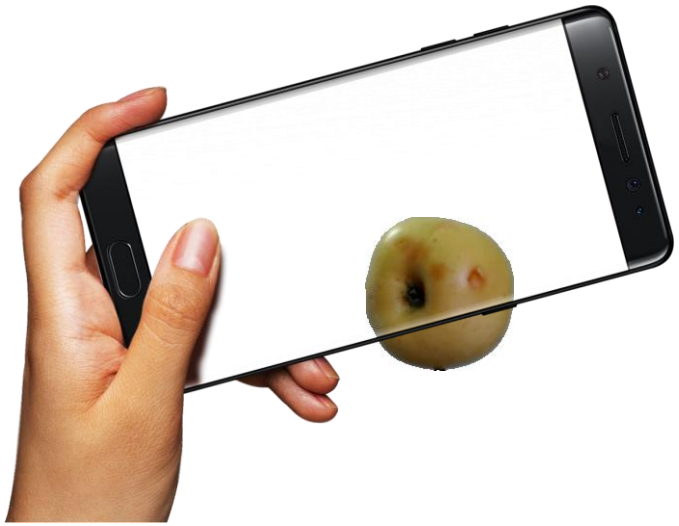
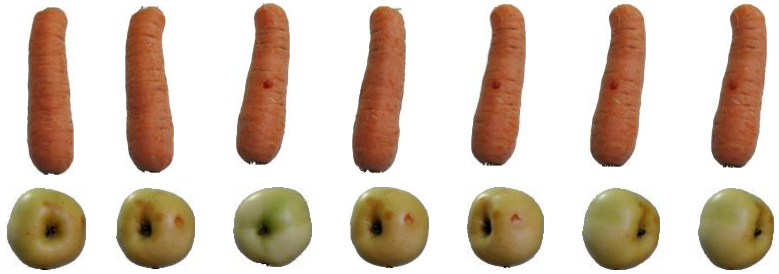
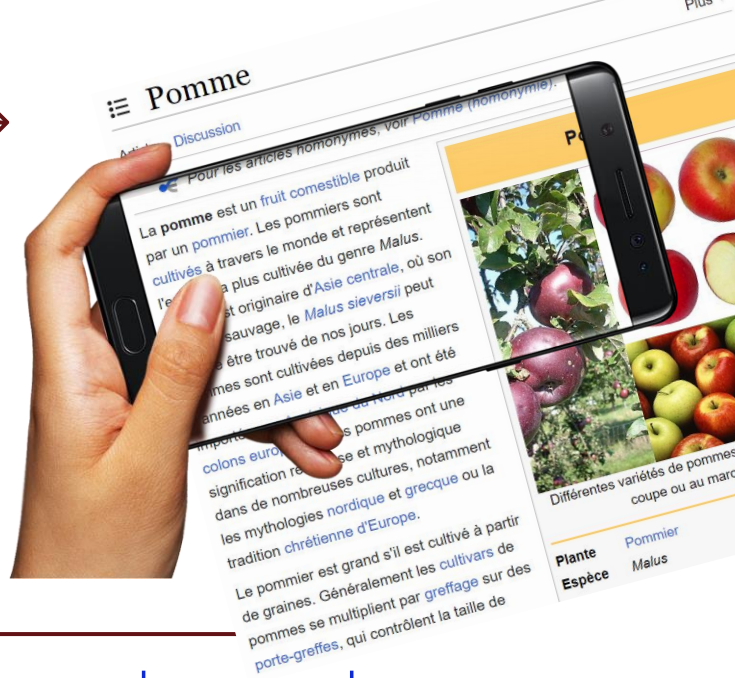
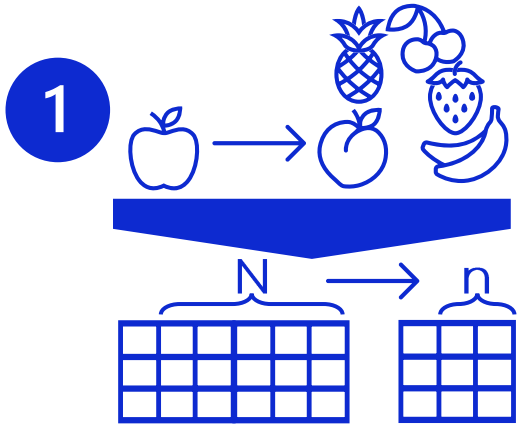


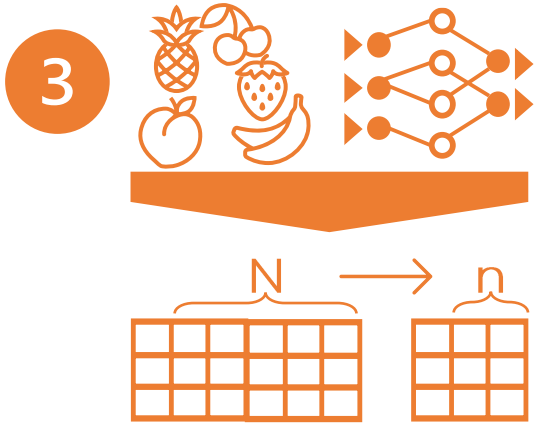


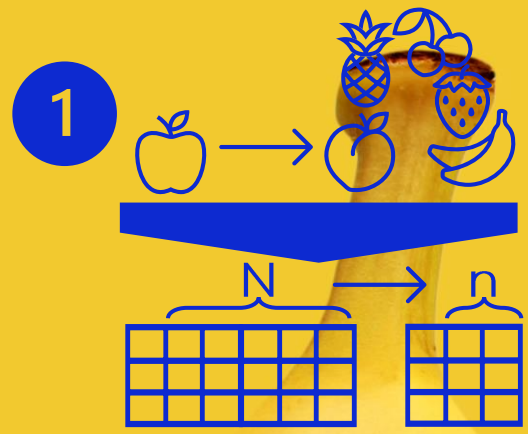


Déploiement d'un modèle dans le cloud

Sébastien De Rosa | OpenClassrooms | Data Scientist | Projet 8







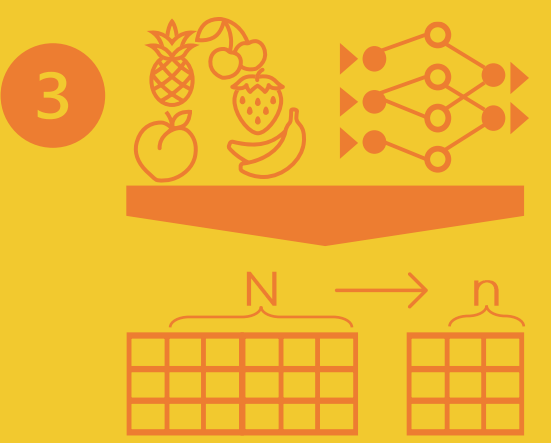
Analyse du besoin

- Acquisition des données
- Visualisation des images
- Besoin réduction dimension
- Solution envisagée

2

Big Data | Rôle et création

- Schéma général
- Solution stockage AWS S3
- Instance AWS EC2 Linux
- Hadoop vs Spark
- Session PySpark



Chaine de Traitement

- Modèle VGG16
- Preprocessing
- Fonctions UDF Pandas
- Sauvegarde dans s3 / Check
- Remarques & préconisations

1 Analyse du besoin

Images cible :

https://www.kaggle.com/datasets/moltean/fruits
Mihai Oltean, Fruits 360 dataset: new research directions, Technical report, 2021

Acquisition des données :

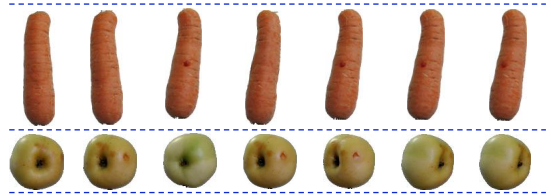
12 455 fichiers .jpg
558 Mo total

3 répertoires :

- Test / 3 110
- Training / 6 231
- Validation / 3 114

fruits-360-original-size / Test :

