Déployez un modèle dans le cloud

Projet 8 – Raphaël Koudache--Louvet

Entreprise : Fruits!

Objectif : Prendre en photographie un fruit et obtenir des informations sur ce fruit => Computer Vision

Problématique : Architecture BigData pour passage à l'échelle

Mission : Chaine de traitement des données avec preprocessing et réduction de dimensionnalité

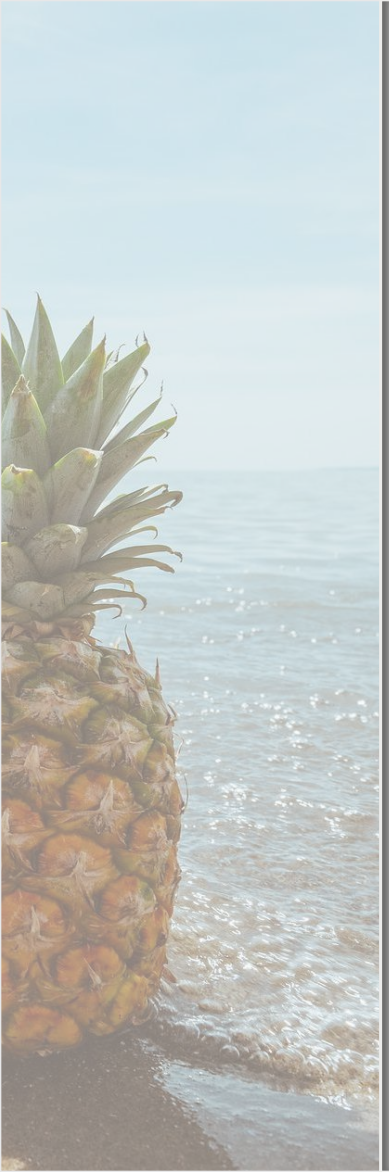
Plan

I – Analyse exploratoire des données

II – Pipeline Pyspark

III – Déploiement dans le cloud

I - Analyse exploratoire des données



Vue d'ensemble

Dossiers : Images tailles « standardisées » + images tailles originales

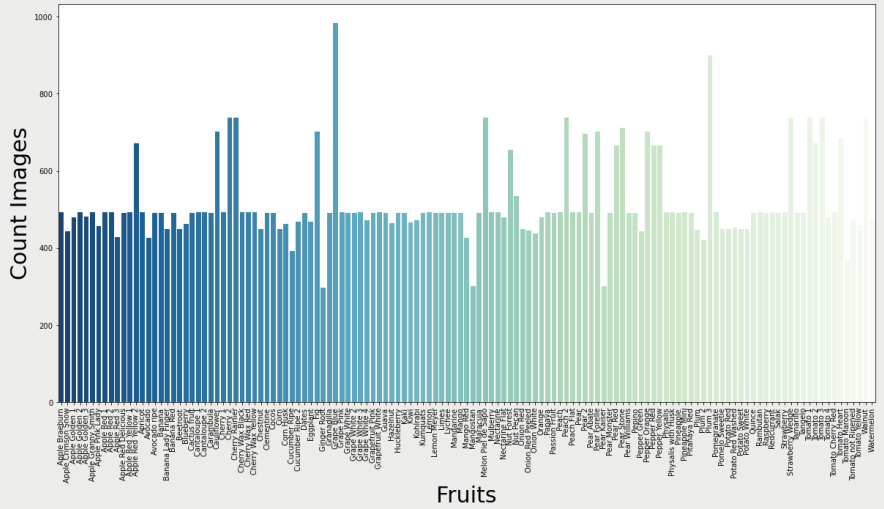
Sous dossiers : Training 50% /Test 25 %/ Validation 25 %

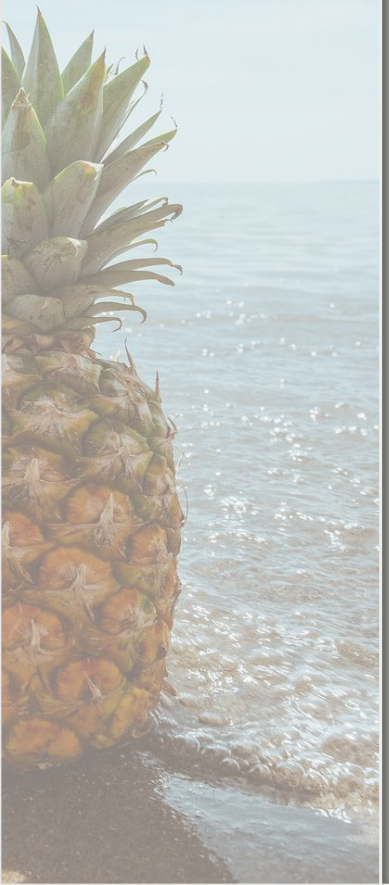
90380
Images

(67692 Train,
22688 Test)

