Hadoop

m2iformation.fr

Sommaire

- Objectifs pédagogiques
- Importance pour le cursus
- Introduction et HDFS
- MapReduce de base
- MapReduce avancé
- Workflow et intégration

Objectifs pédagogiques

Objectifs pédagogiques

- Comprendre l'architecture Hadoop et le fonctionnement de HDFS
- Développer des traitements MapReduce
- Interagir avec le FileSystem Hadoop (lecture, écriture, permissions)
- Exécuter et monitorer des jobs sur le cluster
- Préparer à l'usage de frameworks modernes s'appuyant sur HDFS (Hive, Spark)

Importance pour le cursus

Importance pour le cursus

Même si les technologies comme Spark ou BigQuery ont supplanté Hadoop dans bien des contextes, ce module permet de comprendre la genèse des traitements distribués et de manipuler des volumes massifs de données sur HDFS avec MapReduce.

Le Cloud Data Engineer doit :

- Comprendre les fondements du traitement distribué pour tirer parti d'outils comme Spark ou Dataflow
- Savoir manipuler un cluster Hadoop en contexte d'entreprise ou d'architecture hybride on-premise + cloud
- Gérer efficacement des volumes massifs de données non structurées
- Préparer et structurer les données dans HDFS avant de les traiter dans des pipelines cloud

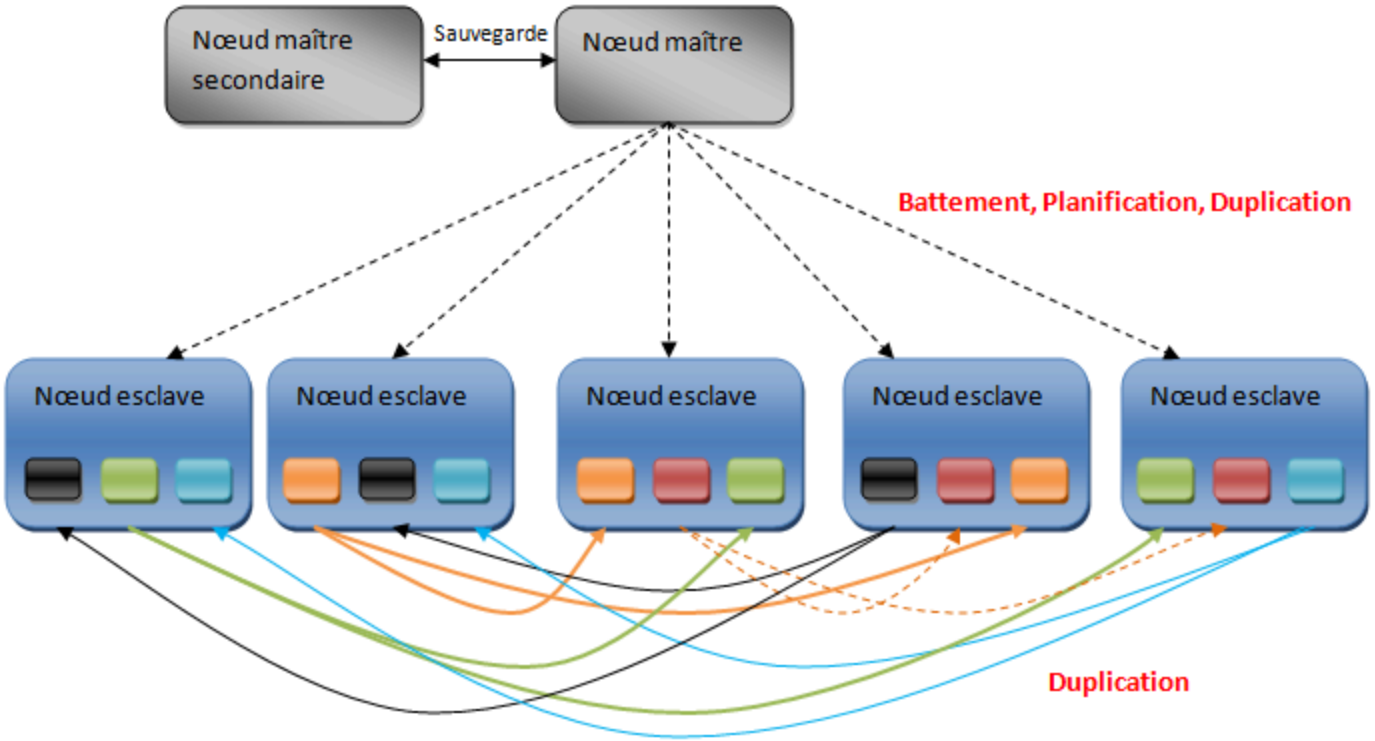
TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction et HDFS

