Spark

15/12/2025 au 17/12/2025

plb

2025

Programme jour 1

- Introduction à HADOOP
- Les composants de HADOOP
- Le système de fichier HDFS
- Introduction à spark
- Les composants de spark
- Api RDD
- Les transformations
- Les actions
- Fonctionnement interne
- RDD Key Value
- Utilisation des filtres et flatmap

Programme jour 2

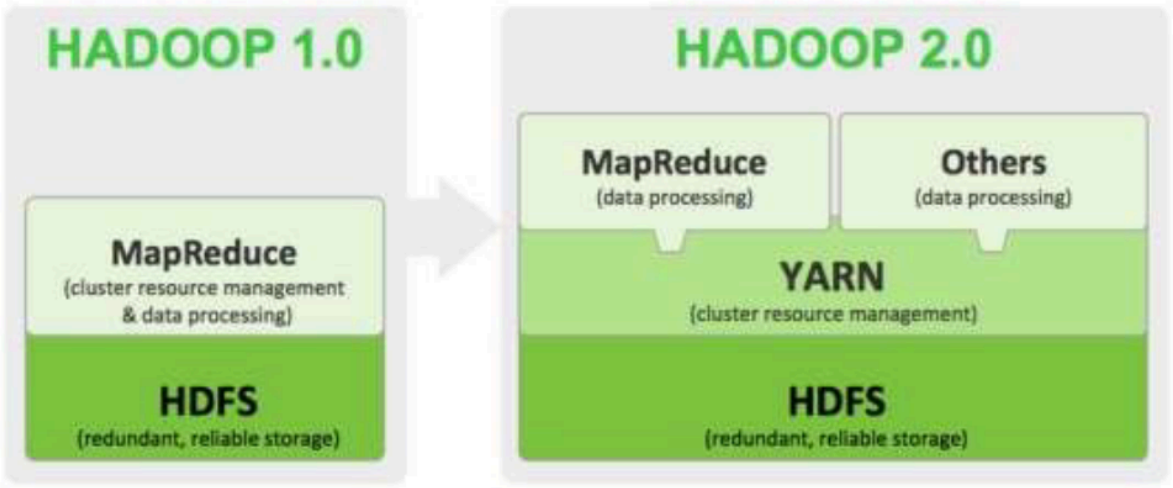
- Principe de SparkSQL
- Utilisation de SparkSQL
- Utilisation des DataSets et DataFrames
- Les fonctions udf
- Utilisation des Broadcast variables.
- Utilisation des Accumulators.
- Persistance dans Spark.
- Introduction à Spark Streaming
- Utilisation de stream API
- Utilisation des Structured Streaming

Programme jours 3

- Utilisation de spark submit.
- Utilisation des API Rest de Spark
- Le partitionning
- Le fonctionnement de Spark avec Yarn
- Bonnes pratiques sur un cluster Spark
- Etude de cas

Définition hadoop

- HDFS : Système de fichier distribué.
- MapReduce : Taches de traitement des données en lot.



Pourquoi hadoop ?

- GFS de Google, en opensource.
- Données/fichiers de grandes taille.
- Puissance de calcul distribuée entre plusieurs machines.
- Stockage distribué.
- Ensemble d’outils.
- Synchronisation entre les composants de cluster.

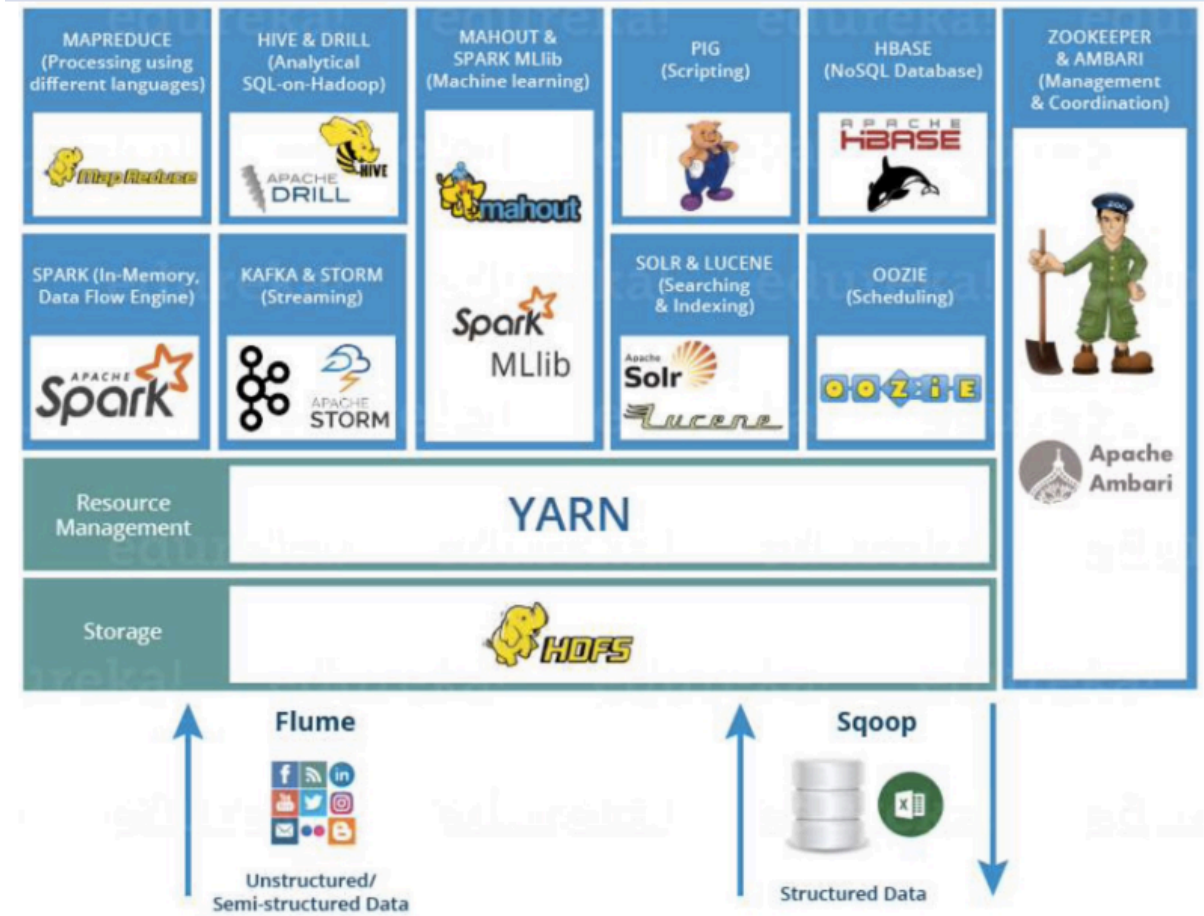
Ecosystème hadoop

- **Hbase**: une base de données NoSQL basée sur HDFS.
- **Hive**: une base de données relationnelle basée sur Hadoop, utilisable en SQL et accessible avec JDBC.
- **Mahout**: un outil logiciel basé sur Hadoop fournissant un framework et de nombreux algorithmes déjà implémentés pour effectuer du machine learning en se basant sur HDFS et MapReduces.
- **Pig**: un outil de scripting basé sur Hadoop permetant de manipuler aisément de grandes quantités de données avec un langage proche du Python ou Bash.

