Hand's on Kafka

Speakers : Matthieu Blanc & Sylvain Lequeur
Format : Hand's on Lab
Date : 20 avril 2016
GitHub : https://github.com/mblanc/hands_on_kafka

Ce Lab est un atelier de découverte de Kafka en Java et Scala.
Pour réaliser ce lab, clôner le repo GitHub hands_on_kafka et suivre les instructions données dans les slides HTML reveal.js/index_java.html.

Pré-requis :

- Java 8
- Maven ou SBT
- Kafka : http://kafka.apache.org/downloads.html (prendre la version kafka_2.11-0.9.0.1 pour Scala 2.11)

Initialement développé pour Scala, Kafka est désormais devenu Java First. J'ai moi même réalisé le Lab avec Java.

Dans le monde du Big Data, Kafka est derrière Spark le projet le plus populaire.
Kafka est un système de messages. Pourquoi utiliser Kafka alors que l'on a déjà JMS, RabitMQ ou bien encore ActiveMQ ?

L'histoire de Kafka commence chez Linkedin. 3 employés cherchaient à mettre des données de différents systèmes dans un cluster Hadoop.
Les sources de données sont nombreuses. Plusieurs fois la même données (point à point).
Notion de « Big Message » et de « Data form » (pendant du Data Lake pour le temps réel).
Ils cherchaient une plateforme dont les critères sont :

- Découplage
- Débit en lecture / écriture très important
- Distribué pour du scaling horizontal
- Une donnée doit pouvoir être consommés par plusieurs clients, en temps réel ou en batch
- Données persistées sur disque

RabitMQ et ActiveMQ ne convenaient pas. Du coup, ils ont écrits un produit from scratch.

Architecture :
Dans Kafka Cluster, les démons sont nommés **Broker**.
Zookeeper assure la cohérence du cluster kafka.

Concept de base : le **log**
A ne pas confondre avec de simples logs Apache.
Ce sont des enregistrements ordonnés, en lecture seule.
Index unique = offset
Un topic est constitué de plusieurs partitions.
Une partition = 1 commit log. L'ordre des messages est garanti.

Structure d'un Message : (key bytes[], value byte[])
La clé est optionnelle.
Les messages envoyés avec la même clé vont se retrouver dans la même partition.

Kafka ne garde pas en mémoire où en est la consommation du topic par chaque consommateur. Chaque consommateur doit maintenir son état. Depuis Kafka 0.9, l'offset du consommateur est stocké dans un topic technique du cluster Kafka.

Les messages vont être écrits dans des segments (fichier de 1Go par défaut).

Rapidité :

- Kafka essaie de bufferisé les écritures / lectures. Il a un système de cache. Des écritures séquentielles sur disque sont plus rapides que des écritures random en mémoire.
- Kafka utilise le PageCache de Linux. Nativement, Linux se sert de la RAM disponible pour mettre en cache les données. Kafka écrit sur disque. Mais en pratique ils sont en mémoire.
- Kafka ne passe pas par la JVM. Sendfile() envoie la données du disque vers la socket réseau sans passer par la JVM

Microsoft a un cluster Kafka de 1000 nœuds.

Il existe une pratique de groupe de consommation. Chaque membre du groupe s'occupe d'une partition. Gestion automatique des pannes. Load balancing.

Pour garantir les pannes, Kafka crée des réplicas. Les réplicas ne sont utilisés que comme backup au cas où un répica leader tombe. Zookeeper élit un leader par système de quorum.

Version 0.9 sortie en décembre 2015

- Sécurité
- Kafka Connect : destiné aux éditeurs de solutions de persistance, se branche sur le commit log d'une base de données

- Kafka Streams : faire du streaming sans Spark

Solution Apache en 2012
Société Confluent crée en 2014

Cas d'utilisations :

- Monitoring de télémétrie
- Analyse des logs

Utilisateurs : Criteo, OVH, Airbnb, Uber, Microsoft

Slides en reveal.js : utiliser la barre d'espace

Le nombre de partition d'un topic détermine le parallélisme maximal de consommation.