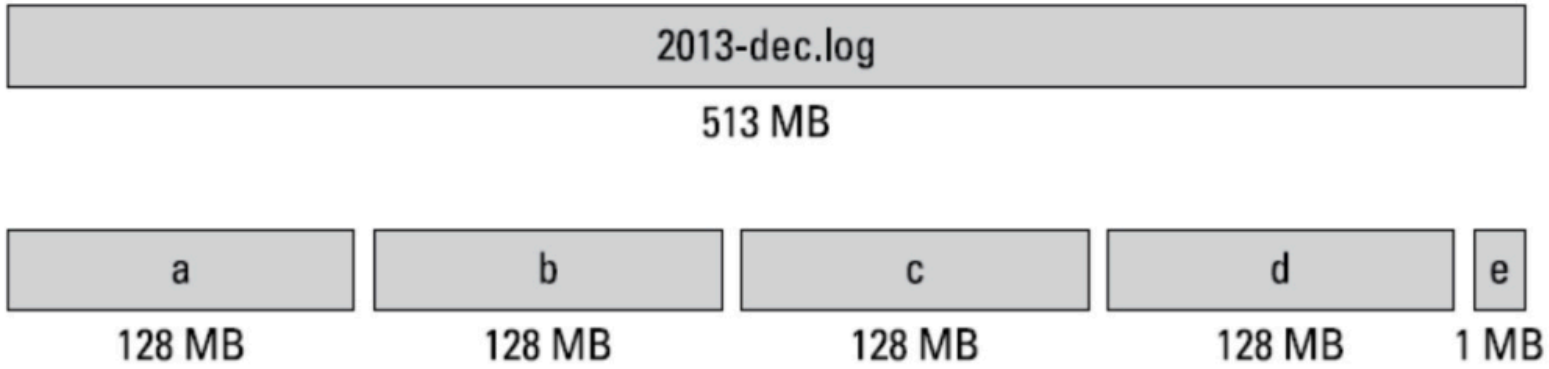
Introduction

- Hadoop est un framework open-source en Java pour applications distribuées et scalables, tolérant les pannes matérielles
- Modules principaux :
  - **Hadoop Common** : bibliothèques partagées.
  - **HDFS** (Hadoop Distributed File System) : système de fichiers distribué
  - **YARN** : gestionnaire de ressources et ordonnanceur
  - **MapReduce** : modèle de traitement de données massives

Architecture maître / esclave



Architecture maître / esclave



- Les fichiers sont divisés en plusieurs blocs de 128, 256, 512 Mo… 1 Go.
- La liste des blocs disponibles est remontée toutes les 6 heures (**BlockReport**).
- L'état du DataNode est remonté toutes les 3 secondes (**heartbeat**).

Architecture maître / esclave

- **NameNode** : nœud maître, gère l'arborescence, les métadonnées, la localisation des blocs. Unique dans le cluster, mais un Second NameNode prend en charge les checkpoints pour assurer la résilience
- **DataNode** : nœud esclave, stocke et restitue les blocs, envoie périodiquement des « heartbeats » au NameNode, réplique selon la politique définie
- **Topologie HDFS** : un seul NameNode, plusieurs DataNodes

Installation

Pré-requis système

- **OS** : Linux (Ubuntu/Debian/CentOS) ou macOS (Windows possible via WSL2).
- **Java** : Hadoop est écrit en Java, JDK 8 ou 11 recommandé.
- **SSH** : utilisé pour la communication entre nœuds (même en mode pseudo-distribué, SSH est requis).
- **Variables d'environnement** : `JAVA_HOME`, `HADOOP_HOME`, `PATH`.

