Apache Kafka

* Kafka est un **broker de messages** métiers
* Broker signifie ici intermédiaire entre deux services (ou applications)
* Les services utilisent des technologies non homogènes or ils doivent communiquer entre eux
* Cela crée un couplage fort ce qui rend les choses très compliqués
* Kafka est un **bus de message** pour palier à ce problème
* Kafka est un système de messagerie distribuée en mode publish-subscribe



* Le producer (producteur) est un général le système opérationnel qui écrit le message sérialisé dans Kafka
* Le consumer (consommateur de messge) est souvent un système analytics
* Kafka est donc une zone qui stocke les messages sérialisés (sous forme de série binaire)
* Hautement scalable (distribué dans un cluster Hadoop)
* Mais peut être en dehors de Hadoop 🡪 distribution **Confluent**
* Durable car persistant (écriture sur disque + réplication…)
* **Evènementiel** (traite les messages **en temps réel**)
* Supporte une énorme charge en production et consommation
* Durée de persistance paramétrable

**Concepts clés**

* Un ou plusieurs brokers (une instance de Kafka) dans le cluster Kafta géré par le Zookeeper
* Le producer écrit le message sur un broker
* Le consumer lit les messages des brokers
* Mode distribué
* Les messages sont écrits dans des **topics** (lignes de message de même nature ~ notion abstraite assimilable à une table un SGBDR)
* Le message peut être de n’importe quel type (Kafka parle qu’en binaire)
* Les topics sont divisés en partitions
* Les partitions sont répliquées sur plusieurs brokers
* Le temps de rétention est paramétrable
* La position d’un message dans la partition lors de la lecture est appelée **offset**
* Lors de la création d’un topic, on décide du nombre de partitions, la seule limite est nombre de partitions (nœud)
* Le nombre de partition détermine le degré de parallélisme d’un topic (plus on a de partitions plus on peut écrire et lire des messages en parallèle 🡪 bien sûr cela dépend aussi du nombre de producer et de consumer)

**Pipeline d’écriture d’un message**

* Dans un broker, l’écriture se fait sur une seule partition appelé leader. Et cette partition leader est répliquée sur d’autres broker et on appelle ses réplicas : follower
* le processus se fait en mode évènementiel (temps réel), du coup les followers ne sont presque jamais à jour.
* Le message est envoyé par le producer au leader et les followers consomment les messages stockés dans la partition leader
* Un KPI nommé **ISR** (In-Sync Replicas) est le nombre de follower synchrones avec le leader
* La définition de "in-sync" dépend de la configuration (gérée par le développeur donc paramétrable), mais par défaut, cela signifie qu'une réplique est ou a été entièrement rattrapée par le leader au cours des 10 dernières secondes.
* Il n'est pas obligatoire d'avoir un ISR égal au nombre de répliques.
* Le paramètre pour cette période est : replica.lag.time.max.ms et a une valeur par défaut du serveur qui peut être remplacée par le développeur.
* Les suiveurs répliquent les données du leader vers eux-mêmes en envoyant des requêtes Fetch périodiquement, par défaut toutes les 500 ms.

**Kafka producer**

* Créer un topic Kafka c’est fournir une habilitation à une appli pour qu’elle puisse consommer le message
* Acks 🡪 Acknowledge qui signifie accusée de réception
* Les messages sont en format clé-valeur
  + **Key** : le hache de la clé détermine la partition qui reçoit le message
    - Si la clé n’est pas spécifiée, la partition qui reçoit est définie Round-robin (répartition équilibrée des charges sur les partitions)
  + **Value** : le contenu du message stocké sous forme d’une liste de byte (série binaire)
    - La valeur peut être de n’importe quel type sérialisable

**Les paramètres de Kafka producer**

* Acquittement des messages (Acks acknowledge)
  + **Acks = 0** aucun acquittement (donc pas de **retries**)
  + **Acks = 1** message acquitté uniquement par le leader
  + **Acks = -1** message acquitté par le leader et les followers
* Sérialisation des messages
  + Key.serializer / value.serializer 🡪 classes de sérialisation
  + Il y a aussi la sérialisation par défaut
* Batching des messages 🡪 la sérialisation a le même coût pour un message de 1 bit et 1 Go
  + Batch.size : limite le nombre de message à batcher
  + Buffer.memory : limite la taille du buffer de batch
  + Linger.ms : limite le temps d’attente avant envoi du batch
* Compression 🡪 lié au flux réseau
  + Compression.type : gzip, izt, snappy, uncompressed
* L’ordre des messages
  + On peut perdre l’ordre des messages au niveau du topic à cause de l’acquittement
  + L’ordre des messages est garanti dans la partition
  + Pour que les messages soient consommés dans l’ordre de leur de production, il faut leur attribuer les mêmes clés
* Renvoi automatique des messages en cas d’échec