Installation et présentation de Spark

Introduction

Apach Spark est un framework de traitements Big Data open source construit pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation. 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.

Spark présente plusieurs avantages par rapport aux autres technologies big data et MapReduce comme Hadoop et Storm. D'abord, Spark propose un framework complet et unifié pour répondre aux besoins de traitements Big Data pour divers jeux de données, divers par leur nature (texte, graphe, etc.) aussi bien que par le type de source (batch ou flux temps-réel).

Ensuite, Spark permet à des applications sur clusters Hadoop d'être exécutées jusqu'à 100 fois plus vite en mémoire, 10 fois plus vite sur disque. Il vous permet d'écrire rapidement des applications en Java, Scala ou Python et inclut un jeu de plus de 80 opérateurs haut-niveau. De plus, il est possible de l'utiliser de façon interactive pour requêter les données depuis un shell.

Enfin, en plus des opérations de Map et Reduce, Spark supporte les requêtes SQL et le streaming de données et propose des fonctionnalités de machine learning et de traitements orientés graphe. Les développeurs peuvent utiliser ces possibilités en stand-alone ou en les combinant en une chaîne de traitement complexe.

Cette première partie de ce rapport propose en premier lieu un aperçu de ce qu'est Spark, ses composants, ses fonctionnalités ainsi qu'une introduction sur SparkSQL. Ensuite, une petite présentation de MLlib et les algorithmes machine learning de cette librairie. Et finalement, expliquer les différentes étapes d'installation de Spark.

Installation et présentation de Spark

1. *Présentation de Spark*

Apache Spark est un framework de calcul distribué in-memory principalement qui permet de faire de l'ETL (Extract Transform and Load), de l'ELT (Extract Load and Transform), de l'analytique avec une librairie riche et complète, du Machine Learning et aussi du traitement de graphes sur des gros volumes de données, avec différents formats en batch ou en pseudo-temps réel.

On peut développer des traitements Spark avec plusieurs langages différents, Scala, Python, Java, SQL et le langage R.

Spark n'est pas une base de données et ne stocke rien, il se base sur des systèmes de stockage comme Hadoop, AWS S3, Cassandra, MongoDB…

1.1. Composants de Spark

1.1.1 Architecture générale de Spark

L'architecture de Spark comprend les trois composants principaux :

- Le stockage des données
- L'API
- Le Framework de gestion des ressources

Le stockage des données :

Spark utilise le système de fichiers HDFS pour le stockage des données. Il peut fonctionner avec n'importe quelle source de données compatible avec Hadoop, dont HDFS, HBase, Cassandra, etc.

L'API :

L'API permet aux développeurs de créer des applications Spark en utilisant une API standard. L'API existe en Scala, Java et Python.

Gestion des ressources :

Spark peut être déployé comme un serveur autonome ou sur un Framework de traitements distribués comme Mesos ou YARN. La figure 1 illustre les composants du modèle d'architecture de Spark.

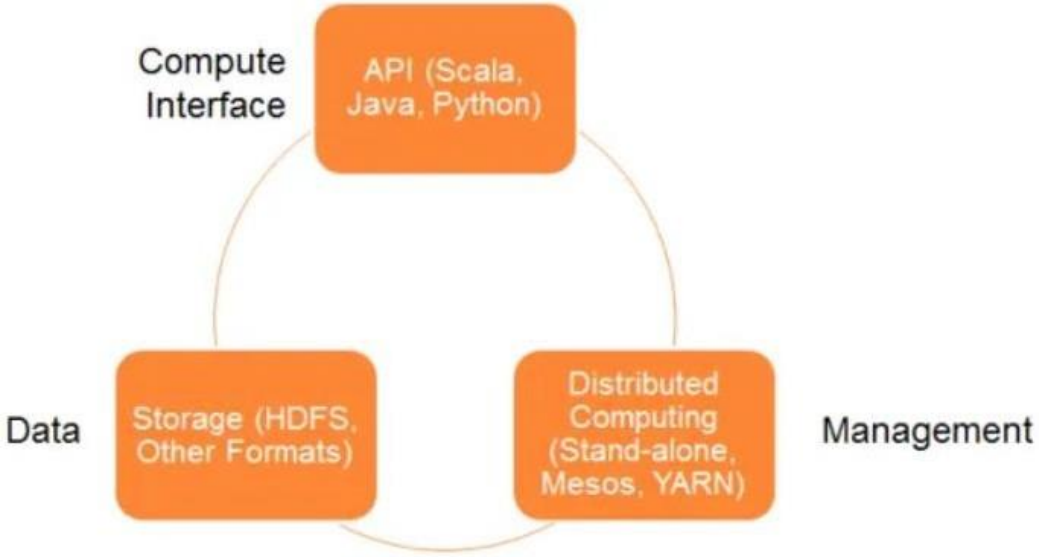


Figure 1 : Spark Architecture

Les « Resilient Distributed Datasets »

Les Resilient Distributed Datasets ou RDD, sont un concept au cœur du Framework Spark. Nous pouvons voir un RDD comme une table dans une base de données. Celui-ci peut porter tout type de données et est stocké par Spark sur différentes partitions.

Les RDD permettent de réarranger les calculs et d'optimiser le traitement. Ils sont aussi tolérants aux pannes car un RDD sait comment recréer et recalculer son ensemble de données. Les RDD sont immutables. Pour obtenir une modification d'un RDD, il faut y appliquer une transformation, qui retournera un nouveau RDD, l'original restera inchangé. Les RDD supportent deux types d'opérations :

- Les transformations
- Les actions

- Les transformations : les transformations ne retournent pas de valeur seule, elles retournent un nouveau RDD. Rien n'est évalué lorsque l'on fait appel à une fonction de transformation, cette fonction prend juste un RDD et retourne un nouveau RDD.

Les fonctions de transformation sont par exemple `map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe et coalesce.`

- Les actions : les actions évaluent et retournent une nouvelle valeur. Au moment où une fonction d'action est appelée sur un objet RDD, toutes les requêtes de traitement des données sont calculées et le résultat est retourné.

Les actions sont par exemple `reduce, collect, count, first, take, countByKey et foreach.`

1.1.2 L'écosystème de Spark

Les briques qui composent Apache Spark sont les suivantes :

Apach Spark Core : Spark Core, comme son nom l'indique, est le moteur d'exécution du framework, la base du framework. Il fournit la répartition des tâches distribuées, le scheduling et des fonctionnalités lectures/écritures de base. L'API RDD est implémentée (Resilient Distributed Datasets) sur Spark Core, qui est une collection logique de données partitionnées sur le cluster.

Les RDD peuvent être créés de deux façons : l'une est en référençant des jeux de données dans des systèmes de stockage externes (ou bien en le créant via le SparkContext) et la seconde consiste à appliquer des transformations (Map, filtre, Reduce, jointure) sur des RDD existants.

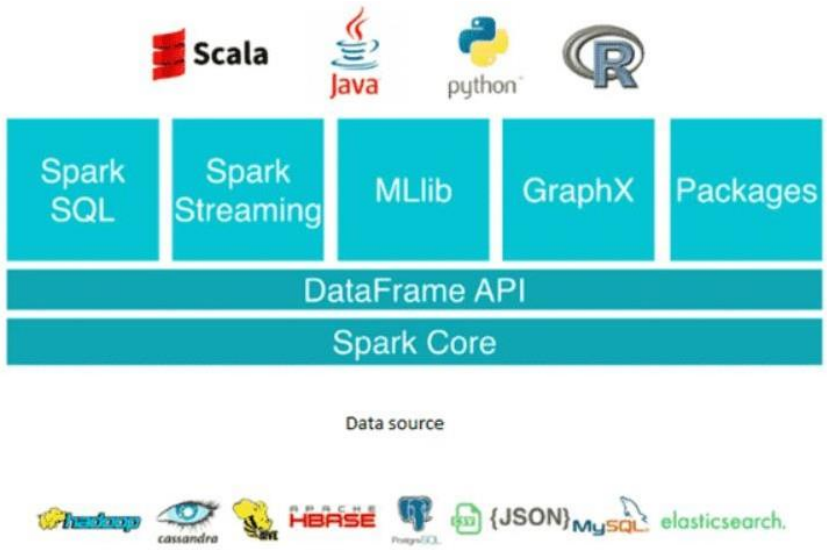


Figure 2 : Composantes du Framework Spark

À côté des API principales de Spark, l'écosystème contient des librairies additionnelles qui permettent de travailler dans le domaine des analyses big data et du machine learning. Parmi ces librairies, on trouve :

- **Spark Streaming :** peut être utilisé pour traitement temps-réel des données en flux. Il s'appuie sur un mode de traitement en "micro batch" et utilise pour les données temps-réel DStream, c'est-à-dire une série de RDD (Resilient Distributed Dataset).