24



173

217 x 649 x 3
422 499 features

Solution envisagée

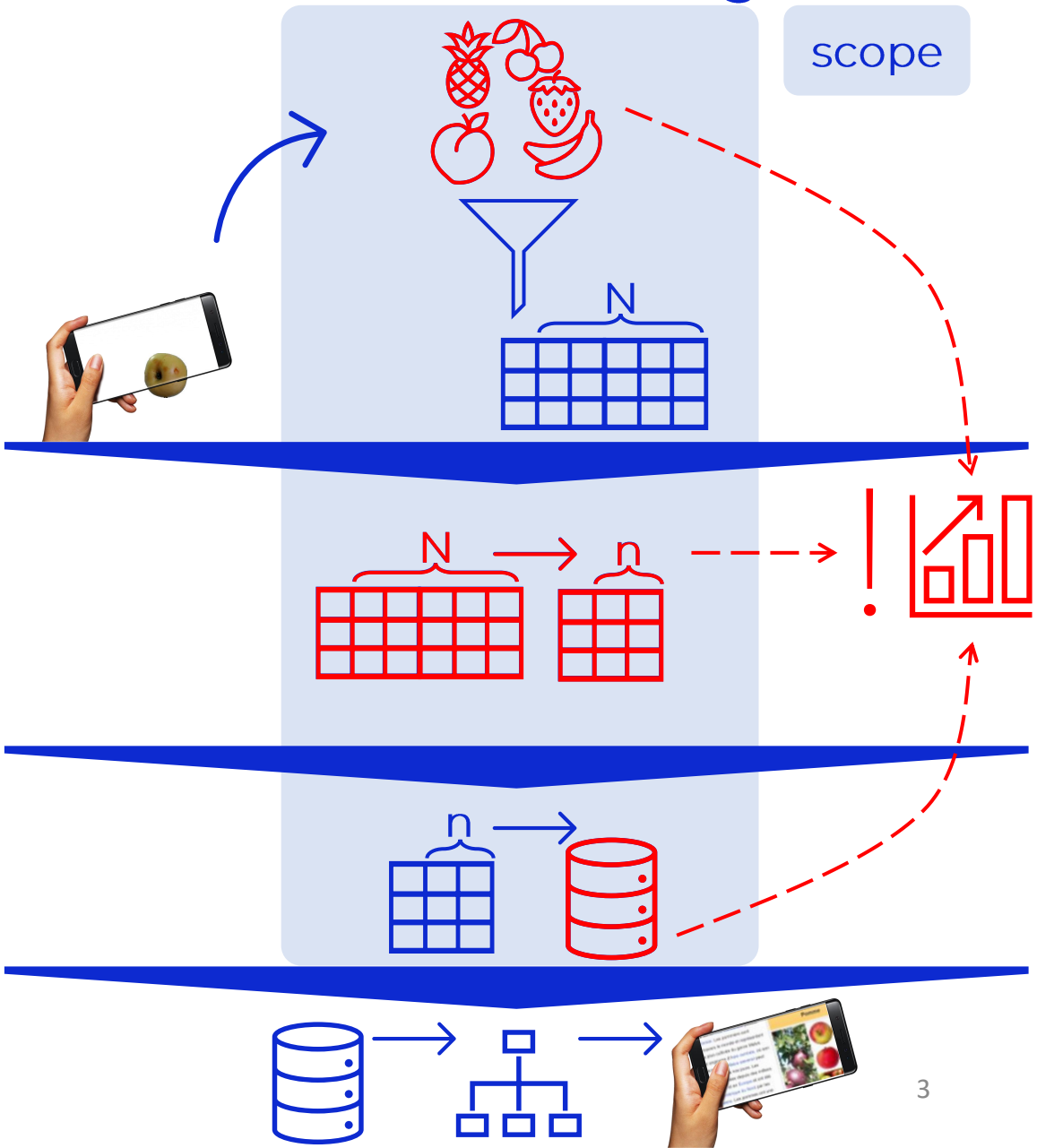
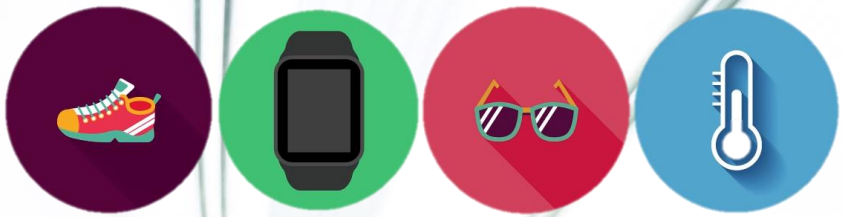


