

NoSQL, systèmes distribués et passage en production de projets Data

Thierry GAMEIRO MARTINS

Séances

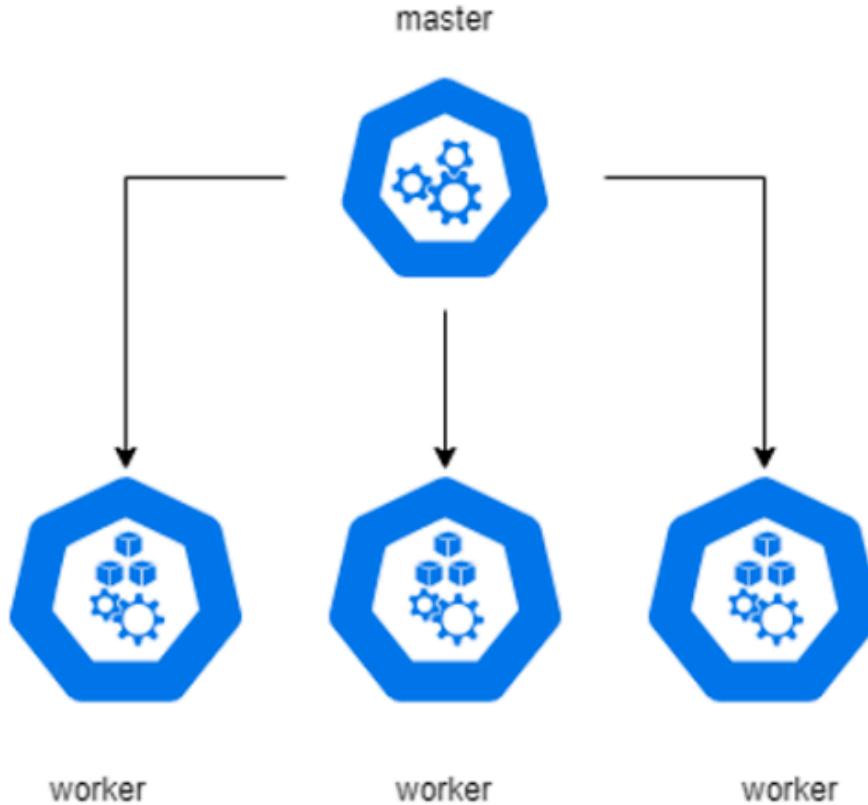
1. Introduction et prise en main d'Onyxia
2. Le stockage des données en NoSQL

3. Les systèmes de traitement distribués

4. Le passage en production
5. Orchestration et pratique DevOps
6. Déploiement conteneurisé sous Kubernetes

L'écosystème Hadoop

Concept : Architecture worker/master



Le framework Hadoop

Hadoop est un framework pour :

- le stockage (HDFS)
- le traitement (MapReduce) de grands ensemble de données

Utilise la puissance de multiple machines
commodity hardware

D'autre outils s'intègrent au framework
Hadoop (*Hive, Yarn, Spark, Sqoop, Flume, Ambari, etc...*)



HDFS

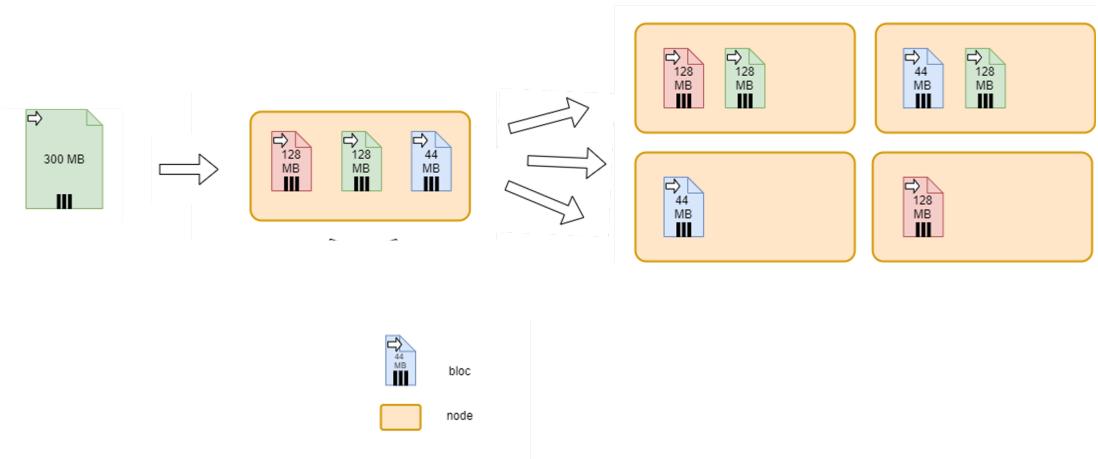
- Hadoop Distributed File System
(HDFS) est un système de fichier hautement disponible
- Basé de GFS de Google
- Réplique les données sur plusieurs machines pour :
 - la tolérance aux pannes
 - une scalabilité élevée
 - héberger de grands volume de données



Lorsque un fichier est ajouté le fichier est :

- découpé en bloc
- répliqué vers d'autres nœuds

Le paramètre de réPLICATION par défaut est à 3, et la taille de bloc par défaut est de 128 Mo





Name Node

Ce nœud dit « master » gère via des fichiers de métadonnées (FsImage et EditsLog) :

- l'arborescence et les états du système de fichier (les permissions, les dates, etc.)
- les informations des DataNodes et la position des blocs sous forme d'objet (stocké en mémoire)



Data Node

Ce nœud dit « worker » sont les nœuds qui stockent les blocs de données :

- transmettent aux clients les blocs correspondants au fichier demandé
- suppriment, stockent et réplique les blocs
- périodiquement, retourne la liste des blocs qu'ils possèdent (*blockreport*)

Exercice

Alice, Bob et Carl ont chacun déposé pendant 1 an des données sur HDFS d'une taille journalière de 1024 Mo, mais répartit différemment :

- Alice : 1 fichier de 1024 Mo
- Bob : 8 fichiers de 128Mo
- Carl : 1024 fichiers de 1 Mo

La configuration de leur cluster HDFS est celle par défaut (réPLICATION DES BLOCS À 3 ET LA TAILLE DES BLOCS À 128 Mo).

