



Parcours OpenClassRooms

Data Scientist

P8 Déployez un modèle dans le cloud



Fruits!

Pictures used for educational purpose only

Sommaire

I. Introduction

II. L'environnement Big Data

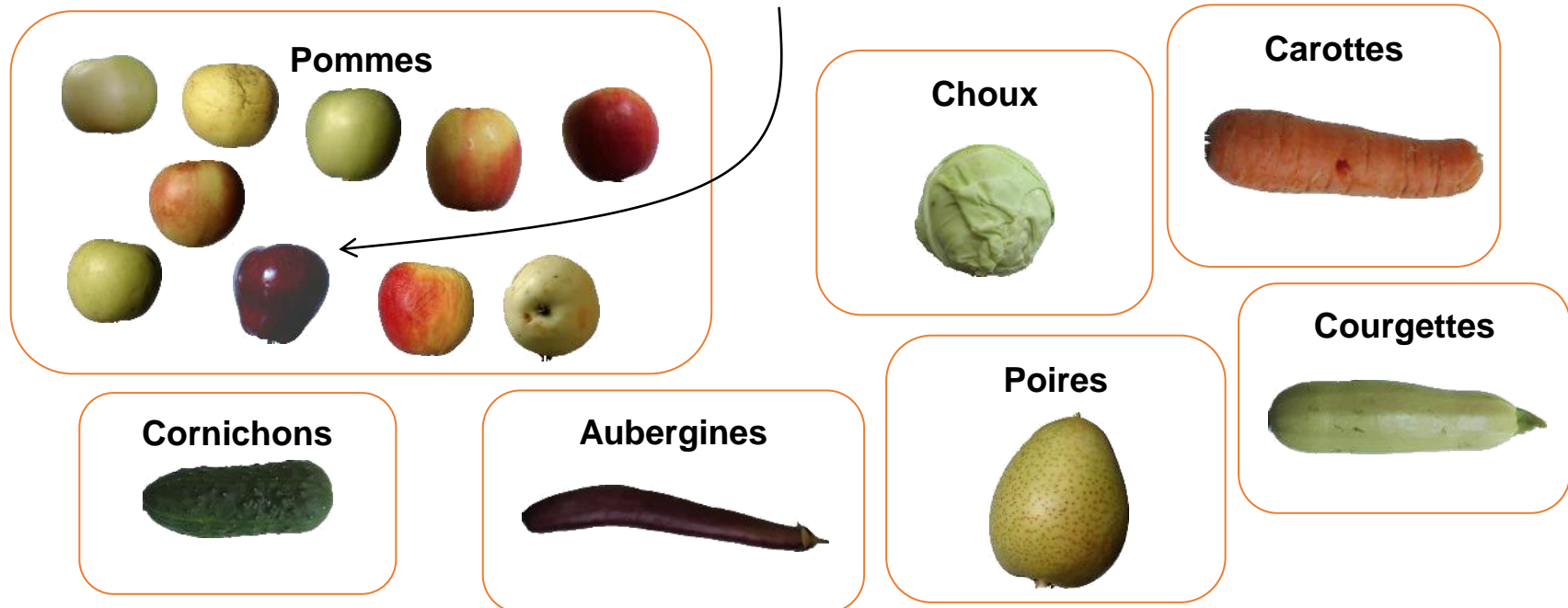
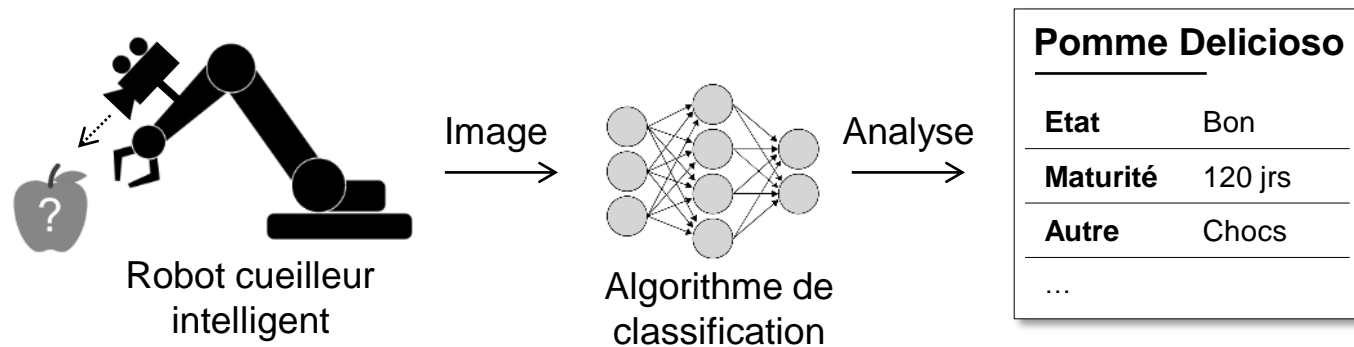
III. Le traitement d'image avec Spark