131
Espèces
Fruits

Number of images per fruits (Training folder)

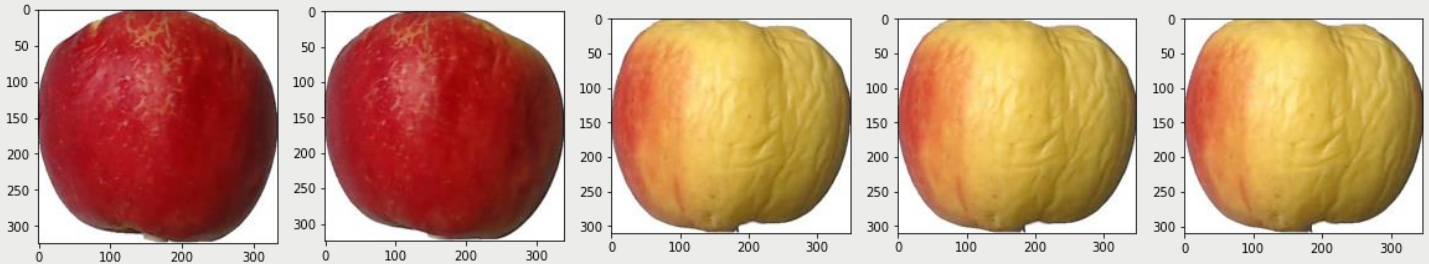


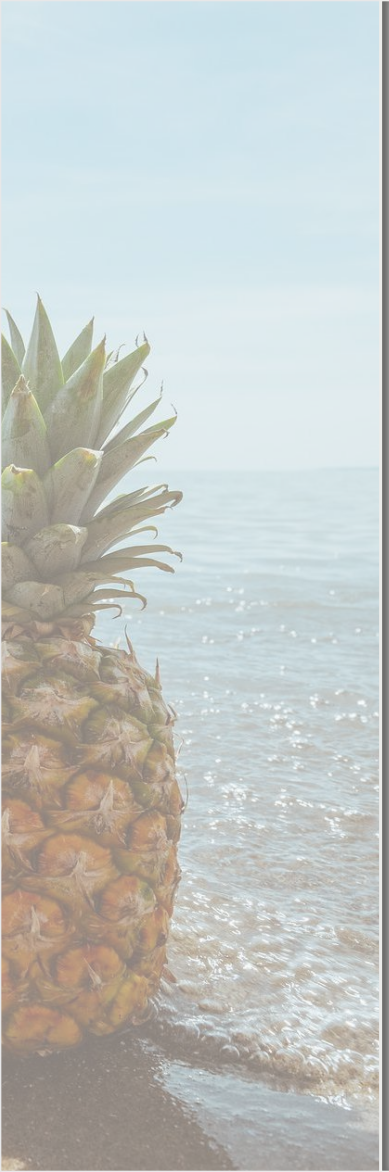


Méthode de création du DataSet & Exemples d'Images

Fruits sont filmés 360°

Images de fruits sont extraites avec un algorithme pour uniformiser le fond





PCA mise à l'échelle

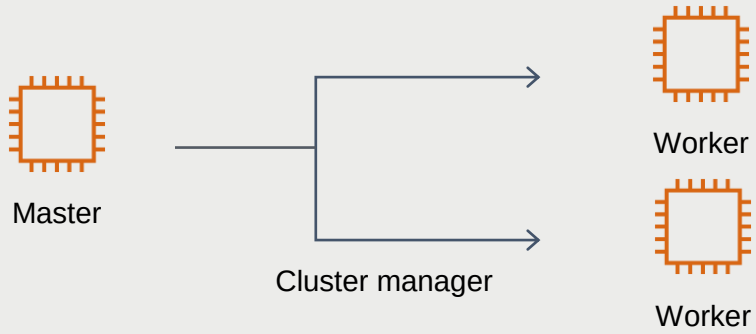
II - Pipeline Pyspark

Calcul distribué

Logique MapReduce (diviser) => Apache Hadoop (distribuer avec un cluster)

Apache Hadoop(SSD)/Apache Spark(RAM)