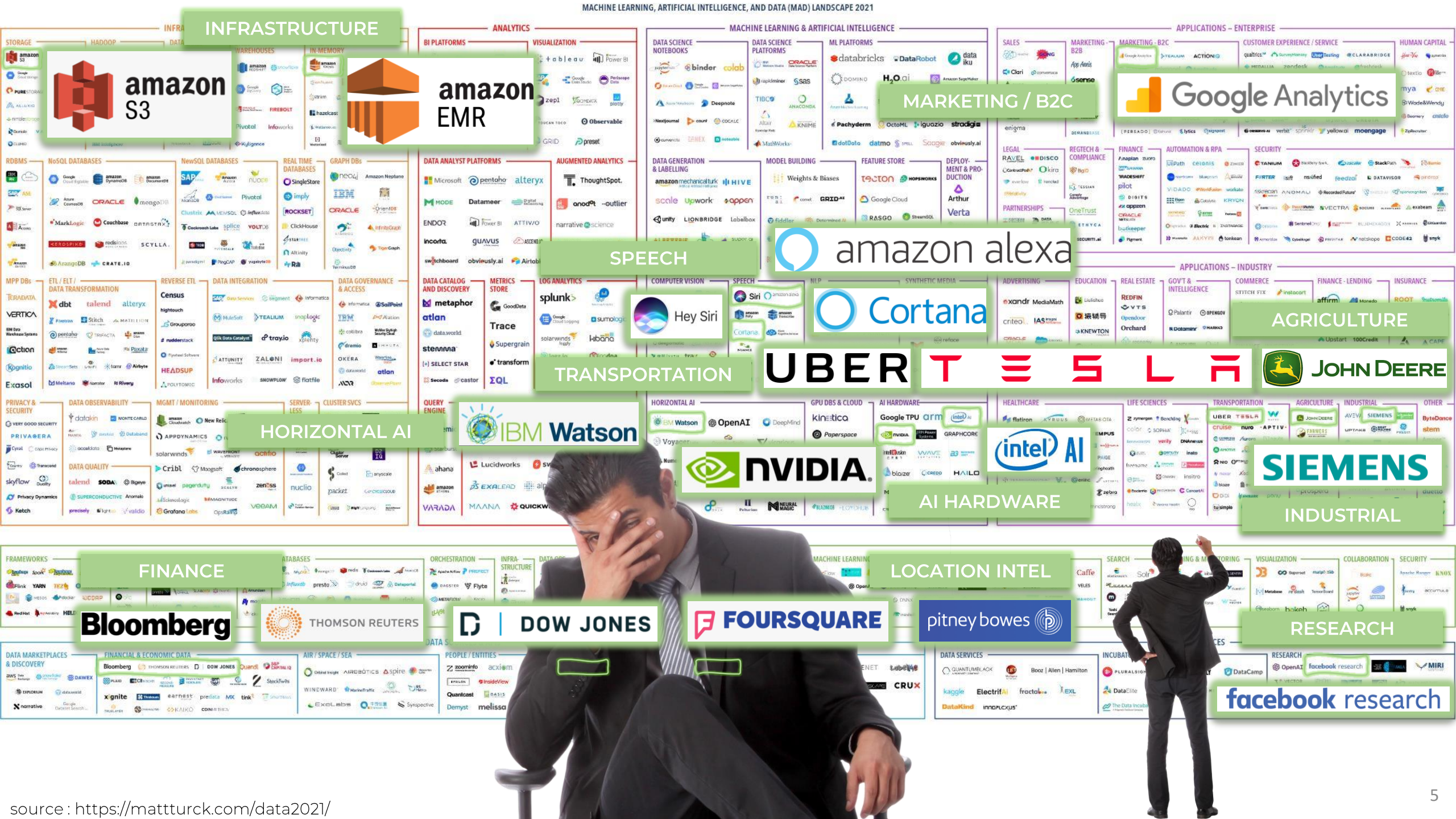
Schéma général

Volume



Vitesse

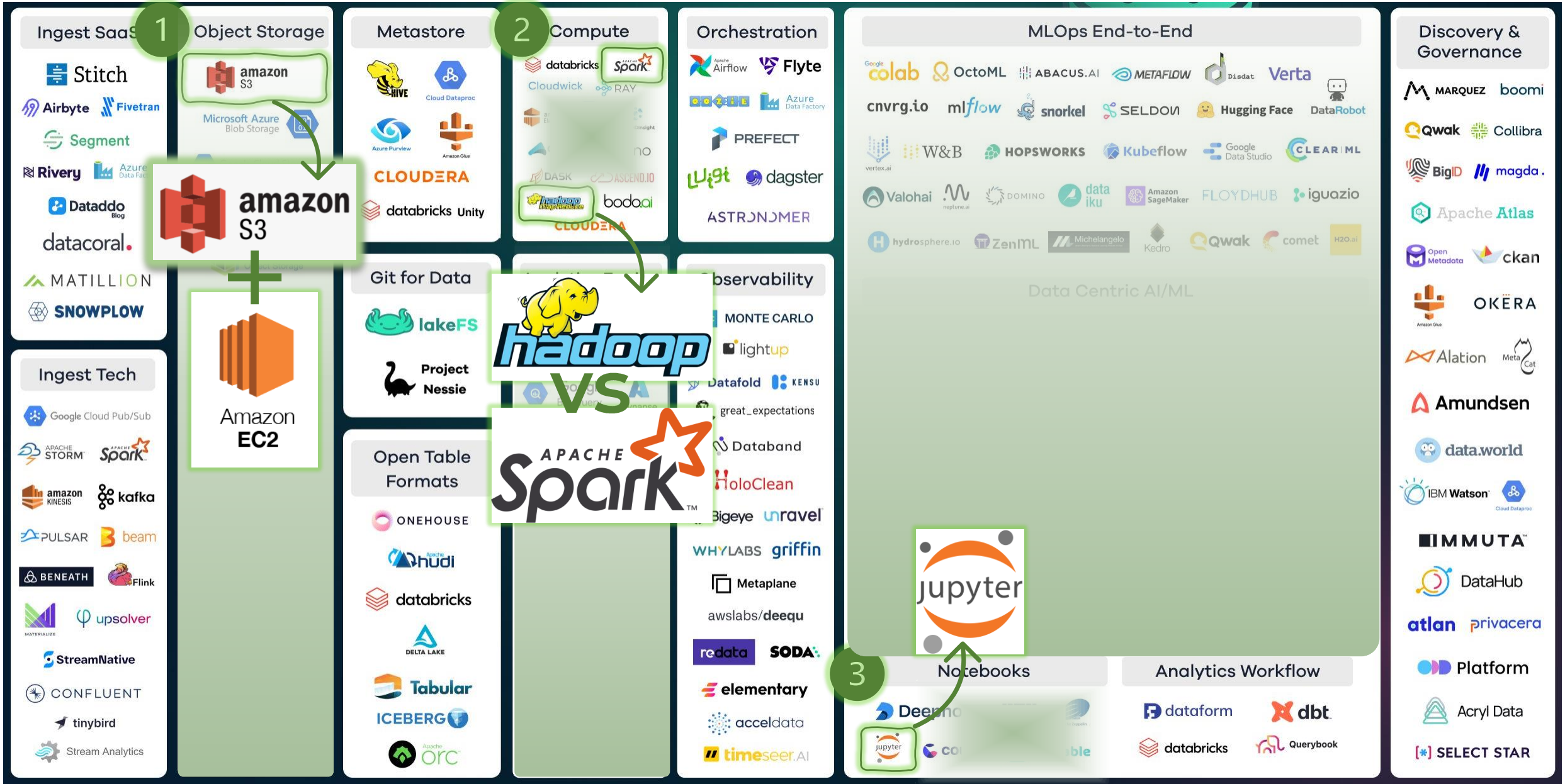
Variété



source : https://mattturck.com/data2021/

2 Big Data | Rôle et création

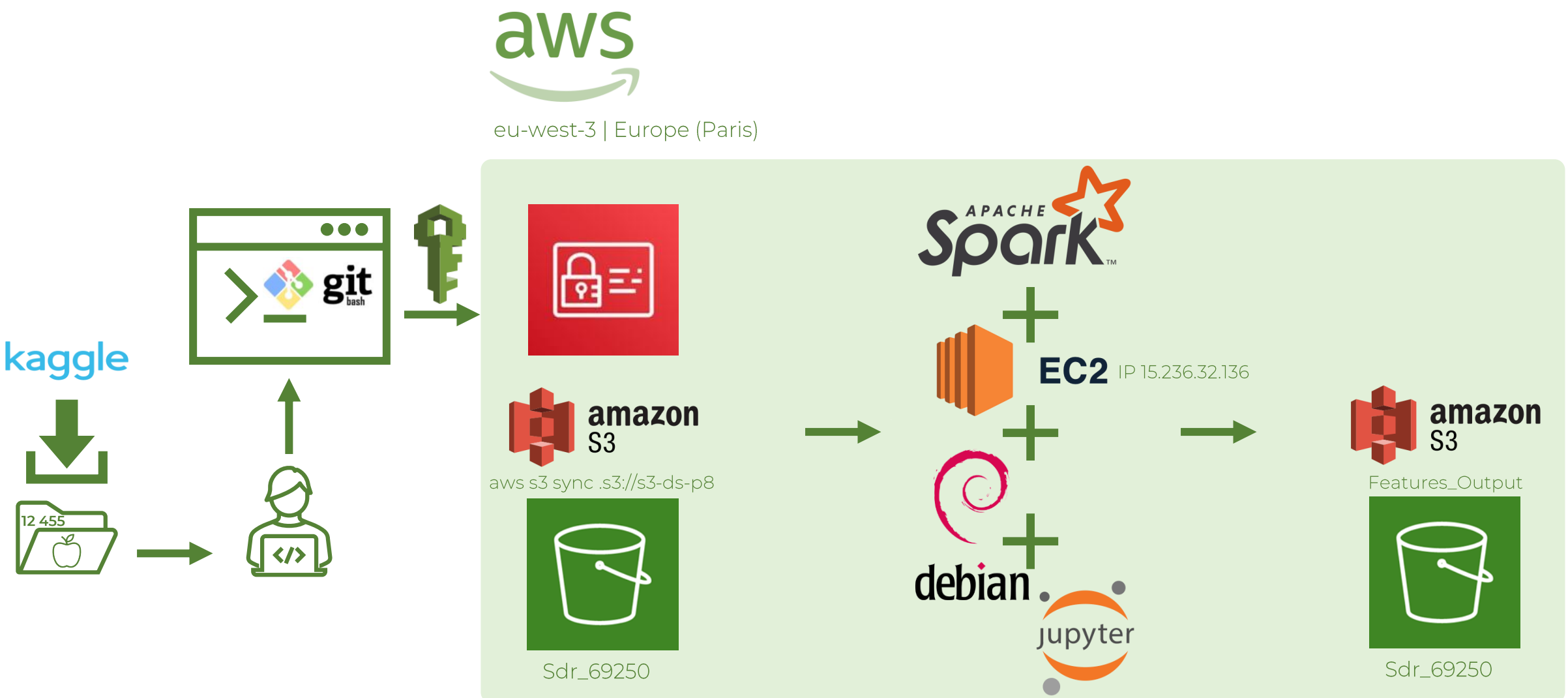
Schéma général



source : https://lakefs.io/blog/the-state-of-data-engineering-2022/

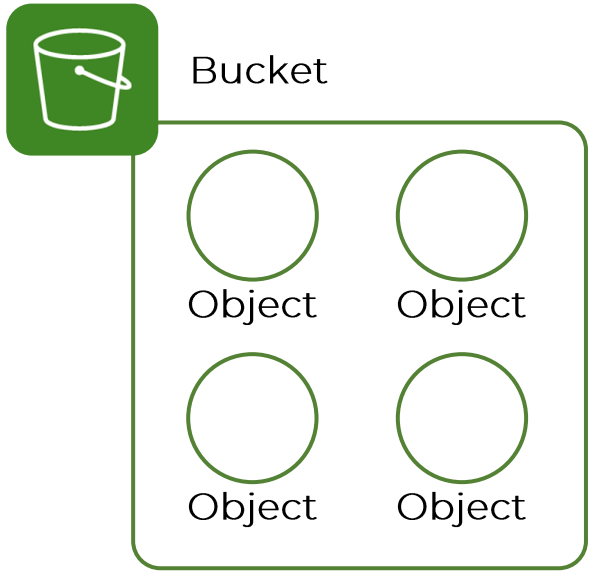
2 Big Data | Rôle et création

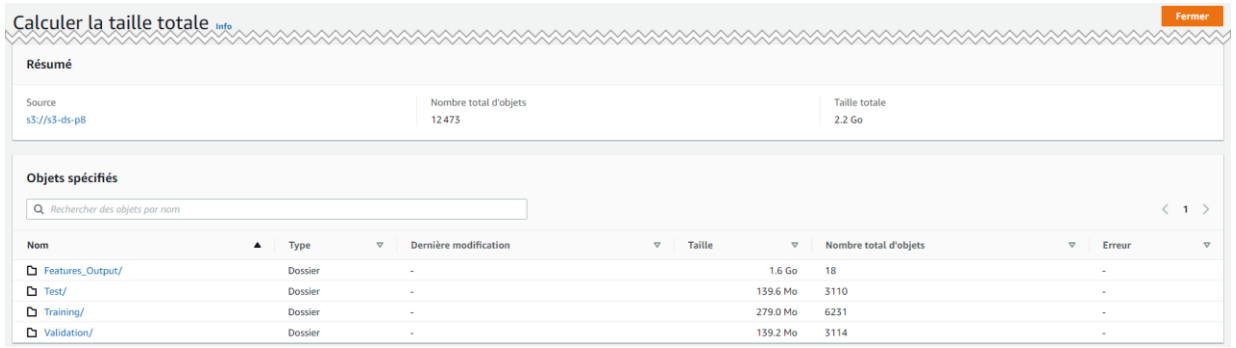
Schéma général



2 Big Data | Rôle et création

Solution stockage AWS S3





Classes de stockage

	S3 Standard	Hiérarchisation intelligente S3*	S3 Standard - IA	S3 One Zone-IA†	S3 Glacier Instant Retrieval	S3 Glacier Flexible Retrieval	S3 Glacier Deep Archive
Conçu pour la durabilité	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)
Conçu pour la disponibilité	99,99 %	99,9 %	99,9 %	99,5 %	99,9 %	99,99 %	99,99 %
Disponibilité SLA	99,9 %	99 %	99 %	99 %	99 %	99,9 %	99,9 %
Zones de disponibilité	≥3	≥3	≥3	1	≥3	≥3	≥3
Frais de capacité minimale par objet	N/A	N/A	128 Ko	128 Ko	128 Ko	40 Ko	40 Ko
Frais minimum de durée de stockage	N/A	N/A	30 jours	30 jours	90 jours	90 jours	180 jours
Frais d'extraction	N/A	N/A	par Go extrait	par Go extrait	par Go extrait	par Go extrait	par Go extrait
Latence du premier octet	millisecondes	millisecondes	millisecondes	millisecondes	millisecondes	minutes ou heures	heures
Type de stockage	Objet	Objet	Objet	Objet	Objet	Objet	Objet
Transitions du cycle de vie	Oui	Oui	Oui	Oui	Oui	Oui	Oui

Fréquence consultation

Coût

Accès sécurisé à s3 :

```python
# identifiants nécessaires pour que PySpark accède à AWS
AWS_CREDENTIALS = json.loads(os.popen("aws sts get-session-token").read())['Credentials']
```

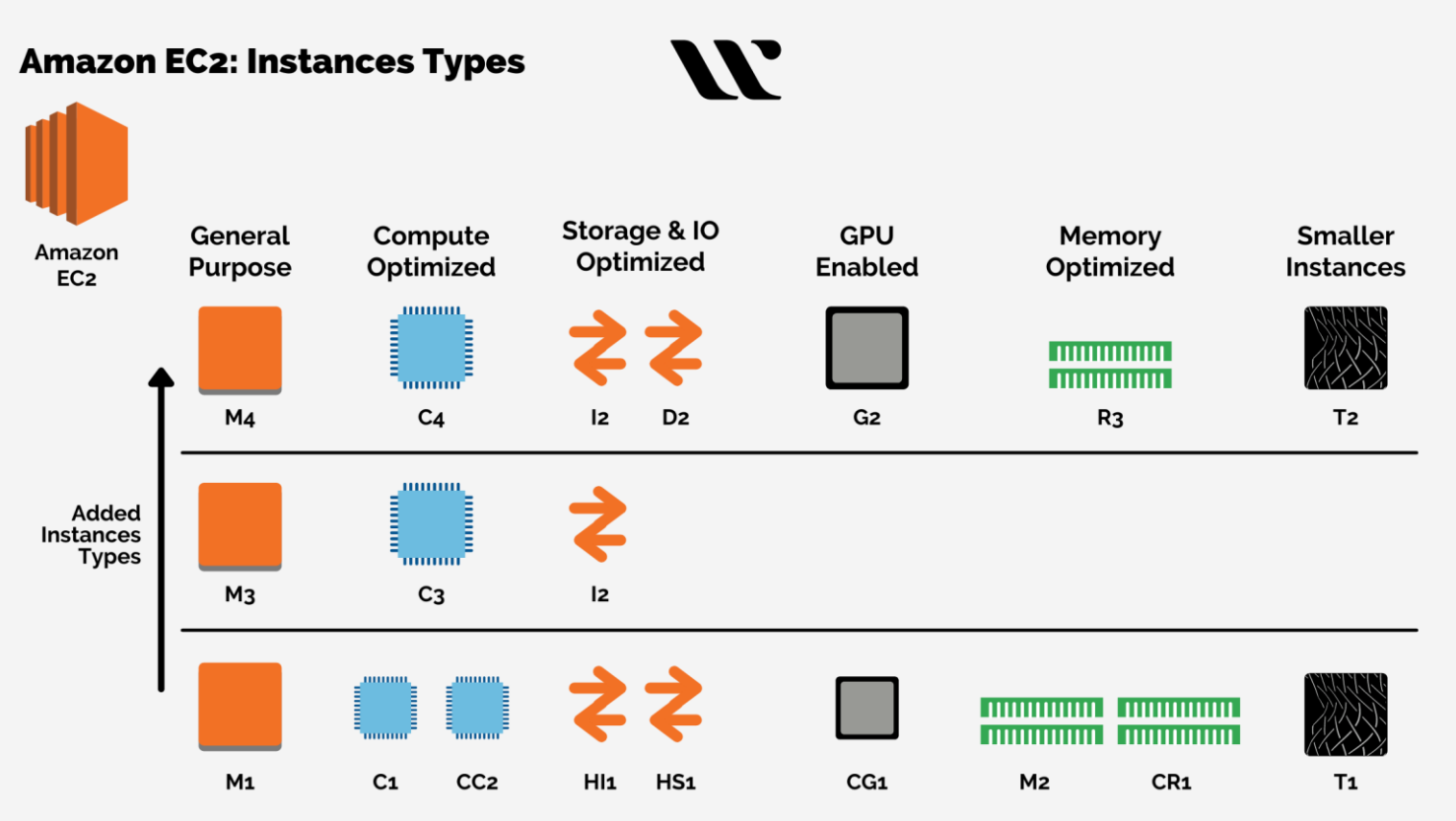
Python

```
# Necessary ressources SparkConf to run PySPark on AWS
# PySPark was coded in Scala itself coded in Java
# To run on AWS, we need to install AWS packages
# For the access we ask for AWS Credential
conf.set('spark.jars.packages', 'org.apache.hadoop:hadoop-aws:3.2.2')
conf.set('spark.hadoop.fs.s3a.aws.credentials.provider', 'org.apache.hadoop.fs.s3a.
TemporaryAWSCredentialsProvider')

# Credentials passed to the SparkConf to get  access to S3 Bucket
conf.set('spark.hadoop.fs.s3a.access.key', AWS_CREDENTIALS['AccessKeyId'])
conf.set('spark.hadoop.fs.s3a.secret.key', AWS_CREDENTIALS['SecretAccessKey'])
conf.set('spark.hadoop.fs.s3a.session.token', AWS_CREDENTIALS['SessionToken'])
```

2 Big Data | Rôle et création

Instance AWS EC2 Linux



Offres tarifaires

- Offre gratuite
- Instances Spot
- Saving Plans
- Instances Réservées
- A la demande
- Hôtes dédiés

Facturation à la seconde

Red Hat Enterprise Linux

Microsoft

ubuntu

debian

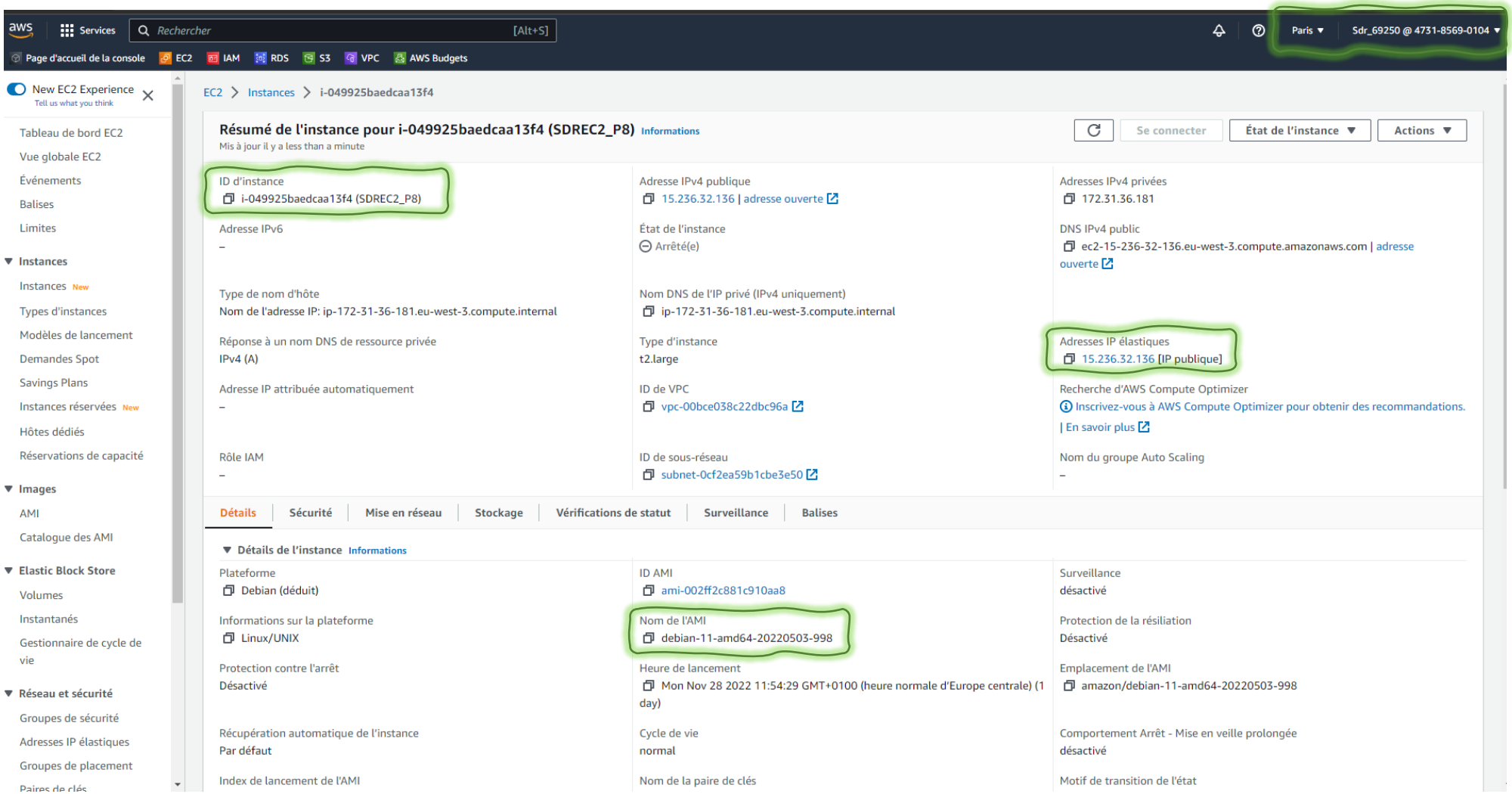
SUSE

aws Amazon Linux

source : https://www.whizlabs.com/blog/amazon-elastic-compute-cloud-guide/

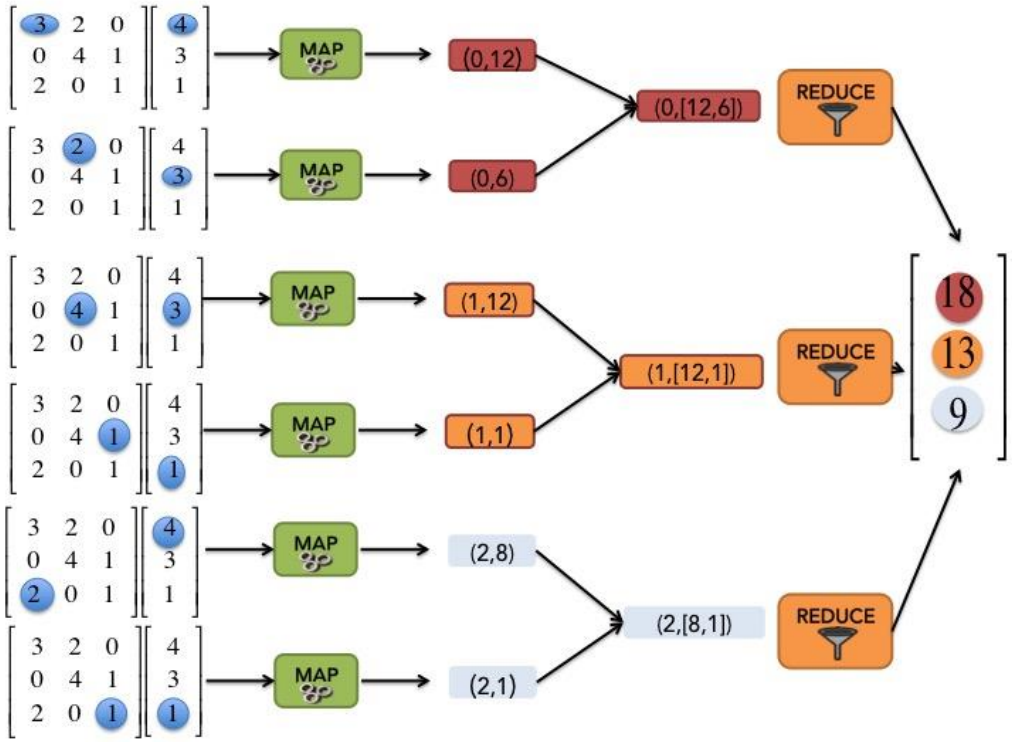
2 Big Data | Rôle et création

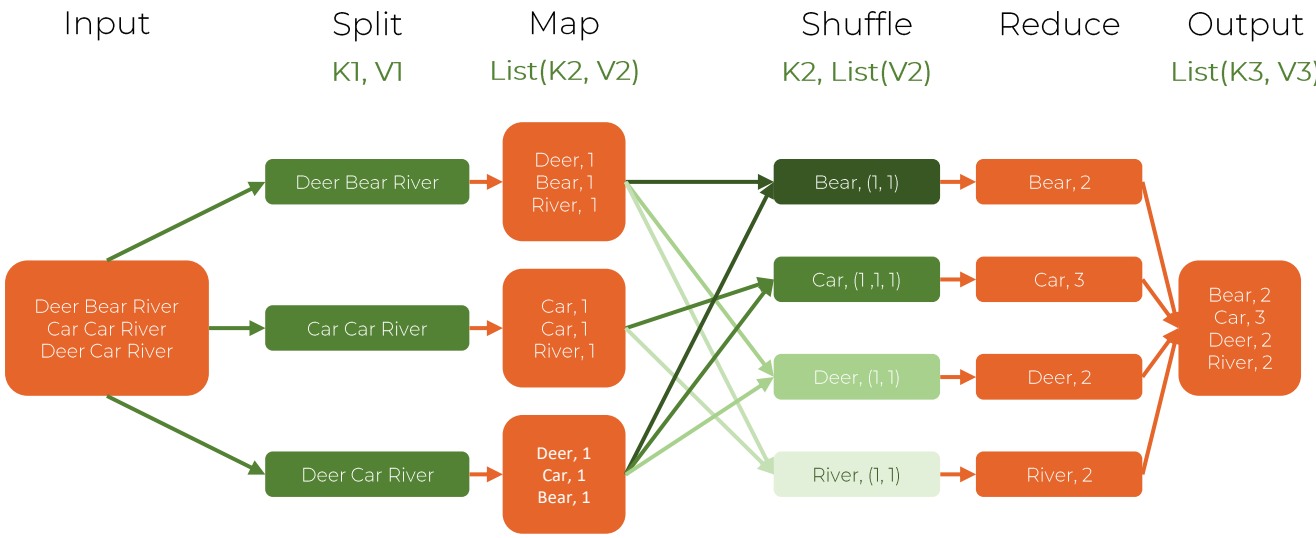
Instance AWS EC2 Linux



Hadoop vs Spark

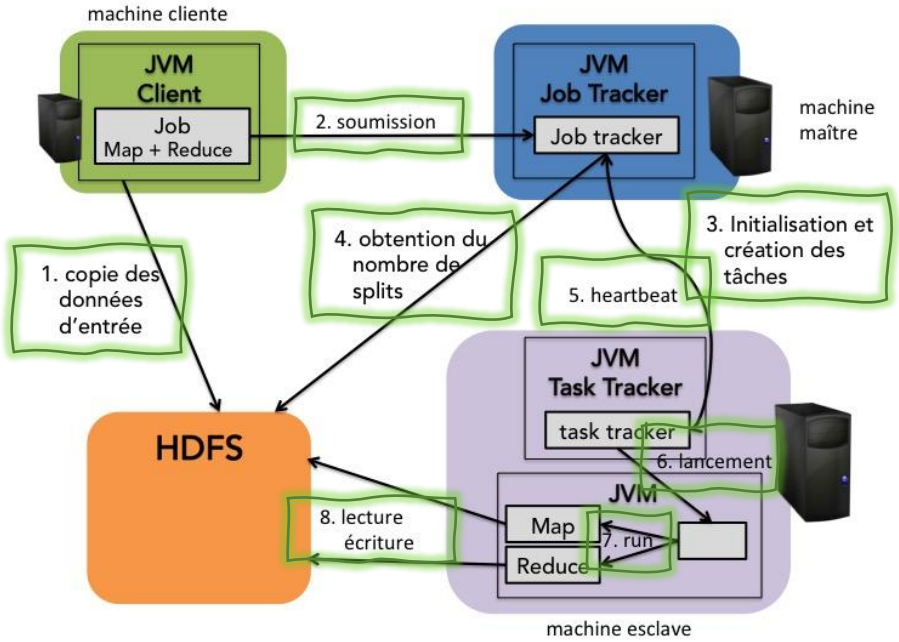
Map Reduce





Hadoop vs Spark

Map Reduce Hadoop 1.x



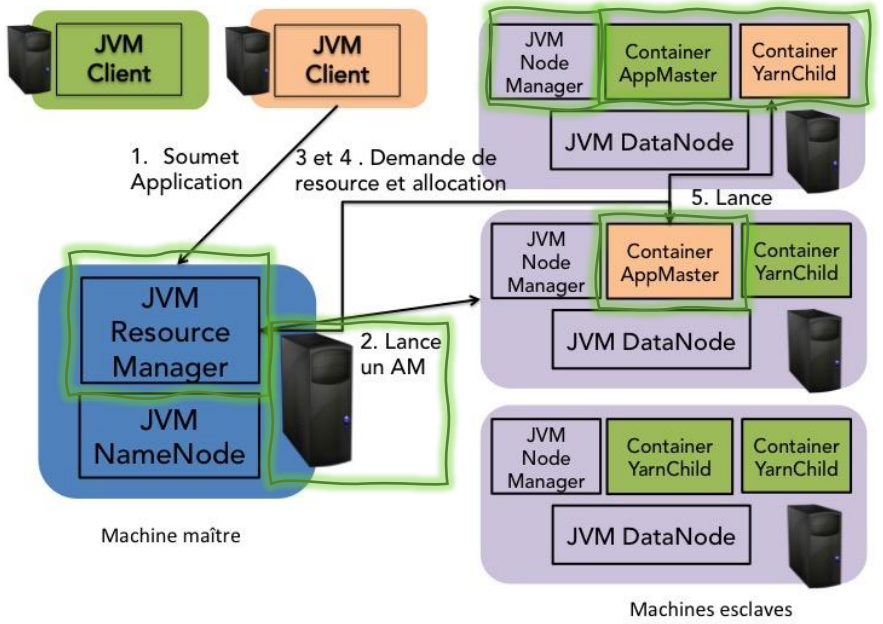
Map Reduce + HDFS

Transformer algorithme

Problèmes complexes nécessitent d'enchainer Map/Reduce

Job Tracker:
- ressources cluster
- ordonnancer jobs

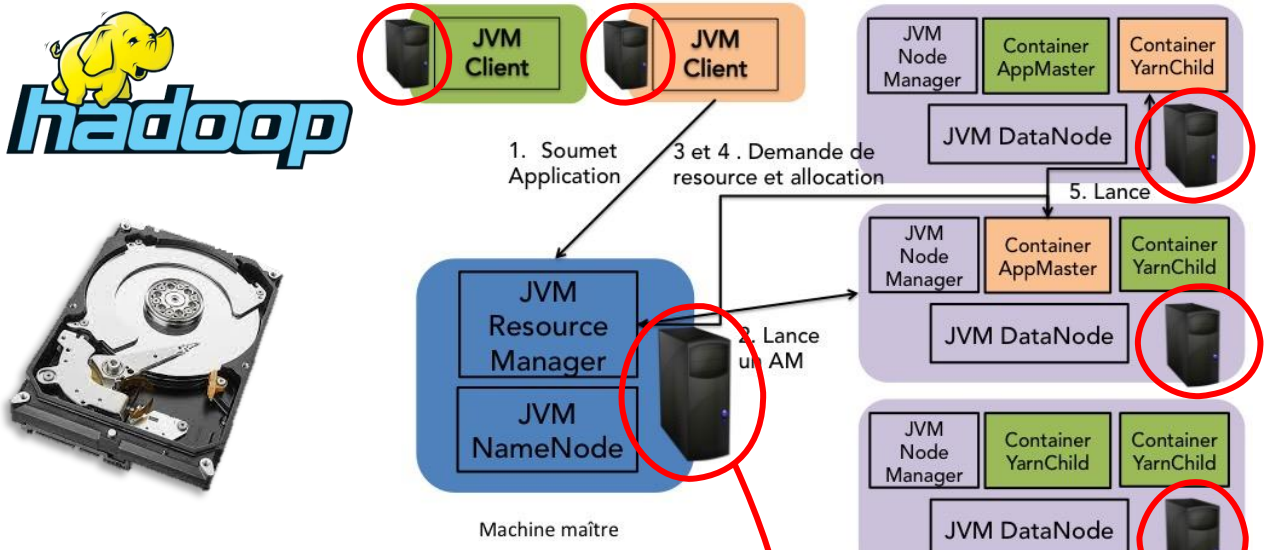
Map Reduce Hadoop 2.x



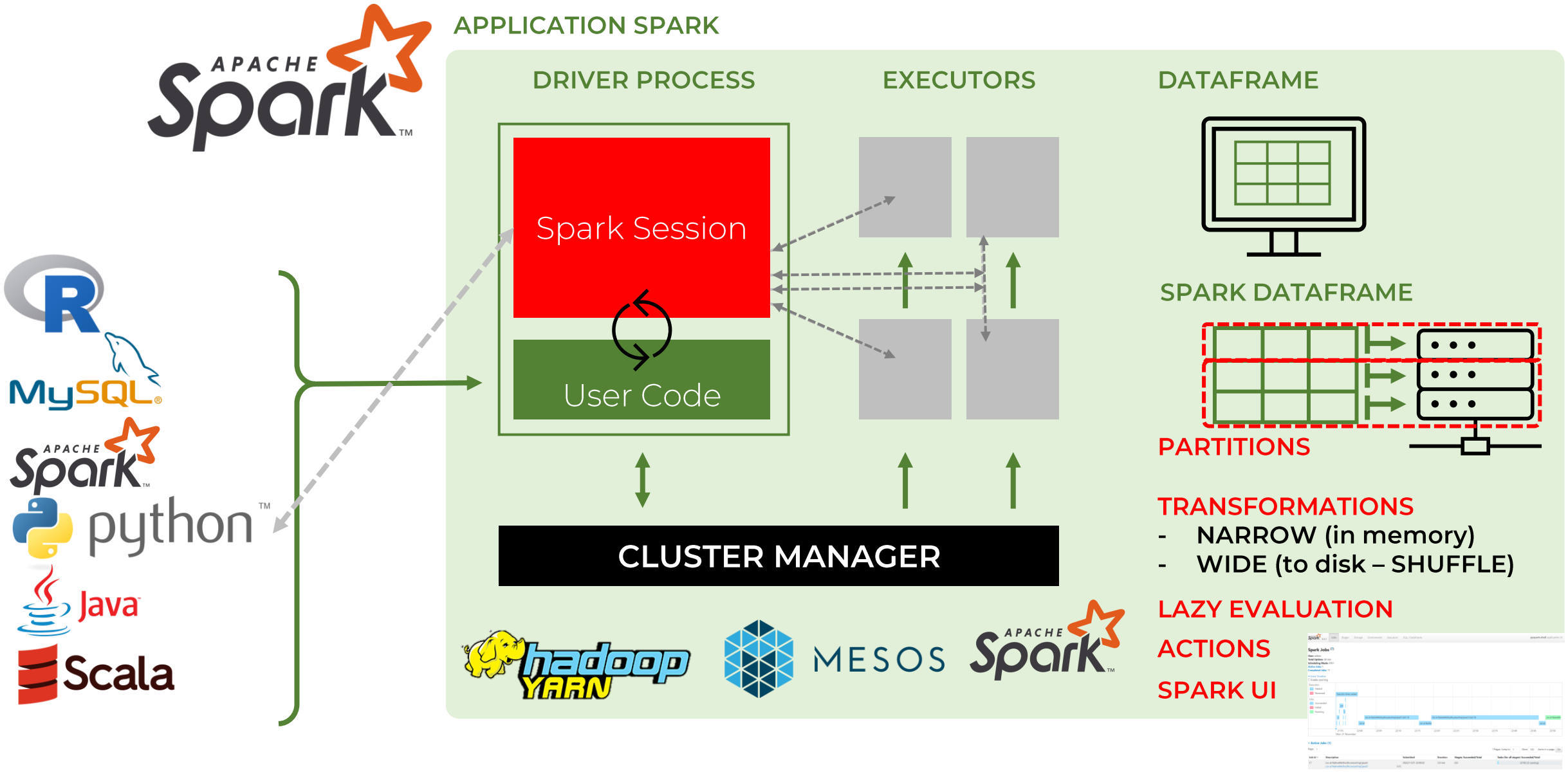
Map Reduce + HDFS + YARN

2 Big Data | Rôle et création

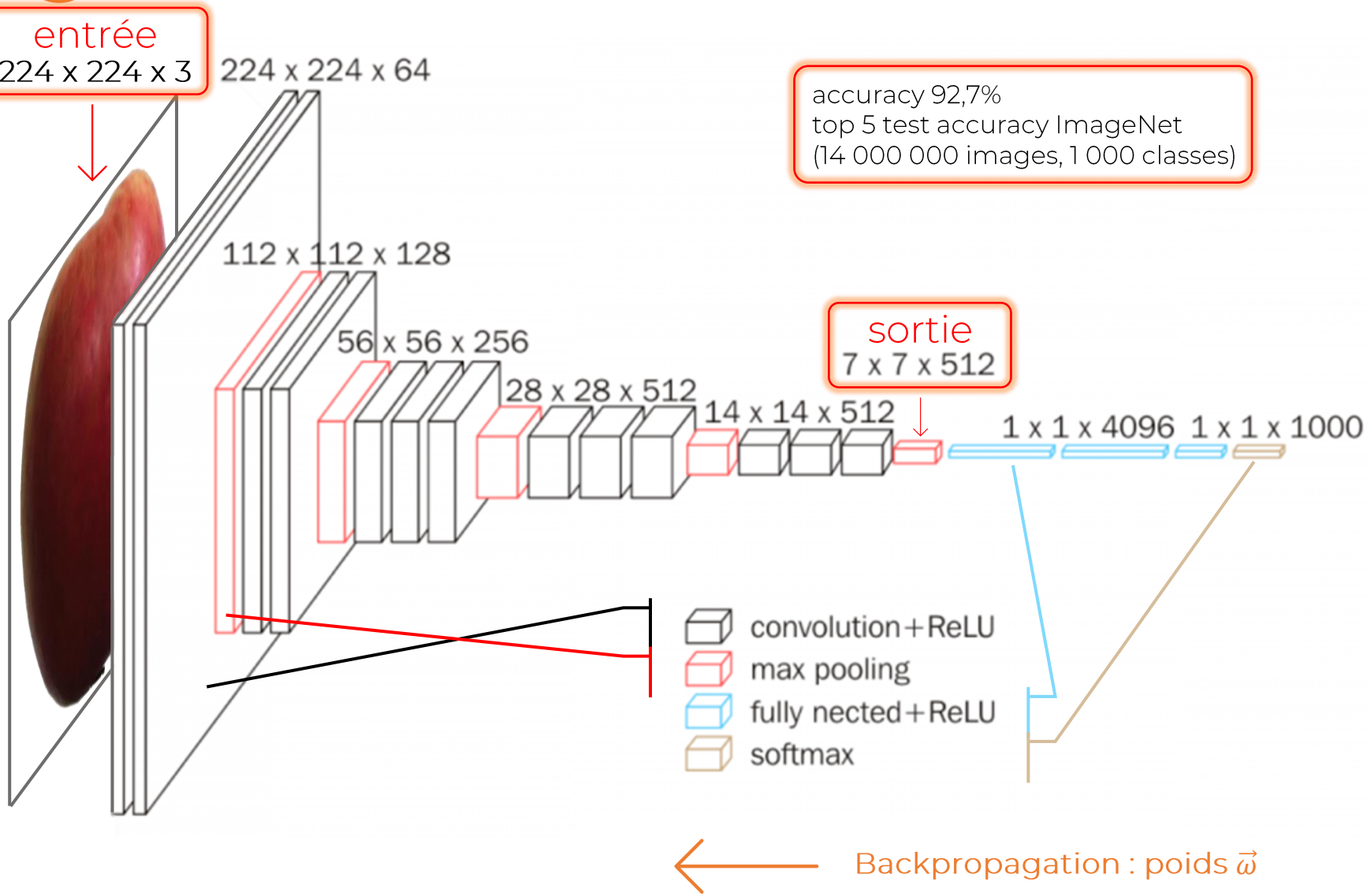
Hadoop vs Spark



Technologie	Latence (s)	Taux de transfert (Go/s)
Disque dur	10^{-2}	0.15


APACHE Spark™
APPLICATION SPARK
DRIVER PROCESS
EXECUTORS
DATAFRAME
Spark Session