- **Spark SQL :** Spark SQL permet d'exposer les jeux de données Spark via API JDBC et d'exécuter des requêtes de type SQL en utilisant les outils BI et de visualisation traditionnels. Spark SQL permet d'extraire, transformer et

charger des données sous différents formats (JSON, Parquet, base de données) et les exposer pour des requêtes ad-hoc.

- **Spark MLlib :** MLlib est une librarie de machine learning qui contient tous les algorithmes et utilitaires d'apprentissage classiques, comme la classification, la régression, le clustering, le filtrage collaboratif, la réduction de dimensions, en plus des primitives d'optimisation sous-jacentes.

- **Spark GraphX :** GraphX est la nouvelle API (en version alpha) pour les traitements de graphes et de parallélisation de graphes. GraphX étend les RDD de Spark en introduisant le Resilient Distributed Dataset Graph, un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arrêtes.

  Pour le support de ces traitements, GraphX expose un jeu d'opérateurs de base (par exemple subgraph, joinVertices, aggregateMessages), ainsi qu'une variante optimisée de l'API Pregel. De plus, GraphX inclut une collection toujours plus importante d'algorithmes et de builders pour simplifier les tâches d'analyse de graphes.

En plus de ces librairies, on peut citer **BlinkDB** et **Tachyon** :

- BlinkDB est un moteur de requêtes approximatif qui peut être utilisé pour exécuter des requêtes SQL interactives sur des volumes de données importants. Il permet à l'utilisateur de troquer la précision contre le temps de réponse. Il fonctionne en exécutant les requêtes sur des extraits des données et présente ses résultats accompagnés d'annotations avec les indicateurs d'erreurs significatifs.

- Tachyon est un système de fichiers distribué qui permet de partager des fichiers de façon fiable à la vitesse des accès en mémoire à travers des frameworks de clusters comme Spark et MapReduce. Il évite les accès disques et le chargement des fichiers fréquemment utilisés en les cachant en mémoire. Ceci permet aux divers frameworks, tâches et requêtes d'accéder aux fichiers en cache rapidement.

Le diagramme suivant (Figure 3) montre les relations entre ces différentes librairies de l'écosystème.

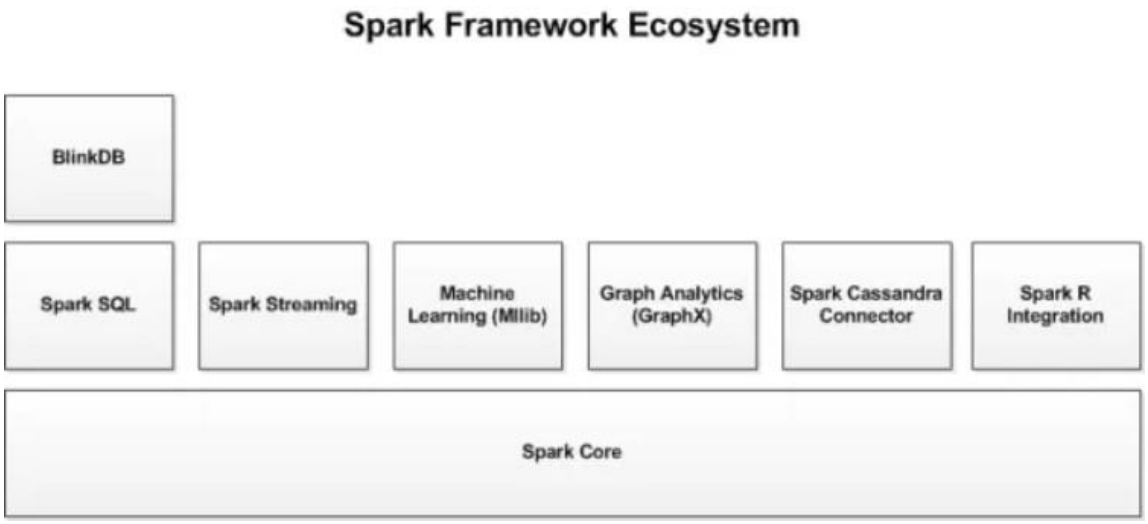


Figure 3 : Spark Framework Librairies

1.2. Fonctionnalités de Spark

1.2.1. Caractéristiques de Spark

- **Spark travaille en mémoire**

La méthode de traitement des données utilisée par Spark est beaucoup plus rapide que le MapReduce de Hadoop. Spark analyse toutes les données en mémoire presque en temps réel tandis que MapReduce va exécuter les opérations en plusieurs étapes. Spark est détenteur du record Daytona Graysort en 2014.

- **Spark intègre des outils d'analyse de données et de Data Science**

Spark Streaming permet d'accéder à des données en temps réel via les outils suivants : **Spark SQL** pour interroger et modifier les données comme avec des requêtes classiques, **Spark MLib** pour des modèles de Machine Learning et **GraphX** pour le calcul et la création de graphes.

- **Spark s'intègre à de nombreux environnements**

Spark peut être exécuté en **mode standalone** ou sur un **cluster**. Sur Hadoop, il est possible d'utiliser les ressources avec Yarn et de stocker des fichiers sous HDFS. Sur Apache Mesos ou Kubernetes. Il est également possible de l'utiliser dans le cloud. Les principales sources de données utilisées sont Cassandra, HBase et Hive, mais il en existe des centaines d'autres.

- **Spark est basé sur un langage rapide, Scala**

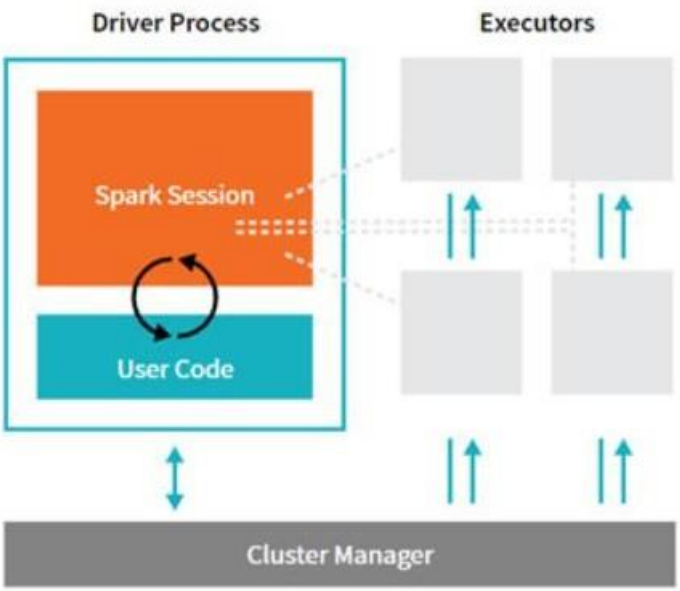
Scala est un langage objet et utilise la **programmation fonctionnelle**. Ce dernier étant régulièrement mis à jour, Spark bénéficie ainsi des améliorations et corrections apportées au langage. Spark peut également être utilisé avec les langages Python, R et Java qui sont plus répandus.

- **Spark et le principe de "Lazy Evaluation"**

Spark évalue les traitements si et seulement s'ils sont nécessaires lors de l'exécution. C'est le principe de **Lazy Evaluation**. Cela permet d'avoir un temps d'exécution plus rapide.

1.2.2. Fonctionnalités principales de Spark

Le cluster de Spark :



Le programme principal (driver) s'exécute sur l'un des nœuds du cluster. Il contient une **Spark Session** qui est le point d'entrée de l'application Spark créée. Le manager du cluster (par exemple Yarn, si Spark est déployé sur un cluster Hadoop) attribue les tâches aux workers qu'ils vont exécuter.

