LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

Projet Big Data Écosystème

LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

Introduction

Pour le choix du sujet, nous avons choisi d'étudier la partie architecture d'un système de musique en streaming.

Il existe évidemment des groupes connus mettant à notre disposition ce type de technologies Deezer, Spotify et Apple Musique pour ne citer que les plus populaires.

Notre objectif est donc de créer une architecture similaire en utilisant exclusivement les technologies vues en cours afin d'éviter de paraphraser une architecture existante.

L'avantage principal de la musique en streaming est l'accès d'un utilisateur à toutes les musiques stockées dans la base de données sans qu'elles ne soient stockées dans l'appareil de l'utilisateur. Cela permet d'ajouter les nouveautés sans interférer et déranger l'utilisateur.

L'utilisateur pouvant décider à tout moment de changer de musique, on doit pouvoir exécuter des requêtes le plus rapidement possible pour accéder à la bibliothèque musicale.

Puis l'utilisation de ce service crée lui-même des données sur l'utilisateur : Quel est son artiste préféré ? Quels sont les genres musicaux écoutés ? Est-ce que l'utilisateur s'intéresse aux nouveautés ? Toutes ses données peuvent être exploitées par la data science afin de créer des suggestions personnalisées pour l'utilisateur.

Puis ces données créent un profil utilisateur qui peut être comparé aux profils des autres utilisateurs. Combiné avec des techniques de data sciences et des algorithmes prédictifs, il est possible de proposer de manière plus précise des suggestions de nouveaux artistes.

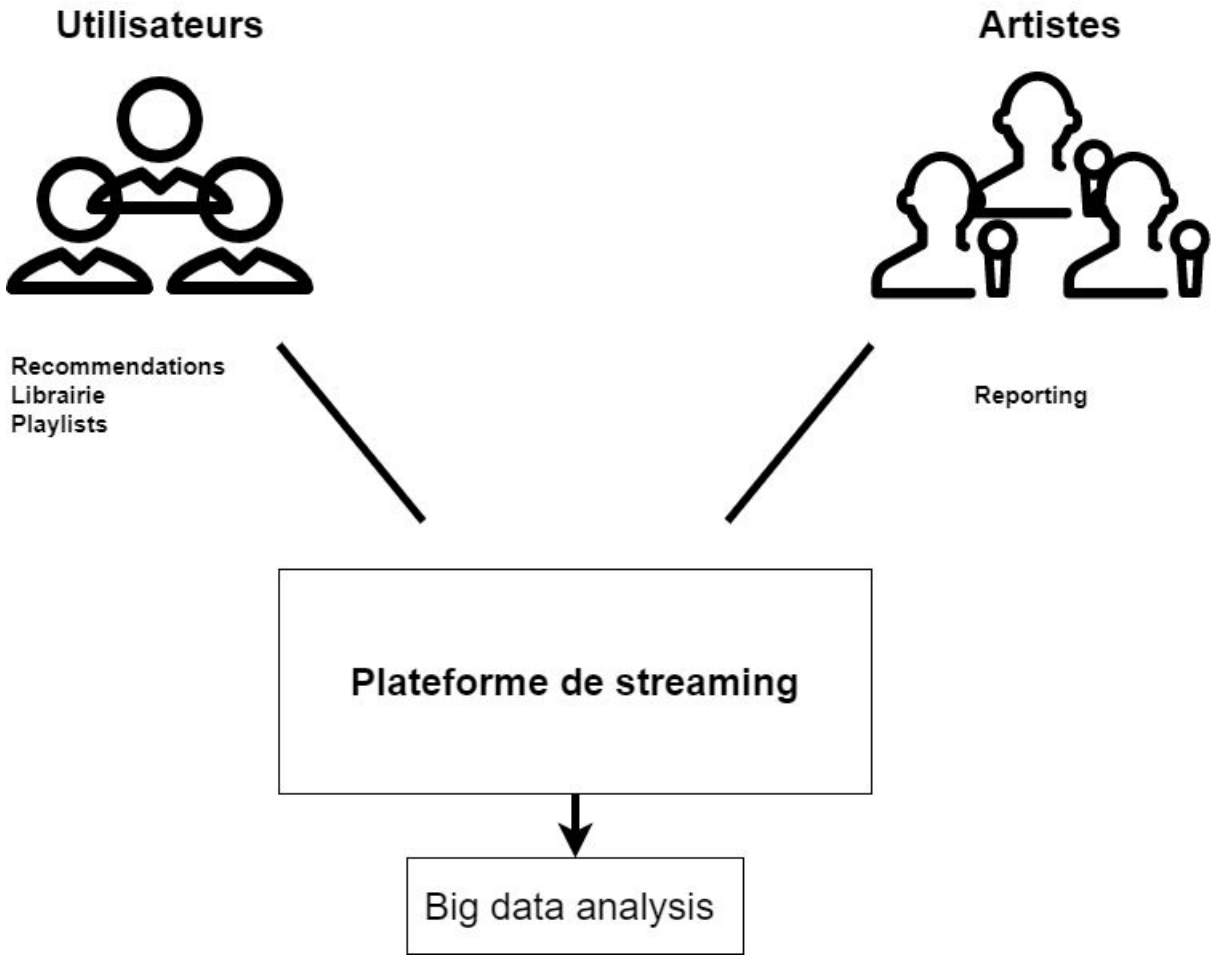
LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

