



Déployez un modèle dans le cloud

Projet 8 du parcours
« Data Scientist » d'OpenClassrooms

Mark Creasey

Sommaire

Déployez un modèle dans le cloud

- Problématique
- Big data et les outils
- Infrastructure pour la traitement
- La chaîne de traitement
- Conclusion et perspectives



1. Problématique

Contexte

- Fruits ! Start-up de l'AgriTech
- Robots pour automatiser traitement / récolte des fruits selon variétés
- Reconnaissance des fruits à partir d'une photo
 - Mobile app - sensibiliser publique à la biodiversité des fruits
 - Robots - Mise à l'échelle d'un plus grand volume de données (Big data)



Objectifs

Architecture Big Data dans le Cloud

- **premières briques** d'une chaîne de traitement des données
 - Preprocessing des images
 - Réduction de dimension des features

Anticiper

- L'augmentation du volume de données
- L'utilisation des calculs distribués

Le jeu de données



<https://www.kaggle.com/datasets/moltean/fruits>
<https://github.com/Horea94/Fruit-Images-Dataset>

Données entraînement/test

- 90380 images de 131 fruits (600 Mb)
- labelisés (label = nom du dossier)
- photographiés a plusieurs angles
- fond d'image éliminé (en blanc)
- en couleur 100px * 100 px * 3 (R,G,B)

Échantillon

- 5 fruit, 5 images
- **Minimiser les coûts** de stockage et traitement pendant développement

2. Big Data et les outils

Les défis de Big Data

Volume

- trop de données pour stocker dans une seule machine
 - système de fichiers distribués

Vélocité

- traitement des images générés (temps réel)
 - calcul en parallèle

Variété

- hétérogénéité de formats de données
 - (images, videos, réseaux, IoT, logs,...)

Autres considérations

valeur, véracité, validité, viabilité,
venue, vocabulaire, volatilité, viscosité,
viralité, visualisation

Hadoop : Calcul distribué en disk



<https://hadoop.apache.org/>

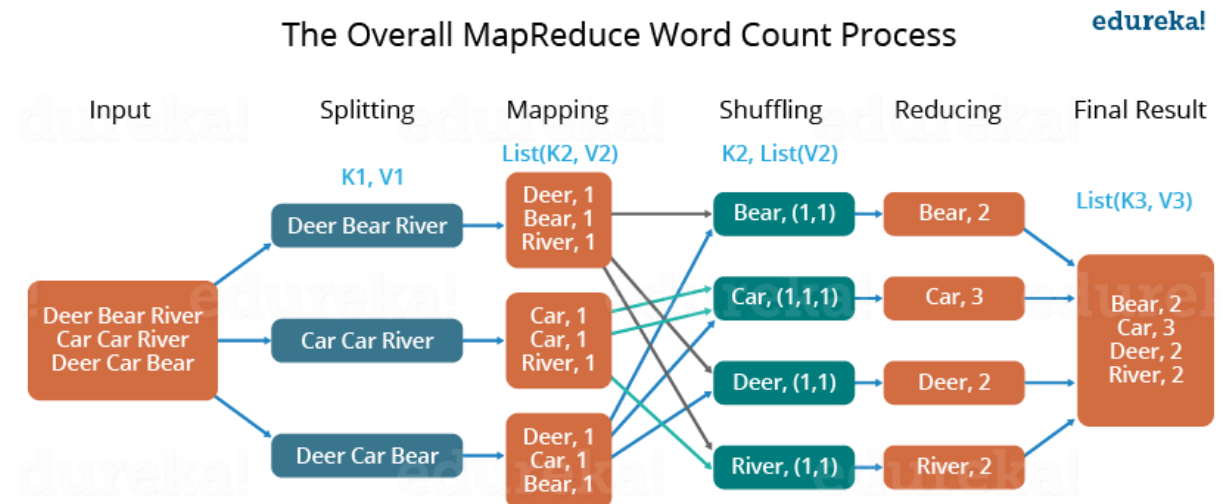
- Permettre l'exécution en parallèle
- Gere la distribution et réplication de tâches
- Tolérance aux pannes de hardware
- l'état de nœud en échec est récréée (1 fois) et relancé sur un autre nœud
- Les machines connectent aux données, les données ne sont pas distribuées aux machines
- Basé sur MapReduce
 - Map: transformation
 - Shuffle : redistribution
 - Reduce : agrégation
- Librairie de source ouvert

HDFS

- Hadoop Distributed File System

EMRFS

- Système de fichiers AWS basé sur HDFS
 - intégration avec S3, encryption, accès IAM

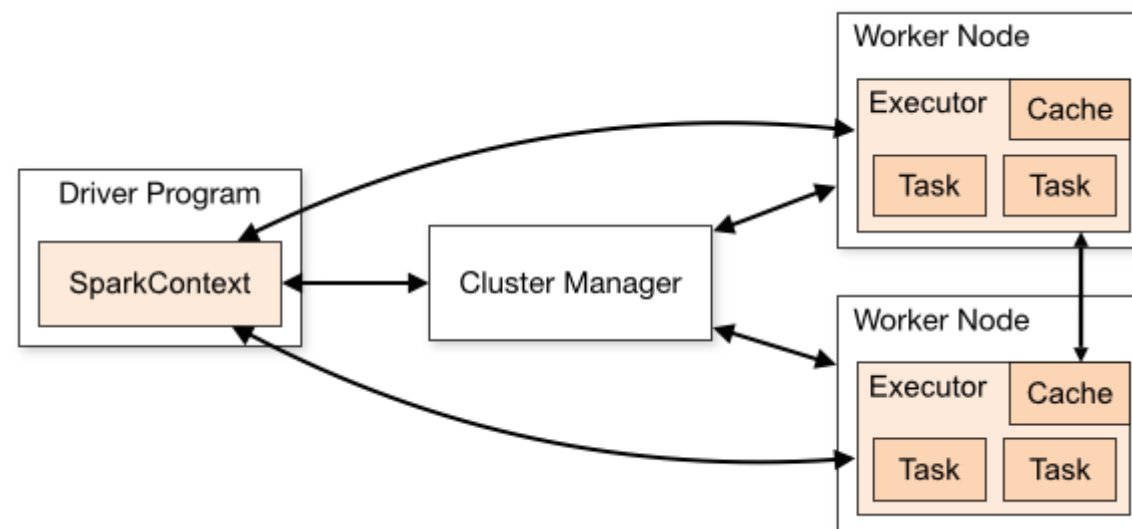


Spark – Calcul distribué en mémoire RAM



<https://spark.apache.org/>

- Mise en cache RAM, parallélisation, optimisation des requêtes
- Jusqu'à 100 fois plus rapide que Hadoop
- Limité à la taille des processeurs
- Basé sur Hadoop MapReduce pour gestion entre processeurs/ machines
- APIs pour java, python (PySpark) R, ...
- Ecrit en scala:
 - strongly typed, functional language
 - moins de risque d'erreur
 - lazy evaluation
- Librairie de source ouvert



<https://spark.apache.org/docs/latest/running-on-mesos.html>

Concepts Spark

SparkSession

- une application Spark, avec configuration bootstrap défini par l'utilisateur

Dataframes

- tables (views) avec colonnes nommées

RDD (Resilient Distributed Dataset)

- objets immuables derrière les Dataframes
- distribuées sur plusieurs processeurs / serveurs

Partitions

- découpage de données pour exécuter une tâche en parallèle

Transformations

- passage d'un RDD vers une autre RDD
 - map, filter, select, where, ..