Spark et le concept des Resilient Distributed Datset (RDD)

Un RDD est une collection distribuée d'objets de même type (RDD d'entiers, de chaines de caractères, d'objets Scala/Java/Python) et est l'équivalent in-memory du stockage HDFS. Les données stockées par un RDD peuvent être non structurée, sans schéma imposé, comme les flux de médias. On peut contrôler leur partitionnement et leur persistance.

Il existe trois méthodes pour créer un RDD :

=> Via une collection d'objets à partir du programme appelant à l'aide de la méthode "**parallelize**" :

```
val RDDNames=spark.sparkContext.parallelize(List('Jean','Paul','Pierre'))
```

=> Par la création depuis la lecture d'un fichier :

```
val myFile=spark.read.text('spark/README.md')
```

=> Et enfin via la transformation d'un RDD existant :

```
val words= myFile.flatMap(line=>line.split(' ')).map(word=>(word,1))
```

1.2.3. Autres utilisations de Spark

Spark et la persistance des RDD

On peut indiquer à Spark de stocker un RDD. L'intérêt est de ne pas avoir à recalculer plusieurs fois le même RDD (par exemple pendant une phase de prototypage).
On applique la fonction **persist()** à un RDD en indiquant le niveau de stockage. On peut également utiliser la fonction **cache()** qui stocke le RDD en mémoire.

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.

Les variables distribuées : Broadcast

Si l'on a besoin d'une variable sur tous les nœuds, il est nécessaire de la passer en paramètre de notre fonction. On peut dans certains cas la distribuer avec la fonction broadcast. La distribution de données est très utile par exemple dans le cadre de jointure. On va ainsi distribuer la plus petite variable sur tous les nœuds, ce qui va permettre de faire des jointures « locales » avec la variable principale.

Les variables partagées : Accumulator

Un **accumulator** est une variable partagée sur plusieurs nœuds qui supporte uniquement des opérations associatives. On peut l'utiliser par exemple pour créer un compteur dans les algorithmes.

Shuffle operations

Certaines opérations comme "**ReduceByKey**" entraînent une réorganisation des données : c'est le "**Shuffling**". Ceci a un impact sur les performances, comme pour **MapReduce**, les données doivent être lues avant d'être regroupées par clé. Si les données ne tiennent pas en mémoire, elles seront écrites sur disque, ce qui va faire chuter les performances.

Ces opérations génèrent de nombreux fichiers temporaires, beaucoup d'espace disque peut ainsi être rempli si les traitements sont longs.

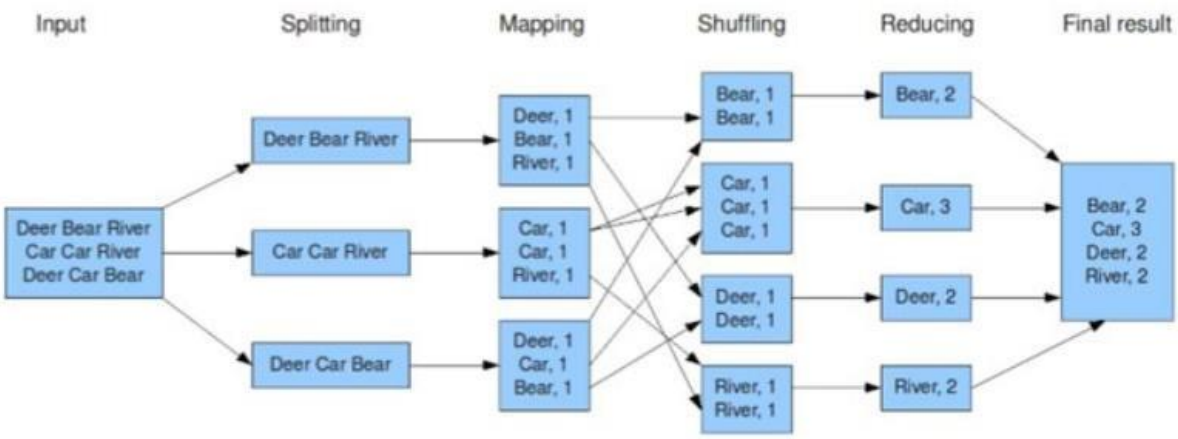


Figure 4 : Le processus de MapReduce

Comme avec le MapReduce, il faut que les traitements Spark se basent sur les données locales. C'est pourquoi, dans le cas de Hadoop par exemple, on déploie le service Spark sur l'ensemble des datanodes du cluster.

1.3. SparkSQL

De nombreux data scientists, data analysts, **data engineers** et utilisateurs de Business Intelligence s'appuient sur des requêtes SQL interactives pour explorer les données. Spark SQL est un module d'**Apache Spark** pour le traitement de données structurées.

Spark SQL fournit une abstraction de programmation appelée *Dataframe* et peut agir comme un moteur de requête SQL distribué. Ce module permet également aux requêtes **Hadoop Hive** non modifiées de s'exécuter 100 fois plus rapidement sur les déploiements et les données existants. En plus, il offre une forte intégration avec le reste de l'écosystème Spark (par exemple, en intégrant le traitement des requêtes SQL à l'apprentissage automatique).

Spark SQL est l'un des modules de Spark pour le traitement des données structurées. Contrairement à l'API de base spark RDD, l'interface offerte par spark SQL fournit à spark plus d'informations sur les structures de données et les calculs en cours, qu'il va utiliser pour effectuer des optimisations supplémentaires.

Il existe de nombreuses façons d'utiliser spark SQL, notamment **l'API SQL, l'API dataframe** et **l'API dataset**. Il convient de noter que le moteur d'exécution est le même quel que soit la manière et le langage utilisés. Grâce à cette unification, les développeurs peuvent facilement passer d'une API à l'autre, ce qui rend le traitement des données plus souple. Concrètement, Spark SQL permettra aux développeurs de :

- Importer des données relationnelles à partir de fichiers Parquet et de tables Hive.
- Exécuter des requêtes SQL sur les données importées et les RDDs existants.
- Écrire facilement des RDDs vers des tables Hive ou des fichiers Parquet.

Spark SQL comprend également un optimiseur basé sur les coûts (*Cost-based Optimizer* – appelé **Catalyst**), le stockage en colonnes et la génération de code pour accélérer les requêtes. En même temps, il peut évoluer vers des milliers de nœuds et des requêtes de plusieurs heures en utilisant le moteur Spark, qui offre une tolérance totale aux pannes en milieu de requête, sans avoir à se soucier d'utiliser un moteur différent pour les données historiques.

1.3.1. Caractéristiques de SparkSQL

- **Intégration avec Spark**

Les requêtes Spark SQL sont intégrées aux programmes Spark. Spark SQL nous permet d'interroger des données structurées à l'intérieur des programmes Spark, en utilisant SQL ou une API DataFrame. Elle peut être utilisée en Java, Scala, Python et R.

- **Accès uniforme aux données**

DataFrames et SQL prennent en charge un moyen commun d'accéder à diverses sources de données, comme Hive, Avro, Parquet, ORC, JSON et JDBC. Cela permet de joindre les données de ces sources. Cela est très utile pour accueillir tous les utilisateurs existants dans Spark SQL.

- **Compatibilité avec Hive**

Spark SQL exécute des requêtes Hive non modifiées sur les données actuelles. Il réécrit le front-end Hive et le meta store, ce qui permet une compatibilité totale avec les données, les requêtes et les UDF Hive actuels.

- **Connectivité standard**

La connexion se fait via JDBC ou ODBC. JDBC et ODBC sont les normes industrielles de connectivité pour les outils de business intelligence.

- **Performance et évolutivité**

Spark SQL intègre un optimiseur basé sur les coûts, la génération de code et le stockage en colonnes pour rendre les requêtes agiles tout en calculant des milliers de nœuds à l'aide du moteur Spark, qui offre une tolérance de panne complète à mi-requête.

