Déployer un modèle dans le cloud

Fruits!

PLE Coline

Plan de la présentation

- Rappel de la problématique
- Présentation du jeu de données
- Présentation du processus de création de l'environnement
- Réalisation de la chaîne de traitements des images dans un environnement Big Data dans le Cloud
- Ballade dans ma session aws
- Synthèse/Conclusion/Discussion

RAPPEL DE LA PROBLEMATIQUE

Fruits!

L'entreprise

"**Fruits!**", très jeune start-up de l'AgriTech, cherche à proposer des solutions innovantes pour la récolte des fruits. La préservation de leur biodiversité serait assurée par des traitements spécifiques pour chaque espèce *via* le développement de robots cueilleurs intelligents.

BESOINS ET MISSIONS

Besoins/Souhaits:

1. Se faire connaître en mettant à disposition du grand public une application mobile qui permettrait aux utilisateurs de prendre en photo un fruit et d'obtenir des informations sur ce fruit.
2. Sensibiliser le grand public à la biodiversité des fruits *via* l'application mobile.
3. Mettre en place une première version du moteur de classification des images de fruits.

Missions en tant que Data Scientist:

1. M'approprier les travaux réalisés par l'alternant.
2. Compléter la chaîne de traitement.
3. Réaliser la classification supervisée à partir des images des produits.

EN RESUME: Mettre en place les premières briques nécessaires de traitement permettant le passage à l'échelle en termes de volume de données !

PRESENTATION DU JEU DE DONNEES

Fruits!

Lien Kaggle pour le téléchargement du jeu de données:
https://www.kaggle.com/datasets/moltean/fruits

Les images:

1. Nombre total d'images: 90483
2. Nombre d'images dans le jeu d'entraînement: 67692 (1 fruit ou légume par image) ~ 75%
3. Nombre d'images dans le jeu de test 22688 (1 fruit ou légume par image) ~ 25% *
4. Nombre d'images dans le jeu multi-fruits 103
5. 131 classes de fruits et légumes 131
6. Taille des images: 100 x 100 pixels

Pomme

Cerise

Noix de coco

Citron

Concombre

Chou fleur

Maïs

Aubergine

PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Installation de awscli et stockage des données dans S3

Fruits!

1. Installation AWS Cli (Interface en ligne de commande d'AWS) et configuration de aws

```
C:\Users\colin>pip install --user awscli
```

Identification de sécurité du compte

```
(projet7) C:\Users\colin>aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: eu-west-3
Default output format [None]: aws s3 ls
```

Pas de donnée s3 (Normal)

Paris

Le bucket (Seau)

Simple Storage Services

Stockage des résultats

2. Stockage des données du projet à traiter

```
(projet7) C:\Users\colin>aws s3 mb s3://p8-data-cloud
make_bucket: p8-data-cloud

(projet7) C:\Users\colin>aws s3 ls
2023-12-07 15:25:02 p8-data-cloud
```

Création du bucket

Bucket crée

Amazon S3 > Compartiments > p8-data-cloud > Fruits_Cloud/

Nom	Type
Apple Golden 2/	Dossier
Cauliflower/	Dossier
Clementine/	Dossier
Corn/	Dossier
Kiwi/	Dossier
Lime/	Dossier
Onion white/	Dossier
Pear/	Dossier
Tomato 1/	Dossier
Walnut/	Dossier

Chargement des dossiers d'images de fruits et légumes

10 images / dossier

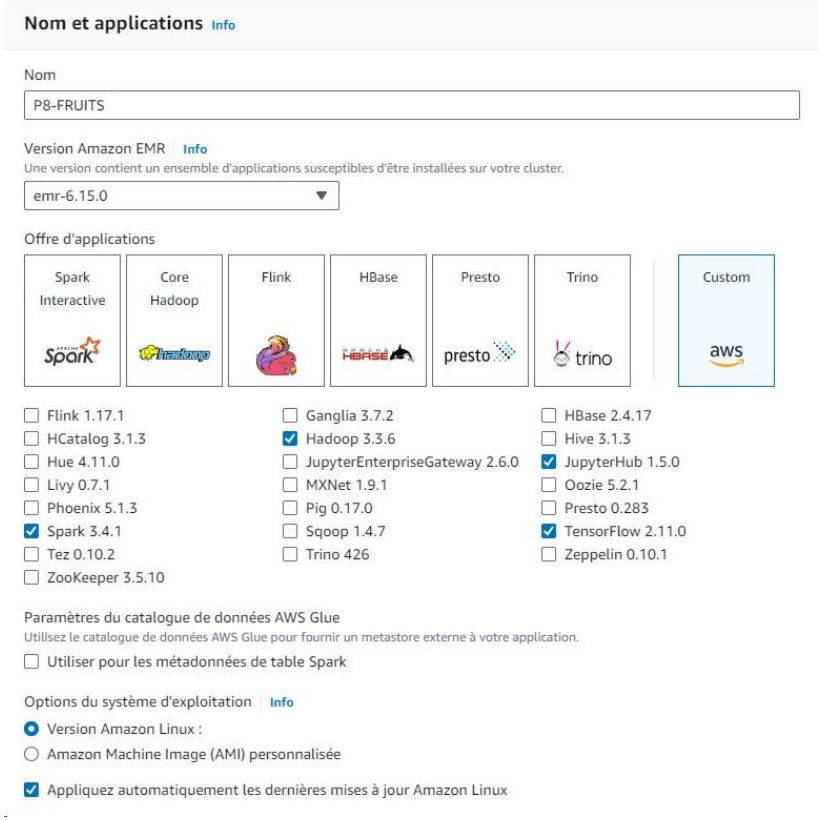
STOCKAGE DANS LA ZONE DE CALCUL EMR EC2

- ✓ 5 fois plus coûteux
- ✓ Données détruites lors de l'éteinte d'un cluster éphémère (recréation des données lors d'un nouveau calcul)

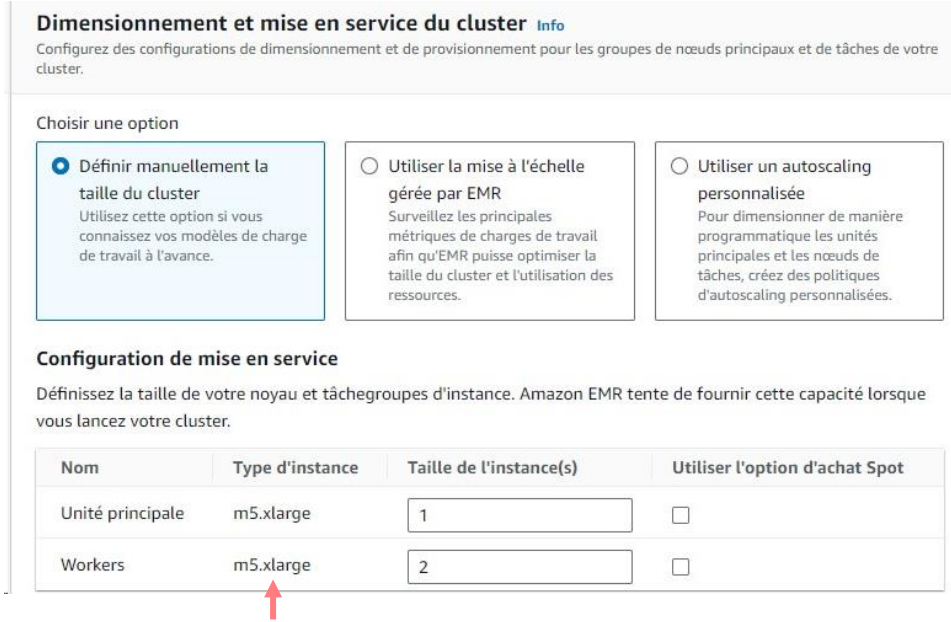
PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Configuration de la plateforme PAAS *(Plateforme As A Service)*

Fruits!

1. Création de l'EMR avec choix des applications



2. Location d'instances EC2 *(Elastic Compute Cloud)*



- Performances équilibrées puissance de calcul / mémoire de stockage
- Utilisation pour le traitement des données = OK
- Bon compromis qualité/prix

3. Localisation d'instances EC2 *(Elastic Compute Cloud)* en accord avec la réglementation RGPD

✓ Hébergement sur des serveurs localisés en Europe: **Paris**, Londres, Francfort, Stockholm, Irlande
✓ Proximité géographique des serveurs: Réduction de la latence → Amélioration des performances

PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Paramétrage de l'EMR

Fruits!

1. Résiliation du cluster

Résiliation du cluster Info

- ○ Résilier manuellement le cluster
- ○ Résilier automatiquement le cluster à la fin de la dernière étape
- ◉ Résilier automatiquement le cluster après le temps d'inactivité (Recommandé)

Temps d'inactivité
Saisissez la durée avant la résiliation de votre cluster.

0 jour ▼ | 01:00:00 ← **pour notre test (cout)**

Choisissez une durée supérieure à 1 minute (00:01:00) et inférieure à 7 jours. L'heure est au format hh:mm:ss (24 heures).

☐ Utiliser la protection contre la résiliation
Protégez vos instances EC2 de la résiliation accidentelle.

2. Ajout des librairies nécessaires par bootstrap

▼ **Actions d'amorçage – *facultatif*** Info
Utilisez les actions d'amorçage pour installer des logiciels ou personnaliser la configuration de votre instance.

Actions d'amorçage (1) — Supprimer | Modifier | Ajouter

	Nom	Emplacement Amazon S3	Arguments
○	bootstrap-emr.sh	s3://p8-data-cloud/bootstrap-emr.sh	-

C: > Users > colin > Desktop > $ bootstrap-emr.sh

```
#!/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
sudo python3 -m pip install pyarrow
sudo python3 -m pip install boto3
sudo python3 -m pip install s3fs
sudo python3 -m pip install fsspec
```

3. Modification des paramètres du logiciel

▼ **Paramètres logiciels - *facultatif*** Info

◉ Entrer la configuration | ○ Charger JSON à partir d'Amazon S3

```
[
  {
    "Classification": "jupyter-s3-conf",
    "Properties": {
      "s3.persistence.bucket": "p8-data-cloud",
      "s3.persistence.enabled": "true"
    }
  }
]
```

ouverture du notebook dans l'EMR

4. Paramètres de sécurité

Configuration de sécurité et paire de clés EC2 - *facultatif* Info

Configuration de sécurité
Sélectionnez les paramètres de chiffrement, d'authentification, d'autorisation et de service de métadonnées d'instance de votre cluster.

Choisir une configuration de sécu | Parcourir
Créer une configuration de sécurité

Paire de clés Amazon EC2 pour SSH sur le cluster Info

coline-ec2 | Parcourir
Créer une paire de clés

coline-ec2.ppk

PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Options de sécurité: IAM *(Identity and Access Management)*

Fruits!

Rôle Identity and Access Management (IAM) Info
Choisissez ou créez une fonction du service et un profil d'instance pour les instances EC2 de votre cluster.

Fonction du service Amazon EMR Info
La fonction du service est un rôle IAM assumé par Amazon EMR pour mettre en service des ressources et effectuer des actions au niveau du service avec d'autres services AWS.

- (●) **Choisir une fonction du service existant**
  Sélectionnez une fonction du service par défaut ou un rôle personnalisé avec des stratégies IAM attachées afin que votre cluster puisse interagir avec d'autres services AWS.
- ( ) **Créez une fonction du service**
  Laissez Amazon EMR créer une nouvelle fonction du service afin que vous puissiez accorder et restreindre l'accès aux ressources d'autres services AWS.

Fonction du service

Creation-cluster-p8 ▼ ⟳

Profil d'instance EC2 pour Amazon EMR
Le profil d'instance attribue un rôle à chaque instance EC2 d'un cluster. Le profil d'instance doit spécifier un rôle qui peut accéder aux ressources pour vos étapes et actions d'amorçage.

- (●) **Choisir un profil d'instance existant**
  Sélectionnez un rôle par défaut ou un profil d'instance personnalisé avec des stratégies IAM attachées afin que votre cluster puisse interagir avec vos ressources dans Amazon S3.
- ( ) **Choisir un profil d'instance**
  Laissez Amazon EMR créer un profil d'instance afin de pouvoir spécifier un ensemble personnalisé de ressources auquel il peut accéder dans Amazon S3.

Profil d'instance

Creation-ec2-s3_p8 ▼ ⟳

Rôle d'autoscaling personnalisé - *facultatif*
Lorsqu'une règle d'autoscaling personnalisée se déclenche, Amazon EMR assume ce rôle pour ajouter et résilier les instances EC2. En savoir plus

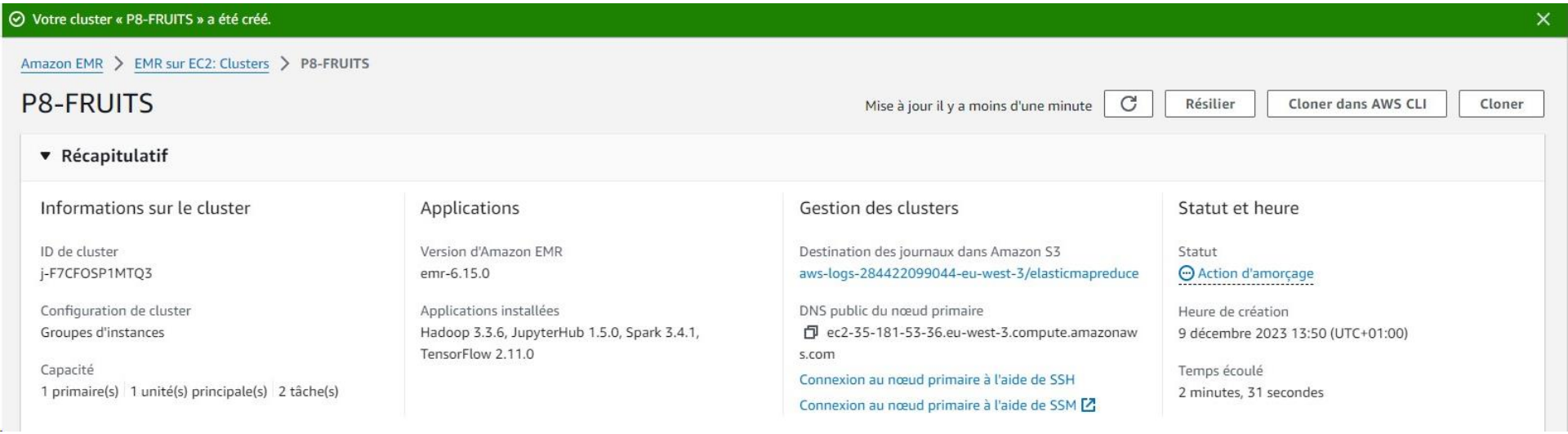
Rôle d'autoscaling personnalisé

Choisir un rôle IAM ▼ ⟳ **Créer un rôle IAM**

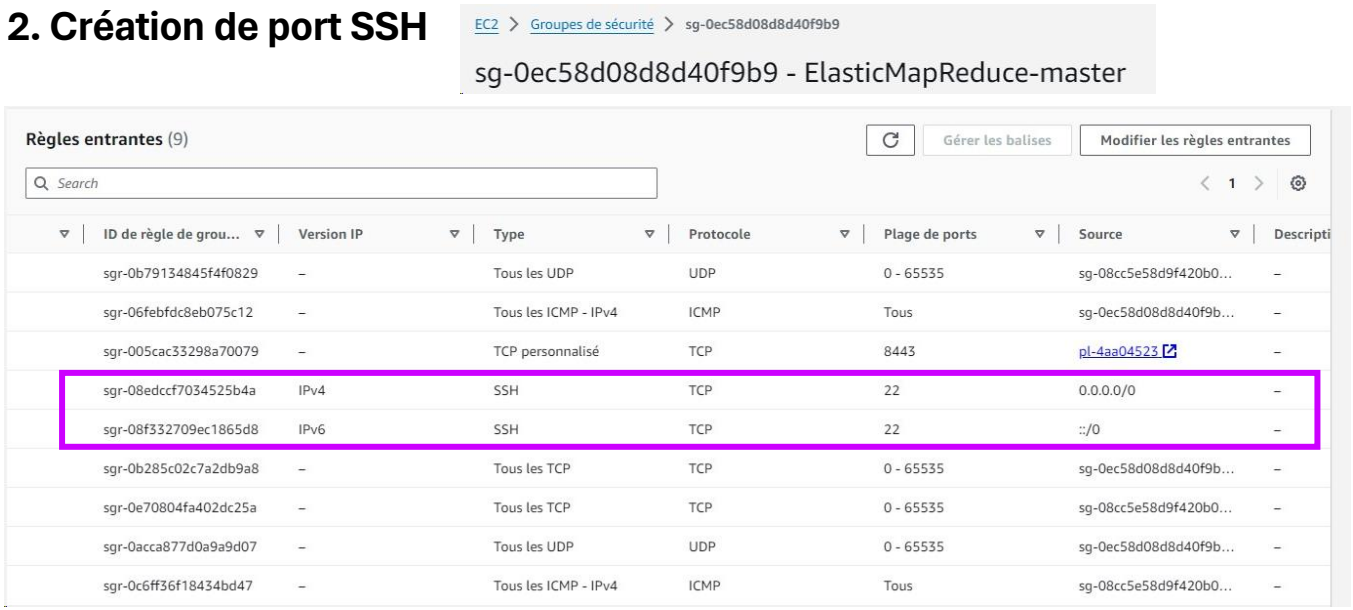
PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Instanciation du cluster et création du tunnel SSH

Fruits!

1. Instanciation du cluster : Environ 8 minutes pour le passage de statut En attente



2. Création de port SSH

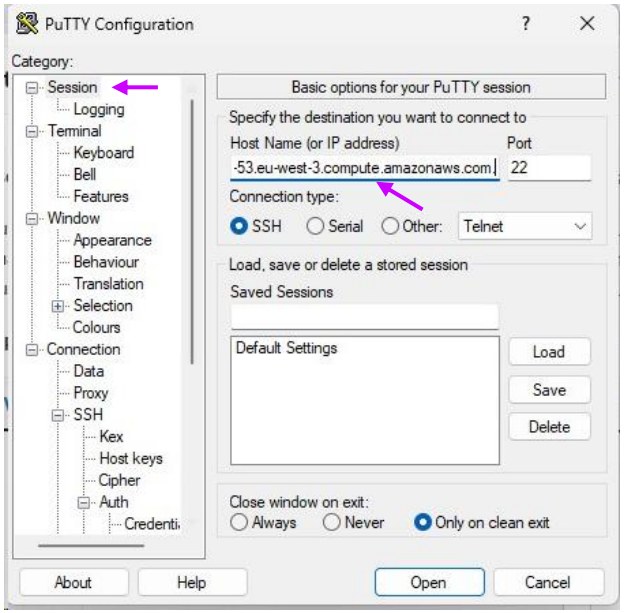


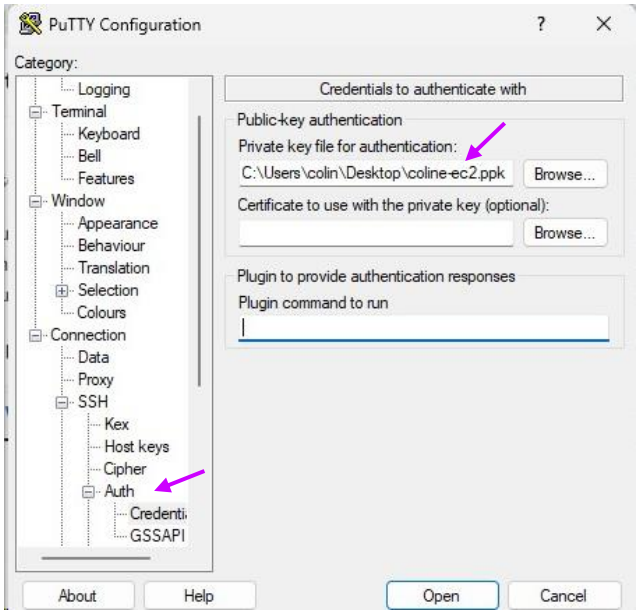
PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Configuration de l'accès à JupyterHub

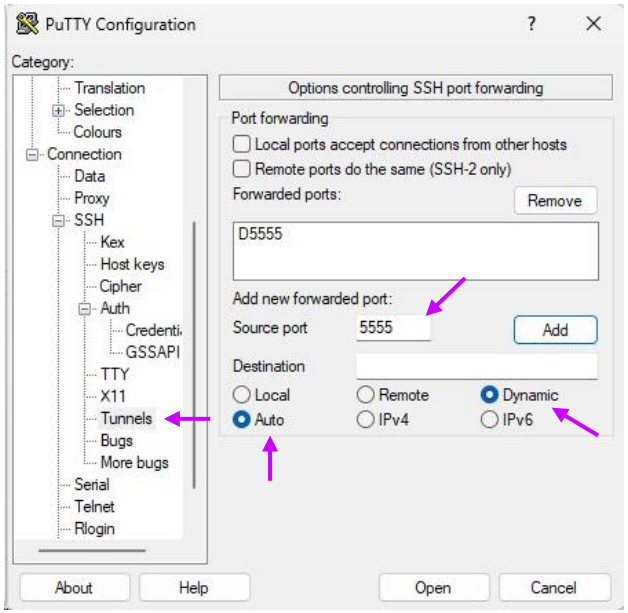
Fruits!

1. Installation de FoxyProxy et paramétrage

2. Utilisation de PuTTY pour l'accès à JupyterHub







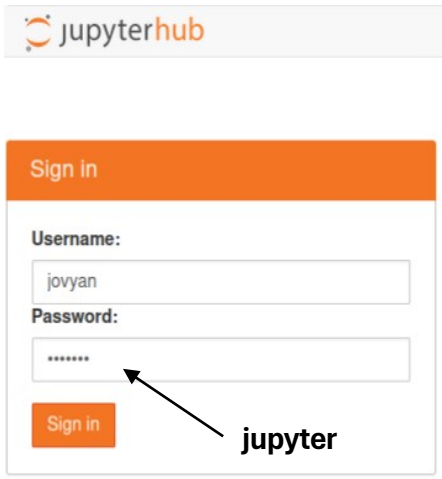
PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Accès et Exécution du notebook avec un kernel PySpark

Fruits!

Connexion à l'EMR établie

```
Using username "hadoop".
Authenticating with public key "coline-ec2"
Last login: Sun Dec 10 15:47:58 2023
       __|  __|_  )
       _|  (     /   Amazon Linux 2
      ___|\___|___|
https://aws.amazon.com/linux/amazon-linux-2/
19 package(s) needed for security, out of 20 available
Run "sudo yum update" to apply all updates.
```

1. EMR → Onglet Applications → JupyterHub



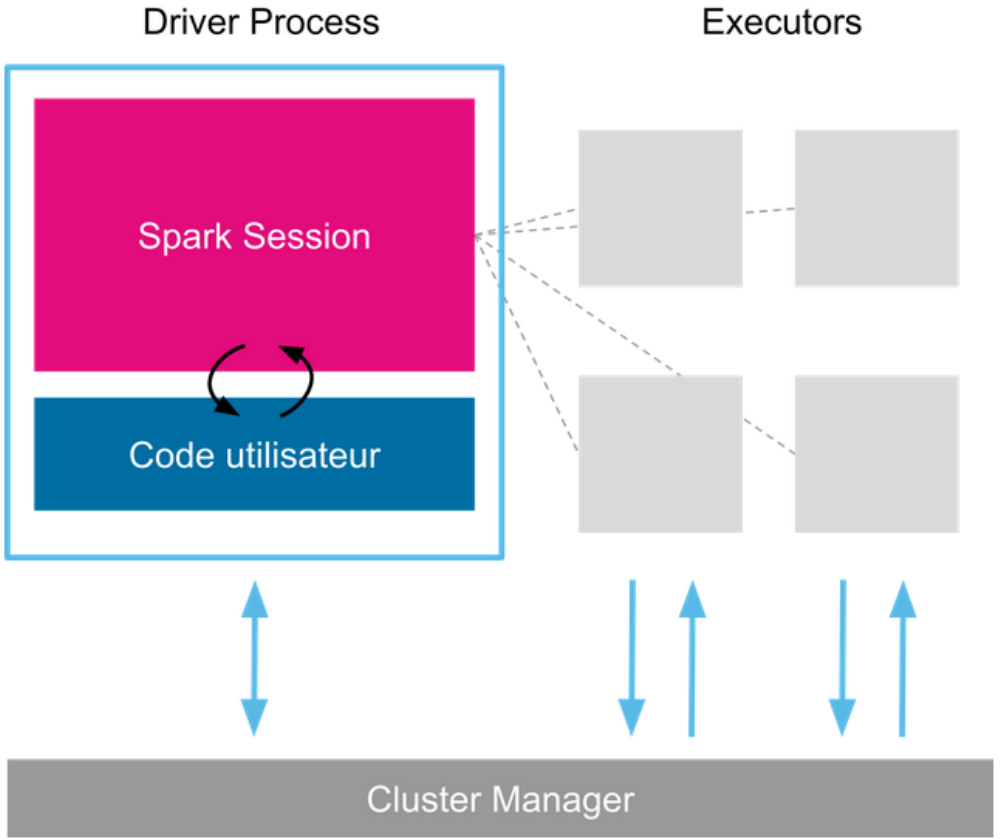
2. Import du Jupyter Notebook et ouverture avec un kernel PySpark

QU'EST-CE QUE PySpark?

Fruits!

1. **PySpark:** outil de communication avec Spark *via* le langage Python

2. Utilisation de **Spark** en local et sur le Cloud
   - ✓ Traitement de base de données massives par utilisation du calcul distribué (plusieurs unités de calcul réparties en clusters au profit d'un seul projet afin de diviser le temps d'exécution d'une requête).
   - ✓ Le driver (ou Spark session) distribue et planifie les tâches entre plusieurs exécuteurs (processus Java Virtual Machine (JVM) distinct dont il est possible de configurer le nombre de CPU et la quantité de mémoire alloué).



REALISATION DE LA CHAINE DE TRAITEMENTS DES IMAGES DANS UN ENVIRONNEMENT BIG DATA DANS LE CLOUD

Fruits!

1. Démarrage de la session Spark automatiquement par exécution d'une ligne vide

```
Entrée [1]: # 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?
4	application_1702223184396_0005	pyspark	idle	Link	Link	None	✓

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

2. Import des librairies nécessaires à l'exécution du script

```
import pandas as pd
from PIL import Image
import numpy as np
import io
import os

import tensorflow as tf
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, udf
from pyspark.sql import SparkSession
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.ml.feature import PCA
```

* Exécution d'une application Spark sur un cluster Hadoop

REALISATION DE LA CHAINE DE TRAITEMENTS DES IMAGES DANS UN ENVIRONNEMENT BIG DATA DANS LE CLOUD

Fruits!

3. Définition des PATH pour le chargement des images et l'enregistrement des résultats

```
PATH = 's3://p8-data-cloud'
PATH_Data = PATH+'/Fruits_Cloud'
PATH_Result = PATH+'/Results'
print('PATH:         '+\
      PATH+'\nPATH_Data:    '+\
      PATH_Data+'\nPATH_Result: '+PATH_Result)
```

```
PATH:         s3://p8-data-cloud
PATH_Data:    s3://p8-data-cloud/Fruits_Cloud
PATH_Result: s3://p8-data-cloud/Results
```

4. Chargement des images

spark.read
Création d'un dataframe

.format
Traitement des fichiers en tant que fichiers binaires (plus de souplesse pour le traitement des images)

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

Filtre des fichiers à lire
Lecture unique des fichiers avec l'extension .jpg (images)

Chargement des images
à partir de l'emplacement spécifié sous format binaire dans un dataframe Spark

Recherche récursive dans les répertoires et sous-répertoires de tous les fichiers correspondant au **filtre spécifié**

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

```
+--------------------+-------------------+------+--------------------+
|                path|   modificationTime|length|             content|
+--------------------+-------------------+------+--------------------+
|s3://p8-data/Test...|2021-07-03 09:00:08|  7353|[FF D8 FF E0 00 1...|
|s3://p8-data/Test...|2021-07-03 09:00:08|  7350|[FF D8 FF E0 00 1...|
|s3://p8-data/Test...|2021-07-03 09:00:08|  7349|[FF D8 FF E0 00 1...|
|s3://p8-data/Test...|2021-07-03 09:00:08|  7348|[FF D8 FF E0 00 1...|
|s3://p8-data/Test...|2021-07-03 09:00:09|  7328|[FF D8 FF E0 00 1...|
+--------------------+-------------------+------+--------------------+
only showing top 5 rows
```

5. Extraction des features

```
# Ajout d'une nouvelle colonne 'label' au dataframe images
images = images.withColumn('label', element_at(split(images['path'], '/'),-2))

# Impression des résultats
images.select('path','label').show(5,False)
```

Extraction de l'avant-dernier élément
→ **Classe du fruit/légume**

Impression des 5 lignes du dataframe sans troncage des colonnes si trop longues (False)

Division de la colonne 'path' en un tableau en utilisant '/' comme délimiteur
→ Création d'un tableau d'éléments correspondant aux différents niveaux du chemin.

```
+-----------------------------------------------------+-----------+
|path                                                 |label      |
+-----------------------------------------------------+-----------+
|s3://p8-data-cloud/Fruits_Cloud/Cauliflower/r_228_100.jpg|Cauliflower|
|s3://p8-data-cloud/Fruits_Cloud/Cauliflower/r_251_100.jpg|Cauliflower|
|s3://p8-data-cloud/Fruits_Cloud/Cauliflower/r_304_100.jpg|Cauliflower|
|s3://p8-data-cloud/Fruits_Cloud/Cauliflower/r_158_100.jpg|Cauliflower|
|s3://p8-data-cloud/Fruits_Cloud/Cauliflower/r_4_100.jpg  |Cauliflower|
+-----------------------------------------------------+-----------+
only showing top 5 rows
```

Vérification du nombre d'images (attendu 100)

```
# Obtention du nombre de lignes (donc images) avec .count() dans Pyspark (en non .shape[0])
images.count()
```

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

100
```

REALISATION DE LA CHAINE DE TRAITEMENTS DES IMAGES DANS UN ENVIRONNEMENT BIG DATA DANS LE CLOUD

Fruits!

5. Préparation du modèle

Transfert learning via MobileNetV2 (rapiditité d'exécution)

```python
def model_fn():
    """
    Retourne un modèle MobileNetV2 sans la dernière couche
    et les poids diffusés pré-entraînés.
    """
    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

```python
# Redimensionnement de nos images en 224x224 pixels (taille originale 100x100 pixels)
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 caractériques des images sous forme 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)

@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)
```

Redimensionnement des images

Conversion en array numpy des images redimensionnées

Stockage des prédictions dans la variable 'preds'

Aplatissement des caractéristiques en une liste de vecteurs de caractéristiques retournées sous forme de séries Pandas

Extraction des caractéristiques des images dans un contexte Spark DataFrame. L'utilisation d'un Pandas UDF de type SCALAR_ITER permet de traiter plusieurs lots de données à la fois. Elle charge le modèle une seule fois pour tous les lots, pouvant améliorer les performances lors du traitement de grandes quantités de données

7. Extraction des features sur 100 images du jeu de données test avec 20 exécuteurs

```
# Extraction des features en utilisant 20 exécuteurs
features_df = images.repartition(20).select(
    col("path"),
    col("label"),
    featurize_udf("content").alias("features")
)

# Visualisation de 5 lignes au hasard du DataFrame obtenu
features_df.show(5, truncate=True)

# Vérification du nombre d'images (nombre de lignes: attendu 100)
features_df.count()
```

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

+--------------------+--------------+--------------------+
|                path|         label|            features|
+--------------------+--------------+--------------------+
|s3://p8-data-clou...|          Lime|[0.10586139, 0.00...|
|s3://p8-data-clou...|          Pear|[0.24130498, 0.0,...|
|s3://p8-data-clou...|          Lime|[0.0, 0.0, 0.0569...|
|s3://p8-data-clou...|Apple Golden 2|[0.063913345, 0.0...|
|s3://p8-data-clou...|   Onion white|[0.12695207, 0.0,...|
+--------------------+--------------+--------------------+
only showing top 5 rows

100
```

8. Réalisation de la PCA avec 137 composantes principales *

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

# Application de la fonction UDF pour la création d'une nouvelle colonne "features_vector"
features_df = features_df.withColumn("features_vector", array_to_vector_udf(features_df["features"]))

# Création d'un objet PCA avec les 137 composantes principales pour atteindre 99% de la variance
pca = PCA(k=137, inputCol="features_vector", outputCol="vectorized_pca_features")

# Application de la PCA sur le DataFrame
model = pca.fit(features_df)
pca_features_df = model.transform(features_df)

# Sélection des colonnes pertinentes et affichage de 5 lignes au hasard
features_df_pca = pca_features_df.select("path", "label", "vectorized_pca_features")
features_df_pca.show(5, truncate=True)
```

← Vectorisation des features

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

+--------------------+-------------+-----------------------+
|                path|        label|vectorized_pca_features|
+--------------------+-------------+-----------------------+
|s3://p8-data-clou...|         Lime|   [-8.4689094507281...|
|s3://p8-data-clou...|Apple Golden 2|   [-7.4866635200894...|
|s3://p8-data-clou...|         Lime|   [-6.7544818278395...|
|s3://p8-data-clou...|         Pear|   [-7.7287299544243...|
|s3://p8-data-clou...|  Onion white|   [11.3108575033495...|
+--------------------+-------------+-----------------------+
only showing top 5 rows
```

* 99% de la variance expliquée lors du test en local sur 300 images *(cf slides supplémentaires)*

9. Restructuration des données après la PCA

```
from pyspark.sql.types import ArrayType, FloatType

# Définition explicite de la fonction UDF
def vector_to_array(vec):
    return vec.toArray().tolist()

# Conversion de la fonction Python en UDF
vector_to_array_udf = udf(vector_to_array, ArrayType(FloatType()))

# Application de la fonction UDF pour la création d'une nouvelle colonne "pca_features"
final_df = features_df_pca.withColumn("pca_features", vector_to_array_udf("vectorized_pca_features"))

# Sélection des colonnes pertinentes et affichage des 5 premières lignes
final_df = final_df.select("path", "label", "pca_features")
final_df.show(5, truncate=True)
```

```
+--------------------+--------------+--------------------+
|                path|         label|        pca_features|
+--------------------+--------------+--------------------+
|s3://p8-data-clou...|          Lime|[-6.754482, 1.691...|
|s3://p8-data-clou...|          Pear|[-7.7287297, 3.23...|
|s3://p8-data-clou...|          Lime|[-8.468909, 1.793...|
|s3://p8-data-clou...|Apple Golden 2|[-7.4866633, -1.4...|
|s3://p8-data-clou...|        Walnut|[6.6853433, 3.577...|
+--------------------+--------------+--------------------+
only showing top 5 rows
```

```
final_df.printSchema()
```

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

```
# Localisation des résultats
print(PATH_Result)
```

```
s3://p8-data-cloud/Results
```

10. Enregistrement des données, chargement des données et validation des résultats

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

```
# Chargement des données depuis mon cloud :
df = pd.read_parquet(PATH_Result, engine='pyarrow')
```

```
df.head()
```

	path	...	pca_features
0	s3://p8-data-cloud/Fruits_Cloud/Lime/262_100.jpg	...	[-6.754482, 1.6913179, -6.994117, -0.10847554,...
1	s3://p8-data-cloud/Fruits_Cloud/Pear/r_7_100.jpg	...	[-7.7287297, 3.235324, -11.676723, -0.0947821,...
2	s3://p8-data-cloud/Fruits_Cloud/Lime/r_52_100.jpg	...	[-8.468909, 1.7938887, -8.9465885, 0.9497765, ...
3	s3://p8-data-cloud/Fruits_Cloud/Apple Golden 2...	...	[-7.4866633, -1.4561422, -7.9126353, 5.1878214...
4	s3://p8-data-cloud/Fruits_Cloud/Walnut/12_100.jpg	...	[6.6853433, 3.577582, -12.337966, -6.595814, -...

[5 rows x 3 columns]

Validation de la dimension du vecteur de caractéristiques des images (attendu: 137)

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

```
(137,)
```

Création d'un dataframe comprenant une colonne pour chaque caractéristique

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

	path	...	pca_feature_137
0	s3://p8-data-cloud/Fruits_Cloud/Lime/262_100.jpg	...	-0.43325
1	s3://p8-data-cloud/Fruits_Cloud/Pear/r_7_100.jpg	...	-0.43325
2	s3://p8-data-cloud/Fruits_Cloud/Lime/r_52_100.jpg	...	-0.43325
3	s3://p8-data-cloud/Fruits_Cloud/Apple Golden 2...	...	-0.43325
4	s3://p8-data-cloud/Fruits_Cloud/Walnut/12_100.jpg	...	-0.43325

[5 rows x 139 columns]

Validation des labels de fruits

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

```
Lime              10
Pear              10
Apple Golden 2    10
Walnut            10
Onion white       10
Tomato 1          10
Kiwi              10
Clementine        10
Cauliflower       10
Corn              10
Name: label, dtype: int64
```

Sauvegarde des données sous format csv

```
# Chemin S3 pour l'enregistrement du fichier CSV
path_s3 = 's3://p8-data-cloud/Results/P8.csv'

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

Ouverture du fichier pour vérification

```
# Chemin S3 du fichier CSV
path_s3 = 's3://p8-data-cloud/Results/P8.csv'

# 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=
```

	path	...	pca_feature_137
0	s3://p8-data-cloud/Fruits_Cloud/Lime/262_100.jpg	...	-0.43325
1	s3://p8-data-cloud/Fruits_Cloud/Pear/r_7_100.jpg	...	-0.43325
2	s3://p8-data-cloud/Fruits_Cloud/Lime/r_52_100.jpg	...	-0.43325
3	s3://p8-data-cloud/Fruits_Cloud/Apple Golden 2...	...	-0.43325
4	s3://p8-data-cloud/Fruits_Cloud/Walnut/12_100.jpg	...	-0.43325

CONCLUSION SUR LE CLOUD AWS

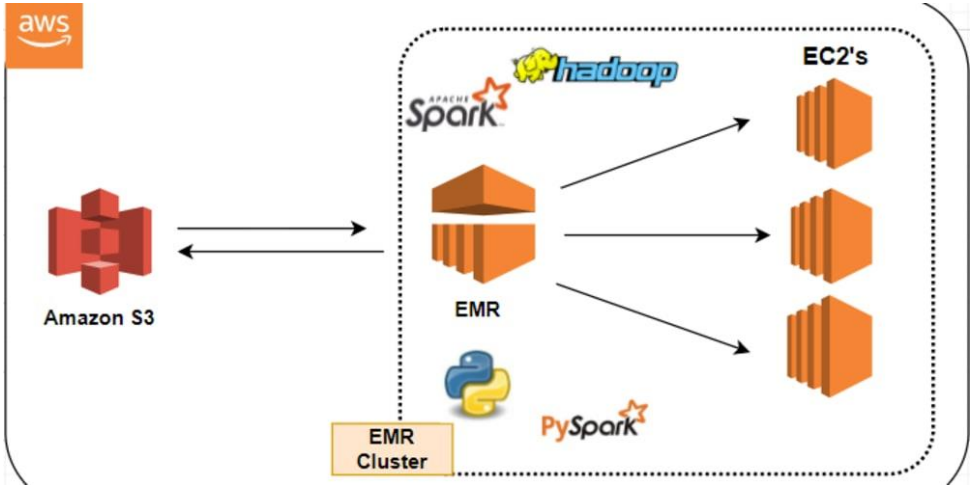
Fruits!

Google Cloud

Microsoft Azure

Leader et créateur (2006)

amazon web services



- ✓ Stockage possible d'un grand volume de données
- ✓ BEMOL: **Coût financier pour notre start-up** (Utilisation non-stop sans résiliation)
  - Augmentation des coûts avec l'augmentation du volume de données (Stockage des données, Puissance de calcul nécessitant une augmentation des instances, Coût supplémentaire pour la bande passante du réseau).
  - Pour le traitement de 100 images avec la location de 3 instances m5.xlarge sur Paris: (0,224$/instance/heure).
  - Autres sites européens possibles: Stockholm (0,204$/instance), Irlande (0,214$/instance), Londres(0,222$/instance) et Francfort (0,230$/instance).

- Débat / Réflexion -

PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Création de la session Spark en local

Fruits!

```
spark = (SparkSession
         .builder
         .appName('P8')
         .master('local')
         .config("spark.sql.parquet.writeLegacyFormat", 'true')
         .getOrCreate()
)
```

Nom de l'application Spark

Mode d'exécution de Spark : Local
(Souvent utilisé pour le développement et les tests)

Obtention d'une instance de la session Spark
- Récupération d'une session Spark existante avec le même nom d'application.
OU
-Création d'une nouvelle session.

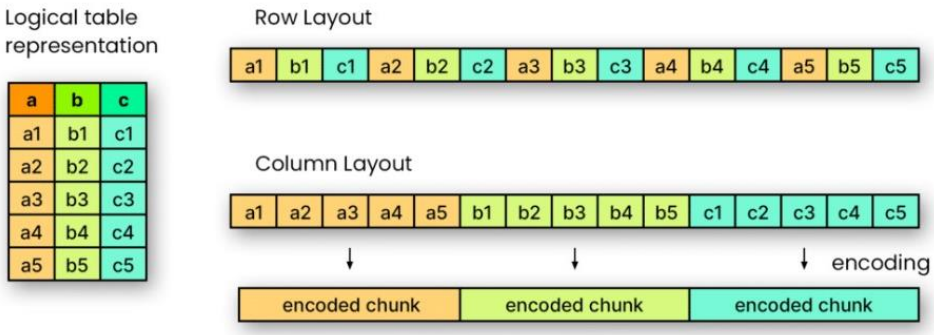
Format de stockage: Parquet
Format de stockage de données performant, efficace en termes de stockage et compatible avec de nombreux frameworks de traitement de données, ce qui en fait un choix populaire pour stocker des données volumineuses dans des environnements distribués

- Compression plus efficace des données
(Les valeurs d'une même colonnes ont tendance à être similaires)

- Lancement des calculs de traitement parallèle des données efficace

Colonnes

Avec Parquet, les données sont stockées **sous forme de colonnes**. Cela signifie que chaque colonne de données est stockée dans un fichier séparé. Cette structure permet une lecture sélective efficace des données en ne lisant que les colonnes nécessaires pour une requête donnée.



Source: https://blent.ai/blog/a/apache-parquet-tout-savoir

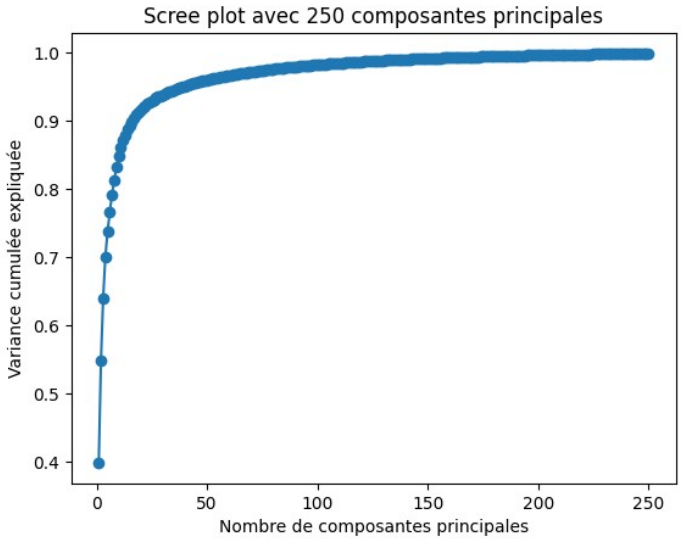
PROCESSUS DE CREATION DE L'ENVIRONNEMENT
Réalisation d'une PCA en local

Fruits!

Détermination du nombre de composantes pour atteindre 99% de la variance expliquée

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

# Application de la fonction UDF pour la création d'une nouvelle colonne "features_vector"
features_df = features_df.withColumn("features_vector", array_to_vector_udf(features_df["features"]))

# Création d'un objet PCA avec un grand nombre de composantes principales
pca_high_dimensions = PCA(k=250, inputCol="features_vector", outputCol="pca_features_high_dimensions")

# Application de la PCA sur le DataFrame
model_high_dimensions = pca_high_dimensions.fit(features_df)

# Obtention de la variance expliquée par chaque composante principale
explained_variance_high_dimensions = model_high_dimensions.explainedVariance

# Calcul de la variance cumulée
cumulative_variance_high_dimensions = [sum(explained_variance_high_dimensions[:i+1]) for i in range(len(explained_variance_high_dimensions))]

# Traçage du Scree plot
plt.plot(range(1, len(explained_variance_high_dimensions) + 1), cumulative_variance_high_dimensions, marker='o')
plt.xlabel("Nombre de composantes principales")
plt.ylabel("Variance cumulée expliquée")
plt.title("Scree plot avec 250 composantes principales")
plt.show()
```

← Vectorisation des features




```
# Recherche du nombre de composantes nécessaires pour atteindre 99% de la variance expliquée
cumulative_variance_threshold = 0.99
num_components_threshold = next(i for i, var in enumerate(cumulative_variance_high_dimensions) if var >= cumulative_variance_threshold)

# Affichage du nombre de composantes nécessaires
print(f"Nombre de composantes pour atteindre {cumulative_variance_threshold * 100}% de la variance expliquée : {num_components_threshold}")
```

```
Nombre de composantes pour atteindre 99.0% de la variance expliquée : 137
```

Réalisation de la PCA avec 137 composantes non seulement en local mais aussi lors du déploiement du modèle dans le Cloud