IV. Conclusion

I. Introduction

1) Le projet

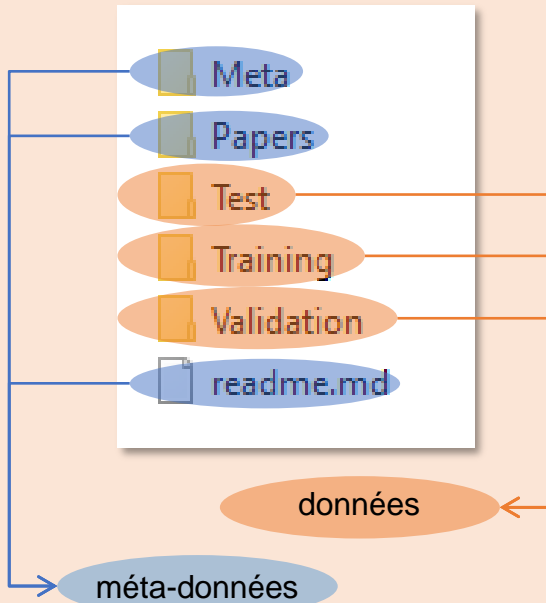
Prototype d'**application mobile** qui permette de photographier un fruit et d'en obtenir des informations.
→ A terme : permettre un traitement spécifique à chaque espèce de fruit & légume en développant des **robots cueilleurs intelligents**.