Chaque objet stocké sur HDFS requiert 150 octets au NameNode.

Calculer la consommation en mémoire sur le NameNode de chaque utilisateur au bout d'un an.

Correction

Alice : 1 fichier de 1024 Mo

- 1 fichier (inode)
- 8 blocs
 $(1024\text{Mo}/128\text{Mo}) \times 3$
(réPLICATION) = 24
- Total = 25 objets
 $(24+1)*150 \text{ octet} = 3,75 \text{ Ko / jours}$
Consommation annuelle :
 $3,75 \text{ Ko} * 365 = 1,37 \text{ Mo}$

Bob : 8 fichiers de 128 Mo

- 8 fichiers (inode)
- 8 blocs $(128\text{Mo}*8 = 1024\text{Mo})$
 $\times 3$ (réPLICATION) = 24
- Total = 32 objets
 $(24+8)*150 \text{ octet} = 4,8 \text{ Ko / jours}$

Consommation annuelle :
 $4,8 \text{ Ko} * 365 = 1,75 \text{ Mo}$

Carl : 1024 fichiers de 1 Mo

- 1024 fichiers (inode)
- 1024 blocs x3
(réPLICATION)=
3072
- Total = 4096 objets
 $(3072+ 1024) * 150 \text{ octet} = 614,4 \text{ Ko / jours}$

Consommation annuelle :
 $614,4 \text{ Ko} * 365 = 224,25 \text{ Mo}$

"Small files are big problem in Hadoop" Szele Balint - Cloudera

Qu'est ce qu'un petit fichier ?

Un fichier considérablement plus petit que la taille d'un bloc

Une forte consommation mémoire sur le NameNode

Chaque fichier, répertoire et bloc dans HDFS sont représenté en tant qu'objet occupant 150 octets dans la mémoire du NameNode

10M de fichiers utiliseraient environ 6 Go de RAM

Un accès à la donnée inefficace

En général, lorsqu'on stocke des petits fichiers, on en stocke beaucoup

La lecture de nombreux petits fichiers entraîne beaucoup de recherches et de sauts entre les DataNodes

DataNode en panne

1. Communications par *heartbeat*

(timeout de 10 minutes), alors :

- le DataNode est hors service
- le contenu des blocs hébergées sont indisponibles

2. NameNode planifie alors la création de nouveaux blocs sur d'autres DataNodes

NameNode en panne

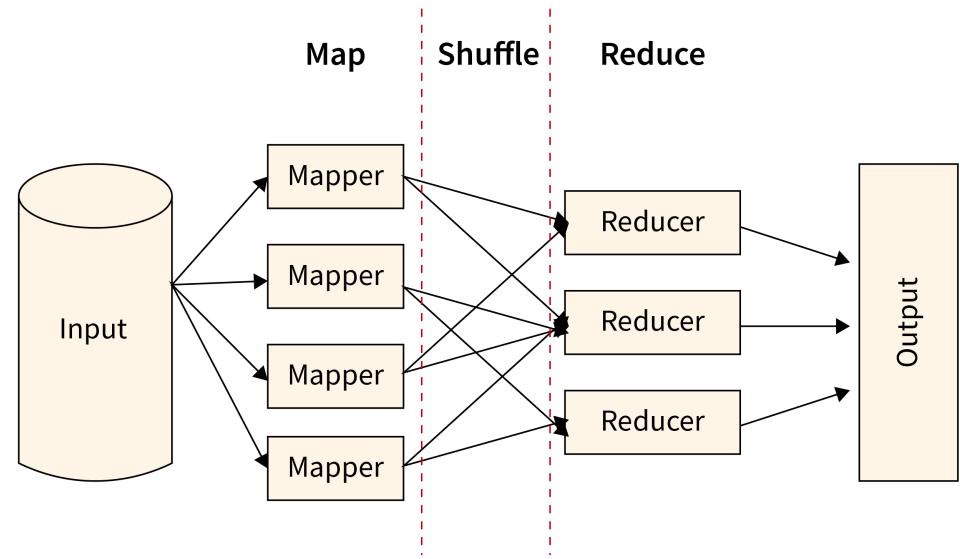
Le NameNode est un *SPOF*

- Installer un *secondary NameNode* qui extrait les modifications et l'état écrit sur le NameNode (*checkpointing*)
- mode HA qui permet d'avoir deux NameNodes (un actif et un passif)
 - Les DataNode envoient des rapports au deux NameNodes
 - Les clients communiquent avec celui actif

hadoop fs -ls	Afficher le contenu du répertoire racine
hadoop fs -put file.txt	Upload un fichier dans hadoop (à partir du répertoire courant linux)
hadoop fs -get file.txt	Download un fichier à partir de hadoop sur votre disque local
hadoop fs -tail file.txt	Lire les dernières lignes du fichier
hadoop fs -cat file.txt	Affiche tout le contenu du fichier
hadoop fs -mv file.txt newfile.txt	Renommer le fichier
hadoop fs -rm newfile.txt	Supprimer le fichier
hadoop fs -mkdir myinput	Créer un répertoire
hadoop fs -cat file.txt less	Lire le fichier page par page

Fonctionnement de MapReduce

- Algorithme de traitement de données parallélisé
 - *map* qui traite une paire de clé/valeur en entrée pour générer un autre ensemble de paires de clé/valeur intermédiaire
 - *reduce* qui fusionne toutes les valeurs intermédiaires identiques et qui les renvoi
- Peut résoudre des tâches courantes (calculer une statistique, filtrer, récupérer des index, jointure, etc.)



