

Formation Data Scientist

**Soutenance Projet 8 :
Déployez un modèle dans le cloud**

-- Mélanie WARY --

CONTEXTE

Développement de robots cueilleurs intelligents permettant de préserver la biodiversité des fruits via des traitements spécifiques pour chaque espèce.



Fruits!

PROBLÉMATIQUE

Mise en place d'une première version du moteur de **classification des images de fruits**, dans un **environnement Big Data**, à travers une application mobile grand public

MISSION

Développer dans un **environnement Big Data** d'une première chaîne de traitement des données qui comprendra le **preprocessing** et une étape de **réduction de dimension**.

CONTRAINTES

Pour préparer le passage à l'échelle en termes de volume de données :

- scripts développés en **Pyspark**
- Recours à une **architecture Big Data** via **AWS**:
 - capacité de stockage avec S3
 - Puissance de calcul avec EC2

JEU DE DONNEES IMAGES

- 131 fruits
- Vus sous plusieurs angles (plusieurs dizaines de photos / fruits)
- Images 100x100 pixels, fond blanc uniforme
- Training set: 67692 images, testing set: 22688 images
- Sample set créé: 3 espèces de fruits, 5-6 images chacun



Fruits!



Sample



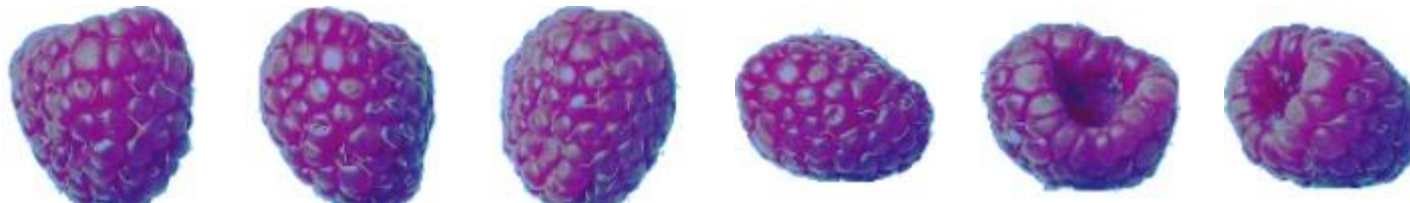
Carambula



Apricot



Raspberry



Problématique

Pyspark

AWS

Traitement

Conclusion

BIG DATA – ENJEUX ET SOLUTIONS

Grand volume de données à analyser

Grande capacité de
stockage

Grande puissance
de calcul

BIG DATA – ENJEUX ET SOLUTIONS

Grand volume de données à analyser

Grande capacité de
stockage

Grande puissance
de calcul





Cloud (AWS):
serveurs de stockage
et de calculs

BIG DATA – ENJEUX ET SOLUTIONS

Grand volume de données à analyser

Stratégie de calcul
pour les analyserGrande capacité de
stockageGrande puissance
de calcul**Framework (Spark)**
de calculs distribués**Cloud (AWS):**
serveurs de stockage
et de calculs



 python™ +  *Spark*™

=

Py*Spark*™

=

solution de calculs distribués

 python™ + 

=



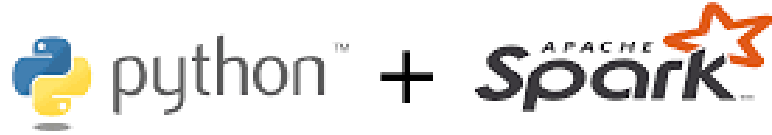
=

solution de calculs distribués**Concept des calculs distribués:**

Diviser

... pour distribuer

... pour régner



=

PySpark

=



solution de calculs distribués

Concept des calculs distribués:

Diviser le jeu de données massives initial en sous-jeux (partitions)...

... **pour distribuer** et traiter indépendamment les partitions sur différents serveurs de calcul distants et autonomes...

... **pour régner** en recombinaison les résultats intermédiaires obtenus sur chaque partition et ainsi construire la solution au problème initial

 python™ + 

=

PySpark

=

solution de calculs distribués**Concept des calculs distribués:**

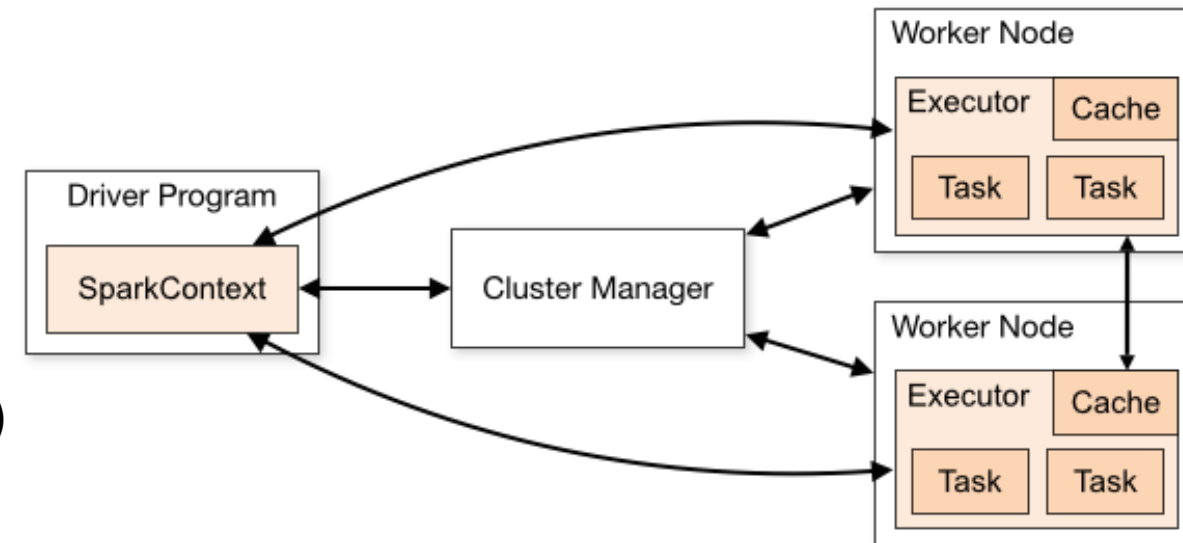
Diviser le jeu de données massives initial en sous-jeux (partitions)...

... **pour distribuer** et traiter indépendamment les partitions sur différents serveurs de calcul distants et autonomes...