Ecosystème hadoop

- **Oozie**: une interface Web de gestion des jobs Hadoop pour les lancer et les planifier aisément en incluant les notions de dépendances de jobs à d'autres jobs.
- **Flume / Sqoop** : ingestion de données.
- **Solr / Lucene** : indexation et recherche.
- **Spark** : analyse des données.

Ecosystème hadoop

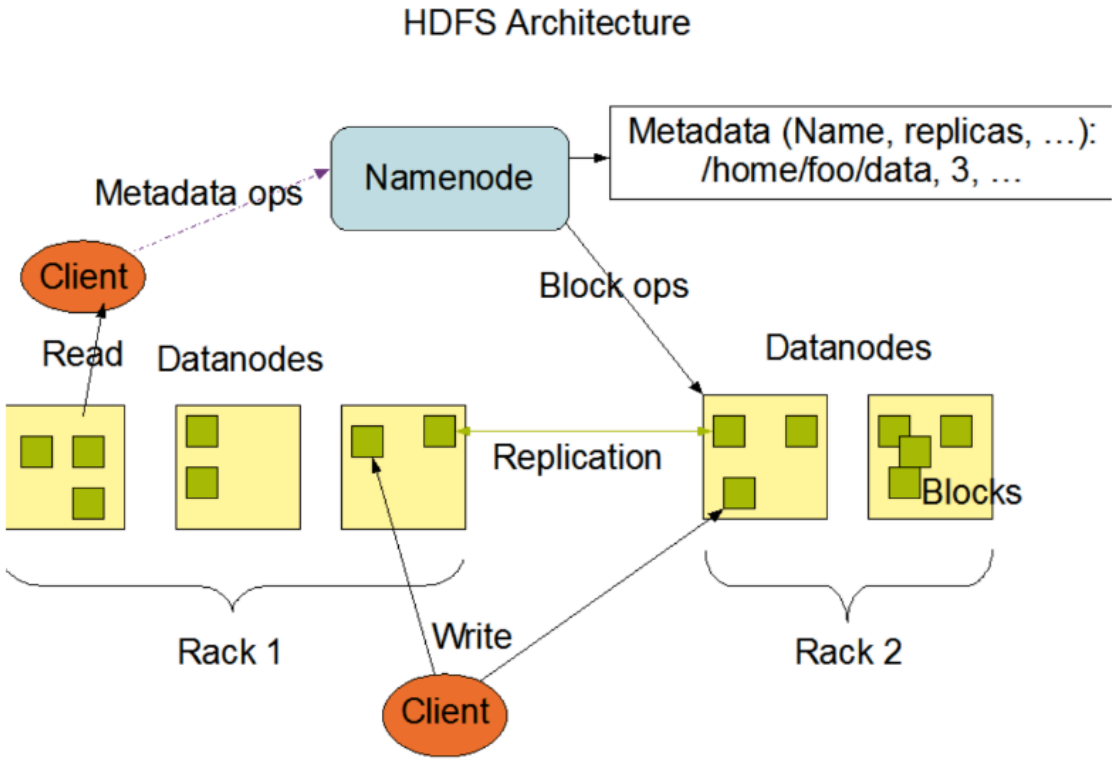


Hadoop installation

1. L’installaDon des différents composants (sur un ou plusieurs nœuds).
2. Utilisation d'un docker compose.

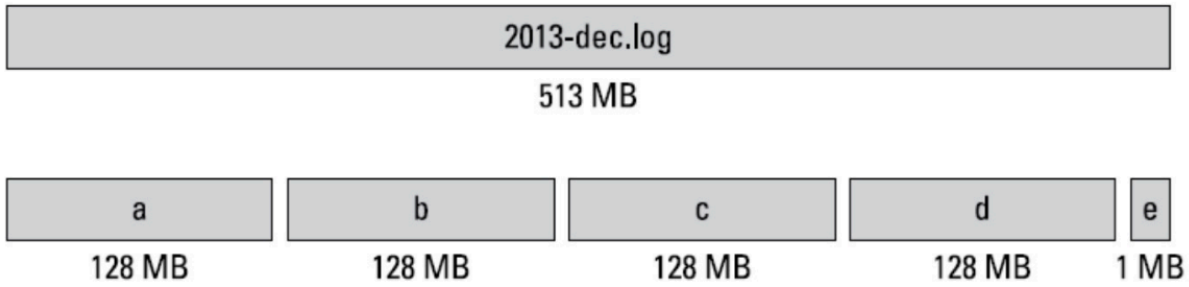
Hadoop HDFS

- Hadoop HDFS (Hadoop Distributed File System) est le mécanisme de gestion et de stockage distribué de fichiers d'Hadoop.