Description détaillée du use case

Dans un service de streaming de musique il y a beaucoup de manipulation de données qui se passe sans que l'utilisateur s'en rende compte. Comment la plateforme arrive-t-elle à stocker des millions de morceaux, à faire des recommandations musicales personnalisées à chacun de ses utilisateurs, à créer des listes de musique populaires en temps réel, enregistrer les playlists de chaque utilisateur etc. Nous allons présenter une architecture d'une plateforme de streaming musicale simplifiée qui expliquera la big data derrière un tel projet.

Admettons que nous voulons que notre service puisse analyser les tendances, les performances de chaque artiste pour ensuite les rapporter à leurs labels discographiques respectifs. Quels outils de systèmes distribués pourrions-nous utiliser afin de mettre en place ce type d'infrastructure?

On peut représenter un schéma ultra-simplifié de la relation entre les utilisateurs, les artistes et la plateforme :



LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

Présentation des technologies impliquées

Pour cette architecture nous allons utiliser les frameworks et outils suivants :

- La plateforme Hadoop
- HDFS comme système de stockage distribué pour tout type de données
- YARN en tant que ressource manager et passerelle entre HDFS et MapReduce/Spark
- MapReduce
- Apache Spark pour du big data processing & stream processing
- Hive pour le data warehousing et de l'analyse
- Kafka pour le système de messagerie entre applications

Apache Hadoop

Hadoop est un framework libre et open source écrit en Java destiné à faciliter la création d'applications distribuées. Basé sur le système Map/Réduce, il permet de segmenter, de stocker et de traiter de grande quantité de données. Le système Hadoop est adapté à notre projet car nous n'avons pas besoin de requêtes avancées.

Hadoop est déjà utilisé pour faire de la recommandation analytique pour les clients. Personnalisant ainsi les suggestions, on peut le retrouver ce modèle chez Netflix, Ebay ou Facebook.