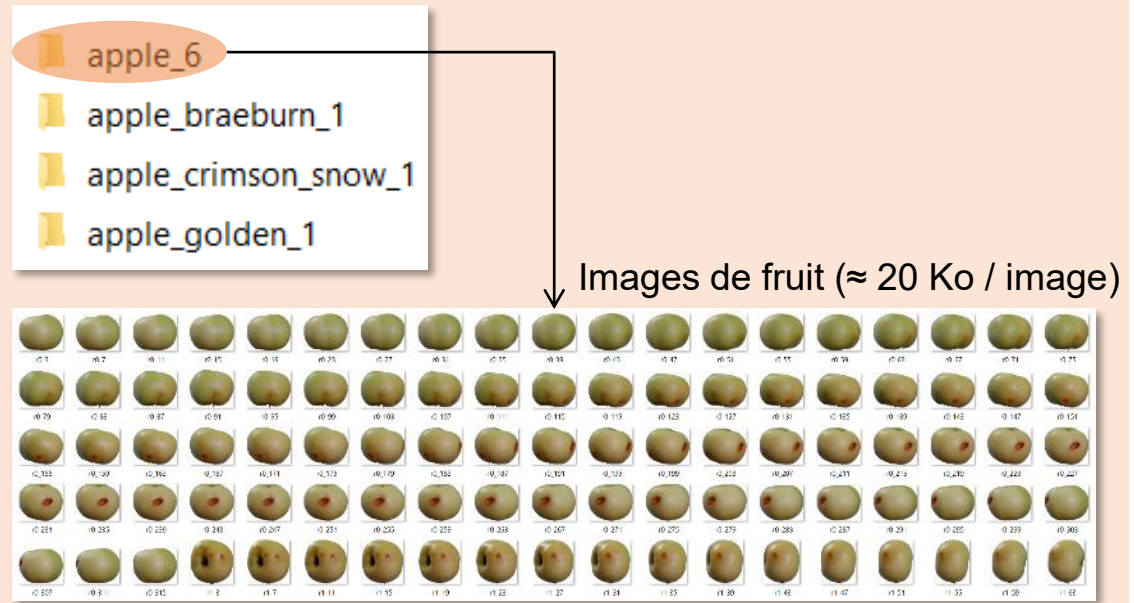
I. Introduction

2) Le jeu de données

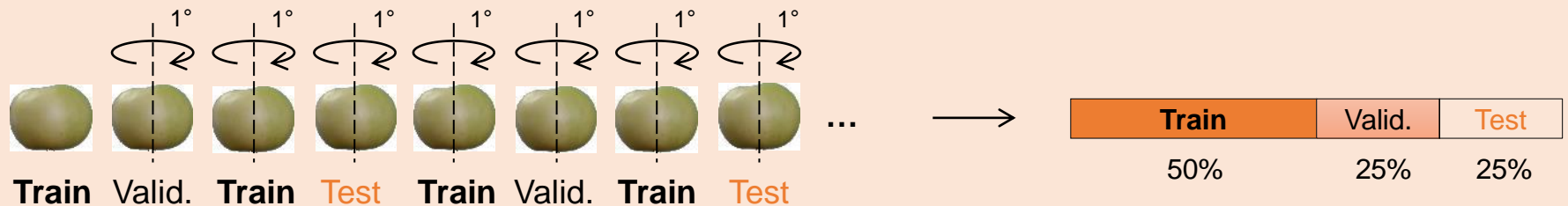
Dossier racine



Dossiers Test, Training et Validation



Répartition des images

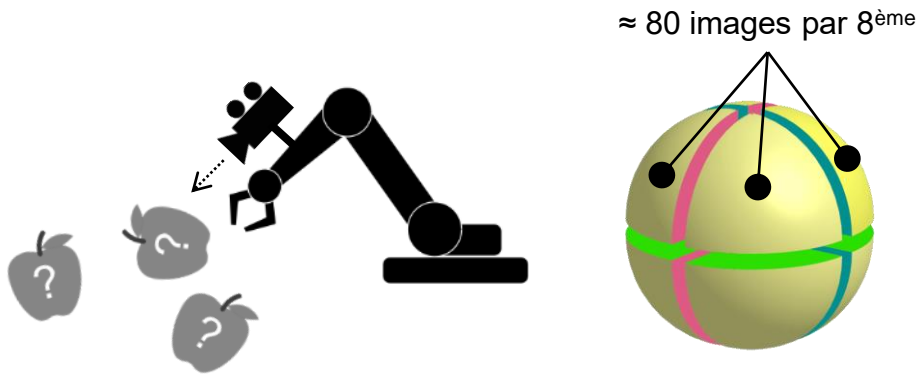


I. Introduction

3) La problématique

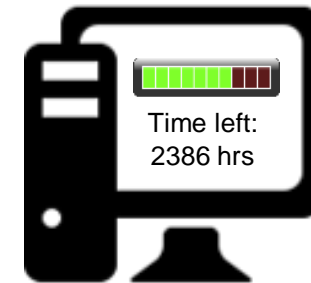
→ 24 fruits dans la base de données pour l'instant.

→ Jusqu'à **650 photos par fruit**, pour anticiper l'angle de vue des futures photos.

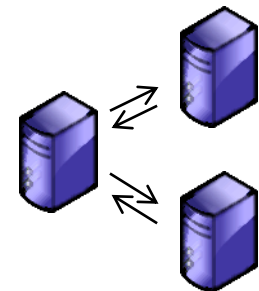


Problème : la base de données va s'agrandir avec de nouveaux fruits.

→ On risque de **dépasser les capacités de calcul** d'un seul ordinateur.

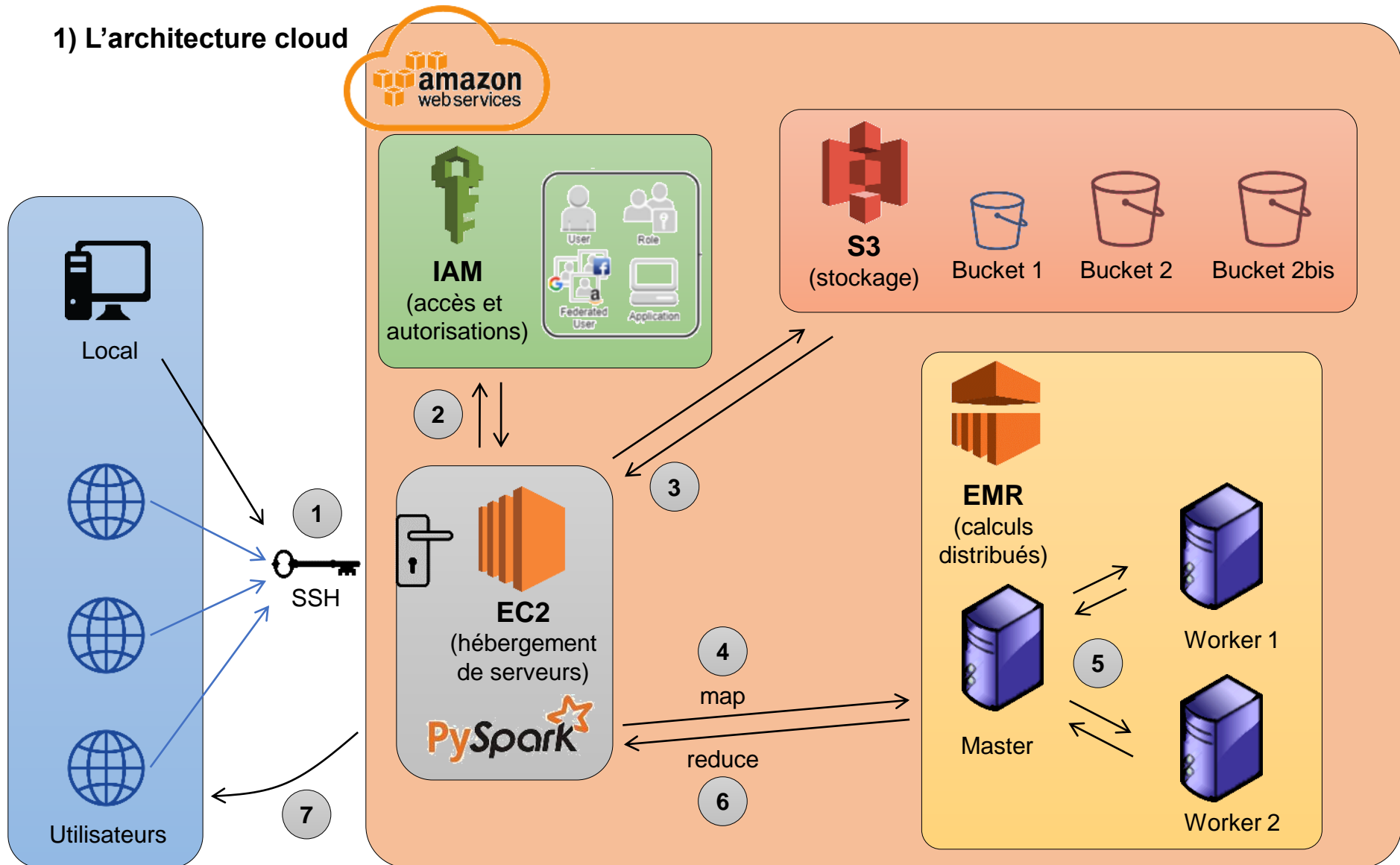


→ Ce volume de donnée important nécessite de passer à l'échelle du **cloud** et d'utiliser le **distributed computing**.