Architecture Master-workers



Pyspark

Apache Spark (Java) => PySpark

Lazy evaluation (Transformations VS actions)

MLSpark

Pyspark

Resilient Distributed Datasets (RDD)

Map

Mais moins facile d'utilisation avec MLSpark

DataFrames

UDF

```
rdd_bytes_images = sc.binaryFiles("s3a://oc-project-8-bucket/Sub_S3/**")
```

Feature Extraction

```
rdd_array_image = rdd_bytes_images.map(lambda x: np.asarray(Image.open(io.BytesIO(x[1])).resize((224,224))))
```

```
model = VGG16(weights="imagenet",
                    pooling='max',
                    include_top=False,
                    input_shape=(224, 224, 3))

for layer in model.layers:
            layer.trainable = False

model.compile()
model.summary()
```

```
rdd_extracted_features = rdd_array_image.map(lambda x : model.predict(np.array([x])))
```

Preprocessing

```
df = spark.read.format("binaryFile").load(s3_object_path)
```

```
feature_extracted_df = df.select(feature_extract_UDF(df.content).alias("features"))
```

```
vector_df = feature_extracted_df.select(to_vector_UDF('features').alias('features'))
```

Feature Extraction VGG16

Deep Learning

Convolutional neural network (CNN)

Architecture VGG16 (implémentée de façon à être remplacée par une autre si nécessaire et si appartient à Keras)

Paramètres entrainables = 0

Dernière couche : Max pooling (possible d'intervertir avec d'autres poolings ou une flatten layer)

Dimensions de sortie : vecteur (1,512) pour une image

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 global_max_pooling2d (Globa  (None, 512)              0
 lMaxPooling2D)

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Pipeline Mise en Place

SparkSession => DataFrames

Pyspark Read Binaries

Pillow => np.array

Tansorflow => VGG16 MaxPooling 512 Dimensions

PysparkML Pipeline StandardScaler + PCA

Pyspark coalesce puis export csv

Preprocessing

Entrée [ ]:
```
df = spark.read.format("binaryFile").load(s3_object_path)
```

Entrée [ ]:
```
feature_extracted_df = df.select(feature_extract_UDF(df.content).alias("features"))
```

Entrée [ ]:
```
vector_df = feature_extracted_df.select(to_vector_UDF('features').alias('features'))
```

Dimentionality Reduction

Entrée [ ]:
```
n_components = 150

std = StandardScaler(inputCol="features", outputCol="scaled")
pca = PCA(inputCol="scaled", outputCol="pca").setK(n_components)
stages = [std, pca]
```

Entrée [ ]:
```
pipeline = Pipeline().setStages(stages)
```

Entrée [ ]:
```
pca_df = pipeline.fit(vector_df).transform(vector_df)
```

Entrée [ ]:
```
pca_df_multiple_columns = pca_df.withColumn("pc", vector_to_array("pca"))\
                          .select([col("pc")[i] for i in range(n_components)])
```

Entrée [ ]:
```
label_df = df.select(label_extract_UDF(df.path).alias("labels"))
```

Entrée [ ]:
```
pca_df_multiple_columns_id = \
    pca_df_multiple_columns.withColumn('id', monotonically_increasing_id())

label_df_id = label_df.withColumn('id', monotonically_increasing_id())
```

Entrée [ ]:
```
labeled_pca_df = label_df_id.join(pca_df_multiple_columns_id, on=['id'], how='inner').drop('id')
```

Entrée [ ]:
```
labeled_pca_df.coalesce(1)\
          .write.save("s3a://oc-project-8-pca-csv/run", format='csv', header=True)
```

III - Déploiement dans le cloud

Amazon Web Services (Vue d'ensemble non exhaustive)

BOTO3

Amazon Elastic Compute Cloud (Amazon EC2)

