Projet 9
Réalisez un traitement dans un environnement Big Data sur le Cloud

Guille Anaïs – Parcours Data Scientist

Mentor : Ahmed Tidiane Balde

Sommaire

I- Problématique

- ***Fruits!*** : Start-up de l'AgriTech qui cherche à proposer des solutions innovantes pour la récolte des fruits
- ***Objectif*** : Création d'une application mobile pour obtenir des informations sur un fruit à partir d'une photo.

Compléter le notebook de l'alternant avec une étape de réduction de dimension en Pyspark

☐ *Migrer la chaîne de traitement dans le Cloud AWS*

II- Présentation du jeu de données

- 90483 images (100x100p) dont **Test** = 22688 images
- 131 classes (labels)

Exemple : *Apple Braeburn*

➢ Le volume de données va rapidement augmenté après la livraison du projet
➢ Définir l'architecture Big Data requise

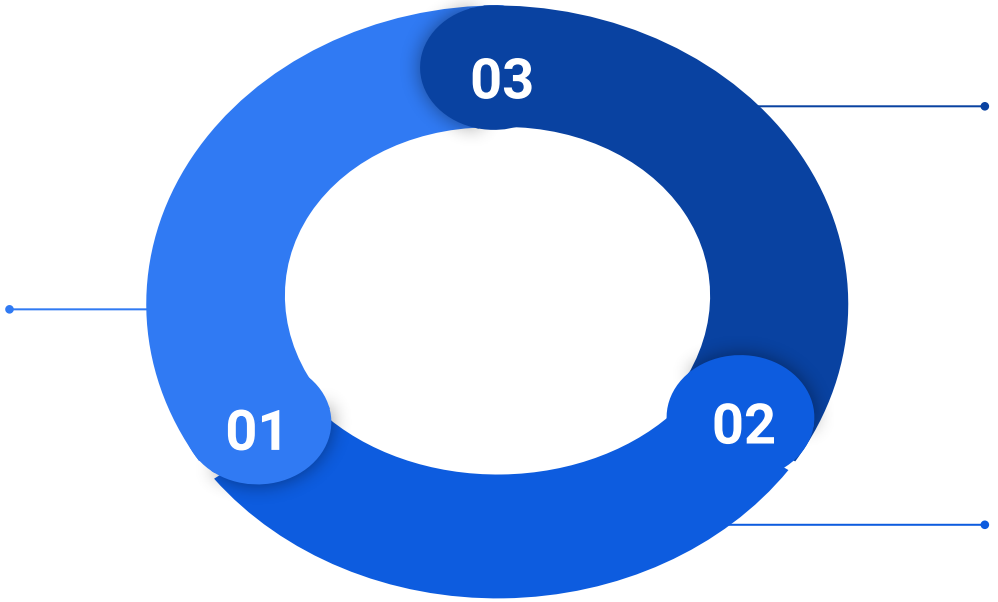
III- Architecture Big Data

1) Introduction au Big Data

- **Big Data** : ensemble des données massives, variées et générées à grande vitesse, nécessitant des technologies avancées pour leur traitement et leur analyse.

Volume

Quantité énorme de données produites. Par exemple, chaque seconde, des millions de message sont postés sur les réseaux sociaux.



Vélocité

Vitesse à laquelle les données sont générées et doivent être traitées (transaction financières analysées en temps réel pour détecter les fraudes potentielles)

Variété

Diversité des formats des données (textes, images, vidéos, sons, emails ...)

- **Technologies utilisées :** : Hadoop, Spark, NoSQL ...

