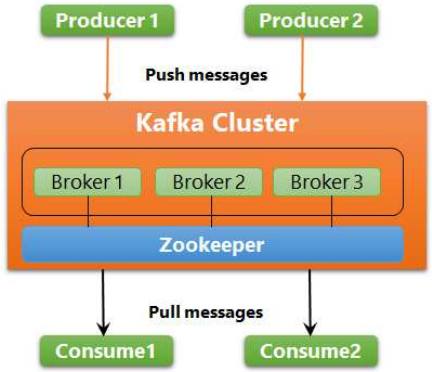
Traitement Distribué en Big Data

- Concepts de base du Big Data
- Stream Processing : Kafka & Kafka Stream
- Batch et Micro Batch Processing : Spark

KAFKA

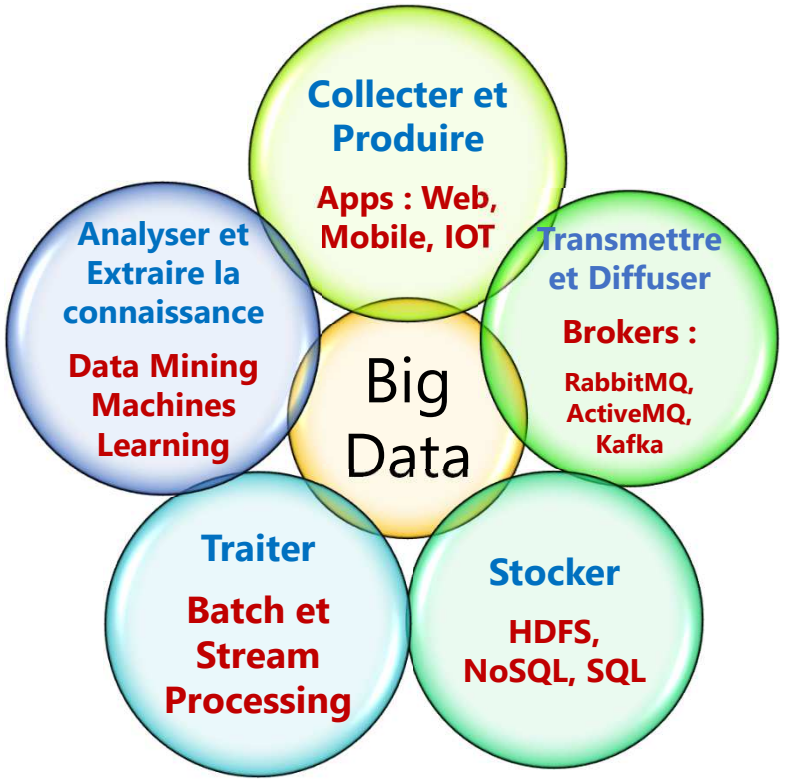


APACHE Spark™

Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

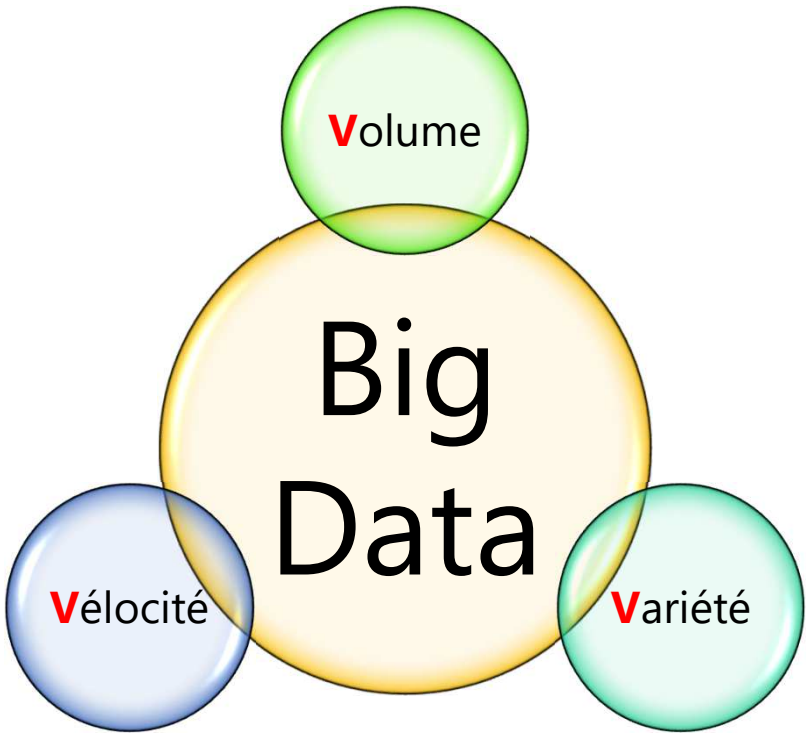
Big Data

- Explosion de la quantité de données générées par :
  - Les applications : Les réseaux sociaux, Les applications web mobiles, Les objets connectés, Les logs etc.
- IL est nécessaire de chercher les moyens qui permettent :
  - **Transmettre et diffuser les données entre les appli distribuées** (Brokers : RabbitMQ, ActiveMQ, Kafka)
  - **Stocker et sécuriser les données d'une manière distribuée** (Hadoop HDFS, NoSQL : Cassandra, MongoDb, Hbase, Elastic Search, etc..
  - **Traiter let Analyser es données d'une manière distribuée en vue d'en extraire la connaissance pour des prises de décisions**
    - Big Data Processing : Batch Processing (Map Reduce, Spark) et Stream Processing) (Spark, Kafka Stream, Flink, Storm, Samza)
    - Data Mining, Machines Learning (TensorFlow, DeepLearning4J, Weka, etc…) pour l'extraction de la connaissance
  - **Analyser et Visualiser les indicateurs de prises de décisions**
    - Big Data visualisation Tools



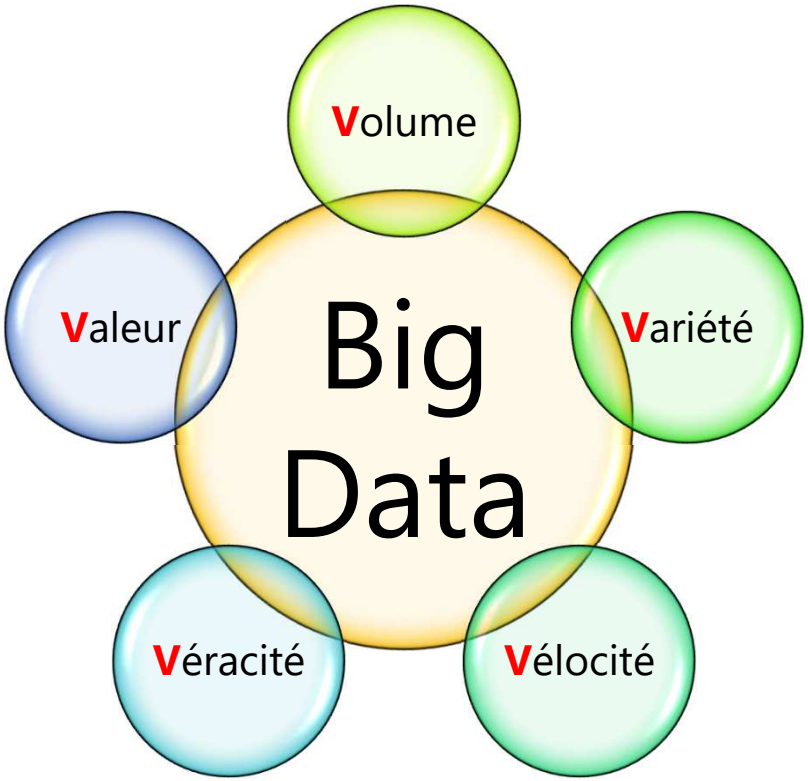
Big Data : 3V

- Volume :
  - Quantité de données (L'ordre des PO)
- Variété :
  - Différents formats de données (Structurées: 20% ou non structurées : 80%)
    - Texte, CSV, XML, JSON, Binaires, BDR, etc …
- Vélocité : Fréquence de l'arrivé des données
  - Exemple : (Twiter)
    - Chaque seconde environ 5 900 tweets sont expédiés sur le site de micro-blogging Twitter. Cela représente 504 millions de tweets par jour ou 184 milliards par an. Cette masse d'information vient alimenter le flot d'informations ("big data") publiée par l'humanité chaque jour sur internet.

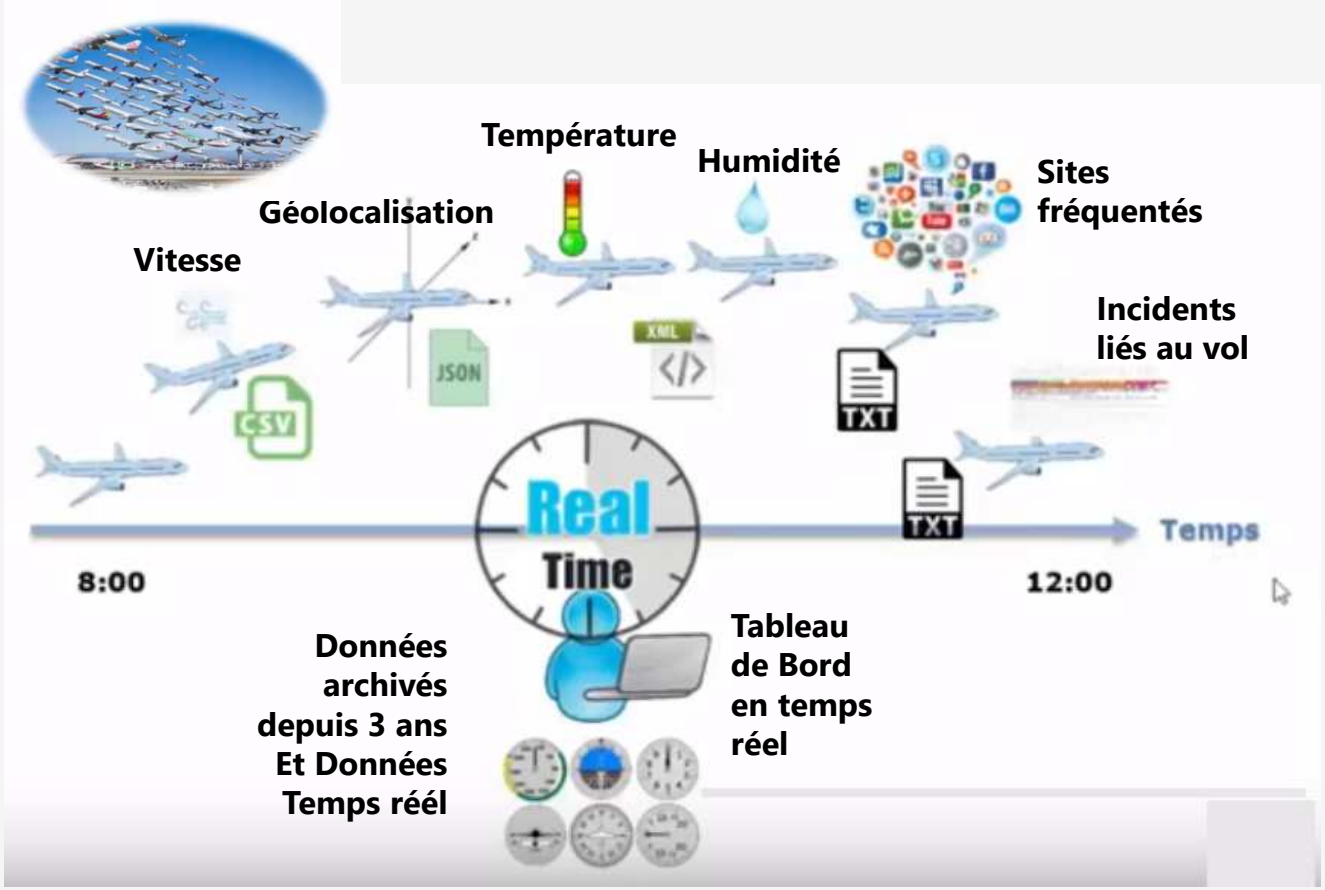


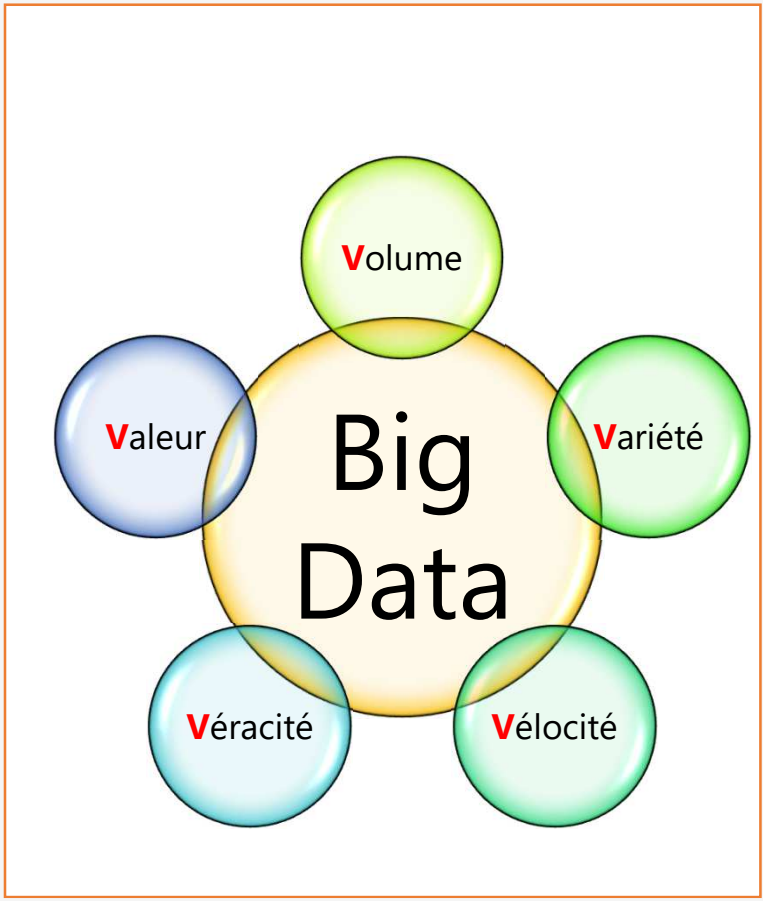
Big Data : 5V

1. Volume
2. Variété
3. Vélocité : Fréquence de l'arrivé des données
4. Véracité :
   - Fiabilité et la crédibilité des données collectées (Sources Fiable)
5. Valeur :
   - Profit et la connaissance que l'on peut extraire de ces données
   - Transformer les données en valeur



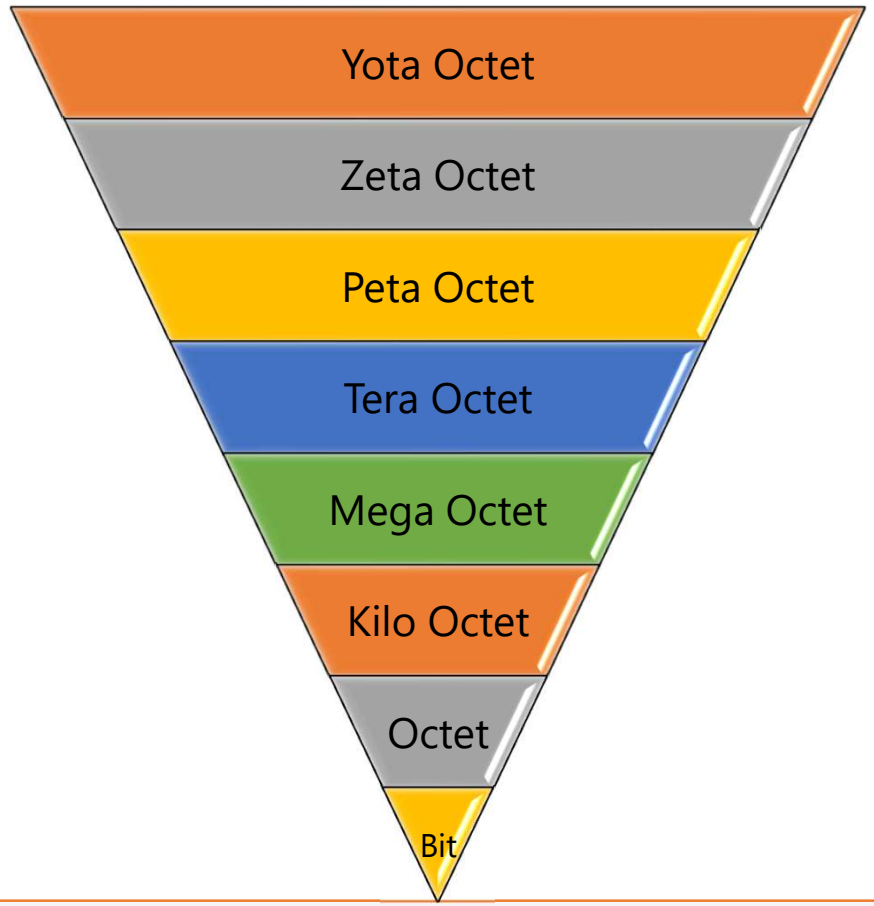
Exemple de problème : Compagnie aérienne



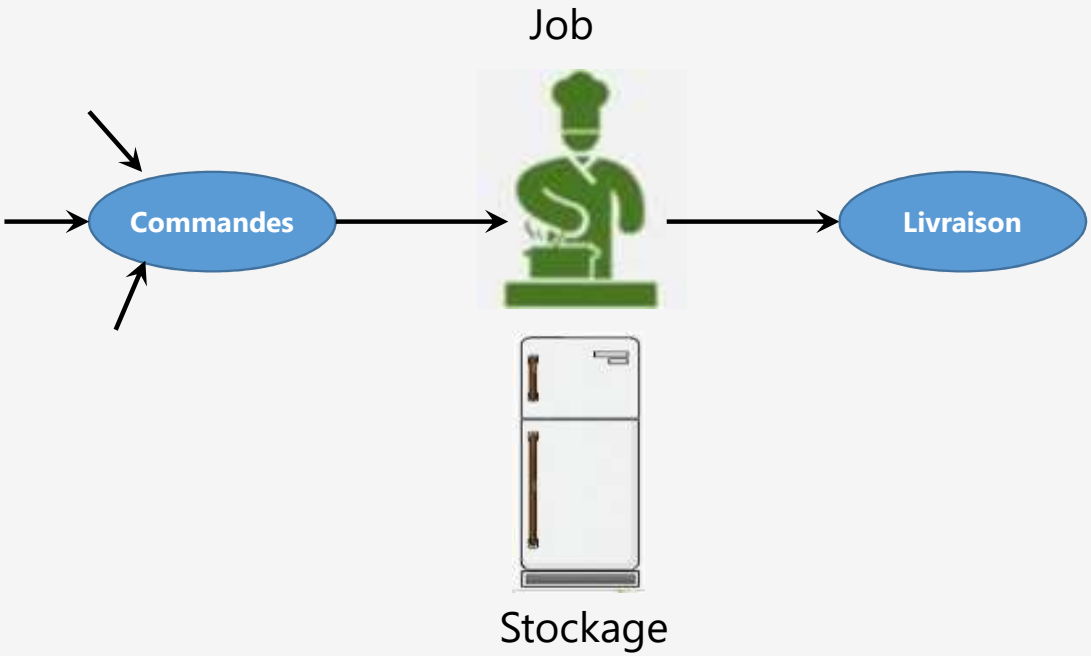


Ordre de mesure de stockage des données

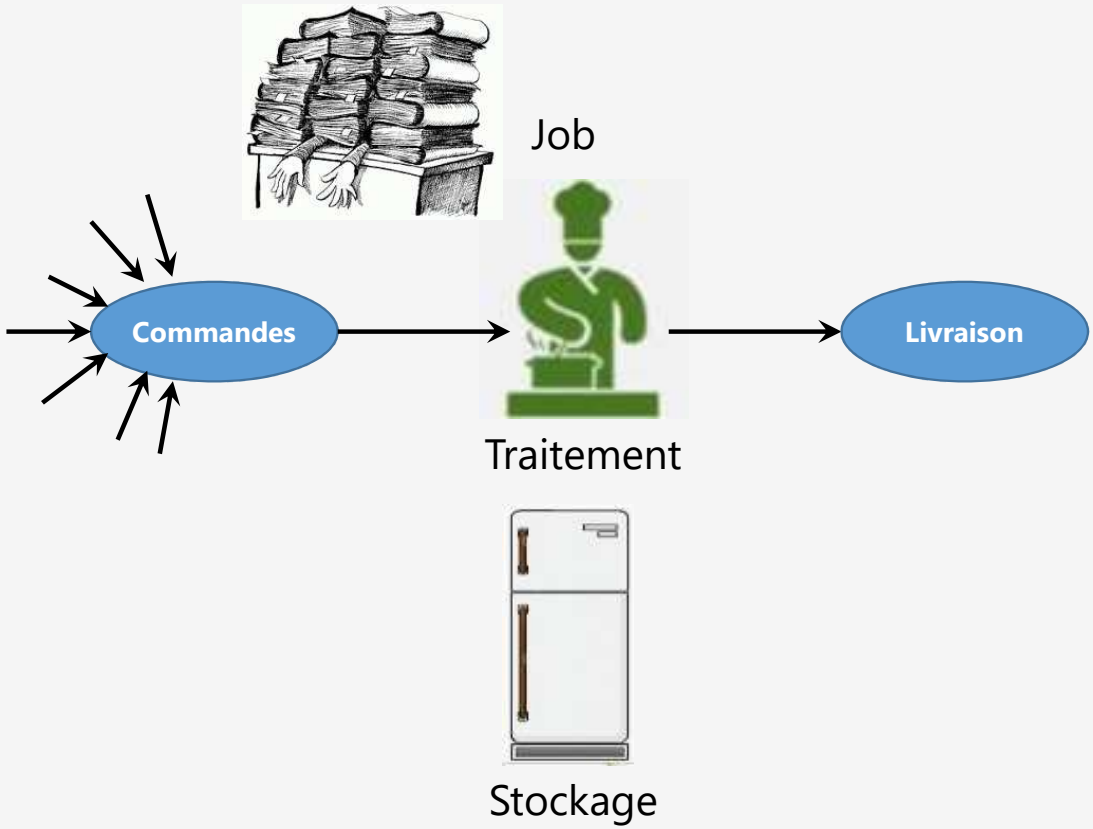
- 1 caractère => 1 octet
- 1 Page de texte =>30 ko (1ko=1024 o)
- 1 Morceau de musique => 5Mo (1Mo=1024 ko)
- 1 Film de 2H => 1Go (1Go=1024 Mo)
- 6 Millions de Livres => 1 To (1To=1024 Go)
- 2 Millard de Photos Rés Moyenne => 1 Po (1 Po=1024 To)
- Toutes les infos produites jusqu'à 2003 => 5 ExaO (1Eo=1024 Po)
- Données produites en 2011 => 1,8 ZetaO (1Zo=1024 Eo)
- En 2013 le plus grand data center au monde est capable de traiter 1 YotaO (1Yo=1024 Eo)



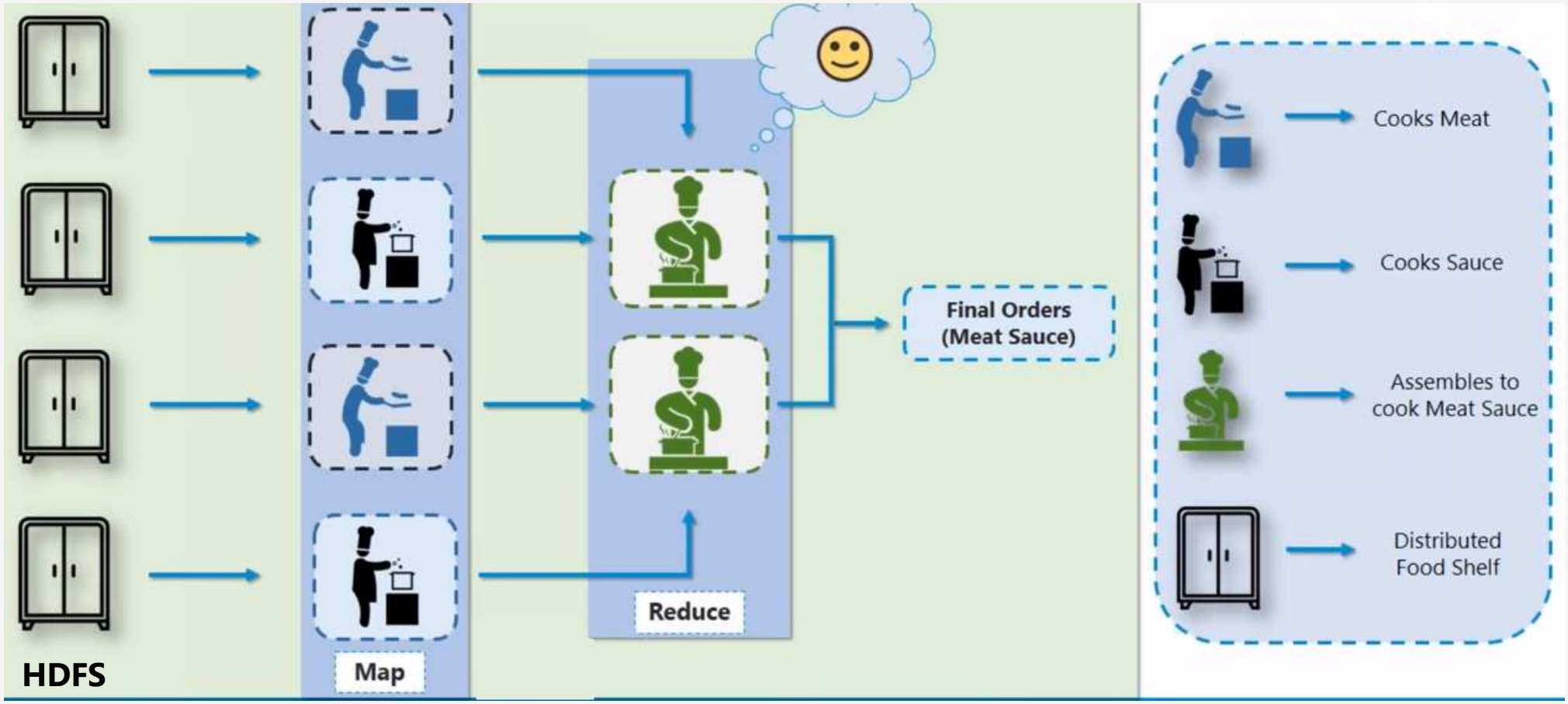
Le Big Data dans la vie courante : Un Cuisinier pour traiter les commandes



Le Big Data dans la vie courante : Un Cuisinier pour traiter les commandes

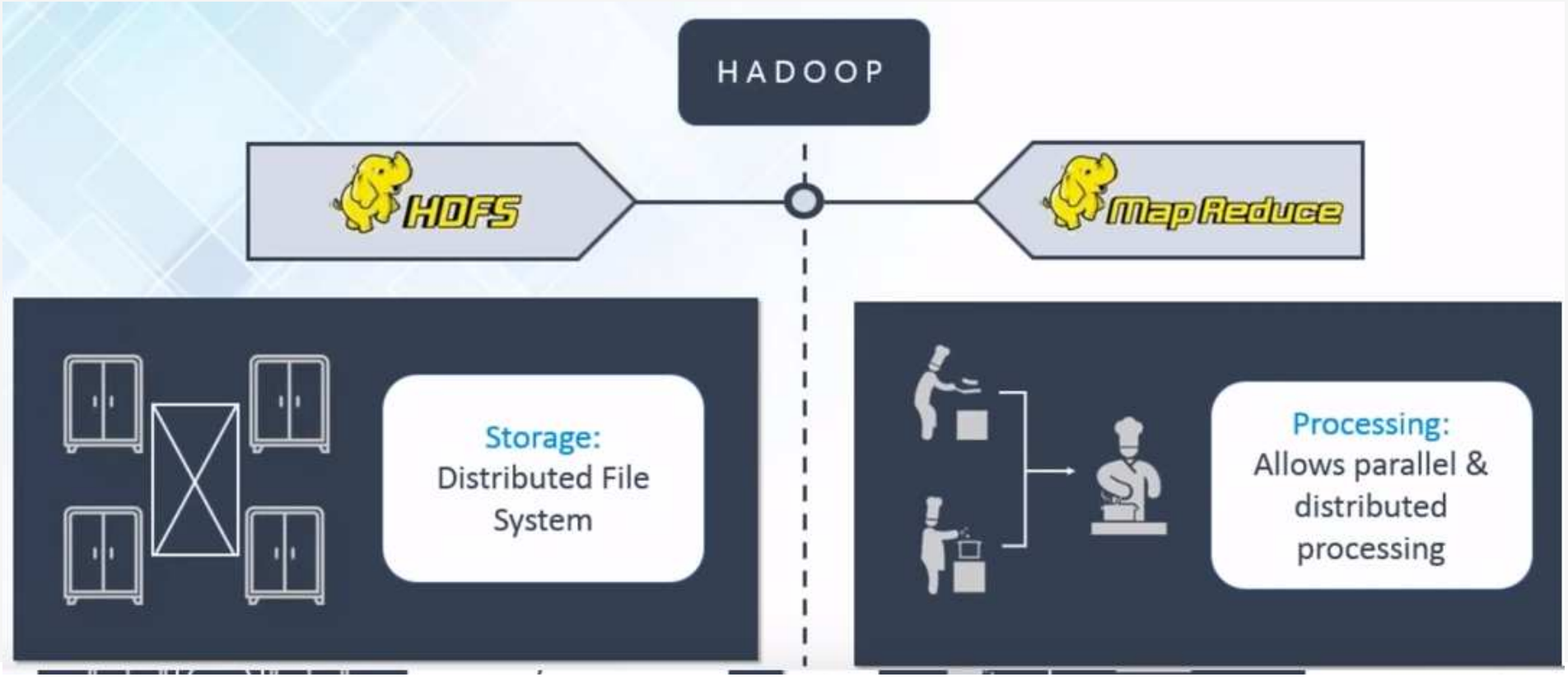


Le Big Data dans la vie courante : Distribuer les tâches et le stockage



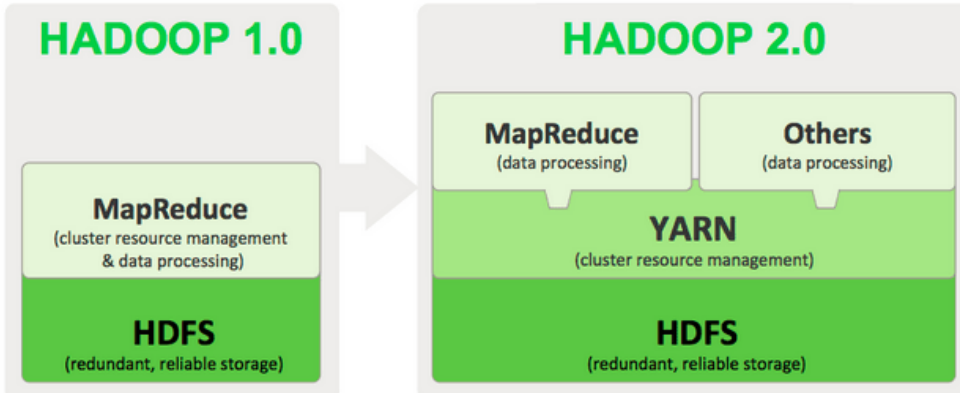
Eco-système Hadoop

- Hadoop est Framework qui permet de **Stocker** et **Traiter** une grande quantité de données de manière **parallèle** et **distribuée**