User Code
SPARK DATAFRAME
PARTITIONS
TRANSFORMATIONS
- NARROW (in memory)
- WIDE (to disk – SHUFFLE)
LAZY EVALUATION
ACTIONS
SPARK UI
CLUSTER MANAGER
R
MySQL
APACHE Spark™
python™
Java
Scala
hadoop YARN
MESOS
APACHE Spark™

3 Chaine de Traitement



accuracy 92,7%
top 5 test accuracy ImageNet
(14 000 000 images, 1 000 classes)

Backpropagation : poids $\vec{\omega}$

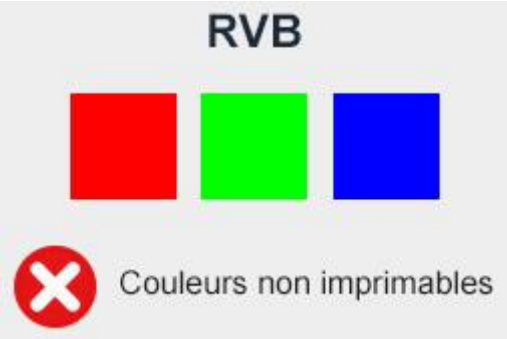
Modèle VGG16

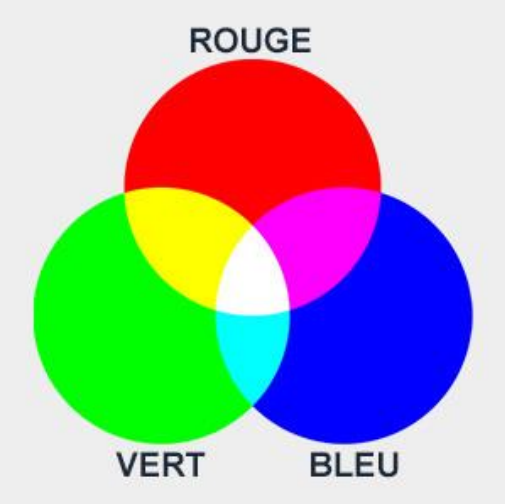
```
# Change input shape dimensions for fine-tuning with Keras
# remove last layer (include_top=False) to get a Tensor
model = VGG16(weights="imagenet", include_top=False)
model.trainable = False
# model = Model(inputs = model.input, outputs = model.output)
model.summary()  # verify that the top layer is removed
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, None, None, 3)]	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

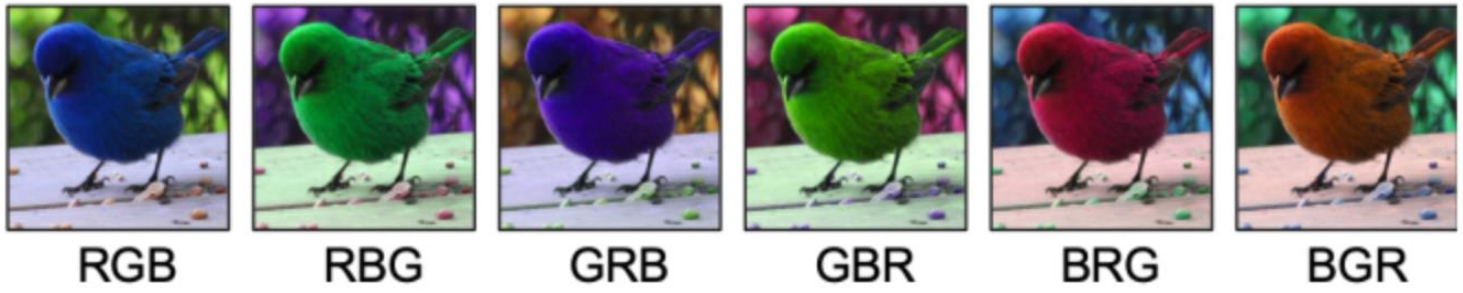
```
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

3 Chaine de Traitement





3! = 6



Preprocessing

- ✓ resize([224, 224])
- ✓ vgg16.preprocess _input

YYY x ZZZ x 3 | RGB

224 x 224 x 3 | BGR

sources : https://www.easyflyer.fr/lp/explications-rvb-rgb/, https://datascience.stackexchange.com/questions/65260/how-important-is-the-channel-order-in-deep-learning-computer-vision-tasks

3 Chaine de Traitement

Fonctions UDF Pandas

```python
@pandas_udf('array<float>', PandasUDFType.SCALAR_ITER)
def featurize_udf(content_series_iter):
```

```python
@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.
    for content_series in content_series_iter:
        yield featurize_series(model, content_series)
```

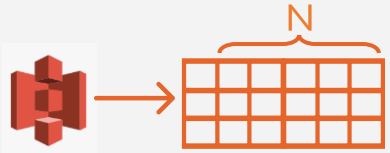
Python

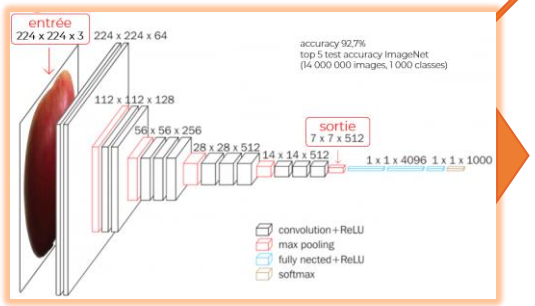
```python
# We can now run featurization on our entire Spark DataFrame.
# NOTE: This can take a long time (about 10 minutes)
# since it applies a large model to the full dataset.
# number of workers in parallel = 5
features_test = images_test.repartition(5).select(col("path"), featurize_udf("content").alias
("features"))
```