UDF (User Defined Function)

- Exemples : vectorization, extraction des features d'un modèle de transfer learning

Actions

- Lance l'évaluation des transformations
 - reduce, count, show, collect, ...

Job (tâche)

- une arborescence (DAG) de transformations et actions à faire

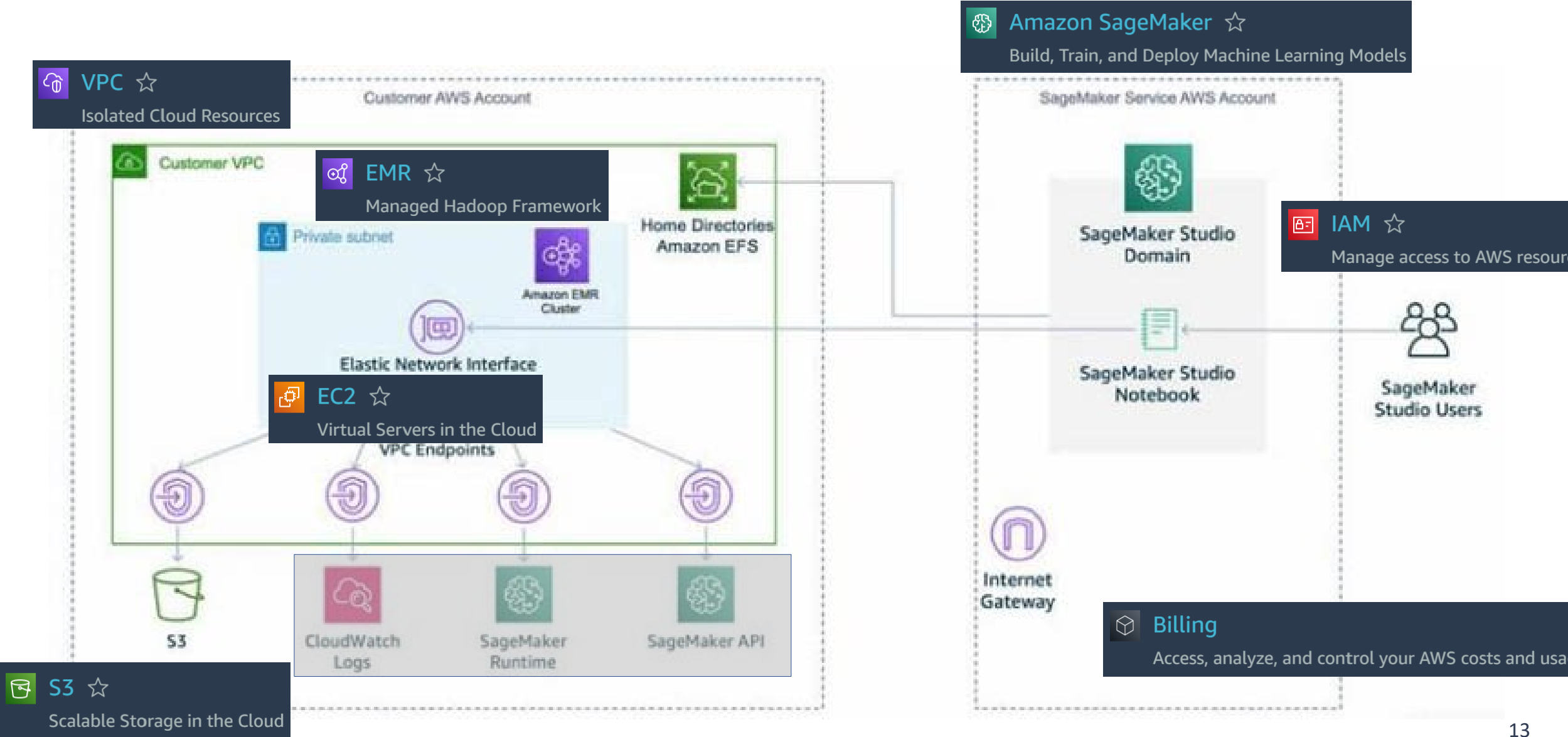
3. L'architecture dans le cloud

Choix du cloud

- Large gamme de puissant processeurs, machines, stockage, environnements
- Elasticité : rapide modification des capacités
- Louer des puissant CPU pour des secondes, minutes, heures
- Stockage illimité, avec réplication, encryption, gestion d'accès
- Gestion de couts - Facturation à l'utilisation



Infrastructure AWS utilisé



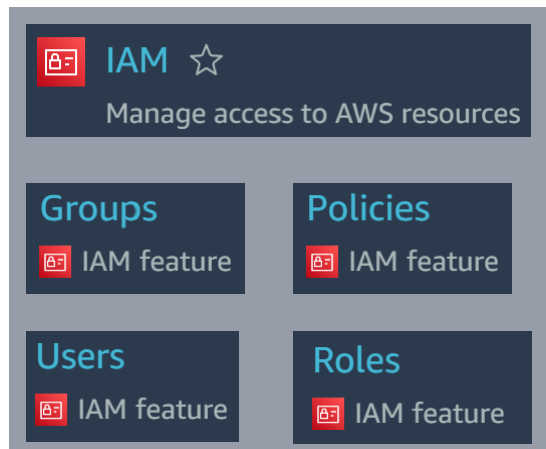
Amazon IAM (Identity and Access Management)

Gestion des droits

- des utilisateurs aux différents services

Etapes

1. Créer **stratégie** d'autorisation
2. Créer **rôle** composé de stratégies
3. Créer **groupe** utilisateur composé de rôles
4. Créer **utilisateur** composé de groupes / rôles