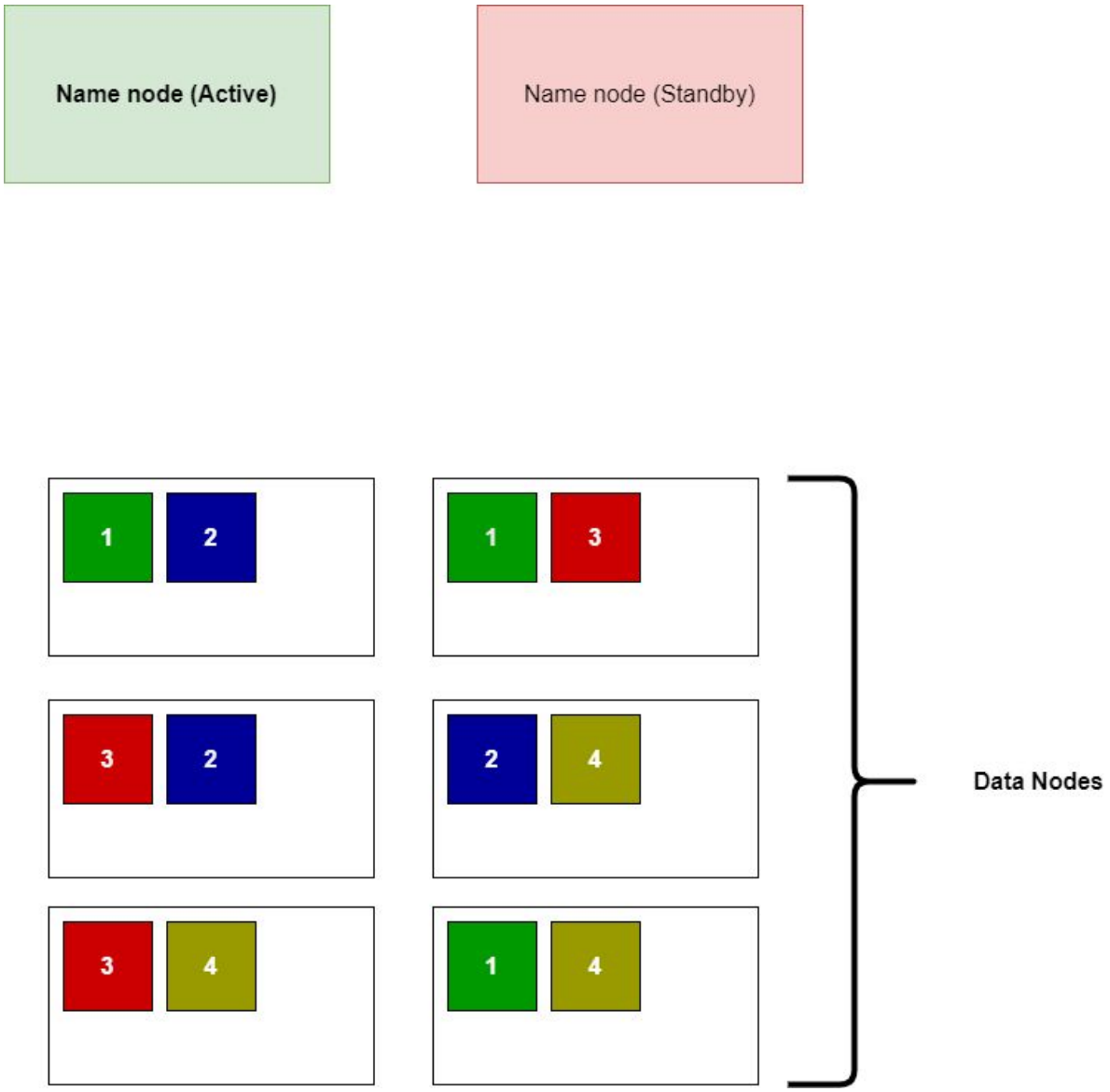
Hadoop Distributed File System (HDFS)

HDFS est un système scalable basé sur le Java permettant de stocker des données sur de nombreuses machines sans organisation préalable. Il nous permettra de stocker nos données, notamment notre bibliothèque musicale. Avec une hiérarchie horizontale, nous évitons une saturation de nos ressources.

HDFS se base sur un système master / esclave composé de clusters et de nodes.

Voici un schéma simplifié de l'architecture d'un cluster dans HDFS :

LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2



Chaque cluster d'une architecture HDFS possède 2 ou plus namenodes. L'une est active tandis que les autres sont en standby. C'est le namenode actif qui s'occupera des opérations du cluster, et le / les standby sont présents au cas où le namenode actif tomberait en panne. Le namenode assure la répartition de blocs de données (représenté par des petits blocs de couleurs numérotés dans les data notes) dans les datanodes.

Un bloc est une partie d'un fichier de taille 128mb qui est dupliqué sur plusieurs datanodes afin d'assurer la disponibilité de ces données.

Les namenodes sont "conscients" des changements effectués dans le cluster car ils partagent des métadonnées.

LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

Pourquoi avons nous choisi HDFS

HDFS est idéal pour stocker des données massives. Le système de répartition des données en blocs dans des nodes nous garantit de ne pas avoir de période d'indisponibilité des fichiers si l'une de nos data node est en panne car ces données auront été dupliquées. HDFS permet aussi une extensibilité très facile, grâce à la hiérarchie horizontale, ce qui nous permet de rajouter du matériel informatique à la suite des autres sans avoir à remplacer les anciens.

