**APACH KAFKA**

Table des matières

[Chapitre 1 INTRODUCTION 2](#_Toc102039837)

[Chapitre 2 CONCEPTS CLES : 2](#_Toc102039838)

[Chapitre 3 PIPELINE D’ECRITURE DE MESAGES : 3](#_Toc102039839)

[Chapitre 4 PARAMETRES DE PRODUCERS KAFKA 4](#_Toc102039840)

[Chapitre 5 MODES DE PRODUCTION : 5](#_Toc102039841)

[a-Production bloquante : 5](#_Toc102039842)

[b-production non bloquante : 5](#_Toc102039843)

[Chapitre 6 CONSUMERS KAFKA : 5](#_Toc102039844)

[Chapitre 7 COMMIT DES MESSAGES CONSOMMES : 6](#_Toc102039845)

[Chapitre 8 STATELESS VS STATEFULL : 7](#_Toc102039846)

[a- Stateless : 7](#_Toc102039847)

[b- Statefull : 7](#_Toc102039848)

[Chapitre 9 MESSAGE DELIVERY : 7](#_Toc102039849)

[a- At Least Once : 7](#_Toc102039850)

[b- At Most Once : 8](#_Toc102039851)

[c- Exactly Once : 8](#_Toc102039852)

# 

# INTRODUCTION

Brocker = serveur = intermédiaire,

Kafka sert à envoyer des messages entre deux ou plusieurs services (applications) d’une entreprise. Kafka est comme une boite aux lettres il nous met à disposition un message et attend qu’on le consomme.

Kafka peut être un service indépendant de hadoop on parle à ce moment là de confluente. Kafka ne stocke que du binaire donc il sérialise chaque objet.

Kafka est très convoité pour ses performances surtout dans le big data quand on parle de l’évènementiel (temps réel). Stockés des milliers voire des millions de messages en temps réel (exemple de LinkedIn en 2014 avec un pic de 200 millions de messages en seconde).

# CONCEPTS CLES :

Kafka cluster :1 ou plusieurs brockers coordonnés via zookeeper.

Brocker : 1 instance de kafka (1 seule machine).

Les producteurs (producers) : écrivent des messages sur des brockers.

Les consommateurs (consumers) : lisent des messages sur des brockers.

Les messages sont écrits dans des topics qui sont divisés en partition.

Topic = une ligne de messages qui sont de même nature. (Nom d’un flux sur lequel les producers écrivent).

Une partition est répliquée sur plusieurs brockers.

Un groupe de consommateurs (consumers group = plusieurs consommateurs) s’occupent 1 seul topic et chaque consumer s’occupe de d’1 seule partition.

***Consumers group***

Consom 1

Consom 2

Consom 3

Consom 4

**TOPIC**

producer

producer

***Producer***: les messages sont publiés à la fin d’une partition.

La position d’un message dans un topic appelé ***offset***.

***Consumer :*** Consomme les messages en conséquence.

***Conception :*** Le topic n’existe qu’à travers les messages. Chaque message écrit sur une partition sera indexé par un offset.

Une partition est une séquence ordonnée et immutable de messages (une liste de messages infinis).

Les messages sont ajoutés uniquement à la fin d’une partition.

Le nombre de partitions d’un topic est configurable

Le nombre de partitions détermine le degré de parallélisme d’un topic.

2 consumers du même groupe s’occupent de deux partitions.

***Réplication :***

**Topic1**

Part 1

T**opic1**

**Topic1**

Part 2

Part 3

Brocker 1 brocker 2 brocker 3

***Un réplica*** dans kafka est un ***follower*** car ***le leader*** reçoit continuellement des messages (en temps réel) donc les followers ne font que suivre et se mettent à jour au fur et à mesure (les followers deviennent alors des consommateurs kafka du leader)

Si un brocker tombe en panne, on élira alors le brocker qui est le plus à jour (à l’aide de l’ISR qui est un paramètre de l’évaluation de degré de mise à jour de chaque réplica dans un brocker).

Le nombre de réplica < au nombre de brockers (si on a 4 brockers dans l’idéal est d’avoir 3 réplica

# PIPELINE D’ECRITURE DE MESAGES :

Le leader prend une décision en fonction des ISR (ISR : In Synchron Replica : nombre de réplicas qui sont à jour (synchrones) avec le leader).

Il peut être défini par le nombre de messages sur lesquels il est en retard (ou estimation du temps de retard qu’il a avec le leader).

Si par malheur un follower n’arrive pas à suivre et à se mettre à jour (pour cause des latences réseaux par exemple) alors le leader peut nommer un autre brocker pour devenir follower.

# PARAMETRES DE PRODUCERS KAFKA

Les clients de kafka sont les différents langages de programmation : java, python, scala, etc

Les messages sont au format (key, value). {map}

***Key :*** le hash de la clé détermine la partition qui recevra le message : en cas d’absence de clé kafka va faire du **Round-Robing (aléatoirement)**.

***Value :*** Présente le message, elle est stockée sous forme d’une série de byte. (Cette valeur peut être de n’importe quel type sérialisable).brocker.one : 9092 =numéro de port de l’écoute de kafka.