[Policies](#) > AmazonSageMaker-ExecutionPolicy-20220707T161932

Summary

Policy ARN arn:aws:iam::125534713363:policy/service-role/AmazonSageMaker-I

Description

Permissions

Policy usage

Tags

Policy versions

Access Advisor

Policy summary

{ } JSON

Edit policy

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Action": [
6         "s3:ListBucket"
7       ],
8       "Effect": "Allow",
9       "Resource": [
10        "arn:aws:s3:::mc-oc-8"
11      ]
12    },
13    {
14      "Action": [
15        "s3:GetObject",
16        "s3:PutObject",
17        "s3:DeleteObject"
18      ],
19      "Effect": "Allow",
20      "Resource": [
21        "arn:aws:s3:::mc-oc-8/*"
22      ]
23    }
24  ]
25 }
```

Amazon S3 (Amazon Simple Storage Service)



Amazon Simple Storage Service (Amazon S3)

Stockage des données

- Bucket (nom unique) = (Key-Value store)
- Objets: images, fichiers texte, données
- Capacité illimitée
- Haute disponibilité (données répliquées)

Pour chaque objet

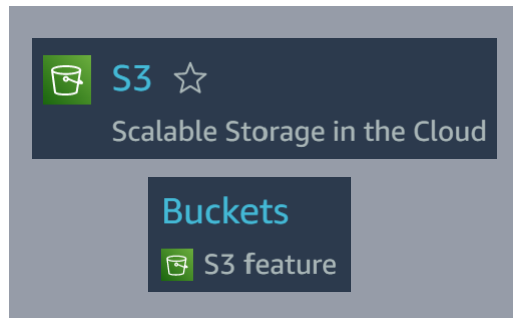
- Droits d'accès,
- chiffrement
- Versioning des objets

Prix dépend de

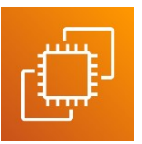
- Vitesse de réponse (Standard, Glacier,..)
- Localité/ réplication
- Fréquence d'accès (volume des requêtes)

Accessible via

- REST (bibliothèque boto),
- S3FS
- AWS CLI
- AWS console (Web)



Amazon EC2 (Amazon Elastic Compute Cloud)



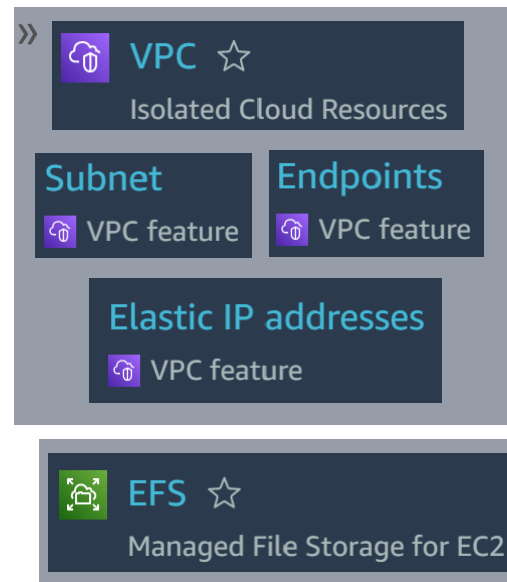
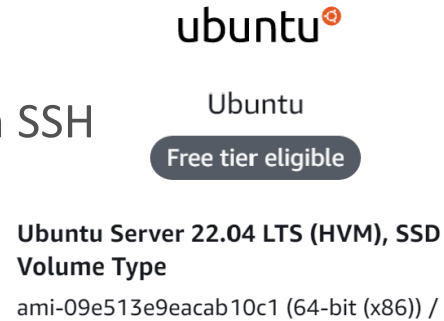
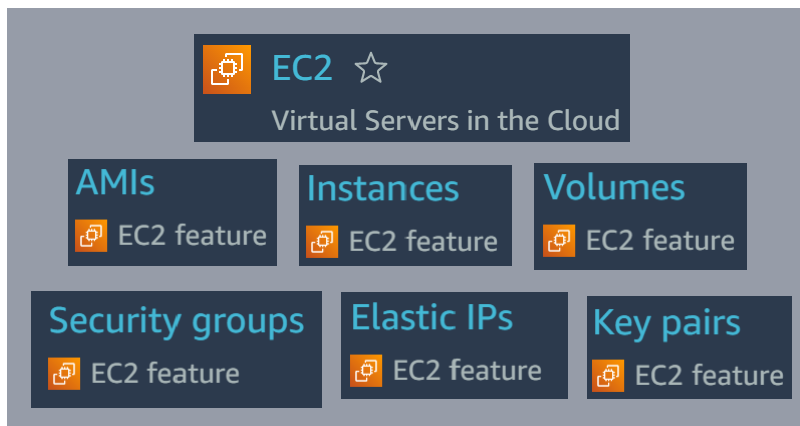
Amazon Elastic Compute
Cloud (Amazon EC2)

Les machines

- Choix: Ubuntu AMI (amazon machine image)
- Création d'un clé privée pour connexion SSH
- Bootstrap (Anaconda, Java, Spark)
- Installer librairies de l'environnement
- Configurer jupyter lab server

Echec: Besoin d'un certificat SSL:

- Scripts PySpark dans Jupyter Lab ne marche pas avec des certificats « self-signed »



Configuration machine

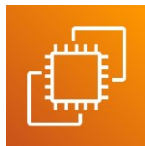
- <https://towardsdatascience.com/how-to-run-a-spark-application-from-an-ec2-instance-a4584d4d490d>

```
sudo apt-get update
wget https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh
bash Anaconda3-2022.05-Linux-x86_64.sh
sudo apt install openjdk-8-jre-headless
sudo apt install scala
wget https://d1cdn.apache.org/spark/spark-3.0.3/spark-3.0.3-bin-hadoop2.7.tgz
sudo tar -zxvf spark-3.0.3-bin-hadoop2.7.tgz
mv spark-3.0.3-bin-hadoop2.7 /home/ubuntu/
```

Configuration jupyter lab server

- <https://docs.aws.amazon.com/dlami/latest/devguide/set-up-jupyter-config.html>

EC2 – pré-configurés sont plus chers



Amazon Elastic Compute
Cloud (Amazon EC2)

2. Create a self-signed SSL certificate. Follow the prompts to fill out your locality as you see fit. You must enter `.` if you wish to leave a prompt blank. Your answers will not impact the functionality of the certificate.

