**APACH SPARK**

Table des matières

[Chapitre 1 INTRODCTION : 2](#_Toc103436719)

[Chapitre 2 : ENVIRONNEMENT DE DEVELOPPEMENT UNIFIE 3](#_Toc103436720)

[Chapitre 3 : LA SUITE DE APACH SPARK 3](#_Toc103436721)

[Chapitre 4 LA LAZY EVALUATION 4](#_Toc103436722)

[Chapitre 5 TERMINOLOGIE 4](#_Toc103436723)

[1-LE DRIVER : (JavaSparkContext ou SparkCession) : 4](#_Toc103436724)

[2-Cluster Manager : 6](#_Toc103436725)

[3-Workers : 6](#_Toc103436726)

[4-Executor : 6](#_Toc103436727)

[Chapitre 6 ANATOMIE D’UNE APPLI SPARK 7](#_Toc103436728)

[Chapitre 7 RDD (Resilient Distributed Dataset) : 7](#_Toc103436729)

[Chapitre 8 OPERATIONS SUR RDD : 8](#_Toc103436730)

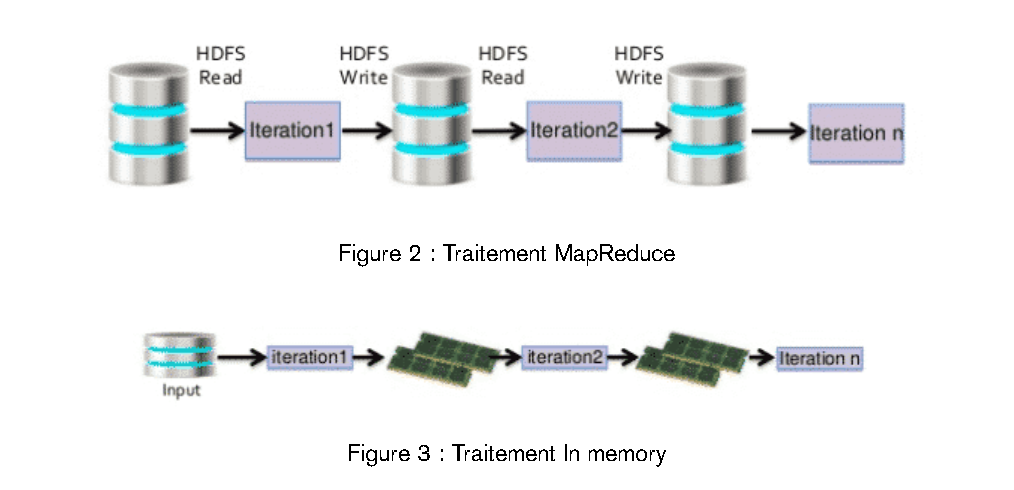
[1-Transformations : 8](#_Toc103436731)

[2-Actions : 9](#_Toc103436732)

# 

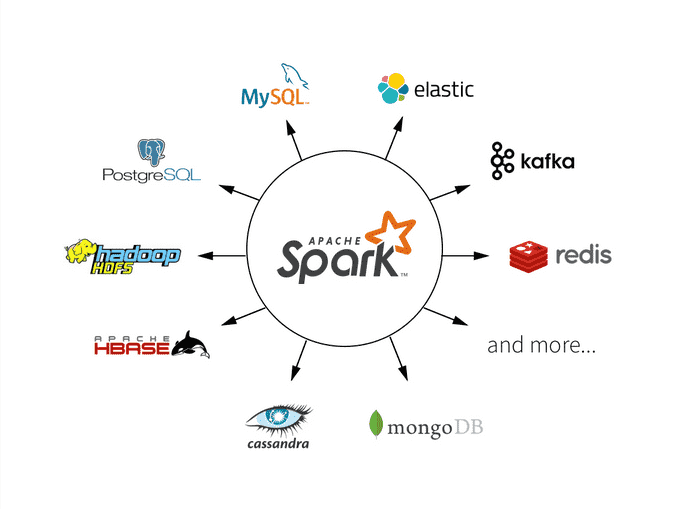
# INTRODCTION :

* Apach spark est un framework de calcul distribué en memory (parce qu’il écrit sur le disque rarement.) => traitement très rapide. (Plus optimisé par rapport à map reduce qui sollicite l’écriture à chaque étape : les workers font des vas et viens entre la ram et le disk)
* Il est écrit en langage scala.
* Spark est facile à utiliser. (Commande : spark submit)
* Possibilité de développement en local (mode standalone, émulateur spark sur Eclipse)
* On peut effectuer différentes tâches avec spark : etl ; sql, Machine Learning, analyses de données,
* Tolérant aux pannes
* Beaucoup plus performant et beaucoup plus complet que map reduce.
* Il est basé sur les RDD (Resilient Distributed Data set) : Une API similaire à l’api collection : consiste à charger dans la RAM de chaque nœud d’un cluster spark la même méthode (variable distribuée) et de traiter les données de chaque nœud avec la même méthode en même temps (en parallèle)
* Librairie très riche : JDBC, SQL, traitement des chaines de caractères… : 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.



**En conclusion :**

Spark (ou Apache Spark) est un framework open source de calcul **distribué in-memory** pour le traitement et l’analyse de données massives. Il s’agit d’un ensemble d’outils structurés selon une architecture définie. Ces composants font de Spark une plate-forme unificatrice riche en fonctionnalités : elle peut être utilisée pour de nombreuses tâches qui devaient auparavant être accomplies **avec plusieurs frameworks** différents.



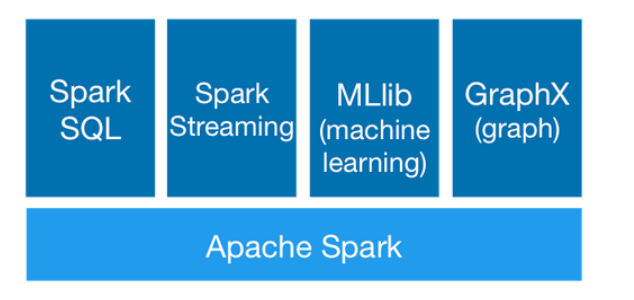
***NB :*** Un framework c’est lui et qui encadre le programmeur qui dessine son programme.

# : ENVIRONNEMENT DE DEVELOPPEMENT UNIFIE

* Il est interactif avec spark shell
* Il supporte plusieurs types de données
* Il supporte plusieurs managers de ressources (YARN, Mesos, Standalone)

Spark utilise, pour exécuter des tâches, un gestionnaire de cluster (Cluster manager) : comme le gestionnaire de groupe autonome de Spark (**Standalone**), **YARN, ou Mesos**. Les demandes Spark sont ensuite soumises à ces gestionnaires de clusters qui accorderont des ressources à la demande afin que celle-ci puisse être effectuer.

# : LA SUITE DE APACH SPARK



**Apache 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.

***NB :Data Frame*** : Un DataFrame représente simplement un tableau de données avec des lignes et des colonnes. La liste qui définit les colonnes et les types à l’intérieur de ces colonnes est appelée le schéma.

# LA LAZY EVALUATION

* Apache spark est basé sur la lazy evaluation .
* Spark possède types d’opérations, **les transformations et les actions.**
* **Les transformations en** Spark sont ce qu’on appelle lazy, cela veut dire que, quand on exécute des fonctions de transformation en Spark, le framework ne va pas les exécuter de suite mais garde un enregistrement de la fonction appelée. L’ensemble de ces opérations vont construire un graphe **DAG (Directed Acyclic Graph)**
* L’ensemble des opérations d’un traitement Spark exécuté lorsqu’une fonction de type action est invoquée dans le programme (exemple : un count, un reduce, un write …).
* Dans l’évaluation paresseuse, les données ne sont pas chargées depuis la source tant que cela n’est pas nécessaire. Ainsi, on évite les exécutions de code coûteuses et inutiles et on gagne en performance lors de l’exécution, c’est ce qui fait, en autre, l’avantage d’Apache Spark vs MapReduce de Hadoop.

# TERMINOLOGIE

LES DIFFERENTS MOTS CLES D’UNE APPLICATION SPARK SONT :

* 1-DRIVER- SPARK CONTEXT (SPARK CESSION)
* 2-CLUSTER MANAGER
* 3-WORKERS
* 4-EXECUTOR

## 1-LE DRIVER : (JavaSparkContext ou SparkCession) :

* Master de l’application : le cockpit pour l’exécution des jobs et des tâches
* C’est le point d’entrée (le main) de chaque application spark : Chaque application commence par instancier un Spark contexte dans le main de l’application.
* Construit et pilote le DAG (Direct Acyclic Graph) (ordonnancement des exécutions)
* Chaque méthode (étape dans le programme) constitue un nœud dans le DAG
* Héberge SparkUI (Spark User Interface) [http://[MasterHostname]:4040 par défaut à cette adresse]
* S’occupe des différents scheduling (ordonnancement des évènements)
* Coordonne les workers
* Grâce à SparkContext, le driver peut instancier d’autres contextes tels que SQLContext, HiveContext et StreamingContext, etc.
* Cependant, depuis la version Apache Spark 2.0, SparkSession peut accéder à toutes les fonctionnalités de Spark via un point d’entrée unifié. En plus de faciliter l’accès aux fonctionnalités Spark, il simplifie également les contextes sous-jacents pour manipuler les données.
* Pour créer un SparkContext, on doit d’abord créer un SparkConf, ce dernier stocke les paramètres de configuration, le Driver les récupère afin de les passer en paramètres au SparkContext.

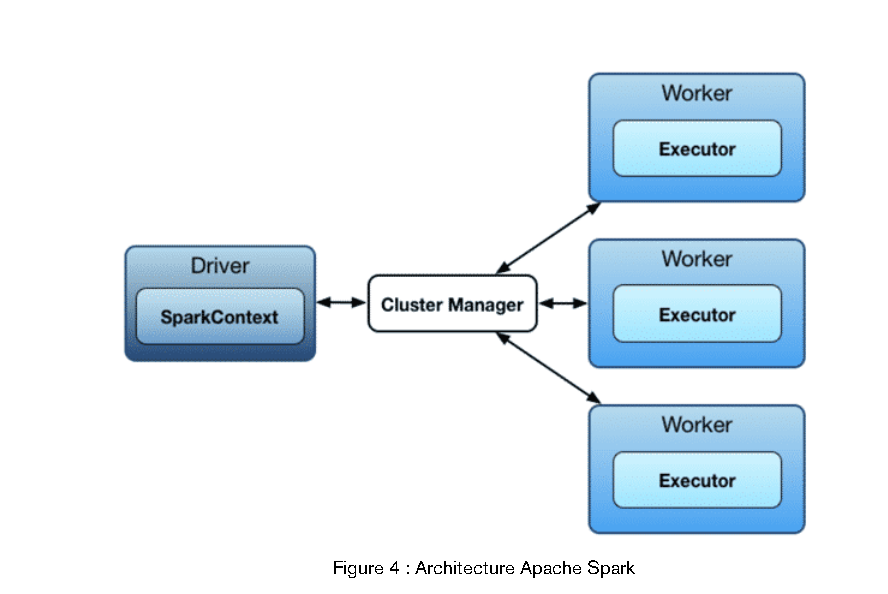
**Modes d’exécutions**

Il existe différents modes de déploiement d’une application Spark :

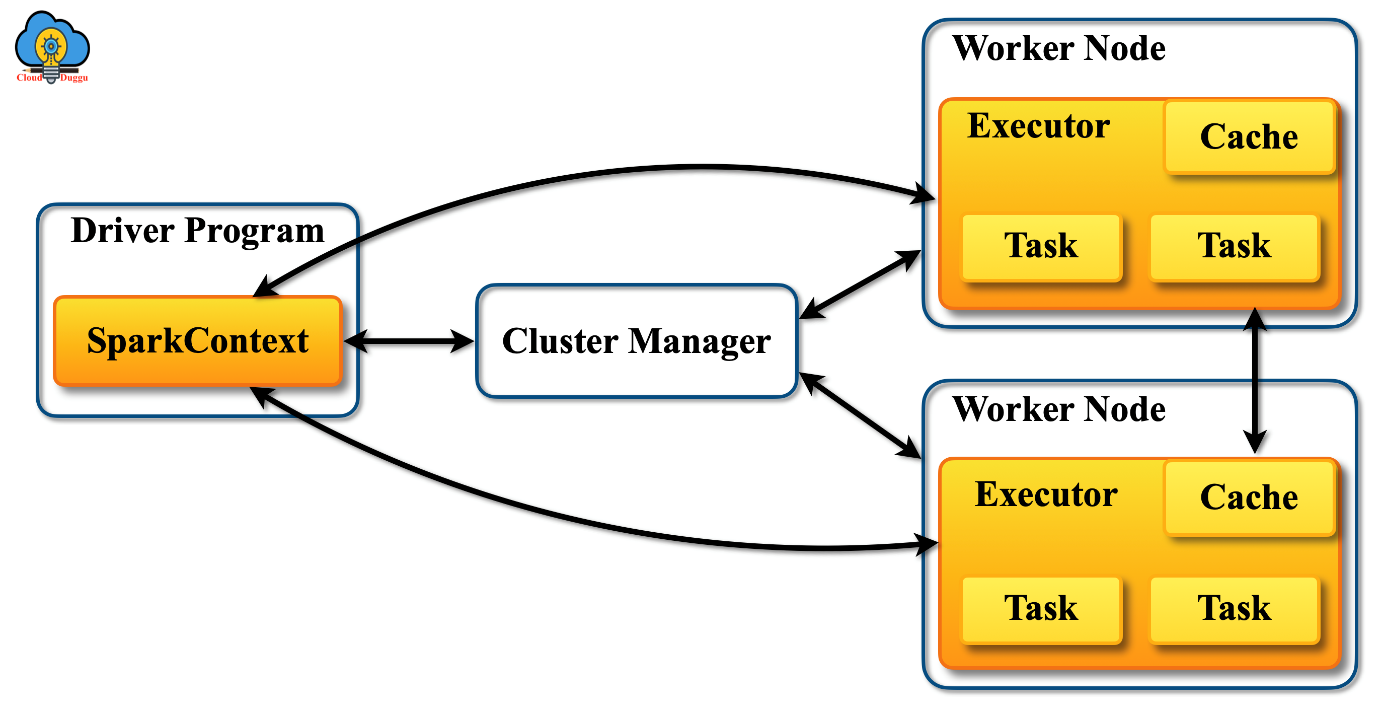
**Le mode Cluster** : C’est le plus commun, l’utilisateur envoie un fichier JAR ou un script Python au Cluster Manager. Ce dernier va instancier un Driver et des Exécuteurs sur les différents noeuds du cluster. Le CM est responsable de tous les processus liés à l’application Spark. Nous l’utiliserons pour traiter notre exemple : il facilite l’allocation des ressources et les libèrent dès que l’application est terminée.

**Le mode Client :** Quasiment identique au mode cluster à la différence que le Driver est instancié sur la machine où le job est soumis, soit à l’extérieur du cluster. Il est souvent utilisé pour le développement d’un programme car les logs sont directement affichés dans le terminal courant et l’instance du driver est liée à la session de l’utilisateur. Ce mode n’est pas recommandé en production car le Edge Node peut rapidement arriver à saturation en termes de ressources et l’Edge Node est un SPOF (Single Point Of Failure). (ça génère beaucoup de latence réseau, et risque de perte du job à cause du débit faible)

**Le mode Local :** le Driver et les Exécuteurs s’exécutent sur la machine sur laquelle l’utilisateur est connecté. Il est seulement recommandé dans le but de tester une application dans un environnement local ou pour l’exécution de tests unitaires.



## 2-Cluster Manager :



* Le role du cluster Manager est de gérer les ressources du cluster (CPU et RAM)
* Responsable de l’allocation des ressources pour chaque application Spark.

## 3-Workers :

* Un worker : C’est un (ou plusieurs) Nœud composant le Cluster Spark dans lequel un ou plusieurs exécuteurs (executor) vont exécuter les taches.
* Quand le SparkContext est instancié, chaque worker (ou plusieurs) lance un exécuteur (ou plusieurs), en fonction des paramètres définis dans le SparkConf. En effet, les exécuteurs sont finalement des JVM (conteneurs) séparées qui vont être créées. Une fois créés, les exécuteurs se connecteront à l’instance Driver. Une fois la connexion faite, ce dernier leur envoie les taches à exécuter par exemple : flatMap, map and reduceByKey. Lorsque le Driver se termine, les exécuteurs s’arrêteront aussi.

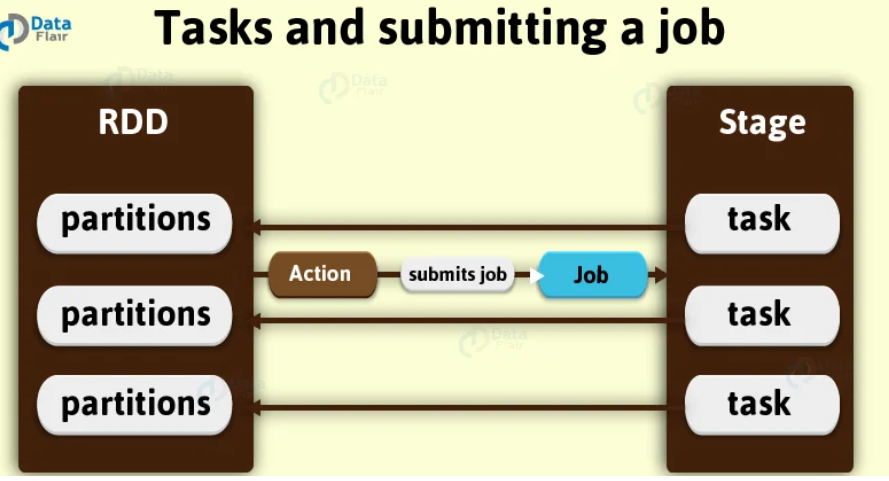
## 4-Executor :

* C’est un conteneur (JVM) qui permet d’exécuter des tâches (TASKS), il tourne sur un nœud du cluster (il s’arrête à l’arrêt de l’application)
* Un Worker peut contenir une plusieurs tâches
* Fournit un stockage en mémoire pour les RDD

**TASK :** S’occupe de la demande des ressources pour le cluster.

**Stage :** ensembles des tasks (taches) qui s’exécutent en parallèle.

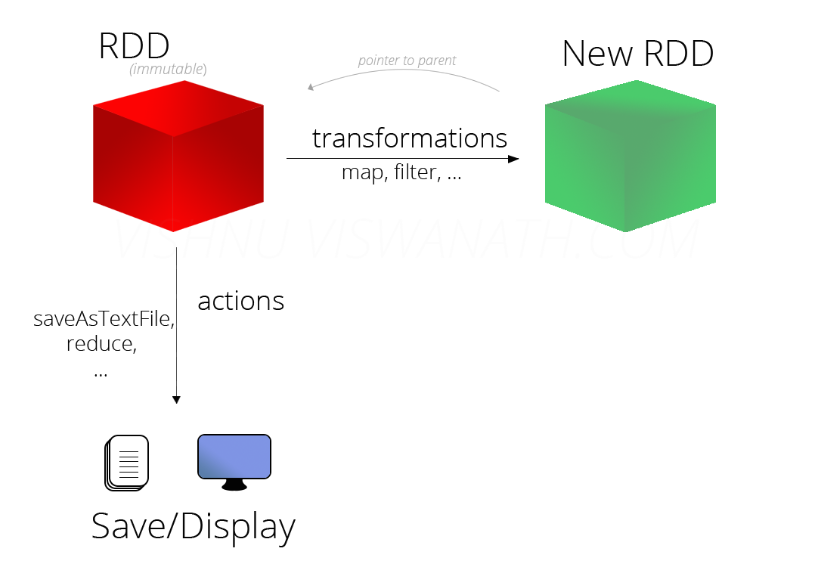
**JOB :**  ensemble des stages.



# ANATOMIE D’UNE APPLI SPARK

PDF cours, bien détaillé

# RDD (Resilient Distributed Dataset) :



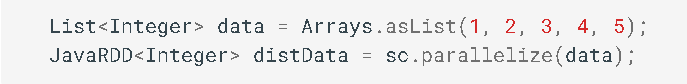
Chaque RDD est immutable et à chaque fois qu’on la transforme réellement on crée une nouvelle RDD

* **R pour Résilient** : capable de récupérer rapidement en cas de problèmes ou de conditions difficiles,
* **D pour Distribué** : partage les données sur les différents nœuds (plusieurs RAM) participants pour une exécution parallèle,
* **D pour Dataset**: une collection de données composée d'éléments séparés mais qui sont manipulés comme une unité compacte.

Les RDDs sont des collections (Lists) en mémoire. Une RDD contient des partitions et chaque partition est gérée par un nœud du cluster. Une RDD permet de faire du traitement en parallèle grâce à ses partitions. On partage un fichier CSV par exemple en plusieurs partitions et chaque nœud charge en mémoire d’opération la même fonction (traitement) partition. Une RDD est typée (exemple RDD de string ou RDD d’entier ou RDD …etc)

Il existe deux moyens de créer les RDDs :

1. Paralléliser une collection existante en mémoire dans le programme Driver.
2. Le générer à partir d'un fichier enregistré sur un support de stockage externe.



créer une RDD avec du code



Créer une RDD à partir d'un fichier externe

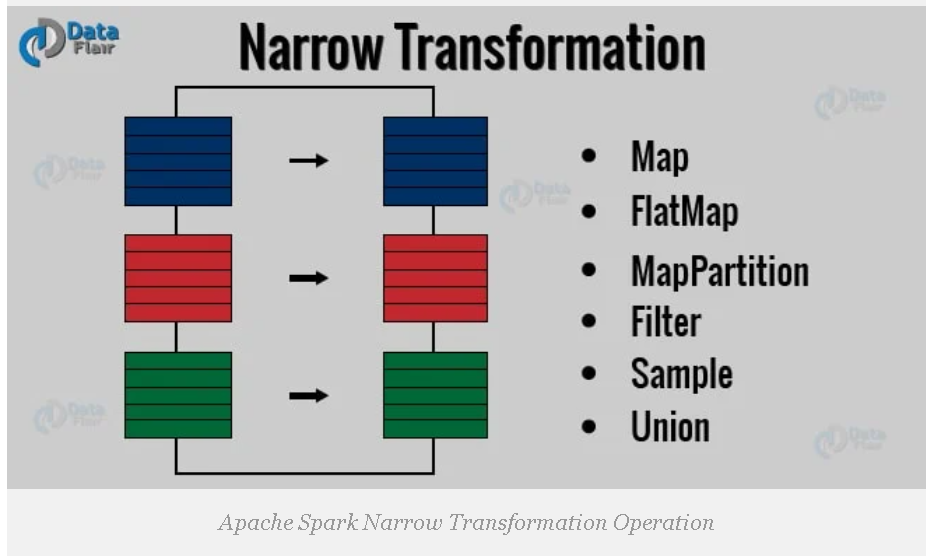
# OPERATIONS SUR RDD :

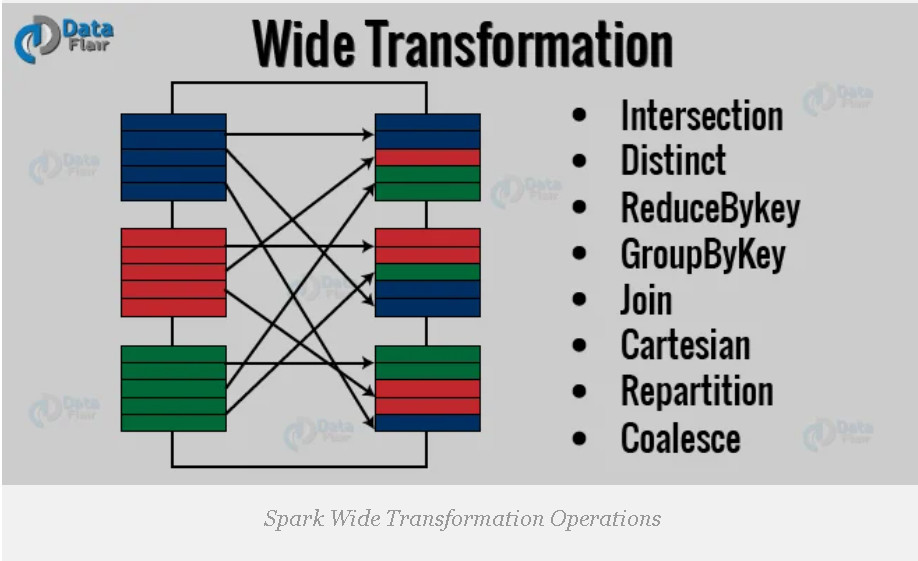
Il existe 2 types d’opération sur les RDD : Les Transformations et Les Actions

## 1-Transformations :

Se sont les opérations paresseuses, (Lazy), Sont à l’origine de la création du DAG par contre le job lui-même ne s’exécute qu’en faisant une Action

Il existe des transformations de type étroite (étroite) et transformation large (wide):





## 2-Actions :

Permettent d’exécuter le calcul et de retourner un résultat au Driver ou l’écrire sur un stockage (disque par exemple). Quand on retourne un calcul pour le Driver il faut que la machine (le nœud) qui héberge le Driver soit adéquate (CPU, Disque et RAM) de recevoir la quantité de données. Si par exemple la machine a 16 G de RAM et on envoie 100 G le programme Spark va planter car la machine ne pourra pas supporter toute la quantité de données.

Les différentes opérations de spark :

<https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/>