SWE 645- HOMEWORK-4

Group Members:

Rachana Thota - G01237600

Tuljasree Bonam - G01179672

Srujan Reddy Tekula - G01240653

Our urls-

http://34.82.142.175:8080/api/students - consumer

http://34.125.205.148:8080/publish - producer

http://tuljaswe645.s3-website.us-east-2.amazonaws.com/ - index page

http://35.228.214.116/create-studentForm - front end

YAML file github repository links:

https://github.com/rachana07/kafka-frontend - Repository link for front-end
https://github.com/rachana07/kafka-prod - Repository link for producer application
https://github.com/rachana07/kafka-consumer -Repository link for consumer application

Overview:

We created a Student Survey Data Broker that allows us to publish and consume student survey records at scale. For this, we have set up a Kafka cluster, created a Kafka topic, developed producer/consumer applications to publish and read messages to/from the topic, and integrated this solution with the previous assignment. For the implementation of a student data broker, we used Strimzi Kafka operator to set up a Kafka cluster with three Kafka brokers along with three Zookeepers on a Kubernetes cluster. Also, the data broker supports message ordering (first-in-first-out) on a per partition basis as per the requirement. We deployed brokers on the Kubernetes cluster and made it accessible using load balancer. The broker uses persistent storage to save student survey records. The persistent storage is implemented using persistent volume claim and storage class concepts of Kubernetes.

Instruction Steps:

1. Creation of Kafka Cluster and generation of Strimzi operator in Kubernetes environment-

We downloaded the Kafka cluster operator using the Helm charts 3 as mentioned in powerpoint slides. Unzip the downloaded tgz file and then install it using the kubectl commands. Wait until the cluster operator installs and a pod for the cluster operator starts running. Now Kafka persistent yaml file present in ppt slides or from GitHub links for Kafka persistent, then change yaml file according to your use case and then use kubectl apply. Wait for a few minutes until the Kafka cluster and zookeeper starts running. Kafka broker is present in the cluster and the later entity operator also starts running. While deploying Kafka cluster yaml file make sure you choose load balancer as an external service to connect from outside the server.

2. **Kafka on Google Cloud Platform**

We created a cluster on Google Cloud Platform and used the following commands to be able to use Kafka. Since we need Strimzi for Kafka, we added the Strimzi repository using this- ~ helm repo add strimzi http://strimzi.io/charts/ ~ . We have generated the Strimzi operator by helm installer with this command - ~ helm install strimzi/strimzi-kafka-operator --generate-name ~. We checked the Strimzi operator using - ~ kubectl -n namespace get all~. After this, we created a YAML file where we configured the number of ZooKeeper replicas as three and applied this to the cluster we created previously. We also checked the presence of Kafka pods as well as ZooKeeper pods using this command- ~ kubectl -n namespace get pods ~. After this, we have created a topic to test your producer and consumer application. Started producer application and then typed a message to topic and then, started consumer and made sure to mention from beginning to end of start command to consume messages from topic.

3. **Developing Spring boot Producer application**

We have started a new spring boot application with spring web service and spring Kafka service as initial dependencies. Then, we created a controller class in which we can pass through the Rest API with all necessary variables, constructors, getters, and setter methods. A configuration class is created in which we write a producer configuration which writes a message to a Kafka topic. This method uses key and data as two parameters. Before sending it, we need to mention the bootstrap server location and port number and then provide serializable classes for string and json. Then we have created another class which sends data to the topic and then we have mentioned the topic name here. First we have tested locally by producing a message to the kafka topic present in locally by running the spring boot application. After that, we connected to the topic present in kubernetes using the load balancer service and port created above in step 1.

4. **Developing spring boot Consumer Application**

Similar to Spring boot Producer application, we have started a new spring boot application and selected JSONObjectify, Spring web service and Spring kafka service as initial dependencies.
Now we have written a class (bean object) in which you can pass through the Rest API with all necessary variables, constructors, getters and setter methods. After that, we have created a

configuration class in which we write a consumer configuration which consumes a message from a kafka topic. This method uses key and data as two parameters. Before sending it to the topic we need to mention the bootstrap server location and port number and then provide deserializable classes for string and json. Then we have created another class which consumes data from the topic. We have mentioned the topic name and group id here. First tested it locally by consuming a message from a kafka topic present locally by running the spring boot application. Then we have connected to the kafka topic present in kubernetes using the load balancer service and port created above in step 1.

5. **Deploying producer and consumer applications in kubernetes**

We have deployed producer and consumer spring boot applications in Kubernetes using gcp cloud platform. Firstly write a docker file for the producer application and then run the application using maven install. This then generates a jar file and using docker creates an image and pushes the producer application to docker hub. Now we have written a yaml file for deploying spring boot applications in kubernetes and make sure you use a load balancer so as to connect to this application from outside kubernetes. Repeat the same steps for consumer application and now make sure you can produce and consume messages. Now, we got endpoints for producer and consumer applications which we use in setting up front end angular applications.

EndPoint for producer: http://34.125.205.148:8080/publish

Endpoint for consumer: http://34.82.142.175:8080/api/students

6. **Setting up Producer and Consumer Applications with Angular.**

In the angular application we need to change the services typescript file with previous links to current producer and consumer app links using load balancer service. We need to use POST Mapping in the producer app to produce messages from angular applications and send data to kafka topic. For that we need to use GET Mapping in consumer apps to consume messages and send it to angular applications. We have used an Arraylist in consumer apps to store messages from kafka topics and then we returned the array list of type students using get mapping to the angular application. Now we have deployed an angular app in kubernetes as we did in homework 3 using a load balancer.Now using load balancer URL and Port of angular app we can fill the student survey form and send data to kafka topic and consume it again by showing entries in list.
Once the application is running locally, we have deployed the application in Kubernetes using gcp cloud platform.

Endpoint for front-end application: http://35.228.214.116/create-studentForm

7. **Created a index page to add the link of our application:**

Using aws s3 service, we have updated the index page by adding assignment 4 link where we can route to the front-end application.

Endpoint for index page: http://tuljaswe645.s3-website.us-east-2.amazonaws.com/

8. **Yaml files**

We have created a github repository in which we created yaml files for spring-boot-producer, spring-boot consumer and front-end applications. Using these urls, we have deployed the respective repositories in the Kubernetes.

Repository link for front-end: https://github.com/rachana07/kafka-frontend
Repository link for producer application: https://github.com/rachana07/kafka-prod
Repository link for consumer application: https://github.com/rachana07/kafka-consumer

Team Contribution:

Kafka Cluster and generation of Strimzi operator in Kubernetes environment **- Tuljasree Bonam**
Creation of spring boot producer and spring boot consumer application and deploying it in Kubernetes - **Rachana Thota**
Creation of front-end application and deploying it in Kubernetes - **Srujan Reddy Tekula**
Documentation and index page - **Rachana Thota and Tuljasree Bonam**

Links and References:

- **https://strimzi.io/docs/operators/in-development/quickstart.html**
- **https://dzone.com/articles/kafka-on-kubernetes-the-strimzi-way-part-1**
- **https://medium.com/@contactsunny/simple-apache-kafka-producer-and-consumer-using-spring-boot-41be672f4e2b**
- **https://developer.okta.com/blog/2019/11/19/java-kafka**
- **https://github.com/strimzi/strimzi-kafka-operator/tree/master/examples**
- **https://github.com/strimzi/strimzi-kafka-operator**
- **https://kubernetes.io/docs/concepts/workloads/controllers/deployment/**
- **https://strimzi.io/blog/2020/04/15/develop-apache-kafka-applications-with-strimzi-and-minikube/**
- **https://www.youtube.com/watch?v=GSh9aHvdZco**
- **https://www.youtube.com/watch?v=KEPB7iG5Fgc**
- **https://strimzi.io/docs/operators/latest/deploying.html**