Yet Another Resource Negotiator (YARN)

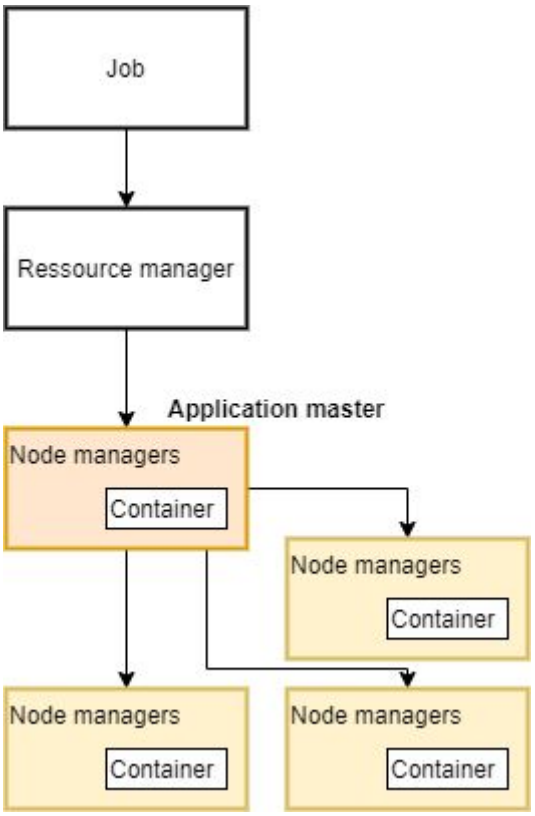
YARN est la fonctionnalité de Hadoop qui a permis de ne plus limiter ce dernier aux fonctions map Reduce. Il s'agit d'un gestionnaire de ressources centrale. Les clusters Hadoop sont maintenant en mesure de lancer des applications d'analyse en temps réel, de streaming data et requêtes interactives sur Apache Spark tout en laissant tourner MapReduce.

Il s'occupe d'accepter les requêtes d'un utilisateur et de les diriger vers les nœuds correspondants.

YARN est composé d'un ressource manager par cluster ainsi que d'un node manager par datanode. Tout comme pour le namenode HDFS, il existe dans un cluster un ressource manager actif ainsi qu'un ou plusieurs en standby.

Chaque node manager est responsable du contrôle d'un container. Un container s'occupe d'allouer des ressources (processeur, mémoire vive, etc.) à un processus / job.

Voici un schéma simplifié de l'architecture YARN :



Le premier container lancé par un node manager est aussi appelé application master, car c'est lui qui va ordonner aux node managers suivants de lancer les autres containers afin qu'ils exécutent leur tâches.

Pourquoi avons nous choisi YARN

YARN est un middleware entre HDFS et MapReduce permettant de gérer les ressources d'un cluster.

YARN fournit un API REST nous permettant de travailler avec d'autres outils Hadoop, notamment Spark.

YARN fait aussi partie du core hadoop et il n'y a pas d'autre alternative dans cet écosystème (sauf dans Hadoop 1.0 ou MapReduce et YARN sont regroupés en un).

LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

MapReduce

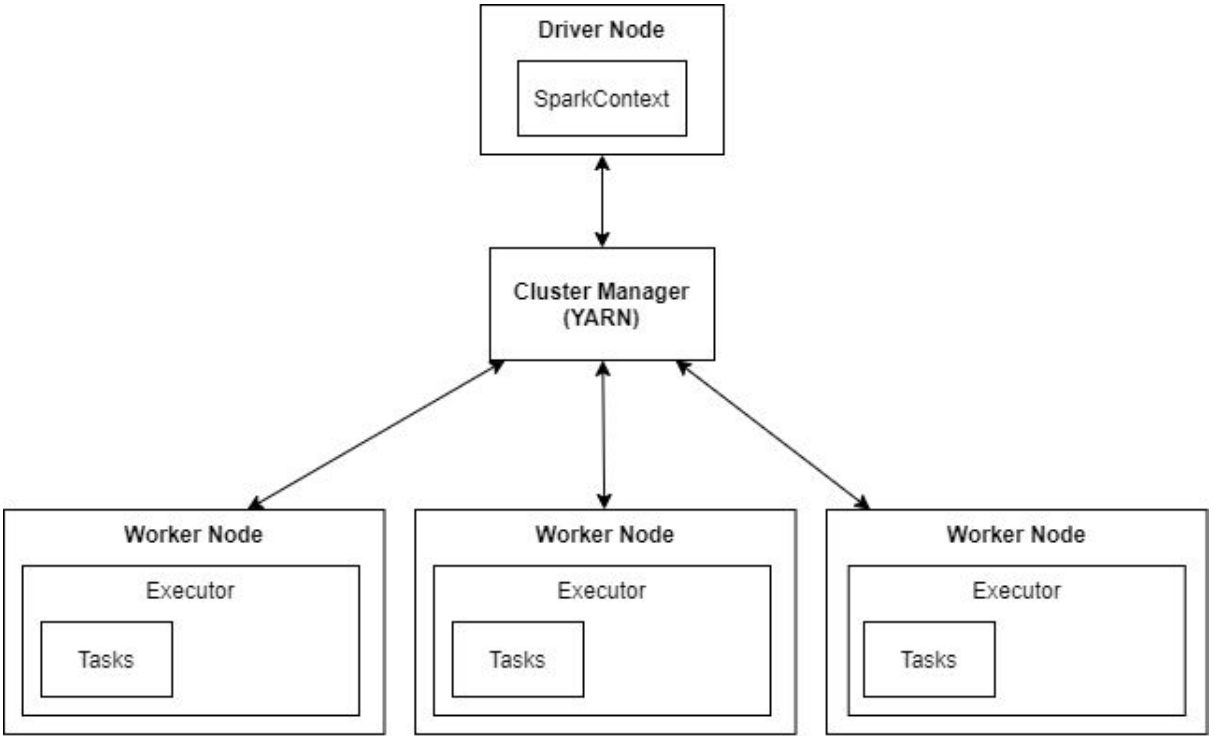
MapReduce est une manière de faire des opérations sur des données qui consiste à mapper une opération sur l'ensemble des données d'un cluster puis de réduire le résultat (Reduce) puis de le retourner.

Apache Spark

Spark est un framework utilisé pour faire du data processing sur un système de fichiers distribués. Spark utilise un type d'ensemble de données appelé *Resilient Distributed Dataset* (RDD) qui est une ensemble d'éléments répartis à travers un cluster.

Un RDD est immuable c'est à dire qu'il peut soit être créé à partir de données existantes soit à partir d'autres RDDs; lui-même ne peut pas être modifié.

Voici un schéma simplifié de l'architecture Spark:



Après avoir fini d'écrire le code, Spark demande au cluster manager les ressources nécessaires pour créer le driver et les executors (un processus lancé sur un worker node. Chaque application à ses propres executors). Le driver node lance l'application créant le SparkContext (fonction qui permet la connexion au cluster Spark, permettant ainsi la création de RDD). Le code est transformé en opérations (tasks) puis ces opérations sont envoyées aux executors afin d'être effectuées sur le RDD.

Pourquoi avons nous choisi Spark

En plus d'avoir la possibilité de faire des opérations *Map* et *Reduce*, Spark nous permet d'avoir d'autres opérations utiles telles que *filter*, *groupByKey*, *sort*, *join*, etc.

Spark fournit aussi plusieurs APIs dont un en Python ce qui est facile d'utilisation ainsi qu'un outil de machine learning, MLlib.

Apache Hive

Hive est utilisé pour traiter des données structurées dans le reporting. Hive utilise un langage de requêtes appelé HiveQL qui est converti en opérations MapReduce.

Apache Kafka

Kafka est une application open source de traitement des flux de données et de mise en file d'attente des messages capable de traiter jusqu'à plusieurs millions de messages par seconde, provenant de différentes sources, et de les acheminer vers plusieurs consommateurs.

Kafka permet le transfert de données entre applications sans accroc, au lieu de se préoccuper de comment envoyer et récupérer des données entres applications.

Kafka fonctionne sur un système publisher/subscriber (publication / abonnement); c'est-à-dire que les données ne sont pas explicitement envoyées d'une application à l'autre, mais plutôt que les données sont classifiés dans des sujets selon leur type; c'est le subscriber qui va "s'abonner" à un des sujets dont il recevra les données.