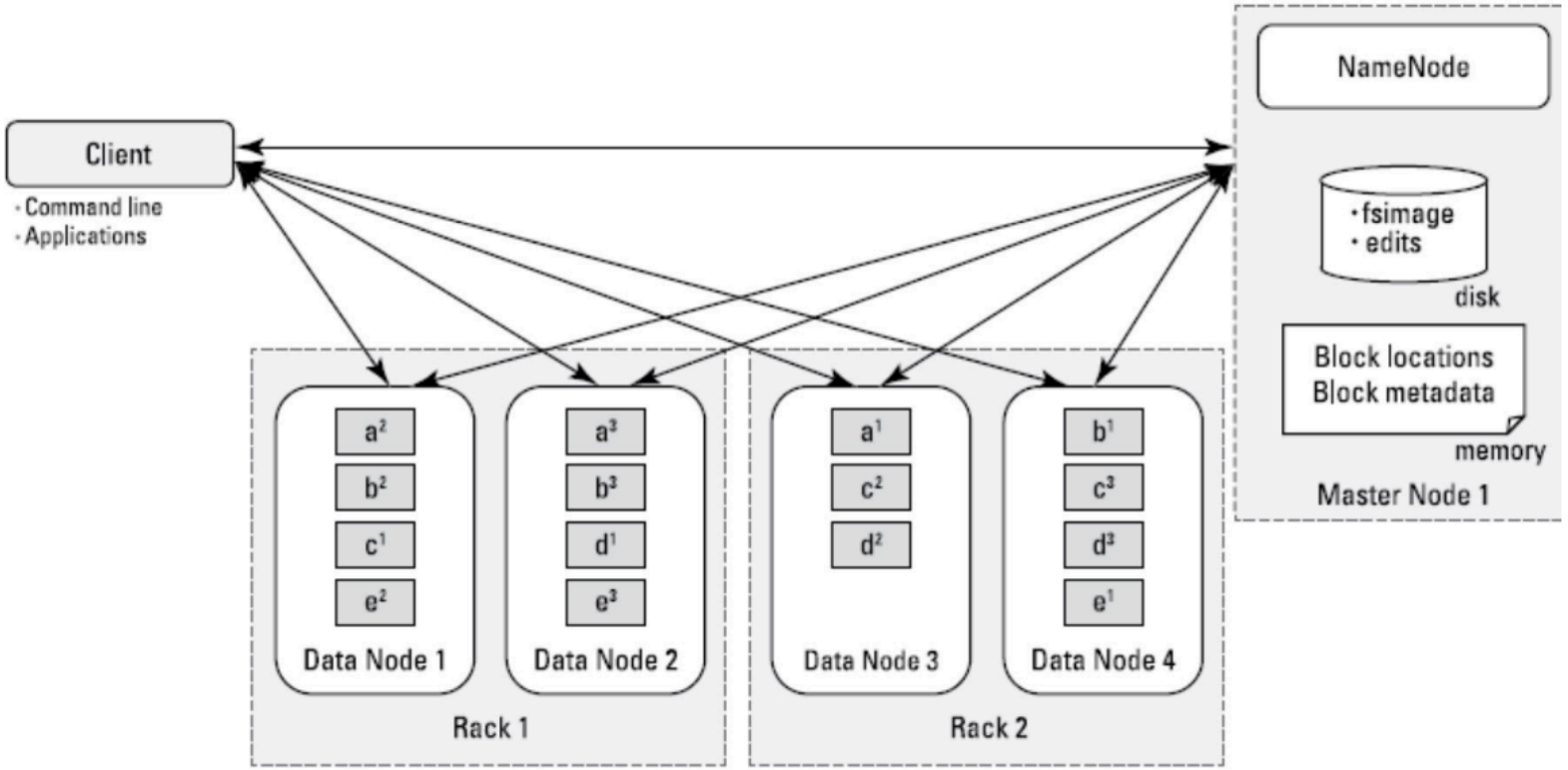


Fonctionnement HDFS

- Les fichiers sont divisés en plusieurs blocks de 128, 256, 512Mo …
- Les blocks sont répliqué et stockés dans les datanodes
- La liste des blocks disponible est remonté périodiquement au namenode = BlockReport (**chaque 6 heures**)
- L'état du datanode est remonté, au namenode=heartbeat (chaque **3 secondes**)



Fonctionnement HDFS



Lire et ecrire sur hdfs

- Il est possible d'interagir avec HDFS via :
  - **API** Java
  - Protocoles réseaux (**http, ftp, webDFS, httpfs ..**)
  - Protocole propriétaire (**Amazone S3 ..**)
  - Directement via la ligne de commande **hdfs dfs -file_cmd** ou **hadoop fs –file_cmd**
  - File-cmd sont **similaires** aux commandes **linux** (ls, mkdir ..)
  - La commande **hadoop dfs** est dépréciée