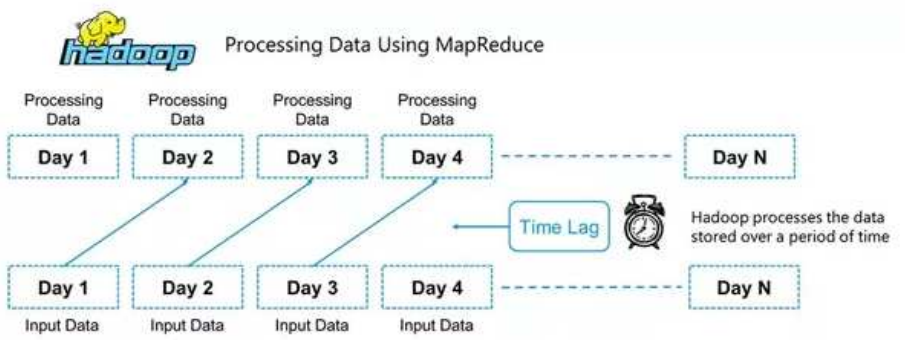
Eco-système du Big Data : Apache Hadoop

- Hadoop est un Framework libre et open source écrit en Java, destiné à faciliter la création d'applications massivement distribuées avec des milliers de nœuds.
  - Au niveau du stockage distribué des données (des pétaoctets de données) : HDFS
  - Au niveau Traitement de données : Map Reduce
  - Avec la prise en charge de caractéristiques fondamentales :
    - Haute disponibilité, Scalabilité, Tolérance aux panes, Reprise après échec, Sécurité (HPC : High Performance Computing)
- Le Framework Hadoop de base se compose des modules suivants :
  - **Hadoop Distributed File System (HDFS) : le système de fichiers distribué**
  - **Hadoop YARN (Yet Another Resource Negotiator): Système de Gestion des Ressources du Cluster**
  - **Hadoop MapReduce : Traitement distribué de données**



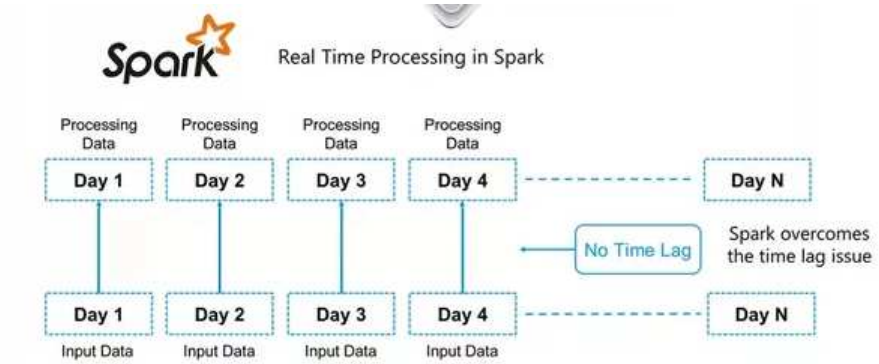
Batch Processing / Stream Processing

- **Batch Processing** (Traitement par lots) :
  - Traitement de blocs de données déjà stockés sur une période donnée.
  - Par exemple, traiter toutes les transactions effectuées par une entreprise financière en une semaine.
  - Ces données contiennent des millions d'enregistrements pour chaque jour pouvant être stockés sous forme de fichiers textes (CSV) ou d'enregistrements stockées dans HDFS, SGBD SQL, NoSQL, etc.
  - Exemple de Framework :
    - Map Reduce
    - Spark



Batch Processing / Stream Processing

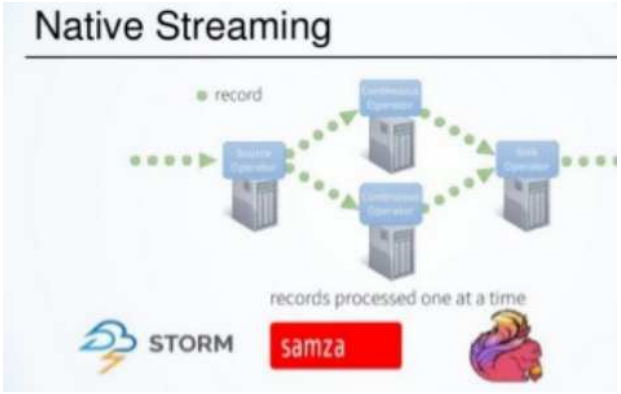
- **Stream Processing** (Traitement de flux) :
  - Contrairement au traitement par lots où les données sont liées avec un début et une fin dans un traitement qui se termine après le traitement de données finies,
  - Le Stream Processing est destiné au traitement deflux de données sans fin arrivant en temps réel de façon continue pendant des jours, des mois, des années et à jamais.
  - Le traitement de flux nous permet de traiter les données en **temps réel**
  - Le traitement de flux permet d'introduire des données dans des outils d'analyse dès qu'elles sont générées et d'obtenir des résultats **d'analyse instantanés**.



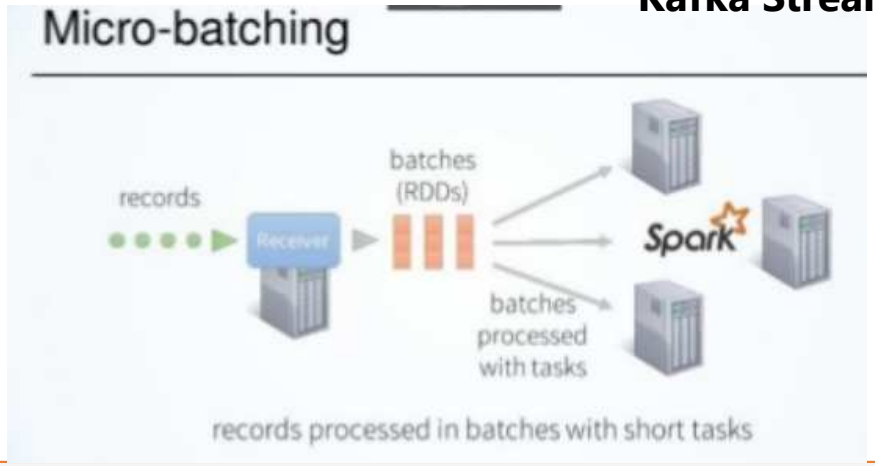
Kafka Streams

Batch Processing / Stream Processing

- **Stream processing :**
  - 2 approches pour mettre en place un Framework Streaming:
    - **Native Streaming** (**Real Time Processing**)
      - Chaque enregistrement entrant est traité dès son arrivée, sans attendre les autres.
      - Exemples: Storm, Flink, Kafka Streams, Samza.
    - **Micro Batch Processing** (Micro Batching)
      - Les enregistrements entrants toutes les quelques secondes sont mis en lots, puis traités en un seul mini-lot avec un délai de quelques secondes.
    - Exemples: Spark Streaming, Storm-Trident.



Kafka Streams



Ecosystème du Big Data

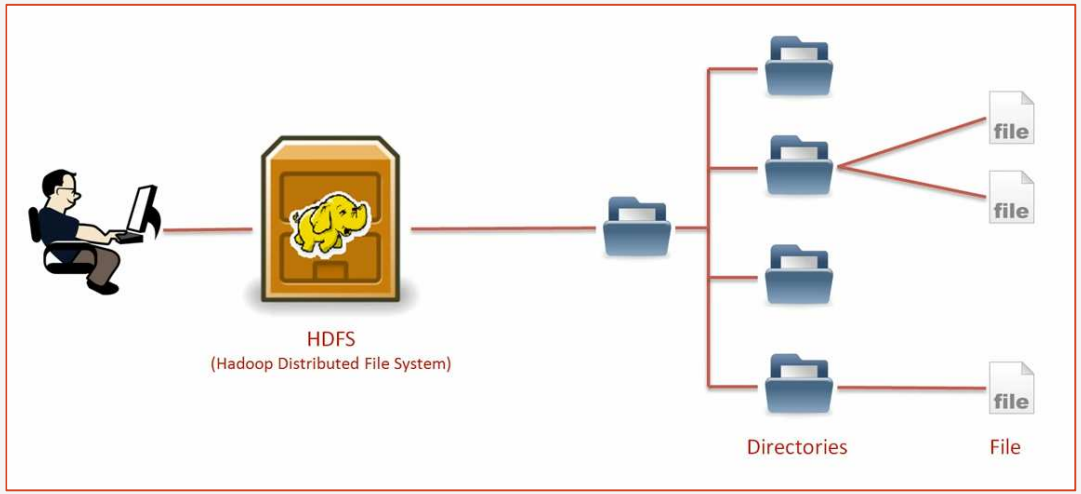
- L'écosystème du Big data s'est élargi avec beaucoup d'outils comme :
  - **Stream Processing :**
    - **Apache Storm** : Framework de calcul et de traitement distribué de flux de données (Stream Processing)
    - **Apache Flink** : Framework de calcul et de traitement distribué de flux distribués (Stream Processing)
    - **Apache Spark** : Framework de traitement distribué de données Big data (Alternative de Map Reduce) et de Stream Processing
    - **Apache Kafka Streams** : Plateforme de Streaming de données en temps réel entre les applications distribuées et Système de Messagerie applicative
  - **Apache Zookeeper** : Système de de gestion de configuration des systèmes distribués pour assurer la coordination entre les nœuds.
  - **SQBD NoSQL :**
    - **Apache Hbase** : SGBD NoSQ, distribué (Stockage structuré pour les grande tables)
    - **MangoDB** : SGBD NoSQL (Not Only SQL) : Les données sont stockées d'une manière distribuée dans plusieurs nœuds sous forme de documents au format JSON
    - **Cassandra** : SGBD NoSQL (Not Only SQL) : Les données sont stockées d'une manière distribuée dans plusieurs nœuds sous forme de documents au format JSON
    - **ElasticSearch** : Moteur de recherche distribué et multi-entité à travers une interface REST.
  - **Hazelcast :** Cache mémoire distribué, SGBD NoSQL en mémoire, Système de Messagerie applicative

Ecosystème du Big Data

- **Apache Pig** : Plateforme de haut niveau de création d'applications Map Reduce (Langage Pig Latin qui ressemble à SQL) au lieu d'écrire du code Java.
- **Apache Hive** : Infrastructure d'entrepot de données pour l'analyse et le requêtage avec un langage proche de SQL
- **Apache Phoenix** : Moteur de Base de donnée relationnel qui repose sur Hbase
- **Apache Impala** : Moteur de requêtes SQL de cloudera pour un système basé sur HDFS et Hbase
- **Apache Flume** : Système de collecte et d'analyse des fichiers logs
- **Apache Sqoop**: Outils sur ligne de commandes pour transférer les données entre les SGBD relationnels et Hadoop.
- **Apache Oozie** : outil d'ordonnancement des flux de Hadoop

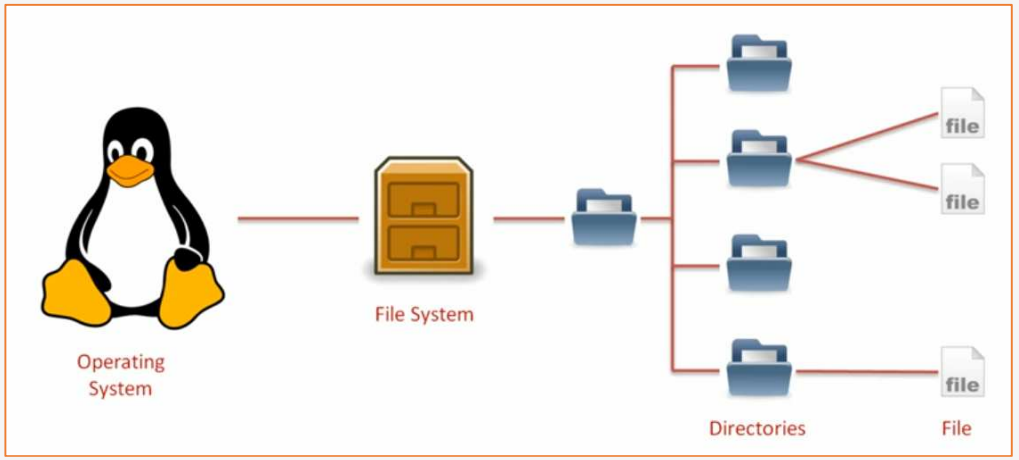
HDFS : Hadoop Distributed File System

- Solution de stockage de données massives (des PO) d'une manière distribuée.
- Tolérant aux panes avec la réplication des données.

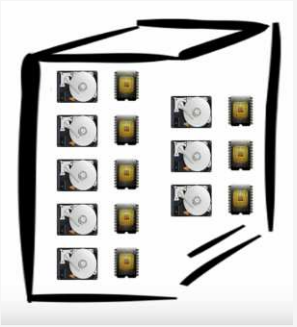


Système de Fichiers classique

- Un Système de fichiers permet de
  - Gérer le répertoires et fichiers
    - Créer, Copier, Supprimer, renommer, déplacer
    - Gérer les droits d'accès
- Les fichiers sont organisés dans une structure hiérarchique dans des répertoires.
- Les dossiers et les fichiers sont stockés dans des unités de stockage (Disque dur, Disques amovibles, etc.) locales organisées en partitions
- Chaque OS dispose de son propre système de fichiers
- Quand la quantité de données gérées atteint les limites de stockage de la machine, on cherche à étendre les capacités en ajoutant d'autres disques durs supplémentaires : Vertical Scaling (Scalabilité Verticale)

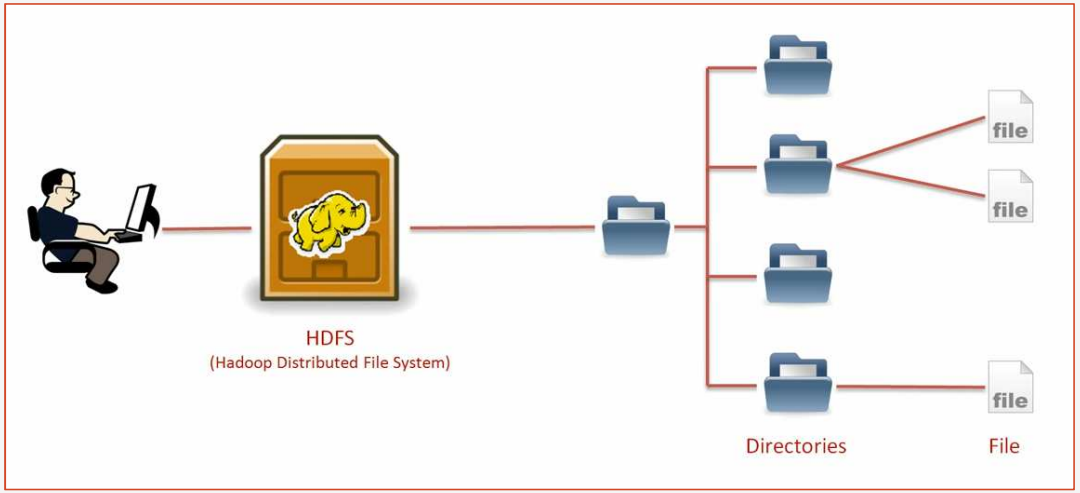


Vertical Scaling



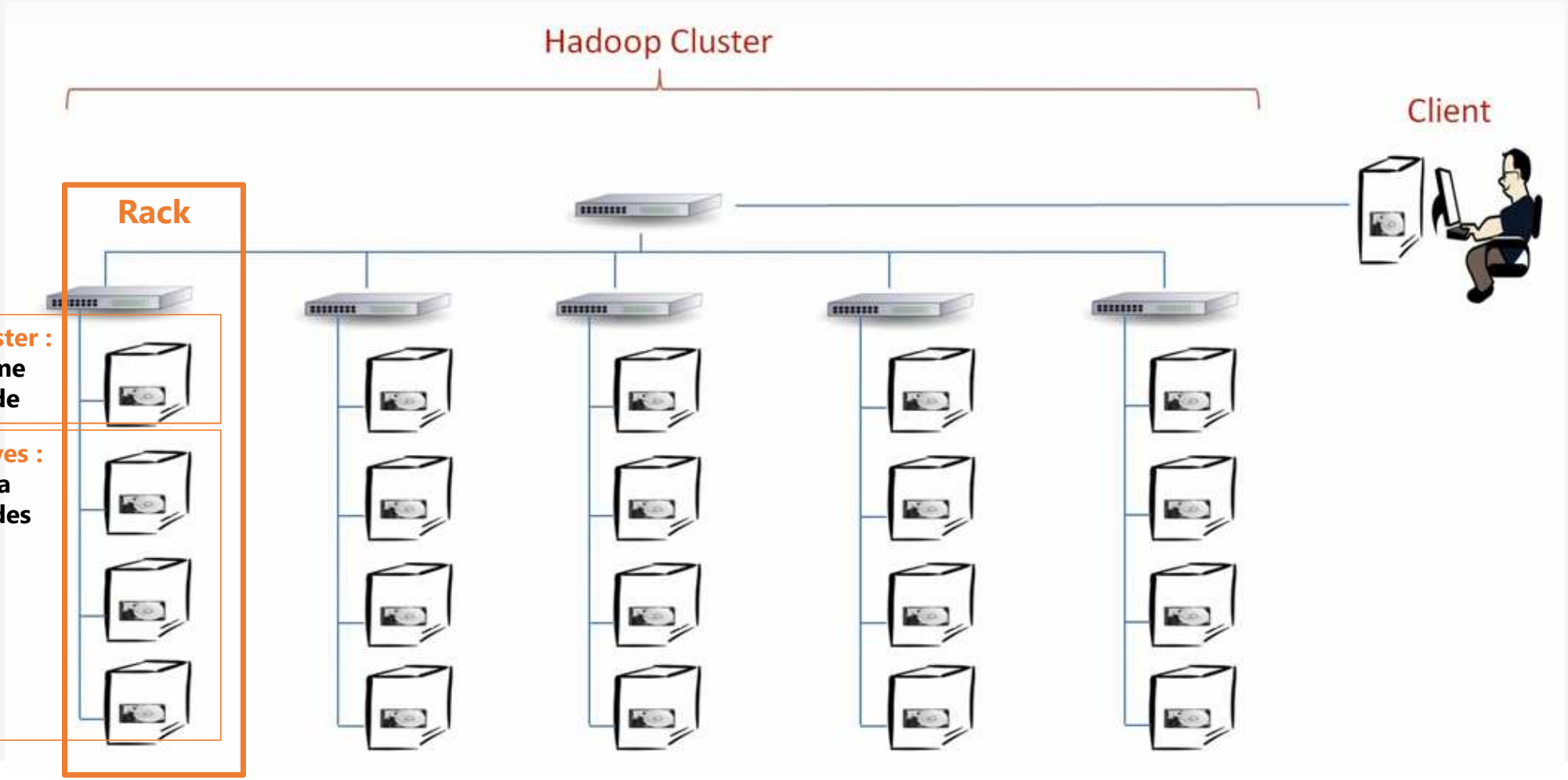
HDFS

- HDFS est aussi un système de fichiers
- Système de Fichiers Distribué
- HDFS permet de combiner les capacités de plusieurs machines distribuées sous forme d'un unique système très large.
- Il permet de créer des répertoires et stocker les données dans des fichiers d'une manière distribuée dans plusieurs machines
- Caractéristiques de HDFS
  - Distribué
  - Scalable Horizontalement
  - Cost effective (Commencer par de simple machines)
  - Fault Tolerant (Tolérent aux panes)



Horizontal Scaling

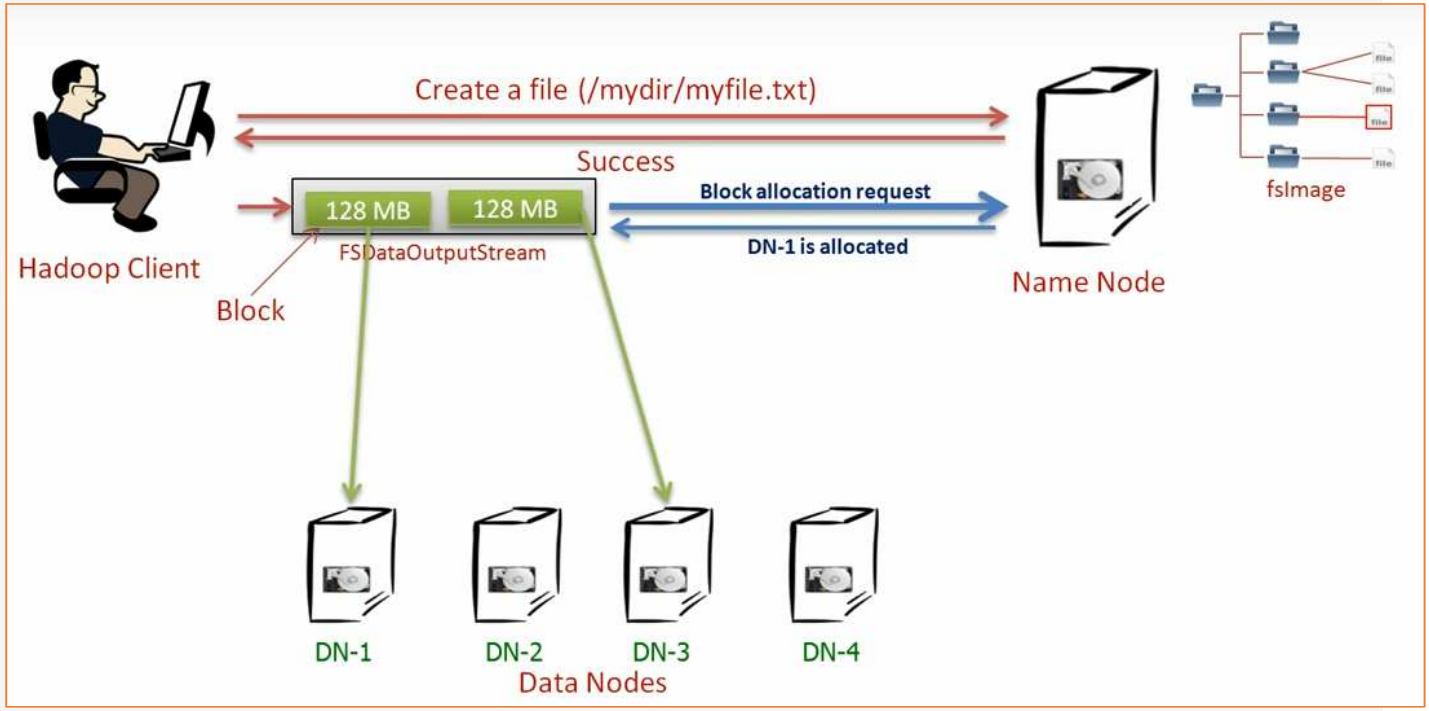
Hadoop Cluster



HDFS

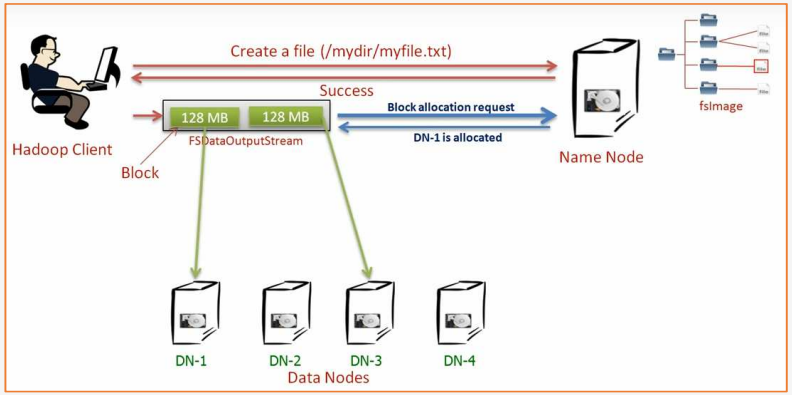
HDFS

1. Le client Hadoop envoie une requête au NN pour de mander de créer un fichier
2. Le NN procède quelques vérifications : Existence, droits d'accès en utilisant In Memory FS Image.
3. Si les vérifications passent, Le NN Crée le dossier dans FSImage et retourne success
4. Le client écrit les données dans FSDataOutputStream
5. Pour chaque block de 128 MB, le client HDFS demande au NN l'allocation du block.
6. Le NN assigne un DN pour ce block.
7. Le streamer envoie les données du block au DN
8. Une fois toutes les données sont écrites, le NN commit les modifications dans le Système de ficher



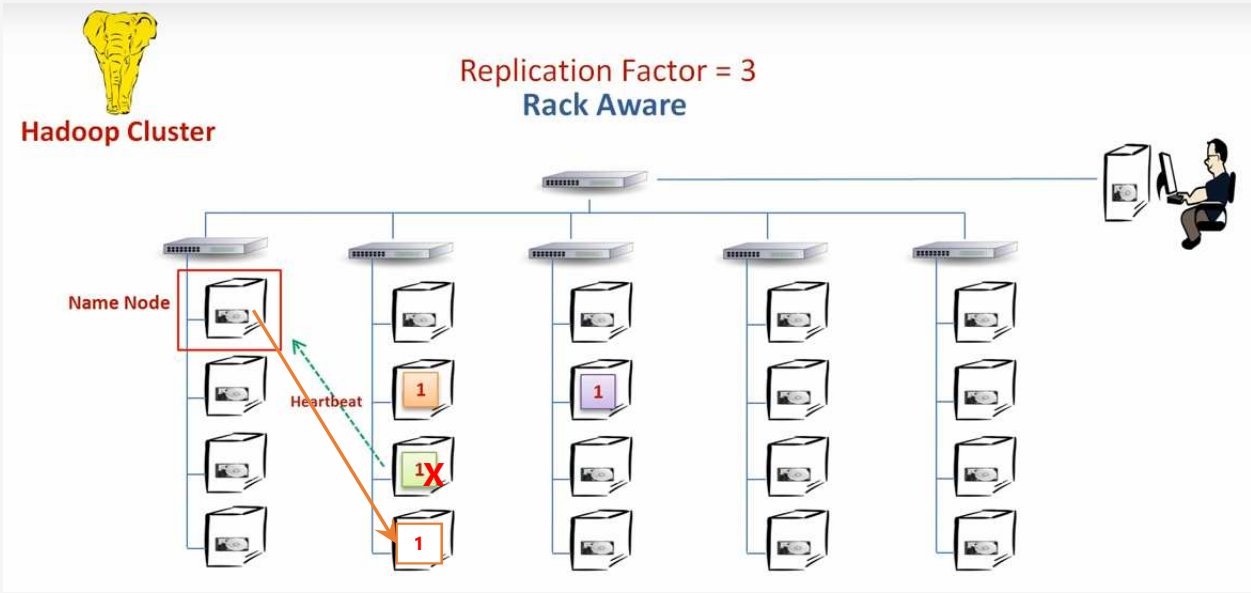
HDFS Architecture : Synthèse

- Un Cluster Hadoop se compose d'un à plusieurs Racks
- Chaque Rack a une architecture Master Esclaves
  - Un Master : Name Node
  - Un à plusieurs Esclaves : Data Nodes
- Le NN Gère l'espace de noms du système de Fichiers
- Toutes les interactions du client HDFS commencent avec NN
- Les DN stockent les données des fichiers en blocks.
- NN maintient un fichier contenant une table d'adressage des blocks des FSImage.
- Le fichier est divisé en plusieurs blocks enregistrés dans des DN.
- La taille par défaut d'un block est de 128MB.
- On peut changer la taille d'un block pour un fichier.
- Les DN envoient des Heartbeats et le rapport des blocks au NN
- Le client interagit directement avec les DN pour la lecture et l'écriture des données de bocks.



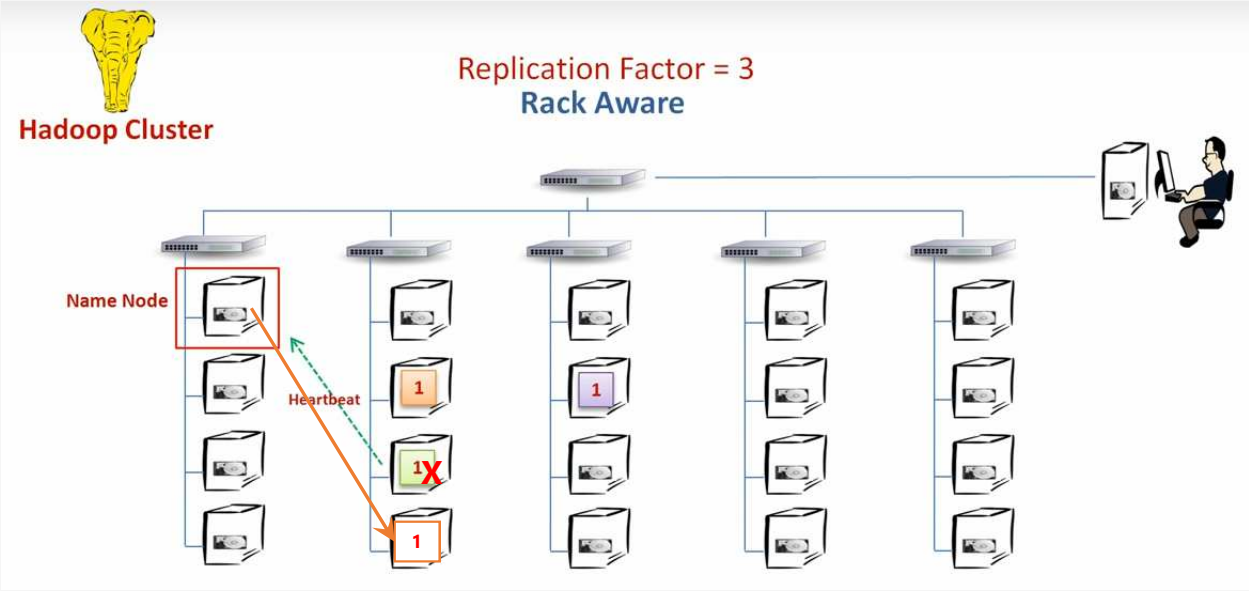
HDFS : Fault Tolerance

- Quand HDFS enregistre un block dans un DN, il enregistre également deux copies de backup dans d'autres DN.
- Si un DN tombe en panne, HDFS restore les blocks perdus à partir des duplicatas dans d'autres DN.
- Le nombre de copies est configurée dans le paramètre REPLICATION_FACTOR qui est par défaut égale à 2.
- La valeur de REPLICATION_FACTOR peut être changée pour chaque fichier.



