CSE 511: Data Processing at Scale

Amey Bhilegaonkar, *Arizona State University*
abhilega@asu.edu

Introduction

This project involves creating a highly scalable and available data processing pipeline using Kubernetes and Kafka, integrated with Docker and Neo4j. The pipeline will ingest a document stream and perform processing operations on it before streaming it into a distributed Neo4j setup for near real-time processing and analytics. The project is divided into two phases, with each carrying equal weightage. During the first phase of the project, we will focus on setting up the orchestrator and Kafka using Minikube. Kafka will be utilized to ingest data from the document stream and distribute it to other components of the pipeline. Moving into phase 2, we will integrate Neo4j into the setup, and the data will be streamed into Neo4j for further analysis and processing. The project involves a report submission and offers bonus grade points for additional work.

Methodology

The aim of this study is to propose a methodology for implementing Kafka and Neo4j as a service using Helm and Minikube in Kubernetes. The proposed methodology consists of the following steps:

1.1. Environment Setup

To begin implementing Kafka and Neo4j as a service, the initial step is to establish the appropriate environment. The initial step in implementing the Kafka and Neo4j services is to install Minikube, Kubernetes, and Helm. Minikube is a local Kubernetes cluster that provides the necessary environment for deploying and managing the services. On the other hand, Helm is a Kubernetes package manager that simplifies the installation and management of applications on the cluster.

1.2 Kafka Installation: This can be achieved by creating a Kafka cluster using the official Kafka Helm chart. The chart contains the necessary configurations required for a basic Kafka cluster.

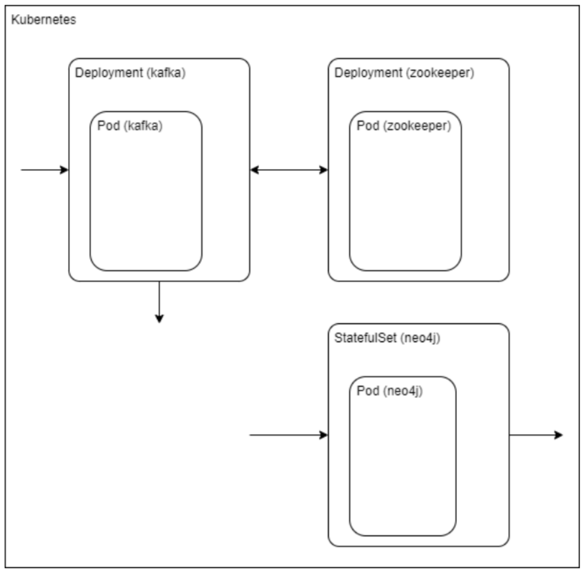
1.3 Data Ingestion: After the Kafka cluster is up and running, the next step is to ingest data into the Kafka cluster. This can be done using Kafka console producer or Kafka Connect. Kafka Connect provides a framework for data integration between Kafka and other data sources.

1.4 Neo4j Installation: After the data processing phase, the subsequent step involves storing the data in Neo4j. For this purpose, an official Neo4j Helm chart can be utilized to create a Neo4j cluster. The chart encompasses all the essential configurations required to set up a fundamental Neo4j cluster.

1.5 Testing and Deployment: Ultimately, to ensure proper functionality of the implementation, it is necessary to conduct rigorous testing. The testing phase should include simple tests such as sending a message over kafka-stream and receive it on the other end. Following successful completion of testing, the implementation can be confidently deployed into production.

2. Results

We will be making the architecture shown in the below logical diagram.



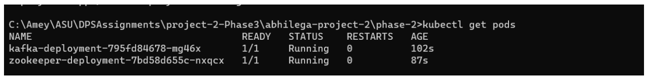
After the successful deployment of pods of zookeeper and kafka as a service we can test the deployments with following code.

Deployment:

1. kubectl apply -f ./zookeeper-setup.yaml
2. kubectl apply -f ./kafka-setup.yaml

Testing Commands:

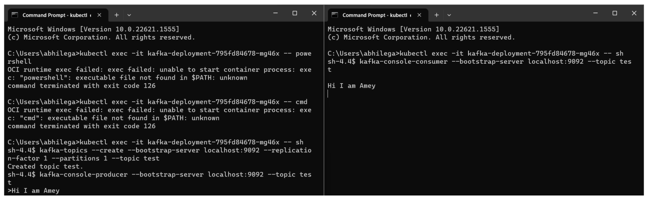
1. kubectl get pods



```
C:\Amey\ASU\DPSAssignments\project-2-Phase3\abhilega-project-2\phase-2>kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
kafka-deployment-795fd84678-mg46x      1/1     Running   0          102s
zookeeper-deployment-7bd58d655c-nxqcx  1/1     Running   0          87s
```

The running pods shows that our services are up and running. Now with the following commands we can forward the port and see if our Kafka broker is producing the messages and is it able to consume them or not:

1. kubectl exec -it kafka-deployment-795fd84678-mg46x -- sh
2. kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test
3. kafka-console-producer --bootstrap-server localhost:9092 --topic test
4. kubectl exec -it kafka-deployment-795fd84678-mg46x -- sh
5. kafka-console-consumer --bootstrap-server localhost:9092 --topic test
6. Hi I am Amey - Send - Terminal 1
7. Hi I am Amey - Receive - Terminal 2



```
Microsoft Windows [Version 10.0.22621.1555]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhilega>kubectl exec -it kafka-deployment-795fd84678-mg46x -- powe
rshell
OCI runtime exec failed: exec failed: unable to start container process: exe
c: "powershell": executable file not found in $PATH: unknown
command terminated with exit code 126

C:\Users\abhilega>kubectl exec -it kafka-deployment-795fd84678-mg46x -- cmd
OCI runtime exec failed: exec failed: unable to start container process: exe
c: "cmd": executable file not found in $PATH: unknown
command terminated with exit code 126

C:\Users\abhilega>kubectl exec -it kafka-deployment-795fd84678-mg46x -- sh
sh-4.4$ kafka-topics --create --bootstrap-server localhost:9092 --replicatio
n-factor 1 --partitions 1 --topic test
Created topic test.
sh-4.4$ kafka-console-producer --bootstrap-server localhost:9092 --topic tes
t
>Hi I am Amey
```

```
Microsoft Windows [Version 10.0.22621.1555]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhilega>kubectl exec -it kafka-deployment-795fd84678-mg46x -- sh
sh-4.4$ kafka-console-consumer --bootstrap-server localhost:9092 --topic tes
t

Hi I am Amey
```

Once the consumer receives the message, our testing can be said to be done.