Mise en place dans WSL

1. Mise à jour système

```
sudo apt update && sudo apt upgrade -y
```

2. Installer Java

```
sudo apt install openjdk-11-jdk -y
java -version
```

Mise en place dans WSL

- 3. Télécharger Hadoop

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.4.0/hadoop-3.4.0.tar.gz
tar -xvzf hadoop-3.4.0.tar.gz
sudo mv hadoop-3.4.0 /usr/local/hadoop
```

- 4. Configuration des variables d'environnement

```
nano ~/.bashrc
# Ajouter dans le fichier
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

Mise en place dans WSL

- Recharger le fichier :

```
# Recharger
source ~/.bashrc
```

- 5. Configuration minimale (mode pseudo-distribué)

```
nano $HADOOP_CONF_DIR/hadoop-env.sh
# Ajouter :
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Mise en place dans WSL

```
nano $HADOOP_CONF_DIR/core-site.xml
# Ajouter :
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Mise en place dans WSL

```
nano $HADOOP_CONF_DIR/hdfs-site.xml
# Ajouter :
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///usr/local/hadoop/data/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///usr/local/hadoop/data/datanode</value>
  </property>
</configuration>
```

Mise en place dans WSL

- Activer **YARN** :

```
nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

Mise en place dans WSL

- Créer les répertoires :

```
mkdir -p /usr/local/hadoop/data/namenode
mkdir -p /usr/local/hadoop/data/datanode
```

- 6. Préparation du système SSH

```
sudo apt install openssh-server -y
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
# Tester
ssh localhost
```

Démarrer les services Hadoop

1. Démarrer les services :

```
start-dfs.sh
start-yarn.sh
```

2. Vérifier :

```
jps
```

- Doit afficher : `NameNode`, `DataNode`, `ResourceManager`, `NodeManager`.

Les commandes de base