III- Architecture Big Data

2) Apache Spark

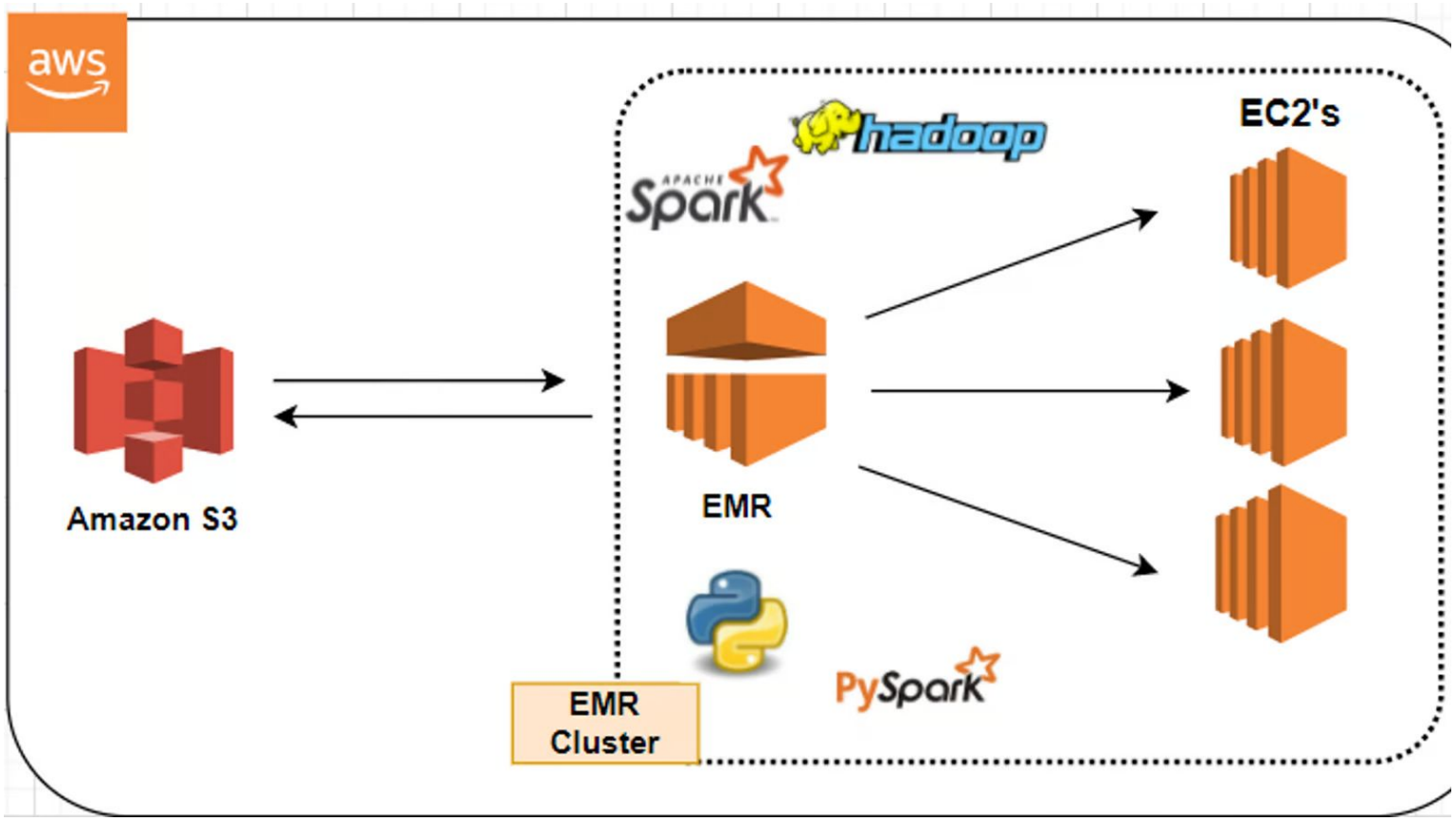
- **Apach Spark** : Framework de calcul distribué open source, conçu spécifiquement pour le traitement rapide et le calcul distribué sur de grands ensembles de données.

- **Calculs distribués** : Distribution des tâches en plusieurs noeuds d'un cluster permettant une augmentation significative des performances

- **Map Reduce :** Modèle de programmation qui simplifie le traitement de grandes quantitées de données en deux phases principales

IV- Création de l'environnement Big Data

1) Infrastructure



IV- Création de l'environnement Big Data

2) Méthodologie

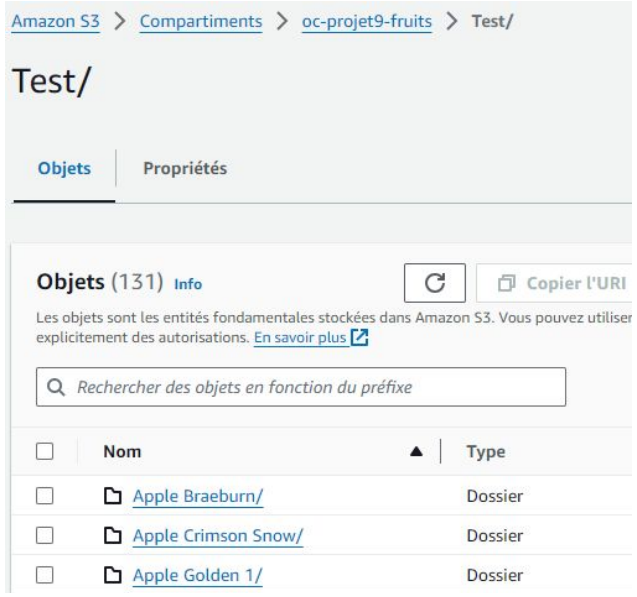
1) Installation de AWS Cli (Interface en ligne de commande) pour l'interaction avec les services

```
(projet9) anaisgll@DESKTOP-KURGKIN:~$ aws configure
AWS Access Key ID [**********KCSD]:
AWS Secret Access Key [**********FoFn]:
Default region name [eu-west-3]:
Default output format [None]:
```

