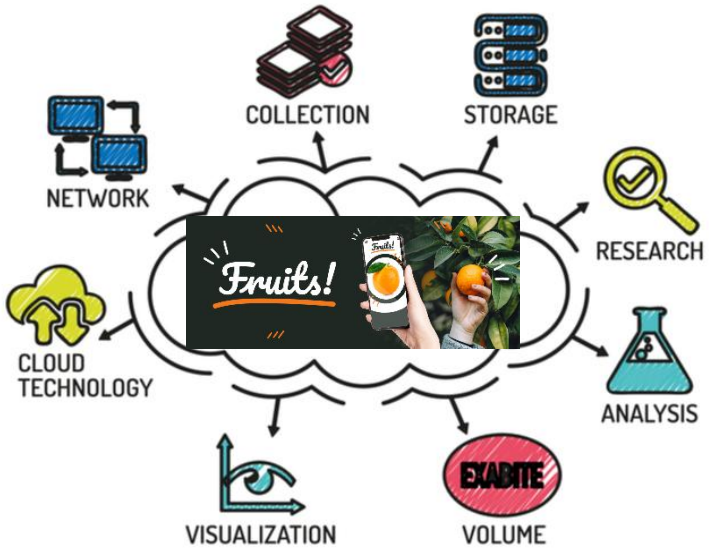
Réalisez un traitement dans un environnement Big Data sur le Cloud



Présenté par : Monsieur Djamel Ferguen
Le xx novembre 2025
Évaluateur(e) : Monsieur/Madame xxx XXX

Sommaire

1-Introduction Générale

Compréhension de la problématique métier

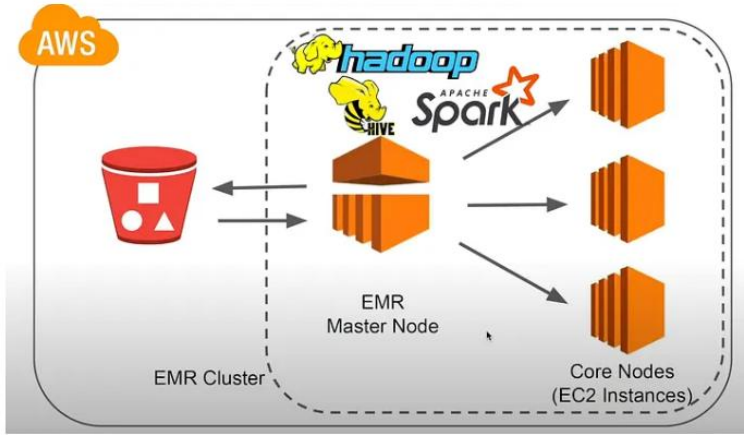
La start-up de l'AgriTech **Fruits** veut proposer une application mobile qui permet aux utilisateurs d'obtenir des informations (nom, espèce, origine probable, informations de biodiversité) sur les photos des fruits photographier.
L'objectif de ce projet est de **concevoir et tester une architecture Big Data** capable de traiter un **grand volume d'images** efficacement. Et de construire la première brique du moteur de classification d'images.
L'enjeu principal est de **comparer les performances entre un environnement local** (sous WSL/Linux) **et un environnement cloud** (AWS EMR avec PySpark), afin d'évaluer l'apport réel du **calcul distribué**.

La mission du ce projet est de :

- Mise en place de deux environnements de travail, WSL Ubuntu en local et AWS EMR dans le cloud
- Construire une chaîne de traitement PySpark pour le dataprocessing , extraction des features et sauvegarde des résultats
- Comparer les performances et la scalabilité , temps de calcul, coût de traitement, hébergement,…

2-Environnement technique & données

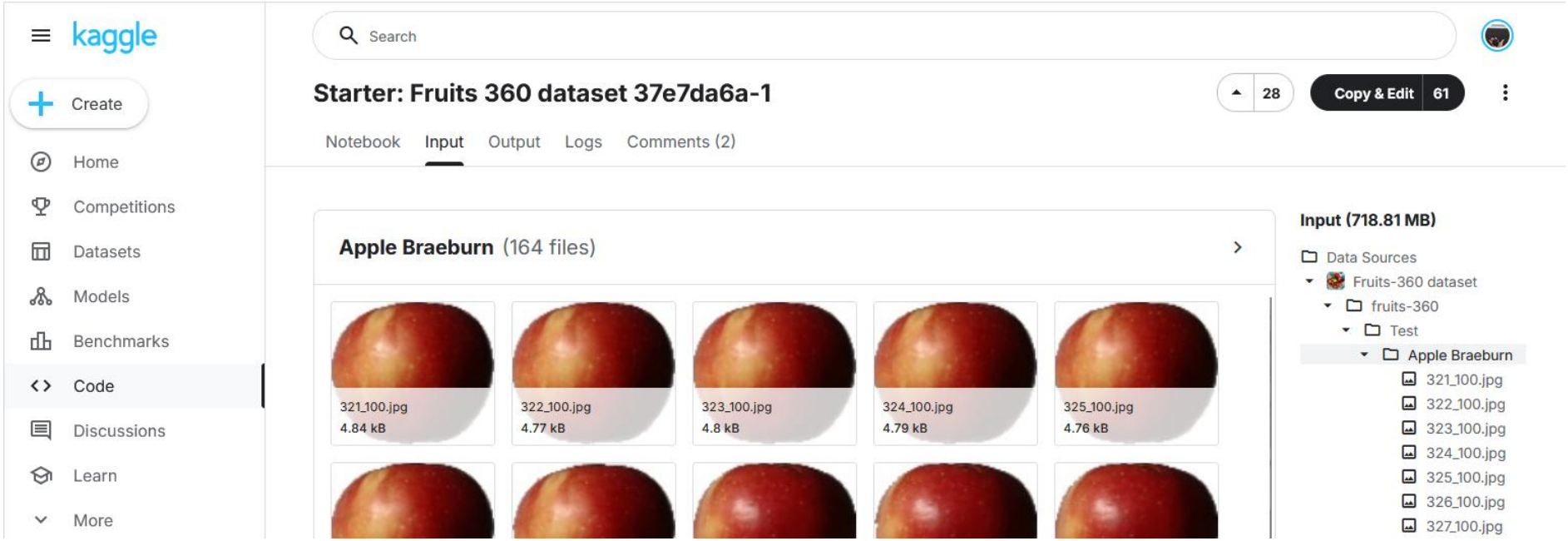
Environnement



- **Outils et langage :** Python 3.10 & PySpark 3.5, ,Jupyter Notebook via EMR
- **Librairies :** pyspark.ml, numpy, pandas, tensorflow.keras, boto3
- **Infrastructure Cloud AWS :** Amazon S3, EMR,EC2, Spark History Server, Rôles IAM
- **Sécurité et conformité :** Données publiques eu-west-3

2-Environnement technique & données

Jeu de données



- Le jeu de données utilisé est un dataset d'images de **fruits** contenant plusieurs miliers d'images réparties en classes : pommes, bananes, oranges, etc.
- En local, on a réduit la taille du dataset a 5928 photos pour éviter la surcharge mémoire
- Pour la partie AWS, sur le cluster EMR, on a utilisé le jeu complet avec 22 668 photos. Elles sont stockées sur **S3**, pour être accessibles à Spark lors de l'exécution sur EMR.
- « Le dataset est volontairement volumineux pour illustrer les différences de performance entre le traitement local et distribué. »

3- Processus de création de l'environnement de travail

Comparaison Local vs Cloud

Avant de passer à la configuration, ci-dessous : la comparaison des deux approches :

•En **local**, le traitement est séquentiel ou pseudo-parallèle sur un seul nœud, limité par la RAM et les cœurs CPU.