```
hdfs dfs -ls /chemin                    # lister le contenu
hdfs dfs -mkdir /chemin                 # créer un répertoire
hdfs dfs -put local.txt /chemin         # copier un fichier local vers HDFS
hdfs dfs -get /chemin/fichier local     # copier depuis HDFS vers local
hdfs dfs -chmod 755 /chemin/fichier     # modifier les permissions
```

Exercice

- Créer un dossier **logs** à l'intérieur du système de fichiers Hadoop
- Afficher le contenu du dossier **logs**
- Envoyer le fichier **access_log.txt** dans le dossier **logs** du HDFS
- Afficher le contenu du fichier (**les 50 premières lignes**)
- Supprimer le fichier **access_log.txt**

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

MapReduce

Modèle MapReduce

MapReduce est un modèle de programmation distribué. Il repose sur deux fonctions principales :

- **Map**
  - Reçoit en entrée une paire *(clé, valeur)*.
  - Produit en sortie **zéro, une ou plusieurs nouvelles paires (clé, valeur)**.

Modèle MapReduce

- **Reduce**
  - Regroupe toutes les valeurs associées à une même clé.
  - Applique une opération de réduction (agrégation, somme, moyenne, comptage, etc.).
  - Produit en sortie une paire *(clé, valeur)* finale.

Exemple : comptage de mots

- **Map** : lit une ligne de texte et émet `(mot, 1)`.
- **Sort** : regroupe tous les `(mot, 1)` par mot.
- **Reduce** : additionne les `1` pour chaque mot et émet `(mot, total)`.

Exercice

Base de travail : `access_log.txt`

1. Compter les requêtes par **IP**.
2. Compter les **erreurs** (4xx, 5xx) et le total d'erreurs.
3. Détecter les tentatives de **login WordPress** : `POST /wp-login.php` par IP.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

MapReduce Avancé

Combiner

- Fonction intermédiaire **optionnelle** exécutée **après le Mapper** mais avant le **Sort**.
- **Rôle** : réduire le volume de données transférées entre Mapper et Reducer.
- S'apparente à un mini-reducer local.
- **Limitation** :
  - Le combiner doit être associatif et commutatif (ex : somme, min, max).
  - Ne garantit pas d'être exécuté (**Hadoop peut l'ignorer**).

Partitioner

- Contrôle la manière dont les paires **clé/valeur** issues du Mapper sont **distribuées** vers les différents **reducers**.
- Par défaut : **HashPartitioner** (clé hachée modulo nombre de reducers).
- **Utilité** :
  - **Garantir** que certaines clés aillent vers le **même** reducer.
  - Répartition **équilibrée** de la charge.
  - **Optimisation** des traitements spécifiques (par ex. regrouper les produits d'une même catégorie).

Grouping Comparator

- Définit **comment** les clés sont **regroupées** lors de la phase de **réduction**.
- **Différent** du sorting comparator (qui trie les clés).
- **Exemple** : si les clés sont (user_id, timestamp), on peut définir un **GroupingComparator** qui ne compare que **user_id**, ce qui permet de grouper toutes les sessions d'un utilisateur dans un seul reduce call.

Optimisations

- Combiner : **réduire** le trafic réseau (shuffle).
- Tri personnalisé : via **SortingComparator** : utile pour forcer l'ordre **temporel**, **alphabétique** ou **numérique** des clés avant entrée dans le Reducer.
- **Usage mémoire** :
  - Utiliser des structures **légères** pour les clés/valeurs.
  - Appliquer **combiner** au maximum pour éviter les **transferts inutiles**.

Quelques propriété

Propriété	Valeur par défaut (≈)	Impact principal
mapreduce.map.memory.mb	1024 MB	Mémoire allouée à chaque Mapper.
mapreduce.reduce.memory.mb	1024 MB	Mémoire allouée à chaque Reducer.
mapreduce.map.java.opts	`-Xmx819m`	Options JVM des Mappers (limiter OOM).
mapreduce.reduce.java.opts	`-Xmx819m`	Options JVM des Reducers.
mapreduce.job.reduces	1	Nombre de Reducers du job.
mapreduce.task.io.sort.mb	100 MB	Taille du buffer de tri en mémoire côté Mapper (impact sur shuffle).

Quelques propriété

Propriété	Valeur par défaut (≈)	Impact principal
mapreduce.task.io.sort.factor	10	Nombre max de segments fusionnés en une passe.
mapreduce.reduce.shuffle.parallelcopies	5	Threads simultanés pour télécharger les outputs Mapper.
mapreduce.map.output.compress	false	Compression des sorties Mapper pour réduire le shuffle.
mapreduce.output.fileoutputformat.compress	false	Compression de la sortie finale.

Exemple

Documentation officiel : [mapred-default.xml]

- **Cas** :
  - Fichier d'entrée : 200 Go CSV.
  - Cluster : 10 nœuds, 8 Go RAM par nœud.
  - Problème : sort très lent, nombreuses erreurs mémoire sur les Reducers.

Commande

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -D mapreduce.map.memory.mb=2048 \
  -D mapreduce.reduce.memory.mb=4096 \
  -D mapreduce.task.io.sort.mb=512 \
  -D mapreduce.map.output.compress=true \
```

- `-D mapreduce.map.memory.mb=2048` : réserve 2 Go aux Mapper.
- `-D mapreduce.reduce.memory.mb=4096` : réserve 4 Go aux Reducer.
- `-D mapreduce.task.io.sort.mb=512` : augmente le buffer mémoire pour trier les données Mapper avant sort.
- `-D mapreduce.map.output.compress=true` : compresse les données Mapper envoyées au Reducer, réduisant le trafic réseau.

Exercice

1. Mapper :
    - Produits → sortie 1 (calcul des ventes par produit).
    - Utilisateurs → sortie 2 (analyse des sessions).
2. Utiliser un combiner pour réduire le poids réseau.
3. Implémenter un partitioner custom (par ex. tous les produits vers reducers 0, les utilisateurs vers reducers 1).
4. Implémenter un grouping comparator pour grouper toutes les transactions par utilisateur.

Monitoring et chaîne de traitement

Monitorer un job hadoop

- **Logs MapReduce** : chaque tâche génère des logs accessibles via la commande `yarn logs`.
- **Types de logs** :
  - stdout / stderr : sortie standard et erreurs des mappeurs et reduceurs.
  - syslog : informations système, exceptions Java/Python.
  - configuration logs : paramètres utilisés (mémoire, splits...).