HDFS : Fault Tolerance

- Si le mode Rack Aware est activé, HDFS s'assure que chaque copie d'un block est enregistrée dans un DN d'un Rack différents des autres copies.
- Ce qui permet de se prévenir d'un éventuel échec d'un Rack.
- Comme les DN envoient des Heart beats au NN, quand un DN échoue, NN créé à nouveau des copies des blocks de ce DN en vue de maintenir le REPLICA_FACTOR.
- Le système de réplication coute cher en termes de stockage
- Il existe des soutions pour économiser le stockage:
  - HDFS 2.0 => « Storage Policies » pour économiser le stockage
  - HDFS3.0 => « Erasure coding » comme alternative à la réplication

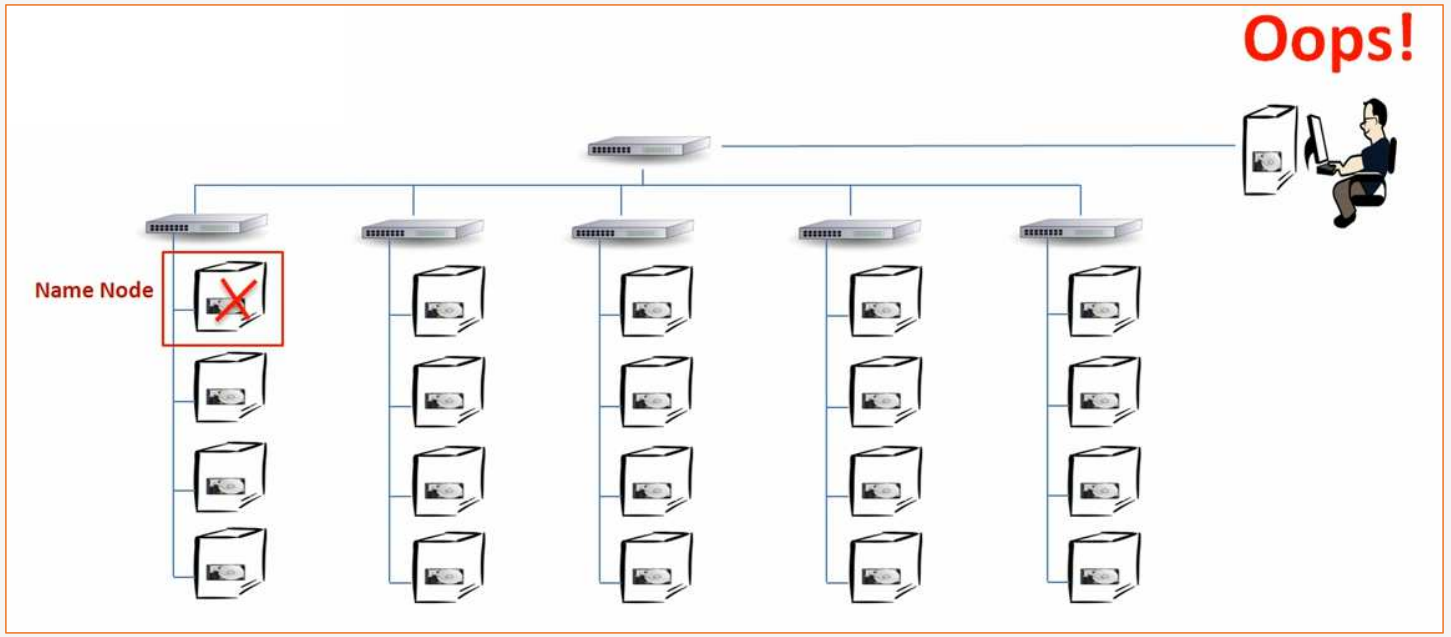


HDFS : High Availability

- La tolérance aux pannes mesure la capacité d'un système distribué à continuer à fonctionner correctement dans le cas d'une panne d'une composant du systèmes (Disque dur, CPU, Ordinateur, Switch, Rack, etc..)
- La haute disponibilité mesure le pourcentage de disponibilité des services offerts par un système à cours, moyen et à long termes.

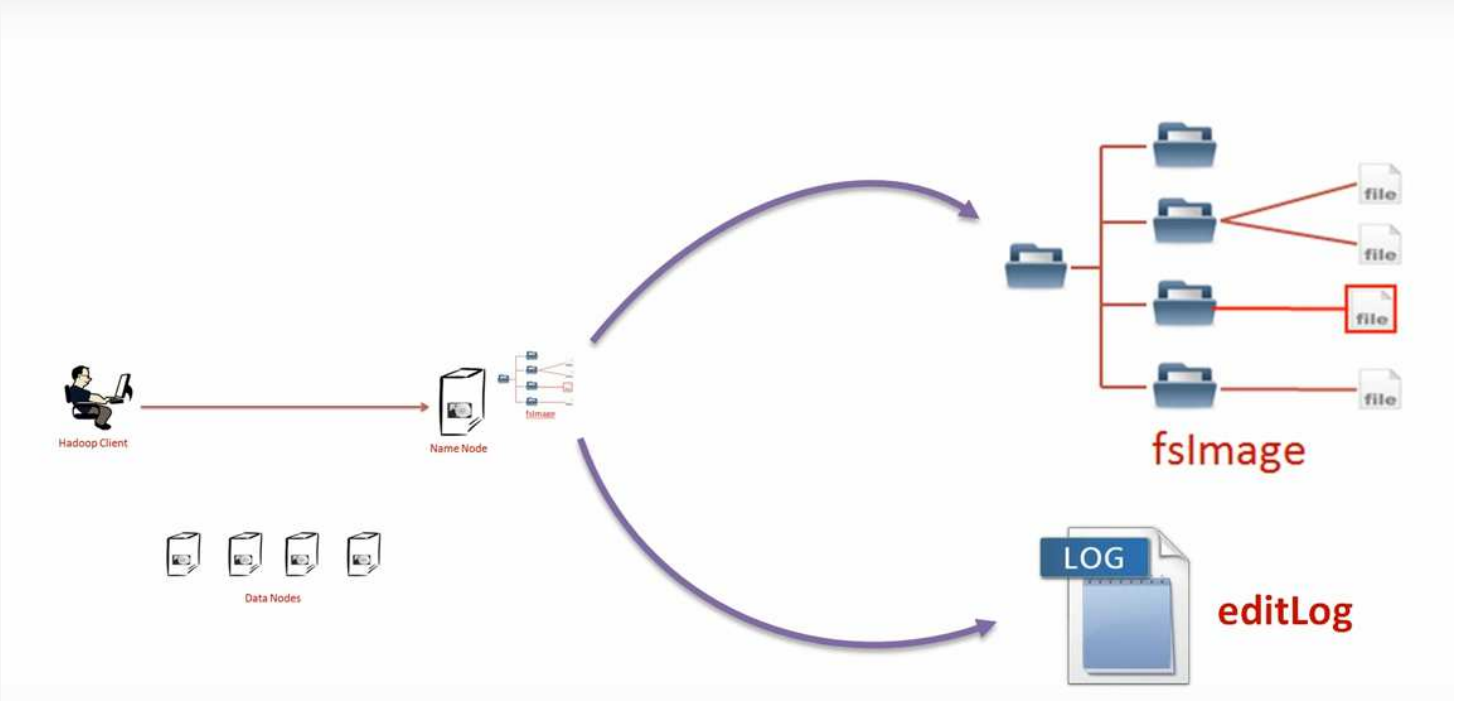
HDFS : High Availability

- The Name Node is a Single Point Of Failure in a Hadoop Cluster
- How to protect the NN Failure ?



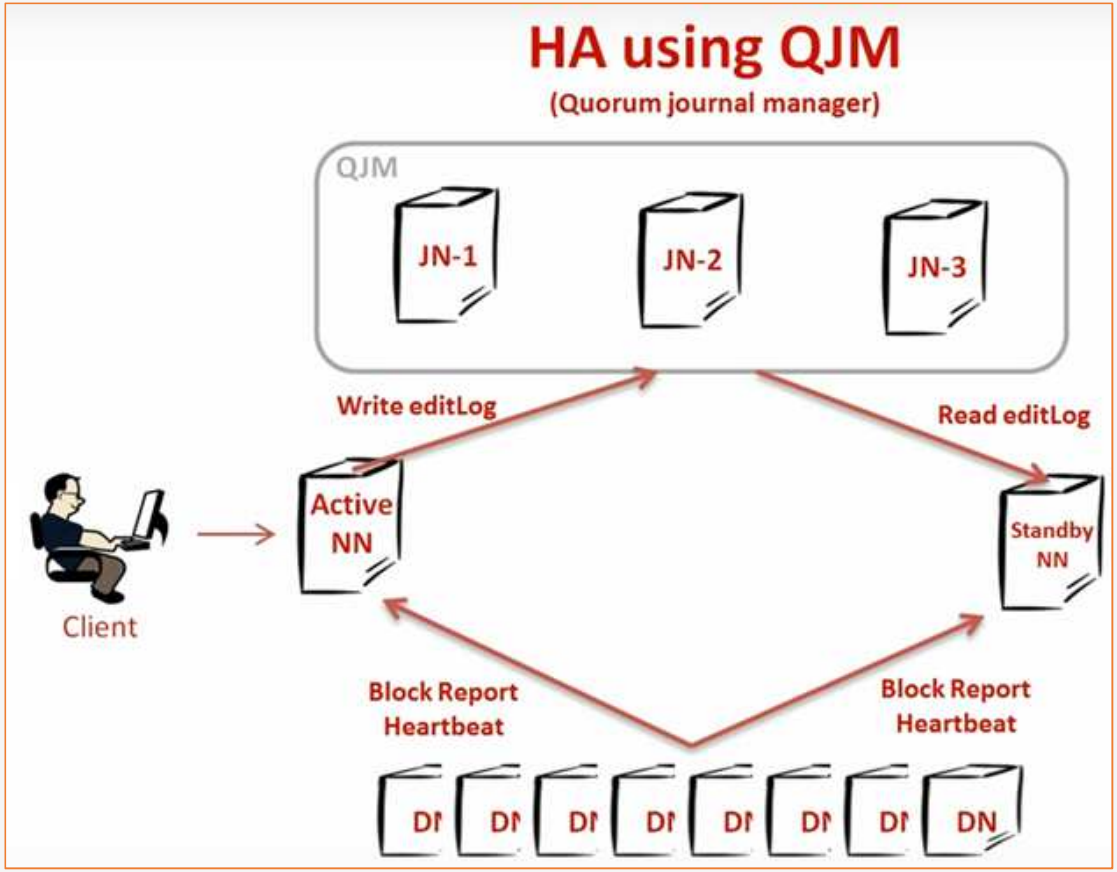
HDFS : High Availability

- The Name Node maintains the entire file system in memory (In Memoty FS image)
- NN, also maintains an edit log in its local disc
- Every time, the NN change every things in the file system, it records changes in the edit log.
- If we have the edit log, we can construct the In Memory FS image



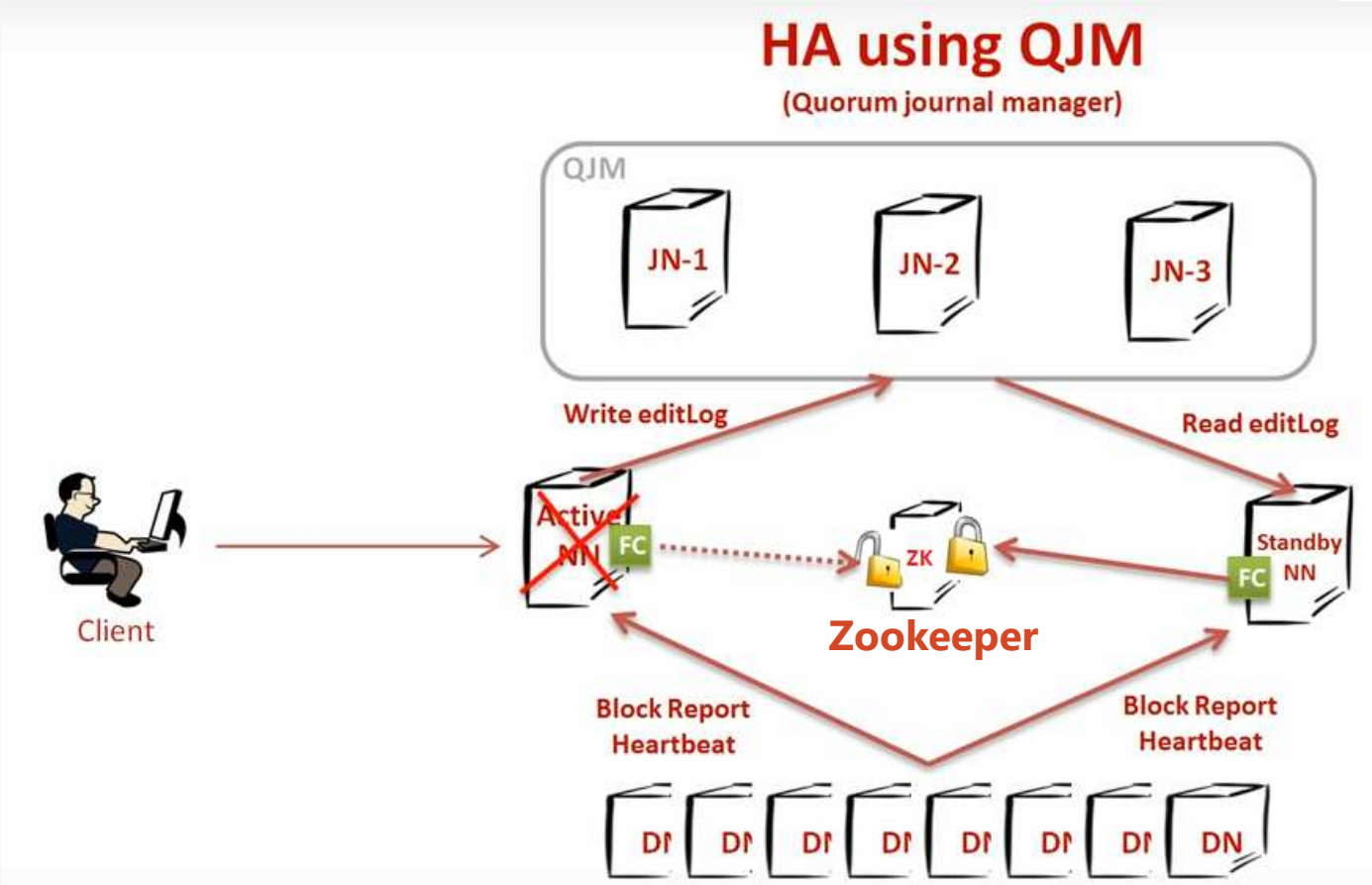
HDFS : High Availability (Quorum Journal Manager)

- The QJM is a set of 3 machines
- Each of these 3 machines execute is configured to execute a journal node daemon.
- Journal Node is a very light weight software, so you can pick any 3 computers of the Hadoop cluster.
- Once you have QJM, the next thing to do is to configure the NN to write the edit log in the QJM instead of the local disk.
- The StandBy NN is a cluster node that is configured to keep it reading the edit log from the QJM
- All the data nodes are configured to send the blocks reports (Blocks Health informations) to both the two Name Nodes



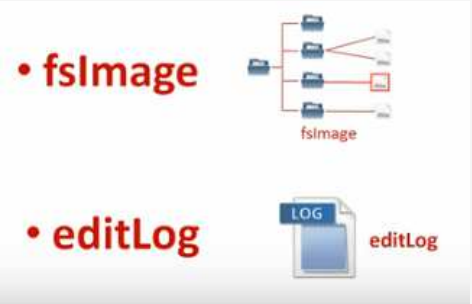
HDFS : High Availability (Quorum Journal Manager)

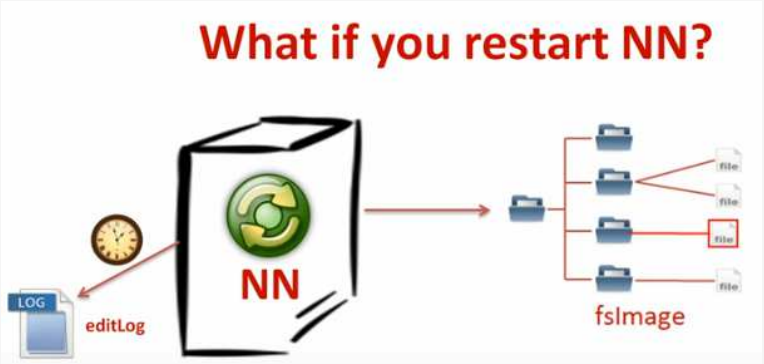
- How the standby Name Node knows the Active Name Node fails and it should took the Active Name Node Role?
- He achieves this by using zookeeper and Fail Over Controllers, that coordinate between Active NN and Standby NN
- Active NN maintains a lock in Zookeeper.
- The StandBy NN streams to get the lock
- In case of the Active NN fails or crashes, the lock expires
- So, the StandBy NN succeed de get the lock and starts the transition from a StandBy NN to Active NN.



HDFS : High Availability (Secondary Name Node)

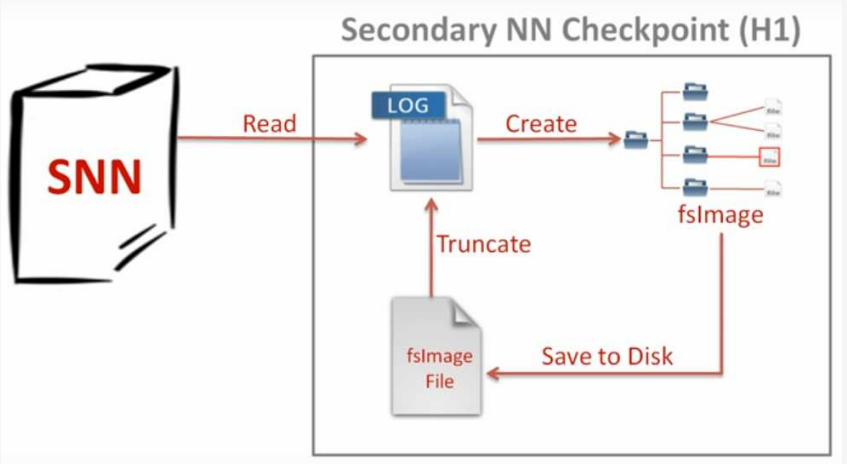
- The Secondary Name Node has a different role than StandBy NN.
- The Secondary NN takes care of a different responsibility.
- The InMemoryFS Image is the latest updated picture of the Hadoop file system Name space.
- The EditLog is the persistent copy of all changers made to the file system
- When we restart the NN, we lost the InMemeory FS Image.
- The NN will reconstruct the InMemeory FS Image By reading data from editLog.
- This operation may take a lot of time because of the editLog becomes bigger and bigger every day.
- The secondary NN is deployed to solve this problem.

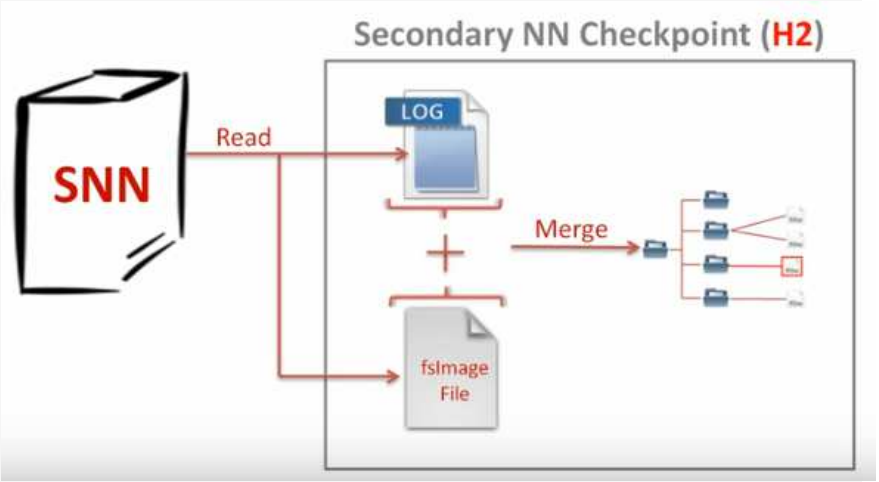




HDFS : High Availability (Secondary Name Node)

- The Secondary Name Node performs a cheek point activity every hour
- During the cheek point,
  - The SecNN reads the EditLog
  - Create the latest state of the File System
  - Save The latest File System in the disk
  - This File is exactly the same of InMemory FS image. It calls On Disk FS Image.
  - Once we have the On Disk FS Image, The Secondary NN truncate the Edit Log because all changers are already applied,
- Next Cheek Point,
  - The SNN read On Disk FS Image
  - Apply all the changes from the edit log that accumulated from the last hour.
  - Replace the old On Disk FS Image by the new One
  - Truncate the editLog once again.
- The time to read the On Disk FS Image is small because the file is smaller than the edit Log.

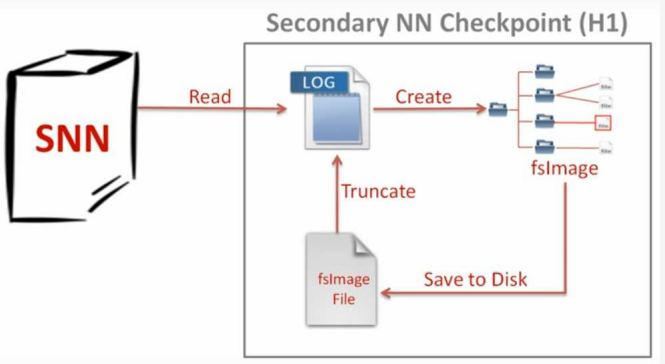


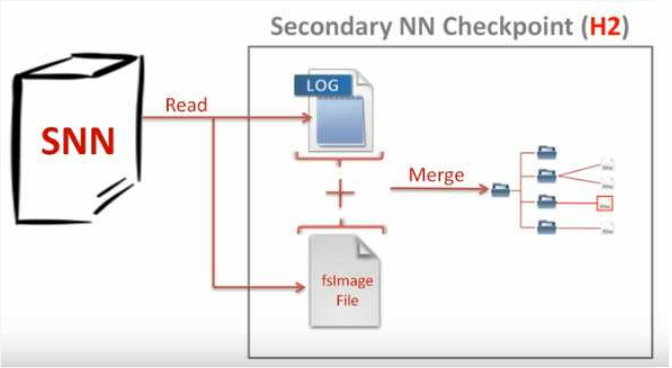


HDFS : Installation

- Télécharger Hadoop :
- Installation de Java
- Téléchargement de Hadoop

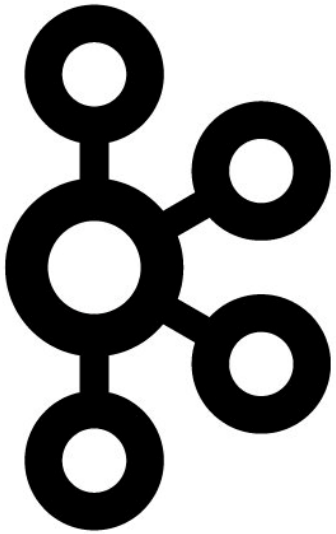
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz



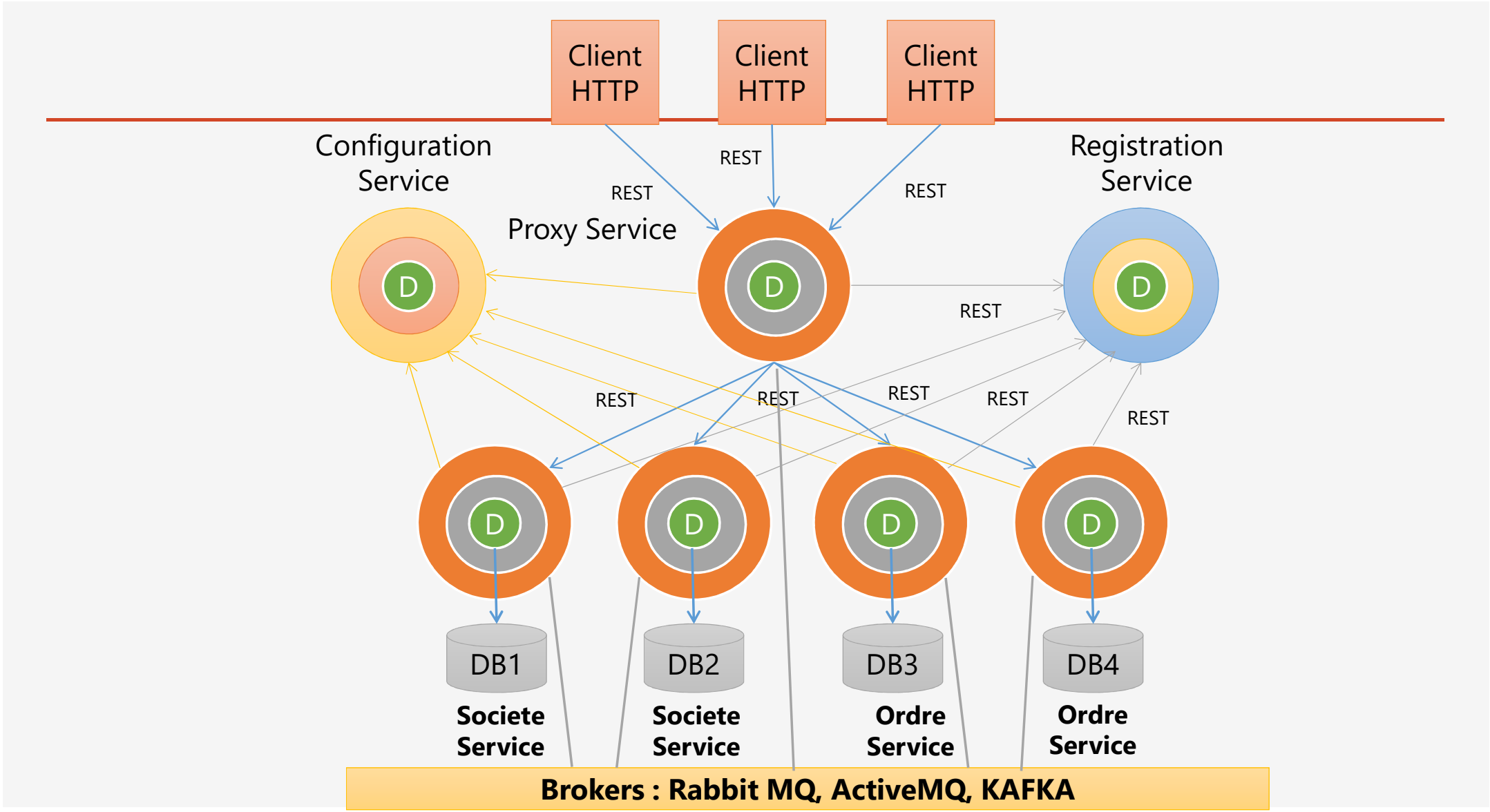


Distributed Big Data Stream Processing avec KAFKA

KAFKA

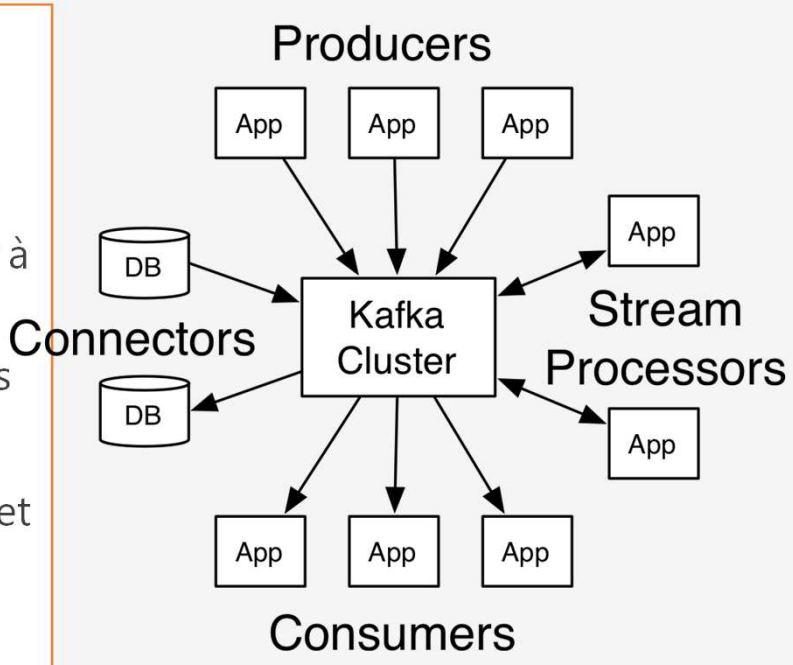


Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications


Client HTTP
Client HTTP
Client HTTP
Configuration Service
Proxy Service
Registration Service
REST
REST
REST
REST
REST
REST
REST
REST
REST
D
DB1
DB2
DB3
DB4
Societe Service
Societe Service
Ordre Service
Ordre Service
Brokers : Rabbit MQ, ActiveMQ, KAFKA

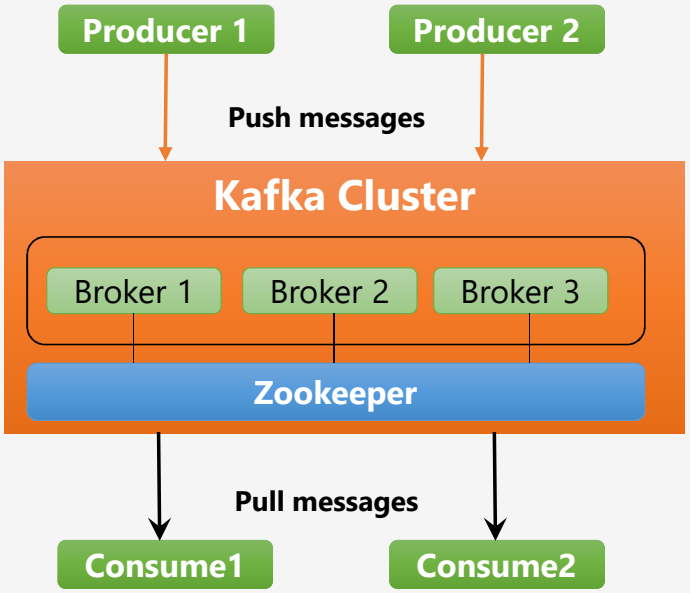
KAFKA

- Apache Kafka est une plate-forme de diffusion (streaming) distribuée développée en Scala et Java.
- Une plate-forme de streaming possède trois fonctionnalités clés:
  - Permettre aux applications clientes Kafka de publiez et s'abonner à des flux d'enregistrements. Similaires à une file d'attente de messages ou à un système de messagerie d'entreprise comme les Brokers JMS (ActiveMQ) ou AMQP (RabbitMQ)
  - Permet de stocker les flux d'enregistrements de manière durable et tolérante aux pannes.
  - Permet de traiter les flux d'enregistrements au fur et à mesure qu'ils se produisent (Real Time Stream Processing)