•En **cloud**, le traitement est distribué sur plusieurs nœuds grâce à EMR, ce qui permet d'exécuter des calculs en parallèle sur des volumes massifs »

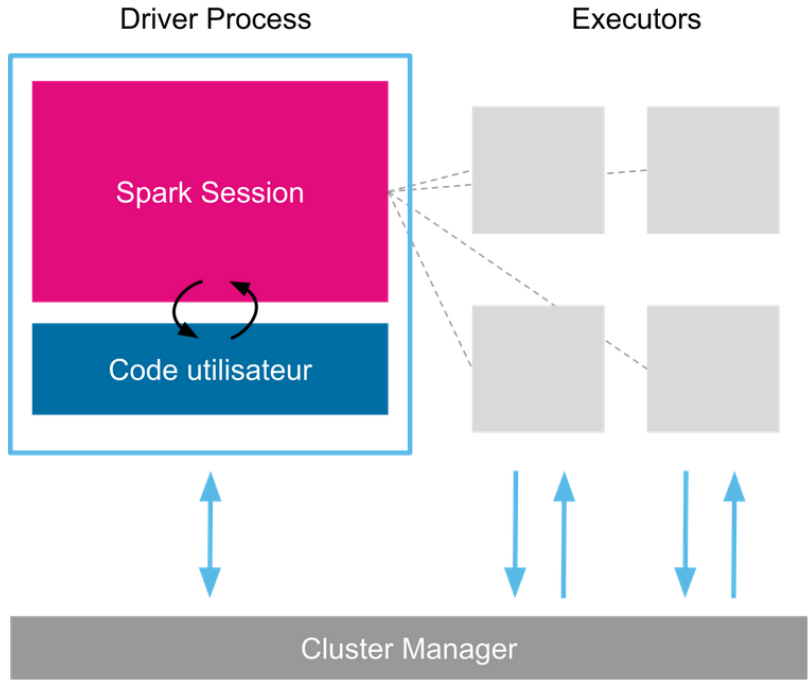
« L'idée est d'avoir un code identique, mais exécuté dans deux environnements différents. »

Caractéristique	**Environnement local (WSL)**	**Environnement Cloud (AWS EMR)**
OS / Base	Ubuntu 22.04 via WSL2 (Windows 11)	Amazon Linux 2023 (EMR 6.x)
Matériel / Instance	CPU : Intel Core i5-3320M (2 cœurs) , RAM : 16 Go, SSD	EC2 m5.large (2 vCPU, 8 Go RAM) x 2 nœuds
Gestion des paquets et Spark	venv + pip + Spark 3.5.2 (installé manuellement)	Conda préconfiguré EMR + Spark 3.X.X (intégré EMR)
Stockage	SSD local /home/djamel/Module_11/data/	Amazon S3 (s3://module11-results/)
Interface et Packages	Jupyter Notebook , pyspark, numpy, tensorflow, scikit-image, python-multipart	pyspark, boto3, s3fs, numpy, pandas, tensorflow, keras, scikit-image
Sécurité/IAM et connectivité	Accés local	Rôles IAM pour EMR et S3 + accés SSH via PEM

3- Processus de création de l'environnement de travail

Architecture locale

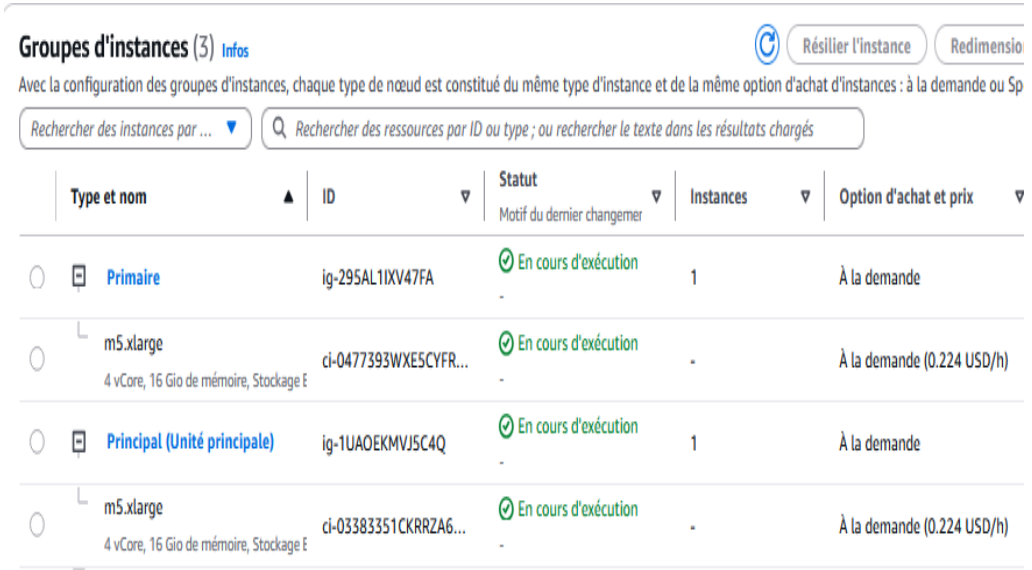
•**PySpark** est une interface Python du framework **Apache Spark**, conçue pour traiter efficacement de **grands volumes de données** grâce au **calcul distribué**.
•En **local**, Spark fonctionne en **mode "standalone"** : une seule machine joue à la fois le rôle du **driver** et des **exécuteurs**, simulant ainsi un cluster miniature.
•Le **driver (SparkSession)** planifie et distribue les tâches, tandis que chaque **exécuteur** (processus JVM) exécute une partie du traitement sur les données.
•Ce mécanisme permet d'optimiser l'utilisation du CPU et de la mémoire même sur une machine locale, avant de passer à une architecture **multi-nœuds sur le cloud**.
•Dans ce projet, le mode local a permis de **valider le pipeline PySpark** (prétraitement, extraction des features) avant de le **déployer sur AWS EMR** pour des performances réelles à grande échelle.



3- Processus de création de l'environnement de travail

Architecture Cloud AWS

- **AWS EMR (Elastic MapReduce) :** une plateforme managée optimisée pour exécuter Spark sur un cluster distribué.
- Chaque nœud du cluster joue un rôle précis : le driver orchestre les tâches, tandis que les workers répartissent le calcul entre plusieurs machines virtuelles**.**
- Les données sont stockées sur **Amazon S3**, un service de stockage scalable et persistant, assurant rapidité d'accès et sécurité.
- La configuration de l'environnement s'effectue via **AWS CLI** et **IAM**, garantissant un contrôle automatisé des ressources et des permissions.
- Cette architecture cloud permet un **traitement parallèle massif**, une **scalabilité automatique** et une **réduction significative du temps de calcul** sur de grands volumes d'images.




```
#!/bin/bash
set -xe  # Affiche toutes les commandes et erreurs pour le debug

# ----------------------------
# Mise à jour des outils pip/setuptools
# ----------------------------
sudo python3 -m pip install --upgrade setuptools
sudo python3 -m pip install --upgrade --ignore-installed pip

# ----------------------------
# Installation des packages Python nécessaires
# ----------------------------
PACKAGES=(
    wheel
    pillow
    pandas==1.2.5
    numpy
    pyarrow
    boto3
    s3fs
    fsspec
)

for pkg in "${PACKAGES[@]}"; do
    sudo python3 -m pip install "$pkg"
done

# ----------------------------
# Vérification
# ----------------------------
echo "Installation terminée. Versions installées :"
python3 -m pip show pip setuptools wheel pillow pandas numpy pyarrow boto3 s3fs fsspec
```

3- Processus de création de l'environnement de travail

Détails des composantes AWS IAM/S3

1. IAM (Identity and Access Management) :

•C’est le **service de gestion des identités et des accès** d’AWS.

•La création des rôles et des politiques d’autorisation et de communication entre les services

•**Autorise le cluster EMR à lire et écrire sur le bucket S3**

•Sécurisation des **communication** entre les services internes AWS.

2. Amazon S3 (Simple Storage Service) :c’est un système de fichiers centralisé, il stocke :

- Les **datasets bruts** (images d’entrée)
- Les **résultats intermédiaires**
- Les **logs Spark** produits lors de l’exécution du job sur EMR.

•Contrairement à EC2, **les données sur S3 sont gardées a la fin du cluster**.

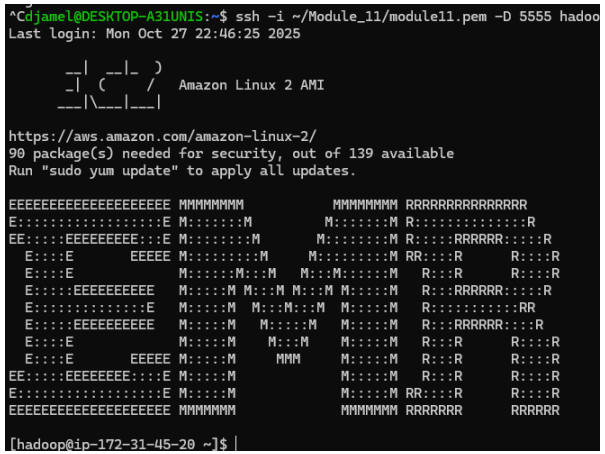
•Le bucket est accessible via un chemin du type s3://p8-data-djamel/.

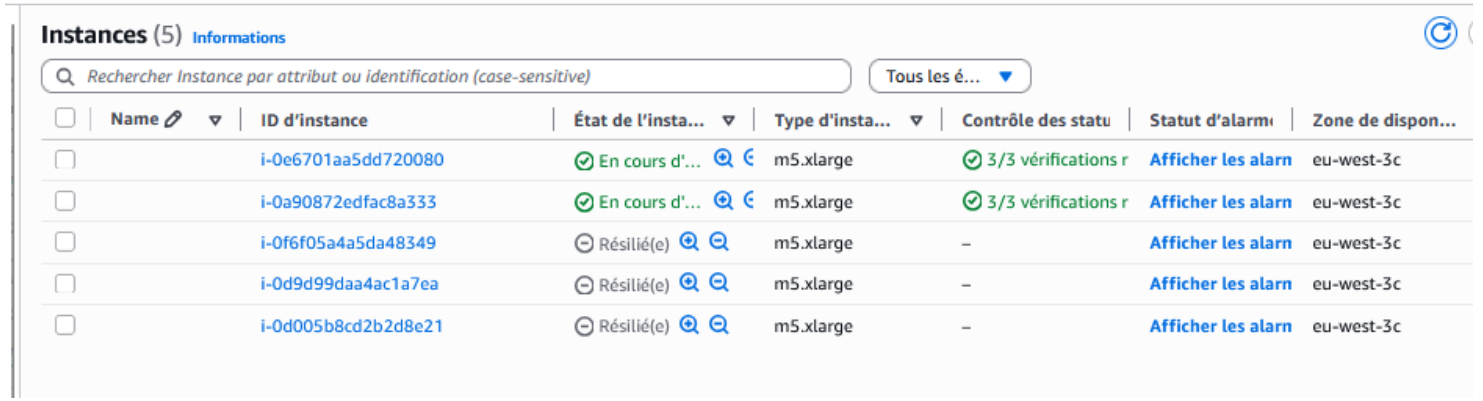
3- Processus de création de l'environnement de travail

Détails des composantes AWS EMR/EC2

3. Amazon EMR (Elastic MapReduce) : une **plateforme managée pour exécuter Spark, Hadoop ou d'autres frameworks Big Data** sur des clusters de machines EC2.
Chaque cluster EMR contient :Un **nœud maître (driver)** : qui planifie et supervise les jobs Spark.
Des **nœuds esclaves (workers)** : qui exécutent les tâches en parallèle.
EMR s'occupe automatiquement de l'installaytion des dépendances, répartitiion des calculs et scalabilité automatiques selon la charge du travail

4. EC2 (Elastic Compute Cloud) : Les **instances EC2** sont les **machines virtuelles physiques** qui composent le cluster EMR.
Chaque instance dispose d'une configuration matérielle spécifique (RAM, vCPU, stockage).
On peut choisir le type d'instance (ex. m5.xlarge) selon les besoins de performance.
Ces instances exécutent le code PySpark sur plusieurs cœurs en parallèle.





3- Processus de création de l'environnement de travail

Détails des composantes AWS RGPD et coût

- L’un des aspects clés de notre architecture était la conformité au Règlement Général sur la Protection des Données (RGPD).
- Toutes les données sont hébergées sur Amazon S3 dans la région Paris (eu-west-3), garantissant leur localisation sur le territoire européen.
- Les accès sont strictement contrôlés via IAM, limitant les permissions aux utilisateurs et services nécessaires.
- Ce dispositif global garantit la confidentialité, l’intégrité et la conformité légale des traitements de données dans le cloud AWS.
- Les données utilisées ont une **finalité exclusivement pédagogique et expérimentale**.
  Aucune **donnée personnelle** n’est traitée ni stockée.
- Les données peuvent être **supprimées à tout moment** et leur **durée d’utilisation est strictement limitée à la durée du projet**.

- Le coût horaire estimé du cluster EMR avec 3 nœuds (EC2 + EMR, région eu-west-3) est d’environ **1,44 USD/heure**.

4- Réalisation de la chaine de traitement de données

Chargement des images

```
# Chemin du bucket
PATH = 's3a://p8-data-djamel/jupyter/hadoop/data'

# Chemin des données et des résultats
PATH_Data = PATH + '/Test1'
PATH_Result = PATH + '/Results'

# Affichage pour vérification
print('PATH:        ' + PATH +
      '\nPATH_Data:   ' + PATH_Data +
      '\nPATH_Result: ' + PATH_Result)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',
PATH:        s3a://p8-data-djamel/jupyter/hadoop/data
PATH_Data:   s3a://p8-data-djamel/jupyter/hadoop/data/Test1
PATH_Result: s3a://p8-data-djamel/jupyter/hadoop/data/Results
```

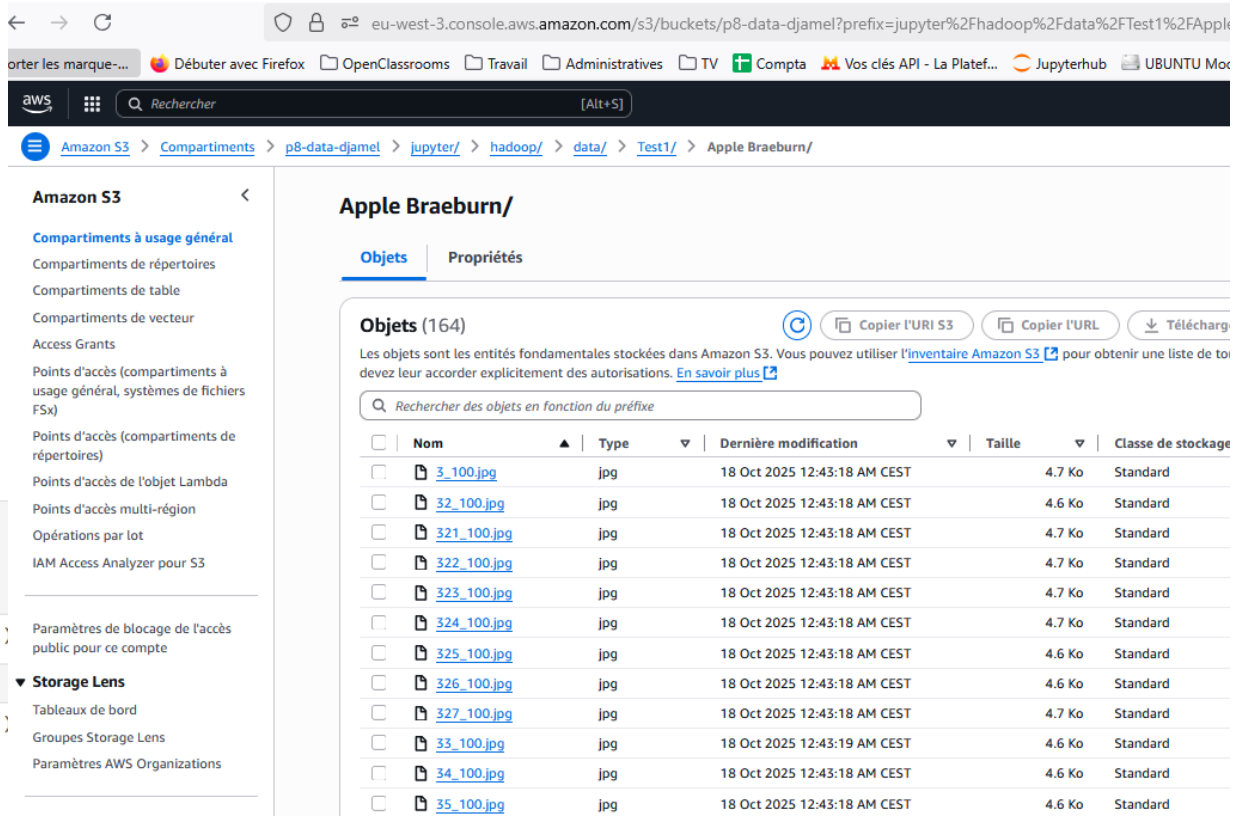
6.10.5.1. 4.10.5.1 Chargement des données

```
images = spark.read.format("binaryFile") \
  .option("pathGlobFilter", "*.jpg") \
  .option("recursiveFileLookup", "true") \
  .load(PATH_Data)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'
```

```
images.show(5)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'
+--------------------+-------------------+------+--------------------+
|                path|   modificationTime|length|             content|
+--------------------+-------------------+------+--------------------+
|s3a://p8-data-dja...|2025-10-17 22:49:14|  7353|[FF D8 FF E0 00 1...|
|s3a://p8-data-dja...|2025-10-17 22:49:14|  7350|[FF D8 FF E0 00 1...|
|s3a://p8-data-dja...|2025-10-17 22:49:14|  7349|[FF D8 FF E0 00 1...|
|s3a://p8-data-dja...|2025-10-17 22:49:14|  7348|[FF D8 FF E0 00 1...|
|s3a://p8-data-dja...|2025-10-17 22:49:14|  7328|[FF D8 FF E0 00 1...|
+--------------------+-------------------+------+--------------------+
only showing top 5 rows
```