```
$ cd ~
$ mkdir ssl
$ cd ssl
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out mycert.pem
```


Note

You might be interested in creating a regular SSL certificate that is third party signed and does not cause the browser to give you a security warning. This process is much more involved. Visit [Jupyter's documentation](#) for more information.

<https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter-config.html>

The table shows current software and infrastructure pricing for services hosted in **EU (Ireland)**. Additional taxes or fees may apply.

NVIDIA GPU-Optimized AMI				
EC2 Instance type		Software/hr	EC2/hr	Total/hr
<input checked="" type="radio"/>	p3.2xlarge ★Vendor Recommended	\$0	\$3.305	\$3.305
<input type="radio"/>	p3.8xlarge	\$0	\$13.22	\$13.22
<input type="radio"/>	p3.16xlarge	\$0	\$26.44	\$26.44
<input type="radio"/>	p3dn.24xlarge	\$0	\$33.711	\$33.711
<input type="radio"/>	p4d.24xlarge	\$0	\$35.397	\$35.397



NVIDIA GPU-Optimized AMI

By: [NVIDIA](#) Latest Version: 22.03.0

The NVIDIA GPU-Optimized AMI is an environment for running the GPU-accelerated deep learning and HPC containers from the NVIDIA NGC catalog. The deep learning containers from NGC catalog require this AMI for GPU acceleration on AWS P4d, P3, G4dn, G5 GPU instances.

Amazon EMR (Amazon Elastic MapReduce)



Amazon EMR

Clusters de machines EC2, gérés par Amazon

- avec Hadoop pre-installés
- Simplifie la création de clusters
- Met à disposition des serveurs de calcul sur mesure avec les environnements déjà installés. (Hadoop, Spark, YARN...)
- Possibilité d'ajouter un bootstrap au démarrage pour installer les librairies nécessaires au script.

- <https://medium.com/analytics-vidhya/how-to-create-a-hadoop-cluster-free-in-aws-cloud-a95154980b11>

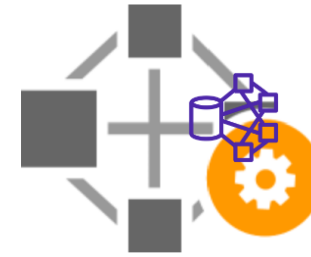
<https://console.aws.amazon.com/elasticmapreduce/>
How Elastic MapReduce Works

Upload



Upload your data and processing application to S3.

Create

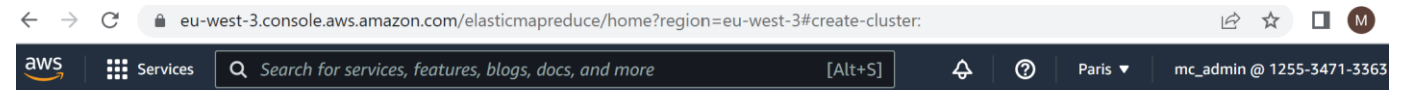


Configure and create your cluster by specifying data inputs, outputs, cluster size, security settings, etc.

Monitor



Monitor the health and progress of your cluster. Retrieve the output in S3.



Create Cluster - Advanced Options [Go to quick options](#)

Step 1: Software and Steps

Step 2: Hardware

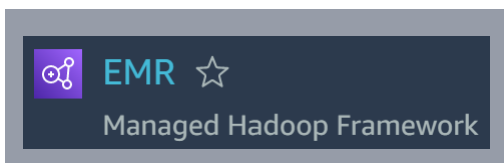
Step 3: General Cluster Settings

Step 4: Security

Software Configuration

Release emr-6.7.0

- | | | |
|---|---|--|
| <input checked="" type="checkbox"/> Hadoop 3.2.1 | <input type="checkbox"/> Zeppelin 0.10.0 | <input checked="" type="checkbox"/> Livy 0.7.1 |
| <input checked="" type="checkbox"/> JupyterHub 1.4.1 | <input type="checkbox"/> Tez 0.9.2 | <input type="checkbox"/> Flink 1.14.2 |
| <input type="checkbox"/> Ganglia 3.7.2 | <input type="checkbox"/> HBase 2.4.4 | <input checked="" type="checkbox"/> Pig 0.17.0 |
| <input checked="" type="checkbox"/> Hive 3.1.3 | <input type="checkbox"/> Presto 0.272 | <input type="checkbox"/> ZooKeeper 3.5.7 |
| <input type="checkbox"/> JupyterEnterpriseGateway 2.1.0 | <input type="checkbox"/> MXNet 1.8.0 | <input type="checkbox"/> Sqoop 1.4.7 |
| <input checked="" type="checkbox"/> Hue 4.10.0 | <input type="checkbox"/> Phoenix 5.1.2 | <input type="checkbox"/> Trino 378 |
| <input type="checkbox"/> Oozie 5.2.1 | <input checked="" type="checkbox"/> Spark 3.2.1 | <input type="checkbox"/> HCatalog 3.1.3 |
| <input checked="" type="checkbox"/> TensorFlow 2.4.1 | | |



Amazon SageMaker

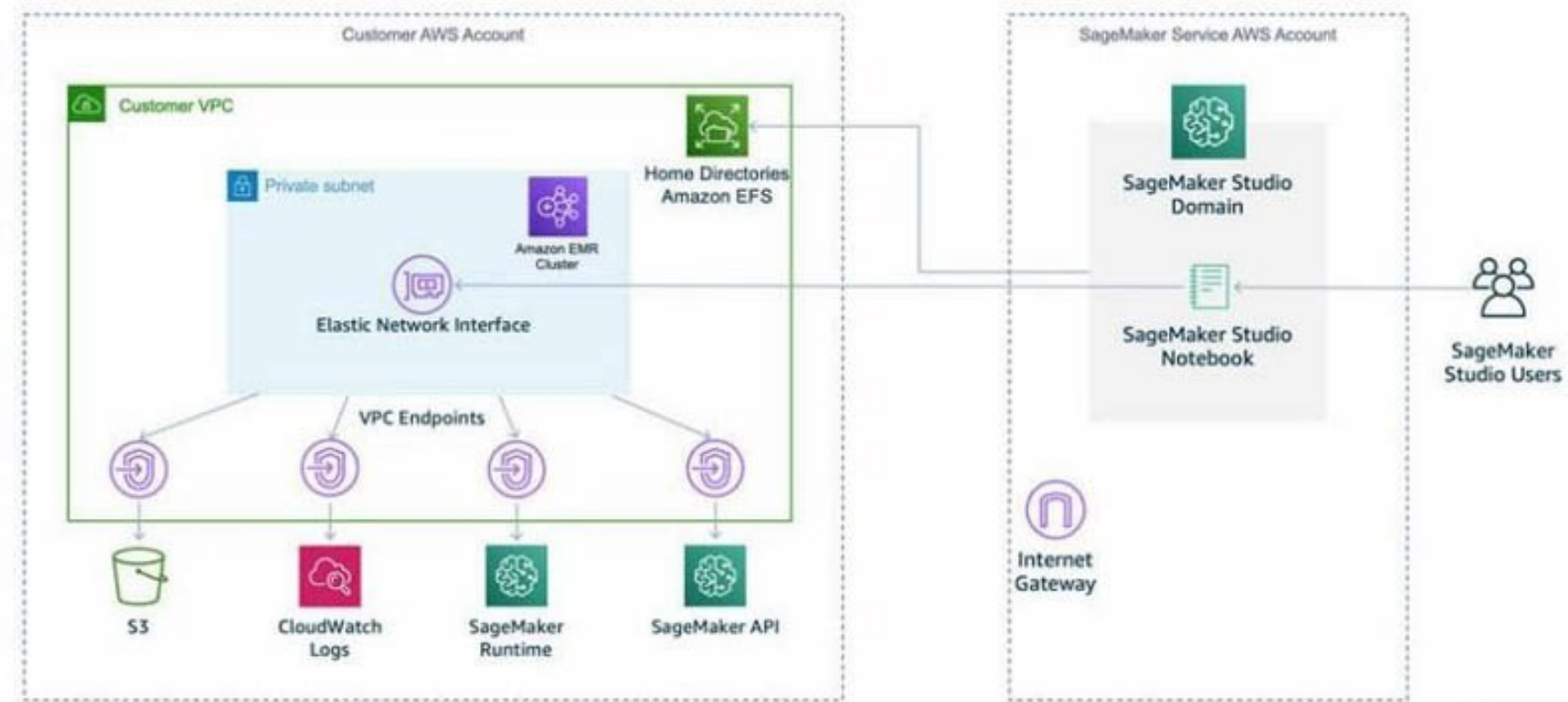


Amazon SageMaker

Service AWS dédié à la data science, outils pour faciliter :

- Préparation de données
- Développement en Jupyter Notebooks
 - environnements conda préconfigurés
- Entraînement sur clusters EMR instantanées
- Déploiement en production
- Monitoring (Logs)

- <https://aws.amazon.com/blogs/machine-learning/perform-interactive-data-engineering-and-data-science-workflows-from-amazon-sagemaker-studio-notebooks/>
- <https://aws.amazon.com/blogs/machine-learning/build-amazon-sagemaker-notebooks-backed-by-spark-in-amazon-emr/>



Amazon SageMaker ☆

Build, Train, and Deploy Machine Learning Models

SageMaker Studio

Amazon SageMaker feature

Etapas dans la création de l'environnement (Sagemaker)



Amazon SageMaker

- Création d'un **rôle IAM et utilisateur avec la stratégie** « AmazonSageMakerFullAccess »
- Création d'un **rôle de service** pour créer SageMaker Notebook instance
 - (sinon - il faut donner vos clés au service – risque de fuit de vos clés)
- Création d'un **bucket S3**
- Création **instance notebook Jupyter** - Sagemaker avec **rôle de service**
- Clone du dossier git
- Créer un nouveau jupyter notebook
- Choisir conda environnement pré-installé
 - CO

Notebook Instance en Sagemaker

- Facile d'arrêter la machine, changer puissance du CPU, puis redémarrer

The screenshot displays the Amazon SageMaker console interface. The top section shows the IAM role configuration for 'AmazonSageMaker-ExecutionRole-20220707T161932'. It includes a summary of the role's creation date (July 07, 2022, 16:19 UTC+02:00) and its ARN. Below this, the 'Permissions' tab is active, showing two attached policies: 'AmazonSageMaker-ExecutionPolicy-20220707T161932' (Customer managed) and 'AmazonSageMakerFullAccess' (AWS managed). The bottom section shows the 'Notebook instances' page, which lists a single instance named 'sagemaker-nb-instance-oc-8' with a status of 'InService' and a creation time of 'Jul 07, 2022 20:56 UTC'. The instance is running on an 'ml.m5.xlarge' instance type. The 'Actions' column for this instance provides links to 'Open Jupyter' and 'Open JupyterLab'.

IAM > Roles > AmazonSageMaker-ExecutionRole-20220707T161932

AmazonSageMaker-ExecutionRole-20220707T161932

SageMaker execution role created from the SageMaker AWS Management Console.

Summary

Creation date
July 07, 2022, 16:19 (UTC+02:00)

ARN
arn:aws:iam::125534713363:role/service-role/AmazonSageMaker-ExecutionRole-20220707T161932

Last activity
13 minutes ago

Maximum session duration
1 hour

Permissions Trust relationships Tags Access Advisor Revoke sessions

Permissions policies (2)

You can attach up to 10 managed policies.

Filter policies by property or policy name and press enter

Policy name	Type	Description
AmazonSageMaker-ExecutionPolicy-20220707T161932	Customer managed	
AmazonSageMakerFullAccess	AWS managed	Provides full access to /

Amazon SageMaker > Notebook instances

Notebook instances

Search notebook instances

Name	Instance	Creation time	Status	Actions
sagemaker-nb-instance-oc-8	ml.m5.xlarge	Jul 07, 2022 20:56 UTC	InService	Open Jupyter Open JupyterLab




Coûts EC2

▼ Elastic Compute Cloud		\$0.00
▶ No Region		-\$0.37
▼ EU (Paris)		\$0.37
Amazon Elastic Compute Cloud running Linux/UNIX		\$0.37
\$0.0528 per On Demand Linux t2.medium Instance Hour	7.023 Hrs	\$0.37
EBS		\$0.00
\$0.00 per GB-month of General Purpose (gp3) provisioned storage under monthly free tier	1.950 GB-Mo	\$0.00

Coûts Sagemaker

▼ SageMaker		\$0.00
▶ No Region		-\$2.07
▼ EU (Paris)		\$2.07
Amazon SageMaker CreateVolume-Gp2		\$0.07
\$0.1624 per GB-Mo of Notebook Instance ML storage	0.456 GB-Mo	\$0.07
Amazon SageMaker RunInstance		\$2.00
\$0.00 for Notebk:ml.t3.medium per hour under monthly free tier	19.239 Hrs	\$0.00
\$0.00 for Studio-Notebook:ml.t3.medium per hour under monthly free tier	2.378 Hrs	\$0.00
\$0.00 per Data Wrangler Interactive ml.m5.4xlarge hour under monthly free tier	0.788 Hrs	\$0.00
\$0.269 per Notebook ml.m5.xlarge hour in EU (Paris)	7.425 Hrs	\$2.00



Billing

Access, analyze, and control your AWS costs and usage.

4. La chaine de traitement PySpark / S3

Les étapes à mettre en place

Stockage des données à traiter dans S3 (images).

- Chargement d'images dans S3 (à partir de Github (ou transfert local via AWS CLI)
- Créer une session Spark

Pour chaque image

- Charger l'image de S3
- Preprocess l'image
- Extraire des features

Avec ensemble des images features

- Reduction des dimensions (PCA)
- Classify images with classifier
- Envoie les résultats vers S3

Note: L'action `show()` exécute le pipeline pour visualiser l'effet de chaque transformation pendant developpement

Initialisation du notebook

Import des librairies

Paramètres utilisateur

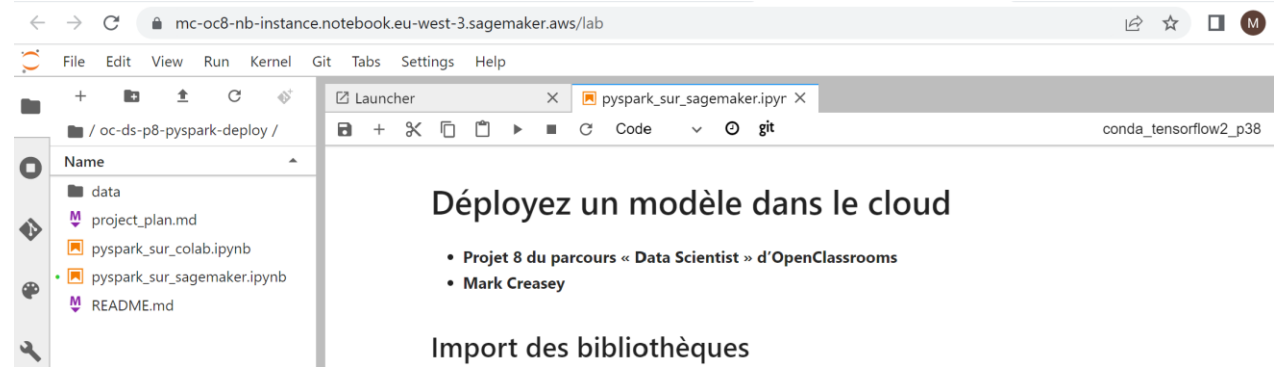
- Chemins data / résultats
 - BUCKET
 - FOLDERS

Recuperation des clés d'accès à S3 (service rôle)

- `Botocore.session().credentials`
- `resource(s3)`