Interfaces Web

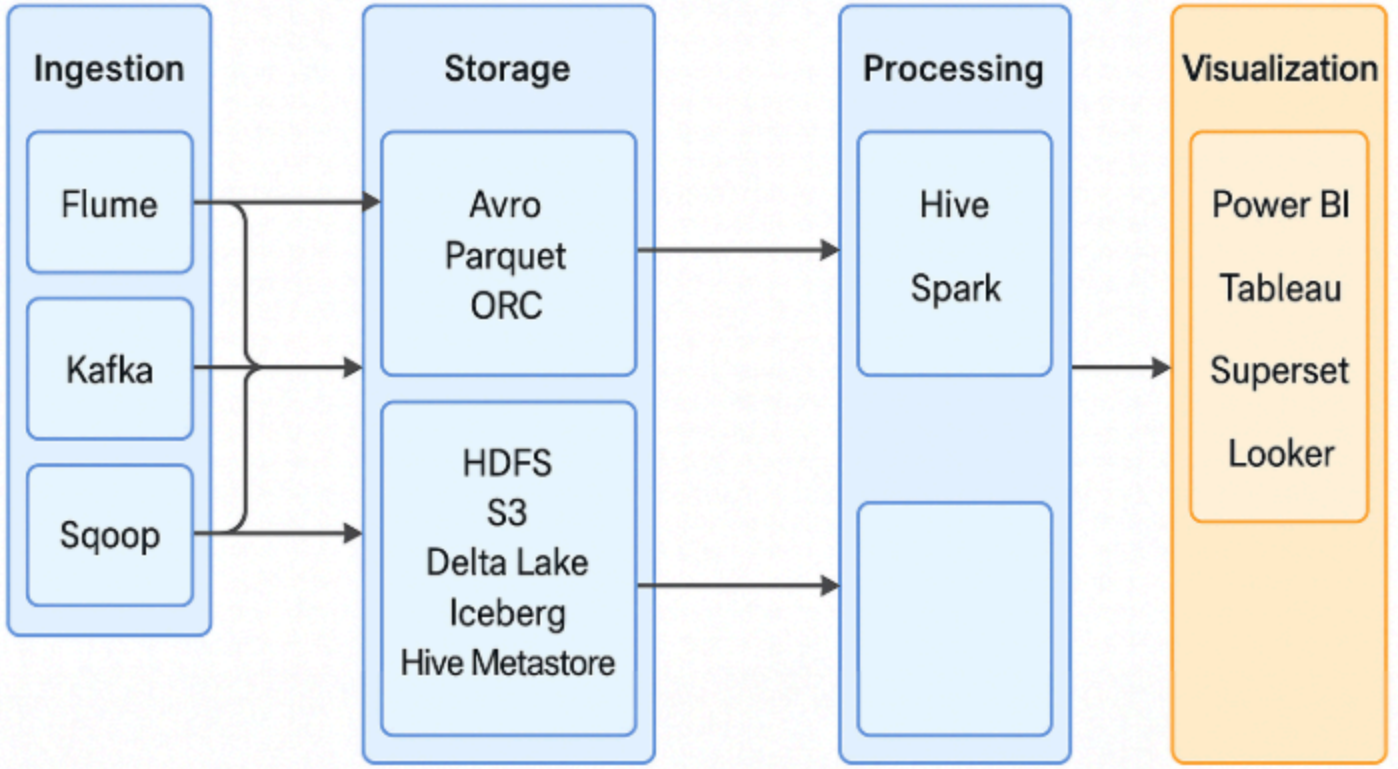
- **ResourceManager** Web UI (port **8088**) : tableau de bord des applications YARN.
  - Suivi des jobs (état, temps d'exécution, ressources consommées).
  - Monitoring en temps réel.
- **NodeManager** Web UI (port **8042**) : état d'un nœud, containers lancés, ressources disponibles.