➔ Identifiants de sécurité du compte

➔ Paris

2) Création du Bucket S3 'oc-projet9-fruits' et dépôt des images

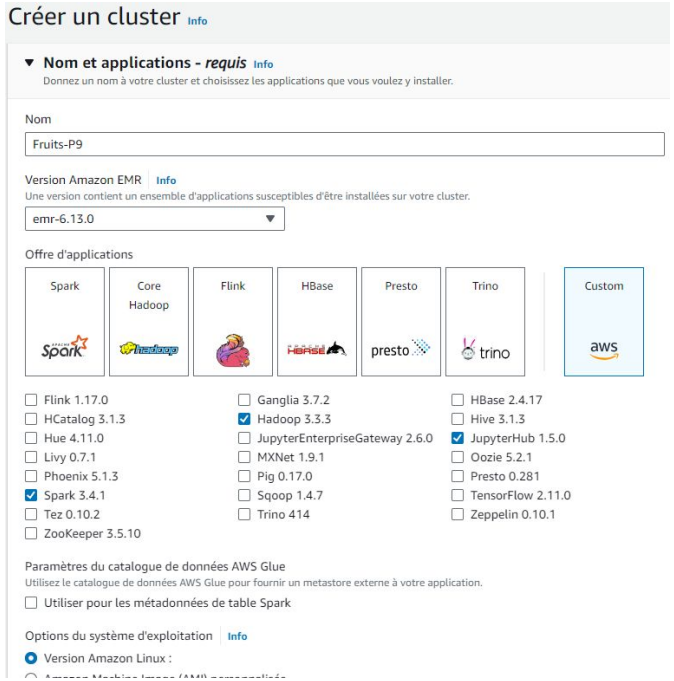


s3://oc-projet9-fruits/Test/

IV- Création de l'environnement Big Data

2) Méthodologie

3) Création du cluster EMR



4) Location d'instances EC2

5) Ajout des librairies nécessaires

Fichier bootstrap comme action d'amorçage

```
#!/bin/bash
sudo python3 -m pip install -U setuptools
sudo python3 -m pip install -U pip
sudo python3 -m pip install wheel
sudo python3 -m pip install pillow
sudo python3 -m pip install pandas==1.2.5
sudo python3 -m pip install pyarrow
sudo python3 -m pip install boto3
sudo python3 -m pip install s3fs
sudo python3 -m pip install fsspec
sudo python3 -m pip install matplotlib
sudo python3 -m pip install -U tensorflow
```

IV- Création de l'environnement Big Data

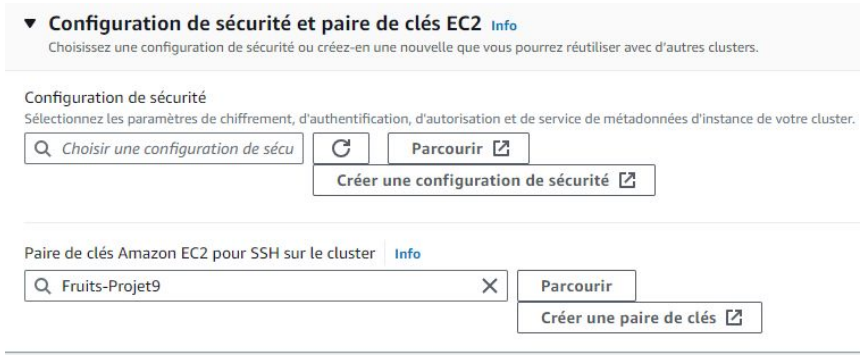
2) Méthodologie

6) Paramètres logiciel

Persistance des données utilisées ou générées par jupyter

7) Configuration de la sécurité

Paire de clés privées/publique pour une connexion sécurisée via tunnel SSH



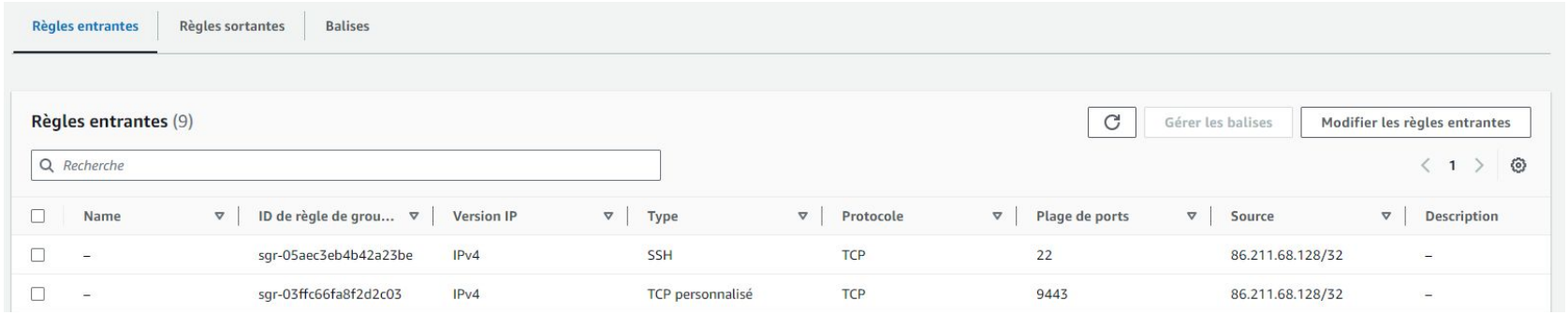
8) Création de rôle spécifique IAM pour notre cluster et nos instances