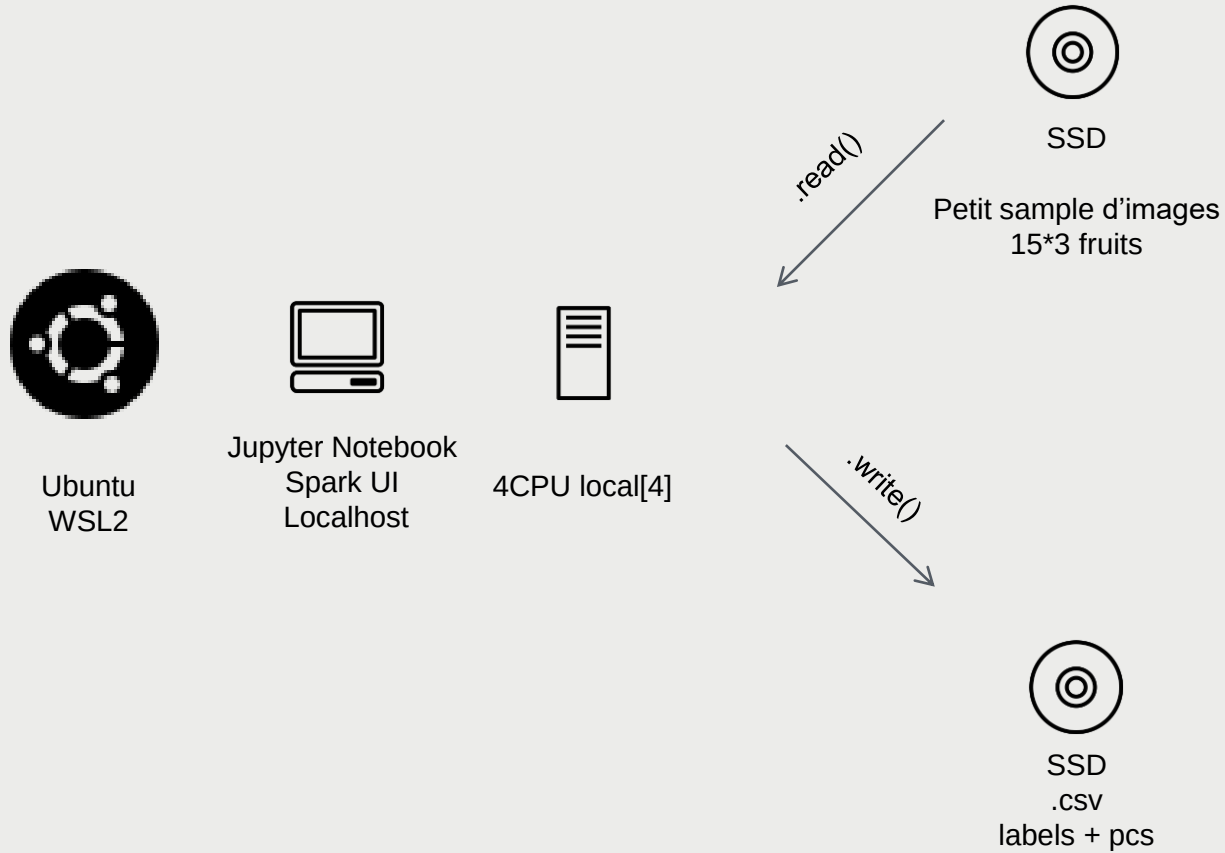
Amazon Simple Storage Service (Amazon S3)

