and set up Neo4j in standalone mode: Neo4j is installed and set up in standalone mode on the Minikube cluster. This is done by downloading and installing the Neo4j Helm chart, which provides an easy way to deploy and manage Neo4j in a Kubernetes cluster.

Create a new YAML file named neo4j-values.yaml: This YAML file is used to configure the Neo4j Helm chart to set the password for the Neo4j database and to install the Graph Data Science library, which gives powerful graph algorithms for visualizing and analyzing graph data.

Create a deployment and service for Neo4j using the modified YAML files: Once the YAML files are created and configured correctly, a deployment and service can be created for Neo4j using the kubectl command-line tool.

3.4 Step 3: Neo4j -> [<3] -> Kafka

Create a new YAML file for connecting kafka and neo4j: I created a file named kafka-neo4j-connector.yaml that specifies the configuration and parameters of the Kafka Connect Neo4j instance and launches the veedata/kafka-neo4j-connect custom image.

```
containers:
  - name: kafka-connect
    image: veedata/kafka-neo4j-connect:latest
    imagePullPolicy: IfNotPresent
    ports:
      - containerPort: 8083
    resources:
      limits:
        memory: "2500Mi"
        cpu: "1000m"
```

Figure 3: Code Snippet to connect kafka with veedata image.

Explanation of the code to connect kafka and neo4j:

- The first part of the code describes the Service. It sets the metadata name to "kafka-connect" and specifies that the Service should select pods with the label "app: kafka-connect". It also defines a TCP port 8083 and maps it to the container port 8083.
- The second part of the code is a separator that separates the Service and Deployment definitions.
- The third part of the code describes the Deployment. It sets the metadata name to "kafka-connect" and specifies that the Deployment should select pods with the label "app: kafka-connect". It also sets the replica count to 1.
- The Deployment then defines a Pod template that specifies a container named "kafka-connect". The container uses the Docker image which has already been installed. It also maps the container port 8083 to the Pod's port 8083 and sets resource limits for memory and CPU.
- Overall, this code creates a Kubernetes Deployment and Service for a Kafka Connect instance that uses the image.

3.5 Step 4: PA To PB

Confirm all the previous steps are working: Before proceeding with this step, it is important to confirm that all the previous steps have been completed successfully. This involves checking that the Kubernetes cluster is up and running, and all the necessary components such as Kafka, Zookeeper, and Neo4j are properly configured and deployed. This is done by running commands to check the status of each component and verifying that they are running without any errors.

Run data_producer.py file: Once all the previous steps have been confirmed to be working, the next step is to run the `data_producer.py` file provided in the project. This file is responsible for generating sample data and publishing it to the Kafka topic. This data will be used to test the PageRank and BFS algorithms in the next step.

Implement PageRank and BFS: This step involves writing code in the interface.py file with the PageRank and Breadth First Search algorithms provided.

Expose the necessary ports: To run the BFS and PageRank algorithms on the data generated by the producer, it is important to expose the ports used by Neo4j and Kafka outside of the Kubernetes environment. To do this, the YAML files used to deploy the services are modified to include configurations that allow the ports to be exposed to the external network.

Verify the algorithms are working correctly by running them on the data generated by the producer: Finally, the PageRank and BFS algorithms need to be run on the data generated by the producer to verify that they are working correctly. This involves running the algorithms on the Neo4j database and analyzing the results. The output of the algorithms should be consistent with what is expected based on the input data.

4. RESULTS

When compared to program efficiency of Phase-1 approach and Phase-2 approach, the approach we used in Phase-2 gave a better understanding in visualizing the data and also in time.

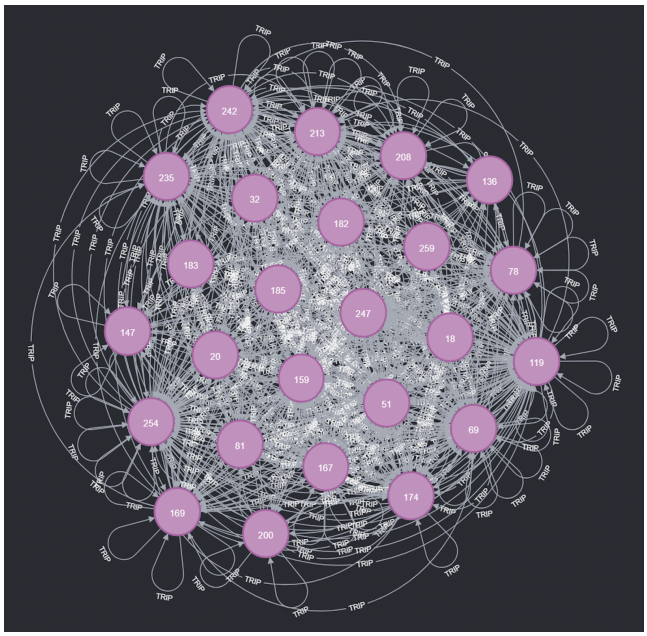

Figure 4: Output graph of Phase-1 approach implementation

In phase-2 after implementing all the command lines so that we can create a service and deployment of zookeeper, kafka and neo4j. I have run the following commands to check the status of the files.