Les interfaces fournies par Spark SQL donnent à Spark plus d'informations sur la structure des données et du calcul effectué. En interne, Spark SQL utilise ces informations supplémentaires pour effectuer une optimisation supplémentaire. Spark SQL peut lire directement à partir de plusieurs sources (fichiers, HDFS, fichiers JSON/Parquet, RDDs existants, Hive, etc.) Il assure l'exécution rapide des requêtes Hive existantes.

1.3.2. Pourquoi utiliser SparkSQL

Spark SQL a été conçu à l'origine comme Apache Hive pour fonctionner au-dessus de Spark et est maintenant intégré à la pile Spark. Apache Hive avait certaines limites.

- Hive lance des travaux MapReduce en interne pour exécuter les requêtes ad-hoc. MapReduce est moins performant lorsqu'il s'agit d'analyser des ensembles de données de taille moyenne (par exemple 200 Go).
- Hive n'a pas de capacité de reprise. Cela signifie que si le traitement s'arrête au milieu d'un flux de travail, vous ne pouvez pas reprendre là où il s'est arrêté.
- Hive ne peut pas déposer des bases de données cryptées en cascade lorsque la corbeille est activée, ce qui entraîne une erreur d'exécution. Pour surmonter ce problème, les utilisateurs doivent utiliser l'option Purge pour ignorer la corbeille au lieu de déposer.

Ces inconvénients ont donné lieu à la naissance de Spark SQL qui permet de surmonter ces inconvénients et remplacer Apache Hive. Mais la question qui se pose est la suivante : **Spark SQL est-il une base de données ?**

Spark SQL n'est pas une base de données mais un module utilisé pour le traitement des données structurées. Il fonctionne principalement sur les DataFrames qui sont l'abstraction de programmation et agissent généralement comme un moteur de requête SQL distribué.

2. Présentation de MLlib

Pour employer de manière efficace MLlib, il est bien entendu nécessaire d'avoir quelques bases en Machine Learning. Le Machine Learning est une branche de l'Intelligence Artificielle qui permet l'analyse et la construction d'algorithmes capables d'apprendre à partir de données d'entrée. On peut distinguer deux catégories principales d'algorithmes : de type supervisé ou non supervisé.

En apprentissage supervisé, on dispose d'un dataset composé de caractéristiques (features) associées à des labels (target). L'objectif est de construire un estimateur capable de prédire le label d'un objet à partir de ses features. L'algorithme apprend alors à partir de données dont on connait le label et est ensuite capable de faire de la prédiction sur de nouvelles données dont on ne connaît pas le label. On distingue les algorithmes de classification, pour lesquels le label à prédire est une classe (prédire un mail comme étant spam / non spam), de ceux de régression, pour lequels il faut prédire une variable continue (prédire la taille d'une personne en fonction de son poids et de son âge par exemple). En apprentissage non-supervisé, on ne dispose pas de label pour nos données. L'objectif est alors de trouver des similarités entre les objets observés, pour les regrouper au sein de clusters.

MLlib est la librairie de Machine Learning de Spark. Tous les algorithmes de cette librairie sont conçus de manière à être optimisés pour le calcul en parallèle sur un cluster. Une des conséquences directes à cela est que, pour de petits datasets qui tiennent en mémoire, un algorithme lancé depuis Spark en local sur votre machine mettra beaucoup plus de temps à s'exécuter que le même algorithme lancé depuis Python ou R, qui sont optimisés pour le mode local. En revanche, les performances deviennent extrêmement intéressantes lorsque les volumétries sont très importantes.

MLlib a été conçu pour une utilisation très simple des algorithmes en les appelant sur des RDD dans un format spécifique, quel que soit l'algorithme choisi. L'architecture se rapproche ainsi de ce que l'on trouve dans la librairie scikit-learn de Python, bien qu'il y ait encore des différences notables qui vont être effacées dans les prochaines versions de l'API.

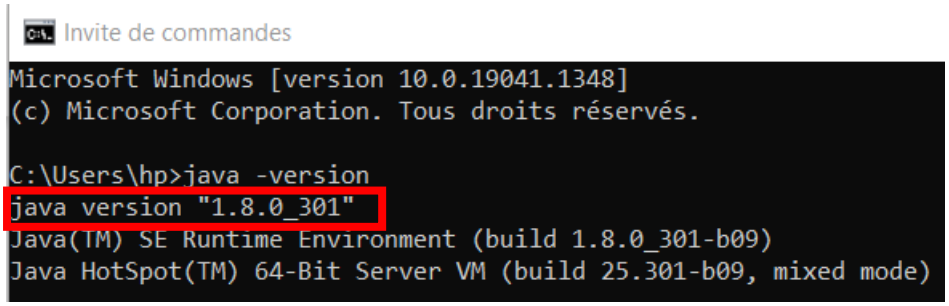
3- Etapes d'installation

Apache Spark est un framework open source qui traite de gros volumes de données de flux provenant de plusieurs sources. Spark est utilisé dans l'informatique distribuée avec des applications d'apprentissage automatique, l'analyse de données et le traitement parallèle aux graphes.

Cette partie montrera **comment installer Apache Spark sur Windows 10** et tester l'installation.

La 1ère étape : Installation du Java 8

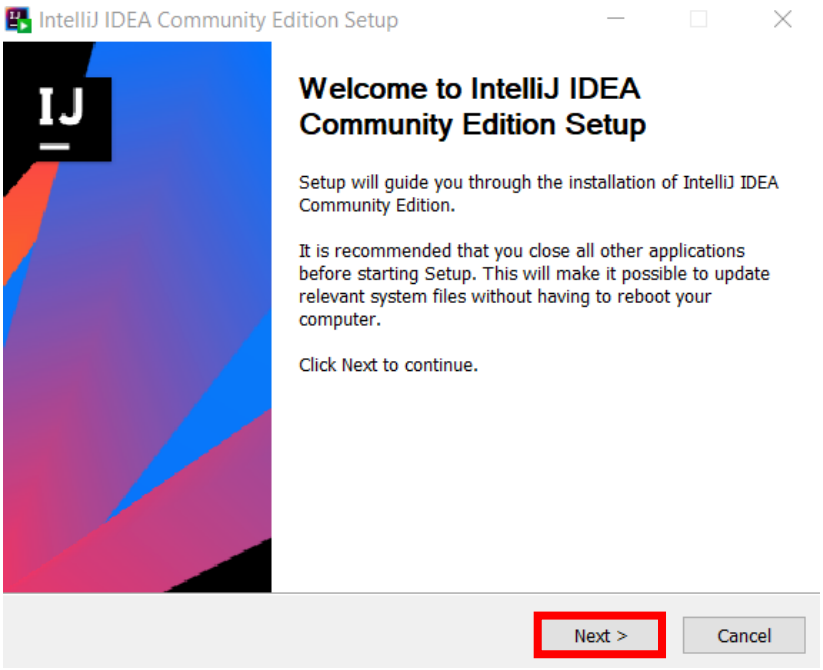
Nous avons téléchargé le JDK depuis le site d'Oracle, la version 1.8. Nous avons vérifié l'installation en tapant dans l'invite de commandes : `java -version`. Si l'installation est correcte, cette commande doit afficher la version de Java installée, comme dans la figure ci-dessous :