IV- Création de l'environnement Big Data

2) Méthodologie

9) Autorisation d'écoute des tunnels SSH

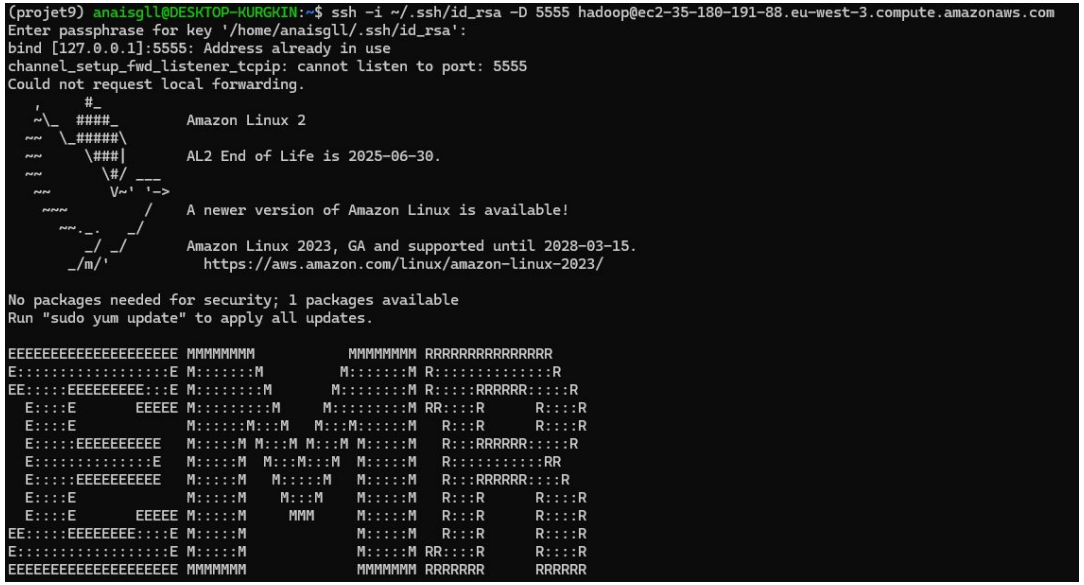
Groupe de sécurité ElasticMapReduce-master



IV- Création de l'environnement Big Data

2) Méthodologie

10) Connexion à l'EMR



11) Paramétrage et installation de FoxyProxy

IV- Création de l'environnement Big Data

2) Méthodologie

12) Ouverture de JupyterHub

Interfaces utilisateur d'application sur le nœud primaire

Celles-ci nécessitent l'activation du tunneling SSH.

Application	URL de l'interface utilisateur
Gestionnaire de ressources	http://ec2-35-180-191-88.eu-west-3.compute.amazonaws.com:8088/
JupyterHub	https://ec2-35-180-191-88.eu-west-3.compute.amazonaws.com:9443/
Nom du nœud HDFS	http://ec2-35-180-191-88.eu-west-3.compute.amazonaws.com:9870/
Serveur d'historique Spark	http://ec2-35-180-191-88.eu-west-3.compute.amazonaws.com:18080/

13) Importation du Jupyter Notebook et ouverture avec un kernel Pyspark

V- Traitement des images

1) Démarrage de la session Spark

4.10.1 Démarrage de la session Spark

```
# L'exécution de cette cellule démarre l'application Spark
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
0	application_1720686246056_0001	pyspark	idle	Link	Link	None	✓

2) Importation des librairies nécessaires

4.10.3 Import des librairies

```
import pandas as pd
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
import io
import os
import tensorflow as tf
from PIL import Image
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras import Model
from pyspark.sql.functions import col, pandas_udf, PandasUDFType, element_at, split
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.ml.feature import StandardScaler, PCA
from pyspark.sql.types import ArrayType, FloatType
```

3) Définition des chemins d'accès pour le chargement des images et 'enregistrement des résultats

4.10.4 Définition des PATH pour charger les images et enregistrer les résultats

Nous accédons directement à nos **données sur S3** comme si elles étaient **stockées localement**.

```
PATH = 's3://oc-projet9-fruits'
PATH_Data = PATH+'/Test'
PATH_Result = PATH+'/Results'
print('PATH:        '+\
      PATH+'\nPATH_Data:    '+\
      PATH_Data+'\nPATH_Result: '+PATH_Result)
```

V- Traitement des images

4) Chargement des images

4.10.5.1 Chargement des données

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

A Jupyter widget could not be displayed because the widget state could not be found. This or if the widget state was not saved in the notebook. You may be able to create the widget

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

A Jupyter widget could not be displayed because the widget state could not be found. This or if the widget state was not saved in the notebook. You may be able to create the widget