4- Réalisation de la chaine de traitement de données

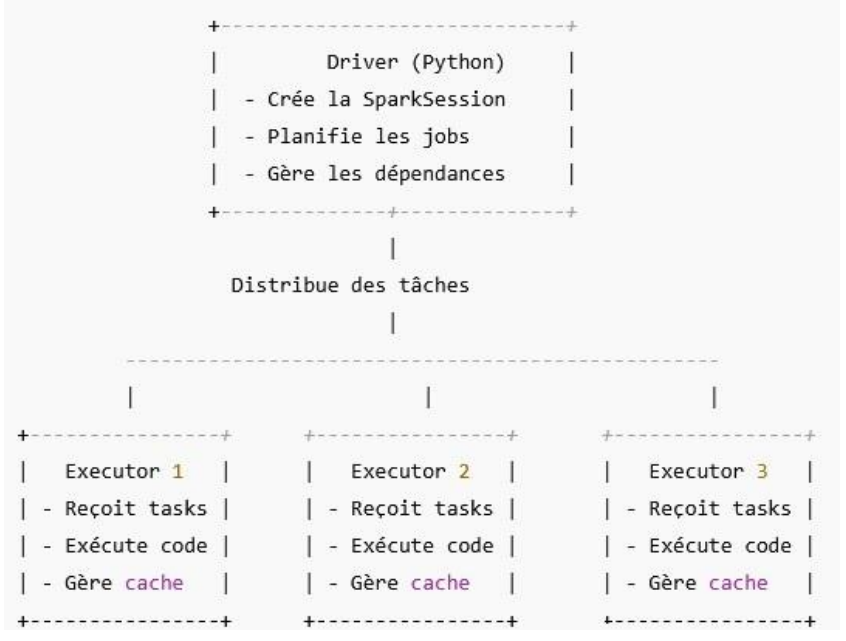
Extraction des features avec MoileNetV2

1. Construction d'un feature extractor MobileNetV2 et broadcaste ses poids via Spark pour les partager aux executors.

Bloc / Couche principale	Type (simplifié)	Sortie (Output Shape)	Paramètres	Remarques
Input / Conv / BN / ReLU	Convolution initiale	(112, 112, 32)	~1K	Première extraction de features
Blocks 1–3	Depthwise + Pointwise Conv	(28, 28, 32)	~20K	Réduction spatiale, 1er niveau
Blocks 4–6	Depthwise Separable Conv	(14, 14, 64)	~40K	Expansion et normalisation
Blocks 7–12	Inverted Residual Blocks	(14, 14, 96)	~160K	Profondeur intermédiaire
Blocks 13–15	Inverted Residual + Downsampling	(7, 7, 160)	~300K	Compression vers feature map finale
Global + Classification	Global AvgPool + Dense	(1, 1, nb_classes)	~1M	Sortie finale du modèle

2.model_fn() : reconstruit le modèle sur chaque executor, et lui charge les poids broadcastés, garantissant des instances identiques sur tous les nœuds (24).

