



Déploiement d'un modèle
de reconnaissance de

Fruits!

dans le CLOUD

Sommaire

1. Contexte – Objectifs
2. Présentation des données (images)
3. Traitement parallélisé
 - 3.1 Choix de Spark
 - 3.2 Architecture de Spark
 - 3.3 Réduction dimensionnelle en local
4. Traitement à l'échelle - AWS
5. Architecture AWS
6. Réduction dimensionnelle avec AWS



Contexte : Start-up de l'AgriTech: **Fruits!**


Ses fondamentaux :

- Respecter la biodiversité des fruits en leur associant un traitement spécifique.
- Sensibiliser le grand public à la biodiversité.
- Être force d'innovation.



Objectifs de la Start-up **Fruits!**

Court terme :

- Mise en place d'une 1ère app mobile de reconnaissance de fruits.
- Développer cette app dans l'écosystème BigData  pour une future mise à l'échelle du volume de données.

Long terme :




- Développement de robots cueilleurs intelligents.

Données : 2 groupes



Training

- **67 692** images de fruits labellisées.
- 62 espèces de fruits dont 25 fruits totalisent 95 variétés, soit un total de **131** catégories.

exemple :  Banana
 Banana Lady Finger
 Banana Red

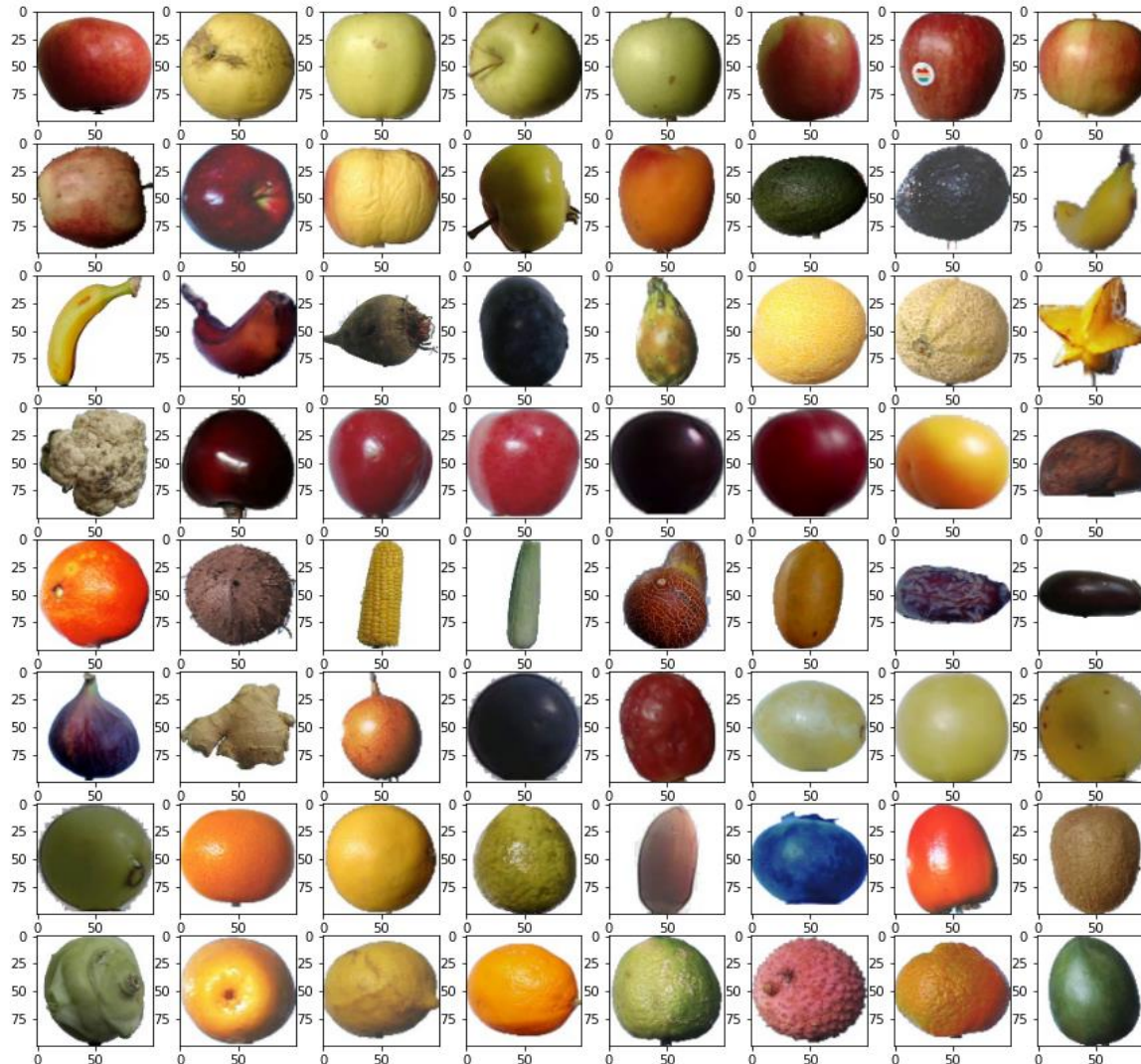
- Fruits présentés sous différents angles.
- 1 fruit/image encouleur sur fond blanc
- Taille de l'image (100 x 100 x 3) pixels.
- image identifiée par son répertoire et son nom