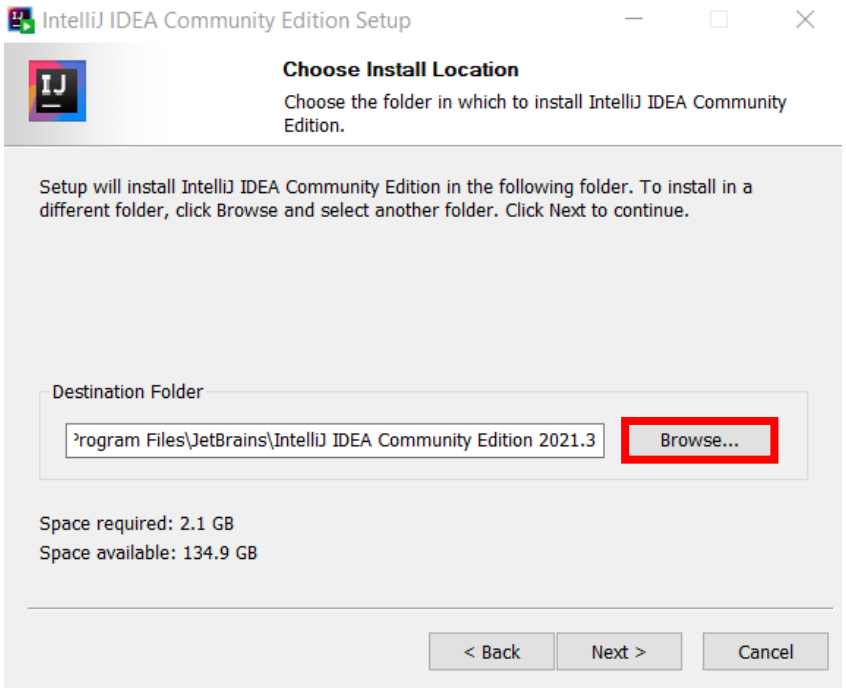
La 2ème étape : Installation du Intellij IDEA

Nous avons commencé par accéder à la page **JetBrains** (https://www.jetbrains.com/) et nous avons cherché dans les outils de développement **IntelliJ IDEA** (version : 2021.3), puis nous avons cliqué sur pour le bouton `télécharger` pour télécharger le logiciel.

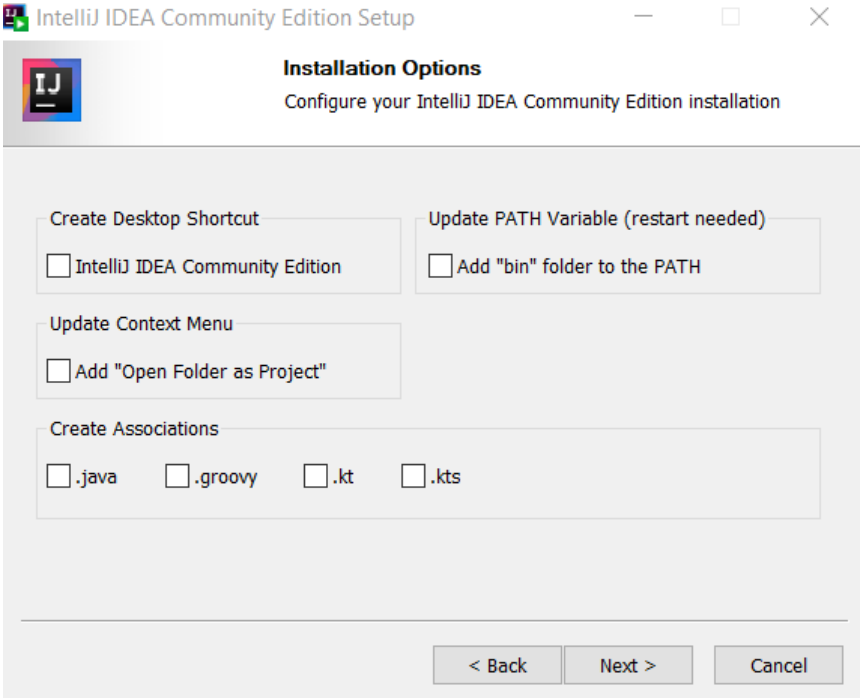
Puis nous avons ouvert le fichier d'installation .exe dans notre dossier **Téléchargements**.



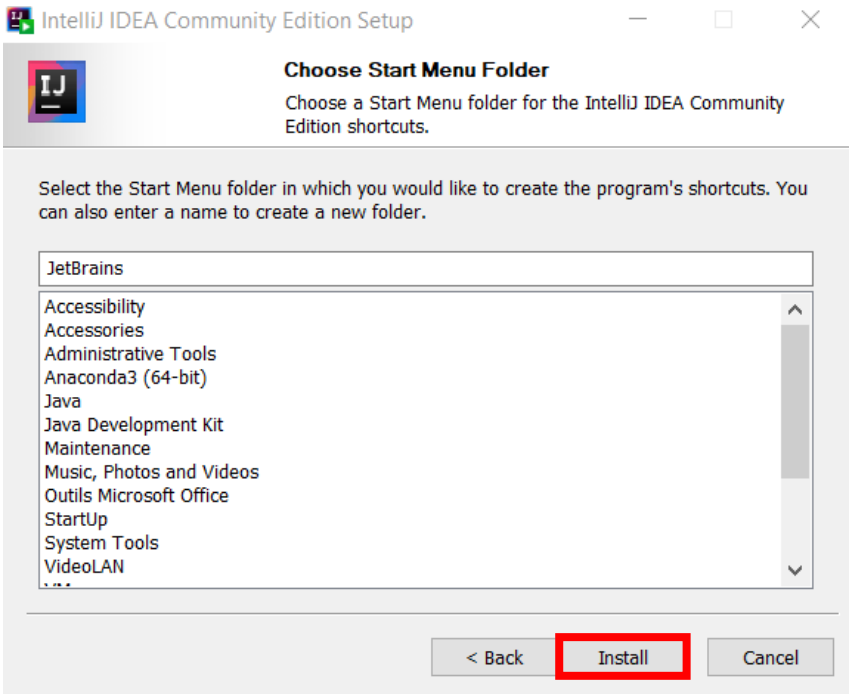
Ensuite cliquer sur Next pour installer le logiciel à l'emplacement par défaut (on peut également modifier le chemin d'installation).



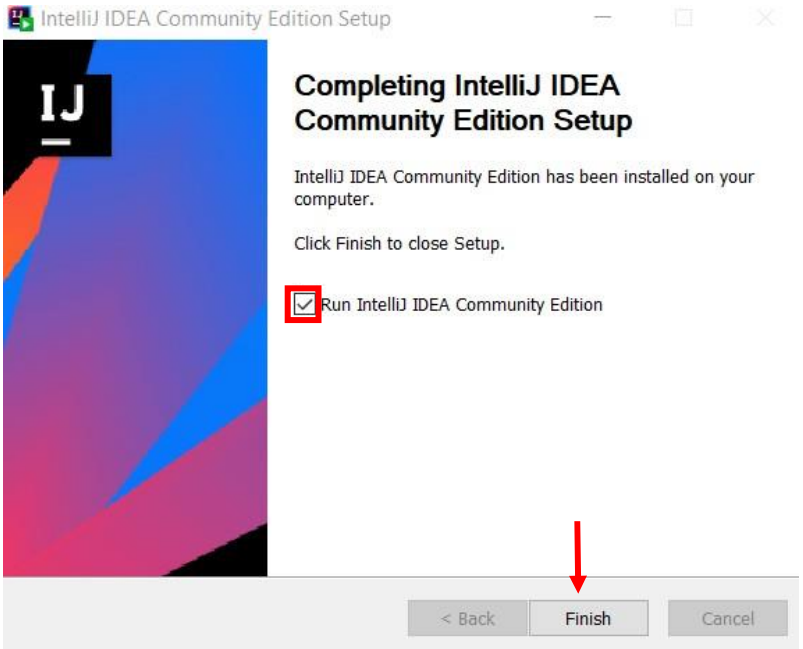
Dans cette étape, on peut choisir des options supplémentaires telles qu'un lien sur le bureau, des associations de fichiers…