Fonction *Map*

Applique une fonction à chaque élément de la séquence et retourne la résultat de la séquence transformée

Exemple

```
liste = [1, 2, 3, 4, 5]
map(lambda x: x**2, liste)
# [1, 4, 9, 16, 25]
```

Fonction *Reduce*

Appliquer une fonction de manière cumulative aux éléments d'une séquence de façon à réduire la séquence à un seul élément

Exemple

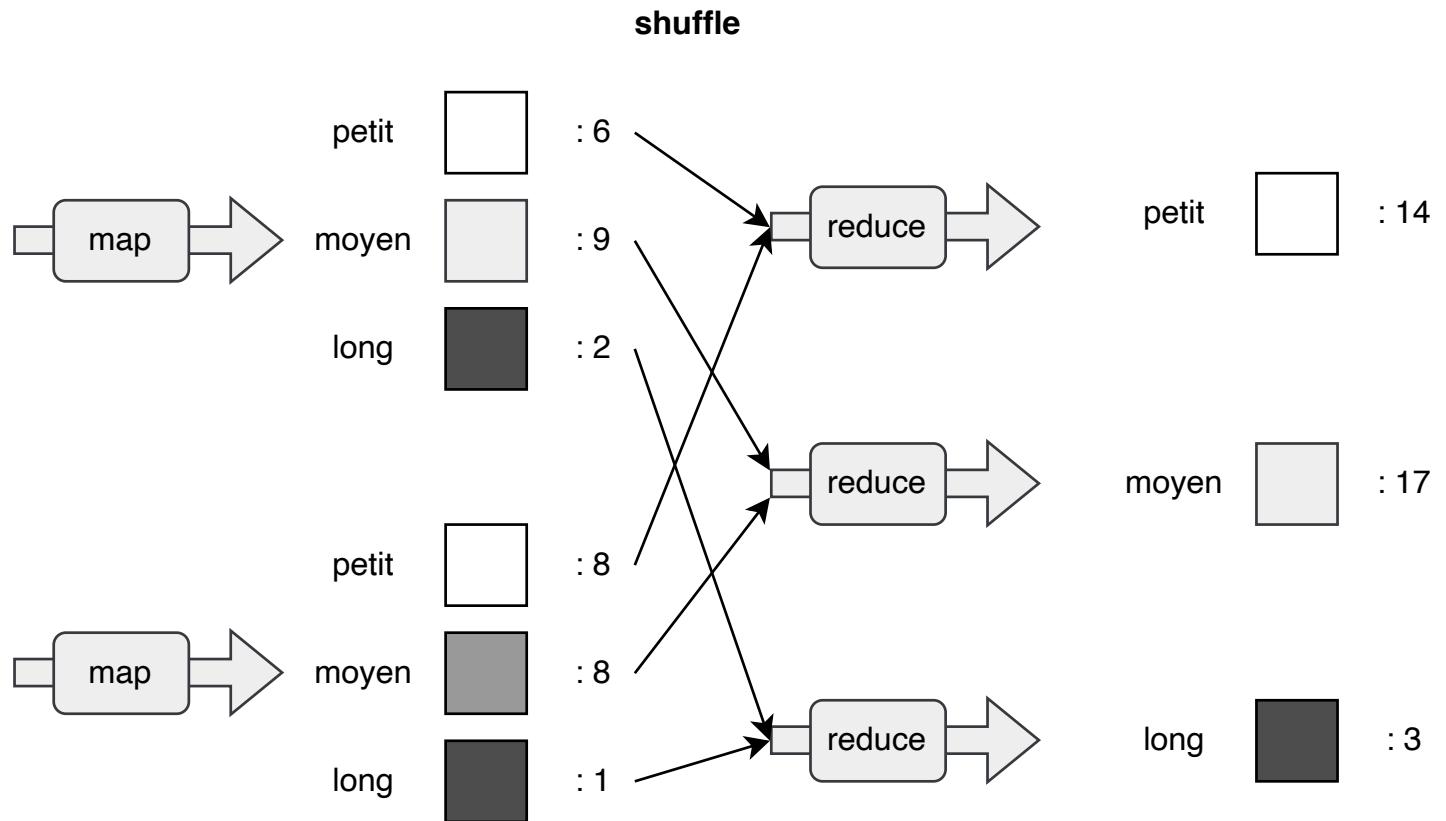
```
liste = [1, 3, 5, 6, 2]
reduce(lambda a, b: a + b, liste)
# 17

reduce(lambda a, b: a if a > b else b, liste)
# 6
```

Exemple : analyse de la longueur des mots

Le big data, littéralement les grosses données, parfois appelée données massives, désignent des ensembles de données volumineux

Dans ces nouveaux ordres de grandeurs, la capture, stockage, la visualisation des données doivent être redéfinis



Lancer un job MapReduce

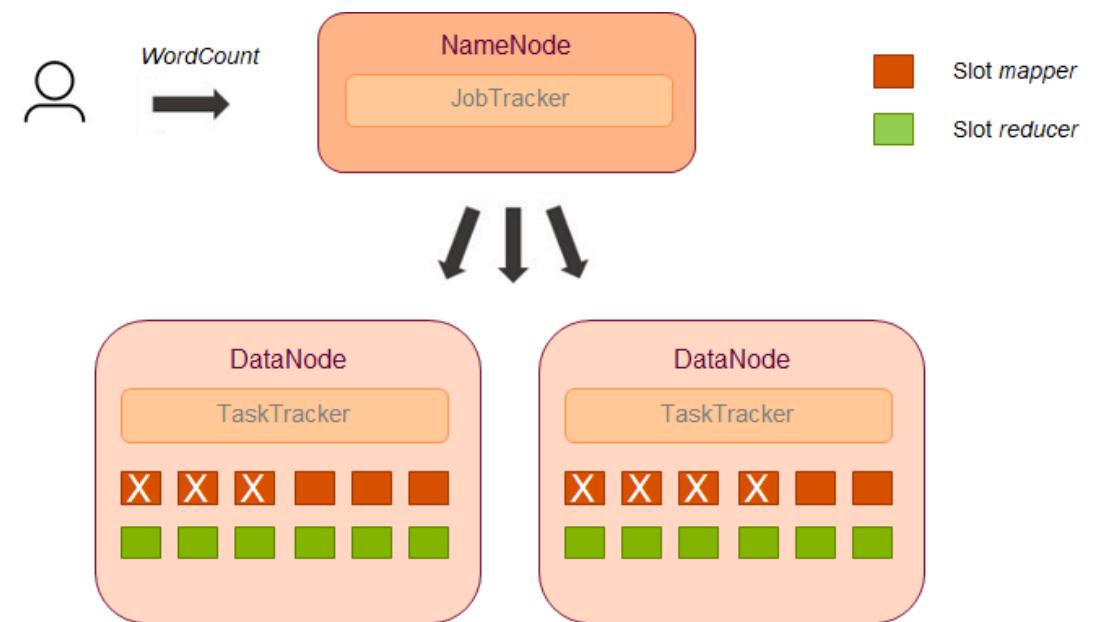
Un Job MapReduce (MRv1) est divisé en deux services :

