ÉNONCE TRAVAUX PRATIQUE – SUJET 3

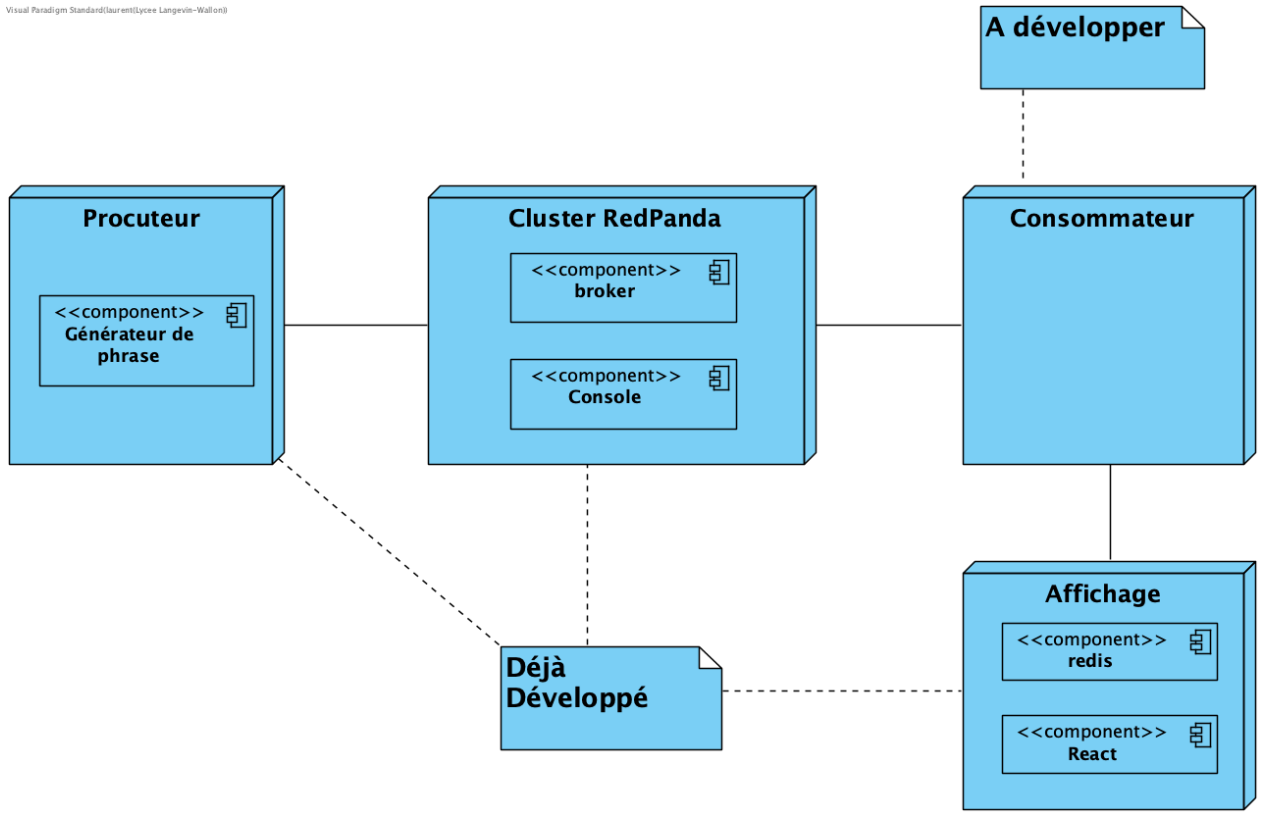
Analyse de mots

Thèmes

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

Présentation

Pour mettre en évidence les modalités d'échange par messagerie, 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. Le système est conçu en plusieurs services :



Le nœud « producteur » est conçu pour générer des données aléatoires et les envoyer dans le **broker RedPanda**. Il est utilisable pour produire des nombres ou bien des phrases. C'est cette dernière option que nous utiliserons.
Le nœud « **RedPanda** » est un cluster qui recevra, sur un Topic prédéfini, les données générés par le Producteur.
Le nœud « Consommateur », qu'il faudra développer durant la séance, devra s'abonner au Topic, recueillir les données produites et les séparer en mots qui devront être comptabilisé.
Le nœud « Affichage » est composé d'un service Redis qui s'occupera de comptabiliser les occurrences des mots, et d'un service **React** pour afficher en direct l'évolution des valeurs.
Les services existants sont à récupérer directement sur des dépôts GitHub aux différentes étapes d'avancement du TP.