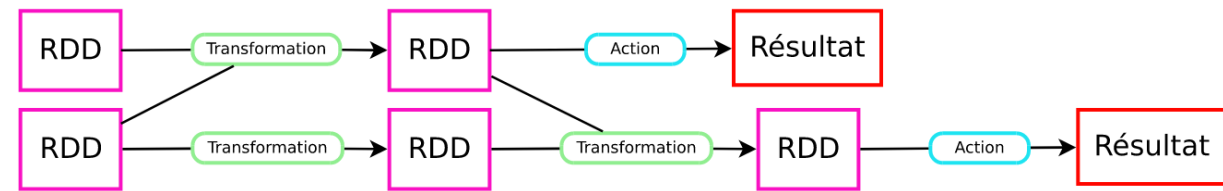
... **pour régner** en recombinaison les résultats intermédiaires obtenus sur chaque partition et ainsi construire la solution au problème initial

Architecture d'un cluster de calcul Spark:

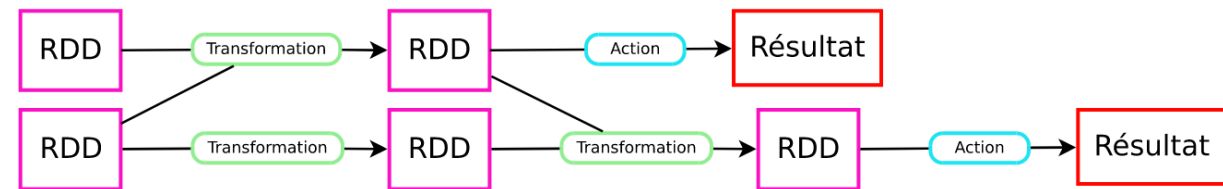
- Le **driver**, qui coordonne l'exécution des différents processus sur le cluster. Il fait appel au...
- ... **cluster manager**, chargé de l'allocation des ressources, i.e...
- ... des **executors**, présents sur les nœuds du cluster (**worker nodes**), qui vont stocker les données (en RAM = gain de temps) et effectuer dessus les **tâches** de calcul reçues de la part du driver, et informer celui-ci de leur complétion.



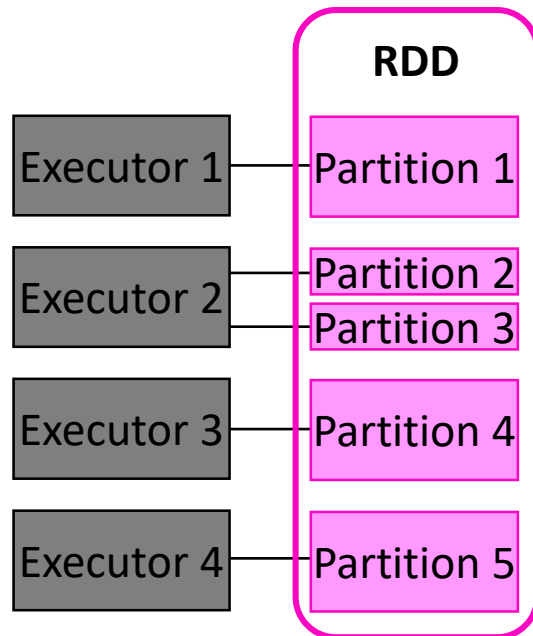
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



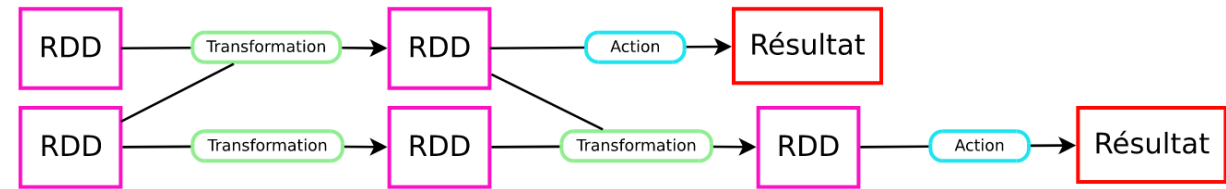
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



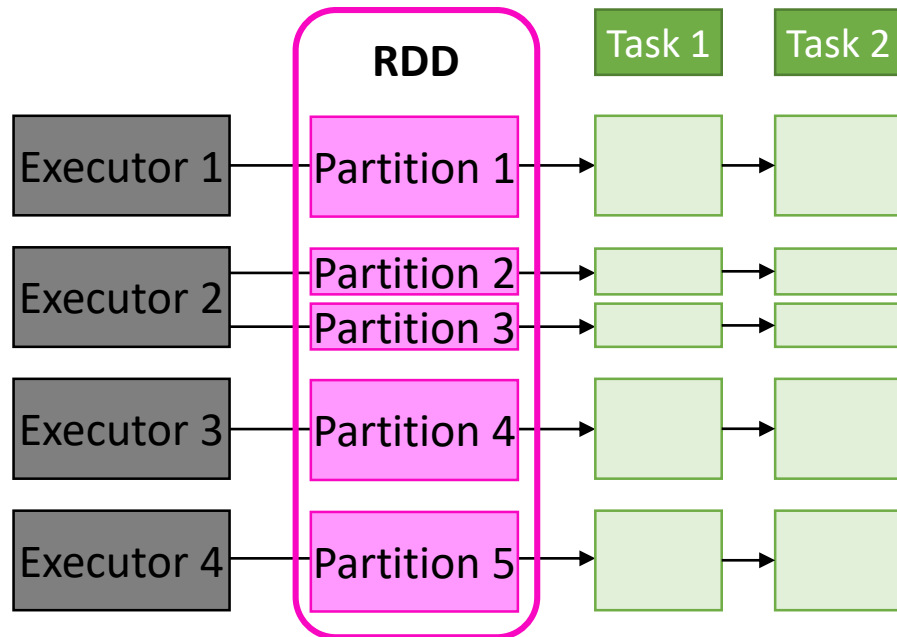
Distribution des calculs:



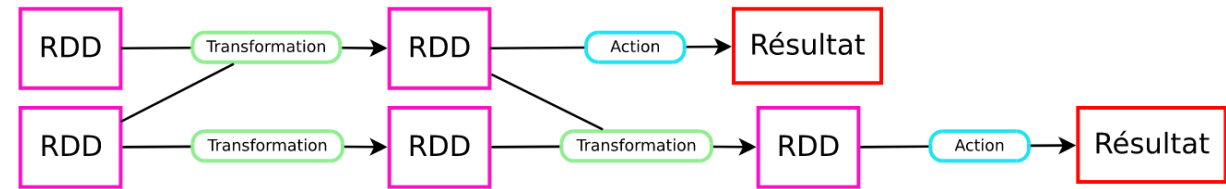
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