```
+--------------------+--------------------+------+--------------------+
|                path|    modificationTime|length|             content|
+--------------------+--------------------+------+--------------------+
|s3://oc-projet9-f...|2024-07-03 13:34:09|  7353|[FF D8 FF E0 00 1...|
|s3://oc-projet9-f...|2024-07-03 13:34:09|  7350|[FF D8 FF E0 00 1...|
|s3://oc-projet9-f...|2024-07-03 13:34:09|  7349|[FF D8 FF E0 00 1...|
|s3://oc-projet9-f...|2024-07-03 13:34:09|  7348|[FF D8 FF E0 00 1...|
|s3://oc-projet9-f...|2024-07-03 13:34:10|  7328|[FF D8 FF E0 00 1...|
+--------------------+--------------------+------+--------------------+
only showing top 5 rows
```

5) Extraction des labels

```
images = images.withColumn('label', element_at(split(images['path'], '/'),-2))
print(images.printSchema())
print(images.select('path','label').show(5,False))
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
root
 |-- path: string (nullable = true)
 |-- modificationTime: timestamp (nullable = true)
 |-- length: long (nullable = true)
 |-- content: binary (nullable = true)
 |-- label: string (nullable = true)

None
+--------------------------------------------------+----------+
|path                                              |label     |
+--------------------------------------------------+----------+
|s3://oc-projet9-fruits/Test/Watermelon/r_106_100.jpg|Watermelon|
|s3://oc-projet9-fruits/Test/Watermelon/r_109_100.jpg|Watermelon|
|s3://oc-projet9-fruits/Test/Watermelon/r_108_100.jpg|Watermelon|
|s3://oc-projet9-fruits/Test/Watermelon/r_107_100.jpg|Watermelon|
|s3://oc-projet9-fruits/Test/Watermelon/r_95_100.jpg |Watermelon|
+--------------------------------------------------+----------+
only showing top 5 rows

None
```

V- Traitement des images

6) Préparation du modèle

```
def model_fn():
    """
    Returns a MobileNetV2 model with top layer removed
    and broadcasted pretrained weights.
    """
    model = MobileNetV2(weights='imagenet',
                        include_top=True,
                        input_shape=(224, 224, 3))
    for layer in model.layers:
        layer.trainable = False
    new_model = Model(inputs=model.input,
                      outputs=model.layers[-2].output)
    new_model.set_weights(brodcast_weights.value)
    return new_model
```

Chargement du modèle, dernière couche incluse

Désactivation de l'apprentissage pour toutes les couches

Création d'un modèle sans la dernière couche

Diffusion des poids du nouveau modèle

```
# Redimension des images en 224x224 pixels (au lieu de 100x100)
def preprocess(content):
    """
    Preprocesses raw image bytes for prediction.
    """
    img = Image.open(io.BytesIO(content)).resize([224, 224])
    arr = img_to_array(img)
    return preprocess_input(arr)

# Obtention des caractéristiques des images sous formes de pd.Series après prédiction par le modèle
def featurize_series(model, content_series):
    """
    Featurize a pd.Series of raw images using the input model.
    :return: a pd.Series of image features
    """
    input = np.stack(content_series.map(preprocess))
    preds = model.predict(input)
    # For some layers, output features will be multi-dimensional tensors.
    # We flatten the feature tensors to vectors for easier storage in Spark DataFrames.
    output = [p.flatten() for p in preds]
    return pd.Series(output)

# Traitement par lot d'image en série
@pandas_udf('array<float>', PandasUDFType.SCALAR_ITER)
def featurize_udf(content_series_iter):
    '''
    This method is a Scalar Iterator pandas UDF wrapping our featurization function.
    The decorator specifies that this returns a Spark DataFrame column of type ArrayType(FloatType).

    :param content_series_iter: This argument is an iterator over batches of data, where each batch
                                is a pandas Series of image data.
    '''
    # With Scalar Iterator pandas UDFs, we can load the model once and then re-use it
    # for multiple data batches.  This amortizes the overhead of loading big models.
    model = model_fn()
    for content_series in content_series_iter:
        yield featurize_series(model, content_series)
```

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

Redimensionnement des images

Conversion en array numpy

Stockage des prédictions dans la variable *preds*

Applatissement des caractéristiques en une liste de vecteur de caractéristiques retournée sous forme de pd.Series

Utilisation d'un Pandas UDF du type SCALAR_ITER : permet de traiter plusieurs lots de données à la fois, en chargeant le modèle une seule fois pour tous les lots
→ Amélioration des performances lors du traitement de grandes quantités de données

V- Traitement des images

7) Extraction des features

Notre jeu de données de **Test** contient **22819 images**.

```
features_df = images.repartition(24).select(col("path"),
                                            col("label"),
                                            featurize_udf("content").alias("features")
                                           )
```

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

Rappel du PATH où seront inscrits les fichiers au format "**parquet**"
contenant nos résultats, à savoir, un DataFrame contenant 3 colonnes :

1. Path des images
2. Label de l'image
3. Vecteur de caractéristiques de l'image

```
print(PATH_Result)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
s3://oc-projet9-fruits/Results
```

Enregistrement des données traitées au format "**parquet**" :

```
features_df.write.mode("overwrite").parquet(PATH_Result)
```

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

V- Traitement des images

8) Réalisation de la PCA avec 200 composantes principales

```
# Définir une fonction UDF pour convertir la colonne "features" en vecteur dense
array_to_dense_vector_udf = udf(lambda arr: Vectors.dense(arr), VectorUDT())

# Appliquer la fonction UDF pour créer une nouvelle colonne "dense_features_vector"
features_df = features_df.withColumn("dense_features_vector", array_to_dense_vector_udf(features_df["features"]))

# Créer un objet PCA avec 200 composantes principales pour réduire la dimensionnalité
pca = PCA(k=200, inputCol="dense_features_vector", outputCol="pca_features_vector")

# Ajuster le modèle PCA sur le DataFrame et transformer les données
pca_model = pca.fit(features_df)
pca_transformed_df = pca_model.transform(features_df)

# Calculer et afficher la variance expliquée par chaque composante principale
explained_variance = pca_model.explainedVariance.toArray()
cumulative_variance = np.cumsum(explained_variance)
print("Variance cumulée expliquée par les 200 premières composantes :", cumulative_variance[-1])
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no l
or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Variance cumul?e expliqu?e par les 200 premi?res composantes : 0.9073574507253982
```

```
# Sélectionner les colonnes pertinentes et afficher les premières lignes
selected_columns_df = pca_transformed_df.select("path", "label", "pca_features_vector")
selected_columns_df.show(5, truncate=True)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no l
or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
+--------------------+-------------+--------------------+
|                path|        label| pca_features_vector|
+--------------------+-------------+--------------------+
|s3://oc-projet9-f...|   Watermelon|[-3.1104719900338...|
|s3://oc-projet9-f...|   Watermelon|[0.36305058561334...|
|s3://oc-projet9-f...|   Watermelon|[-3.3317759899948...|
|s3://oc-projet9-f...|Pineapple Mini|[-5.9637450317861...|
|s3://oc-projet9-f...|   Watermelon|[-3.3636914195690...|
+--------------------+-------------+--------------------+
only showing top 5 rows
```

V- Traitement des images

9) Restructuration des données après la PCA

```
def vector_to_array(vec):
    '''
    Convertir un vecteur en array
    '''
    return vec.toArray().tolist()

# Créer une UDF pour convertir un vecteur en array
vector_to_array_udf = udf(vector_to_array, ArrayType(FloatType()))

# Appliquer la fonction UDF pour créer une nouvelle colonne "pca_features"
final_df = selected_columns_df.withColumn("pca_features", vector_to_array_udf("pca_features_vector"))

# Sélectionner les colonnes pertinentes et afficher les premières lignes
final_df = final_df.select("path", "label", "pca_features")
final_df.show(5, truncate=True)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing th
or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
+--------------------+--------------+--------------------+
|                path|         label|        pca_features|
+--------------------+--------------+--------------------+
|s3://oc-projet9-f...|    Watermelon|[-1.9238999, 4.31...|
|s3://oc-projet9-f...|    Watermelon|[-2.514709, 4.320...|
|s3://oc-projet9-f...|    Watermelon|[-2.2832384, 4.27...|
|s3://oc-projet9-f...|Pineapple Mini|[-5.963745, 4.434...|
|s3://oc-projet9-f...|    Watermelon|[-3.0519247, 3.89...|
+--------------------+--------------+--------------------+
only showing top 5 rows
```

```
final_df.printSchema()
```

A Jupyter widget could not be displayed because the widg
or if the widget state was not saved in the notebook. You r

```
root
 |-- path: string (nullable = true)
 |-- label: string (nullable = true)
 |-- pca_features: array (nullable = true)
 |    |-- element: float (containsNull = true)
```

```
# Localisation des résultats
print(PATH_Result)

# Enregistrement des données au format 'parquet'
final_df.write.mode("overwrite").parquet(PATH_Result)
```

A Jupyter widget could not be displayed because the widget state co
happen if the kernel storing the widget is no longer available, or if the
in the notebook. You may be able to create the widget by running the

```
s3://oc-projet9-fruits/Results
```

V- Traitement des images

10) Chargement des données et validation des résultats

Chargement de données :

```
df = pd.read_parquet(PATH_Result, engine='pyarrow')
```

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

```
print(f'Dimension de df : {df.shape}')
df.head()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
Dimension de df : (22688, 3)
                                                path  ...                                       pca_features
0  s3://oc-projet9-fruits/Test/Watermelon/r_77_10...  ...  [-3.110472, 5.793163, -5.0599937, -3.8391795, ...
1  s3://oc-projet9-fruits/Test/Watermelon/r_181_1...  ...  [0.36305058, 2.873297, -9.020796, -4.2290845, ...
2  s3://oc-projet9-fruits/Test/Watermelon/r_59_10...  ...  [-3.331776, 4.6731443, -6.0287094, -4.575021, ...
3  s3://oc-projet9-fruits/Test/Watermelon/259_100...  ...  [-3.0519247, 3.896526, -4.8615804, -4.664655, ...
4  s3://oc-projet9-fruits/Test/Cauliflower/r_183_...  ...  [-4.6894894, 2.6404457, 1.2758789, -2.3358068,...

[5 rows x 3 columns]
```

Validation de la dimension du vecteur de caractéristiques des image (200)

```
df.loc[0,'pca_features'].shape
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
(200,)
```

Création d'une colonne pour chaque composante tout en conservant la colonne d'origine

```
# Création d'une liste de colonnes à ajouter
new_columns = []

# Ajout des 200 composantes du vecteur en colonnes individuelles
num_components = 200

for i in range(num_components):
    new_columns.append(pd.Series(df['pca_features'].apply(lambda x: x[i] if isinstance(x, np.ndarray) and i < len(x) else np.nan),
                                 name=f'pca_feature_{i+1}'))

# Utilisation pd.concat pour ajouter toutes les colonnes à la fois
complete_df = pd.concat([df] + new_columns, axis=1)
```

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

```
complete_df.shape
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
(22688, 203)
```

```
cloud_df = complete_df.drop('pca_features', axis=1)
```

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

```
# Affichage des 5 premières lignes
cloud_df.head()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
                                                path  ... pca_feature_200
0  s3://oc-projet9-fruits/Test/Watermelon/r_77_10...  ...        0.674343
1  s3://oc-projet9-fruits/Test/Watermelon/r_181_1...  ...       -0.423714
2  s3://oc-projet9-fruits/Test/Watermelon/r_59_10...  ...        0.259353
3  s3://oc-projet9-fruits/Test/Watermelon/259_100...  ...        0.825481
4  s3://oc-projet9-fruits/Test/Cauliflower/r_183_...  ...       -0.170755

[5 rows x 202 columns]
```

Nombre d'images par classe :

```
cloud_df['label'].value_counts()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
Grape Blue        328
Plum 3            304
Walnut            249
Cherry Rainier    246
Peach 2           246
                 ...
Cucumber Ripe     130
Tomato Maroon     127
Pear Kaiser       102
Mangostan         102
Ginger Root        99
Name: label, Length: 131, dtype: int64
```

V- Traitement des images

11) Sauvegarde des données au format .csv

Sauvegarde des données en .csv :

```
# Chemin S3 pour l'enregistrement du fichier CSV
path_s3 = 's3://oc-projet9-fruits/Results/P9.csv'

# Enregistrement du DataFrame en tant que fichier CSV sur S3
cloud_df.to_csv(path_s3, index=False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=La
```

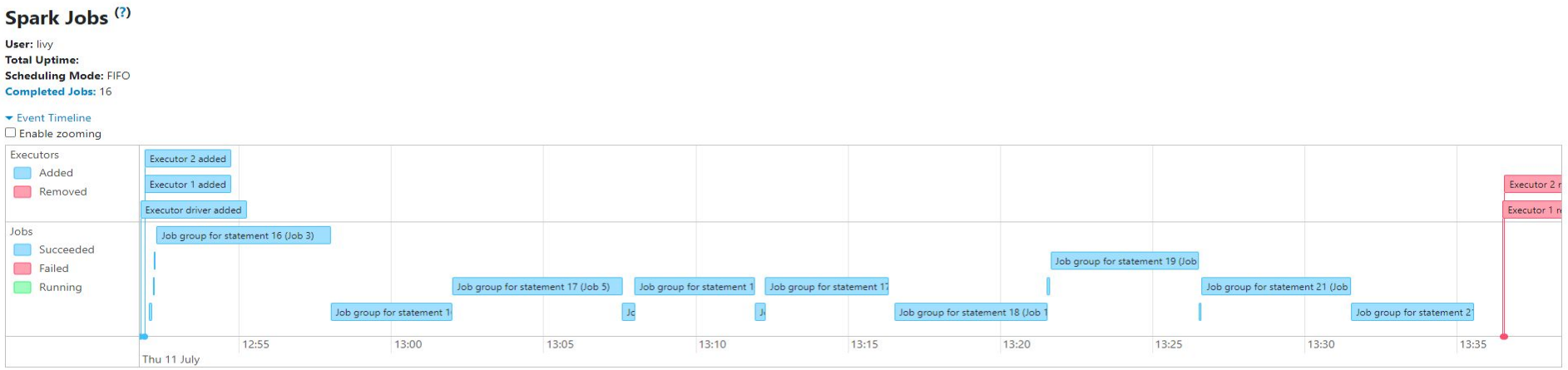
```
# Lecture le fichier CSV depuis S3
cloud_df = pd.read_csv(path_s3)