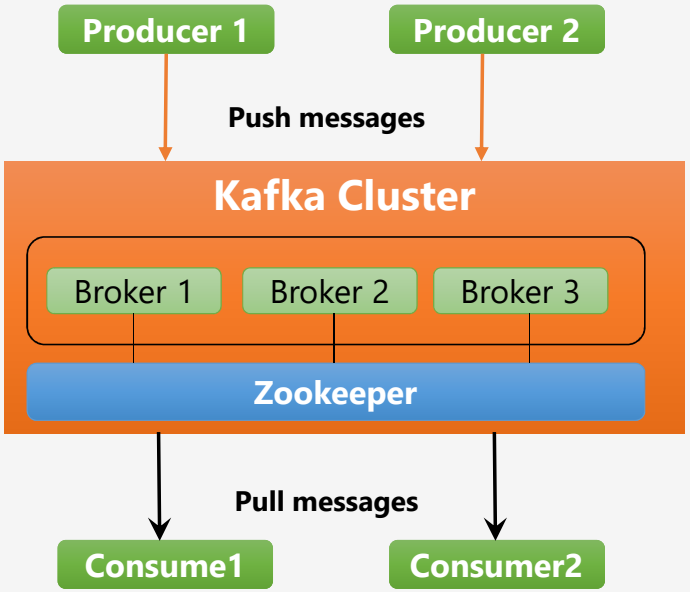
Les API de Kafka

- Kafka a quatre API principales:
  - **Producer API**: Permet à une application de publier un flux d'enregistrements vers un ou plusieurs Topics (Sujets) Kafka.
  - **Consumer API**: Permet à une application de s'abonner à un ou plusieurs Topics et de traiter le flux d'enregistrements qui lui sont transmis.
  - **Streams API**: Permet à une application d'agir en tant que processeur de flux, en
    - Consommant un flux d'entrée provenant d'un ou plusieurs Topics
    - Transformant efficacement les flux d'entrée en flux de sortie
    - Produisant un flux de sortie vers un ou plusieurs Topics en sortie.
  - **Connector API:** Permet de créer et d'exécuter des producteurs ou des consommateurs réutilisables qui connectent des topics Kafka à des applications ou des systèmes de données existants. Par exemple, un connecteur vers une base de données relationnelle peut capturer chaque modification apportée à une table.
- Kafka fourni des API clientes pour différents langages : Java, C++, Node JS, .Net, PHP, Python, etc…



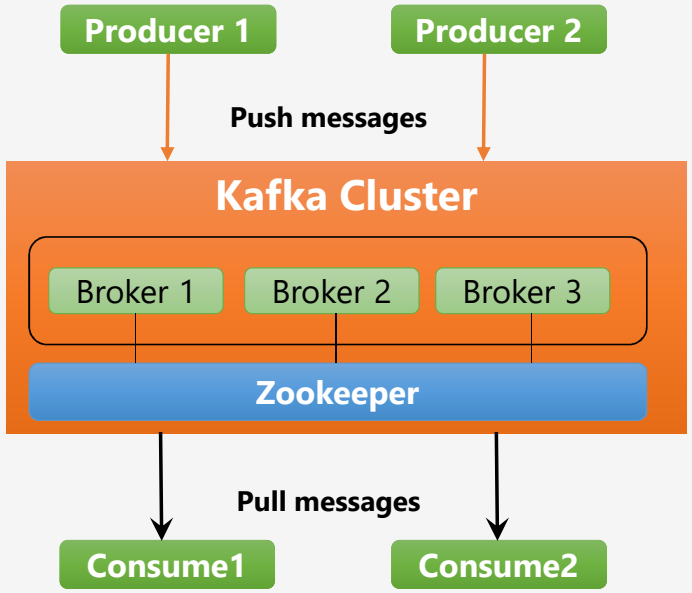
Architecture de KAFKA

- **Kafka Brokers**
  - Pour maintenir l'équilibre de la charge, le cluster Kafka est généralement composé de plusieurs Brokers dont un est élu comme Leader.
  - Cependant, ils sont sans état, c'est pourquoi ils utilisent **ZooKeeper** pour conserver l'état du cluster.
  - Une instance de Kafka Broker peut gérer des centaines de milliers de lectures et d'écritures par seconde.
  - Chaque Broker peut gérer des To de messages.



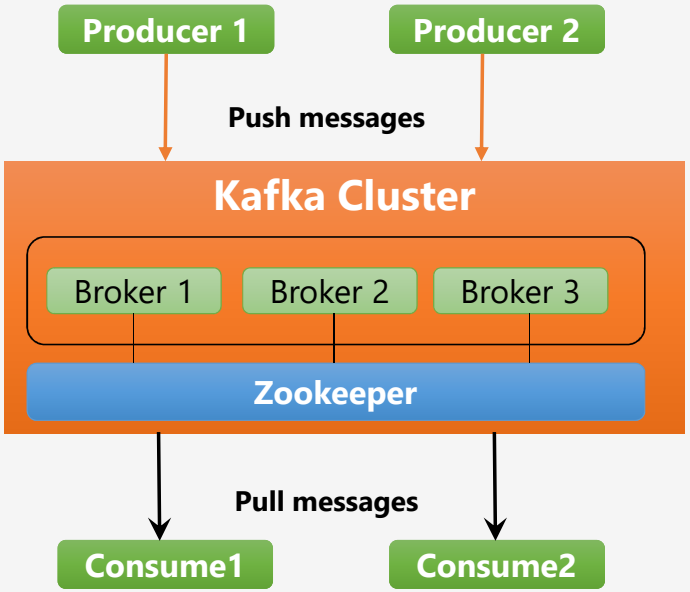
Architecture de KAFKA

- **ZooKeeper :**
  - Kafka broker utilise ZooKeeper pour la gestion et la coordination.
  - Il l'utilise également pour informer le producteur et le consommateur de la présence d'un nouveau broker dans le système Kafka ou d'une défaillance du broker dans le système Kafka.



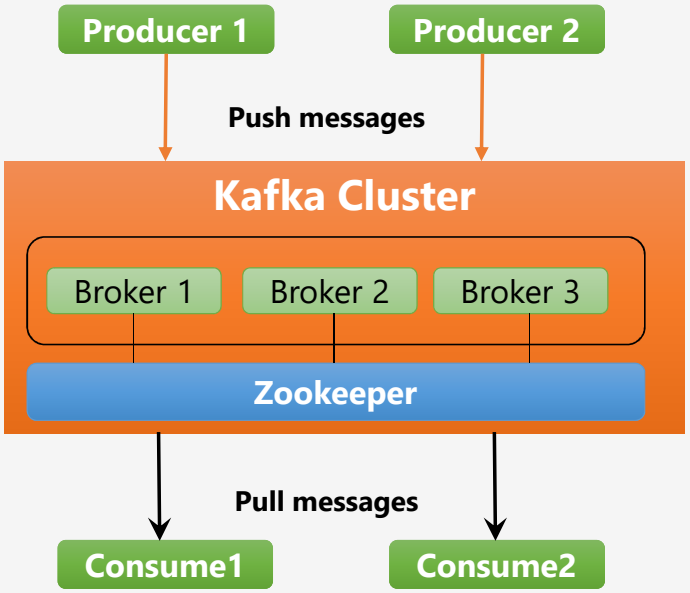
Architecture de KAFKA

- **Producers :**
  - les producteurs de Kafka transmettent les données aux Brokers.
  - Ces données sont à destination des Topics réparties en partitions dans les brokers du cluster
  - Le producteur est responsable du choix de l'enregistrement à affecter à quelle partition du cluster.



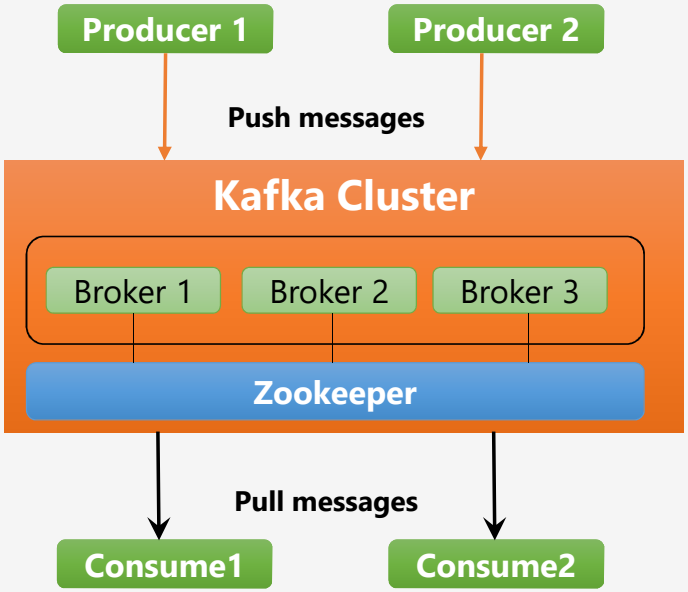
Architecture de KAFKA

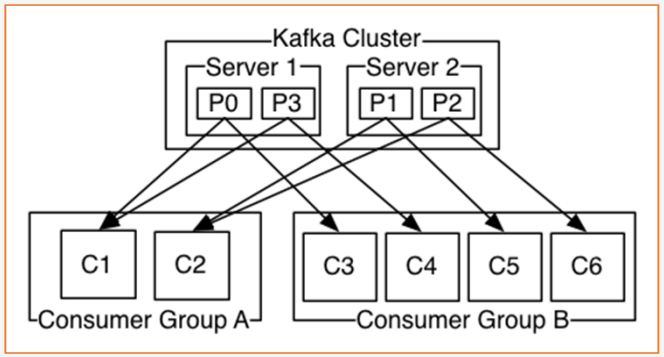
- **Consumers :**
  - Consomme les messages à partir des Topics.
  - En utilisant l'offset de la partitions de Kafka Broker, le consommateur Kafka retient combien de messages ont été consommés parce que les Brokers Kafka sont sans état.
  - Les utilisateurs peuvent rembobiner ou passer à n'importe quel point d'une partition.
  - Chaque enregistrement publié dans un Topic est remis à une instance de consommateur au sein de chaque groupe de consommateurs abonné.



Architecture de KAFKA

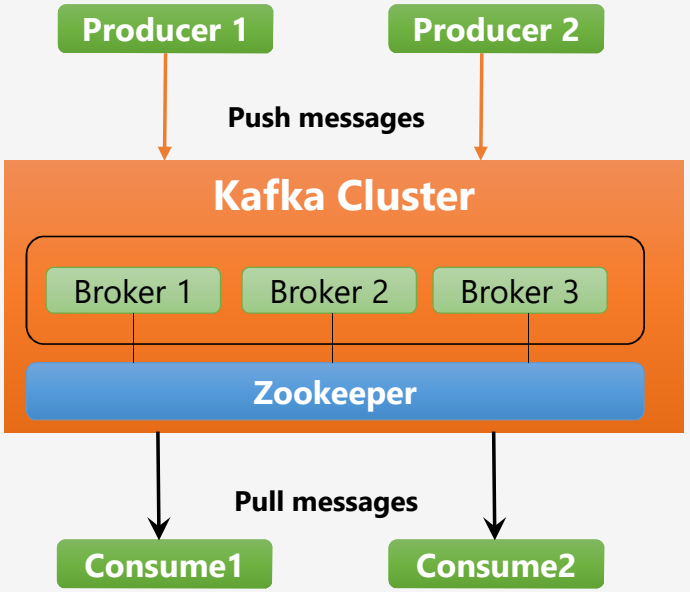
- **Consumers :**
  - Les instances de consommateur peuvent être dans des processus séparés ou sur des machines séparées.
  - Si toutes les instances de consommateur ont le même groupe de consommateurs, la charge des enregistrements sera effectivement équilibrée sur les instances de consommateur.
  - Si toutes les instances de consommateur ont des groupes de consommateurs différents, chaque enregistrement est alors diffusé vers tous les processus de consommation.

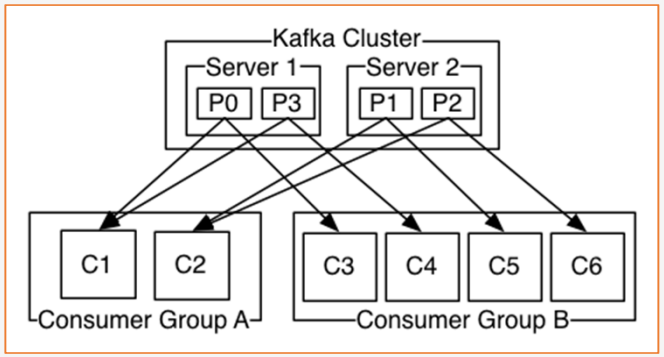




Architecture de KAFKA

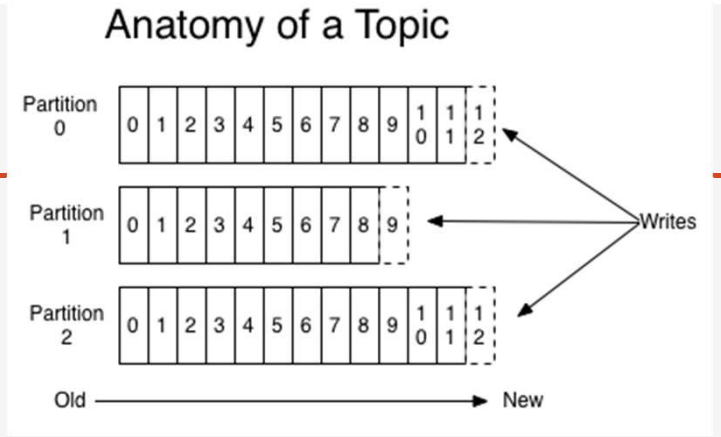
- **Consumer-Group :**
  - Les consommateurs peuvent être organisés en groupes.
  - Des partitions de Topics sont assignées pour équilibrer les assignations entre tous les consommateurs du groupe.
  - Chaque message sera vu par un consommateur du groupe.
  - Si un consommateur s'en va, la partition est assignée à un autre consommateur du groupe.
  - S'il y a plus de consommateurs dans un groupe que de partitions, certains consommateurs resteront inactifs.
  - S'il y a moins de consommateurs dans un groupe que de partitions, certains consommateurs consommeront des messages provenant de plus d'une partition.

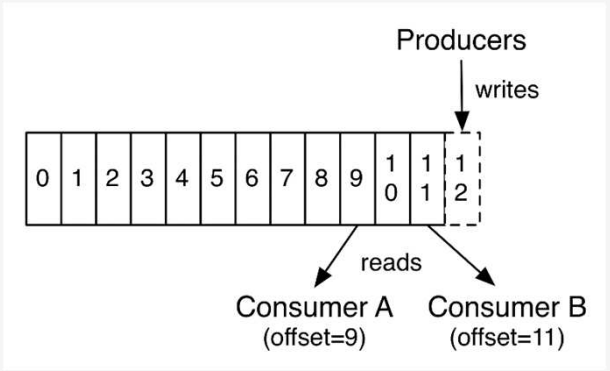


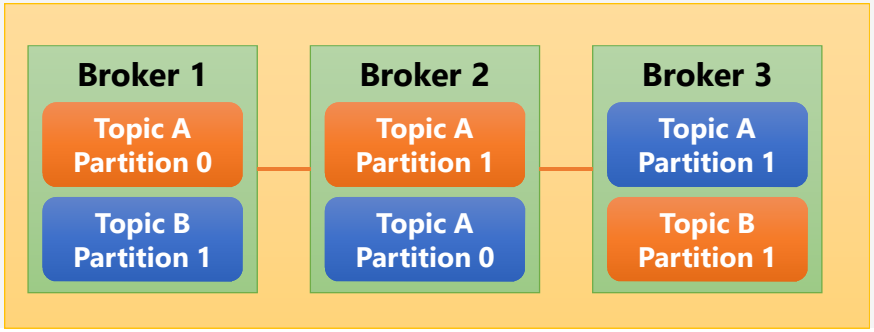


Topics

- Un Topic est une sorte de boite à lettre vers laquelle les messages (enregistrements) sont publiés
- Chaque Topic peut avoir 0 à plusieurs abonnés consommateurs
- Pour chaque topic, le cluster Kafka maintient un journal (Log) partitionné.
- Chaque partition est une séquence d'enregistrements ordonnée et immutable
- Un numéro d'identification séquentiel, appelé offset, est attribué aux enregistrements des partitions. Il identifie de manière unique chaque enregistrement de la partition.
- Le cluster Kafka conserve durablement tous les enregistrements publiés, qu'ils aient été consommés ou non, en utilisant une période de conservation configurable.
- Les performances de Kafka sont en réalité constantes en ce qui concerne la taille des données, de sorte que leur stockage pendant longtemps ne pose pas de problème.

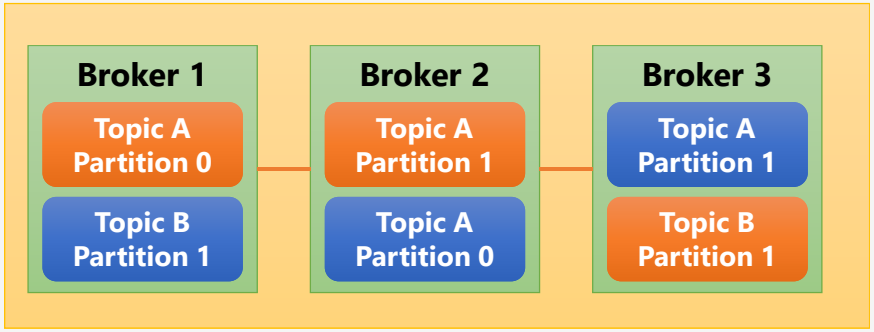






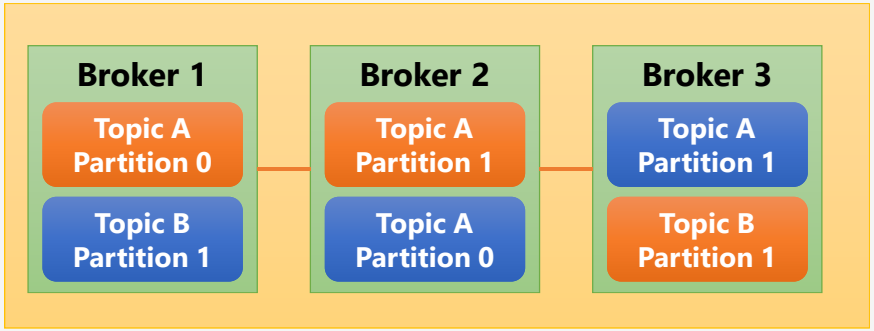
Distribution

- Les partitions du log sont distribuées sur les serveurs du cluster Kafka
- Chaque partition est répliquée sur un nombre configurable de serveurs pour la tolérance aux pannes.
- Chaque partition a un serveur qui joue le rôle de "leader" et zéro ou plusieurs serveurs qui servent de « Followers ».
- Le leader gère toutes les demandes de lecture et d'écriture pour la partition, tandis que les Followers répliquent passivement le leader.
- Si le leader échoue, l'un des Followers deviendra automatiquement le nouveau leader.
- Chaque serveur joue le rôle Leader pour certaines de ses partitions et Follower pour d'autres, de sorte que la charge soit bien équilibrée au sein du cluster.



Geo-Replication

- Kafka MirrorMaker fournit une prise en charge de la réplication géographique pour vos clusters.
- Avec MirrorMaker, les messages sont répliqués dans plusieurs Data centers ou régions de cloud.
- Vous pouvez l'utiliser dans des scénarios
  - Actifs / passifs pour la sauvegarde et la récupération.
  - ou dans des scénarios actifs / actifs pour rapprocher les données de vos utilisateurs ou pour répondre aux exigences de localisation des données.



Installation de Kafka

- Téléchargement et décompression

```
$ wget http://miroir.univ-lorraine.fr/apache/kafka/2.3.0/kafka_2.12-2.3.0.tgz

$ tar -xzf kafka_2.12-2.3.0.tgz
```

Démarrage de KAFKA Server

- Start zookeeper

```
$ cd kafka_2.12-2.3.0
$ bin/zookeeper-server-start.sh config/zookeeper.properties

[2019-09-08 11:43:36,135] INFO binding to port 0.0.0.0/0.0.0.0:2181
(org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Démarrage de KAFKA Server

- Start kafka server

```
$ cd kafka_2.12-2.3.0
$ bin/kafka-server-start.sh config/server.properties
...
[2019-09-08 11:44:54,652] INFO Initiating client connection, connectString=localhost:2181 sessionTimeout=6000
watcher=kafka.zookeeper.ZooKeeperClient$ZooKeeperClientWatcher$@48ae9b55 (org.apache.zookeeper.ZooKeeper)
[2019-09-08 11:44:54,692] INFO [ZooKeeperClient Kafka server] Waiting until connected.
(kafka.zookeeper.ZooKeeperClient)
[2019-09-08 11:44:54,694] INFO Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to
authenticate using SASL (unknown error) (org.apache.zookeeper.ClientCnxn)
[2019-09-08 11:44:54,722] INFO Socket connection established to localhost/127.0.0.1:2181, initiating session
(org.apache.zookeeper.ClientCnxn)
[2019-09-08 11:44:54,805] INFO Session establishment complete on server localhost/127.0.0.1:2181, sessionid =
0x100000219690000, negotiated timeout = 6000 (org.apache.zookeeper.ClientCnxn)
[2019-09-08 11:44:54,811] INFO [ZooKeeperClient Kafka server] Connected. (kafka.zookeeper.ZooKeeperClient)
...
[2019-09-08 11:44:56,489] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.Acceptor)
...
[2019-09-08 11:44:57,477] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
```

Création d'un Topic

- Création d'un Topic

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic test
Created topic test.
$ bin/kafka-topics.sh --list --zookeeper localhost 2181
test
```

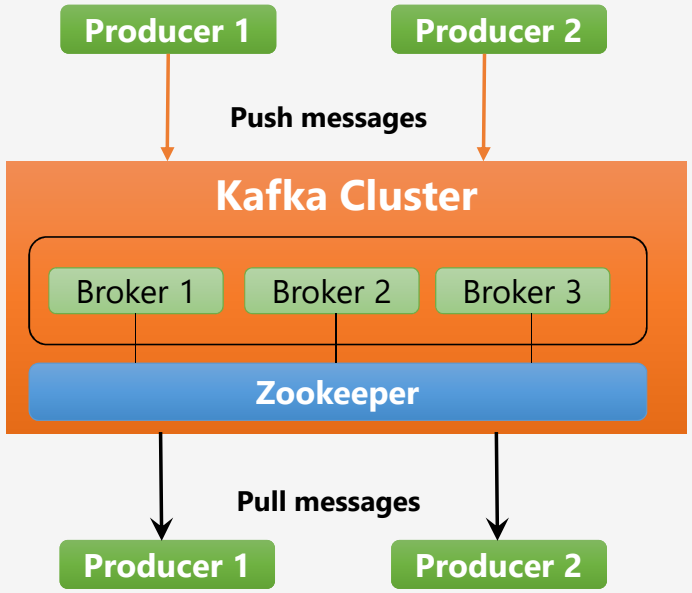
S'abonner à un Topic pour consommer des messages

- S'abonner au topic test pour consommer les messages

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic test --from-beginning
    test
    azerty
    me
    you
    and me
```

- Produire des messages vers le topic

```
$ bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
    >test
    >azerty
    >me
    >you
    >and me
    >
```



Produire des messages vers le topic test

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
>test
>azerty
>me
>you
>and me
>
```

Démarrage d'un Cluster

```
$ cp config/server.properties config/server-1.properties
$ cp config/server.properties config/server-2.properties
```

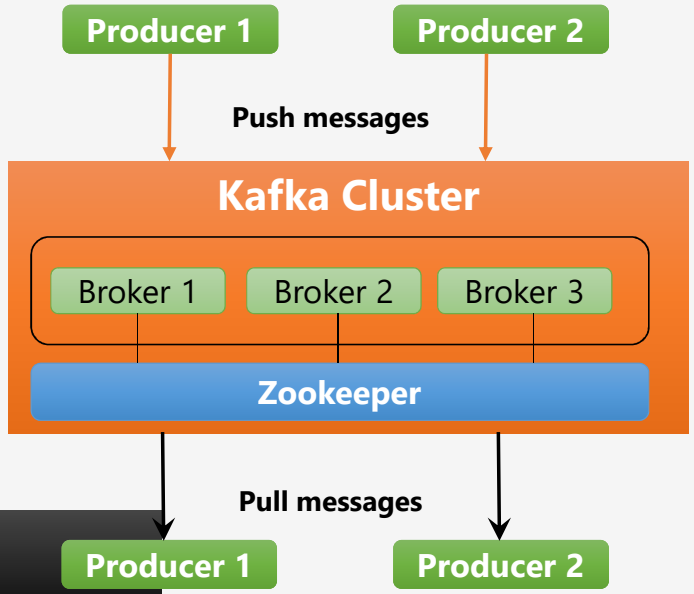
config/server-1.properties:

```
broker.id=1
listeners=PLAINTEXT://:9093
log.dirs=/tmp/kafka-logs-1
```

config/server-2.properties:

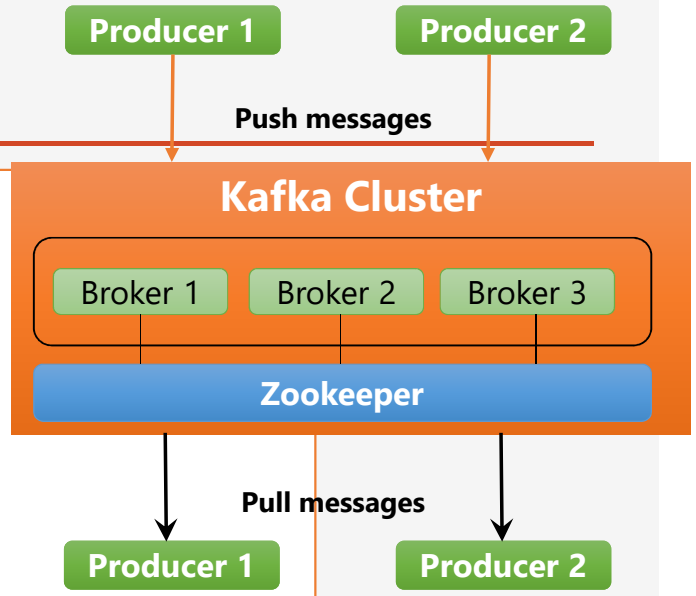
```
broker.id=2
listeners=PLAINTEXT://:9094
log.dirs=/tmp/kafka-logs-2
```

```
$ bin/kafka-server-start.sh config/server-1.properties
...
$ bin/kafka-server-start.sh config/server-2.properties
...
```



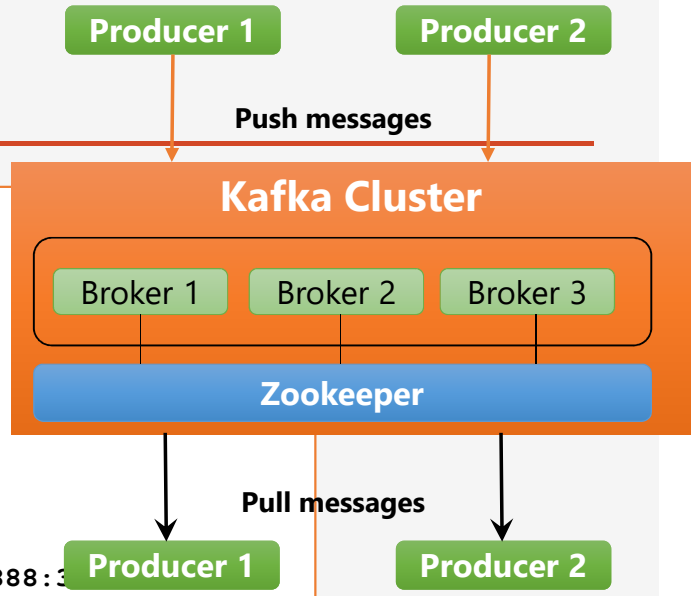
Démarrage d'un Cluster avec Docker-compose

```
version: '2'
services:
  zookeeper-1:
    image: confluentinc/cp-zookeeper:latest
    hostname: zookeeper-1
    ports:
      - "12181:12181"
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 12181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-2:22888:23888;zookeeper-3:32888:33888

  zookeeper-2:
    image: confluentinc/cp-zookeeper:latest
    hostname: zookeeper-2
    ports:
      - "22181:22181"
    environment:
      ZOOKEEPER_SERVER_ID: 2
      ZOOKEEPER_CLIENT_PORT: 22181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-2:22888:23888;zookeeper-3:32888:33888
```



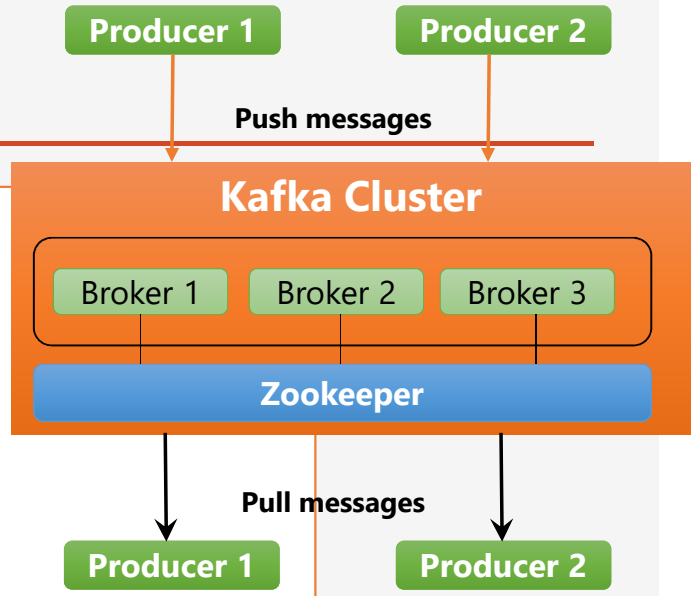
Démarrage d'un Cluster avec Docker-compose

```
zookeeper-3:
    image: confluentinc/cp-zookeeper:latest
    hostname: zookeeper-3
    ports:
      - "32181:32181"
    environment:
      ZOOKEEPER_SERVER_ID: 3
      ZOOKEEPER_CLIENT_PORT: 32181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-2:22888:23888;zookeeper-3:32888:3

  kafka-1:
    image: confluentinc/cp-kafka:latest
    hostname: kafka-1
    ports:
      - "19092:19092"
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:12181,zookeeper-3:12181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-1:19092
```