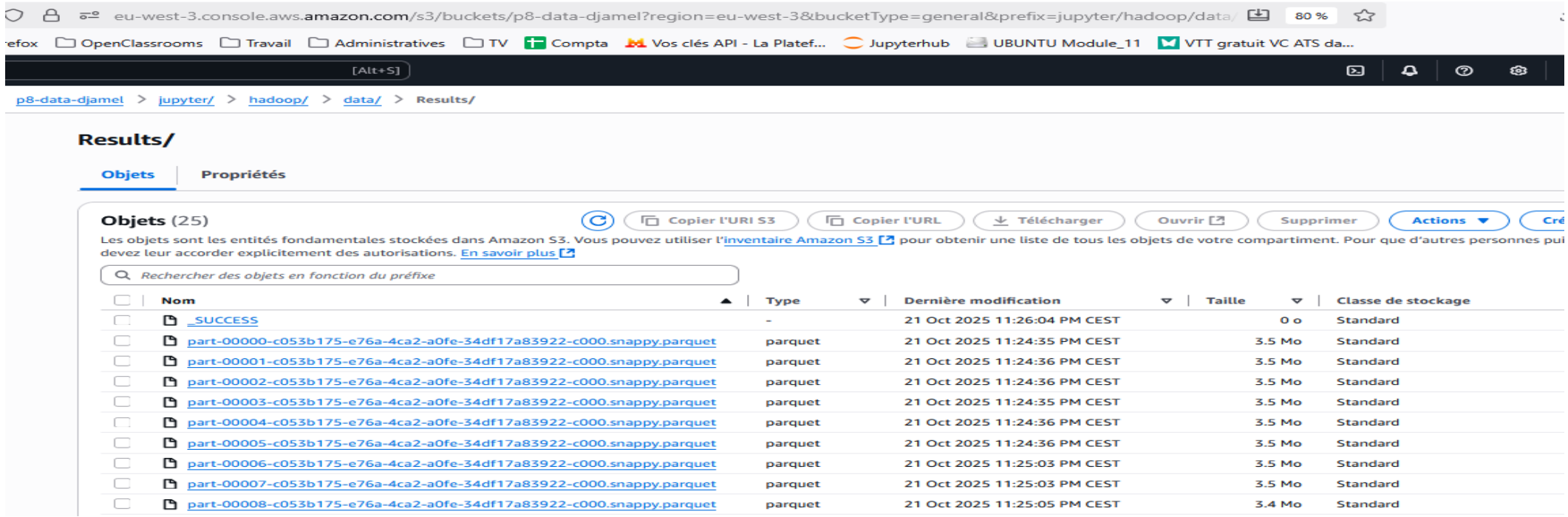
```
|label      |   Features
|
+-----------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------+
|Watermelon |[-3.2145721693585583,5.199837854611351,-5.565745749572388,-4.604803325795239,6.2699719351748575,5.5501799774354454,2.2567232494851837,0.2775
284069498621,-9.197860779543712,0.30289325255917393] |
|Watermelon |[-3.0128234346440395,5.5752400554596475,-6.705040189144154,-5.509112833030747,4.922637869185867,5.535166124420963,-0.11080750761254124,1.871
3580086560297,-11.466360383344364,1.9122219857337244]|
|Watermelon |[-2.74194594419241,4.634202580734169,-7.679576234444822,-4.622096483223681,2.9259509840729603,6.04816817836768,-1.8328910074136375,4.4035230
10780686,-7.868629899352176,3.6315681762651932]      |
|Watermelon |[-3.144155352265274,4.4336217900185995,-6.408481436407661,-5.147563712481808,5.529013635893022,5.798278390532121,-3.050901153672843,4.225713
6907943575,-8.162293993065697,4.446560434689849]     |
|Cauliflower|[-3.86911506646346,2.0796112452411264,2.08482015548923,-2.6016443049197173,-3.416411381487945,2.315963606584788,-0.7273207647494081,1.581431
0925565946,-1.9699532648144298,2.987484478702239]    |
+-----------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```



4- Réalisation de la chaine de traitement de données

Extraction des features avec MoileNetV2

3. preprocess() : prépare les images pour produire des vecteurs 1D par image.
4.featurize_udf : est une **pandas UDF SCALAR_ITER** qui charge le modèle une seule fois par processus worker et applique la featurisation
5.L'appel features_df.write.parquet(PATH_Result) : déclenche l'évaluation distribuée (exécution de l'UDF sur partitions) et écrit les vecteurs en Parquet, c'est l'action qui lance tout le calcul et l'I/O.

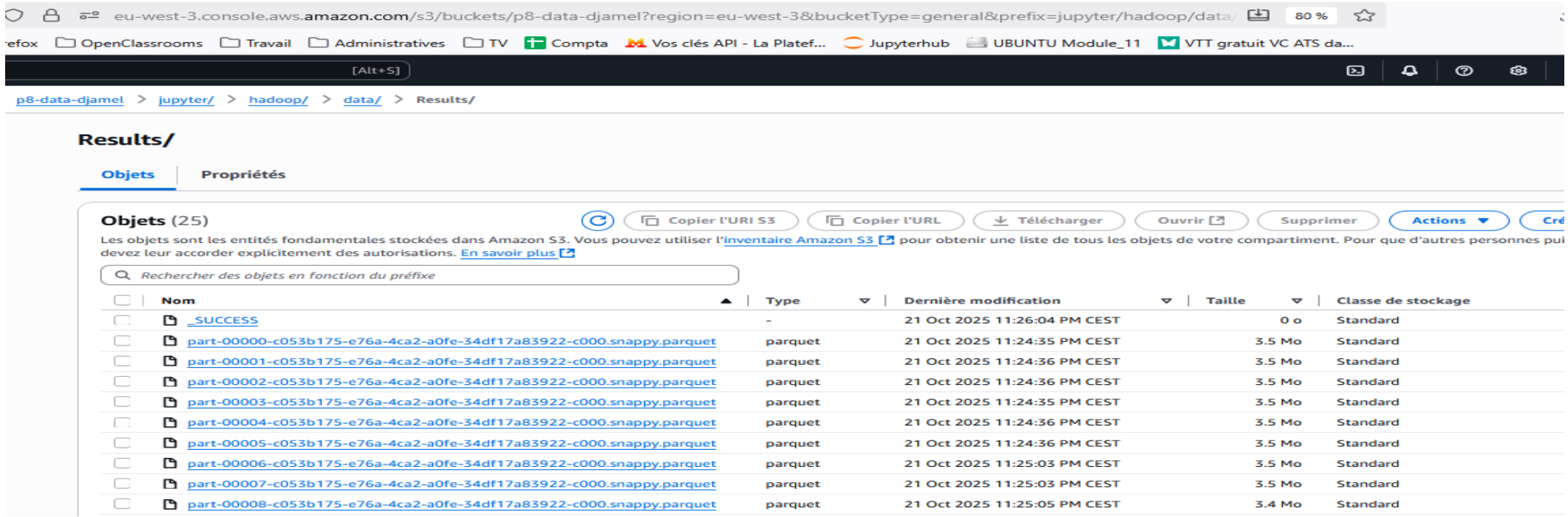


1.Pourquoi seulement 3 colonnes ?

4- Réalisation de la chaine de traitement de données

Extraction des features avec MoileNetV2

3. preprocess() : prépare les images pour produire des vecteurs 1D par image.
4.featurize_udf : est une **pandas UDF SCALAR_ITER** qui charge le modèle une seule fois par processus worker et applique la featurisation
5.L'appel features_df.write.parquet(PATH_Result) : déclenche l'évaluation distribuée (exécution de l'UDF sur partitions) et écrit les vecteurs en Parquet, c'est l'action qui lance tout le calcul et l'I/O.



1.Pourquoi seulement 3 colonnes ?

4- Réalisation de la chaine de traitement de données

Extraction des features avec MoileNetV2

A	B	C
Column1	Column2	Column3
path	label	features_PCA
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Pineapple Mini/79_100.jpg	Pineapple Mini	[-5.19052493201195, 2.673250096270365, 2.4239688282339142, -3.898000923826278, -1.945709
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Watermelon/269_100.jpg	Watermelon	[-3.456503252100896, 4.5416792166161395, -7.138640610434084, -4.857468606217179, 4.29239
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Watermelon/188_100.jpg	Watermelon	[-4.0498776070617675, 6.886111823713373, -6.762385090721115, -5.014435997594039, 3.59493
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Cauliflower/r_176_100.jpg	Cauliflower	[-4.593787921959655, 2.347825025189629, 0.8815296197283519, -2.929708441982677, -3.54368
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Raspberry/210_100.jpg	Raspberry	[-1.7246584077531661, 3.6858307180515992, 1.186423633118103, -8.713059382070714, -4.6291