YARN (Yet Another Resource Negotiator)

- **Rôle** : planificateur et gestionnaire des ressources dans Hadoop.
- **Composants** :
  - **ResourceManager** : répartit les ressources entre applications.
  - **NodeManager** : gère l'exécution des conteneurs sur chaque nœud.
- **Monitoring** :

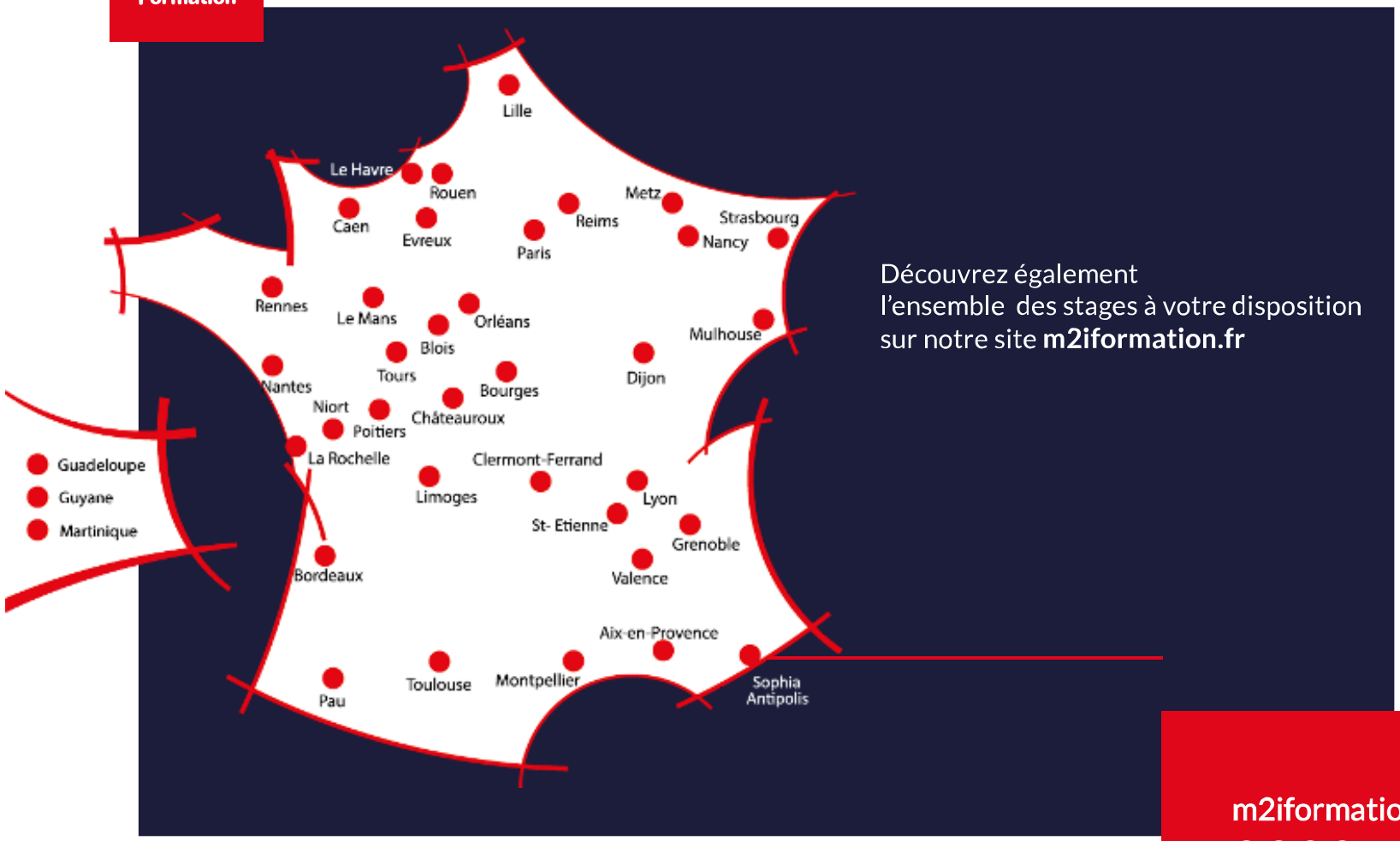
```
yarn application -list
yarn application -status <app_id>
```

Chaîne de traitement



Étapes principales

- **Ingestion** : collecte des données depuis des sources variées (logs applicatifs, csv, bases SQL, API).
- **Stockage** : persistance dans HDFS, Hive, HBase.
- **Traitement** : MapReduce, Spark.
- **Visualisation** : Export vers bases analytiques (**PowerBI**).



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr