Project-2 Phase-2 Bonus

Lei Hu

Introduction:

This project in the ASU of CSE 511 is a Bonus part of Project-2 Phase-2, which goal is to guide students to learn basic knowledge of Kubernetes, and how to install and deploy it in a cloud or experimental environment. In the last phase, we explore deploying neo4j in a local single Docker environment and writing our own Dockfile file to achieve automatically load data into the neo4j database when the Docker image is created. The disadvantage of that method is noticeable in that it requires a human setup and cannot be scalable easily. It is difficult to meet scenarios where daily demand and data volumes are particularly changing. Next phase, we overcome it by deploying all of the components in Kubernetes to stream data from Kafka controlled by Zookeeper into neo4j, and automatically manage them by Kubernetes.

1. Methodology

For me, this is a completely new plaything. I usually want to quick start for forming an overall general impression on it. Then, from each part of it, we can directly to build its environment to install it and test its command following the official documentation, because we already learn the basic concept in the course. And remember official documentation is always our best friend!

1.1. Minikube and Kubectl

Kubernetes is the keystone on which all other components are built on that. Minikube is a stand-alone version created by the official open-source community of Kubernetes and has almost the same functions and constructs. It is perfect for us to familiar with Kubernetes. And Kubectl is the command line tool which provide a large number of subcommands to facilitate the management of various functions in a cluster.

First. I start with basic commands after installing them through an official document. Also, “help” command is helpful in Linux systems. And then collect basic and commonly used commands into the same group. I set 5 groups in my notebook according to their function: creating, setting, exposing, and options. Structuring commands and plus options can be more convenient for us to use. Also, I find a large picture named Kubectl cheat sheet. It also divides all commands into different groups based on user level. I think it is too complex for us now, as a beginner, there are many commands you will not input.

1.2. Yaml

YAML is data-centric instead of markup-focused language. It uses blank or indent to describe relational data’s construct which is more readable and convenient to modify. Kubernetes uses the YAML format to describe a configuration of the pod. It is necessary to understand YAML and YAML constructs in Kubernetes. Another similar language: JSON, Kubernetes is used to communicate information between the pods. Not explored here.

Notice about YAML:

- Case sensitivity: In Kubernetes environment, it is only allowed to use lowercase.
- Strict format: Starting alignment at the same level is required. Usually with two blanks. Not support indentation and blank only.

I follow the demo of YAML provided by Kubernetes to learn about its construction. To understand the definition of each part of lines and modify it to fit my project. Then debug it or re-apply it. In particular, there are many different labels. It is essential to be able to be clear about the purpose of each.

1.3. Kafka

It is the first time I apply Kafka to my project. Kafka is binding with the Zookeeper, which is the brain for coordinating affairs, especially in distributed environment. Kafka pod may run fail because it needs the Zookeeper environment. After the Zookeeper pod start completely, it will run successfully, due to Kubernetes will run Kafka pod continuously. Here we focus on Kafka.

After creating the setup file and deploying Kafka in Minikube, we can use Kubectl exec command to go into the virtual environment of Kafka:

kubectl exec -it podname – bash

Then, we can practice using and test whether Kafka and Zookeeper work well. First, we create a topic with random replication and partitions and view the topic. After that, Producers send messages. Finally, Consumers receive messages. If the Consumers can print the messages, it proves to work well.

2. Discussion

Based on the picture, I know data transform into the stream by the source data and send to Kafka which is the function of data_producer.py. This process works by ip+port:9092. And zookeeper control Kafka by port:2181. The next stop how the Kafka pod sends data to Kafka connect pod is unknown to me. I guess the source in the Kafka connect pod can directly receive data from Kafka consumers. But I want to know the detail. This part I want to discuss.