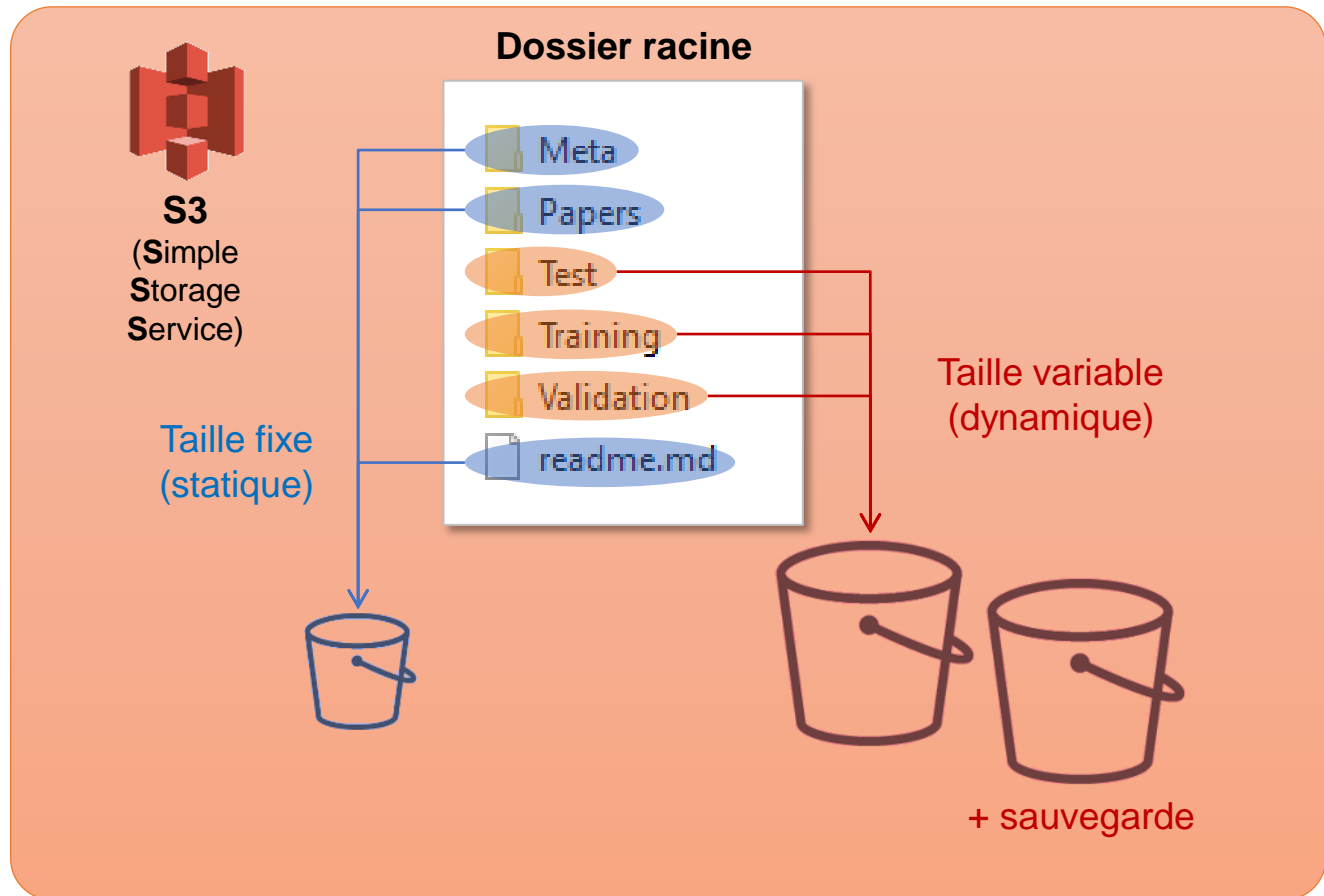
II. L'environnement BigData

1) L'architecture cloud



II. L'environnement BigData

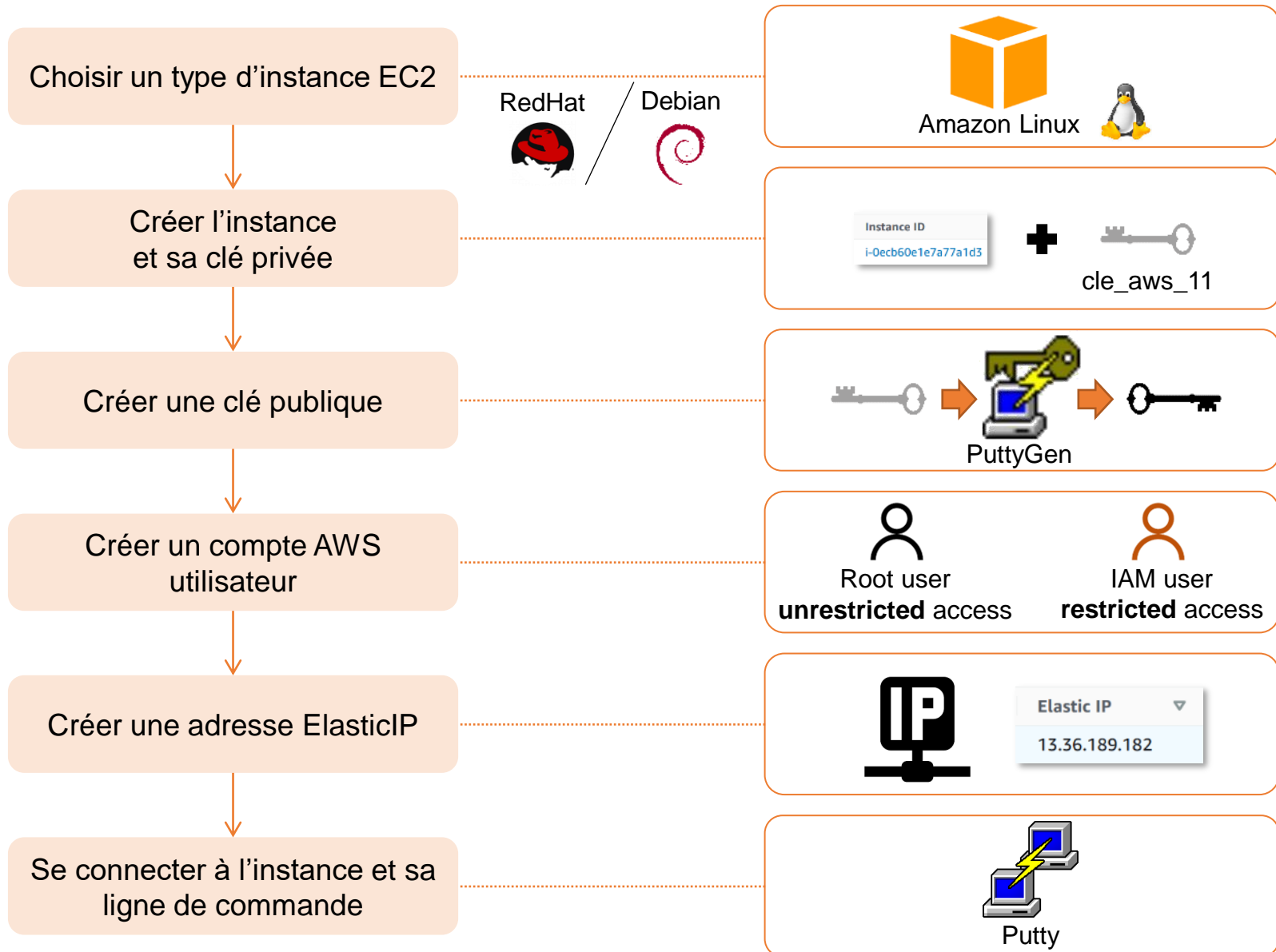
2) Le stockage des données



<input type="radio"/>	ocr-taille-fixe	EU (Paris) eu-west-3	Objects can be public
<input type="radio"/>	ocr-taille-variable	EU (Paris) eu-west-3	Objects can be public
<input type="radio"/>	ocr-taille-variable-backup	EU (Paris) eu-west-3	Bucket and objects not public

II. L'environnement BigData

3) La mise en place de l'instance EC2



