High Scalability and High Availability Data Processing Pipeline

Aditi R Deshpande

Introduction

Software applications that process large amounts of data and user requests are designed for high scalability and high availability. The software application must retain its performance as the volume of data and/or the number of requests increase. This can be achieved by architecting the software application using docker images with software units comprising of an orchestrator such as Kubernetes or Minikube, data streaming pipeline such as Kafka and Zookeeper to keep track of the Kafka node statuses in the cluster. Docker containers instantiated from this image allow easy addition or removal of resources based on the load on the system.

1. Software units of the docker image

A docker image for this project contains software unit comprising of Minikube orchestrator, Zookeeper for coordinating the Kafka nodes and Kafka data pipeline that is instantiated as a docker container at run time.

1.1. Minikube

Minikube is a tool used to create a local Kubernetes environment on a local system such as a laptop. Though it is a Kubernetes distribution, it differs from Kubernetes in that it is supported by multiple OSes and is a single node cluster by default [1].

1.2. Zookeeper

Apache Zookeeper is used for synchronizing and keeping track of the nodes in the Kafka cluster. Zookeeper is required even if there is a single Kafka broker [2].

The Zookeeper service is accessed via the port 2181 configured in the zookeeper-setup.yaml file. The rum time zookeeper container is instantiated from the image confluentinc/cp-zookeeper:7.3.3 and this is also defined in the yaml file.

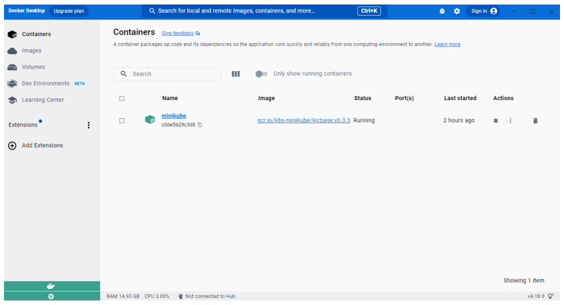
1.3. Kafka

Apache Kafka is an event streaming platform used in this project for high performance streaming of data. It is scalable, has high throughput and high availability [3].

The Kafka service is accessed via the port 9092. The other property name keys and values are defined in the kafka-setup.yaml file. The run time Kafka container is instantiated from the image confluentinc/cp-kafka:7.3.3 and this is also defined in the yaml file.

2. Steps for creation of environment

The file README.md provides the steps used for creating the environment. Fig 1 shows the container "minikube" running. Fig 2 shows the Minikube dashboard with the Zookeeper and Kafka deployment.



Containers

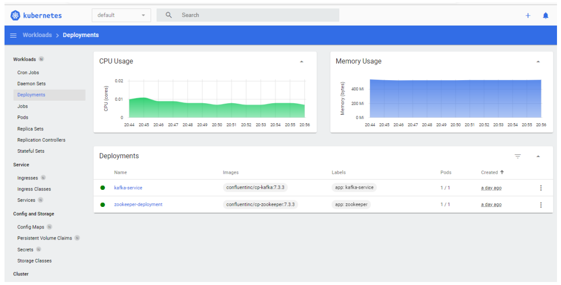
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Name	Image	Status	Port(s)	Last started	Actions
minikube c0de5b29c3d0	gcr.io/k8s-minikube/kicbase:v0.0.3	Running		2 hours ago	

Showing 1 item

RAM 14.93 GB CPU 3.00% Not connected to Hub v4.18.0

Figure 1: Docker – Minikube container <<link to google drive image file>>



Deployments

	Name	Images	Labels	Pods	Created
●	kafka-service	confluentinc/cp-kafka:7.3.3	app: kafka-service	1 / 1	a day ago
●	zookeeper-deployment	confluentinc/cp-zookeeper:7.3.3	app: zookeeper	1 / 1	a day ago

Figure 2: Minikube Dashboard - Deployment of Kafka and Zookeeper <<link to google drive image file>>

3. References

[1] What is Minikube?, accessed 24 April 2023, https://sysdig.com/learn-cloud-native/kubernetes-101/what-is-minikube/

[2] What is Zookeeper and how does it support Kafka?, accessed 24 April 2023, https://dattell.com/data-architecture-blog/what-is-zookeeper-how-does-it-support-kafka/

[3] Apache Kafka, accessed 24 April 2023, https://kafka.apache.org/