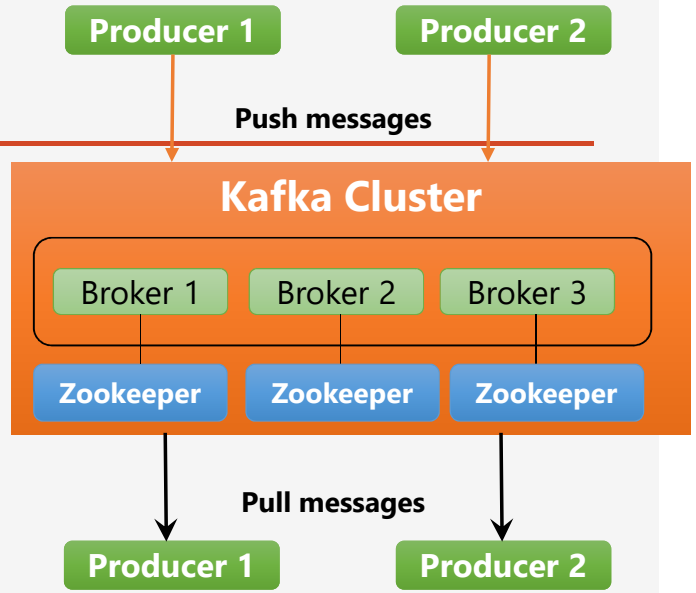
Démarrage d'un Cluster avec Docker-compose

```
kafka-2:
    image: confluentinc/cp-kafka:latest
    hostname: kafka-2
    ports:
      - "29092:29092"
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:12181,zookeeper-3:12181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-2:29092

  kafka-3:
    image: confluentinc/cp-kafka:latest
    hostname: kafka-3
    ports:
      - "39092:39092"
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:12181,zookeeper-3:12181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-3:39092
```



Démarrage d'un Cluster avec Docker-compose

```
C:\tpdockcompose\kafka_cluster>dir
 Le volume dans le lecteur C s'appelle OS
 Le numéro de série du volume est CE70-C12F

 Répertoire de C:\tpdockcompose\kafka_cluster

15/10/2019  11:38    <DIR>          .
15/10/2019  11:38    <DIR>          ..
15/10/2019  11:38             2.381 docker-compose.yml
               1 fichier(s)            2.381 octets
               2 Rép(s)  36.303.675.392 octets libres

C:\tpdockcompose\kafka_cluster>docker-compose up
Creating network "kafka_cluster_default" with the default driver
Creating kafka_cluster_zookeeper-1_1 ... done
Creating kafka_cluster_zookeeper-3_1 ... done
Creating kafka_cluster_zookeeper-2_1 ... done
Creating kafka_cluster_kafka-1_1     ... done
Creating kafka_cluster_kafka-2_1     ... done
Creating kafka_cluster_kafka-3_1     ...
```



Tester la tolérance aux panes

- Now let's test out fault-tolerance. Broker 1 was acting as the leader so let's kill it:

```
$ ps aux | grep server-1.properties
youssfi   8179 13.1 16.2 3633872 331472 pts/3  Sl+  16:48   0:06 java -Xmx1G -Xms1G -server -
XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -
XX:+ExplicitGCInvokesConcurrent -Djava.a>
$ kill -9 8179
On Windows use:
> wmic process where "caption = 'java.exe' and commandline like '%server-1.properties%'" get processid
ProcessId
6016
> taskkill /pid 6016 /f
```

- Leadership has switched to one of the followers and node 1 is no longer in the in-sync replica set, But the messages are still available for consumption even though the leader

```
$ bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic my-replicated-topic
Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
    Topic: my-replicated-topic  Partition: 0    Leader: 2   Replicas: 1,2,0 Isr: 2,0
```

Exemple d'application Java : Producer/Consumer

```xml
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.3.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka_2.12</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>2.3.0</version>
    </dependency>

</dependencies>
```

Exemple Consumer Java

```
import org.apache.kafka.clients.consumer.ConsumerConfig; import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer; import java.time.Duration; import java.util.Collections;
import java.util.Properties; import java.util.concurrent.Executors;import java.util.concurrent.TimeUnit;
public class StreamConsumer {
    private String KAFKA_BROKER_URL="192.168.43.22:9092";    private String TOPIC_NAME="testTopic";
    public static void main(String[] args) {
        new StreamConsumer();
    }
    public StreamConsumer() {
        Properties properties=new Properties();
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,KAFKA_BROKER_URL);
        properties.put(ConsumerConfig.GROUP_ID_CONFIG,"sample-group-test");
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG,"true");
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG,"1000");
        properties.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG,"30000");
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        KafkaConsumer<Integer, String> kafkaConsumer = new KafkaConsumer<Integer, String>(properties);
        kafkaConsumer.subscribe(Collections.singletonList(TOPIC_NAME));
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
            System.out.println("----------------------------------------");
            ConsumerRecords<Integer,String> consumerRecords=kafkaConsumer.poll(Duration.ofMillis(10));
            consumerRecords.forEach(cr->{
                System.out.println("Key=>"+cr.key()+", Value=>"+cr.value()+", offset=>"+cr.offset());
            });
        },1000,1000, TimeUnit.MILLISECONDS);

    }
}
```

Exemple Consumer Java

```
Console App :
Key=>null, Value=>A, offset=>9
Key=>null, Value=>B, offset=>10
Key=>null, Value=>C, offset=>11
Key=>null, Value=>X, offset=>12
Key=>null, Value=>B, offset=>13
Key=>null, Value=>X, offset=>14
Key=>null, Value=>P, offset=>15
```

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --
topic testTopic
>A
>B
>C
>X
>B
>X
>P
```

Exemple Producer Kafka Java

```
import org.apache.kafka.clients.producer.KafkaProducer; import org.apache.kafka.clients.producer.ProducerRecord;
import java.util.Properties; import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
public class StreamProducer {
    private int counter;
     private String KAFKA_BROKER_URL="192.168.43.22:9092";
     private String TOPIC_NAME="testTopic";    private String clientID="client_prod_1";
     public static void main(String[] args) {
         new StreamProducer();
     }
     public StreamProducer() {
         Properties properties = new Properties();
         properties.put("bootstrap.servers", KAFKA_BROKER_URL);
         properties.put("client.id", clientID);
         properties.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
         properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
         KafkaProducer<Integer, String> producer = new KafkaProducer<>(properties);
         Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
             ++counter;String msg=String.valueOf(Math.random()*1000);
             producer.send(new ProducerRecord<Integer, String>(TOPIC_NAME,++counter,msg),
                     (metadata,ex)->{
                         System.out.println("Sending Message key=>"+counter+" Value =>"+msg);
                         System.out.println("Partition => "+metadata.partition()+" Offset=>"+metadata.offset());
                     });
         },1000,1000, TimeUnit.MILLISECONDS);
     }
}
```

Exemple Producer Kafka Java

Consumer

```
Key=>32, Value=>609.6765398053385, offset=>3191
Key=>34, Value=>767.7805809027903, offset=>3192
Key=>36, Value=>787.1553570236956, offset=>3193
Key=>38, Value=>445.8611979566287, offset=>3194
Key=>40, Value=>497.64168259174227, offset=>3195
Key=>42, Value=>757.0605546528924, offset=>3196
Key=>44, Value=>547.2387037943823, offset=>3197
Key=>46, Value=>805.6218738236041, offset=>3198
Key=>48, Value=>199.4860895170022, offset=>3199
...
```

Producer en mode asynchrone

```
Sending Message key=>180 Value =>573.491814792023
Partition => 0 Offset=>3265
Sending Message key=>182 Value =>470.02504583700664
Partition => 0 Offset=>3266
Sending Message key=>184 Value =>988.7152347439281
Partition => 0 Offset=>3267
...
```

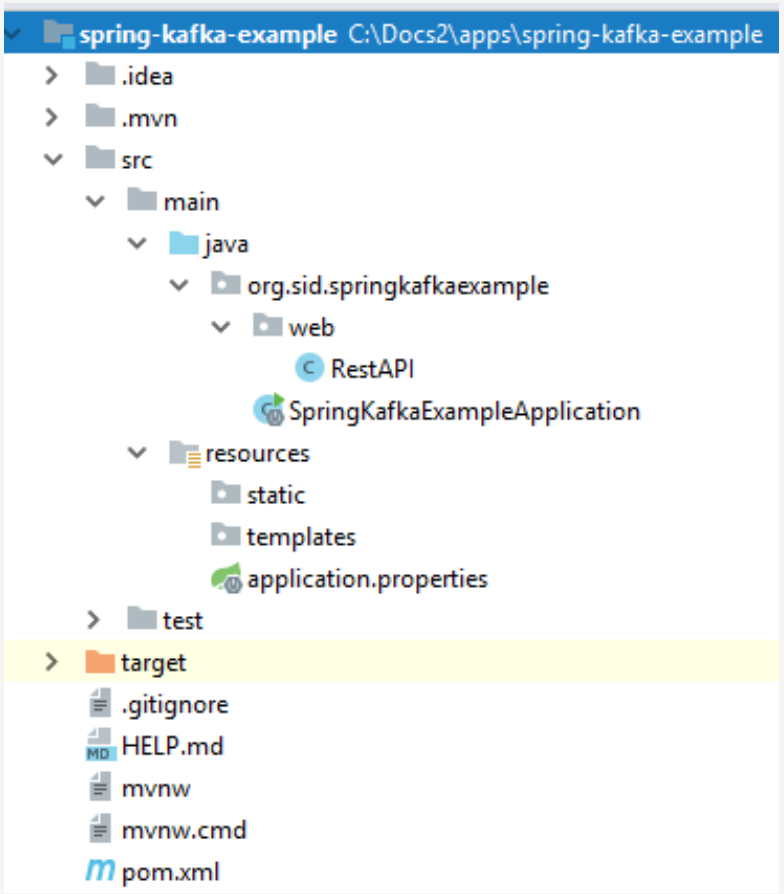
Application Web : Spring et Kafka

Maven dependencies

```
<dependency>
  <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

appliaction.properties

```
server.port=8083
spring.kafka.producer.bootstrap-servers=192.168.57.3:9092
```

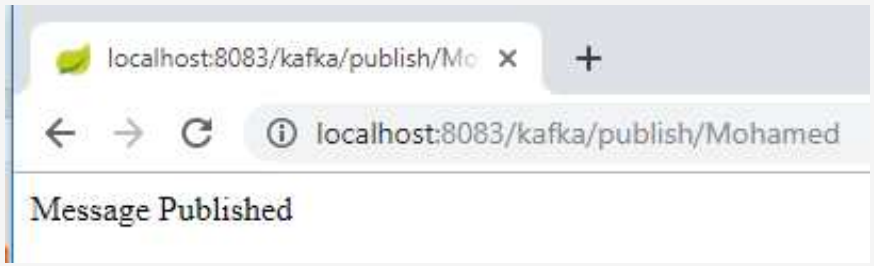


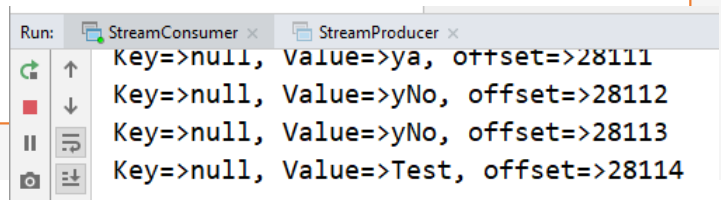
Application Web : Spring et Kafka (Producer, Consumer)

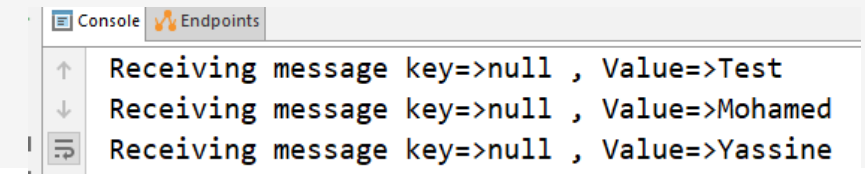
Maven dependencies

```
package org.sid.springkafkaexample.web;
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping("kafka")
public class RestAPI {
    @Autowired
    private KafkaTemplate<Integer, String>
kafkaTemplate;
    private String topic="testTopic";
    @GetMapping("/publish/{message}")
    public String publishMessage(@PathVariable String
message){
        kafkaTemplate.send(topic,message);
        return "Message Published";
    }
}
```

```
package org.sid.demokafka;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;
@Service
public class KafkaConsumer {
    @KafkaListener(topics = "testTopic",groupId = "sample_consumer")
    public void onMessage(ConsumerRecord message){
        System.out.println("Receiving message key=>"+message.key()+"" +
                " , Value=>"+message.value());
    }
}
```







Application Web : Spring et Kafka

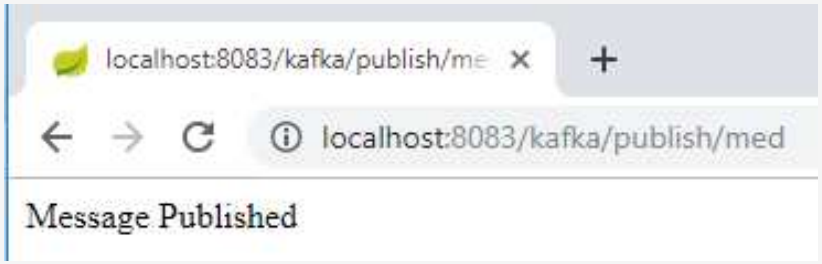
Maven dependencies

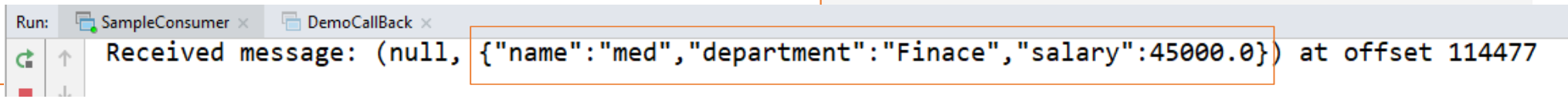
```
package org.sid.springkafkaexample.web;
import org.sid.springkafkaexample.model.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("kafka")
public class RestAPI {
     @Autowired
    private KafkaTemplate<Integer, Employee> kafkaTemplate;
    private String topic="testTopic";
    @GetMapping("/publish/{name}")
    public String publishMessage(@PathVariable String name){
        kafkaTemplate.send(topic,new
Employee(name,"Finace",45000.0));
        return "Message Published";
    }
}
```

```
package org.sid.springkafkaexample.model;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data @NoArgsConstructor @AllArgsConstructor
@ToString
public class Employee {
    private String name;
    private String department;
    private double salary;
}
```

appliaction.properties

```
server.port=8083
spring.kafka.producer.bootstrap-servers=192.168.57.3:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```





Application Web : Spring et Kafka

En Utilisant une classe de configuration au lieu de application.properties

```
package org.sid.springkafkaexample.config;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.sid.springkafkaexample.model.Employee; import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration; import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate; import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.serializer.JsonSerializer;
import java.util.HashMap; import java.util.Map;
@Configuration
public class KafkaConfig {
    @Bean
    ProducerFactory<String, Employee> producerFactory(){
        Map<String,Object> config=new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
       return new DefaultKafkaProducerFactory<>(config);
    }
    @Bean
    KafkaTemplate<String,Employee> kafkaTemplate(){
        return new KafkaTemplate<>(producerFactory());
    }
}
```

appliaction.properties

```
server.port=8083
#spring.kafka.producer.bootstrap-servers=192.168.57.3:9092
#spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
#spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```

Application Web : Spring Kafka Consumer

En Utilisant une classe de configuration au lieu de application.properties

```xml
<dependency>
   <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

```
spring.kafka.consumer.bootstrap-servers=192.168.57.3:9092
spring.kafka.consumer.group-id=enset_uh2c
```

```
kafka-consumer C:\Docs2\apps\kafka-consum
  .idea
  .mvn
  src
    main
      java
        org.sid.kafkaconsumer
          service
            KafkaConsumer
          KafkaConfig
          KafkaConsumerApplication
      resources
        application.properties
  HELP.md
  kafka-consumer.iml
  mvnw
  mvnw.cmd
  pom.xml
```

```java
package org.sid.kafkaconsumer.service; import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;
@Service
public class KafkaConsumer {
    @KafkaListener(topics = {"testTopic"},groupId = "enset_uh2c")
    public void onMessage(String message){
        System.out.println("Consume ====>:"+message);
    }
}
```

```java
package org.sid.kafkaconsumer; import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.kafka.annotation.EnableKafka;
@SpringBootApplication
@EnableKafka
public class KafkaConsumerApplication {
    public static void main(String[] args) {
        SpringApplication.run(KafkaConsumerApplication.class, args);
    }
}
```

```
Consume ====>:{"name":"med","department":"Finace","salary":45000.0}
Consume ====>:f
Consume ====>:Simple message
```

Application Web : Spring Kafka Consumer avec Classe de configuration

En Utilisant une classe de configuration au lieu de application.properties

```
package org.sid.kafkaconsumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;import
org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.config.KafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory; import
org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import java.util.HashMap; import java.util.Map;
@Configuration
public class KafkaConfig {
    @Bean
    ConsumerFactory<String,String> consumerFactory(){
        Map<String, Object> config=new HashMap<>();
        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        config.put(ConsumerConfig.GROUP_ID_CONFIG,"enset_uh2c");
        config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,StringDeserializer.class);
        return new DefaultKafkaConsumerFactory<>(config);
    }
    @Bean
    ConcurrentKafkaListenerContainerFactory<String,String> kafkaListenerContainerFactory(){
        ConcurrentKafkaListenerContainerFactory factory=new
ConcurrentKafkaListenerContainerFactory();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }

}
```

```
Consume ====>:{"name":"med","department":"Finace","salary":45000.0}
Consume ====>:f
Consume ====>:Simple message
```

appliaction.properties

```
#spring.kafka.consumer.bootstrap-servers=192.168.57.3:9092
#spring.kafka.consumer.group-id=enset_uh2c
```

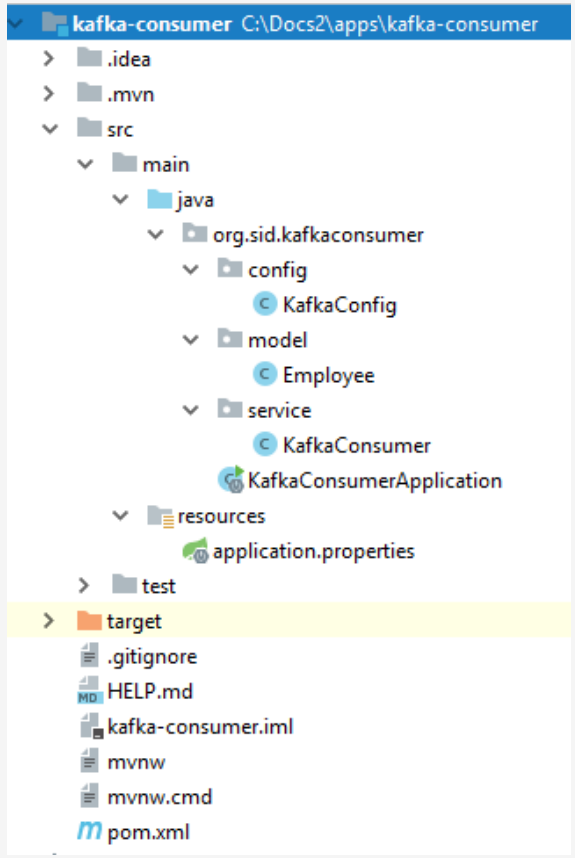
Application Web : Spring Kafka Consumer

Désérialisation des données structuré au format JSON en Objet Java Employee

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>
```

Employee.java

```java
package org.sid.kafkaconsumer.model;
import lombok.AllArgsConstructor; import lombok.Data;
import lombok.NoArgsConstructor; import lombok.ToString;
@Data @NoArgsConstructor @AllArgsConstructor @ToString
public class Employee {
    private String name;
    private String department;
    private double salary;
}
```



Application Web : Spring Kafka Consumer

KafkaConfig.java

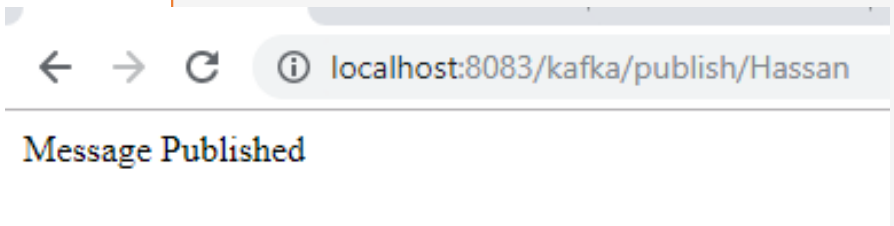
```
package org.sid.kafkaconsumer.config;
import org.apache.kafka.clients.consumer.ConsumerConfig; import org.apache.kafka.common.serialization.StringDeserializer;
import org.sid.kafkaconsumer.model.Employee; import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration; import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory; import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;
import java.util.HashMap; import java.util.Map;
@Configuration
public class KafkaConfig {
    @Bean
    ConsumerFactory<String, Employee> consumerFactory(){
        Map<String, Object> config=new HashMap<>();
        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        config.put(ConsumerConfig.GROUP_ID_CONFIG,"enset_uh2c");
        JsonDeserializer<Employee> jsonDeserializer=new JsonDeserializer<>(Employee.class);
        jsonDeserializer.addTrustedPackages("*");
        jsonDeserializer.setUseTypeHeaders(false);
        return new DefaultKafkaConsumerFactory(config,new StringDeserializer(),jsonDeserializer);
    }
    @Bean
    ConcurrentKafkaListenerContainerFactory<String, Employee> kafkaListenerContainerFactory(){
        ConcurrentKafkaListenerContainerFactory<String, Employee> factory=new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }

}
```

Application Web : Spring Kafka Consumer

KafkaConsumer.java

```
Received:=> Employee(name=med, department=Finace, salary=45000.0)
Received:=> Employee(name=Hassan, department=Finace, salary=45000.0)
```

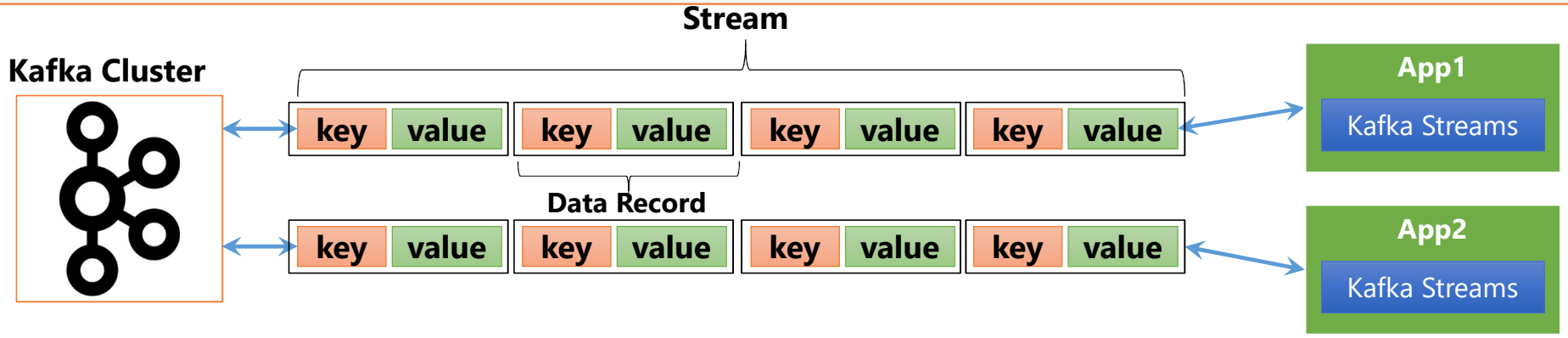
```java
package org.sid.kafkaconsumer.service;

import org.sid.kafkaconsumer.model.Employee;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
public class KafkaConsumer {

    @KafkaListener(topics = {"testTopic"},groupId = "enset_uh2c")
    public void onMessage(Employee employee){
        System.out.println("Received:=>"+employee.toString());

    }
}
```




```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testTopic
>{"name":"med","department":"info","salary":7000}
>
```

Kafka Stream API

- Un Stream est un flux continu d'enregistrements de données en temps réel.
- Le consommateur reçoie continuellement le flux de données sans avoir besoin de les demander.
- Les Streams est une série de données de type paire clé, valeur



- L'API Kafka Streams transforme et enrichie les données des streams
- Supporte le traitement des streams par enregistrements avec une latence faible de millisecondes
- Supporte les transformation statless, les transformations statfull, et les opérations fenêtrées
- Permet d'écrire des applications avec différents langages sous forme de micro-services pour traiter les données en temps réel.
- Vous n'avez pas besoin d'un cluster séparé pour les traitements.
- Les traitements se déroulent au sein de l'application elle-même.
- Kafka streams est scalable et tolèrent aux panes et interagit avec le cluster de Kafka pour récupérer et publier les données et les résultats pour d'autres applications

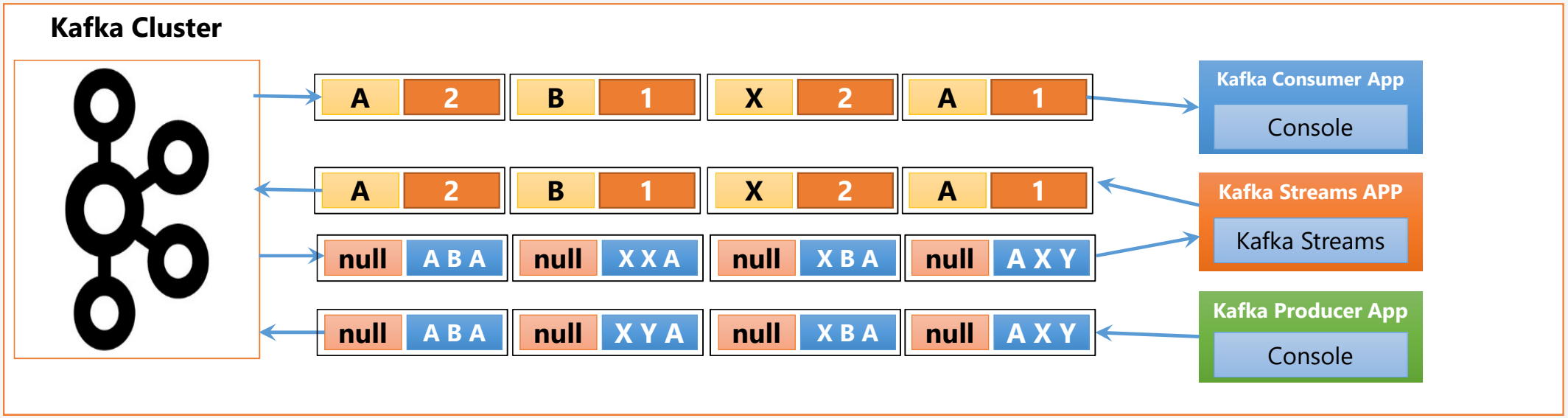
Kafka Stream API

- Let's write the topology using the High Level DSL for our application
- Remember data in Kafka Streams is **< Key , Value >**

1.	Stream from Kafka	< null, "Kafka Kafka Streams">
2.	MapValues lowercase	< null, "kafka kafka streams" >
3.	FlatMapValues split by space	< null, "kafka">, <null, "kafka">, <null, "streams">
4.	SelectKey to apply a key	< "kafka" , "kafka">, <"kafka", "kafka">, <"streams", "streams">
5.	GroupByKey before aggregation	(< "kafka" , "kafka">, <"kafka", "kafka">), (<"streams", "streams">)
6.	Count occurrences in each group	< "kafka" , 2 >, <"streams", 1>
7.	To in order to write the results back to Kafka	data point is written to Kafka

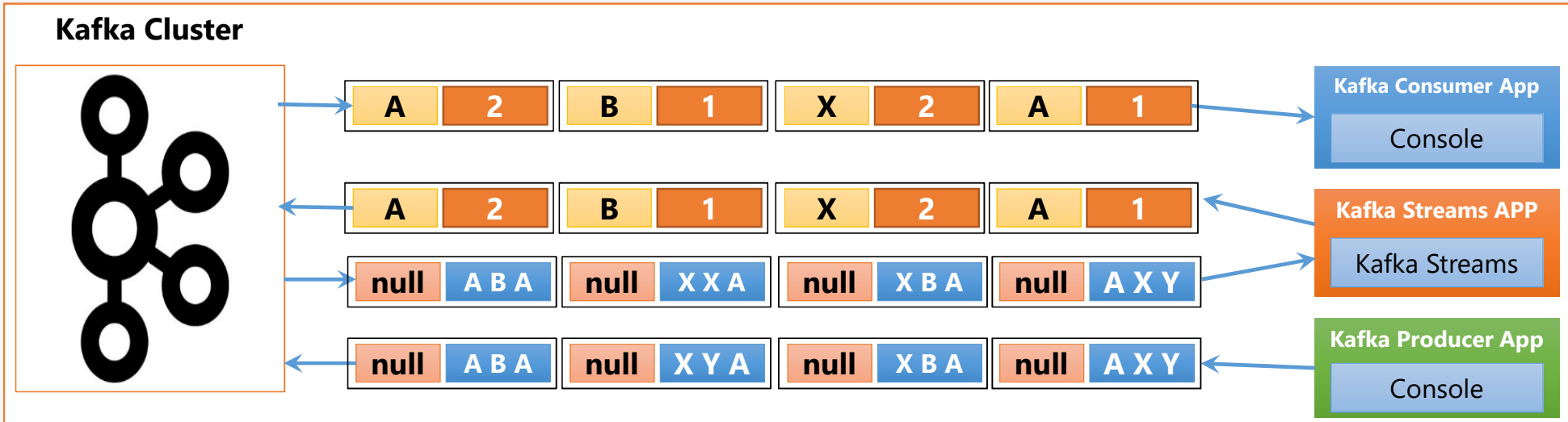
Kafka Stream API

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-application");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.57.3:9092");
props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
```



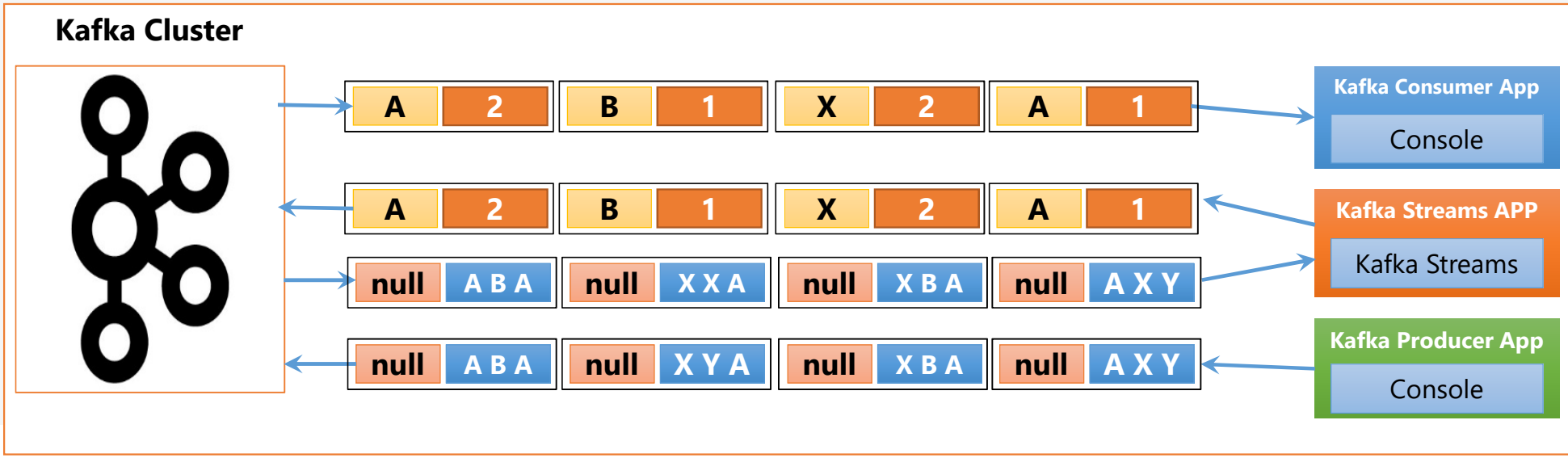
Kafka Stream API