AWS CLI

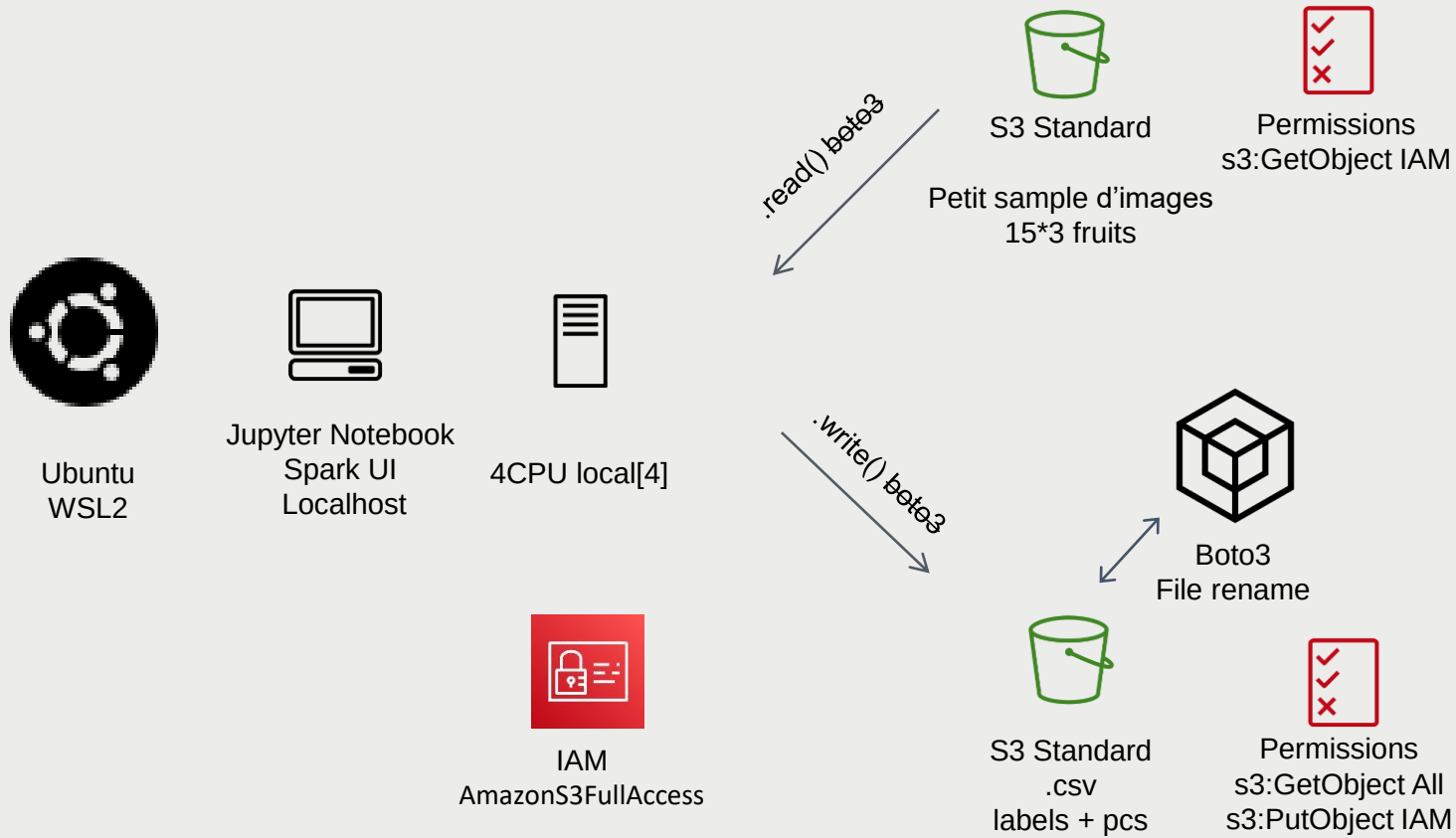
Amazon EMR

Amazon Kinesis Data Firehose

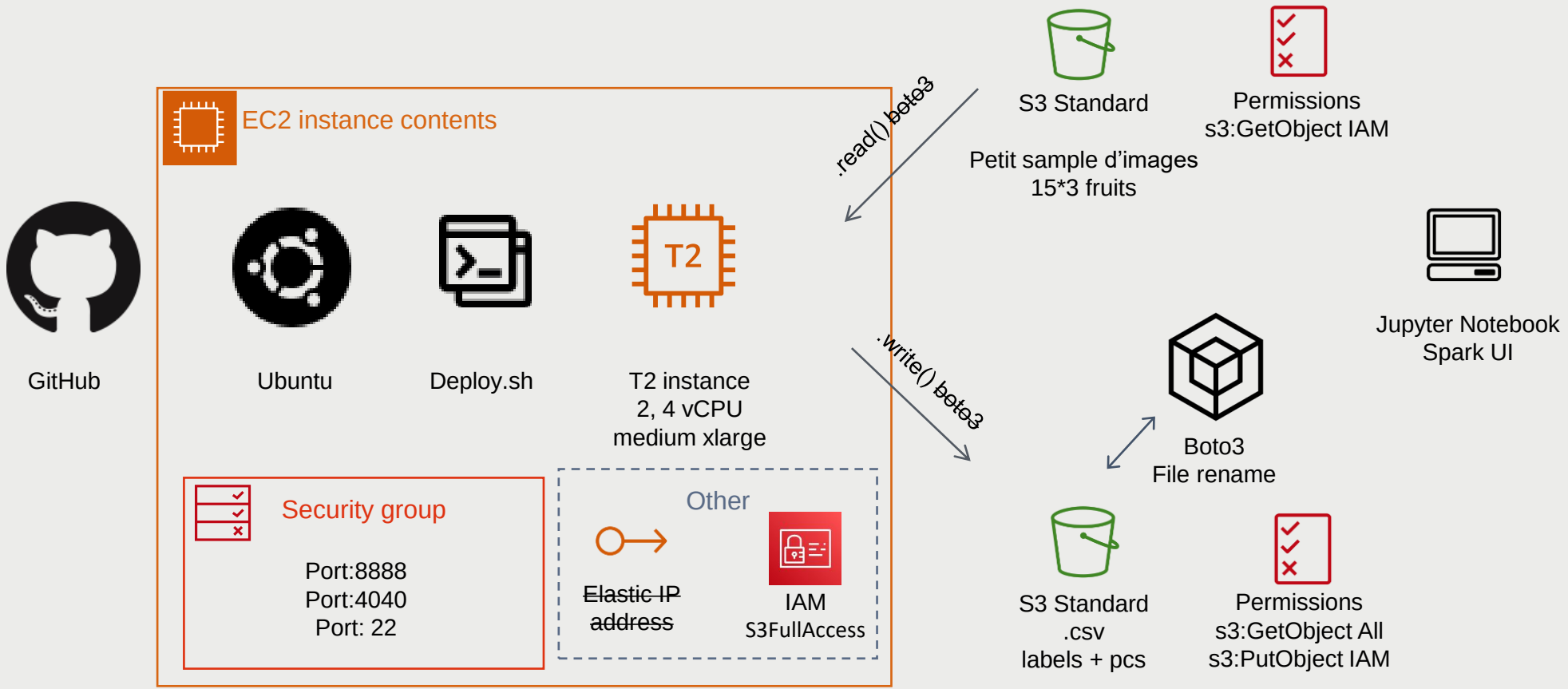
Prototype d’architecture mis en place en local