II. L'environnement BigData

4) Préparation du programme Spark

Mettre en place les règles de sécurité

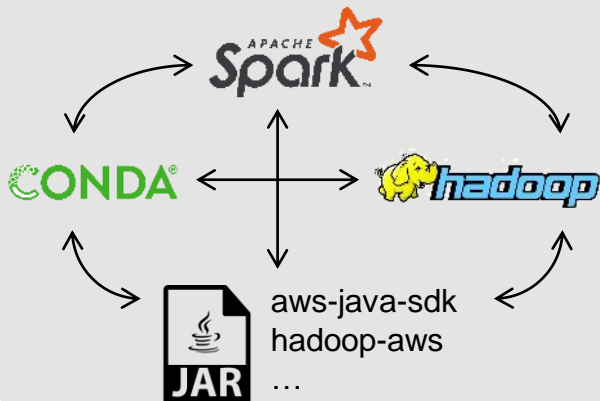
1^{ère} tentative

Installer Conda, Spark, les fichiers .jar

Installer Tensorflow / Keras



Erreurs de mémoire et d'incompatibilité de versions



2^{ème} tentative

Choisir une instance :

→ de taille **t3.large**

→ destinée au **deep learning**

→ avec tensorflow et conda **pré-installés**

Type	Port range
Custom TCP	7077
SSH	22
Custom TCP	8443
Custom TCP	8888
HTTPS	443
Custom TCP	8080 - 8081

