Rapport

Analyse de mots

Introduction

Le but de ce TP est de nous initier aux thèmes suivants :

- Échange de messages
- Le cluster RedPanda
- Producteur / Consommateur
- Utilisation de conteneurs Docker

Nous allons développer un système permettant d'extraire les mots des phrases et comptabiliser leurs occurrences dans un site. L'affichage du site sera dynamique, et à chaque fois que l'ordre d'occurrence des mots est modifié, l'arrangement des informations devra le représenter.

Implémentation

Étape 1

Cette première étape débute par la récupération d'un projet sur GitHub à l'adresse https://github.com/laurentgiustignano/prod-red-panda. Ce projet représente notre 'producteur' et contient plusieurs fichiers de configuration tels que *Dockerfile*, *.env*, ainsi que des sources.

Tout d'abord, nous allons adapter la ligne 24 de la fonction *getStringMessage()* dans le fichier *server.js* afin d'utiliser la variable *NUMBER_WORD* définie dans .env plutôt qu'un chiffre magique (le 3). Pour cela, il est nécessaire d'effectuer plusieurs modifications dans config/config.js ainsi que dans server.js

```
// Ajout d'une nouvelle fonction dans config.js
export const getNumberOfWords = () => {
    return process.env.NUMBER_OF_WORDS || 3;
}
```

```
// Modifications dans server.js

// Imports
import {getConfigNumber, getDebug, getNumberOfWords, getTimeOut,
getTopic, getTypeMessage} from "./config/config.js";

// Constantes
const numberWord = getNumberOfWords()

// Fonction start
async function start() {

    console.log(`Creating topic: ${topic}`)
    await Admin.createTopic(topic)
    console.log("Connecting...")
    const timeRetour = getTimeOut()

    setInterval(() => {
        const user = getUser()
        const message = typeMessage === "texte" ?
            getStringMessage(numberWord) :
            typeMessage === "nombre" ?
                getNumberMessage(configNumber) :
                "Config Error : choisir 'nombre' ou 'texte'"

        Producer.getConnection(topic, user, message)
        if (debug) {
            console.log({topic, user, message})
        }

    }, timeRetour)
}
```

Ensuite, il fallait modifier le fichier *.env*, et plus particulièrement la variable *HOST_IP* afin que le producteur puisse “trouver” notre cluster RedPanda.

```
// Changement effectué
HOST_IP=redpanda-0
```

Une fois cette modification terminée, il est possible de construire notre image docker du Producteur et démarrer le conteneur. Nous pouvons observer des messages de debug dans la console :

Extrait de la console d'exécution

Étape 2

A présent, il faut mettre en place le service qui va s'abonner à notre topic et récupérer l'ensemble des phrases. Pour une première approche, il sera suffisant d'afficher dans la console les messages du broker,

Pour cela, nous allons créer un nouveau projet, et en s'inspirant du producteur, écrire une fonction *connexion()* qui s'abonne au topic '*mon-super-topic*', qui affiche un message dans la console afin de valider la connexion.

Grâce à la fonction *run()* (de la documentation de Kafka.js.org) permettant de consommer les messages, il est possible d'en afficher le contenu.

```
// Dans consumer.js
export async function connexion() {

    try {
        await consumer.connect();
        await consumer.subscribe({topic: topic});

        await consumer.run({
            eachMessage: async ({ message}) => {
                console.log({
                    value: message.value.toString(),
                    date: convertTimestamp(message.timestamp),
                })
            },
```

```
        })
    } catch (error) {
        console.error("Error:", error);
    }
}
```

Pour finir, le timestamp fourni avec chaque message n'est pas directement interprétable. Ainsi, j'ai créé une fonction *convertTimestamp* dans un fichier utils.js pour convertir l'affichage avec le format suivant : dd/mm/aaaa à hh :mm.

```
export const convertTimestamp = (timestamp) => {
    let date = new Date(+timestamp);

    return date.toLocaleString('fr-FR', {
        day: '2-digit',
        month: '2-digit',
        year: 'numeric',
        hour: '2-digit',
        minute: '2-digit',
        hour12: false
    });
}
```

Étape 3

Avant de relier le service à l'affichage, il faut d'abord découper les messages lus dans le topic en mots. Ensuite, il faut compter les occurrences de chaque mot pour l'afficher dans la page web utilisateur.

Pour cela, nous allons utiliser une image docker Redis. Ce service Redis est couplé avec un service Web écrit en React qui permet l'affichage, le tout récupéré à l'adresse https://github.com/laurentgiustignano/occurence-mots.

À l'aide de la bibliothèque *redis* pour *node-js*, il fallait créer les fonctions permettant de se connecter au serveur Redis et effectuer l'incrémentation avec la commande INCR.

Code :

```
import {Kafka} from "kafkajs";
import {getConfigTopic, getLocalBroker} from "../config/config.js";
import { convertTimestamp } from "./utils.js";
import { createClient } from 'redis';
```

```
const isLocalBroker = getLocalBroker()
const redpanda = new Kafka({
    brokers: [
        isLocalBroker ? `${process.env.HOST_IP}:9092` :
'redpanda-0:9092',
        'localhost:19092'],
});

const consumer = redpanda.consumer({groupId: "redpanda-group"});
const topic = getConfigTopic();

const redisOptions = {
    url: "redis://myredis:6379",
    password: "redispwd"
};

const redisClient = createClient(redisOptions);

async function incrementation(mot) {
    await redisClient.INCR(mot);
  }

export async function connexion() {

    try {
        await consumer.connect();
        await consumer.subscribe({topic: topic});

        await consumer.run({
            eachMessage: async ({ message}) => {
                console.log({
                    value: message.value.toString(),
                    date: convertTimestamp(message.timestamp),
                })

                if(message.value) {
                    const words = message.value.message.split(' ');
                    words.forEach(async mot => {
                        await incrementation(mot);
                    });
                }
            },
        })
    } catch (error) {
        console.error("Error:", error);
```

```
    }
}
```

Désormais, il est possible d'accéder au site web de notre nœud d'affichage et ainsi observer la liste des mots évoluer.

Conclusion

Pour conclure, ce projet nous a permis d'apprendre à faire échanger des messages entre un Producteur et Consommateur. De plus, nous avons pu approfondir nos connaissances sur l'utilisation de conteneurs Docker ainsi que le cluster RedPanda.