Ressources indispensables :

- https://docs.redpanda.com/current/get-started/quick-start/
- https://kafka.js.org/docs/getting-started
- https://redis.io/docs

Étape 1 – « Vive les petits producteurs »

La première étape consiste à récupérer le dépôt du producteur sur GitHub. Vous pouvez récupérer le projet avec la commande **git clone https://github.com/laurentgiustignano/prod-red-panda** dans votre répertoire de travail. Ou bien depuis **PhpStorm**, en choisissant : **Get from Version Control** avec la même URL.

Le projet contient :

- **ReadMe.md** qui décrit les différentes options de configuration.
- **.env** qui contient les différents options de configuration pour que le producteur corresponde à notre besoin.
- **Dockerfile** qui permet la génération de l'image Producteur. En théorie, il n'est pas nécessaire de le modifier.
- **Des sources** qui contiennent la logique du producteur de messages, la connexion à un cluster RedPanda et le mécanisme de configuration.

Il faut prendre ses marques avec ce code qui n'est pas le vôtre pour comprendre la logique, adapter le fichier de configuration **.env**, puis créer l'image docker et la lancer. Par défaut, le projet fonctionne en mode **debug**, et vous pourrez visualiser dans la console de votre conteneur les affichages des messages.

- Dans le fichier **server.js**, à la ligne 24 nous utilisons la fonction **getStringMessage()**. Mais le paramètre utilisé n'utilise pas le fichier de configuration. En effet, le champ **NUMBER_WORD** du fichier **.env** n'est pas utilisé. Réalisez les modifications nécessaires dans le fichier **src/config/config.js** et dans **server.js** pour corriger cela.
- Faites les modifications nécessaires dans **.env** pour adapter la configuration à votre environnement, pour que le producteur « trouve » votre cluster **RedPanda**. N'oubliez pas que vous voulons générer des phrases textuelles.
- Construisez votre image **docker** du Producteur, et démarrez un conteneur. Vous devez observer des messages de **debug** dans la console de votre conteneur.
- Observer la présence des messages depuis la console **RedPanda** pour valider le producteur.

Étape 2 – Star de Cinéma : « 26l / 100 km » - Qui suis-je ?

À présent, les messages sont envoyés vers le **broker RedPanda** et vont y rester jusqu'à qu'ils soient consommés. Il faut mettre en place le service qui va s'abonner à votre Topic et récupérer l'ensemble des phrases. Pour une première approche, afficher dans la console les messages du **broker**, sera suffisant pour cette première étape.

- Créer un nouveau projet, et en vous inspirant du producteur, écrire une fonction **connexion()** qui s'abonne à votre **Topic**. Un message dans la console permettra de valider la connexion.
- Dans la documentation de **Kafka.js.org**, identifiez la fonction permettant de consommer les messages. Ainsi, pour chaque message dans le Topic, vous pourrez en afficher le contenu.
- Pour finir, le **timestamp** fourni avec chaque message n'est pas directement interprétable. Créer une fonction permettant de convertir l'affichage avec le format suivant : **dd/mm/aaaa à hh :mm**.

Étape 3 –« Are you Redis »?

Avant de finaliser le projet en reliant ce service au Nœud d'Affichage, il faut réaliser le découpage des messages lu dans le Topic en mot. Ensuite, il faut compter les occurrences de chaque mot pour l'afficher dans la page web utilisateur. Nous allons utiliser une image **docker Redis**. **Redis** est un système de base de données **NoSQL** orienté clé-valeur. Nous enregistrerons dedans chaque occurrence, ou du moins, nous incrémenterons une valeur associée à une clé « mot ». Ainsi, le processus **consommateur** n'aura pas besoin de comptabiliser chaque mot, il lui suffira de demander à **Redis** d'incrémenter le compteur du mot en question.

Ce service **Redis** est couplé avec un service Web écrit en **React** qui permet l'affichage. L'ensemble est à récupérer sur GitHub. Vous pouvez récupérer le projet avec la commande **git clone https://github.com/laurentgiustignano/occurence-mots** dans votre répertoire de travail. Ou bien depuis **PhpStorm**, en choisissant : **Get from Version Control** avec la même URL.

- Modifier votre projet Consommateur pour découper les messages en « mot ».
- À l'aide de la bibliothèque **redis** pour **node-js**, créer les fonctions permettant de se connecter à votre serveur **Redis**. L'incrémentation s'effectue avec la commande **INCR**.
- Remettrez l'ensemble des services en marche (Producteur / Consommateur / Affichage), en allant sur le site web de votre nœud d'affichage, vous devriez observer la liste des mots évoluer.

Bon courage…