Test

- **22 688** images de fruits.
- **131** catégories.
- 1 fruit/image encouleur sur fond blanc
- Taille de l'image (100 x 100 x 3) pixels.

Données : description



- . Les fruits sont centrés dans l'image
- . les images sont de qualité acceptable
- . les images sont de même taille



- . Pas de normalisation de la taille des images
- . Pas de correction de l'exposition des images
- . Pas d'amélioration du contraste

Fruits - Espèces

pomme (13)	abricot	avocat (2)	banane (3)	betterave	myrtille	figue de barbarie
choufleur	cerise (6)	châtaigne	clementine	noix de coco	maïs (2)	concombre (3)
figue	gingembre	fruit de la passion (3)	raisin (6)	pamplemousse (3)	goyave	noisette (2)
chou-rave	kumquat	citron(3)	lychee (2)	mandarine	mangue (2)	mangoustan
mûre	nectarine (2)	noix de pecan	oignon (3)	orange	papaye	pêche (3)
physalis (2)	ananas (2)	pitahaya rouge	prune (3)	grenade	patate (4)	coing
salak	fraise (2)	Tamarillo	tangelo	tomate (9)	noix	pastèque
carambole	aubergine	kaki	melon piel de sapo	poivron (4)	groseille	framboise
cantaloup (2)	datte	airelle	maracuja	poire (9)	kiwi	

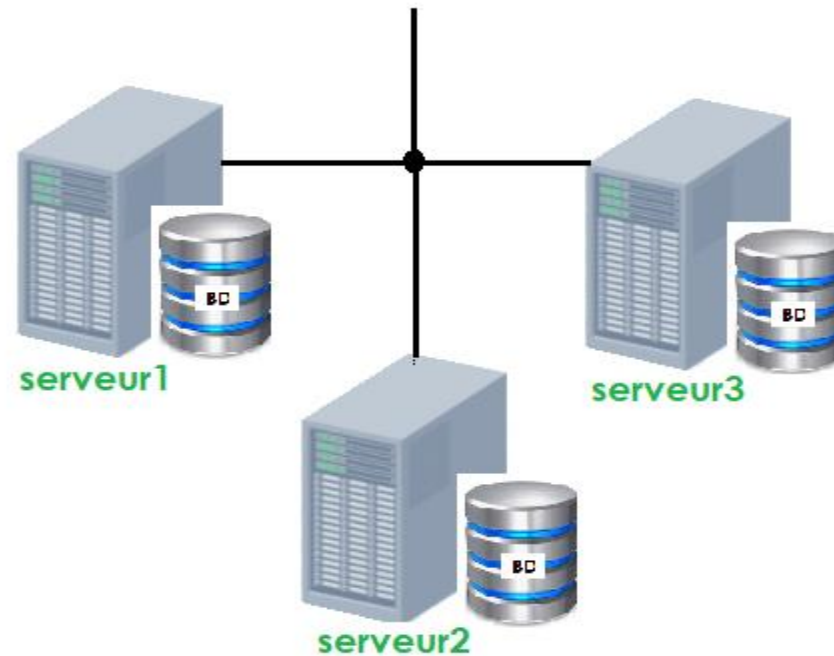
Poids des images (avocat+banane) = **3.21 Mo**

Nombre d'images = **644**

Choix de Spark

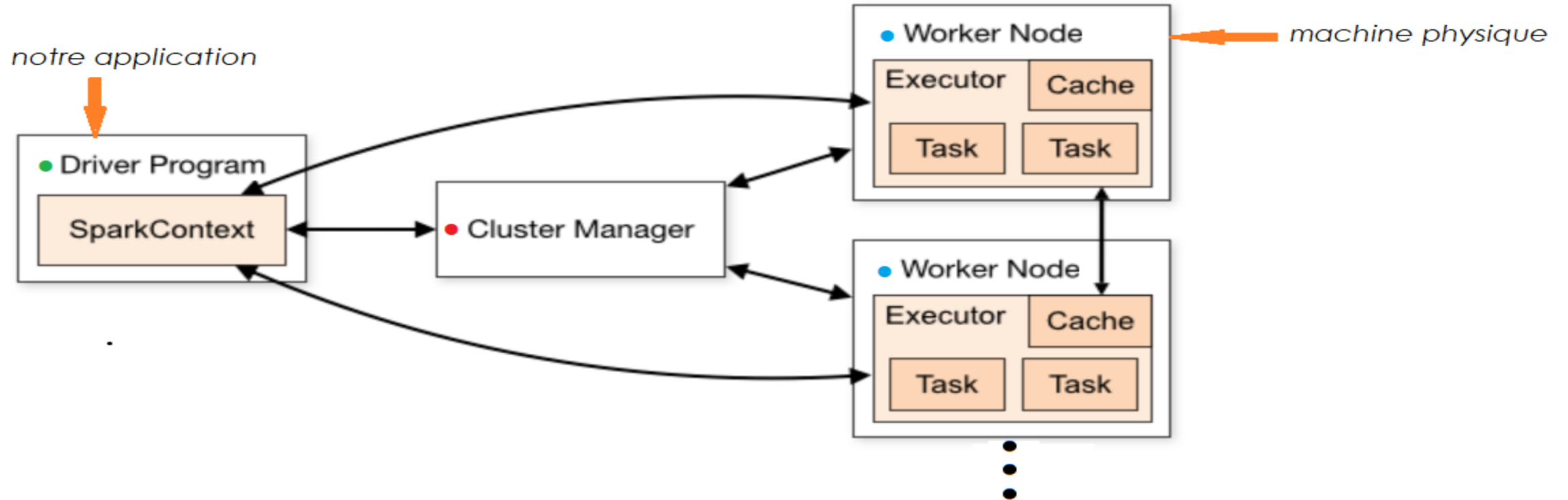


* **PANDAS**: librairie python qui permet de manipuler des données à analyser sur un ordinateur.



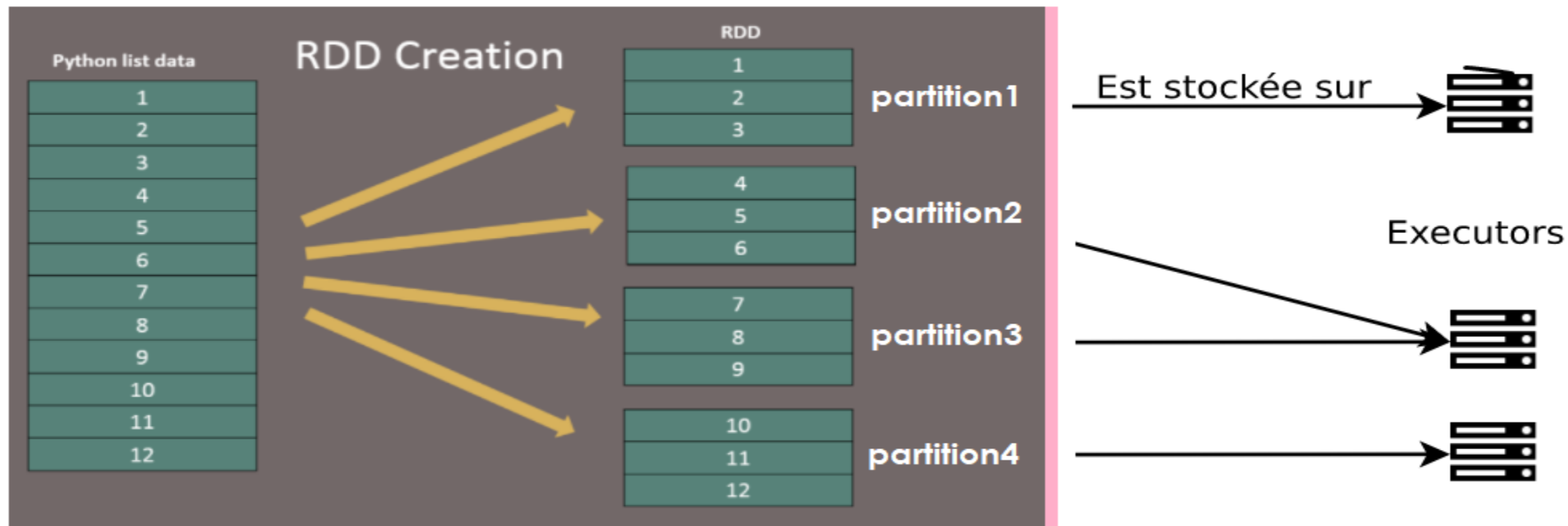
* **SPARK** : framework open-source permettant de paralléliser le traitement de larges volumes de données de manière distribuée.