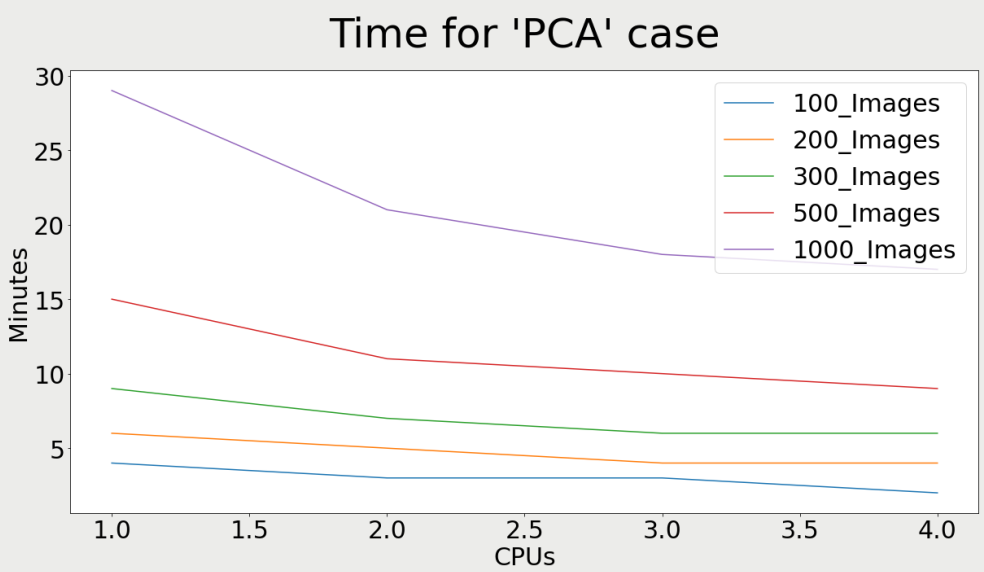
Prototype d'architecture mis en place transition vers AWS