Activer Tensorflow & Conda

```
source activate tensorflow p37
```



Ouvrir un notebook Jupyter

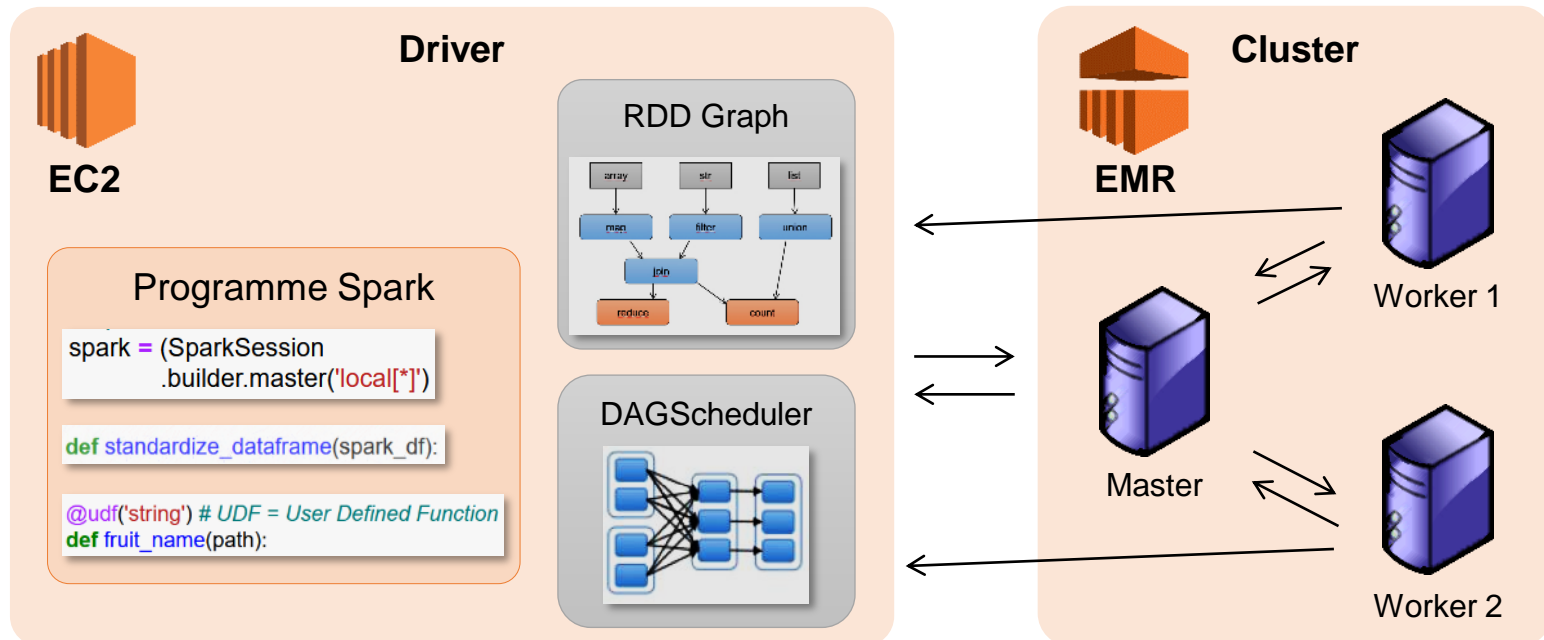


Rédiger et exécuter le programme Spark



II. L'environnement BigData

5) Principe de fonctionnement de Spark



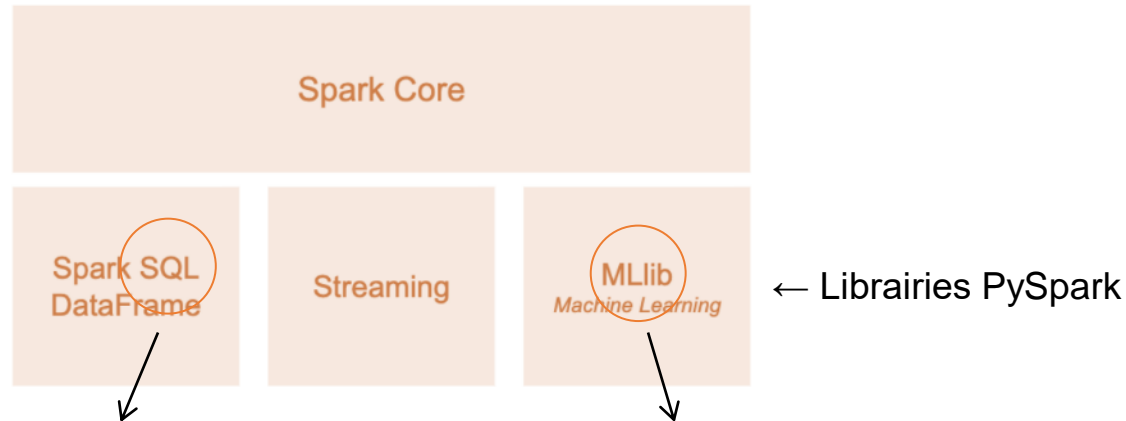
Programme Spark	Exécute les instructions Spark	} Opérations effectuées par Spark en arrière-plan
RDD Graph	Transformations et actions Spark	
DAGScheduler	Distribue les calculs selon le schéma MapReduce de Hadoop	
Master	Répartit les calculs entre les workers	
Workers	Exécutent les calculs	

→ Dans le cadre de ce projet, le master et les workers sont hébergés sur le même serveur, l'instance EC2

II. L'environnement BigData

6) Les librairies PySpark utilisées


(API Python pour Spark)



1) Méthodes propres à la librairie

```
from pyspark.sql import SparkSession

spark = (SparkSession
        .builder.master("local[*]")
```

2) Décorateurs

```
from pyspark.sql.functions import udf

@udf('string') # UDF = User Defined Function
def fruit_name(path):
```

```
from pyspark.ml.feature import StandardScaler
```

```
standardizer = StandardScaler(withMean=True,
                              withStd=True,
                              inputCol='features_vector',
                              outputCol='features_std')
```

→ On peut distribuer les calculs avec des **méthodes propres** aux librairies PySpark, ou bien des **fonctions décorées**.

III. Le traitement d'image avec Spark

2) Configuration du programme Spark

→ Un programme Spark nécessite des paramétrages supplémentaires par rapport à un programme Python habituel.

SparkSession

→ point d'entrée des fonctionnalités de Spark

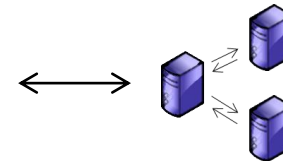
→ englobe tous les types de *context* : Spark, Hive, SQL, ...

```
# Spark session
spark = (SparkSession
        .builder.master('local[*]')
        .appName('p8_ocr')
        .config('spark.hadoop.fs.s3a.access.key', ACCESS_KEY_ID)
        .config('spark.hadoop.fs.s3a.secret.key', SECRET_ACCESS_KEY)
        .config('spark.hadoop.fs.s3a.impl', 'org.apache.hadoop.fs.s3a.S3AFileSystem')
        .config('com.amazonaws.services.s3.enableV4', 'true')
        .config('spark.hadoop.fs.s3a.endpoint', 's3.' + REGION + '.amazonaws.com')
        .getOrCreate())
```

SparkContext

→ connexion à un **cluster** master-workers

```
# Spark context and log level
spark_context = spark.sparkContext
spark_context.setLogLevel('WARN')
```



boto3

→ “*Python Software Development Kit*” conçu pour configurer et gérer des services AWS.

→ nous sert à configurer la connexion à **S3**

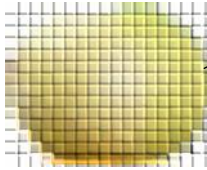
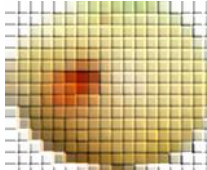
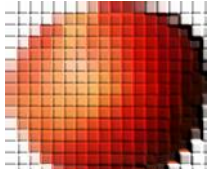
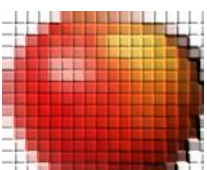
```
# S3 / EC2 authorizations
s3 = boto3.resource('s3',
                    REGION,
                    aws_access_key_id=ACCESS_KEY_ID,
                    aws_secret_access_key=SECRET_ACCESS_KEY)
bucket = s3.Bucket(BUCKET_NAME)
```



→ On peut maintenant charger les images et utiliser les méthodes de PySpark

III. Le traitement d'image avec Spark

3) Tableau de départ

fruit	jpeg	image
apple_6	r0_0.jpeg	
apple_6	r0_2.jpeg	
...
apple_braeburn_1	r0_0.jpeg	
apple_braeburn_1	r0_2.jpeg	
...

≈ 400 x 400 pixels RGB

→ **Objectif** : en faire un tableau qui puisse être exploité par un algorithme de machine learning

III. Le traitement d'image avec Spark

4.1) Etapes de traitement des images

Chargement des images

```
def path_dataframe(img_list):
```

```
@udf('string') # UDF = User Defined Function  
def fruit_name(path):
```

```
@udf('string')  
def jpeg_name(path):
```

```
def readable_columns(path_sdf):
```

Constituer le tableau d'images

Créer des colonnes lisibles

Prédictions

```
resnet_model = ResNet50(include_top=False,  
                        weights=None,  
                        pooling='max',  
                        input_shape=(224, 224, 3))
```

```
def images_sample(image_list, img_nb=2400):
```

```
def clean_image(image):
```

```
def model_image_prediction(model, image):
```

```
def predictions_dataframe(model, data, img_nb=200):
```

Instancier le réseau de neurones

Sélectionner des images aléatoirement

Charger une image

Obtenir la prédiction sur une image

Rassembler toutes les prédictions

III. Le traitement d'image avec Spark

4.2) Etapes de traitement des images

Chargement des images

Prédictions

Fusion et post-processing

```
def merge_dataframe(img_sdf, pred_sdf):
```

```
def postprocess_dataframe(sdf):
```

```
def standardize_dataframe(sdf):
```

Fusionner le tableau des images avec le tableau des prédictions

Réduction de dimension

```
def pca_dataframe(sdf, n_comp=150):
```

Appliquer une réduction de dimension à la matrice

→ Combien de composants choisir pour la réduction de dimension PCA ?

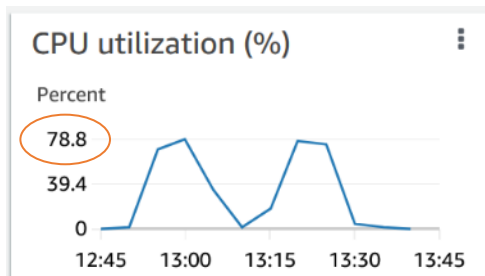
III. Le traitement d'image avec Spark

5) Réduction de dimension PCA

→ Les poids des composantes varient selon le nombre d'images traitées.