1. ***Acks*** : acknowledgement (accusé de réception) :

On le met à un🡺 quand on veut un accusé de réception du leader.

On le met à 0🡺 quand on n’a pas besoin d’un accusé de réception.

On le met à -1🡺 quand on a besoin d’un accusé de réception du leader et de chaque fellower qui le suit

***Use case*** :

On met ***Acks= 0*** c’est le cas des times series (températures par exemple) parceque si on perd une valeur d’une température ce n’est pas grave (la température ce n’est pas une valeur qui change rapidement dans le temps donc si on perd une valeur de température cela n’est pas si grave)

Par contre on le met ***Acks = -1 le*** quand les données sont hyper importantes (ça génère beaucoup de latences réseaux) mais l’importance de la donnée (message) impose que le Producer doit s’assurer de la bonne réception du message de la part du leader et tous ses fellowers.

1. ***Sérialisation des messages :*** key.serializer, value.serializer
2. ***Batching des messages :*** pour contrôler les performances du producer
   * 1. ***Batch.size :*** (methode send)limite le nbre de message à batcher.
     2. ***Buffer.memory :*** limite la taille des messages
     3. ***Linger.ms :*** limite le temps d’attente du producer avant le renvoi du batch.

***NB :*** finalement le producer n’envoie pas directement au brocker mais il passe d’abord par un buffer dont la taille est gérable par la commande suscitée.

1. ***Compression :*** (on peut décider de ne pas compresser et matcher le producer avec consumer) compression.type : les codes supportés sont : gzip, izt, snappy, uncompressed
2. ***L’ordre des messages :*** l’ordre des messages est uniquement dans une partition est uniquement dans une partition pour s’assurer que des messages seront consommés dans leur ordre de production. Il faut qu’ils aient tous la même clé
3. ***Renvoi automatique de message en cas d’echec :***
   * 1. ***retries***: le nombre de fois un message est renvoyé par le producer en cas d’erreur (d’échec).
     2. ***Request.timeout.ms :*** le temps pour timeout une requête.
     3. ***Retry.backoff.ms :*** le temps à attendre avant de renvoyer un message.

Il existe un risque de changement d’ordre des messages si retries >0. Pour garantir l’ordre il faut taper la commande suivante : ***max.inflight.request.per.connection = 1*** (il y a une seule connexion = 1 le deuxième ne passe pas sauf si le message est passé sans échec) par contre cette manipulation crée un goulot d’étranglement (bouchon).

# MODES DE PRODUCTION :

## a-Production bloquante :

quand on met le acsk différent de 0 ( 1 ou -1) : producer.send(record).get

## b-production non bloquante :

producer.send(record,new callback….);

***NB :*** Le nombre de partitions qui rend kafka scale-out. Le nombre d’applications qui produisent et qui consomment des messages et la vitesse de production et de consommation qui définissent le nombre de réplicas qu’il faut pour kafka.

# CONSUMERS KAFKA :

Il existe des consumers dans plusieurs langages de programmation : java, c++, python, ruby etc…

Chaque consumer à chaque fois qu’il consomme un message il commit (soit au broker soit à zookeeper)(le commit = dire à kafka que j’ai consommé un message) :

2 API kafka consumers pour java/scala :

* Old consumer API (zookeeper qui met à jour les offsets de consommation).
* New consumer API : existe pour la version kafka >= 0.9 : les offsets sont gérés par un topic kafka

Si on veut créer un groupe de consumers pour un seul topic donné il suffit de de mettre le paramètre props.put(« group.Id », « test ») (test est le nom du topic qu’on consomme)

Un consumer peut souscrire à consommer plusieurs topics.

* Avanages :
* Les consumers peuvent reprendre la consomamtion à partir de n’importe quel ofset.
* Les consommateurs peuvent choisir les partitions à lire.

***Batcher la consommation :*** lancer le consumer par intervalle de temps

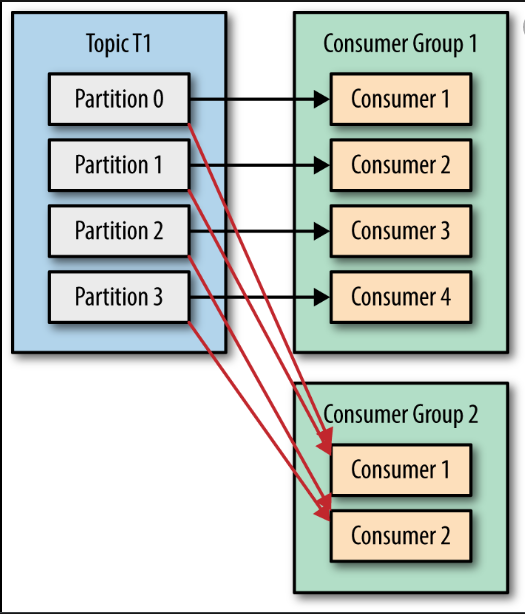
Les consommateur kafka fonctionnement en mode pull

C’est le consommateur qui ramène les messages du broker quand il le souhaite.