```
[5]: sess = sagemaker.Session()
default_bucket=sess.default_bucket()
aws_role = sagemaker.get_execution_role()
aws_region = boto3.Session().region_name
print(f'aws_region : {aws_region}')
print(f'region : {sess.boto_region_name}')
print(f'account_id : {sess.account_id()}')
print(f'default_bucket : {sess.default_bucket()}')
print(f'aws_role : {aws_role}')

aws_region : eu-west-3
region : eu-west-3
account_id : 125534713363
default_bucket : sagemaker-eu-west-3-125534713363
aws_role : arn:aws:iam::125534713363:role/AWSGlueServiceSageMakerNotebookRole-Default
```



```
[2]: import sagemaker # session
import sagemaker_pyspark
import pyspark # spark
import boto3 # S3 bucket
import io
import os

from pyspark import SparkContext, SparkConf
from pyspark.sql import DataFrame, SparkSession
from typing import List
import pyspark.sql.types as T
import pyspark.sql.functions as F

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf # model
```

```
[3]: from platform import python_version

print(f'python version = {python_version()}')
print('versions des bibliothèques utilisées:')
print(' '.join(f'{m.__name__}=={m.__version__}' for m in globals()
).values() if getattr(m, '__version__', None)))

python version = 3.8.12
versions des bibliothèques utilisées:
sagemaker==2.86.2; pyspark==3.0.0; boto3==1.21.42; PIL.Image==9.0.1; numpy==1.20.3; tensorflow==2.7.1
```

Configuration

```
[4]: S3_BUCKET='mc-oc-8'
DATA_FOLDER='data/train'
OUT_FOLDER='output'

print(S3_BUCKET)
```


Démarrage de la session Spark

- Paramétrage d'accès S3
- Permet de créer et manipule les RDD, Dataframe

```
[7]: # Configure Spark to use the SageMaker Spark dependency jars
from pyspark import SparkConf
jars = sagemaker_pyspark.classpath_jars()

classpath = ":".join(sagemaker_pyspark.classpath_jars())
conf = SparkConf().set("spark.driver.extraClassPath", classpath)

# See the SageMaker Spark Github repo under sagemaker-pyspark-sdk
# to learn how to connect to a remote EMR cluster running Spark from a Notebook Instance.
spark = (
    SparkSession.builder
    .config(conf=conf)
    .config('spark.hadoop.fs.s3a.access.key', credentials.access_key)
    .config('spark.hadoop.fs.s3a.secret.key', credentials.secret_key)
    .config('spark.hadoop.fs.s3a.method', credentials.method)
    .config('spark.hadoop.fs.s3a.impl', 'org.apache.hadoop.fs.s3a.S3AFileSystem')
    .master("local[*]")
    .appName("Fruit Image Classification")
    .getOrCreate()
)

spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version	v3.0.0
Master	local[*]
AppName	Fruit Image Classification

```
22/07/10 17:14:40 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

Initialisation d'un spark Dataframe (évaluation « lazy »)

- Liste des chemins ('path') des objets à chargé
- il faut une action comme show() pour démarrer

reference: <https://spark.apache.org/docs/latest/rdd-programming-guide.html>

```
[16]: sc = spark.sparkContext
      filelist_rdd = sc.parallelize(filelist, numSlices=4)
      print (type(filelist_rdd))
      print(filelist_rdd.getNumPartitions())
```

```
<class 'pyspark.rdd.RDD'>
4
```

recover original data

```
[17]: filelist_collect = filelist_rdd.collect()
      print (type(filelist_collect))
      filelist_collect[:3]
```

```
[Stage 0:> (0 + 4) / 4]
<class 'list'>
```

```
[17]: ['data/train/Blueberry/0_100.jpg',
      'data/train/Blueberry/120_100.jpg',
      'data/train/Blueberry/180_100.jpg']
```

Create a pyspark DataFrame

```
[18]: from pyspark.sql import Row
      filelist_df = filelist_rdd.map(lambda x: Row(x)).toDF(['path'])
      print(type(filelist_df))
      filelist_df.printSchema()
```

```
<class 'pyspark.sql.dataframe.DataFrame'>
root
 |-- path: string (nullable = true)
```

Chargement des images (évaluation « lazy »)

- Map : Extraction de la classe ('label') pour chaque image à partir du chemin
- Map : Load image ('binaryData') en bytes pour chaque chemin → un array de bytes
- Map : Preprocess (array de bytes) → autre array de bytes
- UDF : Transformation des features Vectorize
Convert binary data to vectors

```
[32]: from pyspark.ml.linalg import Vectors, VectorUDT
      from pyspark.sql.functions import udf # user defined function

      udf_vectorized = udf(lambda r: Vectors.dense(r), VectorUDT())
      df_images = df_images.withColumn("vector_data", udf_vectorized(F.col("binary_data")))
      df_images.printSchema()
      df_images.show(4)
```

```
root
 |-- path: string (nullable = true)
 |-- label: string (nullable = true)
 |-- binary_data: string (nullable = true)
 |-- vector_data: vector (nullable = true)
```