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
hdfs dfs –mv <src> <dest> # Déplacer les fichier de la source vers la destination
```

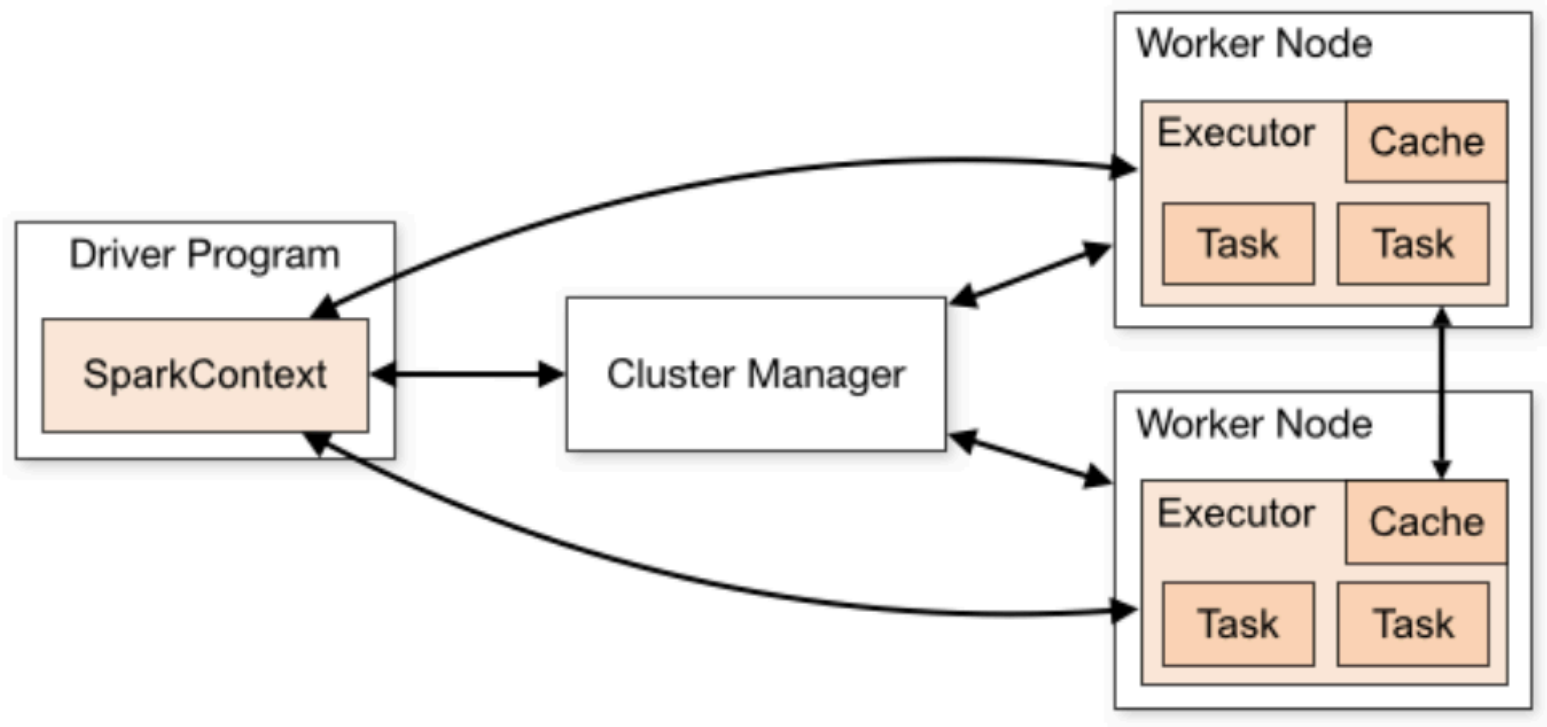
Exercice HDFS

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

Introduction Spark

- Spark est un moteur de traitement de données à grande échelle.
- Spark permet d'analyser et transformer une grande quantité de données.
- Spark permet l'exécution et l'analyse sur un cluster.
- Spark est 100 fois plus rapide que Hadoop MapReduce
- Spark utilise DAG Engine pour l'optimisation des workflows.
- Documentation : https://spark.apache.org/docs/latest/

Introduction Spark



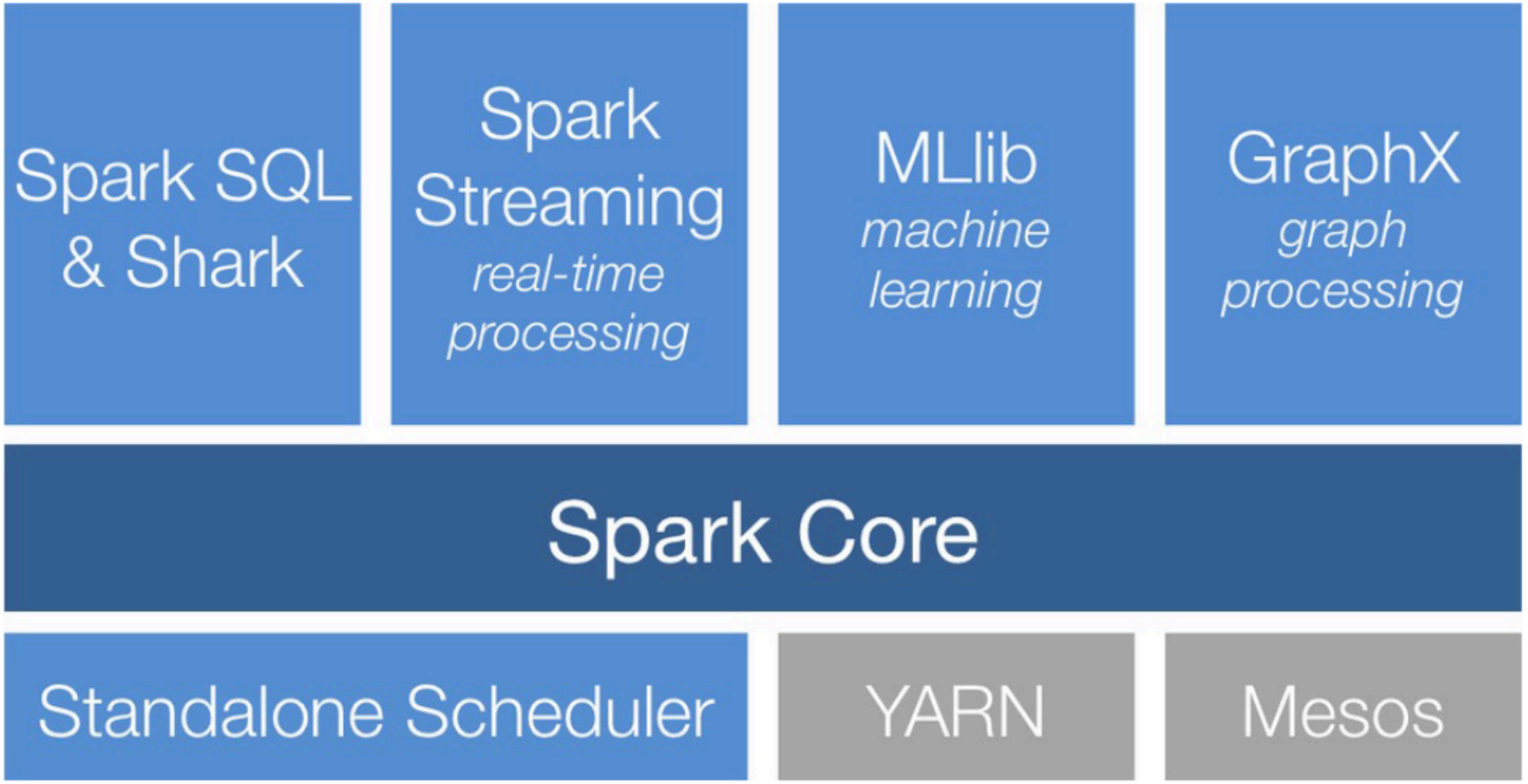
Introduction Spark

- Driver program est un script ou une application qui peut être en Java, scala, python.
- Cluster manager est l'api qui permet de gérer l'orchestration de notre cluster de nœuds.
- Chaque nœud contient un exécuteur.
- Chaque exécuteur possède son propre cache et la liste des tâches à exécuter.
- Les composants de l'architecture spark communiquent et se synchronisent entre eux.

Introduction Spark

- Spark est facile à apprendre.
- Spark supporte une multitude de langages de développement.
- Spark fournit des APIs bas niveau.
- Spark fournit également des Apis Haut niveau.

Les composants de spark



Les composants Spark

- Spark Core est le moteur d'exécution de Spark.
- Spark Streaming est une Api qui permet l'ingestion de données en temps réel.
- Spark SQL permet l'interaction avec Spark à l'aide de requête SQL.
- MLLib est la bibliothèque d'apprentissage automatique de Spark.
- GraphX est une Api qui permet de mettre en place et analyse de connexion entre plusieurs entités.

Nouveautés Spark4

- **API `.plot()` native sur DataFrame** : permet des visualisations directes sans bibliothèques externes (utilise Plotly).
- **Spark Connect activé par défaut** : exécution distante des scripts PySpark avec un client léger.
- **Data Source API Python** : création de connecteurs batch/streaming en pur Python.
- **Support des UDTF en Python** : fonctions retournant des lignes multiples, avec typage flexible.

Spark Core - RDD (Resilient Distributed DataSet)

- RDD sont des lignes de données muables qui sont :
  - Resilient
  - Distributed
  - DataSet
- Les RDD sont créées par notre Driver Program.
- La création se fait à l'aide d'un context (SparkContext).
- Documentation RDD : RDD

Spark Core - RDD

- La création peut se faire à partir d'un fichier text.
  - File://, s3n://, hdfs://
- La création peut se faire également à partir d'une base de données
  - JDBC
  - Cassandra
  - ElasticSearch
  - JSON, csv…

Spark Core - Transformations

- Les opérations appliquées aux RDD sont des transformations.
- Chaque transformation donne une nouvelle RDD.
- Spark core offre une liste de transformations possibles :
  - Map
  - Flatmap
  - Filter
  - Distrinct
- Chaque transformation prend comme paramètre une fonction (Expression lambda).

Spark Core - Transformations

- Spark Context ne procède pas directement à l'exécution de la transformation.
- Spark attend l'appel au deuxième type d'opération exécutable sur RDD (action) pour démarrer la transformation.
- Le choix de l'action influence sur la façon d'exécution de la transformation.

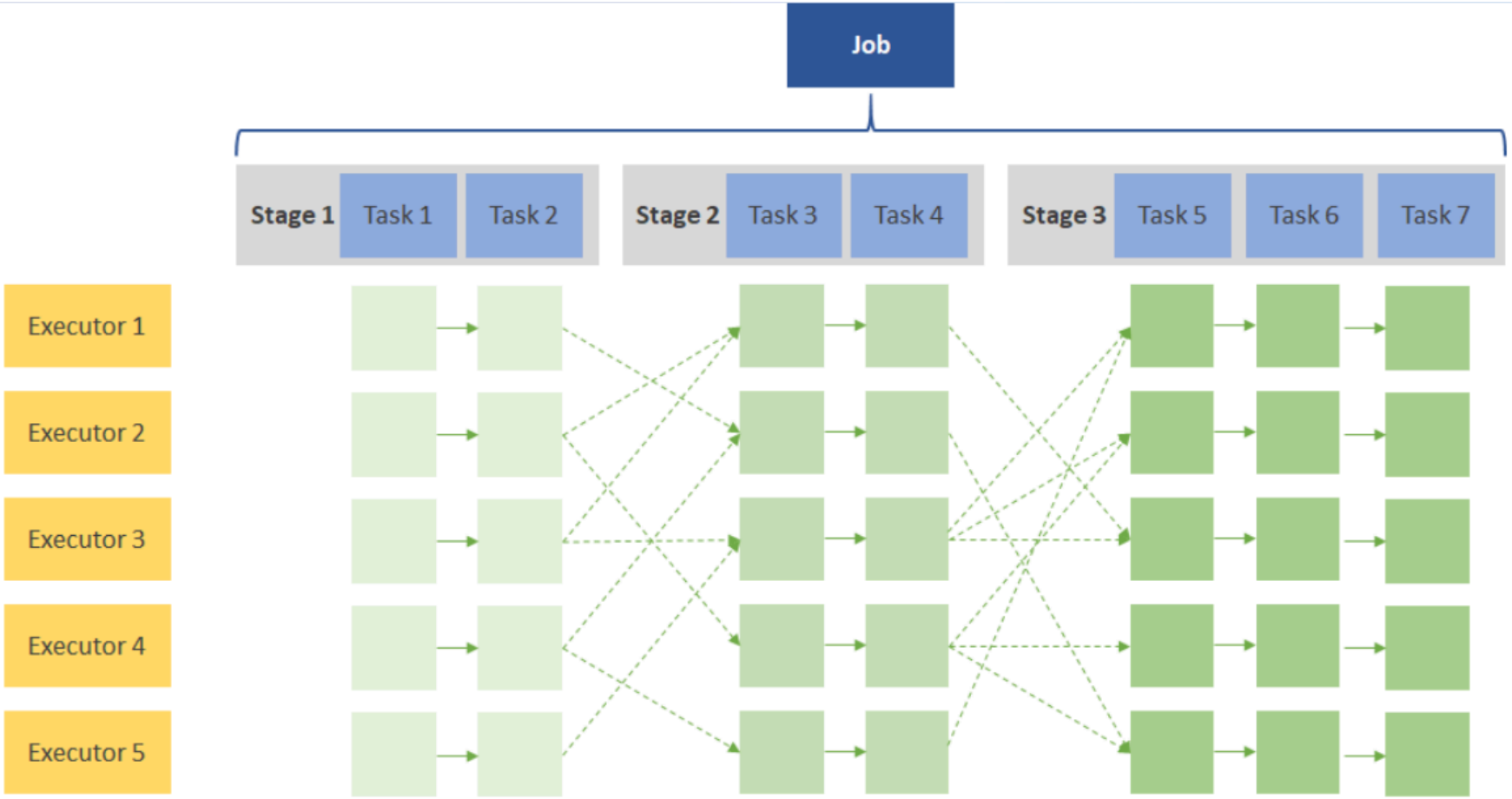
Spark Core - Actions

- L'action est l'opération à appliquer à notre RDD pour récupérer le résultat.
- Spark core offre une liste d'actions possibles.
  - Collect
  - Count
  - CountByValue
  - Reduce
- Une action peut avoir aucun ou plusieurs paramètres.

Spark Core - Rdd - exemple 1

- On utilisera un set de données de plus de 100 000 lignes de notes de films (fichier film.data).
- On souhaite connaitre le nombre de film par note et faire un tri.
- On commence par extraire à partir de chaque ligne la valeur qui nous intéresse.
- On compte le nombre d'éléments par valeur.
- On tri le résultat.

Spark Core - Fonctionnement interne



Spark Core - RDD key value

- RDD key value est une implémentation des données structurées sous format clé valeur.
- RDD key value offre des transformations et des actions en plus.
- Exemple:
  - Pour calculer le nombre moyen d'amis par âge dans un set de données qui contient (personnes avec nombre d'ami).
  - On commence par extraire les données dans une RDD (âge, ami).
  - On applique les transformations mapValues, reducesByKey
  - On applique une action de collect

Spark Core - RDD transformation filter

- Transformation Filter est une opération qui prend comme paramètre une fonction.
- La fonction paramètre est appliquée à chaque élément et renvoie un boolean.
- Le résultat est une nouvelle RDD avec des éléments dont le résultat est positif.
- **Exemple** : A partir d'un set de données météorologique fournit par deux stations météo, on souhaite conserver uniquement les températures minimums.

Spark Core - RDD flatmap

- La fonction map permet de convertir chaque ligne de notre set de données en ligne de RDD.
- Avec map chaque ligne est une row RDD.
- FlatMap permet de convertir chaque ligne de notre set de données en n'importe quel nombre de row RDD.
- **Exemple** :
  - A partir d'un fichier texte, on souhaite compter le nombre de mots.

Fonction de transformation

Transformations	Rôle
map(func)	applique une fonction à chacune des données pour créer un nouveau RDD
