Projet - Kafka Streams - Sujet n°1

Vous êtes arrivé dans la société KazaaMovies, spécialisée dans le streaming de films à destination des particuliers.

Ils viennent de mettre en place après plusieurs années de développement une dimension sociale à leur offre de streaming, laquelle va vous fournir des informations sur les interactions utilisateurs et leurs vues, mais ils n'ont mis en place que la première partie technique: la récupération des données.

Votre but: rentre utilisable ces données, lesquelles arrivent dans plusieurs topics Kafka.

Sources de données

Il y a actuellement deux topics qui vous fournissent des données:

- `views` : il contient des évènements de visionnement d'un film
  - Titre du film, dans quelle catégorie rentre la vue (début, partielle, fini le film)
  - Structure :

```
{
    "_id": 1,
    "title": "Kill Bill",
    "view_category": "half"
}
```

- `likes` : il contient les notes données aux films par les utilisateurs
  - On reçoit un évènement par note donnée
  - Structure

```
{
    "_id": 1,
    "score": 4.8
}
```

Environnement

Vous pouvez partir d'un template selon votre langage:

- Scala: https://git.esgi.nyuu.eu/nekonyuu/kafka-as-a-datahub-project-template-scala
  - Akka HTTP, Kafka Streams
- Java: https://git.esgi.nyuu.eu/nekonyuu/kafka-as-a-datahub-project-template-java
  - Kafka Streams, REST framework à votre charge

Vous devrez aussi reprendre le fichier `docker-compose.yml` des exercices :

- Replacez vous dans le dossier `platform/docker` du dépôt d'exercices

- `docker-compose down`
- remplacer l'entrée `injector` par les lignes suivantes (l'indentation est importante) :

```
injector:
  image: nekonyuu/tp-kafka-movies-views:1.1
  depends_on:
    - broker
  hostname: injector
  environment:
    BROKER_HOST: broker:29092
  container_name: injector
```

- `docker-compose up -d`

Vous êtes parés !

Applications à implémenter

Le Product Owner vous demande de lui fournir les informations suivantes

- Nombre de vues par film, catégorisées par type de vue:
  - arrêt en début de film (< 10% du film vu)
    - depuis le lancement
    - sur la dernière minute
    - sur les cinq dernières minutes
  - arrêt en milieu de film (< 90% du film vu)
    - depuis le lancement
    - sur la dernière minute
    - sur les cinq dernières minutes
  - film terminé (> 90% du film vu)
    - depuis le lancement
    - sur la dernière minute
    - sur les cinq dernières minutes
- Top 10
  - des films ayant les meilleurs retours (score > 4)
  - des films ayant les moins bons retours (score < 2)
  - Tip: vous aurez besoin d'une moyenne mobile pour calculer cela depuis le lancement du stream.

Tout ceci doit être exposé sur une API REST (format de sortie JSON) ayant le schéma suivant:

- GET /movies/:id
  - Donne le nombre de vues et leur distribution pour un ID de film donné

```
{
  "_id": 1,
  "title": "Movie title",
  "view_count": 200,
  "stats": {
```

```
    "past": {
      "start_only": 100,
      "half": 10,
      "full": 90
    },
    "last_minute": {
      "start_only": 80,
      "half": 2,
      "full": 0
    },
    "last_five_minutes": {
      "start_only": 2000,
      "half": 100,
      "full": 90
    },
}
```


- GET /stats/ten/best/score
    - Donne les 10 meilleurs films de tous les temps selon leur score moyen, trié par score décroissant


```
[
  {
    "title": "movie title 1",
    "score": 9.98
  },
  {
    "title": "movie title 2",
    "score": 9.7
  },
  {
    "title": "movie title 3",
    "score": 8.1
  },
]
```


- GET /stats/ten/best/views
    - Donne les 10 meilleurs films de tous les temps selon leurs vues, trié par nombre de vues décroissant


```
[
  {
    "title": "movie title 1",
    "views": 3500
  },
  {
    "title": "movie title 2",
    "views": 2800
  },
  {
    "title": "movie title 3",
    "views": 2778
  },
]
```

- GET /stats/ten/worst/score
    - Donne les 10 pires films de tous les temps selon leur score moyen, trié par score croissant

```
[
  {
    "title": "movie title 1",
    "score": 2.1
  },
  {
    "title": "movie title 2",
    "score": 2.21
  },
  {
    "title": "movie title 3",
    "score": 3
  },
]
```

- GET /stats/ten/worst/views
    - Donne les 10 pires films de tous les temps selon leurs vues, trié par nombre de vues croissant

```
[
  {
    "id": 2000,
    "title": "movie title 1",
    "views": 2
  },
  {
    "id": 2,
    "title": "movie title 2",
    "views": 5
  },
  {
    "id": 12,
    "title": "movie title 3",
    "views": 12
  },
]
```

Scope

Il n'est pas demandé que l'application implémentée pour ces besoins gère le lancement d'instances multiples (ce qui impliquerait la gestion de stores distribués et l'usage de RPC).

Toutefois, un effort fait en ce sens pourra compter comme bonus.

Documentation

- Kafka Stream DSL documentation: https://docs.confluent.io/current/streams/developer-guide/dsl-api.html
- Kafka Stream Scala doc: https://kafka.apache.org/20/documentation/streams/developer-guide/dsl-api.html#scala-dsl
- Kafka Stream Configuration Documentation: https://docs.confluent.io/current/streams/developer-guide/config-streams.html
- Kafka Streams Interactive Queries: https://docs.confluent.io/current/streams/developer-guide/interactive-queries.html
- Akka HTTP (REST API): https://doc.akka.io/docs/akka-http/current/routing-dsl/index.html
- Play JSON: https://www.playframework.com/documentation/2.8.x/ScalaJson