Tweetoscope

Aaron BRODERICK Olivier HAZARD Julien MICHEL

December 11, 2023

1 Secret and git

It may not be a good idea to put the token in the git repository. Indeed even if our repository may not be accessible at first if we put it in private mode, we think it may still be scrapped by people who would try to get some API tokens. However it depends on the reliability of the gitlab private mode: if it is really reliable, it should not be a problem to put the token on gitlab. As our repository is in public, it would necessarily be a bad idea to put the token in the repository.

2 Architectural Choices

2.1 Global architechture

In our architecture, a topic `Tweets` comes first to gather the tweets using the source file (or the API). Then, we can use a `consumer group` to filter, first of all, tweets' text. We can also gather further information such as the language in order to filter more precisely the tweets. This way, we gather the filtered tweets in a second topic that we call `FilteredTweets`. Finally, we use a service to gather the hashtags in a last topic `hashtags`. Then, we can use a counter like the `Aggregator` from the first labwork to get the number of occurrences of each hashtag.

2.2 Structure of the exchanged messages (JSON formatted)

`Tweets` topic:

- key: `void`
- value : `Tweet` object similar to JSON format that contain all data

`Filtered tweets` topic

- key: `void`
- value : `Tweet` object similar to JSON format that contain all data

`Filtered tweets` topic

- key: `void`
- value : `String` object containing the hashtag

3 Risk Analysis

3.1 Initial Risk Analysis

Since we have 2 brokers for our 2 partitions for each topic, the crash of 1 broker would only slow down the application. if 2 of them crash at the same time for the same topic, the application is hanging and the some results could be lost during the issue.

If a consumer or a producer lags or crashes, it can cause the application to lose data or to slow down.

3.2 Risk mitigation using Kubernetes

In the end, we only managed to make the project work with one broker when using Kubernetes. We use an infrastructure pod to service similar to the one used in the lab-work 3B. When we kill a pod in a deployment, it rebuilds itself after a short time. It's the expected behaviour when using deployment and it allows our service to be resilient.

4 Conclusion

4.1 Lessons from the project

By completing this project, we learned how the development of a service works from its minimal version to the deployment in production (here on the DCE). We've also learned how to use Kafka, Docker and Kubernetes and how to do continuous integration with git. What's more, we've learned how to cope with difficulties to went in front and that we will detail in the next section.

4.2 The difficulties

The first difficulty that we had was how to exploit Kafka to change the structure of the service and especially the understanding of the serializers and deserializers to send and get the tweets in the topics. Then there were about the same difficulties between Docker and Kubernetes : no graphical interface for the visualisor, problems when trying to use 2 brokers and difficulties to use environment variables.