filter(func)	permet d'éliminer certaines données. Exemple : rdd.filter(lambda x : x % 2 != 0) permet d'éliminer les nombres pairs
flatMap(func)	similaire à map, mais chaque élément d'entrée peut être mappé à 0 ou plusieurs éléments de sortie (donc, func devrait retourner une liste plutôt qu'un seul élément)
distinct()	supprime les doublons
groupsByKey()	transforme des clés-valeurs (K,V) en (K,W) où W est un object itérable. Par exemple, (K,U) et (K,V) seront transformées en (K,[U,V])
reduceByKey(func)	applique une fonction de réduction aux valeurs de chaque clé. La fonction de réduction est appelée avec deux arguments

Fonction de transformation

Transformations	Rôle
sortByKey	utilisée pour trier le résultat par clé
join(rdd)	join(rdd) permet de réaliser une jointure avec 2 RDD Key Value, ce qui a le même sens que dans les bases de données relationnelles.À noter qu'il existe également des fonctions OuterJoin et fullOuterJoin. Les jointures sont réalisées sur la clé
reduce(func)	agrège les éléments de l'ensemble de données en utilisant une fonction func (qui prend deux arguments et en renvoie un) cela donnera une valeur finale
take(n)	retourne une liste avec les n premiers éléments du RDD
collect()	retourne toutes les données contenues dans le RDD sous la forme d'une liste.
Count()	retourne le nombre de données contenues dans le RDD

Exercice 1

- A partir d'un fichier csv qui contient sur chaque ligne (id client, id article, montant).
- Ecrire un driver programme qui permet de connaître le montant dépensé par chaque client.
- **Aide** : Séparer chaque ligne avec la délimitation « , ».
  - Créer une pair RDD avec id client et montant (transformation map).
  - Réduire par id client (transformation reduceByKey).
  - Récupérer une liste de résultats (action collect)

Spark SQL - RDD structurés

- Un RDD est une ”boî te” (typée) destinée à contenir n’importe quel document, sans aucun préjugé sur sa structure (ou son absence de structure)
- Cela rend le système très généraliste, mais empêche une manipulation fine des constituants des documents, comme par exemple le filtrage en fonction de la valeur d’un champ.
- C’est au programmeur de fournir la fonction effectuant le filtre. Spark propose (depuis la version 1.3, avec des améliorations en 1.6 puis 2.0) des RDD structurées dans lesquels les données sont sous forme tabulaire. Le schéma de ces données est connu.

Spark SQL - RDD structurés

- Ces structures, Datasets et Dataframes, sont assimilables à des tables relationnelles.
- Documentation SparkSQL : https://spark.apache.org/docs/latest/sql-getting-started.html

Spark SQL - Dataset et DataFrames

- La connaissance du schéma - et éventuellement de leur type - permet à Spark de proposer des opérations plus fines, et des optimisations inspirées des techniques d'évaluation de requêtes dans les systèmes relationnels.
- On se ramène à une implantation distribuée du langage SQL.
- En interne, un avantage important de la connaissance du schéma est d'éviter de recourir à la sérialisation des objets Java (opération effectuée dans le cas des RDD pour écrire sur disque et échanger des données en réseau).

Spark SQL - Dataset et DataFrames

- Ces RDDs structurés sont nommées :
  - Dataset quand le type des colonnes est connu
  - Dataframe quand ce n'est pas le cas (ensemble de Row)
- Un Dataframe n'est rien d'autre qu'un Dataset contenant des lignes de type Row dont le schéma précis n'est pas connu

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Spark SQL - Exercice 1

Spark- fonction udf

- Les UDF (**User Defined Functions**) permettent de créer une nouvelle colonne dans un dataframe qui sera le résultat d'un calcul pouvant utiliser les valeurs d'une (ou plusieurs) colonne(s) existante(s).
- Une **UDF** prend en argument :
  - un objet de type colonne
- Et retourne :
  - une valeur de type Type

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Spark udf - Exercice

Spark - Utilisation des broadcast variables

- Dans **Spark RDD**, **DataFrame** et **DataSet** les variables de diffusion sont des variables partagées en **lecture seule** qui sont mises en **cache** et disponibles sur tous les **nœuds** d'un **cluster**.
- Les variables de diffusion peuvent être **accessibles** ou **utilisées** par les tâches.
- Au lieu d'envoyer ces données avec chaque tâche, Spark **distribue** des variables de diffusion à la machine en utilisant des **algorithmes** de diffusion efficaces pour réduire les coûts de **communication**.

Spark - Utilisation des broadcast variables

- Exemple d'utilisation :
  - Le set de données film ne contient pas les **noms**, mais uniquement leur **Id**.
  - Si on souhaite **récupérer** le nom du film à parir d'un **deuxième** set de données qui contient à la fois **l'id et le nom**.
  - Au lieu de **charger** les données dans chaque tâche d'exécution et utiliser par exemple les **jointures SQL**, nous utiliserons une variable de diffusion pour mettre en **cache** sur chaque machine et les tâches utiliseront ces informations lors de l'exécution.

Spark - Utilisation des broadcast variables

- Les variables de diffusion sont utilisées de la même manière pour **RDD**, **DataFrame** et **Dataset**.
- Spark **décompose** le job en **stage** qui ont distribué les transformations et les actions sont exécutées avec dans le stage.