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Pineapple/311_100.jpg	Pineapple	[-5.215521359841995, 5.664221265541015, 0.8794791153786552, -3.534723540208323, -3.93030
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Raspberry/104_100.jpg	Raspberry	[0.04772453897630575, 5.599549807343923, 2.3487211650722846, -7.9160978906028285, -3.368
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Pineapple/193_100.jpg	Pineapple	[-6.1915798577919405, 5.4987280682305455, -0.011867692543329303, -4.51496210025523, -3.23
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Pineapple Mini/r_17_100.jpg	Pineapple Mini	[-6.311538005194658, 6.775812285689277, 0.16235825348515354, -3.7785663279248403, -2.063
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Pineapple Mini/r_164_100.jpg	Pineapple Mini	[-4.219393062930552, 2.713759003157621, 0.3718717640774511, -2.54803578738543, -1.656009
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Cucumber Ripe/r_204_100.jpg	Cucumber Ripe	[-0.9958924049487277, 6.93779585152625, -2.764951758913287, -6.148718850701011, 7.418536
s3a://p8-data-djamel/jupyter/hadoop/data/Test1/Pear 2/r_255_100.jpg	Pear 2	[3.890981708941182, -4.447643442214211, -8.482720838698755, -6.784440585357592, 7.435206

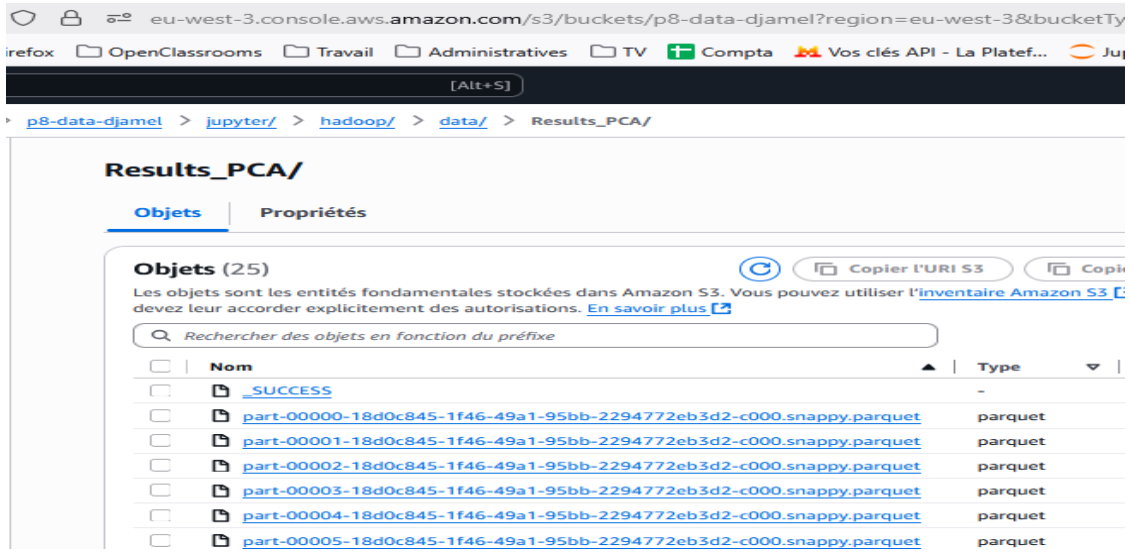
- Les données d'images sont transformées en vecteurs de caractéristiques numériques, regroupant plusieurs centaines de valeurs par image.
- Dans le DataFrame Spark, ces valeurs sont stockées dans une seule colonne de type Vector, accompagnée du chemin et du label de l'image.
- Ainsi, le tableau final affiche seulement trois colonnes visibles — chemin, label et vecteur de features — même si chaque vecteur contient plus de 1000 dimensions.

4- Réalisation de la chaine de traitement de données

Réduction de dimension avec PCA

•Pour réduire la complexité, on applique une réduction de dimension PCA, avec K= 15 dimensions **en conserve 72% de la variance.**

•**Le résultat est un DataFrame Spark réduit, beaucoup plus compact avec un temps de calcul de 300 secondes**



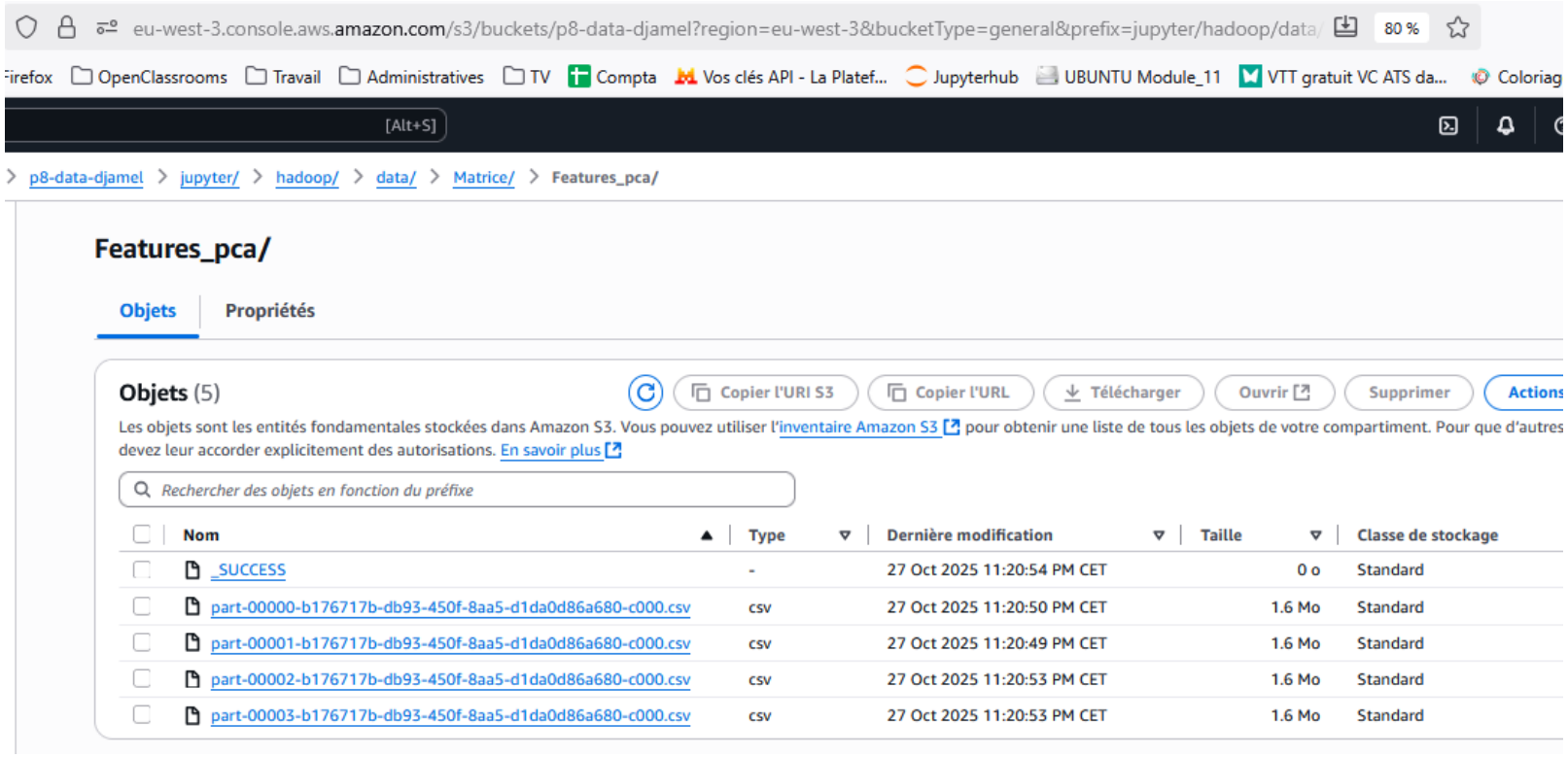
•La PCA nécessite un **calcul lourd** : covariance + décomposition en valeurs propres sur tout le dataset.

•En environnement distribué, elle entraîne des **échanges réseau et des opérations de shuffle** entre nœuds Spark.

•Le **gain de performance** vient **après**, lors des traitements suivants sur les données réduites.

4- Réalisation de la chaine de traitement de données

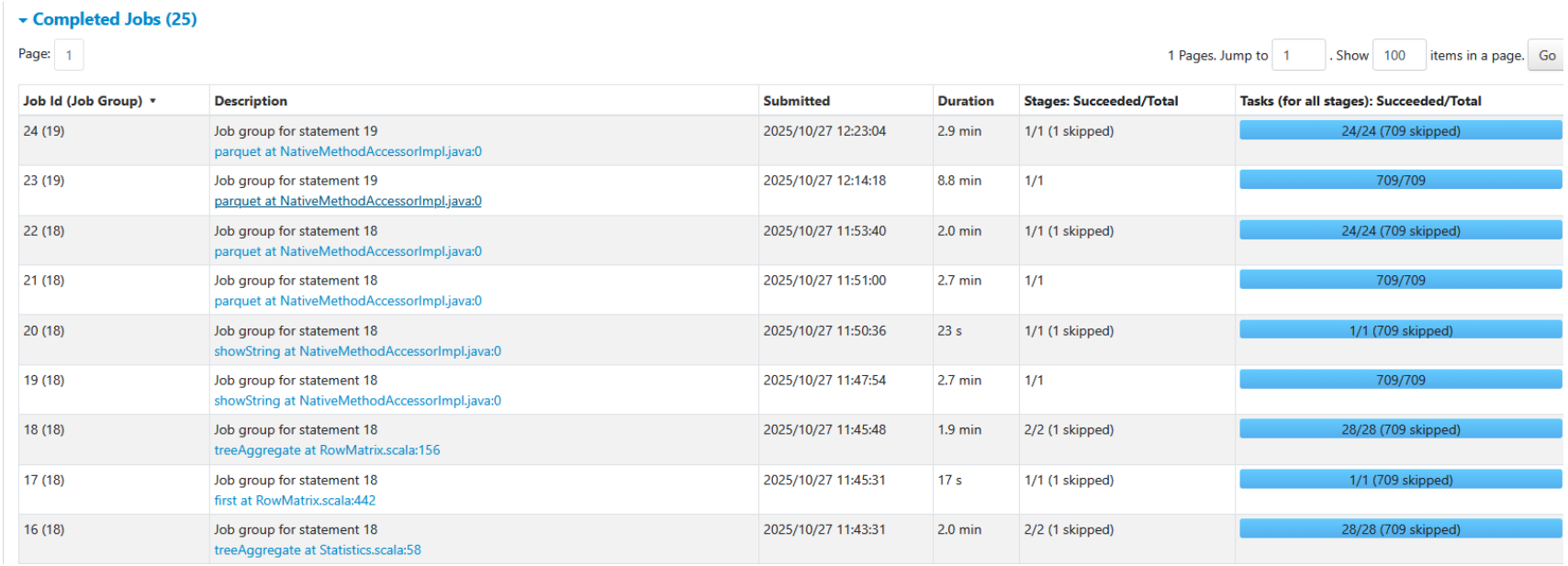
Export CSV



- **Objectif :** Le fichier CSV contient les caractéristiques extraites des images après PCA. Il sert à stocker les données prêtes pour l'analyse ou l'entraînement de modèles, sans avoir à recharger toutes les images brutes.
- **Contenu :**Chaque ligne représente une image du jeu de données initial, identifiée par son nom ou son chemin, suivie des composantes principales issues du PCA (par exemple, pc1, pc2, pc3, ...).
- **Usage dans le cloud :**Le CSV est hébergé dans un espace de stockage cloud S3

4- Réalisation de la chaine de traitement de données

Serveur d'Historique Spark



▾ Completed Jobs (25)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
24 (19)	Job group for statement 19 parquet at NativeMethodAccessorImpl.java:0	2025/10/27 12:23:04	2.9 min	1/1 (1 skipped)	24/24 (709 skipped)
23 (19)	Job group for statement 19 parquet at NativeMethodAccessorImpl.java:0	2025/10/27 12:14:18	8.8 min	1/1	709/709