- **JobTracker**

- l'envoi d'application de MapReduce
- la gestion des ressources
- l'ordonnancement des jobs

- **TaskTracker**

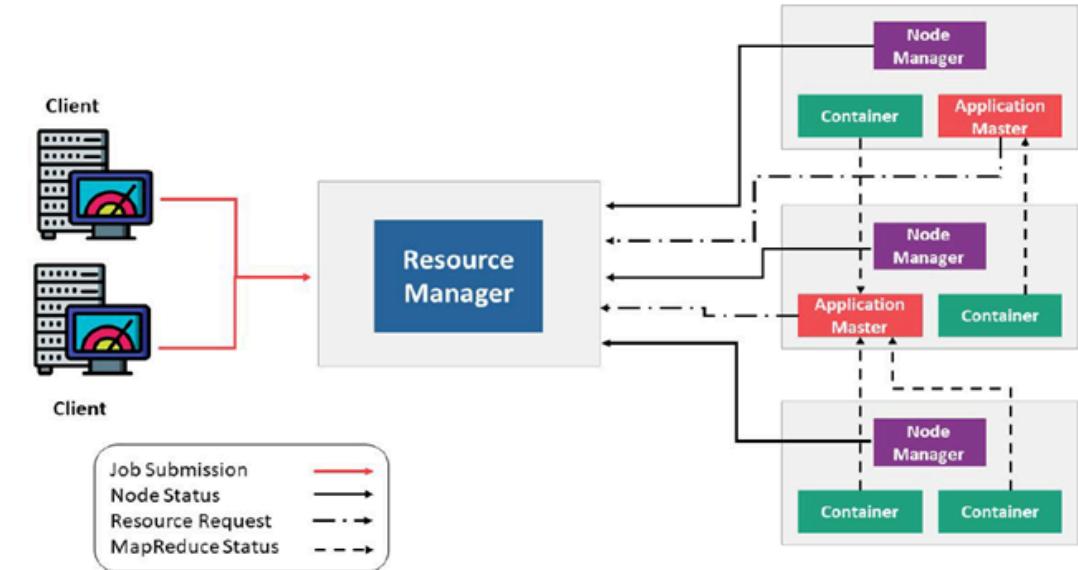
- l'exécution des jobs Map, Reduce, Shuffle/Sort, etc.



YARN (Yet Another Resource Negotiator)

YARN permet de gérer l'allocation des ressources dans un cluster Hadoop (v2)

1. Le Resource Manager déclenche un Application Master sur un Node Manager
2. L'Application Master demande auprès du Resource Manager des Containers
3. Le Resource Manager récupère informations des blocs pour placer les Containers
4. L'Application Master renvoie le résultat au Resource Manager puis au client



MapReduce : approche bas niveau

- L'approche traditionnelle consiste à utiliser des scripts en Java
- Une librairie python permet aussi de le faire (mrjob):
 - Définir un mapper et un reducer
 - Lancer la commande : python mr.py -r hadoop hdfs://my_home/input.txt

Package utilisé :

<https://mrjob.readthedocs.io/>

Le fichier `mr.py` :

```
from mrjob.job import MRJob
class Count(MRJob):
    """
    The below mapper() function defines the mapper for MapReduce and takes
    key value argument and generates the output in tuple format .
    The mapper below is splitting the line and generating a word with its own
    count i.e. 1 """
    def mapper(self, _, line):
        for word in line.split():
            yield(word, 1)
    """
    The below reducer() is aggregating the result according to their key and
    producing the output in a key-value format with its total count"""
    def reducer(self, word, counts):
        yield(word, sum(counts))

    """
    the below 2 lines are ensuring the execution of mrjob, the program will not
    execute without them"""
if __name__ == '__main__':
    Count.run()
```