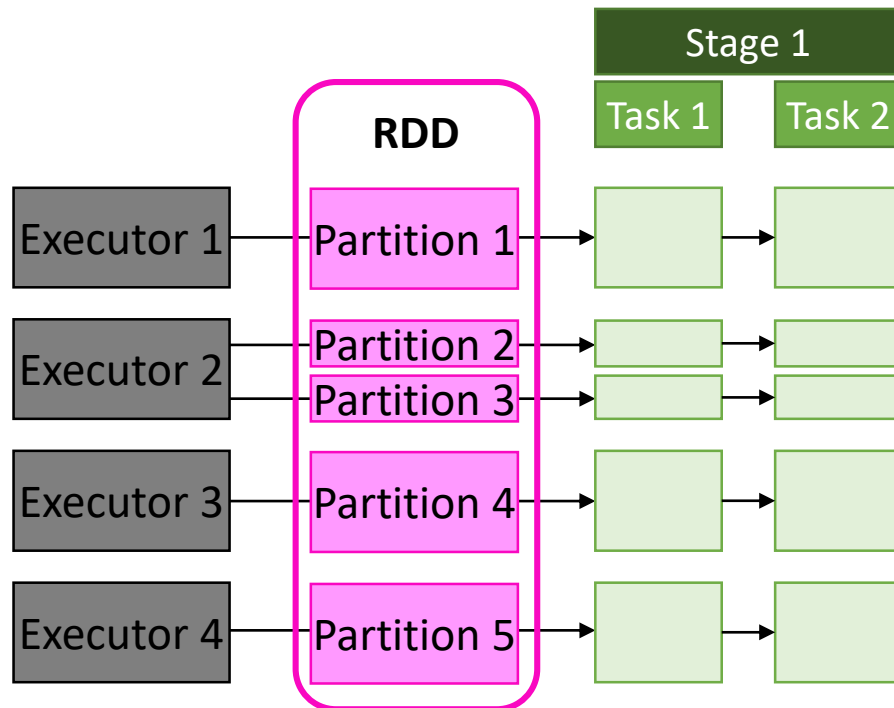
Distribution des calculs:



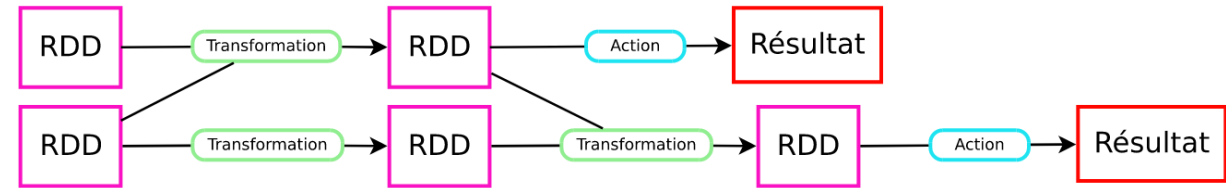
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (≈ jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



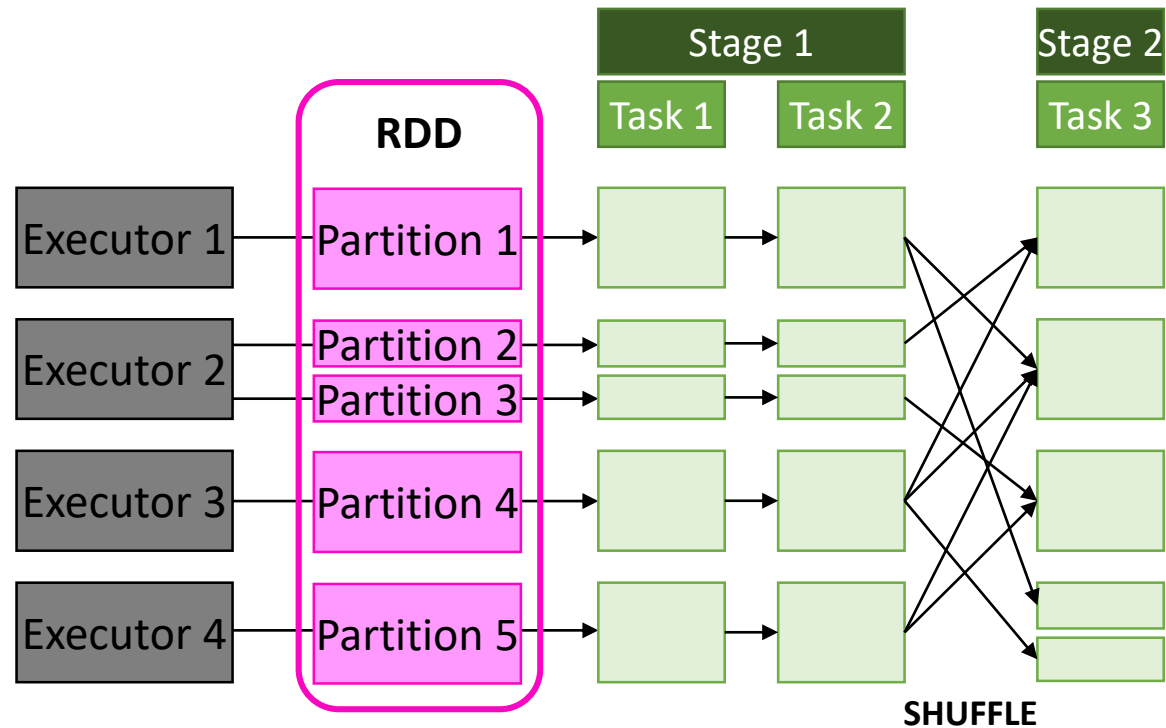
Distribution des calculs:



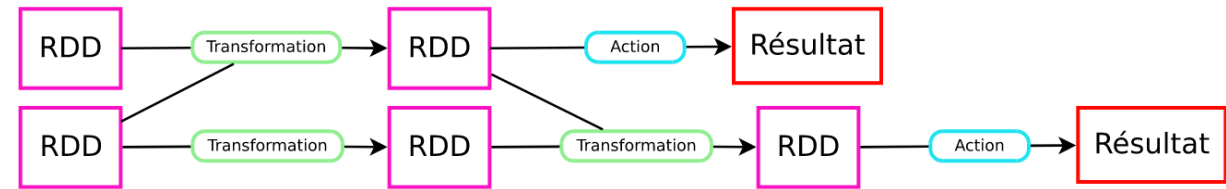
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



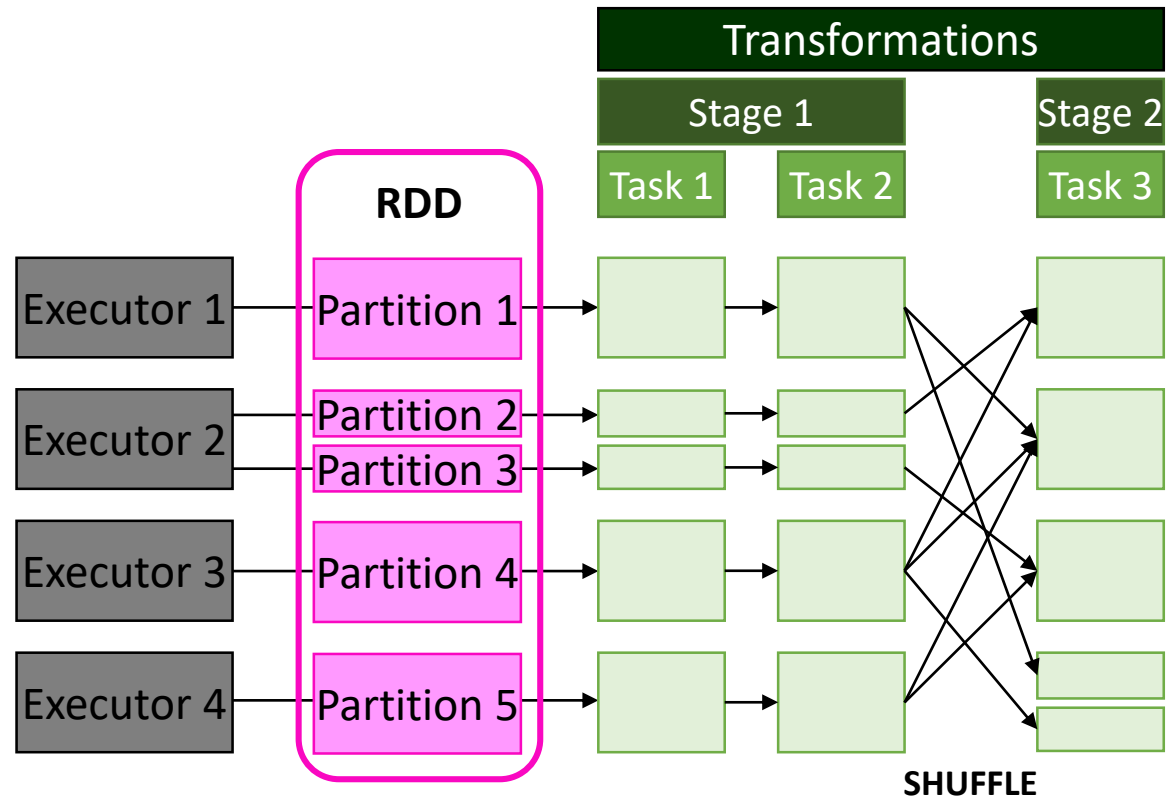
Distribution des calculs:



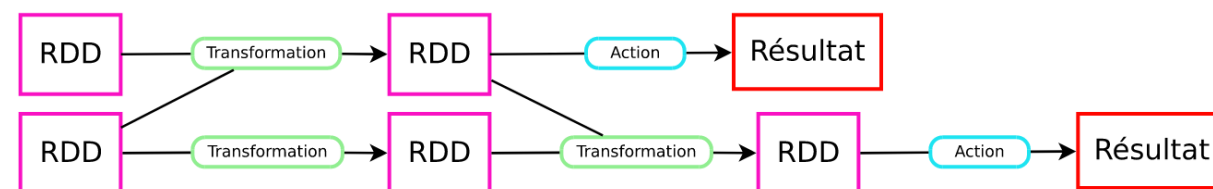
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



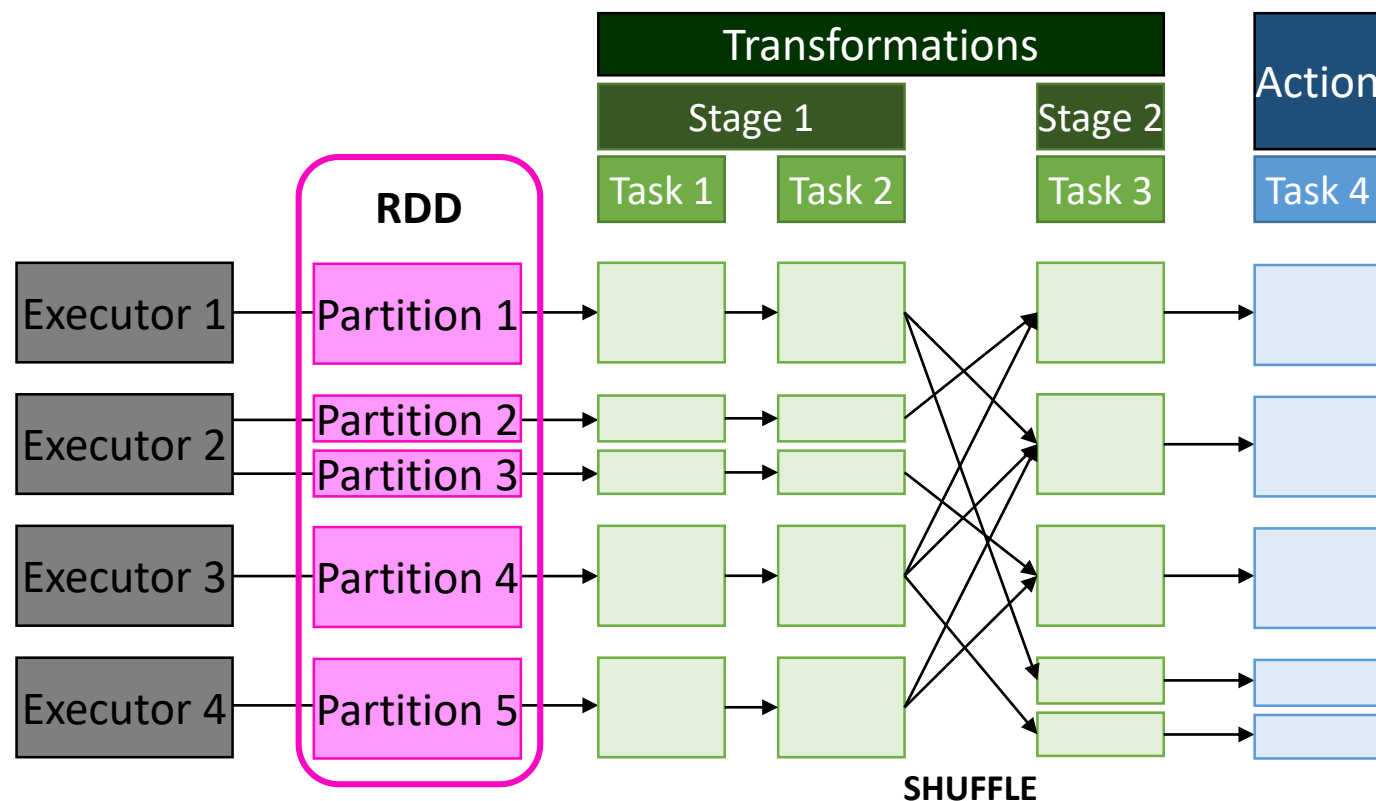
Distribution des calculs:



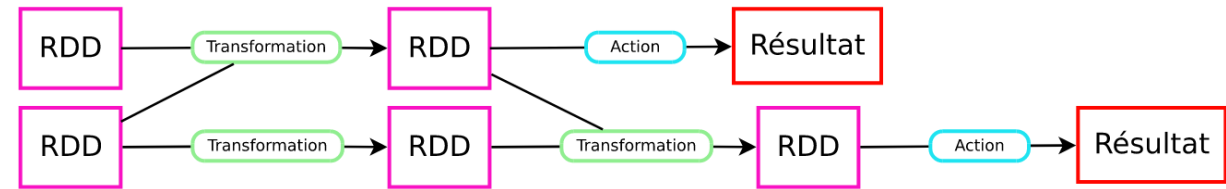
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