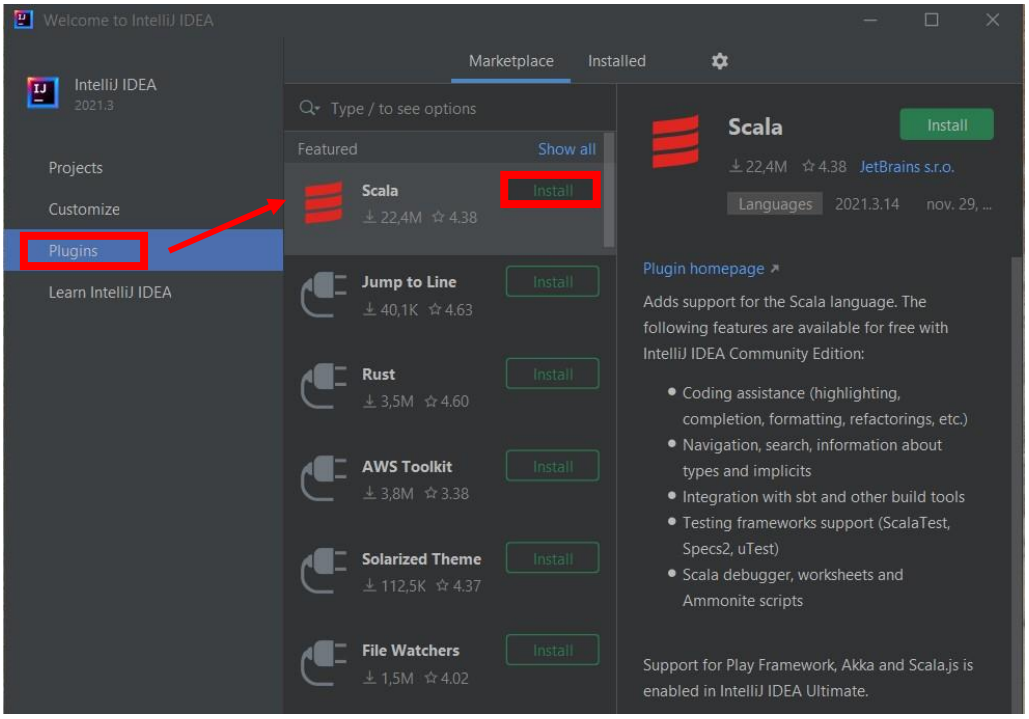
Puis cliquer sur Install pour démarrer l'installation.



Finalement, cocher la case pour démarrer le programma et cliquer sur Finish.



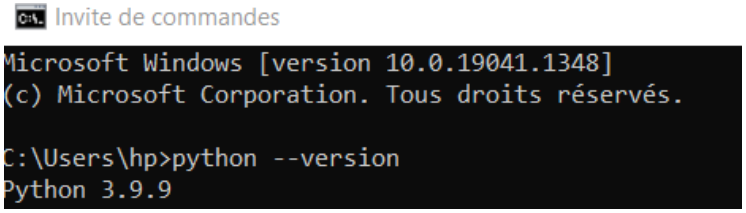
Nous arrivons à présent sur la page d'accueil permettant de créer un nouveau projet. Ici, il faut cliquer sur le bouton 'Plugins' à gauche de l'écran, puis installer **Scala**. Après l'installation, il faut redémarrer IntelliJ IDEA.



La 3ème étape : Installation du Python

Pour installer le gestionnaire de packages Python, nous avons accédé à https://www.python.org/ dans notre navigateur Web. Ensuite passer la souris sur l'option de menu **Télécharger** et cliquez sur **Python 3.9.9** (c'est la dernière version actuellement). Une fois le téléchargement est terminé, nous avons exécuté le fichier.

Nous pouvons vérifiée si Python est bien installé à l'aide de l'invite de commande, en tapant : `python --version`

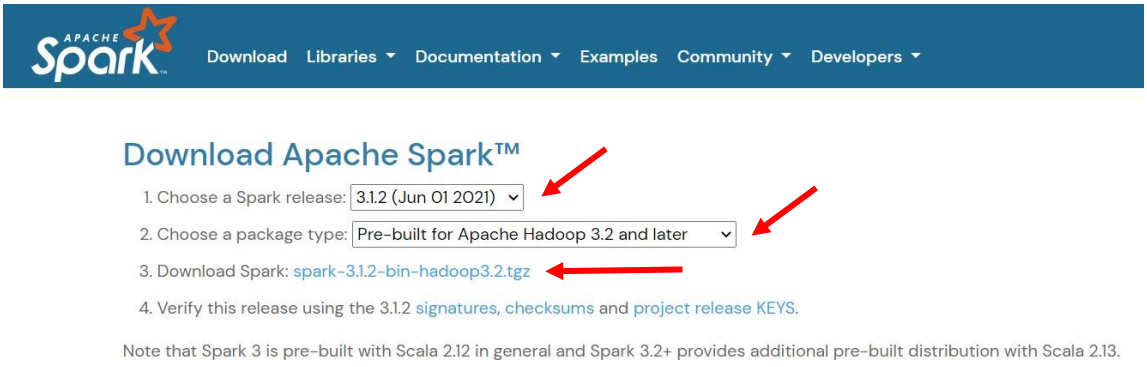


La 4ème étape : Téléchargement du Apache Spark

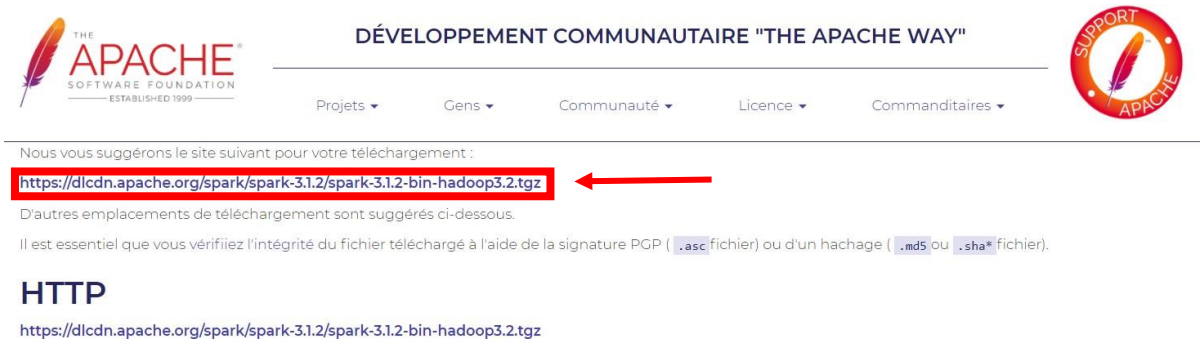
Tout d'abord, ouvrir un navigateur et accéder à https://spark.apache.org/downloads.html.

- Nous avons choisi comme version de Spark 3.1.2 (1 Juin 2021).
- Et pour le type de package, nous avons laissé la sélection Pre-built for Apache Hadoop 3.2 and later.

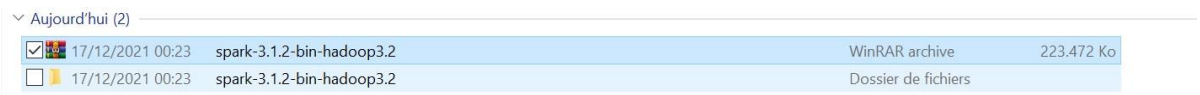
Ensuite cliquer sur le lien spark-3.1.2-bin-hadoop3.2.tgz.



Une page avec une liste de liens se charge où il y a différents serveurs à partir desquels on peut télécharger. Nous avons choisi le premier lien dans la liste puis nous avons enregistré le fichier dans notre dossier **Téléchargements**.



Apres le téléchargement de l'archive RAR, nous l'avons décompressé et nous l'avons mis dans le disque local (C :)



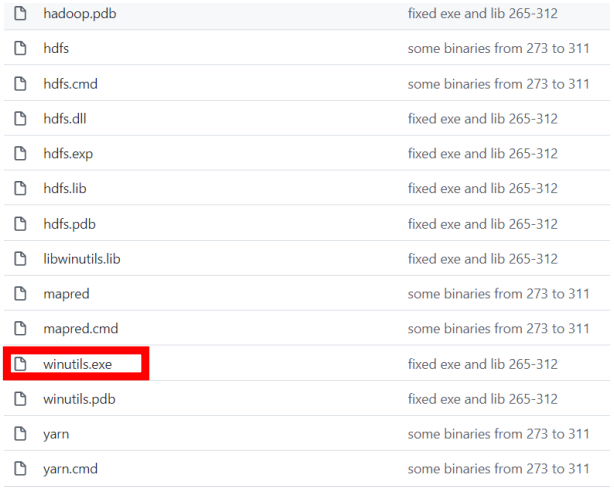
La 5ème étape : Installation du Apache Spark

L'installation d'Apache Spark implique l'extraction du fichier téléchargé à l'emplacement souhaité.

Nous avons créé un nouveau dossier nommé **Spark** à la racine de notre lecteur **C:**.