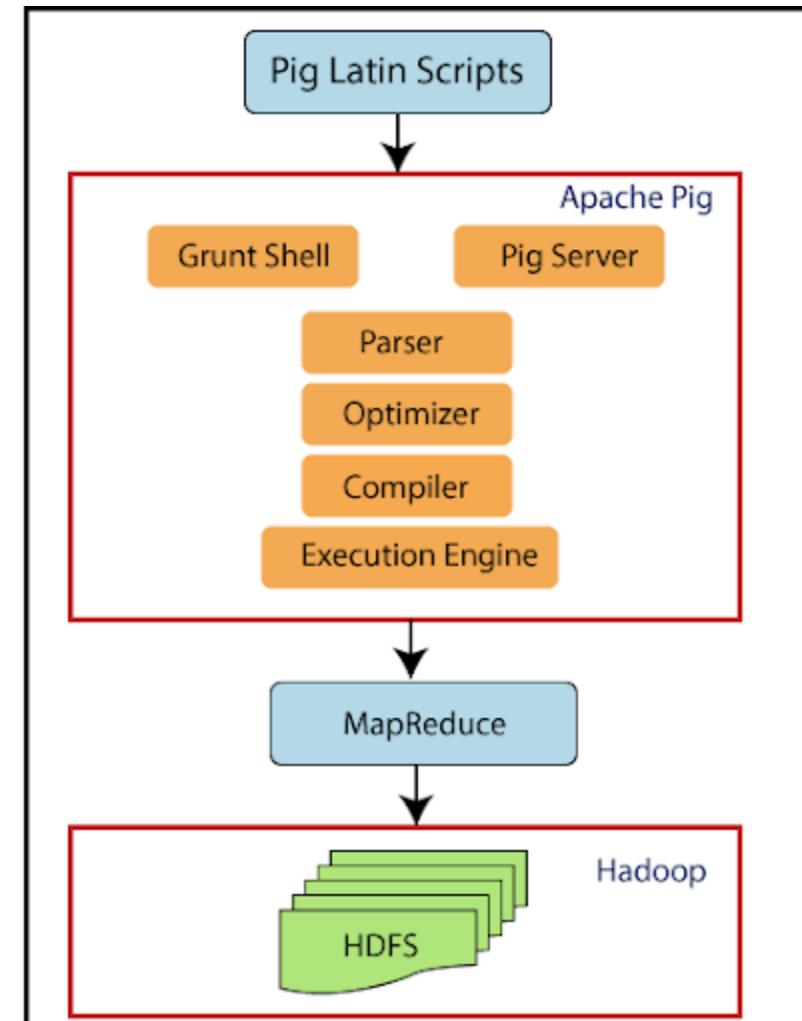
MapReduce : Pig

Langage haut niveau de traitement de données dans Hadoop

- *PigLatin* : approche par script
- *Grunt* : approche interactive

Un site de démonstration

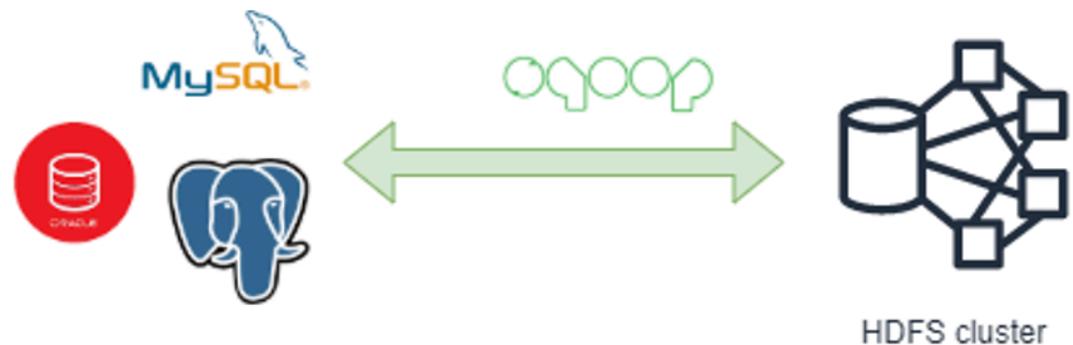
<https://www.cloudduggu.com/pig/>



MapReduce : Sqoop

Transférer des données entre HDFS et des bases de données relationnelles
(PostgreSQL, Oracle, MySQL, etc.)

- S'appuie sur la base de données pour décrire le schéma des données à importer
- Utilise MapReduce pour importer et exporter les données par partitions (fonctionnement en parallèle)



1. On possède une base de données PostgreSQL

```
postgres=# select * from random;
      time       | device_id |    cpu_usage
-----+-----+-----
2021-01-01 00:00:00+00 | 1 | 41.43055907188948
2021-01-01 01:00:00+00 | 1 | 66.20212435469352
2021-01-01 02:00:00+00 | 1 | 72.3934147227349
2021-01-01 03:00:00+00 | 1 | 29.88575987317068
2021-01-01 04:00:00+00 | 1 | 83.01145546264992
2021-01-01 05:00:00+00 | 1 | 51.34576596547164
2021-01-01 06:00:00+00 | 1 | 33.432123976358284
2021-01-01 07:00:00+00 | 1 | 16.765001543521365
2021-01-01 08:00:00+00 | 1 | 31.96407483666095
2021-01-01 09:00:00+00 | 1 | 48.616304426744335
2021-01-01 10:00:00+00 | 1 | 54.78725586668669
2021-01-01 11:00:00+00 | 1 | 97.50712636799008
2021-01-01 00:00:00+00 | 2 | 37.35620776060185
2021-01-01 01:00:00+00 | 2 | 59.436617757339974
2021-01-01 02:00:00+00 | 2 | 89.13828286054685
2021-01-01 03:00:00+00 | 2 | 44.877800338518604
2021-01-01 04:00:00+00 | 2 | 21.06702220931527
2021-01-01 05:00:00+00 | 2 | 51.84837315716084
2021-01-01 06:00:00+00 | 2 | 31.915236141604364
2021-01-01 07:00:00+00 | 2 | 12.190253241881809
```

4. Les données sont disponibles sur HDFS

```
root@hadoop-master:~# hdfs dfs -cat random/part-m-00000
2021-01-01 00:00:00.0,1,41.43055907188948
2021-01-01 01:00:00.0,1,66.20212435469352
2021-01-01 02:00:00.0,1,72.3934147227349
2021-01-01 03:00:00.0,1,29.88575987317068
```

2. La commande *sqoop import* permet de se connecter à la base de données et écrire sur HDFS (avec MapReduce) les données

```
root@hadoop-master:~# sqoop import --connect jdbc:postgresql://postgres:5432/
postgres --username test --password test --table random --split-by device_id
23/11/04 05:06:08 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/5f4e4447d30f1ab797ed8d2e2db26380/random.jar
23/11/04 05:06:08 WARN manager.PostgresqlManager: It looks like you are importing from postgresql.
23/11/04 05:06:08 WARN manager.PostgresqlManager: This transfer can be faster! Use the --direct
23/11/04 05:06:08 WARN manager.PostgresqlManager: option to exercise a postgresql-specific fast path.
23/11/04 05:06:08 INFO mapreduce.ImportJobBase: Beginning import of random
23/11/04 05:06:08 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
23/11/04 05:06:08 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
23/11/04 05:06:08 INFO client.RMProxy: Connecting to ResourceManager at hadoop-master/172.18.0.2:8032
23/11/04 05:06:10 INFO db.DataDrivenDBInputFormat: Using read committed transaction isolation
23/11/04 05:06:10 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN("device_id"), MAX("device_id") FROM "random"
23/11/04 05:06:10 INFO db.IntegerSplitter: Split size: 1; Num splits: 2 from: 1 to: 4
23/11/04 05:06:10 INFO mapreduce.JobSubmitter: number of splits:2
23/11/04 05:06:10 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1699009340983_0013
23/11/04 05:06:10 INFO impl.YarnClientImpl: Submitted application application_1699009340983_0013
23/11/04 05:06:10 INFO mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_1699009340983_0013/
23/11/04 05:06:10 INFO mapreduce.Job: Job: Running job: job_1699009340983_0013
23/11/04 05:06:10 INFO mapreduce.Job: Job: job_1699009340983_0013 running in uber mode : false
23/11/04 05:06:16 INFO mapreduce.Job: map 0% reduce 0%
23/11/04 05:06:22 INFO mapreduce.Job: map 100% reduce 0%
23/11/04 05:06:22 INFO mapreduce.Job: Job: job_1699009340983_0013 completed successfully
23/11/04 05:06:22 INFO mapreduce.Job: Counters: 30
```