Architecture de Spark



- **Cluster Manager** : instancie et supervise les différents workers. (gestion des erreurs, ...)
- **Driver Program** : répartie les tâches sur les différents executors.
- **Worker Node** : un worker instancie un executor qui exécute n tâches.

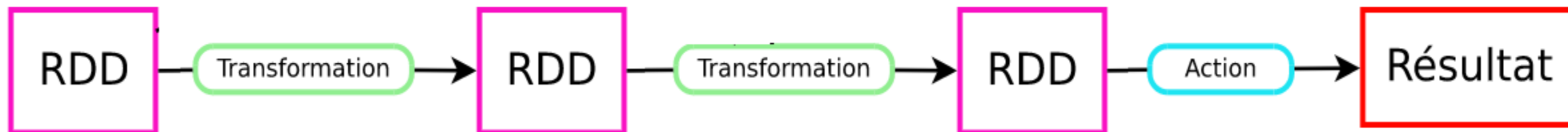
La structure de données essentiel de Spark : le RDD



RDD (**R**esilient **D**istributed **D**ataset) : **collection de données distribuée en lecture seule.**

- Un RDD est organisé en lignes, des blocs non modifiables qui n'excèdent 2 Go.
- Une partition de RDD est un ensemble de lignes traitées par le même processus.
- Un RDD se crée soit à partir d'une source de donnée stable soit depuis un autre RDD.

DAG (Directed Acyclic Graph)



✚ Le **DAG** recense l'ensemble des transformations et actions à réaliser sur les RDD.

Opération	résultat
Transformation	donne en sortie un autre RDD. (ex: map, filter, ...)
Action	donne en sortie... autre chose qu'un RDD. (ex: show, ...)

N.B: Seule une Action provoque l'évaluation des Transformations lazy précédentes.

Réduction dimensionnelle en local

1) Instantiation d'un `contexte_Spark`

```
spark = SparkSession.builder.appName('name').getOrCreate()  
sc = SparkContext.getOrCreate()
```

2) Enregistrement des données dans un `dataframe_Spark`

```
# Création d'un DataFrame PySpark avec une 1ère colonne  
# représentant le chemin complet des images.  
  
lst_path = []  
sub_folders = os.listdir(folder)  
print(sub_folders)  
  
for f in sub_folders:  
    lst_categ = os.listdir(folder+f)  
    for file in lst_categ:  
        lst_path.append(folder+f+"/"+file)  
  
print("Nombre d'images chargées :", len(lst_path))  
  
rdd = sc.parallelize(lst_path)  
row_rdd = rdd.map(lambda x: Row(x))  
df = spark.createDataFrame(row_rdd, [ci])
```

Réduction dimensionnelle en local

Fonctions parallélisées

```
# 2eme colonne du dataframe: catégorie du fruit
```

```
def give_categ(chemin):  
    return chemin.split('/')[-2]
```

```
# 3eme colonne du dataframe: matrice de chaque image mise à plat
```

```
def image_to_list(chemin):  
    try:  
        image = Image.open(chemin)  
        image = np.asarray(image)  
        image = image.flatten().tolist()  
        return image  
    except:  
        return [0]
```

*** retourne la matrice de chaque image mise à plat, soit un tableau (100,100,3) aplati en une liste de taille 3000**

```
# 4eme colonne du dataframe: descripteurs de chaque image
```

```
def give_descripteurs(chemin):  
  
    image = np.asarray(Image.open(chemin))  
    orb = cv2.ORB_create(nfeatures=nombre_variable)  
    keypoints, desc = orb.detectAndCompute(image, None)  
  
    if desc is None:  
        desc = 0  
    else:  
        desc = desc.flatten().tolist()  
    return desc
```