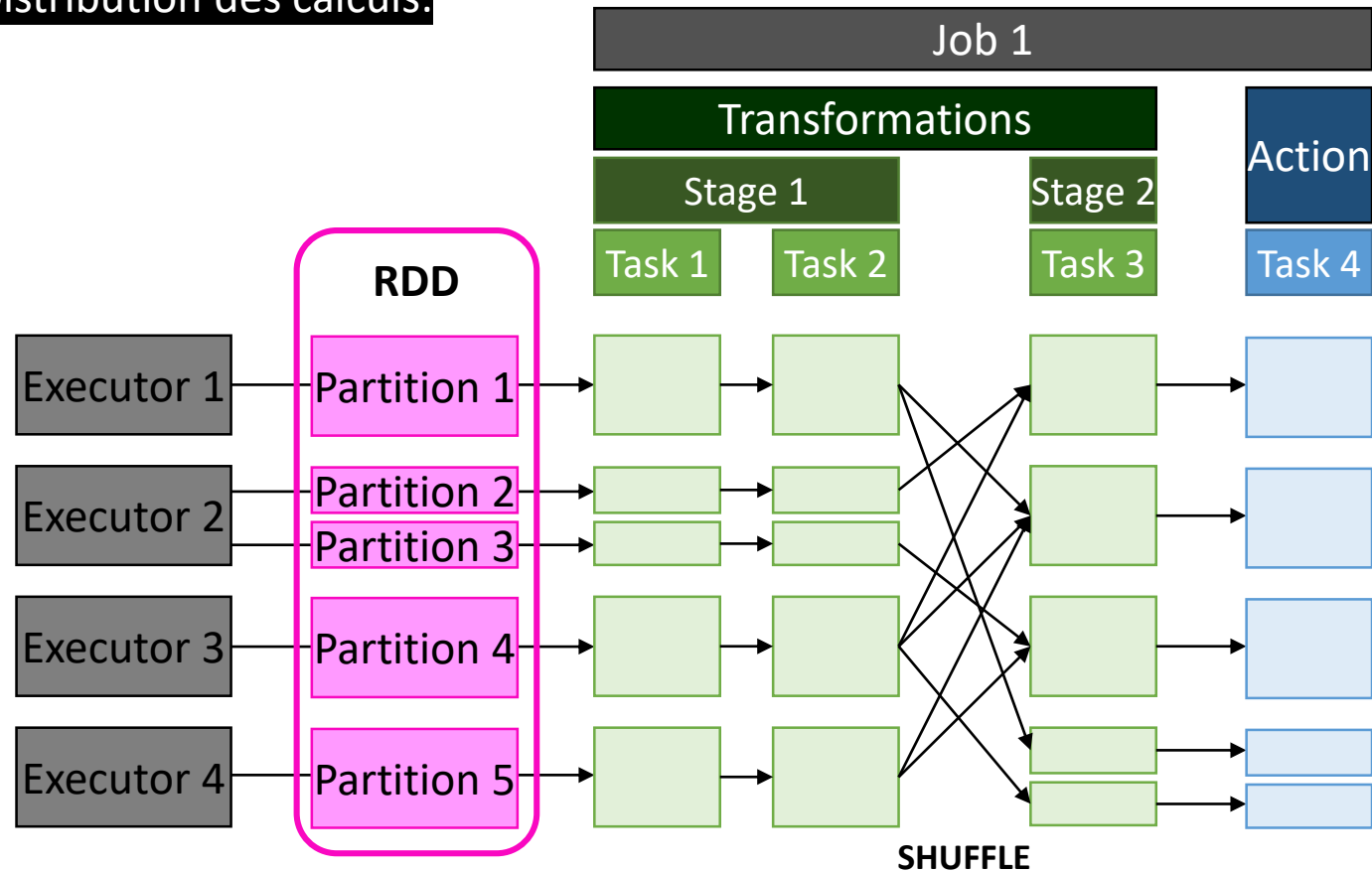
Distribution des calculs:



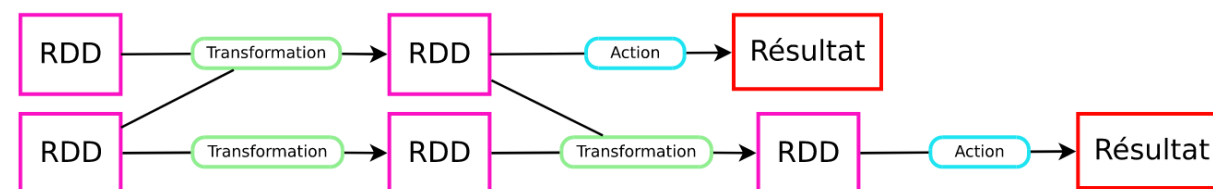
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



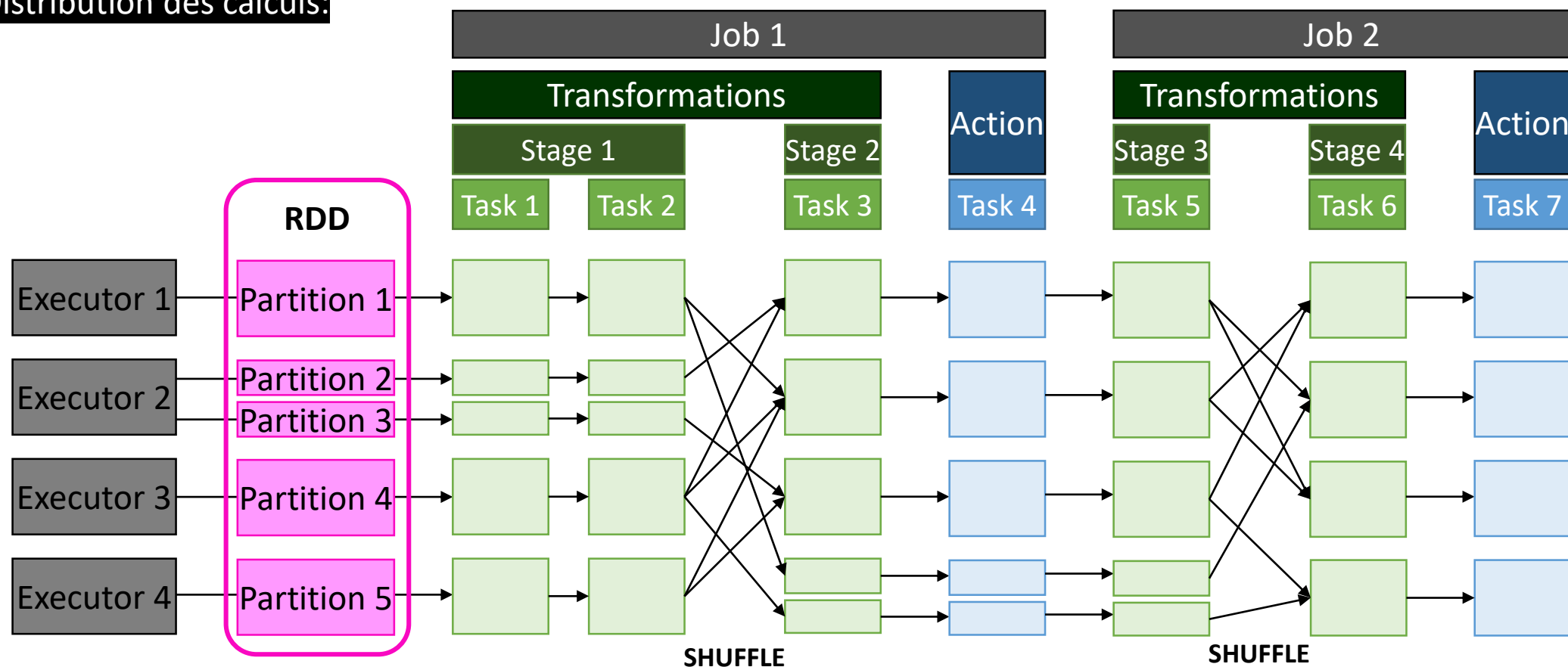
Distribution des calculs:



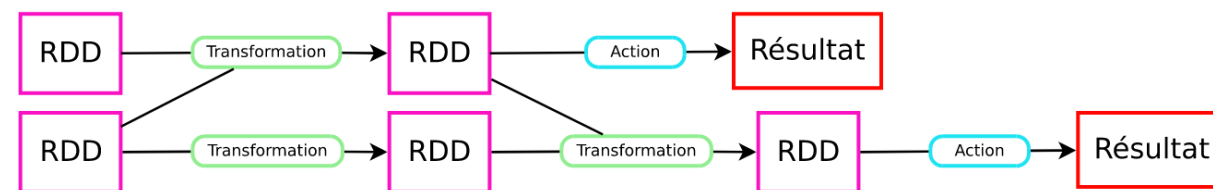
Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



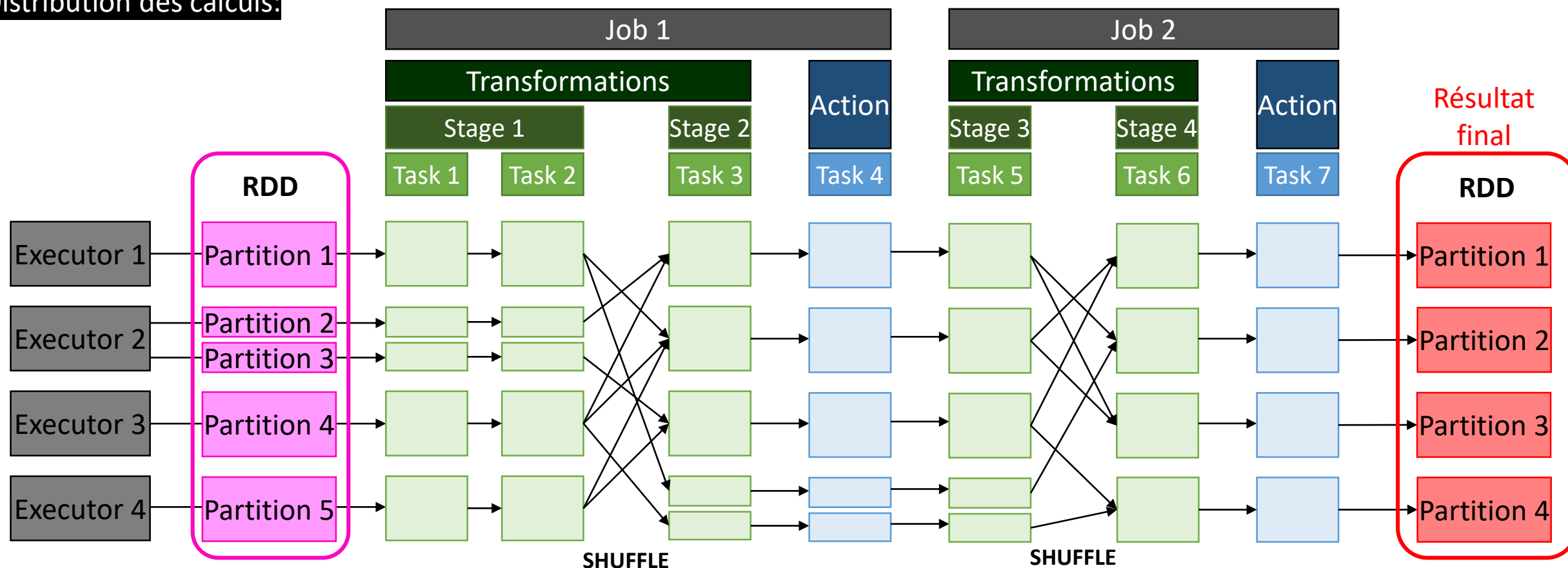
Distribution des calculs:



Données dupliquées + calculs distribués sous forme de graphe acyclique orienté : chaque **nœud** (\approx jeu de données avant/après transformation/action dessus) peut être reconstitué à partir de ses nœuds parents = **tolérance aux pannes**



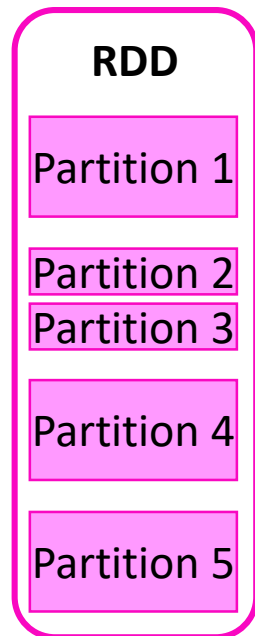
Distribution des calculs:



Formats de stockage des données

RDD = Resilient Distributed Dataset

= structure de données fondamentale de Spark, permettant de stocker des données *non structurées* sous forme d'une collection d'objets distribués sur plusieurs nœuds.



Formats de stockage des données

RDD = Resilient Distributed Dataset

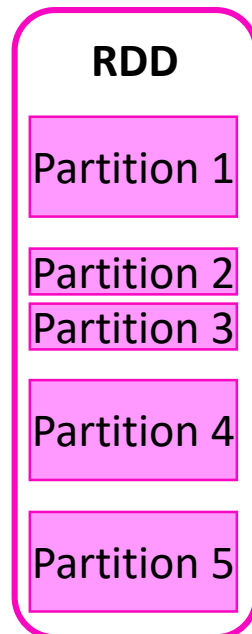
= structure de données fondamentale de Spark, permettant de stocker des données **non structurées** sous forme d'une collection d'objets distribués sur plusieurs nœuds.



Dataframe

= permet de stocker des données **structurées** sous forme d'une collection d'objets distribués organisés dans des colonnes labélisées.

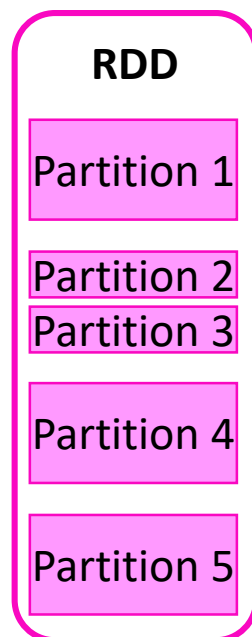
→ **plus simple à manipuler pour l'analyse de données**



Formats de stockage des données

RDD = Resilient Distributed Dataset

= structure de données fondamentale de Spark, permettant de stocker des données **non structurées** sous forme d'une collection d'objets distribués sur plusieurs nœuds.



Dataframe

= permet de stocker des données **structurées** sous forme d'une collection d'objets distribués organisés dans des colonnes labélisées.

→ **plus simple à manipuler pour l'analyse de données**

Des bibliothèques qui simplifient l'implémentation distribuée d'algorithmes d'analyse de données

Spark SQL

Spark ML
/MLlib

SparkDL

Spark Core API

Java

Scala

Python

R

SQL

BIG DATA – ENJEUX ET SOLUTIONS

Grand volume de données à analyser

Stratégie de calcul
pour les analyserGrande puissance
de calculGrande capacité de
stockage**Framework (Spark)**
de calculs distribués**Cloud (AWS):**
serveurs de stockage
et de calculs



= l'un des fournisseurs de services de cloud, proposant notamment:



amazon
EC2

Elastic Computing Cloud

= service permet de louer et gérer des **serveurs de calcul** pour y exécuter nos programmes

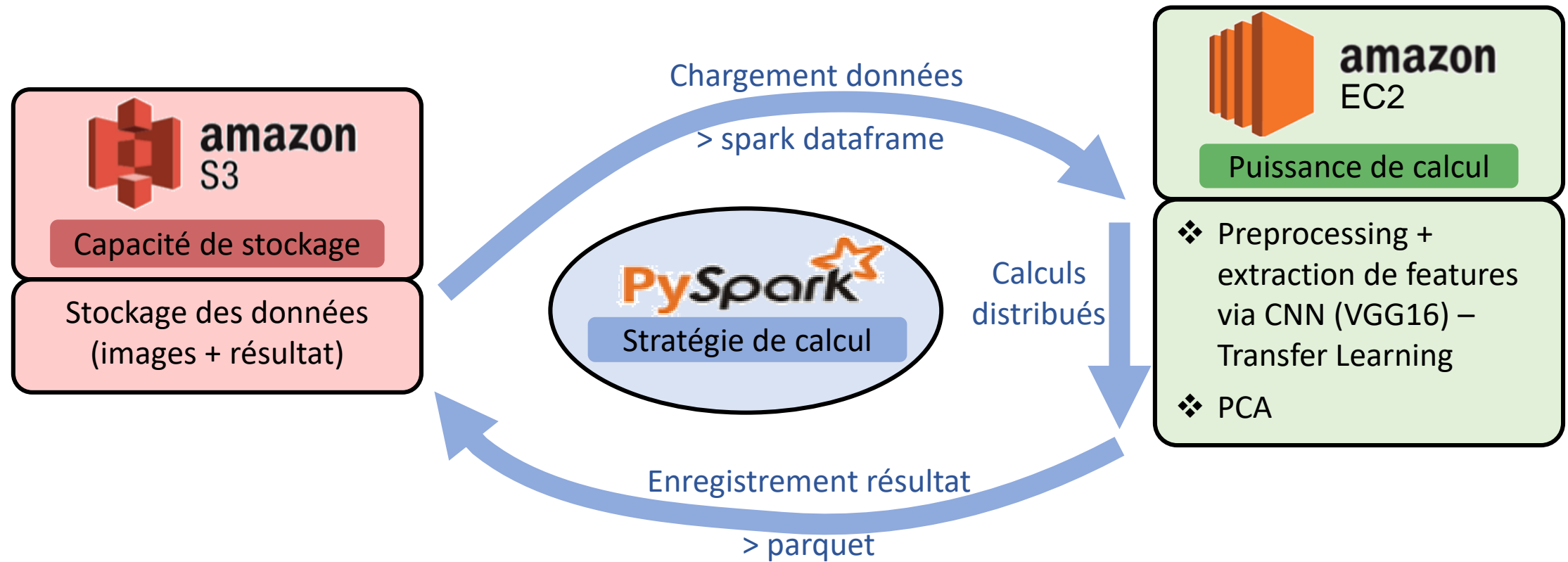


amazon
S3

Simple Storage Service

= service de **stockage** et de distribution de fichiers.

BUT = Développer dans un environnement Big Data d'une première chaîne de traitement des données qui comprendra le **preprocessing** et une étape de **réduction de dimension**.



**amazon**
EC2

Puissance de calcul

**Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0fb653ca2d3203ac1**

Éligible à l'offre gratuite
Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
Type de périphérique racine: ebs Type de virtualisation: hvm

Type d'instance

[Modifier le type d'instance](#)

Type d'instance	ECU	vCPU	Mémoire (Gio)	Stockage d'instance (Go)	Disponible en version optimisée pour EBS	Performances réseau
t2.xlarge	-	4	16	EBS uniquement 30 Go	-	Moderate

Groupes de sécurité

[Modifier les groupes de sécurité](#)

Nom du groupe de sécurité launch-wizard-1

Description launch-wizard-1 created 2022-01-17T15:38:04.820+01:00

Type	Protocole	Plage de ports	Source	Description
SSH	TCP	22	0.0.0.0/0	
Règle TCP	TCP	8888	0.0.0.0/0	

Configuration:

- Installation Java, Spark, Anaconda
- Stockage AWS credentials pour accès S3
- Environnement virtuel (avec librairies nécessaires à l'exécution des calculs)
 - Notebook jupyter → accessible sur machine local



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark

```
Entrée [1]: # Make Spark available on jupyter notebook  
import findspark  
findspark.init()  
import pyspark
```



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3

```
Entrée [2]: # In order to be able to read data via S3A we need a couple of dependencies /  
# we need to make sure the hadoop-aws and aws-java-sdk packages are available when we load spark:  
# (In a Jupyter Notebook this has to be done in the first cell (???))  
  
import os  
os.environ['PYSPARK_SUBMIT_ARGS'] = "--packages com.amazonaws:aws-java-sdk-pom:1.10.34,org.apache.hadoop:ha
```

```
Entrée [3]: # We need the aws credentials in order to be able to access the s3 bucket.  
# We can use the configparser package to read the credentials from the standard aws file.  
import configparser  
config = configparser.ConfigParser()  
config.read(os.path.expanduser("~/aws/credentials"))  
aws_profile = 'default'  
access_id = config.get(aws_profile, "aws_access_key_id")  
access_key = config.get(aws_profile, "aws_secret_access_key")  
  
os.environ["AWS_ACCESS_KEY_ID"] = access_id  
os.environ["AWS_SECRET_ACCESS_KEY"] = access_key
```



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3
3. Instanciation d'une session Spark (gestion des propriétés de notre application + chargement des données)

```
Entrée [5]: # Initiate a spark session
            from pyspark.sql import SparkSession
            spark = SparkSession.builder.master("local[*]").appName("Fruits").getOrCreate()
```



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3
3. Instanciation d'une session Spark (gestion des propriétés de notre application + chargement des données)

- #### 4. Chargement depuis S3 des données sous forme de DF Spark, au format binaire

[illegible]



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3
3. Instanciation d'une session Spark (gestion des propriétés de notre application + chargement des données)

4. Chargement depuis S3 des données sous forme de DF Spark, au format binaire
5. Extraction de la classe de fruits à partir du nom de dossier

path	content	class
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Raspberry
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Raspberry
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Raspberry
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Raspberry
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Raspberry
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Raspberry
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Apricot
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Carambula
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Carambula
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Carambula
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Carambula
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Apricot
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Carambula
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Apricot
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Apricot
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Carambula
s3a://mw-projet8/...	[FF D8 FF E0 00 1...	Apricot



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3
3. Instanciation d'une session Spark (gestion des propriétés de notre application + chargement des données)

4. Chargement depuis S3 des données sous forme de DF Spark, au format binaire
5. Extraction de la classe de fruits à partir du nom de dossier
6. Prétraitement des images + extraction de features via VGG16 (base convolutionnelle + max pooling > dimension 512)

path	feats	class
s3a://mw-projet8/...	[84.41843, 0.0, 2...	Raspberry
s3a://mw-projet8/...	[73.28788, 0.0, 5...	Raspberry
s3a://mw-projet8/...	[55.543953, 0.0, ...	Raspberry
s3a://mw-projet8/...	[58.78025, 0.0, 0...	Raspberry
s3a://mw-projet8/...	[91.687996, 0.0, ...	Raspberry
s3a://mw-projet8/...	[111.665955, 0.0, ...	Raspberry
s3a://mw-projet8/...	[22.240831, 0.0, ...	Apricot
s3a://mw-projet8/...	[29.920937, 0.0, ...	Carambula
s3a://mw-projet8/...	[87.79086, 0.0, 0...	Carambula
s3a://mw-projet8/...	[0.0, 0.0, 0.0, 0...	Carambula
s3a://mw-projet8/...	[0.0, 0.0, 0.0, 0...	Apricot
s3a://mw-projet8/...	[0.0, 0.0, 5.0133...	Carambula
s3a://mw-projet8/...	[0.0, 0.0, 0.0, 0...	Apricot
s3a://mw-projet8/...	[12.374738, 0.0, ...	Apricot
s3a://mw-projet8/...	[40.393505, 0.0, ...	Carambula
s3a://mw-projet8/...	[0.0, 0.0, 0.0, 0...	Apricot



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3
3. Instanciation d'une session Spark (gestion des propriétés de notre application + chargement des données)

4. Chargement depuis S3 des données sous forme de DF Spark, au format binaire
5. Extraction de la classe de fruits à partir du nom de dossier
6. Prétraitement des images + extraction de features via VGG16 (base convolutionnelle + max pooling > dimension 512)
7. Vectorisation des features + PCA (k=10, 91% de la variance expliquée)

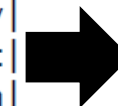
path	pcaFeats	class
s3a://mw-projet8/...	[-337.66382289951...	Raspberry
s3a://mw-projet8/...	[-396.13623237731...	Raspberry
s3a://mw-projet8/...	[-367.48886253625...	Raspberry
s3a://mw-projet8/...	[-293.92288677219...	Raspberry
s3a://mw-projet8/...	[-313.55582483546...	Raspberry
s3a://mw-projet8/...	[-238.79940081688...	Raspberry
s3a://mw-projet8/...	[140.391543232928...	Apricot
s3a://mw-projet8/...	[122.776078262433...	Carambula
s3a://mw-projet8/...	[120.739281632793...	Carambula
s3a://mw-projet8/...	[30.0916380926493...	Carambula
s3a://mw-projet8/...	[165.811282695150...	Apricot
s3a://mw-projet8/...	[55.3952460712250...	Carambula
s3a://mw-projet8/...	[170.636606334598...	Apricot
s3a://mw-projet8/...	[186.604950094278...	Apricot
s3a://mw-projet8/...	[-34.380393850124...	Carambula
s3a://mw-projet8/...	[171.020018262152...	Apricot



1. Rendre possible l'utilisation de Spark: le localiser sur l'instance EC2 avec findspark
2. Rendre possible le chargement des données depuis S3:
 - Installation des dépendances nécessaires (hadoop-aws et aws-java-sdk)
 - Lecture des clés de connexion AWS à S3
3. Instanciation d'une session Spark (gestion des propriétés de notre application + chargement des données)

4. Chargement depuis S3 des données sous forme de DF Spark, au format binaire
5. Extraction de la classe de fruits à partir du nom de dossier
6. Prétraitement des images + extraction de features via VGG16 (base convolutionnelle + max pooling > dimension 512)
7. Vectorisation des features + PCA (k=10, 91% de la variance expliquée)
8. Export vers S3 du DF Spark contenant les résultats (path + classe + features réduites en dimension 10) au format parquet

path	pcaFeats	class
s3a://mw-projet8/...	[-337.66382289951...	Raspberry
s3a://mw-projet8/...	[-396.13623237731...	Raspberry
s3a://mw-projet8/...	[-367.48886253625...	Raspberry
s3a://mw-projet8/...	[-293.92288677219...	Raspberry
s3a://mw-projet8/...	[-313.55582483546...	Raspberry
s3a://mw-projet8/...	[-238.79940081688...	Raspberry
s3a://mw-projet8/...	[140.391543232928...	Apricot
s3a://mw-projet8/...	[122.776078262433...	Carambula
s3a://mw-projet8/...	[120.739281632793...	Carambula
s3a://mw-projet8/...	[30.0916380926493...	Carambula
s3a://mw-projet8/...	[165.811282695150...	Apricot
s3a://mw-projet8/...	[55.3952460712250...	Carambula
s3a://mw-projet8/...	[170.636606334598...	Apricot
s3a://mw-projet8/...	[186.604950094278...	Apricot
s3a://mw-projet8/...	[-34.380393850124...	Carambula
s3a://mw-projet8/...	[171.020018262152...	Apricot



Amazon S3 > mw-projet8

mw-projet8 [Info](#)

[Objets](#) [Propriétés](#) [Autorisations](#)

Objets (3)

Les objets sont les entités fondamentales stockées dans votre bucket. Ils peuvent être des fichiers ou des dossiers. Les autres personnes peuvent accéder à vos objets si vous leur avez accordé des permissions.

[Charger](#)

[Copier l'URI S3](#)

Rechercher des objets en fonction de...

<input type="checkbox"/>	Nom
<input type="checkbox"/>	Data/
<input type="checkbox"/>	Sample/
<input type="checkbox"/>	SampleResult.parquet/

Conclusion:*Rappel: MISSION*

Développer dans un **environnement Big Data** d'une première chaîne de traitement des données qui comprendra le **preprocessing** et une étape de **réduction de dimension**.



- **AWS S3** Capacité de stockage
 - **AWS EC2** Puissance de calcul
 - **CNN (VGG16) - Transfert learning**
 - **PCA**
- } **PySpark**
Stratégie de calcul

Perspectives:

- Développement avec spark2 pour pouvoir utiliser sparkDL
- Optimisation de l'extraction de features pour la classification:
 - Mise en place d'une classification simple (type k-means)
 - Test d'autres CNN et/ou autres méthodes d'extraction de features
 - Test sur images « brutes » (e.g. dataset « test-multiple_fruits »)
- Utilisation d'un cluster d'instances pour accompagner le passage à l'échelle