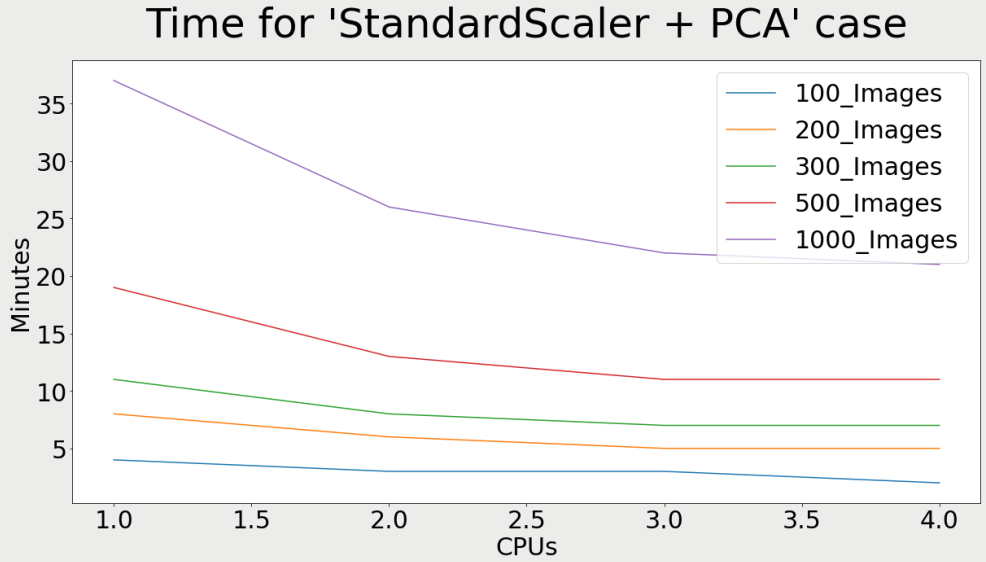


Prototype d'architecture mis en place sur AWS



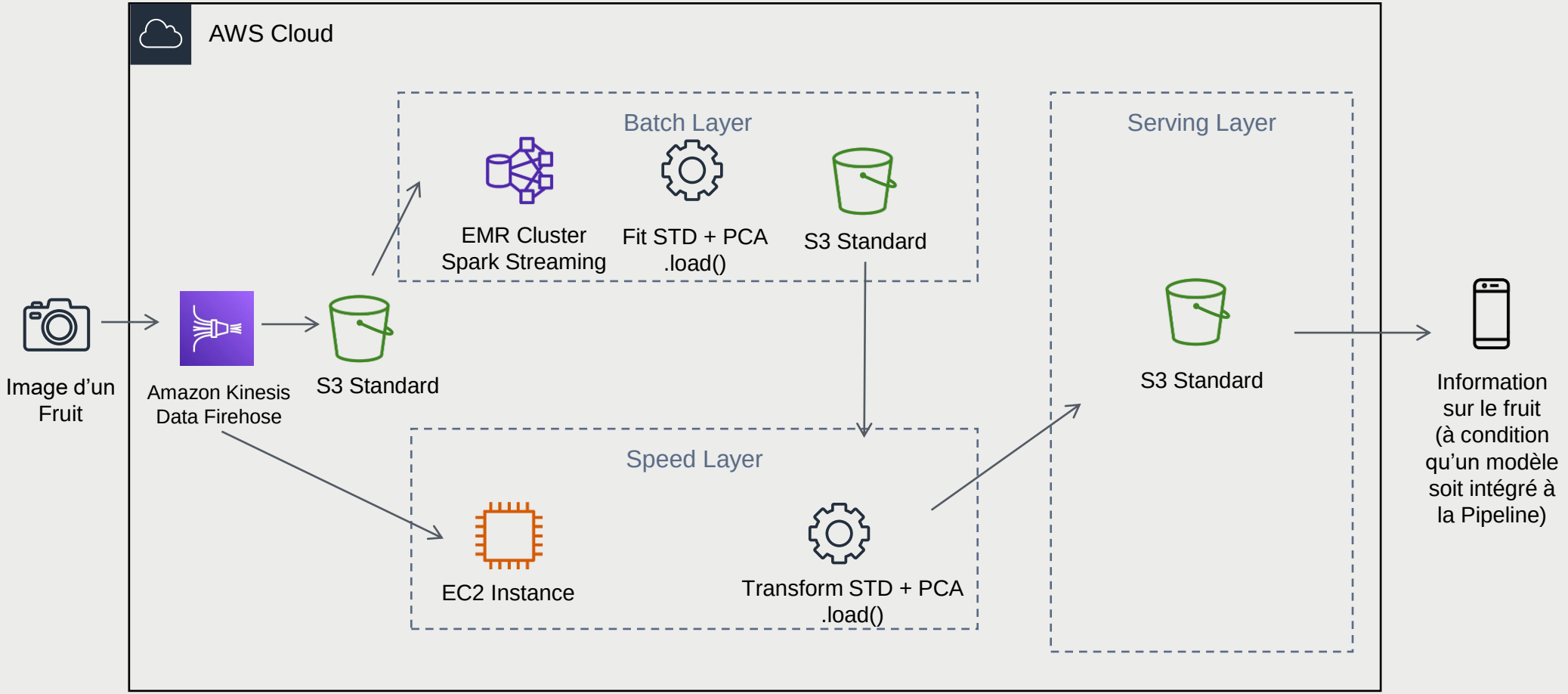
Monitoring





t2.xlarge (4vCPU)
Port 4040 Spark UI
Amélioration performance possible : Tensorflow sur GPU
(entre autres)