```
[Stage 16:>                                     (0 + 1) / 1]
+-----+-----+-----+-----+
|      path      | label | binary_data | vector_data |
+-----+-----+-----+-----+
|data/train/Bluebe...|Blueberry|[255, 255, 255, 2...|[255.0,255.0,255....|
|data/train/Bluebe...|Blueberry|[255, 255, 255, 2...|[255.0,255.0,255....|
```

```
[20]: df_images = (filelist_df.withColumn('label',F.element_at(F.split(filelist_df['path'],"/"),3)))

      df_images.show(2,False)
```

```
+-----+-----+
|path      |label |
+-----+-----+
|data/train/Blueberry/0_100.jpg |Blueberry|
|data/train/Blueberry/0_101.jpg |Blueberry|
+-----+-----+
[29]: only
```

```
def load_binary(file_key):
    """generic method to get any object from S3"""
    s3 = boto3.resource('s3')
    bucket= s3.Bucket(S3_BUCKET)
    #print('file_key',file_key)
    obj = bucket.Object(key=file_key)
    response=obj.get()
    im =response['Body']

    # Until we are able to create correct binary data structure
    # for input to VGG16 - using spark.read.format('image')
    # we simulate model.predict(image) --> 1024 features
    # reduce number of features as we are testing on a small machine (2GB RAM)

    img =Image.open(im).resize([16,16])
    # si on ne change pas en list --> erreur

    return np.array(img).flatten().tolist()

print(load_binary(object_key)[:40])
```

```
[252, 255, 253, 253, 255, 253, 253, 255, 253, 253, 255, 254, 235, 226, 215, 214,
39, 186, 160, 136, 181, 167, 146, 181, 167, 148, 207, 192, 176, 237, 234, 228, 2]
```

```
[30]: # Convert a Python function to PySpark UDF
      # udf_binary= F.udf(load_binary,T.ArrayType(T.DoubleType()))
      udf_binary= F.udf(load_binary)

      df_images = df_images.withColumn('binary_data', udf_binary(F.col('path')))
      df_images.printSchema()
```

Initialisation d'un modèle VGG16 pré-entraîné

- Note: Fonctionnel si on charge avec `spark.read.image()`
 - voir colab notebook sur google colab.
 - <https://docs.databricks.com/applications/machine-learning/preprocess-data/transfer-learning-tensorflow.html>
- Créer Model: VGG16 sans la couche finale.
- Broadcast des poids du CNN à tous les nœuds du cluster dans spark context
- Map : Predict (preprocessed image) → features

```
[33]: import tensorflow as tf
from tensorflow.keras.applications import vgg16
from tensorflow.keras.preprocessing.image import img_to_array
```

```
[34]: def create_vgg16_model():
    model_ = vgg16.VGG16(
        include_top=False, # Couche softmax de classification supprimée
        weights='imagenet', # Poids pré-entraînés sur Imagenet
        pooling='max' # Utilisation du max de pooling
    )
    return model_

model=create_vgg16_model()
bc_model_weights = spark.sparkContext.broadcast(model.get_weights())

def model_vgg100_fn():
    """
    Returns a VGG16 model with top layer removed and broadcasted pretrained weights.
    """
    model_ = vgg16.VGG16(include_top=False, weights=None, pooling='max')
    model_.set_weights(bc_model_weights.value)
    return model_
```

```
[35]: import numpy as np
import pandas as pd

def preprocess_vgg100(content):
    """
    Preprocesses raw image bytes for prediction.
    """
    arr= np.array(content)
    #img = Image.open(io.BytesIO(content)).resize([100, 100])
    #arr = img_to_array(img)
    return vgg16.preprocess_input(arr)

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)
```

```
from pyspark.sql.functions import col, pandas_udf, PandasUDFType
from typing import Iterator

@pandas_udf('array<float>')
def featurize_udf(content_series_iter: Iterator[pd.Series])>Iterator[pd.Series]:
    """
    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()
```

```
# Pandas UDFs on large records (e.g., very large images) can run into Out Of Memory (OOM) errors.
# If you hit such errors in the cell below, try reducing the Arrow batch size via `maxRecordsPerBatch`
spark.conf.set("spark.sql.execution.arrow.maxRecordsPerBatch", 1024)
```

```
# We can now run featurization on our entire Spark DataFrame.
# NOTE: This can take a long time (about 10 minutes) since it applies a large model to the full dataset
df_features = df_images.repartition(16).select(col("path"), featurize_udf("content").alias("features"))
df_features.write.mode("overwrite").parquet("tmp/fruit_image_features")
```

Reduction de dimension avec Spark.mlib

- Mise à l'échelle

```
[40]: from pyspark.ml.feature import StandardScaler

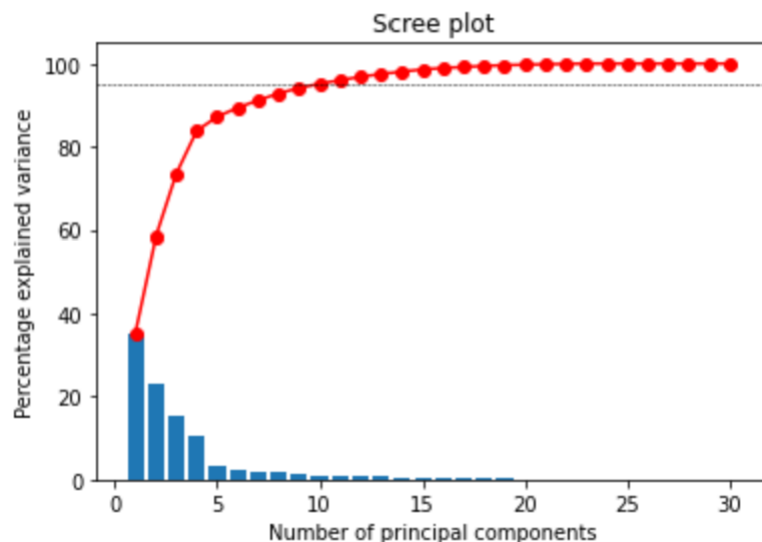
scaler = StandardScaler(inputCol='features', outputCol='scaled_features', withStd=True, withMean=True)
model = scaler.fit(df_features)
df_scaled = model.transform(df_features)
```

- PCA

```
[42]: from pyspark.ml.feature import PCA

pca = PCA(k=30, inputCol='scaled_features', outputCol='pca_features')
pca_model = pca.fit(df_scaled)
```

```
display_scee_plot(pca_model)
```



- Sélectionner top 15 composantes

```
[44]: pca = PCA(k=15, inputCol='scaled_features', outputCol='pca_features')
pca_model = pca.fit(df_scaled)

df_pca_features = pca_model.transform(df_scaled)
df_pca_features.show()
```

[Stage 31:=====>

(1 + 2) / 3]

Enregistrement des résultats sur S3

Convertir en fichier .csv, puis envoi au S3

```
[45]: df_pandas = df_pca_features.toPandas()
```

```
[46]: df_pandas.head(1)
```

	path	label	features	scaled_features	pca_features
0	data/train/Blueberry/0_100.jpg	Blueberry	[255.0, 255.0, 255.0, 255.0, 0.4377975178854348, 0.764...	[0.7582875444051542, 11.498376432906252, 0.49...	[-29.410668360678567, 255...