```
StreamsBuilder builder = new StreamsBuilder();
// Serializers/deserializers (serde) for String and Long types
final Serde<String> stringSerde = Serdes.String();
final Serde<Long> longSerde = Serdes.Long();
// Construct a `KStream` from the input topic "streams-plaintext-input", where message values
// represent lines of text (for the sake of this example, we ignore whatever may be stored
// in the message keys).
KStream<String, String> textLines = builder.stream("streams-plaintext-input",
                                    Consumed.with(stringSerde, stringSerde));
```



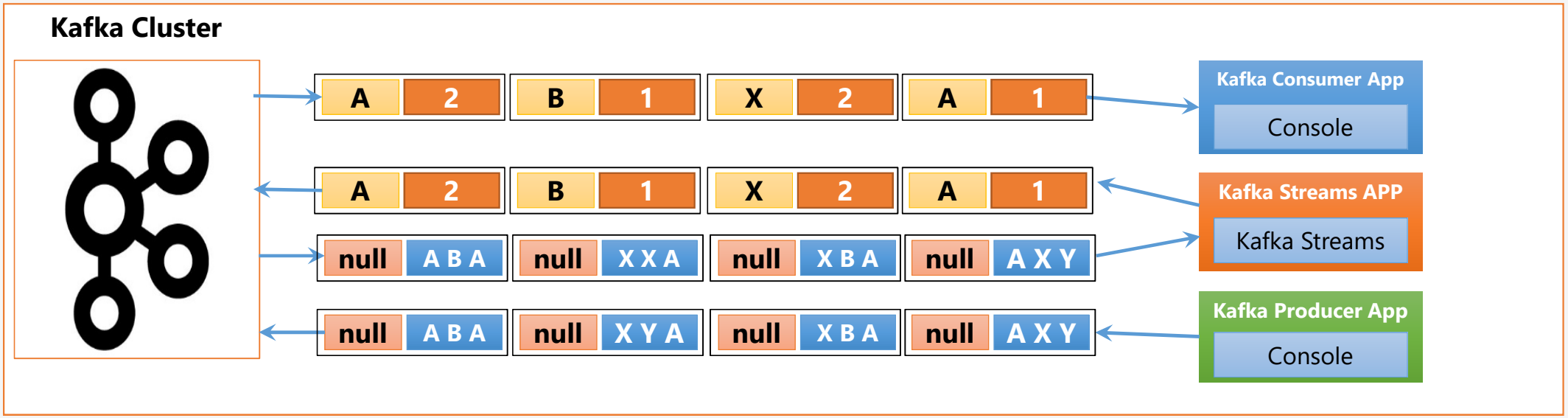
Kafka Stream API

```
KTable<String, Long> wordCounts = textLines
                // Split each text line, by whitespace, into words.
                .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
              // Group the text words as message keys
                .groupBy((key, value) -> value)
              // Count the occurrences of each word (message key).
                .count();
                 // Store the running counts as a changelog stream to the output topic.
wordCounts.toStream().to("streams-wordcount-output", Produced.with(Serdes.String(), Serdes.Long()));
```



Kafka Stream API

```
// Start Kafka Streams
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```



Kafka Stream API

```
<dependencies>
   <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka_2.12</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.3.0</version>
    </dependency>

</dependencies>
```

Kafka Stream Producer

```java
import org.apache.kafka.clients.producer.KafkaProducer; import org.apache.kafka.clients.producer.ProducerRecord;
import java.util.*; import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
public class StreamProducer {
    private int counter;
    private String KAFKA_BROKER_URL="192.168.43.22:9092";    private String TOPIC_NAME="testTopic";
    private String clientID="client_prod_1";    private String message;
    public static void main(String[] args) {
        new StreamProducer();
    }
    public StreamProducer() {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", KAFKA_BROKER_URL);
        properties.put("client.id", clientID);
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String,String> producer=new KafkaProducer<String, String>(properties);
        Random random=new Random();
        List<Character> characters= new ArrayList<>();
        for (char i = 'A'; i <'Z' ; i++) {   characters.add(i);   }
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
            ++counter;  message="";  for (int i = 0; i < 10; i++) {
                message+=" "+characters.get(random.nextInt(characters.size()));
            }
            producer.send(new ProducerRecord<String, String>(TOPIC_NAME,""+(++counter),message),
                    (metadata,ex)->{
                        System.out.println("Sending Message key=>"+counter+" Value =>"+message);
                        System.out.println("Partition => "+metadata.partition()+" Offset=>"+metadata.offset());
                    });
        },1000,1000, TimeUnit.MILLISECONDS);
    }
}
```

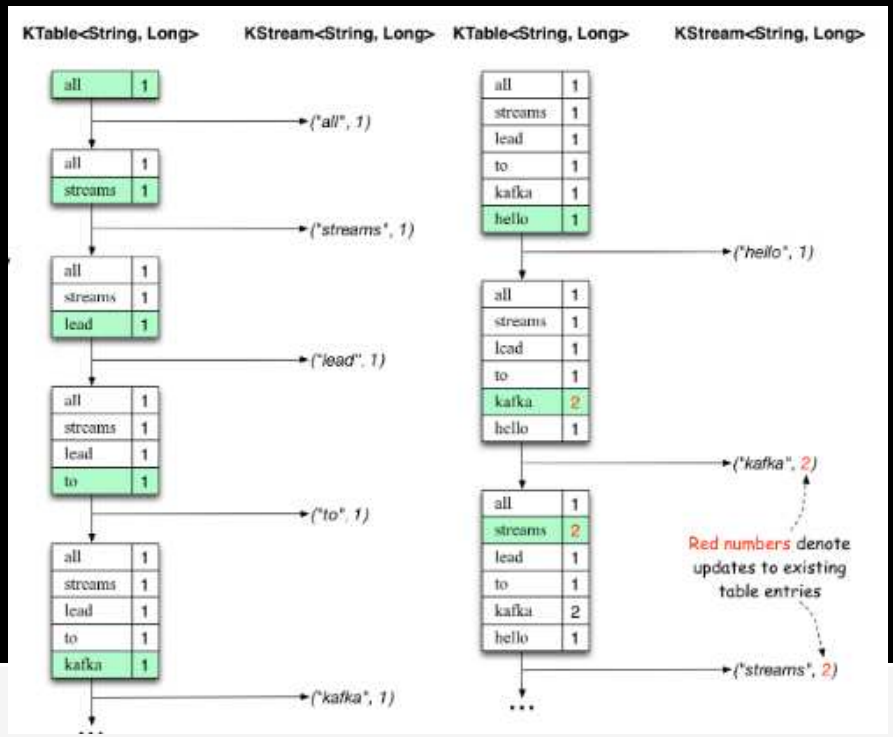
```
youssfi@youssfi-VirtualBox:~/kafka_2.12-2.3.0$ bin/kafka-console-consumer.sh --
bootstrap-server localhost:9092 --topic testTopic
 P J Q I M W A M C K
 J T Y Y P O G D J L
 E K P V V B F R O E
 C R U O I D W K S C
 B T B P L I X S D B
```

Kafka Stream Application

```
import org.apache.kafka.common.serialization.Serdes;import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Materialized; import org.apache.kafka.streams.kstream.Produced;
import java.util.Arrays;import java.util.Properties;
public class KafkaStreamsApp {
    public static void main(String[] args) {
        Properties properties=new Properties();
        properties.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.43.22:9092");
        properties.put(StreamsConfig.APPLICATION_ID_CONFIG,"k-stream-app");
        properties.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
        properties.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
        properties.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
        StreamsBuilder streamsBuilder=new StreamsBuilder();
        KStream<String,String> textLines=streamsBuilder.stream("testTopic");
        KTable<String,Long> wordCounts=textLines
                .flatMapValues(textLine-> Arrays.asList(textLine.split("\\W+")))
                .groupBy((k,word)->word)
                .count(Materialized.as("count-store"));
        wordCounts.toStream().to("wcTopic", Produced.with(Serdes.String(),Serdes.Long()));
        Topology topology=streamsBuilder.build();
        KafkaStreams kafkaStreams=new KafkaStreams(topology,properties);
        kafkaStreams.start();

    }
}
```

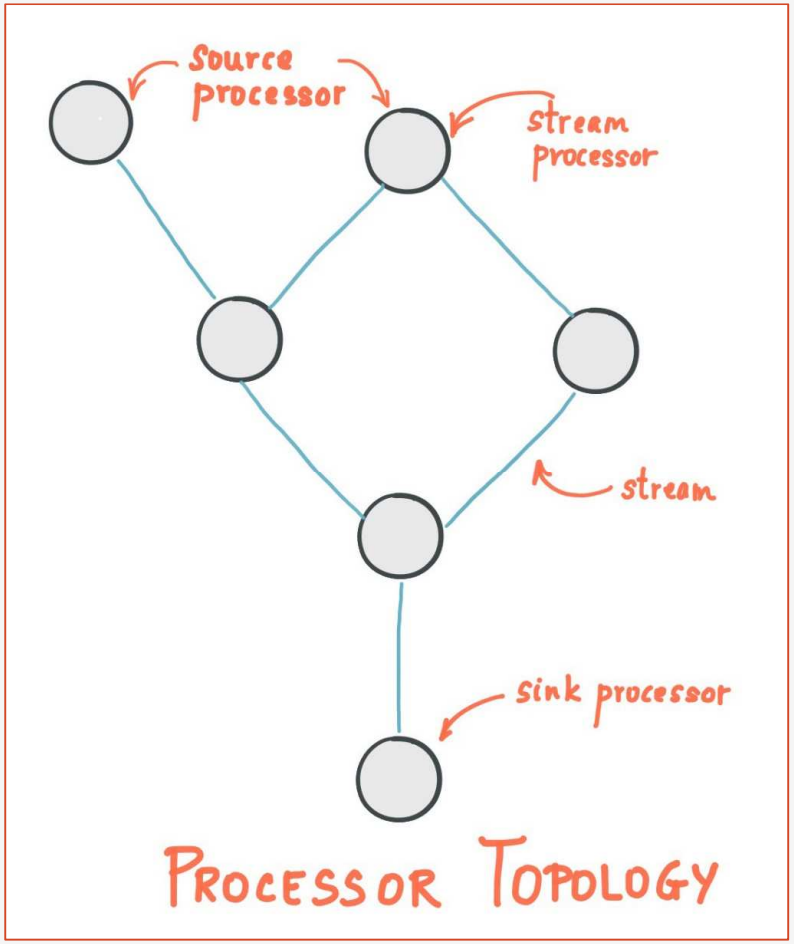
Kafka Stream API (Stream Processing Consumer)

```
youssfi@youssfi-VirtualBox:~/kafka_2.12-2.3.0$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --
topic wcTopic --property print.key=true --property print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property
value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
         827
N        347
Y        328
S        311
W        330
H        365
G        338
V        322
         828
O        309
M        325
C        331
F        274
S        312
U        322
D        333
G        339
```



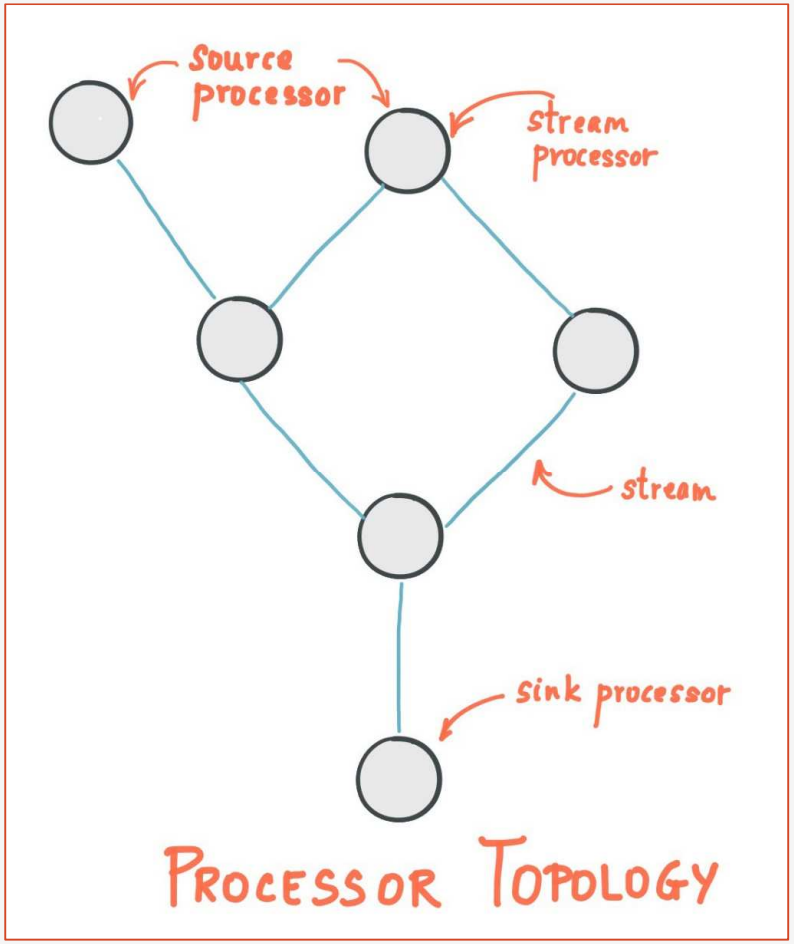
Concepts de base de Kafka : Stream Processing Topology

- Un **Stream** représente un ensemble de données illimité et constamment mis à jour. C'est une séquence d'enregistrements de données immuables, dans laquelle un enregistrement de données est défini comme une paire clé-valeur.
- **Une application de traitement de flux** est un programme qui utilise la bibliothèque Kafka Streams. Il définit sa logique de calcul par le biais d'une ou de plusieurs **topologies de processeurs (Processor Topologies)**.
- **Une topologie de processeurs** étant un graphe de processeurs de flux (**Stream Processors**) (nœuds) connectés par des Steram (arêtes).
- **Un processeur de flux (Stream Processor)** est un nœud de la topologie du processeurs; il représente une étape de traitement pour transformer les données des Streams en recevant un enregistrement en d'entrée à un instant donné à partir de ses processeurs en amont dans la topologie, en lui appliquant son **opérateur (map, ftatMap, groupBy, count, etc..)**, et peut ensuite produire un ou plusieurs enregistrements en sortie à ses processeurs en aval.



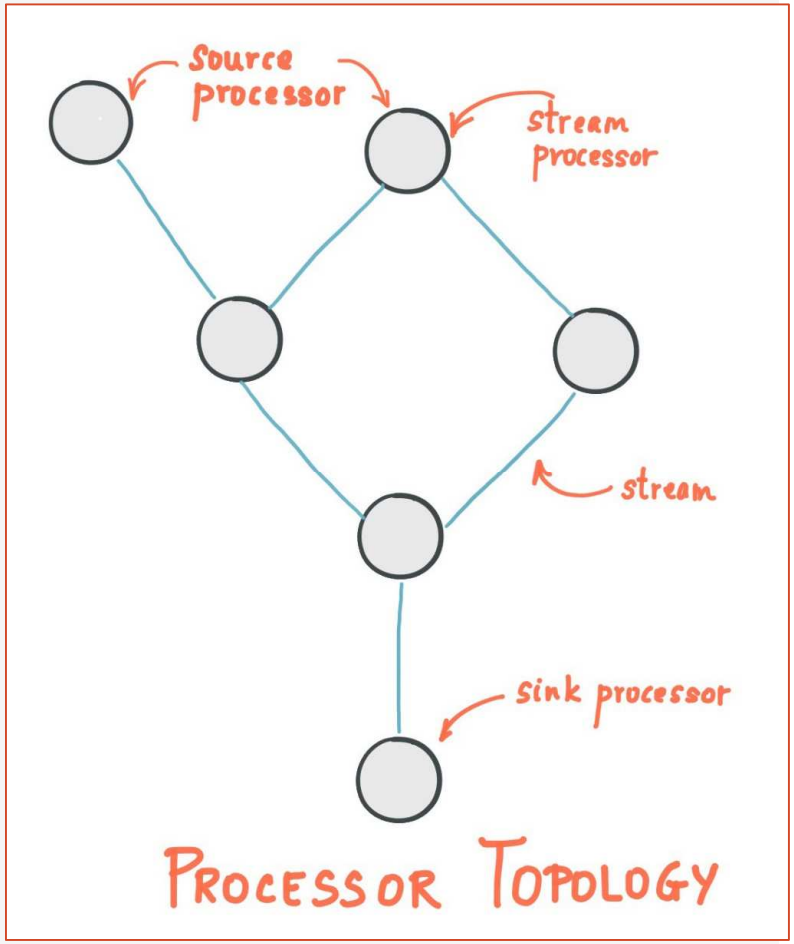
Concepts de base de Kafka : Stream Processing Topology

- Kafka Streams propose deux manières de définir la topologie de traitement de flux:
  - **Kafka Streams DSL** : fournit les opérations de transformation de données les plus courantes, telles que mappage, filtrage, jointure et agrégations prêtes à l'emploi;
  - **Lower-level Processor API** : permet aux développeurs de définir et de connecter des processeurs personnalisés, ainsi que d'interagir avec les state stores.



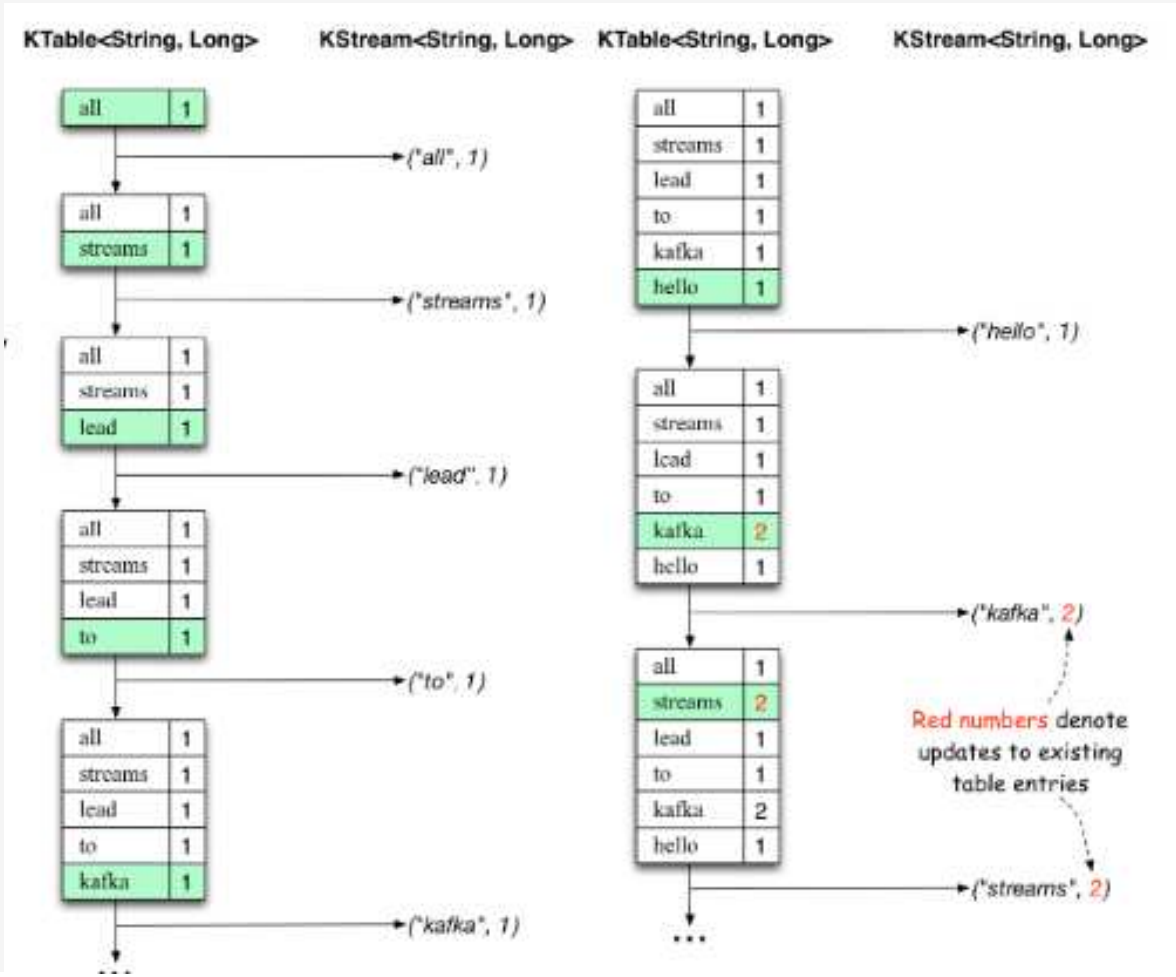
Concepts de base de Kafka : Time

- Un aspect essentiel du traitement de flux est la notion de temps, ainsi que sa modélisation et son intégration. Par exemple, certaines opérations telles que le fenêtrage (**windowing**) sont définies en fonction des intervalles d e temps.
- Les notions courantes de temps dans les flux sont:
  - **Event Time** : (Heure de l'événement) - Heure à laquelle un événement ou un enregistrement de données s'est produit, c'est-à-dire qu'il a été créé à l'origine "à la source« (Temps de capture de l'événement par un capteur GPS par exemple).
  - **Processing Time** (Durée de traitement) - Peut être de l'ordre de la milliseconde ou de la seconde (pour les pipelines Real Time basés sur Apache Kafka et Kafka Streams) ou de minutes et d'heures (Les pipelines par Batch Processing, basés sur Apache Hadoop ou Apache Spark).
  - **Ingestion Time** (Heure d'ingestion) - Heure à laquelle un événement ou un enregistrement de données est stocké dans une partition de Topic par un Broker Kafka.
- Le choix entre **Event Time** et **Ingestion Time** se fait en réalité via la configuration de Kafka (et non de Kafka Streams)
- Kafka Streams attribue un horodatage (timestamp) à chaque enregistrement de données via l'interface TimestampExtractor. Ces horodatages par enregistrement décrivent la progression d'un flux en termes de temps et sont exploités par des opérations dépendantes du temps telles que les opérations de windowing



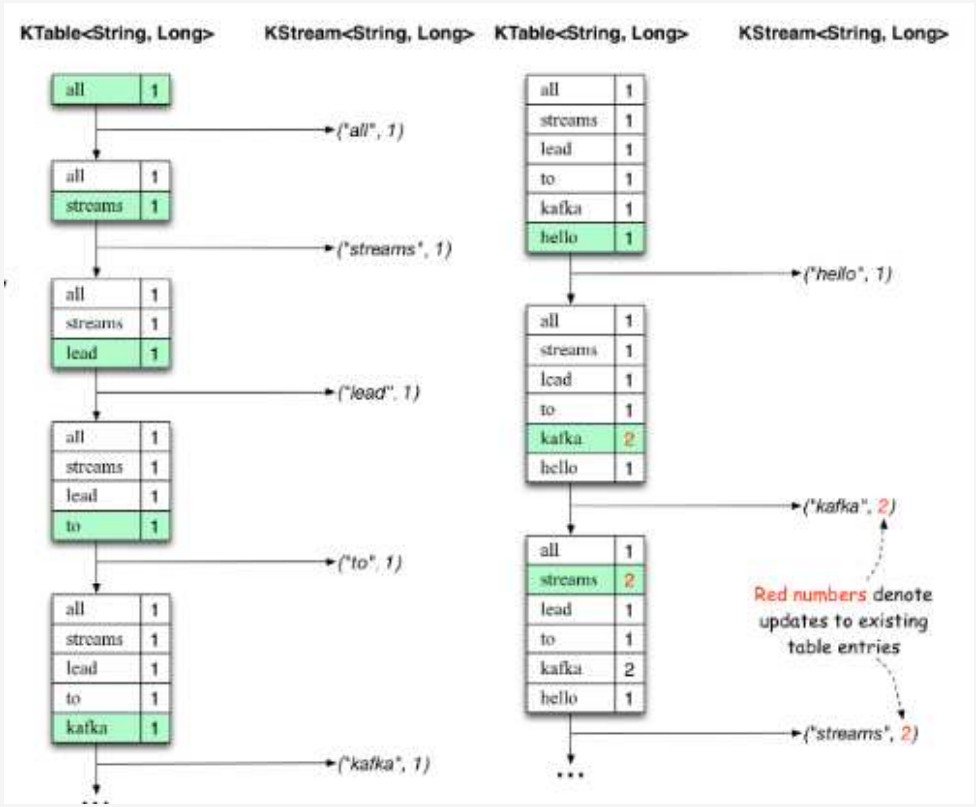
Concepts de base de Kafka : Agregations

- Une opération d'agrégation prend un flux ou une table d'entrée et génère une nouvelle table en combinant plusieurs enregistrements d'entrée en un seul enregistrement de sortie. Des exemples d'agrégations sont le calcul du nombre (**count**) ou de la somme (**sum**).
- Dans le DSL de Kafka Streams, un flux d'entrée d'une agrégation peut être un **KStream** ou un **KTable**, mais le flux de sortie sera toujours un **KTable**.
- Cela permet à Kafka Streams de mettre à jour une valeur globale lors de l'arrivée tardive d'autres enregistrements après que la valeur a été produite et émise.



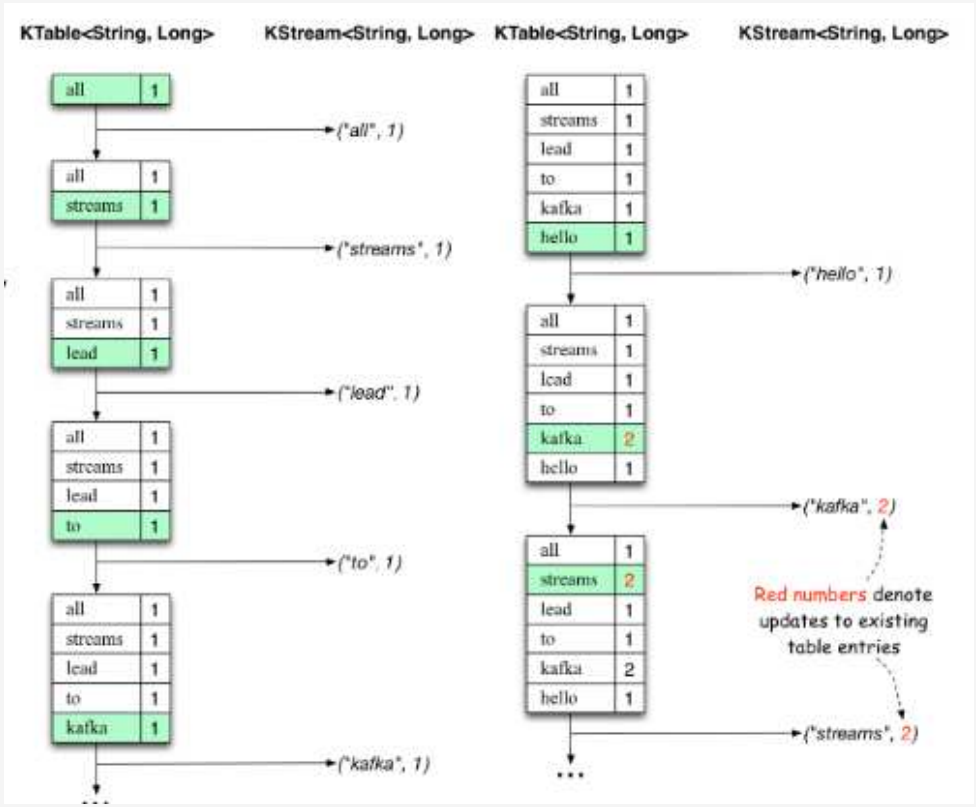
Concepts de base de Kafka : Windowing

- Le fenêtrage vous permet de contrôler comment regrouper les enregistrements ayant la même clé pour des opérations avec état telles que des agrégations ou des jointures dans des fenêtres.
- Avec les ces opérations, vous pouvez spécifier une période de rétention pour la fenêtre.
- Cette période de rétention détermine la durée pendant laquelle Kafka Stream attendra des enregistrements de données tardifs pour une fenêtre donnée.