Ensuite, télécharger le fichier **winutils.exe** pour la version Hadoop sous-jacente pour l'installation Spark que nous avons téléchargé.

Puis, nous avons accédé à cette URL https://github.com/cdarlint/winutils et dans le dossier bin, nous avons localisé **winutils.exe** et cliqué dessus.



Puis cliquer sur le bouton **Télécharger** sur le côté droit pour télécharger le fichier.

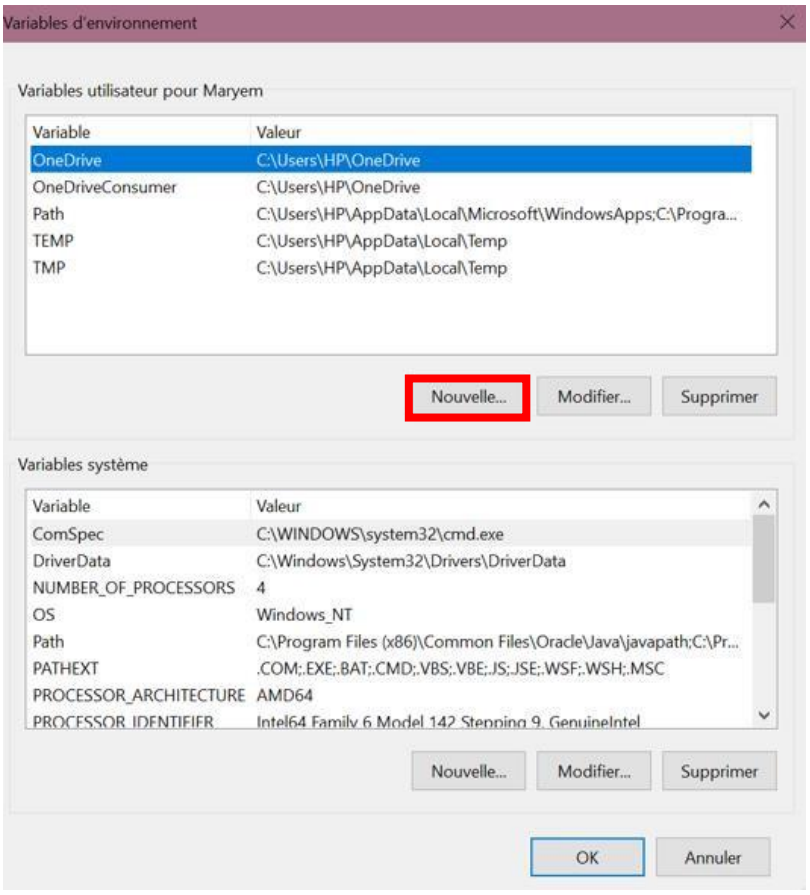
Finalement, nous avons créé de nouveaux dossiers *Hadoop* et **bin** sur C: à l'aide de l'Explorateur Windows. Et puis, Copier le fichier winutils.exe du dossier Téléchargements vers **C:\hadoop\bin**.

La 6ème étape : Configurer des variables d'environnement

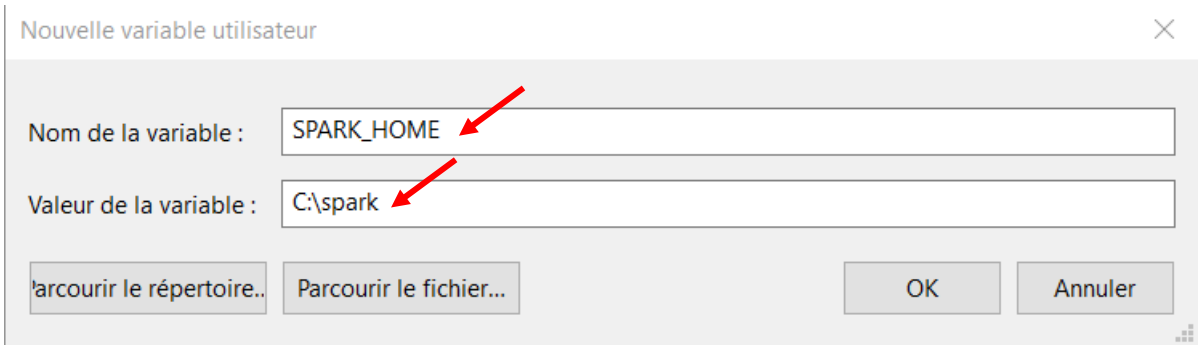
La configuration **des variables d'environnement** dans Windows ajoute les emplacements Spark et Hadoop à notre PATH système. Il nous permet d'exécuter le shell Spark directement à partir d'une fenêtre d'invite de commande.

Nous avons cliqué sur **Démarrer** et tapé *environnement*. Ensuite, sélectionner le résultat intitulé **Modifier les variables d'environnement système**.

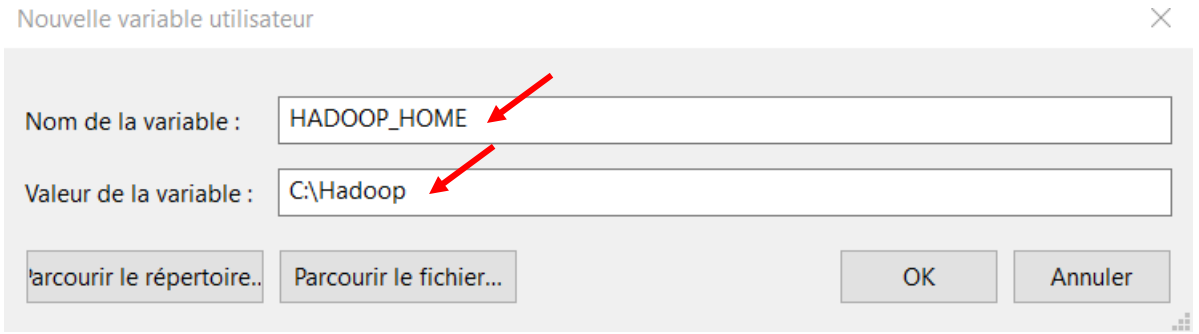
Une boîte de dialogue Propriétés système s'affiche. Dans le coin inférieur droit, nous avons cliqué **sur Variables d'environnement**, puis sur **Nouveau** dans la fenêtre suivante :

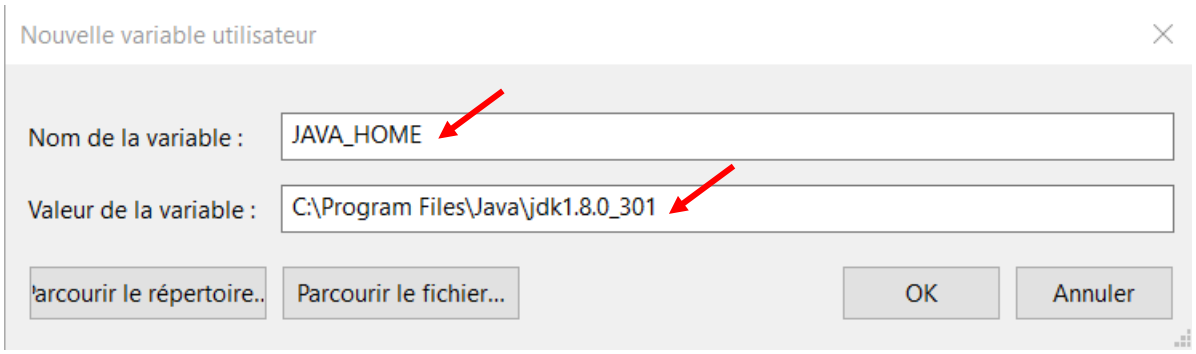


Pour *Nom de variable,* nous avons saisie **SPARK_HOME**, et pour la *valeur de la variable,* nous avons saisie **C:\spark** et cliquer sur OK.