- Les stages **ultérieures** sont également **divisées** en tâches.
- Spark **diffuse** les données communes (**réutilisables**) nécessaires aux tâches de chaque stage.
- Les données **diffusées** sont mises en **cache** au format sérialisé et désérialisées avant l'**exécution** de chaque tâche.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Spark broadcast - Exercice

Spark - Utilisation des accumulators

- Les **accumulators** Spark sont des **variables partagées** qui ne sont "ajoutées" que par une opération associative et commutative et sont utilisées pour effectuer des compteurs (**similaire aux compteurs Map-reduce**) ou des opérations de sommes.
- Par défaut, Spark prend en charge la création accumulators de **n'importe quel type numérique** et offre la possibilité d'ajouter des types d'accumulateurs personnalisés.

Spark - Utilisation des accumulators

- Les programmeurs peuvent créer les accumulateurs suivants
  - accumulateurs nommés.
  - accumulateurs sans nom.
- L'action d'écriture de l'accumulator peut avoir lieu uniquement dans l'action.
- La récupération de la valeur de l'accumulator ne peut avoir lieu que dans le driver program

persistance dans spark

- Spark **Cache et Persist** sont des techniques d'**optimisation** dans **DataFrame/Dataset** pour des applications **Spark** itératives et **interactives** afin d'améliorer les performances des Jobs.

- En utilisant les méthodes **cache()** et **persist**(), Spark fournit un **mécanisme** d'optimisation pour stocker le calcul **intermédiaire** d'un **Spark DataFrame** afin qu'il puisse être **réutilisé** dans des actions **ultérieures**.

persistance dans spark

- Lorsque l’on conserve un ensemble de données, chaque **nœud** stocke ses données **partitionnées** en mémoire et les réutilisent dans d'autres **actions** sur cet ensemble de données.

- Les données persistantes de Spark sur les **nœuds** sont tolérantes aux pannes, ce qui signifie que si une partition d'un ensemble de données est **perdue**, elle sera automatiquement **recalculée** à l'aide des transformations d'origines qui l'ont créée.

Persistance dans spark : Avantage

- Réduction des **coûts**, les **calculs** Spark sont très **coûteux**, la **réutilisation** des **calculs** est utilisée pour les réduire.
- Gain de **temps**, La **réutilisation** de calculs répétés permet de gagner un temps non **négligable**.
- Temps d'**exécution**, économise le temps d'**exécution** du travail, nous pouvons aussi effectuer plus de travaux sur le même **cluster**.

Persistance dans spark - Mode d'utilisation

- Spark dataframe et dataSet utilise, par défaut, une persistance **MEMORY_AND_DISK**.
- RDD utilise, par défaut, une persistance **MEMORY_ONLY**.
- Les autres modes sont :
  - MEMORY_ONLY_SER
  - MEMORY_ONLY_2
  - MEMORY_ONLY_SER_2
  - MEMORY_AND_DISK
  - MEMORY_AND_DISK_SER

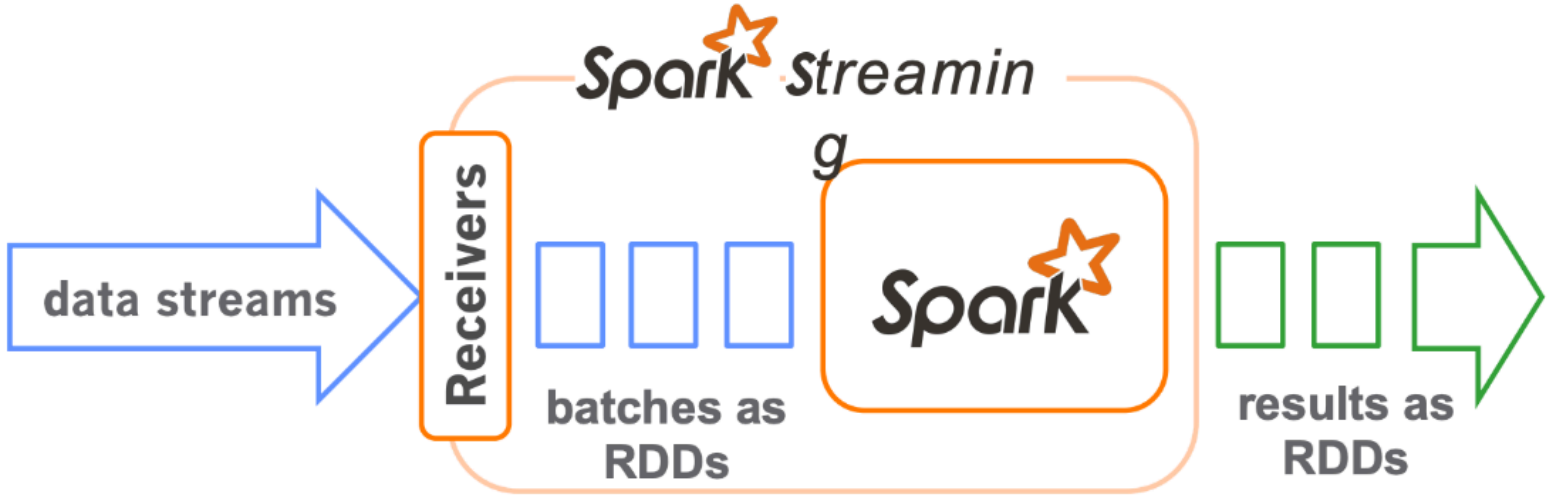
Spark Streaming

Spark Streaming

- **Spark Streaming** est une api Spark qui permet de :
  - **Recevoir des flux** de données à partir de sources d'entrée.
  - **Traiter les données** dans un cluster
  - **Envoyer les données** vers une nouvelle source.

Spark Streaming - fonctionnement

- Découpe les flux de données en lots de quelques secondes
- Traite chaque lot de données comme des RDD à l'aide des opérations.
- Les résultats traités sont envoyés par lots.



Spark streaming - Input Sources

- Spark Streaming **ingère** des données provenant de **différents types de sources** d'entrée pour un traitement en temps réel.
- **Rate** : il **générera automatiquement** des données comprenant 2 colonnes **timestamp** et **value** généralement utilisé à des fins de test.
- **Socket** : cette source de données **écoutera** le socket spécifié et **ingérera** toutes les données dans **Spark Streaming**.
- **Fichier** : cette source écoutera un **répertoire** particulier en tant que **flux de données**.
- **Kafka** : cette source lira les données d'**Apache Kafka**.

Spark Streaming - Output sink

- Dans **Spark Streaming**, les output sinks **stockent** les résultats dans un **stockage externe**.
- **Console** : affiche le contenu du DataFrame sur la console.
- **File**: stocke le contenu d'un **DataFrame** dans un fichier d'un répertoire. Les formats de fichiers pris en charge sont **csv, json, orc et parquet**.
- **Kafka** : publie des données dans un sujet Kafka.
- **Foreach sink** : s'applique à **chaque ligne** d'un DataFrame et peut être utilisé lors de l'**écriture** d'une logique **personnalisée** pour stocker des données

Spark streaming - Exercice

Utilisation de spark-submit

- Pour démarrer une application spark sur un cluster, il faut :
  - S'assurer de ne pas avoir de chemin vers des ressources locales
  - Créer un jar file avec inteliJ par exemple.
  - Exécuter notre application avec spark-submit
  - spark-submit -class <Main class> --jar <dependencies> --files <file> <jarfile>
- Pour **générer** un package avec les **dépendances**, on peut uiliser : **Maven**, **Gradle**, **SBT** pour scala

Spark Api Rest

- **Spark** nous offre la possibilité d'**interagir** avec notre cluster à l'aide d'une **API REST**.
- Pour **utiliser** l'api REST, l'option rest doit être **activée** dans la **configuration** spark.
- **spark.master.rest.enabled true**
- Après **redémarrage** des services, API REST est **actif**.

Spark API - Request création d'un job