Command line : kubectl get pods

```
(base) saideepthikorupolu@Saideepthi Phase-2 % kubectl get pods
NAME                                  READY   STATUS    RESTARTS      AGE
kafka-connect-f8b87bbbc-595g9         0/1     Pending   0             102m
kafka-deployment-74b7f4d6bb-rt7x7     1/1     Running   2 (104m ago)  106m
my-neo4j-release-0                    1/1     Running   0             103m
zookeeper-deployment-799b4bf9b9-gbh2d 1/1     Running   0             104m
```

Figure 5: Output after running the above command

Command line : kubectl get all

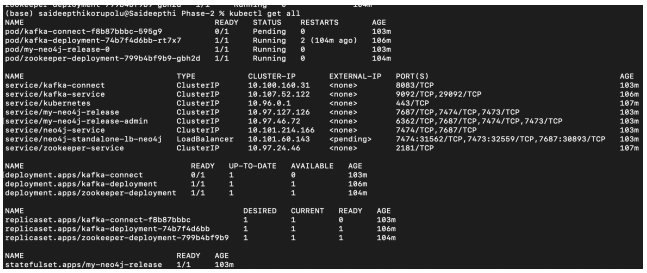

Figure 6: All the cluster information of the files

By running these commands we get all the status and running time and also information of the clusters and so more.

5. CONCLUSION

Neo4j and Docker for handling large amounts of data. While Neo4j and Docker can be effective in managing data, there are some drawbacks in terms of scalability and availability. For instance, as data grows larger and more complex, it becomes difficult to manage in a Docker environment. Additionally, if there are issues with the server or container, it can lead to downtime and data unavailability.

To address these limitations, other technologies such as Kafka and Kubernetes can be used. Kafka is highly scalable and fault-tolerant, meaning that it can handle high-volume data streams without any loss of data or performance. Kubernetes, on the other hand, is highly scalable, resilient, and portable, making it an ideal choice for managing large-scale applications.

By using Kafka and Kubernetes in Phase-2 of the project, the limitations of using Neo4j and Docker have been overcome. This would ensure that the data pipeline is highly scalable, resilient, and fault-tolerant, which would be crucial for handling large amounts of data in real-time. Additionally, it would ensure that the data is available at all times, even in the event of server or container failure. Overall, using Kafka and Kubernetes would enhance the efficiency and effectiveness of the data pipeline and enable the team to handle larger and more complex data sets.

The project also faced some challenges along the way, such as setting up and configuring the Kubernetes cluster and debugging the code for the algorithms. However, these challenges were overcome. Also by implementing two popular algorithms, Breadth-First Search and PageRank, to analyze the data generated by the data producer. These algorithms were able to extract meaningful insights from the data and provide valuable information that can be used for further analysis and decision-making.

Overall, the use of Kafka and Kubernetes in the data pipeline greatly enhanced the efficiency and effectiveness of the system, and allowed for the handling of larger and more complex data sets. This project provides a valuable example of how modern technologies can be used to manage and process large amounts of data in real-time, and the potential for these technologies to drive innovation and progress in various fields.

6. CONTRIBUTION AND LESSONS LEARNED

This is a solo project and while working on this project I have achieved some major results.

Achievements:

Successful integration of Apache Kafka, Neo4j, and Kubernetes, providing a scalable and fault-tolerant environment for data analytics.

Implementation of data analytics algorithms like PageRank and BFS on the provided dataset to generate useful insights.

Efficient use of resources like memory and CPU, ensuring smooth execution of algorithms on the Kubernetes environment.

Creation of a step-by-step guide for the project, including YAML files and bash scripts, making it easy to reproduce and deploy the entire system.

With achievements I also faced some challenges and as the project progressed I also learned some skills and knowledge to solve them.

Challenges and lessons learned:

Challenge 1 : Setting up and configuring the Kubernetes environment, including the installation of necessary tools like kubectl and minikube.

Lessons Learned :

- Understanding how to install and set up Kubernetes and its associated tools
- Learning how to deploy and manage containerized applications using Kubernetes

Challenge 2 : Configuring the Neo4j and Kafka services to ensure they communicate correctly, including setting the proper credentials and exposing the necessary ports.

Lessons Learned :

- Understanding the communication protocols and security measures required for two different services to communicate with each other
- Learning how to configure and manage different services within a complex data pipeline

Challenge 3 : Ensuring the data_producer.py file works correctly and generates data in the expected format.

Lessons Learned :

- Learning how to write code to generate data in a specific format
- Understanding how to test and debug code to ensure it is functioning as intended

Challenge 4 : Implementing the PageRank and BFS algorithms on the provided dataset, ensuring they generate useful insights without errors or bugs.

Lessons Learned :

- Learning how to apply data analytics algorithms to real-world datasets
- Understanding how to interpret the results generated by these algorithms and how they can provide useful insights

Challenge 5 : Exposing the necessary ports outside the Kubernetes environment to access the results of the data analytics algorithms.

Lessons Learned :

- Learning how to manage network ports and security measures in a complex data pipeline
- Understanding how to provide external access to the results generated by data analytics algorithms.

Overall, the project phase-2 demonstrated the power and flexibility of Apache Kafka, Neo4j, and Kubernetes in a data analytics setting. While there were challenges along the way, the achieved results make this project a valuable resource for anyone interested in learning more about these technologies and their potential applications.

7. REFERENCES

[1] Kafka : https://en.wikipedia.org/wiki/Apache_Kafka
[2] Neo4j: https://en.wikipedia.org/wiki/Neo4j
[3] Docker: https://en.wikipedia.org/wiki/Docker_(software)
[4] Kubernetes: https://en.wikipedia.org/wiki/Kubernetes
[5] Kafka : https://kafka.apache.org/documentation/
[6] Docker: https://docs.docker.com/
[7] Neo4j: https://neo4j.com/docs/
[8] Kubernetes: https://kubernetes.io/docs/home/
[9] Zhichao Cao Class lectures and slides