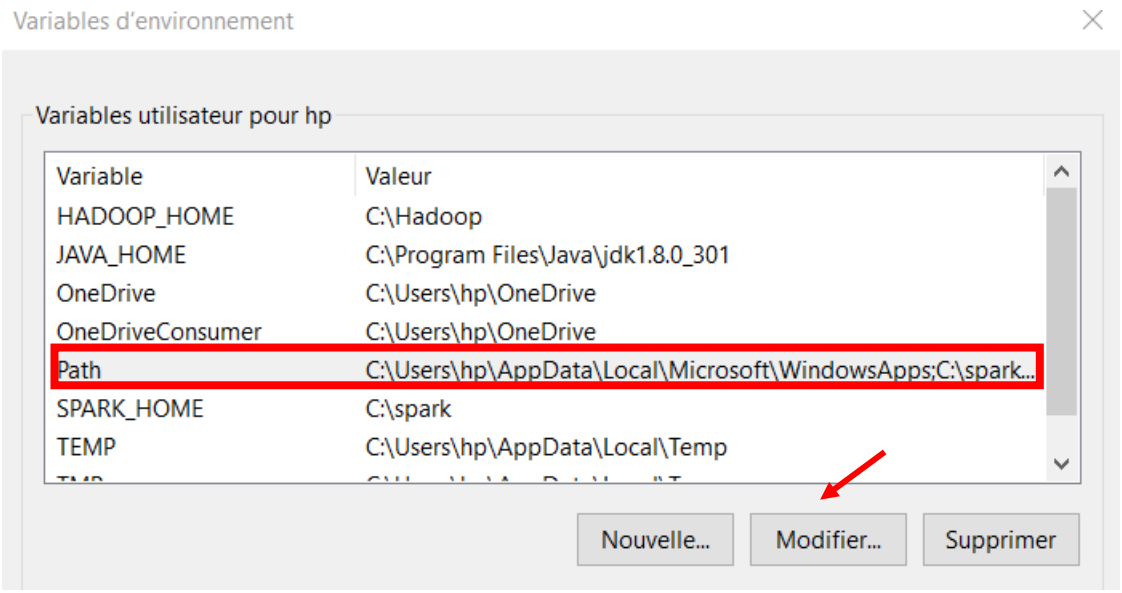


La même chose pour **HADOOP_HOME**, et **JAVA_HOME**.



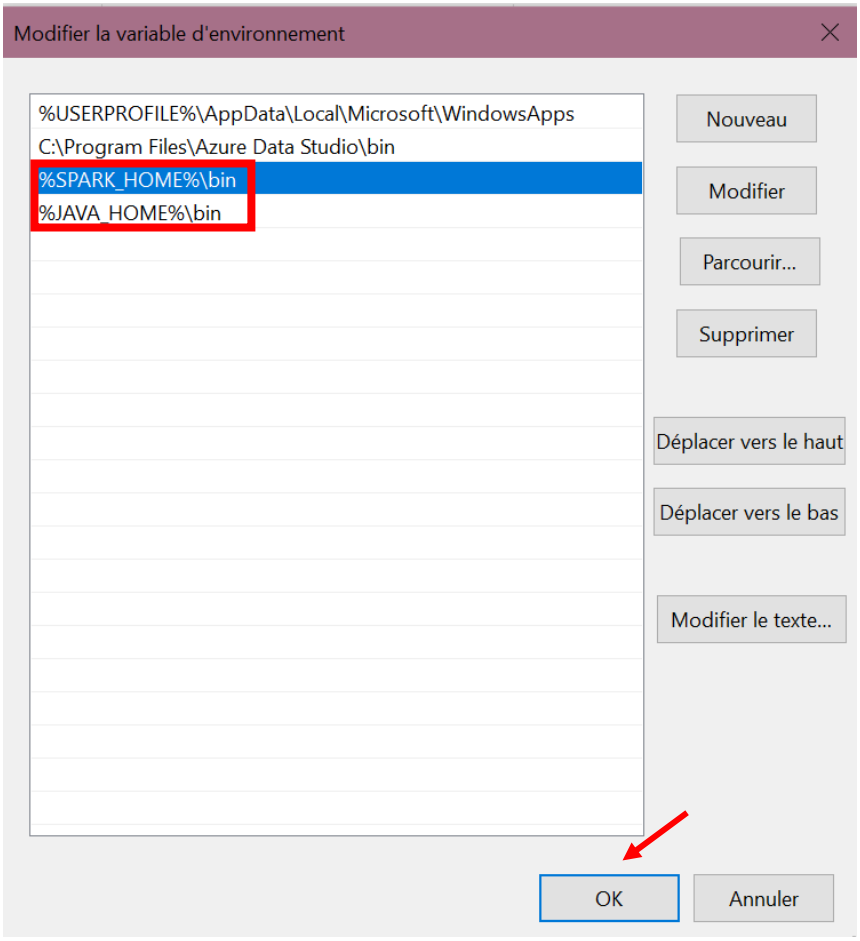


Dans la zone supérieure, nous avons cliqué sur l'entrée **Chemin**, puis sur **Modifier**. Il faut être prudent lorsque nous modifions le chemin du système et éviter de supprimer des entrées déjà sur la liste.



Nous devrions voir une boîte avec des entrées sur la gauche. Sur la droite, cliqueons sur **Nouveau**.

Le système met en surbrillance une nouvelle ligne. Nous avons écrit le chemin d'accès au dossier Spark **%SPARK_HOME%\bin**, et la même chose pour le dossier Java. Puis, cliquer sur **OK** pour fermer toutes les fenêtres ouvertes.

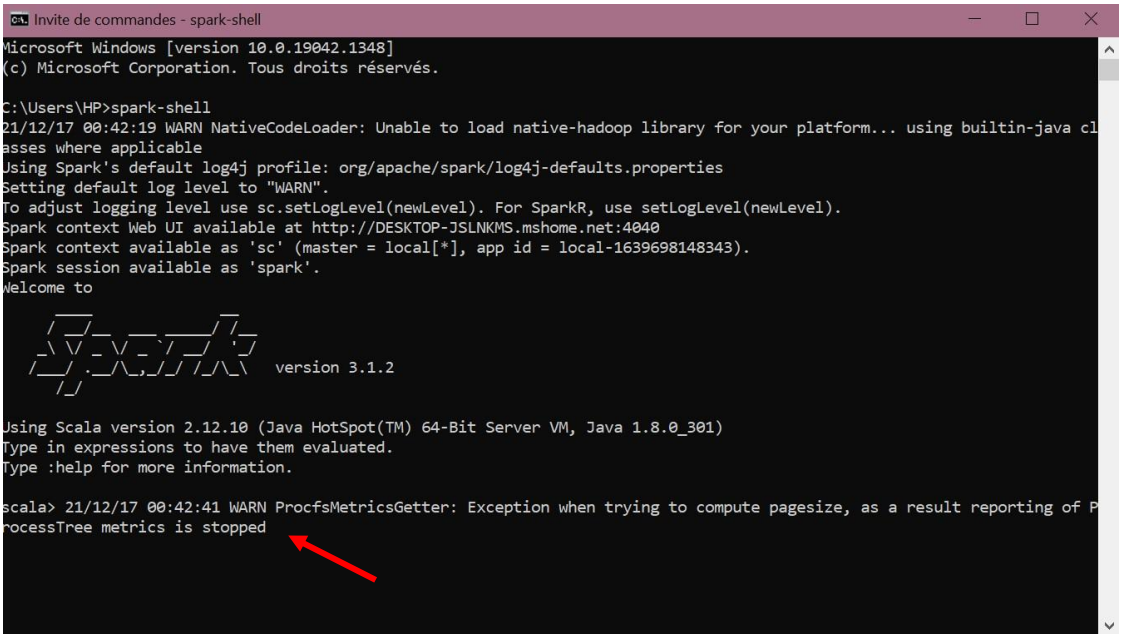


La 7ème étape : Lancer Spark

Ouvrir une nouvelle fenêtre d'invite de commande et taper `spark-shell`

Le système doit afficher plusieurs lignes indiquant l'état de la demande. Nous avons obtenu une fenêtre contextuelle Java. Il suffit de sélectionner **Autoriser l'accès** pour continuer.

Enfin, le logo Spark apparaît et l'invite affiche le **shell Scala**.



Finalement, ouvrir un navigateur Web et accéder à **http://localhost:4040/**.