*** retourne les descripteurs de chaque image**

Réduction dimensionnelle en local

Transformation

```
# Remplissage de la colonne catégorie
udf_categ = udf(give_categ, StringType())
df = df.withColumn("categ", udf_categ(ci))*

# Remplissage de la colonne image.
# Chaque image a été transformé en array numpy puis vecteur unitaire
udf_image = udf(image_to_list, ArrayType(IntegerType()))
df = df.withColumn("image", udf_image(ci))*
df = df.filter(df.image.isNotNull())*

# Remplissage de la colonne descripteur avec les descripteurs de
# chaque image transformés en vecteur unitaire
udf_desc = udf(give_descripteurs, ArrayType(IntegerType()))
df = df.withColumn("descripteurs", udf_desc(ci))*
df = df.filter(df.descripteurs.isNotNull())*
```

* Un **UDF** (User Defined Functions) permet de créer une nouvelle colonne dans un dataframe.

Action

```
# Les transformations sont lancées par "df.show"
start_time = time.time()
df.show() *

print("Temps execution %sec ---" % (time.time() - start_time))
```

Réduction dimensionnelle en local

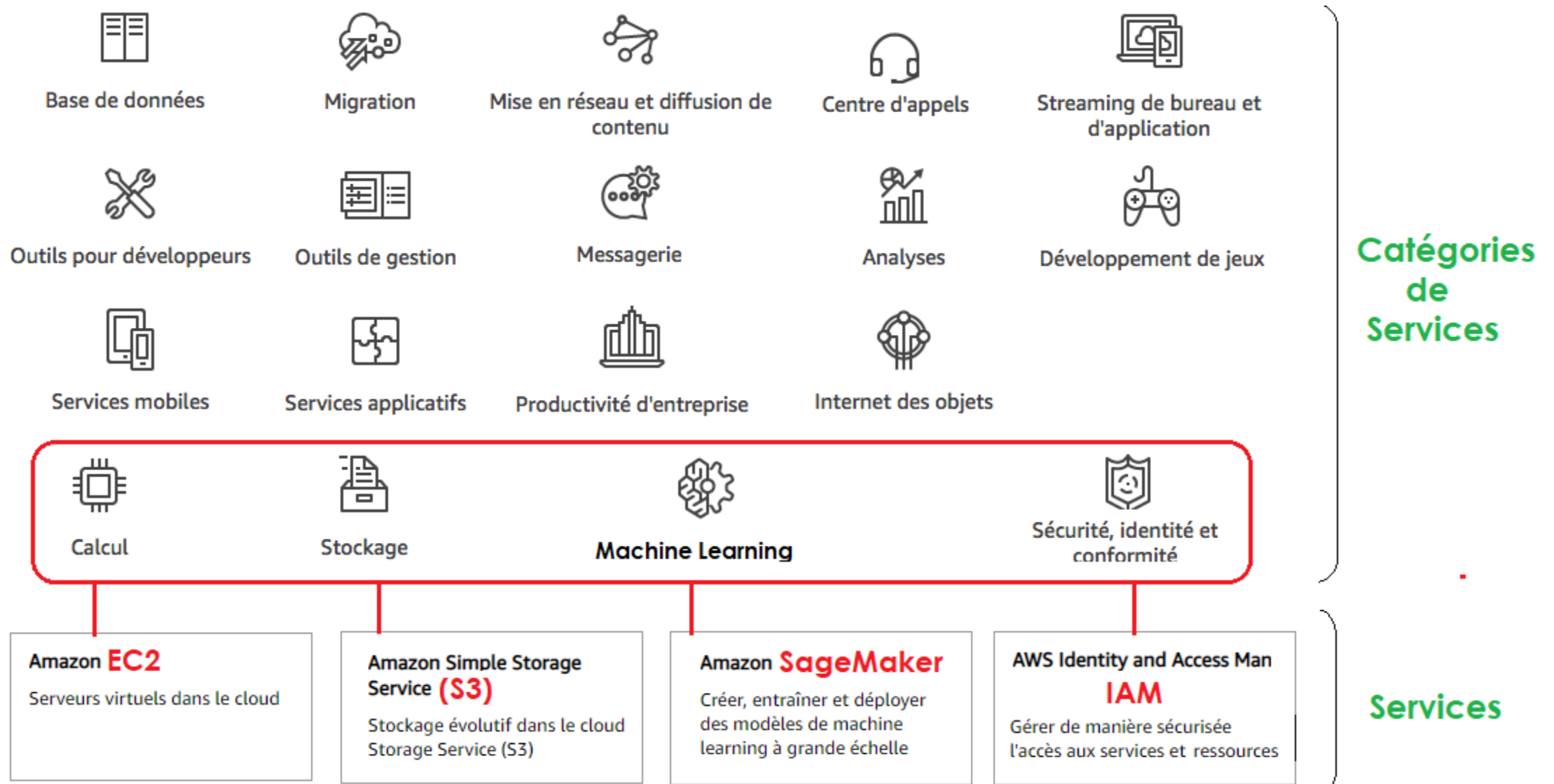
Dataframe final :