3. Par défaut, 4 mappers sont créés ce qui donne 4 fichiers de sortie sur HDFS contenant chacun une partie des données

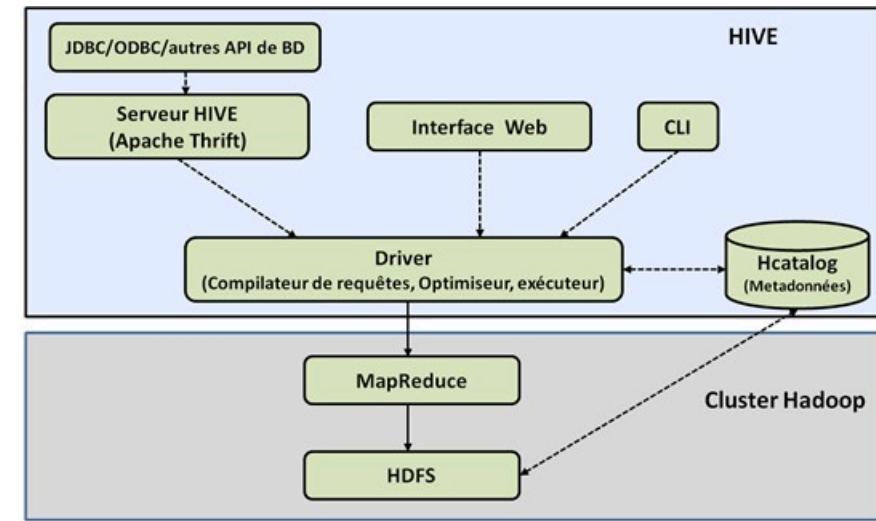
```
root@hadoop-master:~# hdfs dfs -ls random/
Found 5 items
-rw-r--r-- 2 root supergroup          0 2023-11-04 05:16 random/_SUCCESS
-rw-r--r-- 2 root supergroup      506 2023-11-04 05:16 random/part-m-00000
-rw-r--r-- 2 root supergroup      509 2023-11-04 05:16 random/part-m-00001
-rw-r--r-- 2 root supergroup      502 2023-11-04 05:16 random/part-m-00002
-rw-r--r-- 2 root supergroup      506 2023-11-04 05:16 random/part-m-00003
```

MapReduce : Apache Hive



Apache Hive est un data warehouse pour Hadoop

- Permet de manipuler les données comme du SQL avec HiveQL (*HQL*)
- Convertit les requêtes en jobs MapReduce
- Les métadonnées ou metastore sont stockées dans une BDD relationnelle



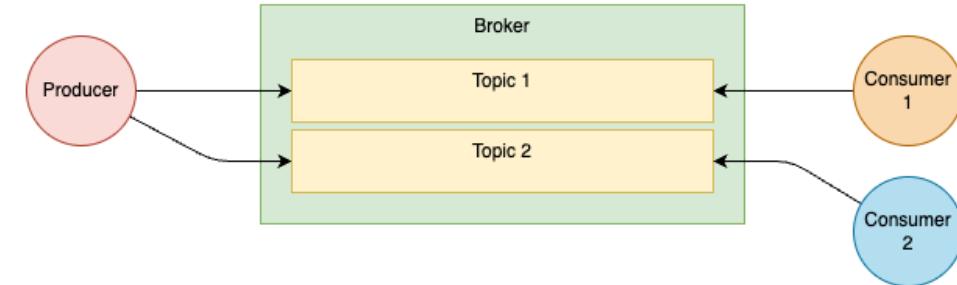
Apache Kafka



Kafka est un système de message :

- Permet de gérer des flux de données en temps réel
- Stocke les messages de manière persistante
- Distribué et résilient à la panne
- S'intègre à de nombreuses sources de données

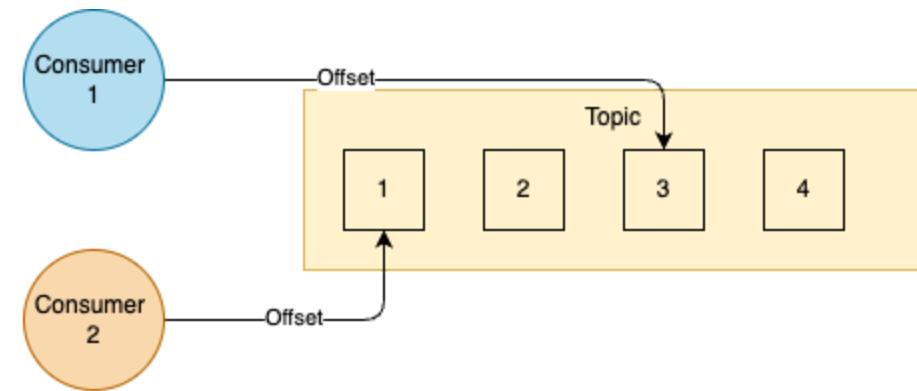
Quand ? : évènements, log, metrics, etc.