```
[47]: def write_dataframe_to_csv_on_s3(dataframe, bucket, filename):  
    """ Write a dataframe to a CSV on S3 """  
    # Create buffer  
    csv_buffer = io.StringIO()  
    # Write dataframe to buffer  
    dataframe.to_csv(csv_buffer, sep=",", header=None, index=None)  
    # Create S3 object  
    s3_resource = boto3.resource("s3")  
    # Write buffer to S3 object  
    s3_resource.Object(bucket, filename).put(Body=csv_buffer.getvalue())  
    print(f'Writing {len(dataframe)} records to {filename} on bucket s3a://{bucket}')  
  
write_dataframe_to_csv_on_s3(df_pandas, S3_BUCKET, f'{OUT_FOLDER}/resultats_pca.csv')
```

```
Writing 24 records to output/resultats_pca.csv on bucket s3a://mc-oc-8
```

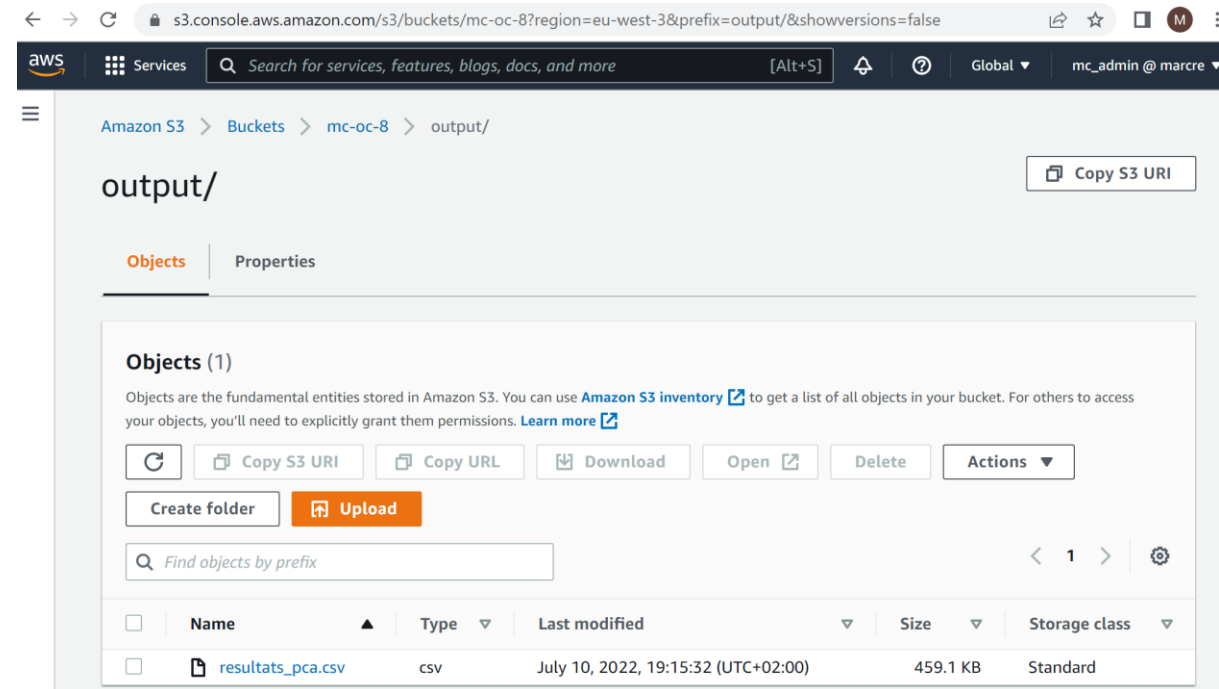
```
[48]: !aws s3 ls s3://mc-oc-8/output/ --recursive --human-readable --summarize
```

```
2022-07-10 17:15:32 459.1 KiB output/resultats_pca.csv
```

```
Total Objects: 1  
Total Size: 459.1 KiB
```

Vérification des résultats

- Présence de fichier CSV



Conclusions et perspectives

Conclusions

- Google Colab/Databricks Community Edition
 - Apprentissage gratuit de PySpark, tutoriels sur plateforme Databricks
- S3 - Stockage des données volumineuses à faible coût.
- EC2 - Configuration des serveurs Ubuntu pour
 - développement à faible coût
 - Personnalisation des librairies ML dans les instances EC2 dans des clusters EMR
- Sagemaker Notebook Instance
 - Rapidement changer de puissance, environnement conda, développer des modèles
 - Intégrer briques de traitement avec stockage S3
 - Parallélisation des calculs distribués (PySpark, Spark)
 - Mise en place des premières briques de traitement
- EMR
 - Service de création de clusters rapide et facilement scalable si le volume augmente (EMR)

Limites

EC2

- Compatibilité de versions entre les librairies.
- Besoin d'installation d'un certificat SSL qui n'est pas « self-signed »
- Besoin de mettre à jour régulièrement – couts devops / administration

EMR

- Limité sur les bibliothèques disponibles

SageMaker

- peut rapidement devenir très couteuse
- Pas encore trouvé le bon paramétrage pour Spark à fin de lire de S3 via S3FS
 - Très couteuse en requêtes sinon

SageMaker Studio

- pas transparent
- quels services seront lancés par l'ouverture d'un notebook, et à quel prix
- couts commence dès que tu ouvre un notebook, car chaque notebook a son propre serveur EC2
- Boites noires - difficulté des erreurs complexes

Améliorations / Prochaine étapes

Mise en œuvre d'autres briques

- Classification des images
- Entraînement sur EMR cluster
- Tester vitesse sur plus grande quantité de données

Création EMR Cluster

- Conversion de notebook en script.py
- submit job
- test de la chaîne de traitement

Création des EC2 custom pour EMR

- Clusters EMR des serveurs EC2 à partir d'un snapshot

Déploiement en production

- Surveillance via logs / Spark History Server pour évaluer performance

Maîtriser des coûts

- Budgets/ alarmes
- Optimiser taille Go / CPU / nb clusters pour prix, vitesse / nb images

Considère les prix d'autres fournisseurs cloud

Questions

images: Mark Creasey si aucune attribution.

- mrcreasey@gmail.com

Code source: <https://github.com/mrcreasey/oc-ds-p8-spark-deploy>

- Merci !