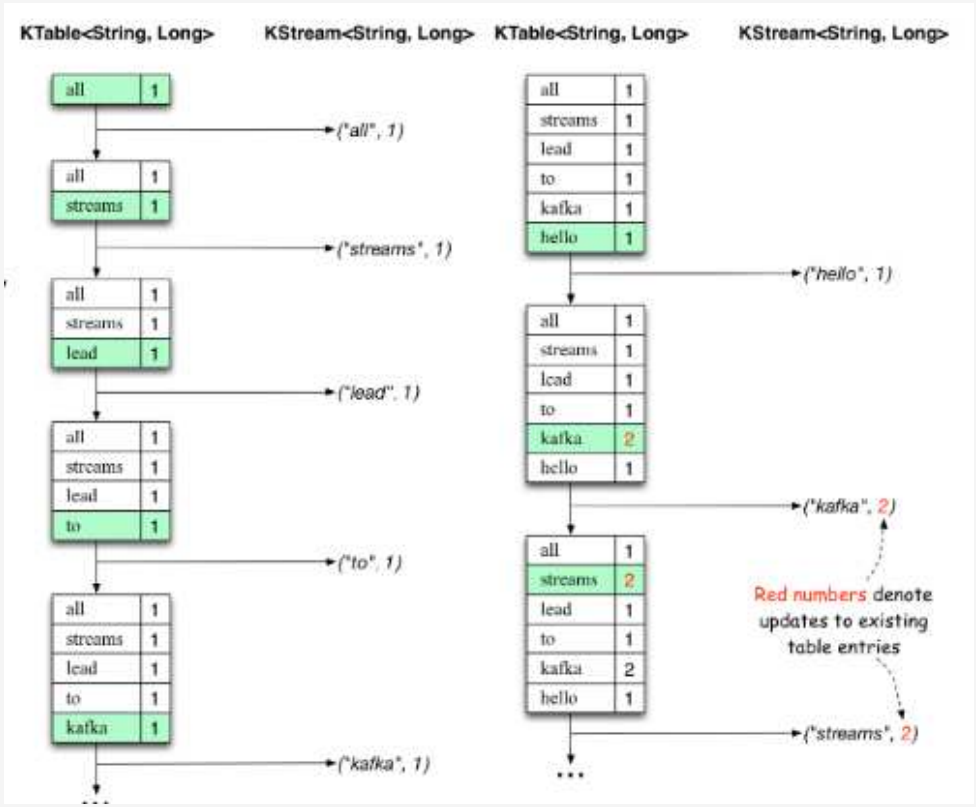
Concepts de base de Kafka : Duality of Streams and Tables

- Lorsque vous implémentez des cas d'utilisation de traitement de flux, vous avez généralement besoin de flux (Streams) et de bases de données.
- L'API Streams de Kafka fournit un support pour les flux et les tables.
- de telles fonctionnalités par le biais de ses abstractions principales pour les flux et les tables.
- il existe en fait une relation étroite entre les flux et les tables, la dite dualité **Stream-Table Duality**.
- Kafka exploite cette dualité de nombreuses pour rendre vos applications :
  - Élastiques,
  - Prendre en charge le traitement avec état tolérant aux pannes
  - Exécuter des requêtes interactives sur les derniers résultats de traitement de votre application.
- Les développeurs peuvent aussi exploiter cette dualité pour leurs propres applications.



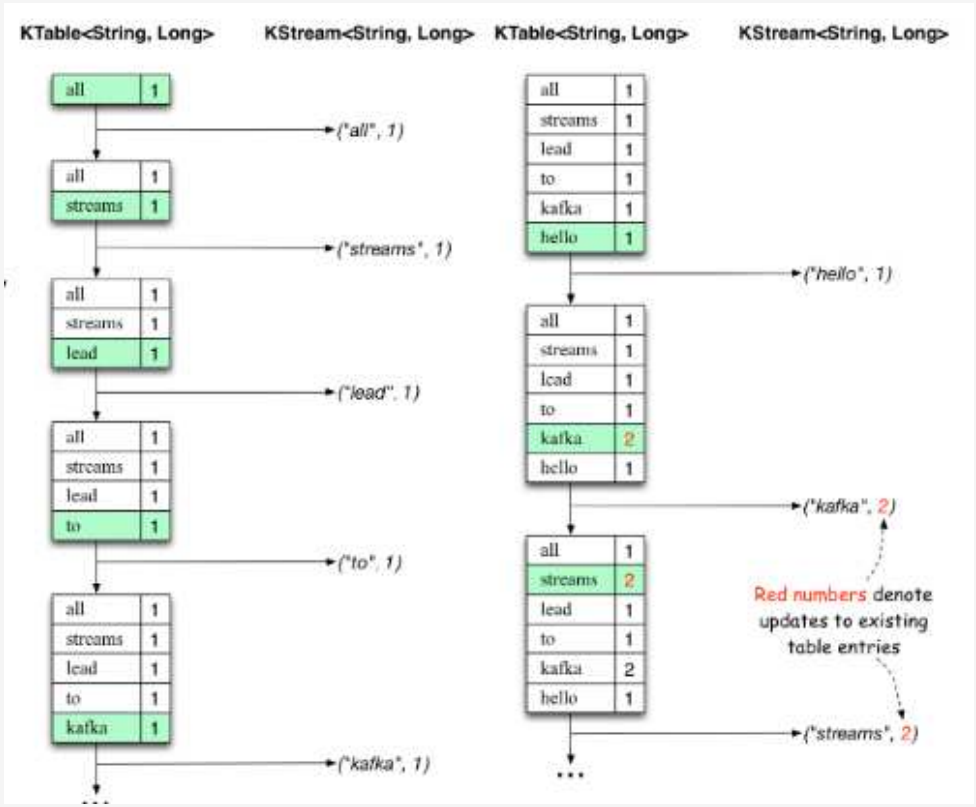
Concepts de base de Kafka : States

- Certaines applications de traitement de flux ne nécessitent pas d'état (**Statless Operations**), ce qui signifie que le traitement d'un message est indépendant du traitement de tous les autres messages.
- Cependant, le maintien de l'état (**Statful Operations**) ouvre de nombreuses possibilités pour les applications de traitement de flux sophistiquées: vous pouvez joindre des flux d'entrée (**Join**) ou regrouper (**GroupBy**) des enregistrements de données. Le DSL de Kafka Streams fournit nombre de ces opérateurs avec état (**Statful Operations**).
- Kafka Streams fournit des magasins d'état (**Data Stores**), que les applications de traitement de flux peuvent utiliser pour stocker et interroger des données.
- C'est une capacité importante lors de la mise en œuvre d'opérations avec état. Chaque tâche de Kafka Streams intègre un ou plusieurs magasins d'Etats accessibles via des API pour stocker et interroger les données nécessaires au traitement.

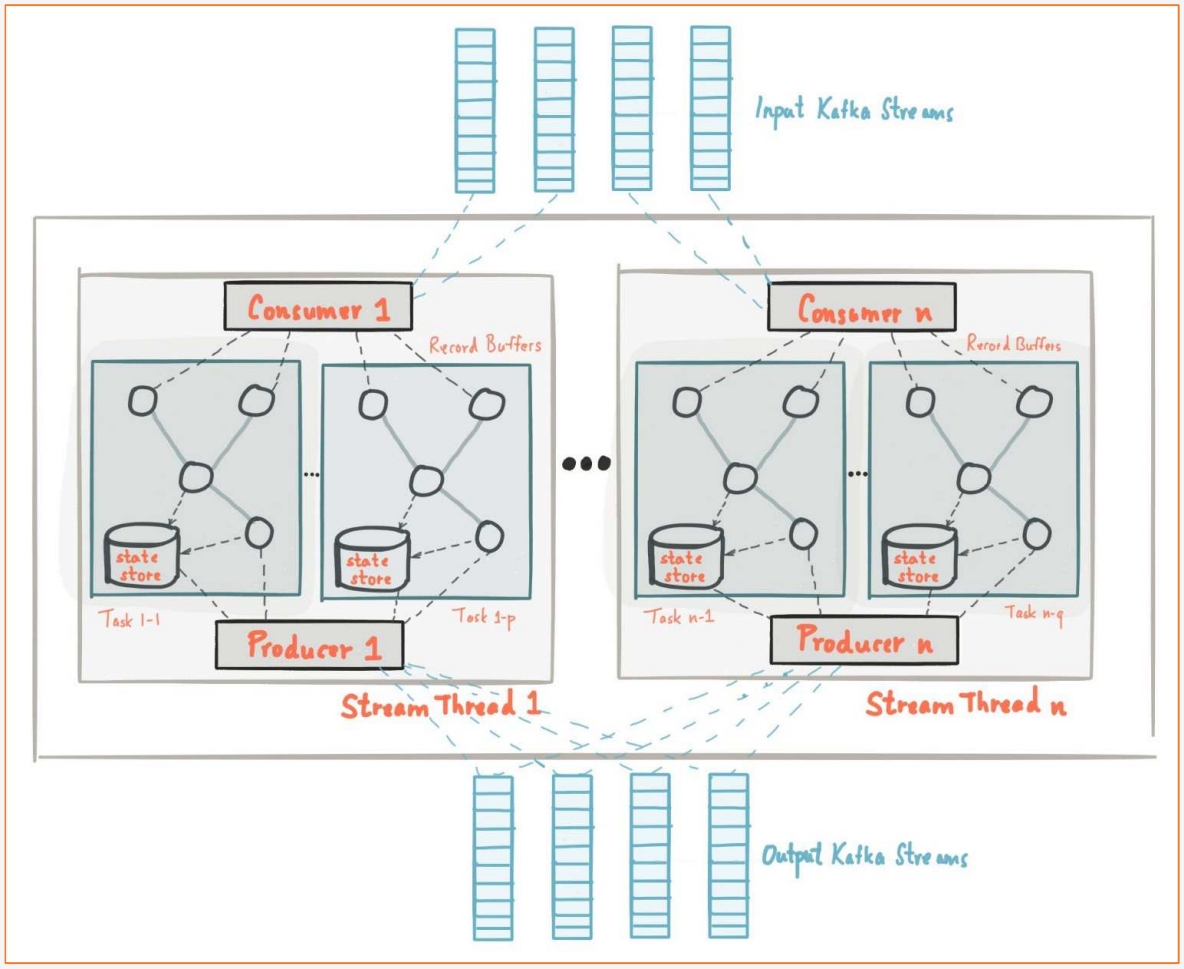


Concepts de base de Kafka : States

- Ces magasins d'état peuvent être **un magasin de clé-valeur persistant**, **un hashmap en mémoire** ou **une autre structure de données pratique**.
- Kafka Streams offre une tolérance aux pannes et une récupération automatique pour les magasins d'état locaux.
- Kafka Streams permet des requêtes directes en lecture seule sur les magasins d'état à l'aide de méthodes, de threads, de processus ou d'applications externes à l'application de traitement de flux ayant créé les magasins d'état.
- Ceci est fourni par une fonctionnalité appelée **Interactive Queries**.
- Tous les magasins sont nommés et Interactive Queries n'expose que les opérations de lecture de l'implémentation sous-jacente.



Concepts de base de Kafka : Architecture

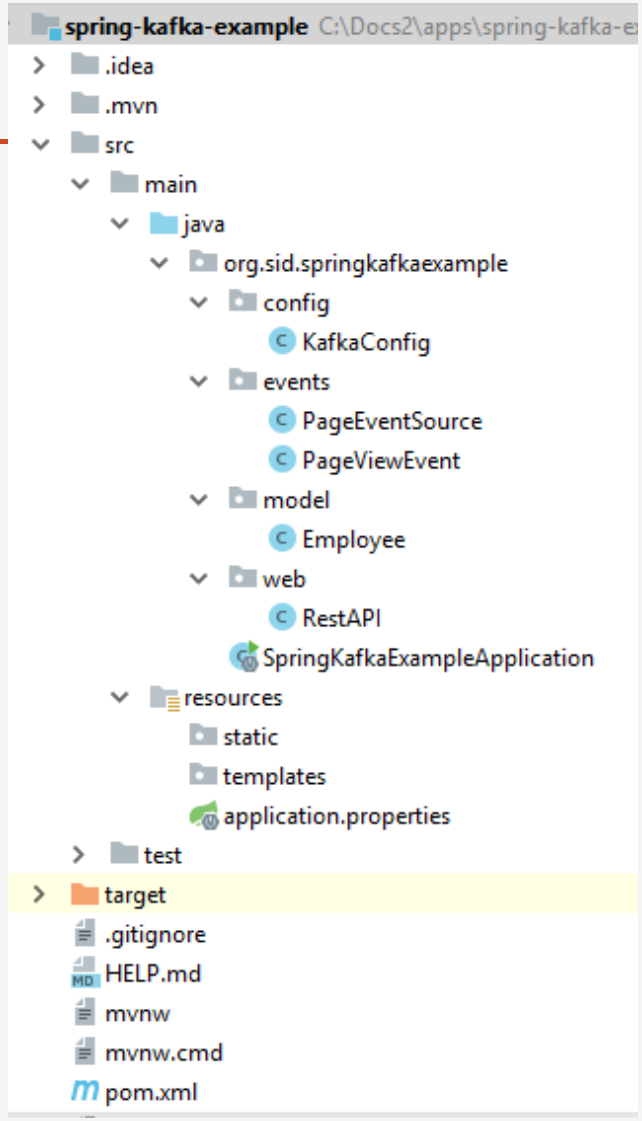


Kafka-Stream et Spring Boot : Kafka Producer App

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>


    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

application.properties

```
server.port=8083
```



Kafka-Stream et Spring Boot : Kafka Producer App

```java
package org.sid.springkafkaexample.events;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class PageViewEvent {
    private String userId;
    private String page;
    private int duration;
}
```

```java
package org.sid.springkafkaexample.web;
import org.sid.springkafkaexample.events.PageViewEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("kafka")
public class RestAPI {
     @Autowired
    private KafkaTemplate<String, PageViewEvent> kafkaTemplate;
    private String topic="pvts";
    @GetMapping("/publish/{user}-{page}-{duration}")
    public PageViewEvent publishMessage(
            @PathVariable String user,
            @PathVariable String page,
            @PathVariable int duration){
        PageViewEvent pageViewEvent=new PageViewEvent(user,page,duration);
        kafkaTemplate.send(topic,pageViewEvent.getUserId(),pageViewEvent);
        return pageViewEvent;
    }
}
```

Kafka-Stream et Spring Boot : Kafka Producer App

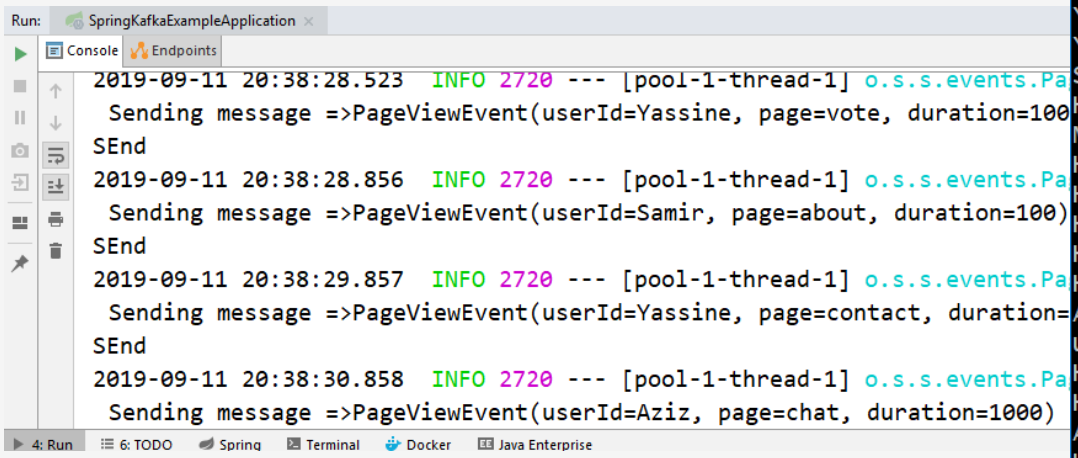
```
package org.sid.springkafkaexample.events;

import lombok.extern.slf4j.Slf4j; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments; import org.springframework.boot.ApplicationRunner;
import org.springframework.kafka.core.KafkaTemplate; import org.springframework.stereotype.Component;
import java.util.Arrays; import java.util.List; import java.util.Random;
import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
@Component
@Slf4j
public  class PageEventSource implements ApplicationRunner {
    @Autowired
    private KafkaTemplate<String,PageViewEvent>  kafkaTemplate;
    @Override
    public void run(ApplicationArguments applicationArguments)  {
        System.out.println("..................................................;");
        List<String> names= Arrays.asList("Hassan","Mohamed","Hanane","Yassine","Samir","Aziz");
        List<String> pages= Arrays.asList("blog","chat","profile","about","contact","vote","search");
        Runnable runnable=()->{
            String rPage=pages.get(new Random().nextInt(pages.size()));
            String rName=names.get(new Random().nextInt(names.size()));
            PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,Math.random()>0.5?100:1000);
// PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,100+(int)(Math.random()*1000));
            System.out.println("SEnd");
            kafkaTemplate.send("pvts",pageViewEvent.getUserId(),pageViewEvent);
            log.info("Sending message =>"+pageViewEvent.toString());
        };
        System.out.println("-----------------------------------------");
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,1,1, TimeUnit.SECONDS);
//Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,100,100, TimeUnit.MILLISECONDS);
    }
}
```

Kafka-Stream et Spring Boot : Kafka Producer App

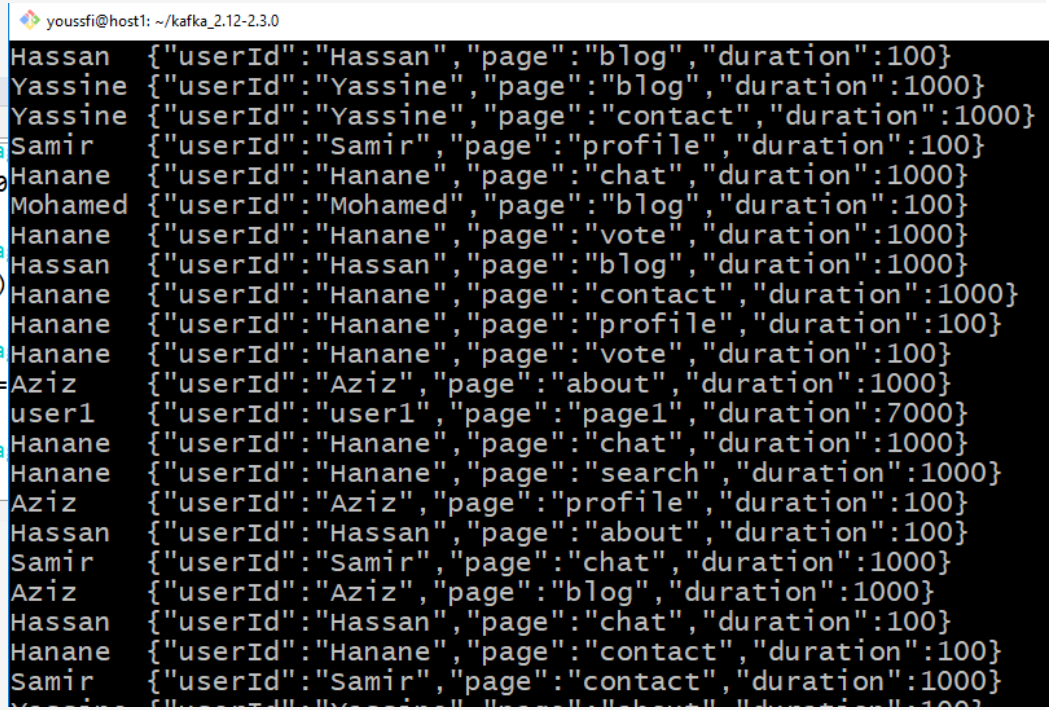
```
package org.sid.springkafkaexample.config;

import org.apache.kafka.clients.producer.ProducerConfig; import org.apache.kafka.common.serialization.StringSerializer;
import org.sid.springkafkaexample.events.PageViewEvent; import org.sid.springkafkaexample.model.Employee;
import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory; import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory; import org.springframework.kafka.support.serializer.JsonSerializer;
import java.util.HashMap; import java.util.Map;

@Configuration
public class KafkaConfig {
    @Bean
    ProducerFactory<String, PageViewEvent> producerFactory(){
        Map<String,Object> config=new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
       return new DefaultKafkaProducerFactory<>(config);
    }
    @Bean
    KafkaTemplate<String, PageViewEvent> kafkaTemplate(){
        return new KafkaTemplate<>(producerFactory());
    }
}
```

Kafka-Stream et Spring Boot : Kafka Producer App

Producer App

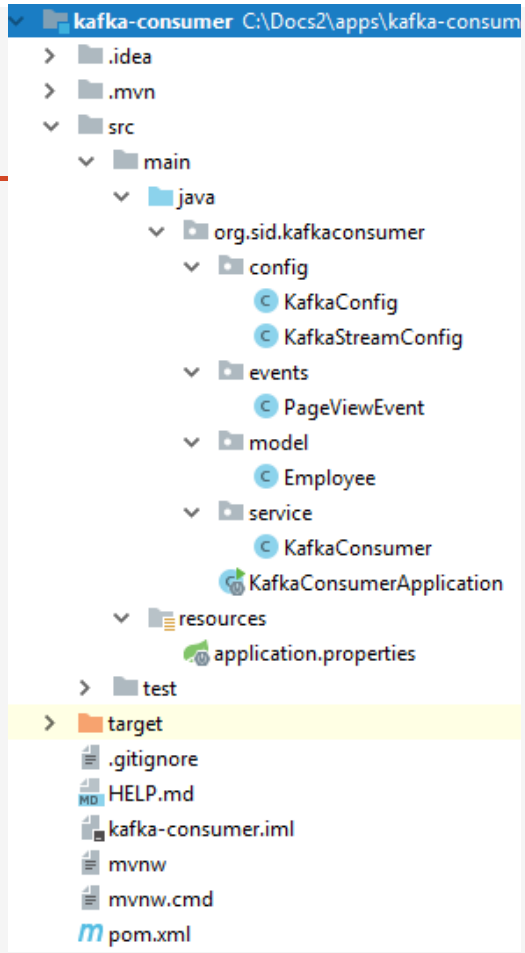


Kafka Consumer Console



Kafka-Stream et Spring Boot : Kafka Stream Appli

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```



application.properties

```
#spring.kafka.consumer.bootstrap-servers=192.168.57.3:9092
#spring.kafka.consumer.group-id=enset_uh2c
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
package org.sid.kafkaconsumer.config;
import com.fasterxml.jackson.databind.ObjectMapper; import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KeyValue; import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.*; import org.sid.kafkaconsumer.events.PageViewEvent;
import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka; import org.springframework.kafka.annotation.EnableKafkaStreams;
import org.springframework.kafka.annotation.KafkaStreamsDefaultConfiguration;import org.springframework.kafka.config.KafkaStreamsConfiguration;
import org.springframework.kafka.config.StreamsBuilderFactoryBean; import java.io.IOException; import java.util.HashMap; import java.util.Map;
@Configuration
@EnableKafka
@EnableKafkaStreams
public class KafkaStreamConfig {
    ObjectMapper objectMapper=new ObjectMapper();
    @Bean(name = KafkaStreamsDefaultConfiguration.DEFAULT_STREAMS_CONFIG_BEAN_NAME)
    public KafkaStreamsConfiguration kafkaStreamsConfiguration(){
        Map<String,Object> conf=new HashMap<>();
       conf.put(StreamsConfig.APPLICATION_ID_CONFIG, "Page-Event-Stream");
       conf.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.57.3:9092");
       conf.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       conf.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       conf.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
        return new KafkaStreamsConfiguration(conf);
    }
 @Bean
    public StreamsBuilderFactoryBean factoryBean(){
       return new StreamsBuilderFactoryBean(kafkaStreamsConfiguration());
    }
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
    @Bean
    public KStream<String, Integer> process() throws Exception{
        KStream<String, String> stream = factoryBean().getObject().stream("pvts");
        KStream<String, Integer> counts = stream
                .map((k, v) -> new KeyValue<>(k, pageViewEvent(v)))
                .filter((k, v) -> v.getDuration() > 200)
                .map((k,v)->new KeyValue<>(k,v.getDuration()));
                //.groupByKey()
                //.windowedBy(TimeWindows.of(4000))
                //.count(Materialized.as("Page_Count")).toStream();
        counts.to("pvs-out-5",Produced.valueSerde(Serdes.Integer()));
        //stream.map((k,v)->new KeyValue<>(v,k)).to("xxx");
        return counts;
    }
    public PageViewEvent pageViewEvent(String strObject){
        PageViewEvent pageViewEvent = new PageViewEvent();
        try {
            pageViewEvent = objectMapper.readValue(strObject, PageViewEvent.class);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return pageViewEvent;
    }

}
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs-out-5 --
property print.key=true --property print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property
value.deserializer=org.apache.kafka.common.serialization.IntegerDeserializer
Mohamed 1000
Hassan  1000
Yassine 1000
Mohamed 1000
Mohamed 1000
Yassine 1000
Aziz    1000
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
@Configuration
@EnableKafka
@EnableKafkaStreams
public class KafkaStreamConfig {
    ObjectMapper objectMapper=new ObjectMapper();
    @Bean(name = KafkaStreamsDefaultConfiguration.DEFAULT_STREAMS_CONFIG_BEAN_NAME)
    public KafkaStreamsConfiguration kafkaStreamsConfiguration(){
        Map<String,Object> conf=new HashMap<>();
       conf.put(StreamsConfig.APPLICATION_ID_CONFIG, "Page-Event-Stream");
       conf.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.57.3:9092");
       conf.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       conf.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
       conf.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
        conf.put(StreamsConfig.POLL_MS_CONFIG, 1000);
        return new KafkaStreamsConfiguration(conf);
    }
   @Bean
    public StreamsBuilderFactoryBean factoryBean(){
       return new StreamsBuilderFactoryBean(kafkaStreamsConfiguration());
    }
```

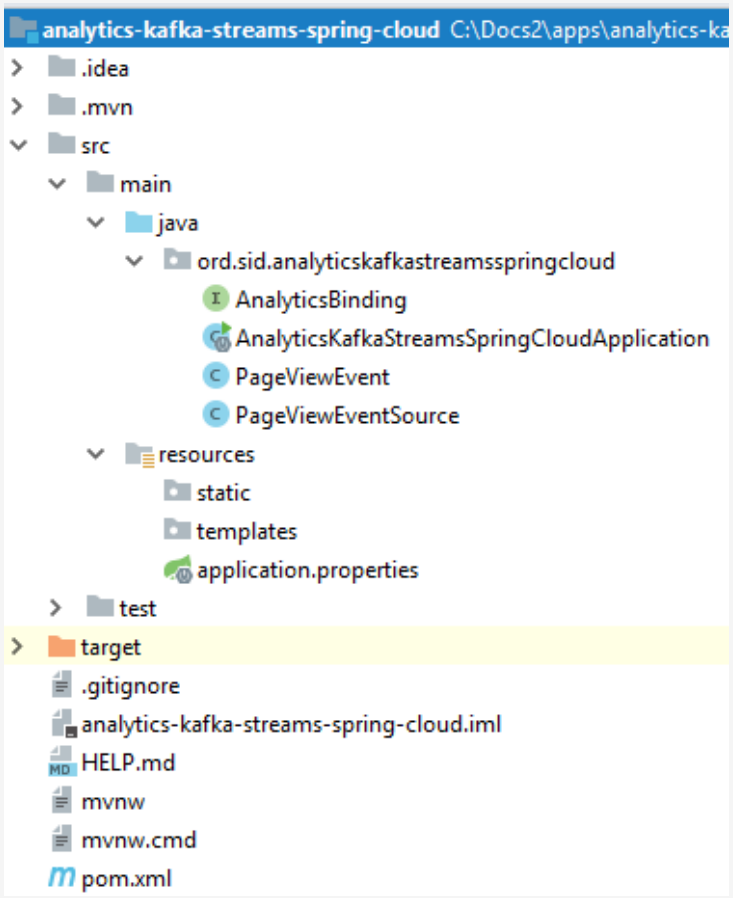
Kafka-Stream et Spring Boot : Kafka Stream Appli

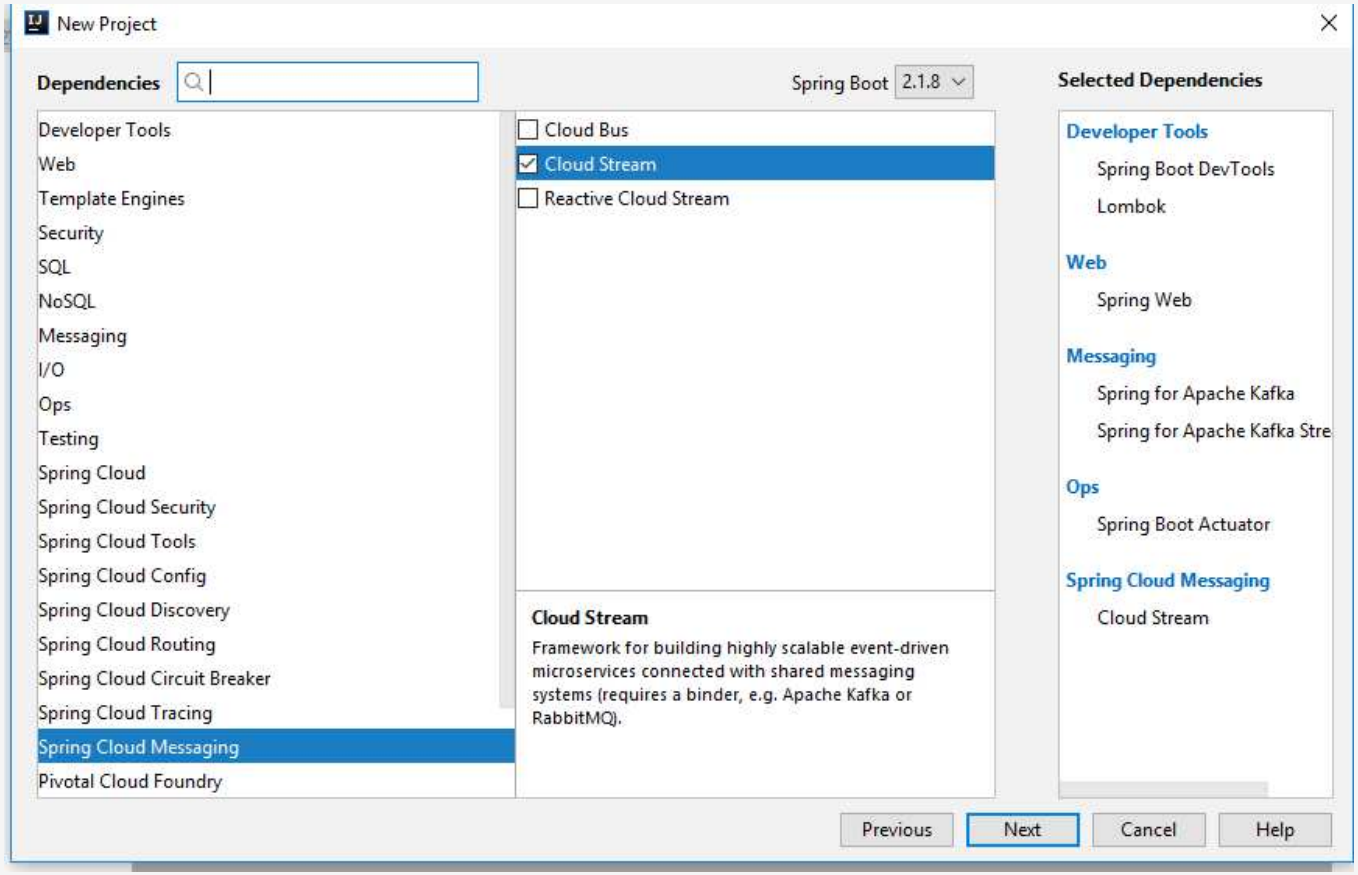
```
@Bean
    public KStream<String, Long> process() throws Exception{
        KStream<String, String> stream = factoryBean().getObject().stream("pvts");
        KStream<String, Long> counts = stream
                .map((k, v) -> new KeyValue<>(k, pageViewEvent(v)))
                .filter((k, v) -> v.getDuration() > 80)
                .map((k,v)->new KeyValue<>(v.getPage(),"0"))
                .groupByKey()
                .windowedBy(TimeWindows.of(5000))
                .count(Materialized.as("Page_Count")).toStream()
                .map((k,v)->new KeyValue<>(k.key(),v));
        counts.to("pvs-out-6",Produced.valueSerde(Serdes.Long()));
        return counts;
    }
    public PageViewEvent pageViewEvent(String strObject){
        PageViewEvent pageViewEvent = new PageViewEvent();
        try {
             pageViewEvent = objectMapper.readValue(strObject, PageViewEvent.class);
             System.out.println(pageViewEvent.getPage());
           } catch (IOException e) {
            e.printStackTrace();
        }
        return pageViewEvent;
    }
}
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs-out-6 --
property print.key=true --property print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property
value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
search  8
blog    5
chat    4
vote    5
contact 7
profile 6

contact 9
vote    6
blog    6
search  10
chat    7
about   1

blog    1
chat    1
...
```