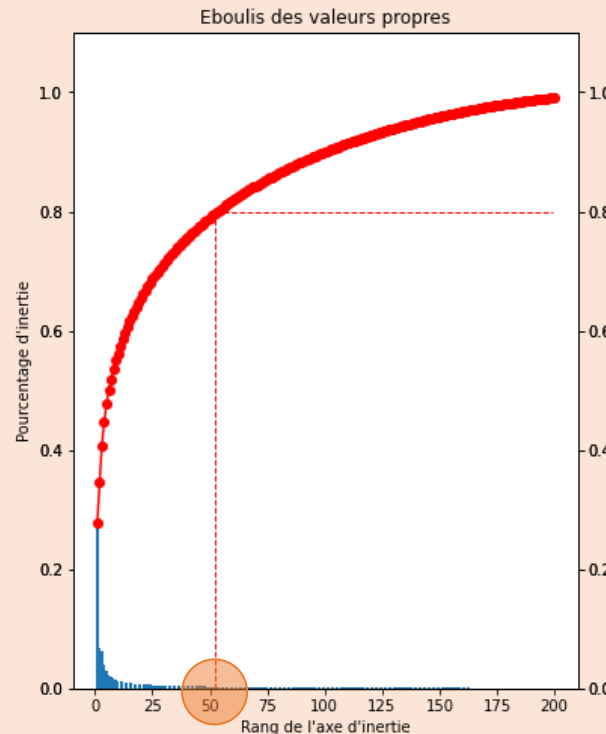
→ Ces graphiques varient avec les aléas de la méthode `random.choices`

→ Sollicitation du serveur pour 2400 images :



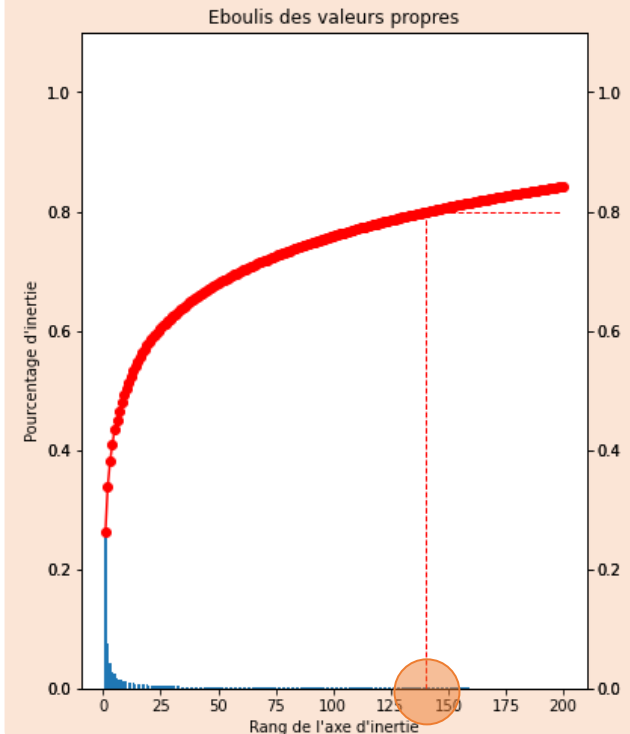
240 images (10 / fruit)

→ 80% dans ≈ 50 composantes



2400 images (100 / fruit)

→ 80% dans ≈ 140 composantes



→ En augmentant le nombre d'images traitées, les 80% semblent converger vers **150** composantes principales.

IV. Conclusion

1) Bilan et perspectives



Le jeu de données

- Prêt pour l'analyse dans sa forme actuelle.
- Destiné à s'agrandir, il nécessite l'appui du **cloud** (distributed computing)
- Majorité d'espèces de pommes (15 pommes sur 24 fruits) : risque de **biais**



L'environnement BigData

- Les échanges d'information se font entre les **utilisateurs**, l'**instance** EC2 et les **buckets** S3
- La **RAM** de l'instance est importante pour le bon fonctionnement du programme Spark
- La **compatibilité des versions** est déterminante



Le traitement d'image avec Spark

- Peu d'images : nécessite un transfer learning avec ResNet50 de Keras
- Les calculs distribués se font avec des **méthodes PySpark**, ou des **fonctions décorées** (UDF)
- La réduction de dimension PCA semble être optimale pour **150** composants.



Perspectives

- La parallélisation des calculs s'est faite sur le même serveur (mode local) : basculer sur un véritable **cluster EMR**.
- Diversifier les fruits
- Etudier d'autres solutions Cloud : Azure, Google Cloud, OVHcloud, ...
- Prendre les photos en proportions similaires selon l'axe de rotation
- Feature augmentation

IV. Conclusion

2) Recommandations pour le passage à l'échelle

Critères de qualité d'un outil cloud

1. Facilité de passage à l'échelle
2. Facilité de maintenance
3. Facilité d'exploitation des données

AWS 5 pillars

1. Operational excellence
2. Security
3. Reliability
4. Performance efficiency
5. Cost optimization

Fin de la présentation



Merci pour votre attention