- **Producer** : service qui envoie les messages
- **Consumer** : service qui lit les messages
- **Topic** : file de messages
- **Broker** : serveur Kafka
- **Cluster** : ensemble de brokers

Système de topics et offset

- **Topic** : conteneur de messages pour une catégorie
 - Stockage des messages
 - Plusieurs consumer peuvent lire un même topic
- **Offset** : position d'un message dans un topic
 - Permet aux consommateurs de suivre la progression de lecture
 - Indice stocké dans Kafka
 - Reprise possible par le consumer (*commit, latest ou earlier*)



Création de topics

The screenshot shows the Apache Kafka UI interface. On the left, there's a sidebar with navigation links: Dashboard, test (selected), Brokers, Topics (selected), Consumers, and Kafka Connect. The main area is titled "Topics". It features a search bar labeled "Search by Topic Name" and a toggle switch for "Show Internal Topics". Below these are three buttons: "Delete selected topics", "Copy selected topic", and "Purge messages of selected topics". A table lists the following topics:

	Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size	Actions
<input type="checkbox"/>	file	1	0	1	6	428 Bytes	⋮
<input type="checkbox"/>	mongodb.defaultdb.usagers.defaultdb.usagers	1	0	1	1	690 Bytes	⋮
<input type="checkbox"/>	quickstart-config	1	0	3	94	62 KB	⋮
<input type="checkbox"/>	quickstart-offsets	25	0	3	2	1 KB	⋮
<input type="checkbox"/>	quickstart-status	5	0	3	43	93 KB	⋮
<input type="checkbox"/>	test	1	0	1	0	0 Bytes	⋮

Ou via la CLI

```
bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhost:9092
```

Envoyer et recevoir des messages

Producer

```
from kafka import KafkaProducer
import json

producer = KafkaProducer(
    bootstrap_servers=['kafka:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

producer.send('test_topic', value={'a': 1})
```

Consumer

```
from kafka import KafkaConsumer
import json

consumer = KafkaConsumer(
    'test_topic',
    bootstrap_servers=['kafka:9092'],
    auto_offset_reset='latest',
    enable_auto_commit=True,
    value_deserializer=lambda x: json.loads(x.decode('utf-8'))
)

for message in consumer:
    print(f"Message reçu : {message.value}")
```

Ou via la CLI :

```
$ bin/kafka-console-producer.sh
--topic quickstart-events
--bootstrap-server kafka:9092
>This is my first event
>This is my second event
```

Ou via la CLI :

```
$ bin/kafka-console-consumer.sh
--topic quickstart-events
--from-beginning
--bootstrap-server kafka:9092
This is my first event
This is my second event
```

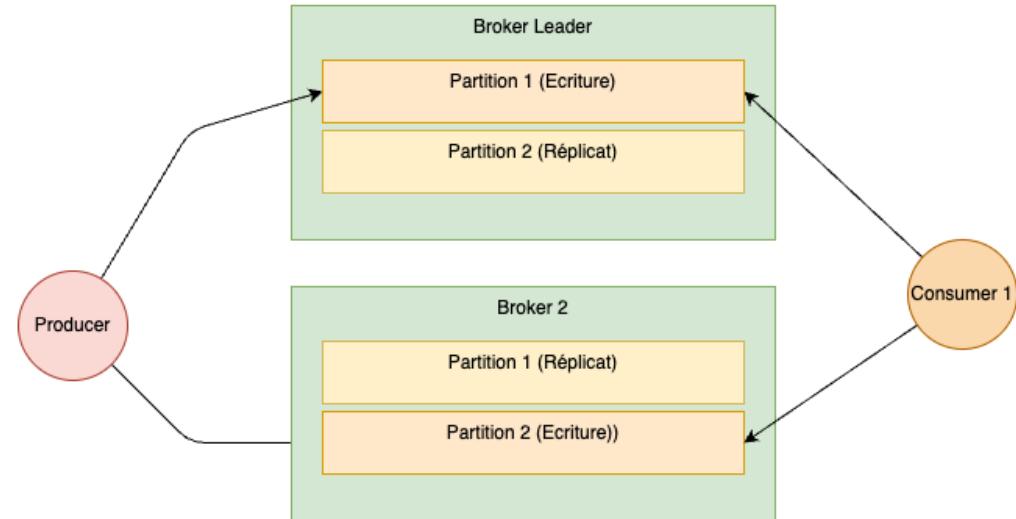
Partitionnement et réPLICATION

- **Partitionnement :**

- Divise les topics pour distribuer la charge
- Fonction de hashage pour le partitionnement

- **RéPLICATION :**

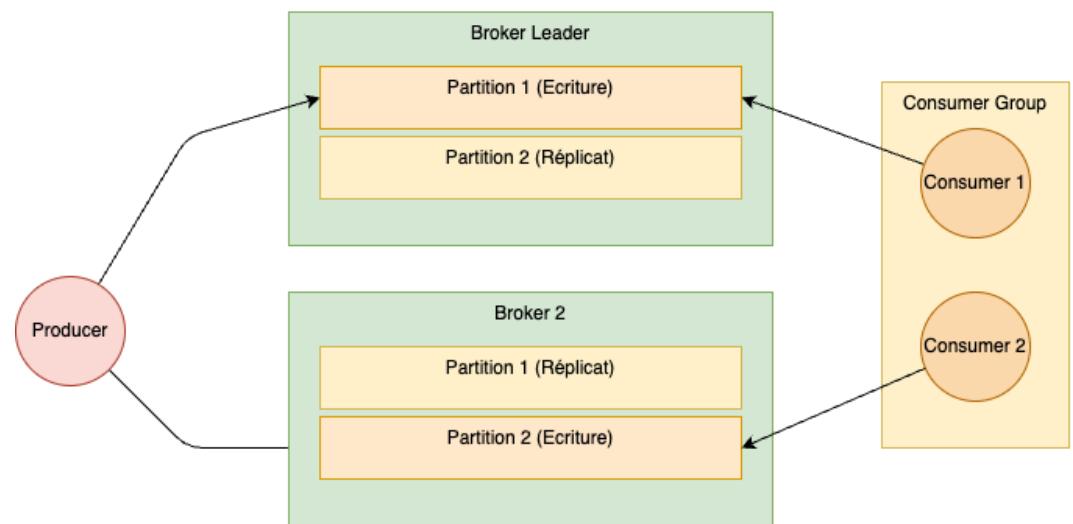
- Chaque partition est répliquée (HA)
- Un des réplicas est élu comme *Leader* (écriture)