Dans ce type de système de messaging system, les consommateurs (ceux qui s'abonnent) peuvent s'abonner à plusieurs sujets.

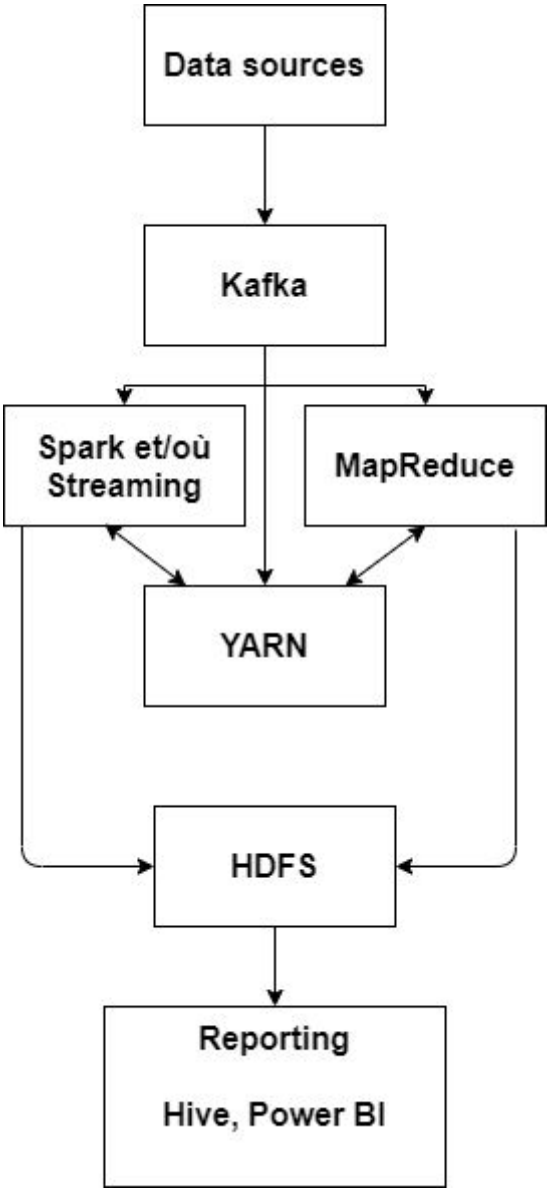
Pourquoi avons nous choisi Kafka

Kafka serait utile dans notre cas car nous aimerions analyser les tendances en temps réel et collecter des métadonnées (par exemple le temps d'écoute d'une chanson par un utilisateur à tout moment donné). Kafka est réactif et rapide et facilement extensible.

LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

Architecture générale du projet

Voici un schéma de l'architecture de notre projet :



Explication

On commence par récupérer les données quelles qu'elles soient (structurées ou non). En se connectant à Kafka, les données sont récupérées dans un cluster de Kafka, composé de brokers. On met en place plusieurs brokers afin de respecter la haute disponibilité des serveurs. Un broker reçoit les messages des *producers*, et les envoie aux *consumers*.
Lorsqu'un message est envoyé à un *producer* à un broker il est reçu dans un sujet ce qui permet de trier les données. Ces sujets sont eux aussi divisés en plusieurs partitions.

LAFAY Baptiste
LANCMAN Nathan
ING5 BDA Gr2

On associe les données à des sujets qui seront récupérés par les *consumers*.

Dans le cas d'un service de streaming de musique, on peut imaginer qu'il y aurait plusieurs types de sujets, certains concernant seulement les artistes et d'autres concernant les utilisateurs, des sujets traitant le nombre d'une certaine chanson x et un autre sujet traitant un artiste y.

Les données sont ensuite récupérées automatiquement grâce aux systèmes de publication / abonnement par YARN et Spark et MapReduce. On peut imaginer que Spark reçoit des écoutes individuelles de chansons différentes d'un seul et même artiste et son job serait d'effectuer une opération MapReduce afin de savoir combien de fois quelles musiques d'un artiste y ont été écoutées sur une période donnée.

Ces données sont ensuite stockées dans HDFS, puis on peut y accéder facilement à l'aide d'outils de business intelligence tels que Hive et Power BI.