```
curl -X POST http://192.168.1.1:6066/v1/submissions/create --header "Content-Type:application/json;charset=UTF-8" --data '{
  "appResource": "/home/hduser/sparkbatchapp.jar",
  "sparkProperties": {
    "spark.executor.memory": "8g",
    "spark.master": "spark://192.168.1.1:7077",
    "spark.driver.memory": "8g",
    "spark.driver.cores": "2",
    "spark.eventLog.enabled": "false",
    "spark.app.name": "Spark REST API - PI",
    "spark.submit.deployMode": "cluster",
    "spark.jars": "/home/user/spark-examples_versionxx.jar",
    "spark.driver.supervise": "true"
  },
  "clientSparkVersion": "2.4.0",
  "mainClass": "org.apache.spark.examples.SparkPi",
  "environmentVariables": {
    "SPARK_ENV_LOADED": "1"
  },
  "action": "CreateSubmissionRequest",
  "appArgs": [
    "80"
  ]
}'
```

Spark API - Réponse création d'un job

```
{
  "action" : "CreateSubmissionResponse",
  "message" : "Driver successfully submitted as driver-20200923223841-0001",
  "serverSparkVersion" : "2.4.0",
  "submissionId" : "driver-20200923223841-0001",
  "success" : true
}
```

Spark Api - Récupérer l'état d'un job