# Affichage des 5 premières lignes
cloud_df.head()
```

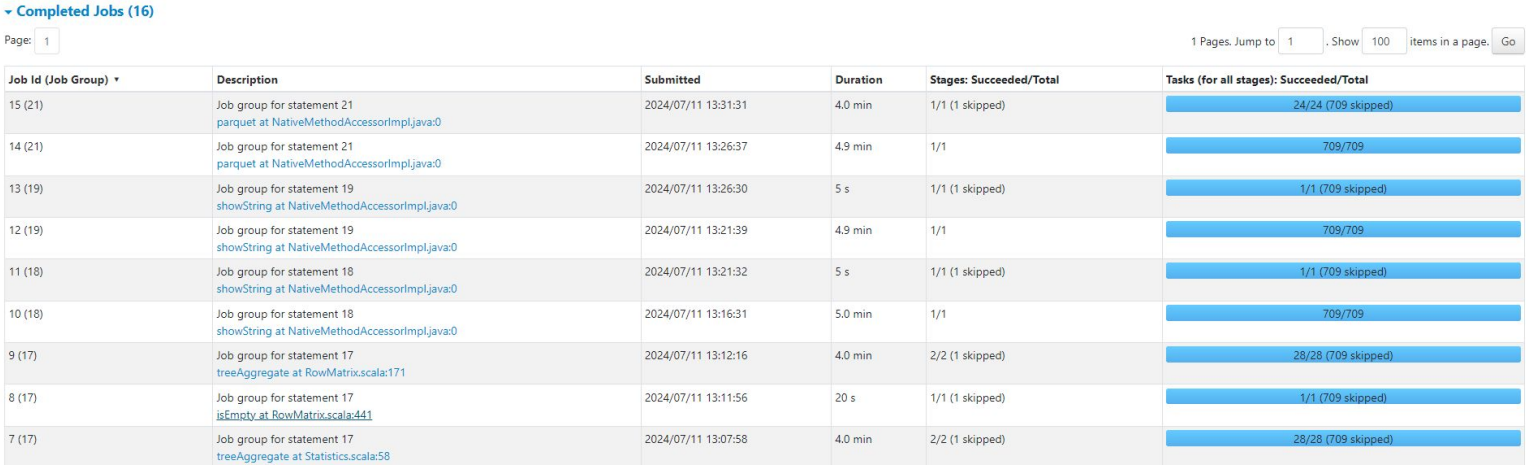
```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=La
                                                path  ... pca_feature_200
0  s3://oc-projet9-fruits/Test/Watermelon/r_77_10...  ...        0.674343
1  s3://oc-projet9-fruits/Test/Watermelon/r_181_1...  ...       -0.423714
2  s3://oc-projet9-fruits/Test/Watermelon/r_59_10...  ...        0.259353
3  s3://oc-projet9-fruits/Test/Watermelon/259_100...  ...        0.825481
4  s3://oc-projet9-fruits/Test/Cauliflower/r_183_...  ...       -0.170755

[5 rows x 202 columns]
```

```
mkdir -p ~/s3-bucket-backup
cd ~/s3-bucket-backup
aws s3 sync s3://oc-projet9-fruits ./
```

V- Traitement des images

12) Démonstration d'exécution dans le cloud





Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
15 (21)	Job group for statement 21 parquet at NativeMethodAccessorImpl.java:0	2024/07/11 13:31:31	4.0 min	1/1 (1 skipped)	24/24 (709 skipped)
14 (21)	Job group for statement 21 parquet at NativeMethodAccessorImpl.java:0	2024/07/11 13:26:37	4.9 min	1/1	709/709
13 (19)	Job group for statement 19 showString at NativeMethodAccessorImpl.java:0	2024/07/11 13:26:30	5 s	1/1 (1 skipped)	1/1 (709 skipped)
12 (19)	Job group for statement 19 showString at NativeMethodAccessorImpl.java:0	2024/07/11 13:21:39	4.9 min	1/1	709/709
11 (18)	Job group for statement 18 showString at NativeMethodAccessorImpl.java:0	2024/07/11 13:21:32	5 s	1/1 (1 skipped)	1/1 (709 skipped)
10 (18)	Job group for statement 18 showString at NativeMethodAccessorImpl.java:0	2024/07/11 13:16:31	5.0 min	1/1	709/709
9 (17)	Job group for statement 17 treeAggregate at RowMatrix.scala:171	2024/07/11 13:12:16	4.0 min	2/2 (1 skipped)	28/28 (709 skipped)
8 (17)	Job group for statement 17 isEmpty at RowMatrix.scala:441	2024/07/11 13:11:56	20 s	1/1 (1 skipped)	1/1 (709 skipped)
7 (17)	Job group for statement 17 treeAggregate at Statistics.scala:58	2024/07/11 13:07:58	4.0 min	2/2 (1 skipped)	28/28 (709 skipped)

VIII - Conclusion

Mise en Place d’une Architecture Big Data

- **Composants utilisés :** EMR, S3, IAM, EC2
- **Chaîne de Traitement :** Du chargement des données à l'analyse avancée

Considérations

- **Avantages :**
  - Répond précisément à nos besoins et contraintes.
  - Capable de gérer et d'analyser de grandes quantités de données efficacement.
  - Permet des traitements et des analyses rapides grâce à la parallélisation des calculs.
- **Inconvénient :**
  - Nécessite un investissement financier significatif pour une utilisation prolongée.

Merci pour votre attention