Spring Cloud avec Kafka





Spring Cloud avec Kafka

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream-binder-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream-binder-kafka-
streams</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream-test-support</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Spring Cloud avec Kafka

PageViewEvent.java

```
package ord.sid.analyticskafkastreamsspringcloud;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PageViewEvent{
    private String userId,page;
    private long duration;
}
```

Spring Cloud avec Kafka

AnalyticsBindings.java

```
package ord.sid.analyticskafkastreamsspringcloud;

import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;

public interface AnalyticsBinding{
    String PAGE_VIEWS_OUT="pvout";
    @Output(PAGE_VIEWS_OUT)
    MessageChannel pageViewsOut();
}
```

Spring Cloud avec Kafka

PageViewEventSource.java

```
package ord.sid.analyticskafkastreamsspringcloud;
import lombok.extern.slf4j.Slf4j; import org.springframework.boot.ApplicationArguments; import org.springframework.boot.ApplicationRunner;
import org.springframework.kafka.support.KafkaHeaders; import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel; import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Component; import java.util.Arrays; import java.util.List; import java.util.Random;
import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
@Component
@Slf4j
public class PageViewEventSource implements ApplicationRunner {
    private MessageChannel pageViewsOutChannel;
    @SuppressWarnings("SpringJavaInjectionPointsAutowiringInspection")
    public PageViewEventSource(AnalyticsBinding analyticsBinding) {
        this.pageViewsOutChannel = analyticsBinding.pageViewsOut();
    }
```

Spring Cloud avec Kafka

PageViewEventSource.java

```
@Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("..................................................;");
        List<String> names= Arrays.asList("Hassan","Mohamed","Hanane","Yassine","Samir","Aziz");
        List<String> pages= Arrays.asList("blog","chat","profile","about","contact","vote","search");
        Runnable runnable=()->{
            String rPage=pages.get(new Random().nextInt(pages.size()));
            String rName=names.get(new Random().nextInt(names.size()));
            PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,100+(int)(Math.random()*1000));
            Message<PageViewEvent> eventMessage = MessageBuilder
                    .withPayload(pageViewEvent)
                    .setHeader(KafkaHeaders.MESSAGE_KEY, pageViewEvent.getUserId().getBytes())
                    .build();
            try {
                this.pageViewsOutChannel.send(eventMessage); log.info("Sending"+eventMessage.toString());
            } catch (Exception e) {              e.printStackTrace();
            }
            log.info("Sending message =>"+pageViewEvent.toString());
        };
        System.out.println("-------------------------------------------");
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,1000,1000, TimeUnit.MILLISECONDS);
    }
}
```

Spring Cloud avec Kafka

application.properties

```
#defaults
spring.cloud.stream.kafka.streams.binder.configuration.commit.interval.mms=1000
spring.cloud.stream.kafka.streams.binder.configuration.default.key.serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.binder.configuration.default.value.serde=org.apache.kafka.common.serialization.Serdes$StringSerde

# Page view out Bindings

spring.cloud.stream.bindings.pvout.destination=pvs
spring.cloud.stream.bindings.pvout.producer.header-mode=raw
# Kafka Brokers hosts
spring.kafka.bootstrap-servers=192.168.57.3:9092
```

Spring Cloud avec Kafka

```
package ord.sid.analyticskafkastreamsspringcloud;
import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;
@SpringBootApplication
@EnableBinding(AnalyticsBinding.class)
@Slf4j
public class AnalyticsKafkaStreamsSpringCloudApplication {
    public static void main(String[] args) {
        SpringApplication.run(AnalyticsKafkaStreamsSpringCloudApplication.class, args);
    }
}
```

Spring Cloud avec Kafka

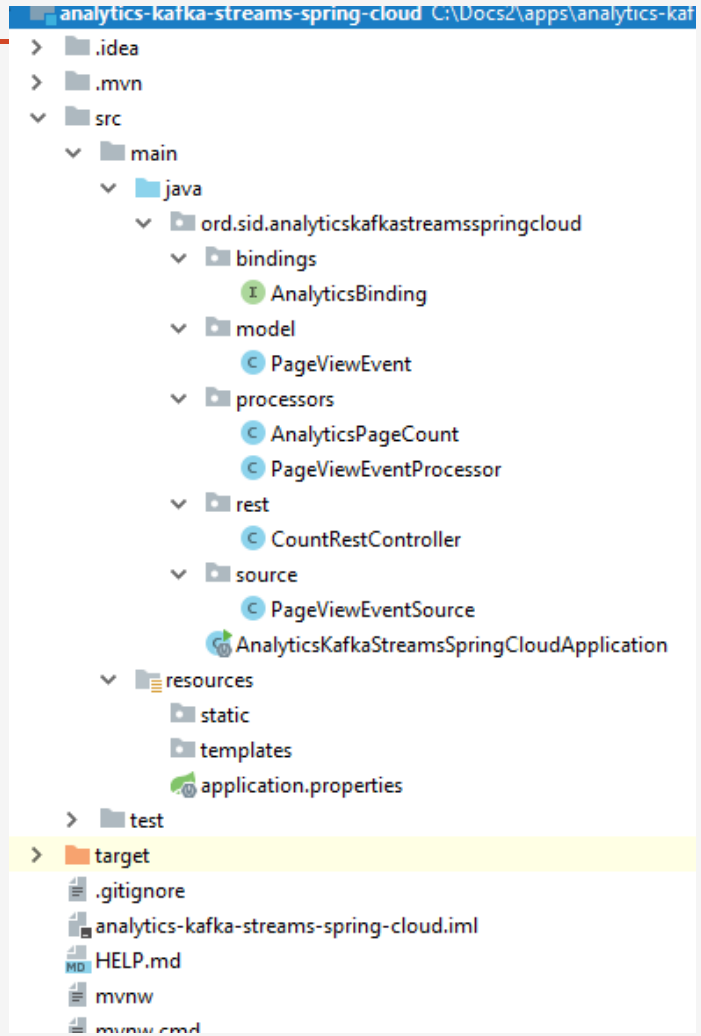
Application Console

```
2019-09-12 17:28:28.624  INFO 14516 --- [pool-1-thread-1] lyticsKafkaStreamsSpringCloudApplication : Sending message
GenericMessage [payload=PageViewEvent(userId=Hanane, page=about, duration=593), headers={id=e129e113-f7ee-a068-a603-
797eb1f9a1f3, kafka_messageKey=[B@7c2c7662, contentType=application/json, timestamp=1568305708500}]
2019-09-12 17:28:28.624  INFO 14516 --- [pool-1-thread-1] lyticsKafkaStreamsSpringCloudApplication : Sending message
=>PageViewEvent(userId=Hanane, page=about, duration=593)
2019-09-12 17:28:29.492  INFO 14516 --- [pool-1-thread-1] lyticsKafkaStreamsSpringCloudApplication : Sending message
GenericMessage [payload=PageViewEvent(userId=Hanane, page=chat, duration=1046), headers={id=6a6b0911-efd9-52e6-ab2d-
caa38db80e97, kafka_messageKey=[B@321d4e3, contentType=application/json, timestamp=1568305709492}]
```

Kafka Consumer Console

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs --
property print.key=true --property print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
Hanane   {"userId":"Hanane","page":"search","duration":926}
Hanane   {"userId":"Hanane","page":"chat","duration":955}
Hassan   {"userId":"Hassan","page":"search","duration":403}
Hanane   {"userId":"Hanane","page":"contact","duration":140}
Mohamed  {"userId":"Mohamed","page":"contact","duration":126}
Aziz     {"userId":"Aziz","page":"profile","duration":1069}
Hassan   {"userId":"Hassan","page":"search","duration":879}
Yassine  {"userId":"Yassine","page":"about","duration":999}
Samir    {"userId":"Samir","page":"search","duration":195}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams



Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream-binder-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream-binder-kafka-
streams</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-stream-test-support</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```java
package ord.sid.analyticskafkastreamsspringcloud.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PageViewEvent{
    private String userId,page;
    private long duration;
}
```

```java
package ord.sid.analyticskafkastreamsspringcloud.bindings;

import
ord.sid.analyticskafkastreamsspringcloud.model.PageViewEvent;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;

public interface AnalyticsBinding{
    String PAGE_VIEWS_OUT="pvout";
    String PAGE_VIEWS_IN="pvin";
    String PAGE_VIEW_COUNT_OUT = "pcout";
    String PAGE_VIEW_COUNT_IN = "pcin";
    String PAGE_COUNT_MV = "PAGE_COUNT";

    @Output(PAGE_VIEWS_OUT)
    MessageChannel pageViewsOut();
    @Input(PAGE_VIEWS_IN)
    KStream<String, PageViewEvent> pageViewsIn();

    @Output(PAGE_VIEW_COUNT_OUT)
    KStream<String,Long> pageCountOutput();
    @Input(PAGE_VIEW_COUNT_IN)
    KTable<String,Long> pageCountInput();
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package
ord.sid.analyticskafkastreamsspringcloud.source;

import lombok.extern.slf4j.Slf4j;
import
ord.sid.analyticskafkastreamsspringcloud.bindings.Analy
ticsBinding;
import
ord.sid.analyticskafkastreamsspringcloud.model.PageView
Event;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.kafka.support.KafkaHeaders;
import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel;
import
org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Component;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
```

```
@Component
@Slf4j
public class PageViewEventSource implements ApplicationRunner {
    private MessageChannel pageViewsOutChannel;
    @SuppressWarnings("SpringJavaInjectionPointsAutowiringInspection")
    public PageViewEventSource(AnalyticsBinding analyticsBinding) {
        this.pageViewsOutChannel = analyticsBinding.pageViewsOut();
    }
    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("...............................................;");
        List<String> names= Arrays.asList("Hassan","Mohamed","Hanane","Yassine","Samir","Aziz");
        List<String> pages= Arrays.asList("blog","chat","profile","about","contact","vote","search");
        Runnable runnable=()->{
            String rPage=pages.get(new Random().nextInt(pages.size()));
            String rName=names.get(new Random().nextInt(names.size()));
            PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,100+(int)(Math.random()*1000));
            //kafkaTemplate.send("pvts",pageViewEvent.getUserId(),pageViewEvent);
            Message<PageViewEvent> eventMessage = MessageBuilder
                    .withPayload(pageViewEvent)
                    .setHeader(KafkaHeaders.MESSAGE_KEY, pageViewEvent.getUserId())
                    .build();
            try {
                this.pageViewsOutChannel.send(eventMessage);
                log.info("Sending message "+eventMessage.toString());
            } catch (Exception e) {
                e.printStackTrace();
            }
        };
        System.out.println("-----------------------------------------");
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,1000,1000, TimeUnit.MILLISECONDS);
    }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package ord.sid.analyticskafkastreamsspringcloud.processors;

import lombok.extern.slf4j.Slf4j; import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;
import ord.sid.analyticskafkastreamsspringcloud.model.PageViewEvent; import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.kstream.KStream; import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Materialized; import org.apache.kafka.streams.kstream.TimeWindows;
import org.springframework.cloud.stream.annotation.Input; import org.springframework.cloud.stream.annotation.StreamListener;
import org.springframework.messaging.handler.annotation.SendTo; import org.springframework.stereotype.Component; import java.util.Date;
@Component
@Slf4j
public class PageViewEventProcessor {
    @StreamListener
    @SendTo({AnalyticsBinding.PAGE_VIEW_COUNT_OUT})
    public KStream<String, Long> process(@Input(AnalyticsBinding.PAGE_VIEWS_IN) KStream<String, PageViewEvent>
events){
        return events
                .filter((k, v) -> v.getDuration() > 5)
                //.map((k,v)->new KeyValue<>(v.getPage(),"0"));
                .map((k, v) -> new KeyValue<>(v.getPage(), "0"))
                .groupByKey()
                //.windowedBy(TimeWindows.of(10000))
                .count(Materialized.as(AnalyticsBinding.PAGE_COUNT_MV)).toStream();
//.map((k,v)->new KeyValue<>("["+new Date(k.window().start())+"=>"+new Date(k.window().end())+"]"+":=>"+k.key(),v));

    }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
#spring.cloud.stream.kafka.streams.binder.application-id=page-views-count-app-2
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringSerializer
#defaults
spring.cloud.stream.kafka.streams.binder.configuration.commit.interval.mms=1000
spring.cloud.stream.kafka.streams.binder.configuration.default.key.serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.binder.configuration.default.value.serde=org.apache.kafka.common.serialization.Serdes$StringSerde
# Page view out Bindings
spring.cloud.stream.bindings.pvout.destination=pvs
spring.cloud.stream.bindings.pvout.producer.header-mode=raw
#Page view in
spring.cloud.stream.bindings.pvin.destination=pvs
spring.cloud.stream.bindings.pvin.consumer.header-mode=raw
spring.cloud.stream.kafka.streams.bindings.pvin.consumer.application-id=page-views-app-id
# Kafka Brokers hosts
spring.kafka.bootstrap-servers=192.168.57.3:9092
#Page Count out
spring.cloud.stream.bindings.pcout.destination=pcs2
spring.cloud.stream.bindings.pcout.producer.use-native-encoding=true
spring.cloud.stream.kafka.streams.bindings.pcout.producer.key-serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.bindings.pcout.producer.value-serde=org.apache.kafka.common.serialization.Serdes$LongSerde
spring.cloud.stream.bindings.pcout.producer.header-mode=raw
#Page Count In
spring.cloud.stream.bindings.pcin.destination=pcs2
spring.cloud.stream.bindings.pcin.consumer.use-native-decoding=true
spring.cloud.stream.bindings.pcin.group=pcs-gr
#spring.cloud.stream.bindings.pcin.content-type=application/json
spring.cloud.stream.kafka.streams.bindings.pcin.consumer.key-serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.bindings.pcin.consumer.value-serde=org.apache.kafka.common.serialization.Serdes$LongSerde
spring.cloud.stream.bindings.pcin.consumer.header-mode=raw
spring.cloud.stream.kafka.streams.bindings.pcin.consumer.application-id=analytics-app-id
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

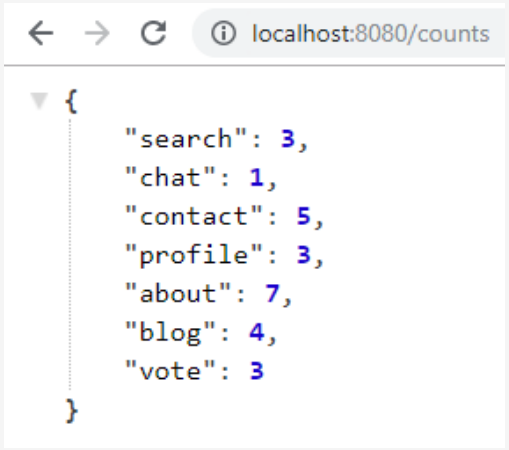
```
package ord.sid.analyticskafkastreamsspringcloud.processors;
import lombok.extern.slf4j.Slf4j;
import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;
import org.apache.kafka.streams.kstream.KTable;
import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.StreamListener;
import org.springframework.stereotype.Component;
@Component
@Slf4j
public class AnalyticsPageCount {
  @StreamListener
  public void processData(@Input(AnalyticsBinding.PAGE_VIEW_COUNT_IN) KTable<String,Long> counts){
      counts.toStream().foreach((k,v)->{
          log.info(k+"=>"+v);
      });
  }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package ord.sid.analyticskafkastreamsspringcloud.rest;
import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.state.KeyValueIterator;import org.apache.kafka.streams.state.QueryableStoreTypes;
import org.apache.kafka.streams.state.ReadOnlyKeyValueStore;import org.springframework.cloud.stream.binder.kafka.streams.InteractiveQueryService;
import org.springframework.web.bind.annotation.GetMapping;import org.springframework.web.bind.annotation.RestController;
import java.util.HashMap;import java.util.Map;
@RestController
public class CountRestController {
    //@Depricated private QueryableStoreRegistry queryableStoreRegistry;
    private InteractiveQueryService queryableStoreRegistry;
    public CountRestController(InteractiveQueryService queryableStoreRegistry) {
        this.queryableStoreRegistry = queryableStoreRegistry;
    }
    @GetMapping("/counts")
    public Map<String,Long> counts(){
        Map<String,Long> results=new HashMap<>();
        ReadOnlyKeyValueStore<String, Long> queryableStoreType =
this.queryableStoreRegistry.getQueryableStore(AnalyticsBinding.PAGE_COUNT_MV, QueryableStoreTypes.keyValueStore());
        KeyValueIterator<String,Long> all=queryableStoreType.all();
        while(all.hasNext()){
            KeyValue<String,Long> item=all.next();
            results.put(item.key,item.value);
        }
        return results;
    }
}
```

Spring Cloud avec Kafka : WindowStore

```
public Map<String,Long> counts(){
    Map<String,Long> results=new HashMap<>();
    ReadOnlyWindowStore<String, Long> queryableStoreType =
this.queryableStoreRegistry.getQueryableStore(AnalyticsBinding.PAGE_COUNT_MV,
QueryableStoreTypes.windowStore());
    KeyValueIterator<Windowed<String>,Long> all=queryableStoreType.all();

    while(all.hasNext()){
        KeyValue<Windowed<String>,Long> item=all.next();
        results.put(item.key.key(),item.value);
    }
    return results;
}
```

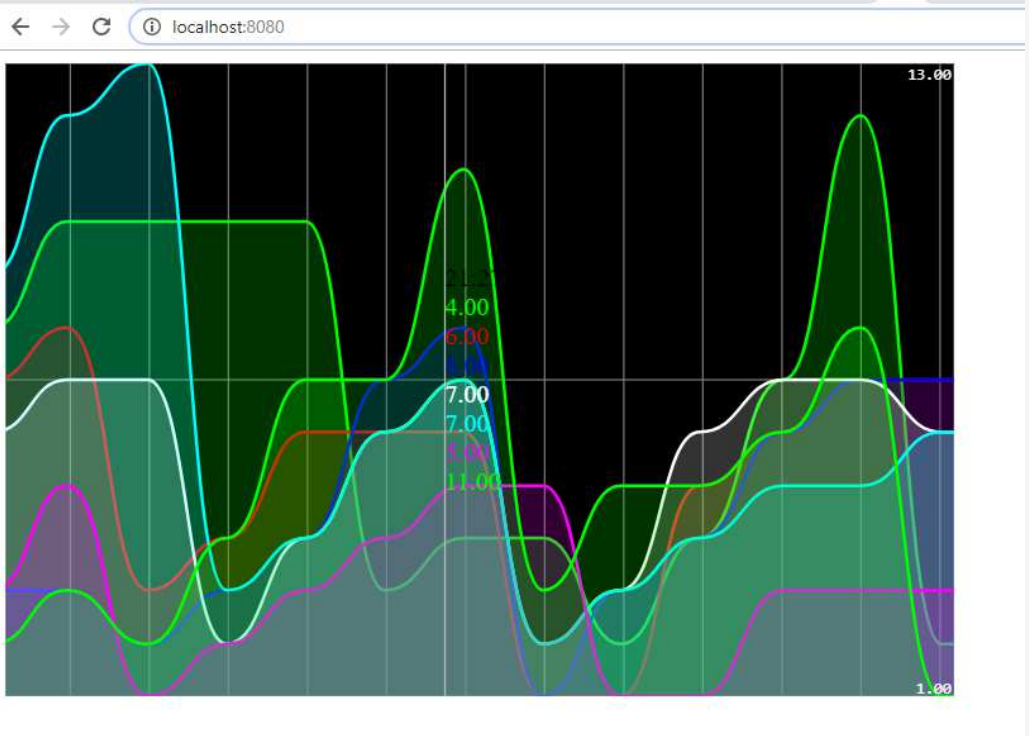


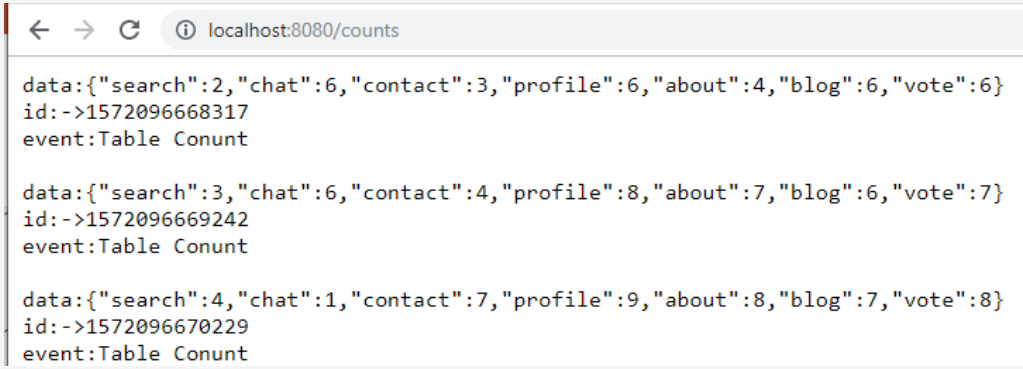

```
@StreamListener
@SendTo(AnalyticsBinding.PAGE_VIEW_COUNT)
public KStream<String,Long> process(@Input(AnalyticsBinding.PAGE_VIEW_IN)
KStream<String,PageViewEvent> pageViewEventKStream){

    return
            pageViewEventKStream.filter((k,v)->v.getDuration()>5)
            .map((k,v)->new KeyValue<>(v.getPage(),"0"))
                    .groupByKey()
            .windowedBy(TimeWindows.of(5000))
                    .count(Materialized.as(AnalyticsBinding.PAGE_COUNT_MV))
                    .toStream()
            .map((k,v)->new KeyValue<>(k.key(),v));
}
```

Spring Cloud avec Kafka : WindowStore with Server Sent Event

```
@GetMapping(value = "/counts",produces = MediaType.APPLICATION_STREAM_JSON_VALUE)
public SseEmitter counts() throws Exception{

    SseEmitter emitter = new SseEmitter();
   Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()-> {
       try {
           Map<String, Long> results = new HashMap<>();
           ReadOnlyWindowStore<String, Long> queryableStoreType =
this.queryableStoreRegistry.getQueryableStore(AnalyticsBinding.PAGE_COUNT_MV,
QueryableStoreTypes.windowStore());
           KeyValueIterator<Windowed<String>, Long> all = queryableStoreType.all();
           while (all.hasNext()) {
               KeyValue<Windowed<String>, Long> item = all.next();
               results.put(item.key.key(), item.value);
           }
           SseEmitter.SseEventBuilder event = SseEmitter.event()
                   .data(new ObjectMapper().writeValueAsString(results))
                   .id("->" + System.currentTimeMillis())
                   .name("Table Conunt");
           emitter.send(event);
       } catch (Exception ex) {
           emitter.completeWithError(ex);
       }
   },1000,1000,TimeUnit.MILLISECONDS);
       return emitter;
}
```

Spring Cloud avec Kafka : WindowStore with Server Sent Event





Spring Cloud avec Kafka : WindowStore with Server Sent Event

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Analytics</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/smoothie/1.34.0/smoothie.min.js"></script>
</head>
<body>

<canvas id="chart2" width="600" height="400"></canvas>
<script>
    var index=-1;
    randomColor = function() {
        ++index;
        if (index >= colors.length) index = 0; return colors[index];
    }
    var pages=["search","chat","contact","profile","about","blog","vote"];
    var colors=[
        { sroke : 'rgba(0, 255, 0, 1)', fill : 'rgba(0, 255, 0, 0.2)' },
        { sroke : 'rgba(255, 0, 0, 1)', fill : 'rgba(255, 0, 0, 0.2)'},
        { sroke : 'rgba(0, 0, 255, 1)', fill : 'rgba(0, 0, 255, 0.2)'},
        { sroke : 'rgba(255, 255, 255, 1)', fill : 'rgba(255, 255, 255, 0.2)'},
        { sroke : 'rgba(0, 255, 255, 1)', fill : 'rgba(0, 255, 255, 0.2)'},
        { sroke : 'rgba(255, 0, 255, 1)', fill : 'rgba(255, 0, 255, 0.2)'}
        ];
```

Spring Cloud avec Kafka : WindowStore with Server Sent Event

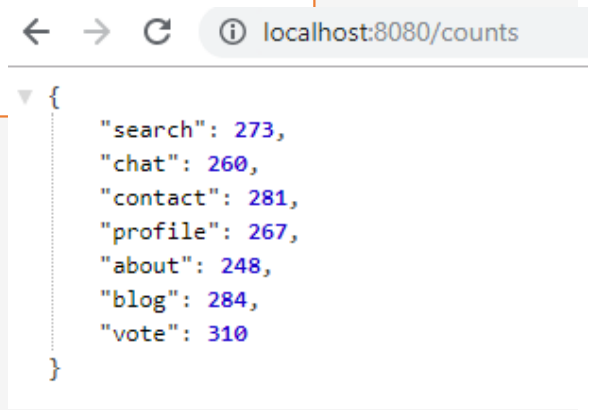
```
var courbe = [];
    var smoothieChart = new SmoothieChart({tooltip: true});
    smoothieChart.streamTo(document.getElementById("chart2"), 500);
    pages.forEach(function(v){
       courbe[v]=new TimeSeries();
        col = randomColor();
        smoothieChart.addTimeSeries(courbe[v], {strokeStyle : col.sroke, fillStyle : col.fill, lineWidth : 2
        });
    });
    var stockEventSource= new EventSource("counts");
    stockEventSource.addEventListener("message", function (event) {
        pages.forEach(function(v){
            val=JSON.parse(event.data)[v];
            courbe[v].append(new Date().getTime(),val);
        });
    }, false);

</script>
</body>
</html>
</body>
</html>
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package ord.sid.analyticskafkastreamsspringcloud;
import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;

@SpringBootApplication
@EnableBinding(AnalyticsBinding.class)
public class AnalyticsKafkaStreamsSpringCloudApplication {
    public static void main(String[] args) {
        SpringApplication.run(AnalyticsKafkaStreamsSpringCloudApplication.class, args);
    }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
2019-09-13 17:32:01.572  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Hassan, page=about, duration=1092), headers={id=af90bc14-23cb-d912-5b39-776945977044,
kafka_messageKey=Hassan, contentType=application/json, timestamp=1568392321572}]
2019-09-13 17:32:02.573  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Hanane, page=chat, duration=583), headers={id=24409bf8-4c8d-e110-1d05-300f4238eb58,
kafka_messageKey=Hanane, contentType=application/json, timestamp=1568392322572}]
2019-09-13 17:32:03.573  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Samir, page=contact, duration=1066), headers={id=22e05b15-90ce-23ac-2a48-85272f88c045,
kafka_messageKey=Samir, contentType=application/json, timestamp=1568392323573}]
2019-09-13 17:32:04.572  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Samir, page=search, duration=795), headers={id=8c327648-6fdb-ed1d-707b-08f0f4f101f5,
kafka_messageKey=Samir, contentType=application/json, timestamp=1568392324572}]
2019-09-13 17:32:04.623  INFO 18160 --- [read-2-producer] org.apache.kafka.clients.Metadata        : Cluster ID: OTtUmHuBS1SmXvIxNVzYtA
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : profile=>265
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : blog=>283
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : search=>272
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : vote=>308
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : about=>248
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : chat=>260
2019-09-13 17:32:04.625  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : contact=>281
```



Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
0$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs --property
print.key=true --property print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
Hassan  {"userId":"Hassan","page":"vote","duration":844}
Samir   {"userId":"Samir","page":"search","duration":551}
Mohamed {"userId":"Mohamed","page":"about","duration":704}
Samir   {"userId":"Samir","page":"contact","duration":972}
Samir   {"userId":"Samir","page":"chat","duration":883}
Hassan  {"userId":"Hassan","page":"vote","duration":135}
Hanane  {"userId":"Hanane","page":"blog","duration":236}
Samir   {"userId":"Samir","page":"vote","duration":731}
Hassan  {"userId":"Hassan","page":"search","duration":503}
Aziz    {"userId":"Aziz","page":"contact","duration":484}
Samir   {"userId":"Samir","page":"about","duration":810}
Samir   {"userId":"Samir","page":"chat","duration":141}
Aziz    {"userId":"Aziz","page":"search","duration":1011}
Yassine {"userId":"Yassine","page":"contact","duration":178}
Hanane  {"userId":"Hanane","page":"profile","duration":424}
Samir   {"userId":"Samir","page":"about","duration":1042}
Mohamed {"userId":"Mohamed","page":"vote","duration":923}
Hassan  {"userId":"Hassan","page":"blog","duration":494}
Hassan  {"userId":"Hassan","page":"search","duration":225}
Mohamed {"userId":"Mohamed","page":"about","duration":985}
Yassine {"userId":"Yassine","page":"contact","duration":730}
Hanane  {"userId":"Hanane","page":"contact","duration":947}
```

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic pcs2 --property print.key=true --property
print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDes
erializer --property
value.deserializer=org.apache.kafka.common.serialization.LongDes
erializer
profile 265
blog    283
search  272
vote    308
about   248
chat    260
contact 281
```

Apache Spark

- Apache Spark est un framework de traitements distribué en Big Data
- Celui-ci a originellement été développé par AMPLab, de l'Université UC Berkeley, en 2009 et passé open source sous forme de projet Apache en 2010.
- C'est un moteur de calcul distribué sur des données massives à partir des sources de données distribuées provenant de tout système de partitionnement de données sur lesquels on peut appliquer des traitements parallèles comme HDFS, Mongo Db, Cassandra, ElasticSearch etc.
- Spark est un moteur d'exécution avec des opérateurs de haut niveau (Map, Reduce, Group, Join)
- Des opérateurs qui prennent des fonctions en paramètres et appliquent ces fonctions à des données dans un environnement distribuées.