chemin_image	categ	image	descripteurs
images/Avocado/0_...	Avocado	[255, 255, 255, 2...	[197, 253, 189, 2...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[212, 236, 159, 2...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[228, 36, 157, 24...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[220, 140, 157, 1...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[244, 36, 189, 25...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[188, 164, 156, 1...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[68, 229, 157, 11...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[124, 165, 29, 11...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[228, 165, 157, 1...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[244, 36, 189, 11...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[84, 228, 159, 23...
images/Avocado/10...	Avocado	[255, 255, 255, 2...	[193, 45, 29, 101...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[229, 245, 157, 1...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[228, 231, 253, 1...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[228, 100, 189, 1...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[245, 100, 157, 9...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[252, 116, 63, 11...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[252, 116, 253, 1...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[228, 164, 156, 1...
images/Avocado/11...	Avocado	[255, 255, 255, 2...	[212, 132, 156, 1...

only showing top 20 rows

Temps execution 7.45508074760437ec ---

Traitement à l'échelle – AWS



Traitement à l'échelle – AWS

Mise en place de notre application sur le cloud AWS en plusieurs étapes :

- ✓ Création d'un compartiment dans S3
- ✓ Chargement des images dans ce compartiment
- ✓ Création d'une instance SageMaker
- ✓ Création d'un rôle permettant à l'instance d'accéder à S3
- ✓ Création d'un notebook Jupyter via SageMaker

N.B : ces étapes se font après la création d'un compte AWS.

AWS - Création d'un compartiment et chargement des Images

Compartiments (1) [Info](#)

Les compartiments sont des conteneurs pour les données stockées dans S3. [En savoir plus](#)

< 1 > ⚙

  Copier l'ARN  Vider  Supprimer  Créer un compartiment




	Nom ▲	Région AWS ▼	Accéder ▼	Date de création ▼
<input type="radio"/>	lcanonne-p8	UE (Irlande) eu-west-1	Compartiment et objets non publics	03 Dec 2021 10:09:50 AM CET

Objets (2)

Les objets sont les entités fondamentales stockées dans Amazon S3. Vous pouvez utiliser l'[inventaire Amazon S3](#) pour obtenir une liste de tous les objets de votre compartiment. Pour que d'autres personnes puissent accéder à vos objets, vous devez leur accorder explicitement des autorisations. [En savoir plus](#)

  Copier l'URI S3  Copier l'URL  Télécharger  Ouvrir  Supprimer  Actions ▼  Créer un dossier  Charger

< 1 > ⚙

<input type="checkbox"/>	Nom ▲	Type ▼	Dernière modification ▼	Taille ▼	Classe de stockage ▼
<input type="checkbox"/>	 images/ 	Dossier	-	-	-
<input type="checkbox"/>	 outputresultat/	Dossier	-	-	-

AWS – Création d'un rôle

Sélectionner une entité de confiance

- ☒ Service AWS
Autorisez les services AWS tels qu'EC2, Lambda ou autre à effectuer des actions dans ce compte.

Cas d'utilisation

Cas d'utilisation courants

- ☒ EC2
Allows EC2 instances to call AWS services on your behalf.



Le rôle créé, permet à l'instance de SageMaker d'accéder aux images du compartiment S3 (read/write) et d'utiliser la puissance des serveurs EC2.

Ajouter des autorisations

Stratégies des autorisations (739)

Choisissez une ou plusieurs stratégies à attacher à votre nouveau rôle.