Avantages :

* Chaque consumer consomme à son ritjme
* Améliorer la stabilité des applications en aval du consumer.(les applications consommatrices conçues pour 1 charge normale pas pour des pics de consommation.

Chaque consommateur appartient à un groupe de consumers.



# COMMIT DES MESSAGES CONSOMMES :

Selon le use case on peut choisir :

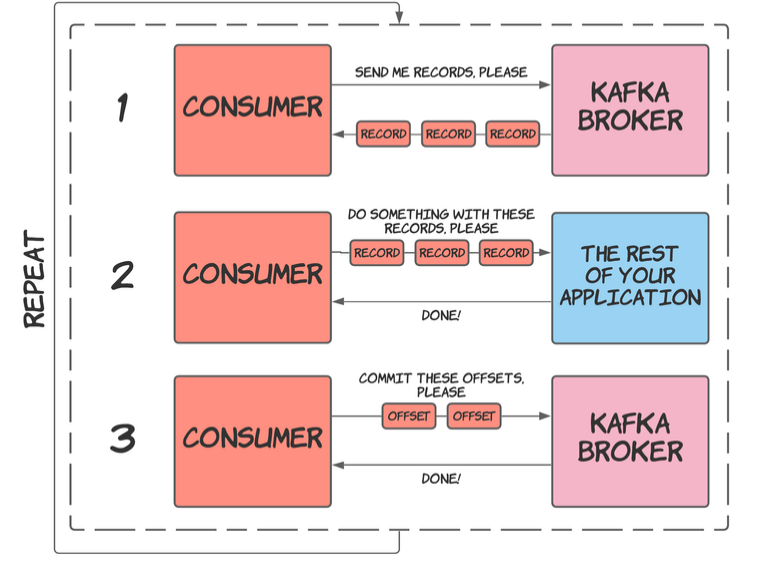
1. Commit automatique (automatic commit) : la gestion du commit est laissée au consommateur.
2. Commit manuel : le commit des message est à la charge du développeur (gérer dans le code) : prop.put (‘’enable.auto.comit’, false) 🡺 manuel.

Avantage des commit automatiques : Simplifie le code

Inconvénient du commit automatique : risque de perte de messages.

Avantage du commit manuel : maitrise totale du cycle de consommateur

Inconvénient : risque de double consommation (suite à des erreurs réseaux.) =>(zookeeper n’a pas l’instruction synchronisée avant le commit)



# STATELESS VS STATEFULL :

(upsert : update et insert au même temps)

## a- Stateless :

Chaque message est atomique : (pas de relation entre les messages)

Exemple : calcul de tva sur chaque produit

Mettre des strings en majuscule

Pas besoin de connaitre les anciens messages pour pouvoir effectuer un traitement

## b- Statefull :

le contraire de stateless : il faut connaître les anciens messages pour pouvoir effectuer un traitement.

Exemple calcul de chiffres d’affaires en faisant la somme de toutes les ventes. (chaque message kafka est une vente)

# MESSAGE DELIVERY :

## a- At Least Once :

Pas de perte de message mais on accepte de dupliquer deux fois.

Cette méthode on l’utilise surtout dans l’insertion dans les bases de données relationnelles. C’est le mode de kafka par défaut.

## b- At Most Once :

On accepte de perdre les messages par contre la duplication de messages n’est pas acceptables et la perte de messages est tolérée.

Kafka : configurer le Producer sans retry (retries =0)

## c- Exactly Once : Producer Idempotence:

le message est reçu par les consumers une et une seule fois. (la duplication et la perte de messages ne sont pas acceptées)

Pour s’y faire il faut avoir une coopération entre les producers et les consumers & de kafka.

Ça nous offre une garantie de traiter chaque message une et une seule fois.

Cette option n’est pas offerte par tous les systèmes de traitement en temps réel.

La duplication des messages est due :

* la non réception d’ack par le producer.
* Double consommation d’1 message due au crash d’1 consumer après processing du message mais avant de le commiter.
* Duplication de production / process zombies : elles peuvent apporter pour cause de coupure temporairement de réseaux ce qui provoquera le lancement de nouvelles instances. => deux instances font les mêmes taches => duplication du message.

Configiration : enable.idempotence = true.

Avantages :

* Pas de réplication de messages due à un renvoi pour cause d’indisponibilité d’un broker ou d’erreurs de connexion
* Respect d’ordre’ d’envoi des messages.

***NB :*** Les messages sont toujours envoyés d’une façon sérialisée par les producers.

***La serialisation :*** est le processus qui vise à créer et à transformer un objet en mémoire d’une façon optimale (compacte et rendre en série binaire) :

1. Quand on envoie sur le réseau
2. Quand on envoie sur le disk ;

***Transient :*** c’est les attributs exclus de la sérialisation d’un objet (les méthodes d’un objet ne font pas partie car elles décrivent un comportement de l’objet et elle n’est chargée qu’une seule fois en mémoire d’opérations).

***GCP :*** Pop sub = kafka (équivalent de kafka dans google cloud platform)