Architecture lambda (adaptée)



Gestion des coûts

Services AWS utiles :

AWS Cost Explorer

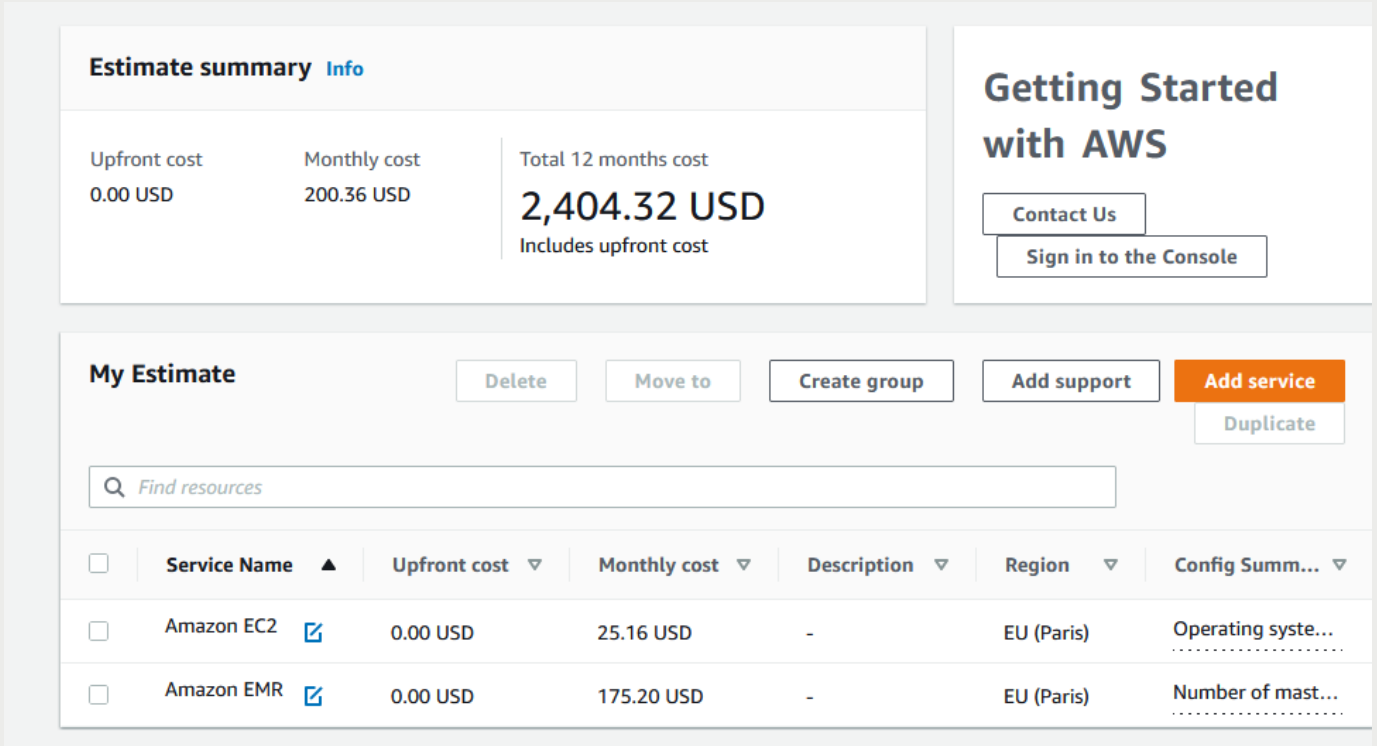
.

Pour Monitorer

. Pour Mieux apprehender coûts plus difficiles à anticiper dans notre cas : Stockage, Transfert de données

Savings Plans
Les instances réservées permettent de faire des économies
Différents types de Stockages

Couts prévisibles minimaux :



Pour aller plus loin dans le projet de fruits!

HTTPS

URLs Présignées

VPC

Amazon Relational Database Service (Amazon RDS)

Amazon DynamoDB

Conclusion

Mise à l'échelle => calcul distribué => PySpark

Images => Feature Extraction + Réduction de dimentionalité (DataFrame, VGG16, ML Pipeline)

Architecture AWS (S3 buckets, EC2) et coûts associés

Développements possibles du projet => (EMR, PySpark Streaming, lambda Architecture, ML pipeline avec un modèle, Base de données [SQL, noSQL], sécurité de l'architecture)

Merci pour votre attention !