22 (18)	Job group for statement 18 parquet at NativeMethodAccessorImpl.java:0	2025/10/27 11:53:40	2.0 min	1/1 (1 skipped)	24/24 (709 skipped)
21 (18)	Job group for statement 18 parquet at NativeMethodAccessorImpl.java:0	2025/10/27 11:51:00	2.7 min	1/1	709/709
20 (18)	Job group for statement 18 showString at NativeMethodAccessorImpl.java:0	2025/10/27 11:50:36	23 s	1/1 (1 skipped)	1/1 (709 skipped)
19 (18)	Job group for statement 18 showString at NativeMethodAccessorImpl.java:0	2025/10/27 11:47:54	2.7 min	1/1	709/709
18 (18)	Job group for statement 18 treeAggregate at RowMatrix.scala:156	2025/10/27 11:45:48	1.9 min	2/2 (1 skipped)	28/28 (709 skipped)
17 (18)	Job group for statement 18 first at RowMatrix.scala:442	2025/10/27 11:45:31	17 s	1/1 (1 skipped)	1/1 (709 skipped)
16 (18)	Job group for statement 18 treeAggregate at Statistics.scala:58	2025/10/27 11:43:31	2.0 min	2/2 (1 skipped)	28/28 (709 skipped)

- Chaque **Job Spark** correspond à une opération majeure (lecture, PCA, écriture Parquet…).
- Le **Spark History Server** permet de **visualiser la durée, la progression et le succès** de chaque tâche.
- On peut y **analyser les performances du cluster**, identifier les goulets d'étranglement et optimiser les traitements futurs.
- Exemple : l'écriture du Parquet features_df.write.mode("overwrite") a pris environ **5 minutes**, tandis que la PCA complète (~710 tâches) a nécessité environ **5 minutes**

4- Réalisation de la chaine de traitement de données

Comparaison

Environnement	Taille du Dataset (photos)	Temps total de traitement fichiers .parquet	Remarques
Local	5928	500 secondes	Exécution séquentielle, forte charge CPU
EMR Cluster	22688	300 secondes	Traitement parallèle efficace
EMR Cluster PCA	22688	300 + 1000 secondes	PCA légèrement coûteuse, mais meilleure compacité

- En exécution locale, le traitement complet (chargement, extraction et écriture des features) est beacoup plus lent et limité en ressources CPU/RAM
- Sur EMR (Elastic MapReduce),le calcul est distribué sur plusieurs nœuds, ce qui permt un gain significatif en temps d'excution global
- L'application de la PCA ajoute une étape de calcul supplémentaire qui augmente temporairement le temps total. Cependant, la PCA réduit considérablement la taille des données finales, ce qui accélère les traitements ultérieurs (stockage, transfert, apprentissage)

4- Réalisation de la chaine de traitement de données

Démonstration

7- Conclusion Générale

- Le projet a permis de concevoir une chaîne complète de traitement d'images Big Data basée sur PySpark, depuis le chargement jusqu'à la sauvegarde distribuée.
- Le déploiement sur Amazon EMR a démontré la scalabilité et la robustesse du calcul distribué face à un grand volume d'images.
- L'exécution EMR a engendré un coût de calcul maîtrisé, proportionnel au temps d'exécution et à la taille du cluster utilisé.
- La configuration des rôles et politiques IAM a assuré un accès sécurisé et contrôlé aux ressources AWS, renforçant la fiabilité de l'infrastructure.
- L'intégration de la PCA a permis d'évaluer l'impact de la réduction de dimension sur la performance du traitement.
- Le projet s'est inscrit dans une démarche conforme aux principes RGPD, en manipulant uniquement des données non personnelles et anonymisées.
- L'ensemble constitue une base solide pour le déploiement d'applications IA distribuées, sécurisées et conformes aux exigences industrielles modernes.

Merci pour votre attention