Consumer Group

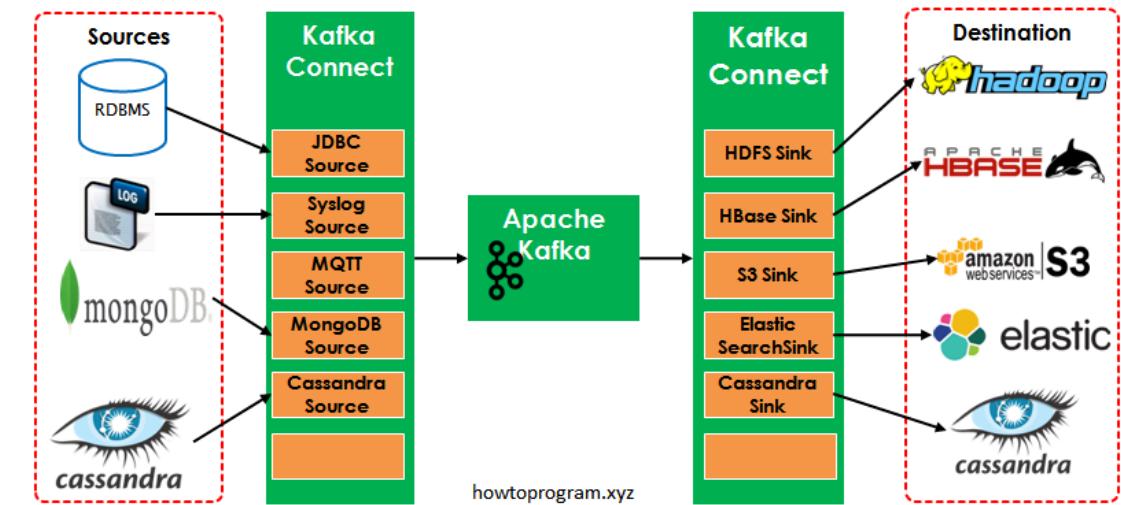
Ensemble de consommateurs de la même entité pour lire un topic

- Partage de la charge (*1 partition = 1 consumer*)
- Nécessite de faire attention au surdimensionnement de consumer



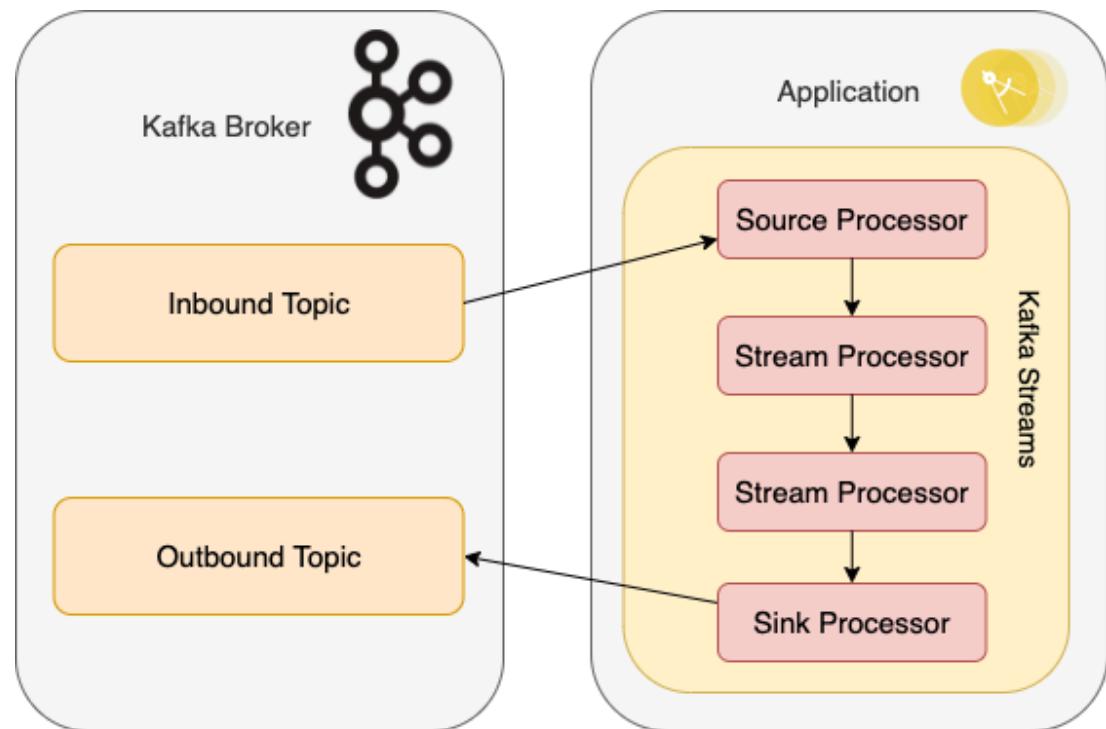
Kafka Connect

- Simplifie la connexion de Kafka avec des systèmes externes (bases de données, systèmes de fichiers, etc.)
- Prend en charge des **Connectors** pour les sources et les destinations (*sink*)
- Gère la scalabilité et la tolérance aux pannes



Kafka Streams

- Bibliothèque de traitement de flux (Java)
- Permet le traitement de données en continu (transformations, agrégations et jointures)
- Intégré avec Kafka pour une faible latence et une tolérance aux pannes élevée



Kafka Bridge

- API qui permet l'accès en tant que producer et consumer à Kafka via le protocole HTTP
- Simplifie l'accès pour des applications ou microservices

Exemple

```
curl http://kafka:8080/topics
#
[
  {
    "topic": "bridge-quickstart-topic",
    "key": "my-key",
    "value": "sales-lead-0001",
    "partition": 0,
    "offset": 0
  },
  #
]
```