```
curl http://192.168.1.1:6066/v1/submissions/status/driver-20200923223841-0001
```

```
{
  "action" : "SubmissionStatusResponse",
  "driverState" : "FINISHED",
  "serverSparkVersion" : "2.4.0",
  "submissionId" : "driver-20200923223841-0001",
  "success" : true,
  "workerHostPort" : "192.168.1.1:38451",
  "workerId" : "worker-20200923223841-192.168.1.2-34469"
}
```

Spark Api - Arrêter un job

```
curl -X POST http://192.168.1.1:6066/v1/submissions/kill/driver-20200923223841-0001
```

```
{
  "action" : "KillSubmissionResponse",
  "message" : "Kill request for driver-20200923223841-0001 submitted",
  "serverSparkVersion" : "2.4.0",
  "submissionId" : "driver-20200923223841-0001",
  "success" : true
}
```

Spark - Partitionning

- Spark n'étant pas totalement **magique**, il faut toujours avoir une réflexion sur la façon de **partitionner** nos données.
- Par exemple, dans le cas d'une **jointure** d'un dataSet avec 1000000 sur **lui-même**, sans **partition**, spark **refusera** l'exécution de nos **transformations** et **actions**.
- Nous pouvons **utiliser** les fonctions .**repartition**() sur un dataframe ou **partitionBy**() sur un RDD, avant l'exécution d'un nombre **très élevé** d'opérations.
- **Join**(), **groupWith**(), **groupByKey**(), **reduceByKey**(), **combineByKey**().

Spark Partitionning - Choix du nombre

- Un nombre très petit de **partitions**, nous permets de tirer **avantage** de la puissance de calcule de notre cluster.
- Un nombre **trop** élevé de partition peut avoir des soucis de **synchronisation** des données entre les **partitions**.
- Le nombre de **partitions** peut être au **minimum** égal au **nombre de cores** ou d'executor disponibles.
- Généralement pour un nombre d'opérations très **élevé**, on peut commencer à **100 partitions**

Spark cluster - bonnes pratiques

- **Utilisation** des **DataSet/DataFrame** au lieu des **RDD**
- **coalesce**() et **repartition**()
- **mapPartitions**() et **map**()
- **UDF** en dernier recours
- Mettre en place la **persistance** et le **cache**.
- Réduire au **max** les opérations qui nécessite le **mélange des données**.
- **Désactiver** le Debug en prod.

Spark - MLlib

- **MLlib** est une librairie de **machine learning** qui contient tous les **algoritmes** d'apprentissage classiques.
- **Traite** la classification, la **régression**, le **clustering**, le **filtrage collaboratif**, la **réductions** de dimensions.
- Supporte **Scala**, **java** et **python**

Algorithmes de MLlib

- **Statistics** : Description, correlation
- **Clustering** : **K-means.**
- **Collaborative** filtering: **ALS**
- **Classification** : **SVMs**, naive bayes, decision tree.
- **Régression**: Régression linéaire, logistic regression
- **Dimensionality**: SVD, PCA

Pourquoi MLlib

- **Facilement** scalable.
- **Grande performance**.
- Large **documentation** et **API** facilement utilisable.
- **Facilement** maintenable.

MLlib - Types de données

- **MLlib** propose des vecteurs locaux et matrices locales et également des matrices distribuées.
- **Dense vector** : Un vecteur d'un seul tableur de valeur.
- **Sparse vector** : Deux tableaux sont mis en correspondance, un pour les indices et un deuxième pour les valeurs.
- **LabelPoint** : Type de données spécifique aux algorithmes d'apprentissage et associé à un label.

MLlib - Types de données

- **Rating** : Type de données spécifique aux systèmes de recommandation et donc à l'algorithme de factorisation ALS.
- **Model** : La classe model est le résultat d'un **algorithme d'apprentissage** qui dispose de la fonction **predict()** pour appliquer le modèle à une nouvelle observation ou à une **RDD**

Etude de cas

- On souhaite mettre en place une **pipeline** d'**ingestion de données** qui permet de compter le nombre d'**interaction** de nos clients avec nos **différentes plateformes**.
- A chaque **interaction**, un message est envoyé dans notre broker kafka sur le topic "**interact**"
- Le message contient les **informations** suivantes : **idClient**, **nom**, **prénom**, **date de naissance**, **lieu de naissance**
- A la fin des **transformations**, le nombre d'**interaction** à jour, ainsi que l'**identifiant** du client sont envoyés vers une **API REST**.

Etude de cas

- Dans le cadre ou **idClient** est null, on demandera à une api rest de nous fournir, à partir des autres informations (**nom, prénom, date et lieu de naissance**) l'identifiant du client.

Etude de cas - Endpoint API

- **Récupération des informations client** :
  - **POST** /getcustomer - data {firstname, lastname} - response {customerId, firstname, lastname}
- **Envoie des données** :
  - **POST** /data - data {customerId, intercationNumber}

Merci pour votre attention

Des questions ?