Report of TweetoScope

Ilyas Moutawwakil, Hao Wang, Youness Laklouch

December 13, 2022

1 Introduction

In this project, we will create a java application displaying the trending hashtags in real-time from twitter using **Kafka** middle-ware to optimize the communication between different services. Then we want to deploy this application using **Docker** for containerization and **Kubernetes** for orchestration. Finally, we will use CI/CD to optimize the development process.

2 Secrets and Git

In this project, one needs a **BEARER_TOKEN** from the **Twitter API** for developers. Since its confidential and each member/user needs his own token to access the Twitter stream (using the same token simultaneously is impossible), we decided to make it an environment variable. The token in this case can be hidden, inferred from environment or stored in environment files. In all cases, it's not a good idea to push personal data to the Git repository or to hard code variables that change depending on the context (person, time,...).

3 Architectural Choices

In the architecture that we've built, the stream of tweets is received from the twitter API then recorded in a **Kafka Topic : Tweets**. One single partition is used here because we only receive 1 percent of publicly available Tweets in real-time which are randomly sampled and also there is just one worker.

For the purpose of communicating between topics and finally sending Tweet information to the final Kafka consumer, our visualizer, we need to create a **Serializer** and **Deserializer** adapted for Tweet instances. Here, we use JSON-formatted string as information carrier with special date-time adapter.

Then, we use Kafka Stream Services to filter out all the Tweets received from the previous topics based on certain rules like the language of Tweet, country where it has been posted, Tweet length, etc... This Kafka Service called **Filter** in the architecture, contains also : a hashtags extractor and a hashtags counter to rank the most frequent hashtags in real-time then send them to the next topic **Filtered-Tweets** where only count information is recorded.

Finally, a visualizer to display the trending hashtag or topics on twitter is at the end of our solution.

4 Risk Analysis

In this section, we will discuss the impact of failure occurred on different component. Different components work separately. So if any failure occurs, the whole application would hang instead of crashing :

- If the Stream crashes, the other two (Filter and Visualizer) would still be working but of course they won't change anymore, since there are no more tweets coming their way
- If the Filter crashes, same thing the other two would be running but the Visualizer would show erroneous results.
- If the Visualizer crashes, nothing would change in the other two.

5 Risk mitigation using Kubernetes

From what we can see in the figure below, when we delete the filter-deployment-694b4594b7 pod running on ic45, we see that Kubernetes is sticking to the deployment we described and it creates a new filter-deployment-694b4594b7 pod running on ic22.

```
cpusdi1_25@ic25:~$ kubectl get pods -o wide
NAME                                   READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
filter-deployment-5f85f57fd7-x9kpq     1/1     Running   0          3m10s   10.2.42.16   ic45   <none>           <none>
kafka                                  1/1     Running   0          7m27s   10.2.16.16   ic13   <none>           <none>
stream-deployment-694b4594b7-49btk     1/1     Running   0          3m10s   10.2.30.13   ic30   <none>           <none>
visualizer-deployment-7dffb4fd67-g22tk 1/1     Running   0          3m10s   10.2.28.10   ic20   <none>           <none>
zookeeper-pod                          1/1     Running   0          7m27s   10.2.24.12   ic34   <none>           <none>
cpusdi1_25@ic25:~$ kubectl delete pod filter-deployment-5f85f57fd7-x9kpq
pod "filter-deployment-5f85f57fd7-x9kpq" deleted
^Ccpusdi1_25@ic25:~$ kubectl get pods -o wide
NAME                                   READY   STATUS              RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
filter-deployment-5f85f57fd7-xsx85     0/1     ContainerCreating   0          53s     <none>       ic22   <none>           <none>
kafka                                  1/1     Running             0          8m47s   10.2.16.16   ic13   <none>           <none>
stream-deployment-694b4594b7-49btk     1/1     Running             0          4m30s   10.2.30.13   ic30   <none>           <none>
visualizer-deployment-7dffb4fd67-g22tk 1/1     Running             0          4m30s   10.2.28.10   ic20   <none>           <none>
zookeeper-pod                          1/1     Running             0          8m47s   10.2.24.12   ic34   <none>           <none>
cpusdi1_25@ic25:~$
```

6 what we have learned by completing this project

In this project we learned how hard to make my Kafka Application scalabale and fault tolerant using DevOps. We also learned that code is very fragile and instable and one little change in two developers environments can make it work on one machine but not on the other. And that's exactly the kind of questions we solved with CI/CD.

7 Which parts were the most difficult and why ?

The most difficult parts were the ones that deviated from what we did in Tutorials like X11 forwarding which we didn't know how to make it work but in the end we found a solution.