🔍 Filtrer les stratégies par nom de propriété ou de stratégie et appuyer sur Entrée

<input type="checkbox"/>	Nom de la politique 	Type
<input type="checkbox"/>	 AmazonSageMaker-ExecutionPolicy-20211130T132804	Gérées ...

AmazonSageMaker-ExecutionRole-20211130T132804

SageMaker execution role created from the SageMaker AWS Management Console.

AWS – Création d'une instance SageMaker

Créer une instance de bloc-notes

Paramètres d'instances de blocs-notes

Nom de l'instance de bloc-notes

AmazonSageMaker-InstanceNotebook-20211130T132804

Maximum de 63 caractères alphanumériques. Puis incluent les traits d'union (-), mais pas d'espaces. Doit être unique au sein de votre compte dans une région AWS.

Type d'instance de bloc-notes

ml.t2.medium

Autorisations et chiffrement

Rôle IAM

Les instances de bloc-notes nécessitent des autorisations pour appeler d'autres services, y compris SageMaker et S3. Choisissez un rôle ou laissez-nous en créer un avec la [AmazonSageMakerFullAccess](#) stratégie IAM attachée.

AmazonSageMaker-ExecutionRole-20211130T132804

Annuler

Créer une instance de bloc-notes

N.B: La création de l'instance de bloc-notes prend quelques minutes.

Il faut attendre que son statut passe à "En service" avant d'ouvrir un bloc-note.

AWS – Création d'un notebook Jupyter

jupyter Proj8_pyspark_sagemaker Dernière Sauvegarde : il y a 12 minutes (modifié) Logout

File Edit View Insert Cell Kernel Widgets Help conda_python3

Exécuter Code nbdiff

Choix du kernel

```
# Les transformations sont lancées par "df.show"
start_time = time.time()
df.show()

print("Temps execution %sec ---" % (time.time() - start_time))
```

path_img	categ	image	descriptors
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[196, 164, 156, 1...
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[124, 245, 157, 1...
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[116, 233, 157, 9...
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[220, 140, 157, 1...
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[213, 124, 157, 1...
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[240, 173, 188, 1...
s3://lcanonne-p8/...	Avocado	[255, 255, 255, 2...	[100, 225, 148, 4...

Temps execution 75.36493754386902ec ---

Save df in format parquet then copy to s3 with sagemaker session

```
df.repartition(1).write.mode('overwrite').parquet('resultat') # resultat = name of folder where the dataframe
                                                                # will be stored in sagemaker instance

upload_data = sagemaker.Session().upload_data(bucket=mybucket,
                                                path='resultat',
                                                key_prefix='outputresultat') # local file in sagemaker instance
                                                                # bucket where we stored parquet in s3
print('upload_data: {}'.format(upload_data))
```

upload_data: s3://lcanonne-p8/outputresultat

AWS – Réduction dimensionnelle

Accéder aux images dans le compartiment de S3

Librairie permettant d'accéder au compartiment S3

```
import boto3
```

Variables permettant de construire le chemin complet de localisation des images dans S3

```
mybucket = "lcanonne-p8"  
prefix   = "images"  
folder   = "s3://{}/{}".format(mybucket, prefix)
```

Ouverture d'une session client de S3 sur la région 'eu-west-1'

```
region      = boto3.Session().region_name  
session     = boto3.session.Session(region_name=region)    # Ouverture d'une Session sur la région 'eu-west-1' (Ireland)  
s3_client   = session.client('s3')                        # indique que le client utilise le service S3 de AWS  
s3          = boto3.resource("s3", region_name=region)     # référence sur le service S3
```

Liste des répertoires contenant les images (Avocado/, Banana/)

```
sub_folders = s3_client.list_objects_v2(Bucket=mybucket, Prefix=prefix)
```

AWS – Réduction dimensionnelle

Enregistrement des Résultats

```
df.repartition(1).write.mode('overwrite').parquet('resultat') # resultat = name of folder where the dataframe  
# will be stored in sagemaker instance
```

- * Choix du format de stockage **Apache Parquet** adapté aux très gros volumes de données
- * Compression des données
- * De bonnes performances en lecture et en écriture

Enregistrement du résultat dans S3

```
upload_data = sagemaker.Session().upload_data(bucket=mybucket,  
                                              path='resultat', # local file in sagemaker instance  
                                              key_prefix='outputresultat') # bucket where we stored parquet in s3  
print('upload_data: {}'.format(upload_data))
```


Conclusion & Perspectives.