Python


```python
# creation of dataframe for Test files
# filter on file extension, recursive file lookup, bucket path
images_test = spark.read.format("binaryFile") \
  .option("pathGlobFilter", "*.jpg") \
  .option("recursiveFileLookup", "true") \
  .load("s3a://s3-ds-p8/Test/*")
```

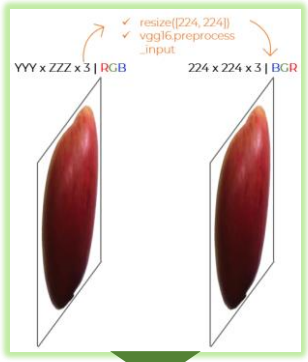
Python






```python
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))
    feats = 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 feats]
    return pd.Series(output)
```

Python




```python
# kept the resize command, as preprocess_input() did not take care of it
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)
```

Python

3 Chaine de Traitement

Sauvegarde dans s3 / Check

```
features_test.withColumn("features", col("features").cast("string"))\
.write.mode('overwrite')\
.option("header",True)\
.csv("s3a://s3-ds-p8/Features_Output/Test/")
```

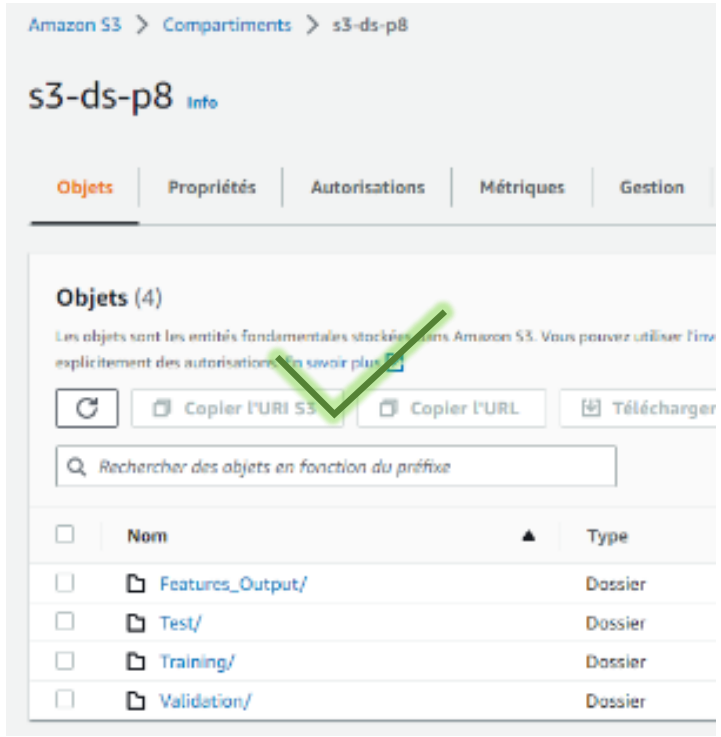
```
features_training.withColumn("features", col("features").cast("string"))\
.write.mode('overwrite')\
.option("header",True)\
.csv("s3a://s3-ds-p8/Features_Output/Training/")
```

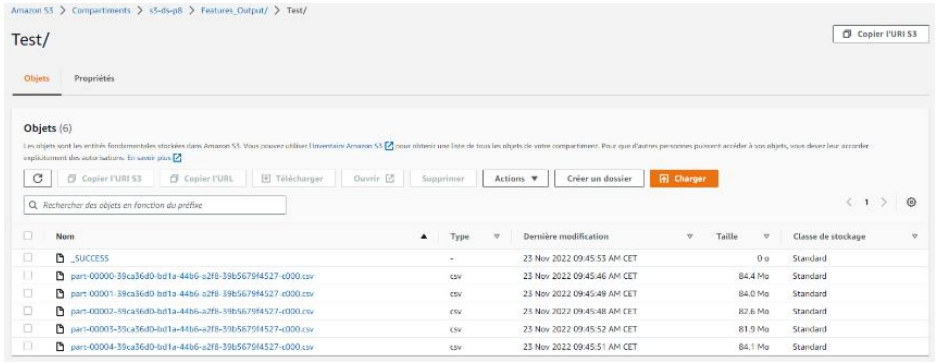
```
features_validation.withColumn("features", col("features").cast("string"))\
.write.mode('overwrite')\
.option("header",True)\
.csv("s3a://s3-ds-p8/Features_Output/Validation/")
```

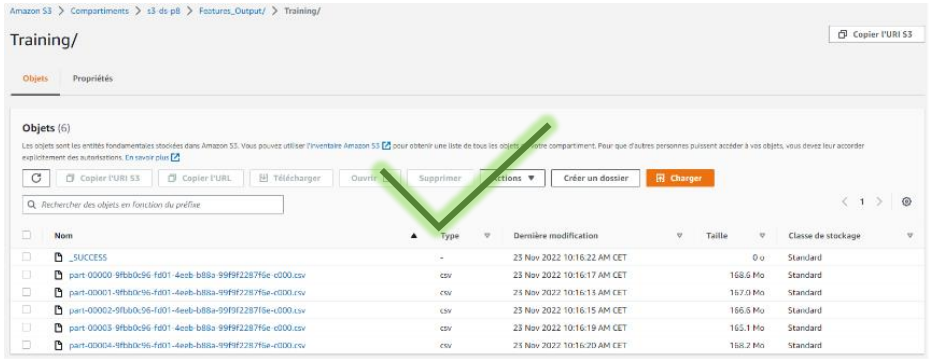
Test	15min 24s
Training	30min 28s
Validation	23min 30s

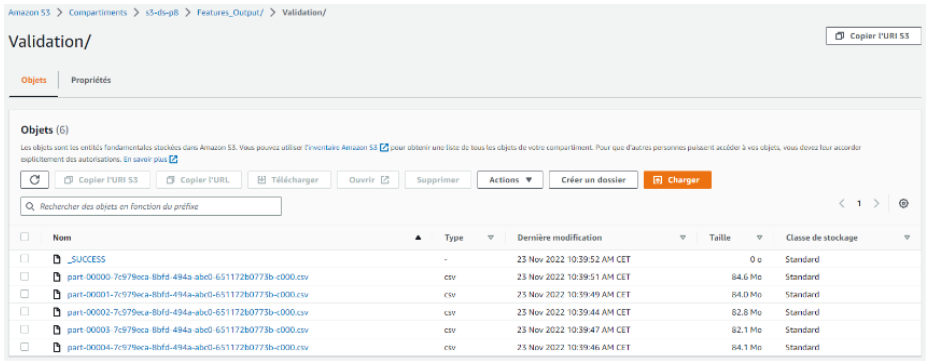
3 Chaine de Traitement

Sauvegarde dans s3 / Check





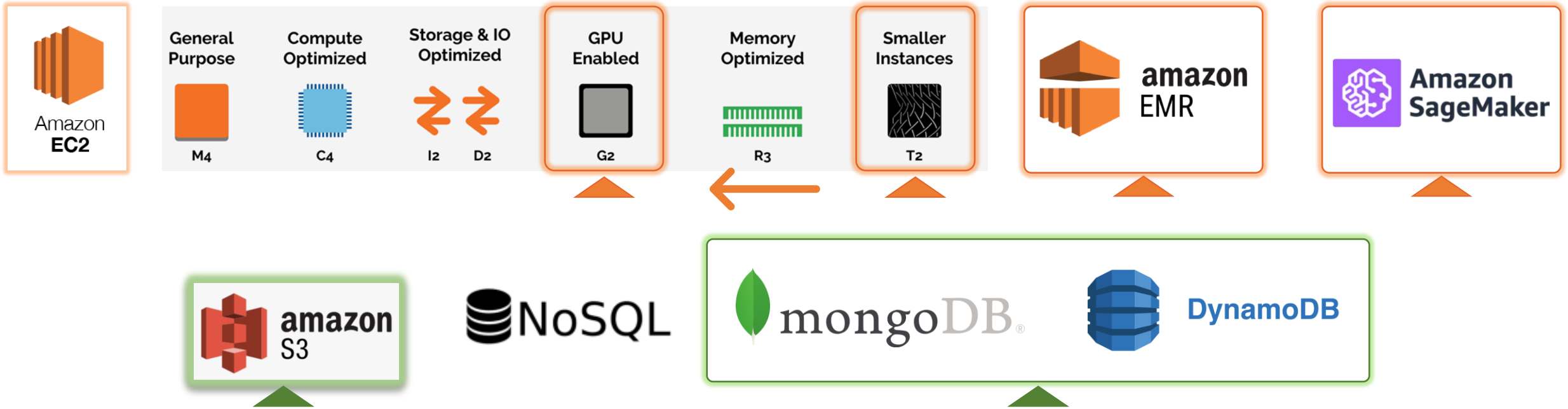


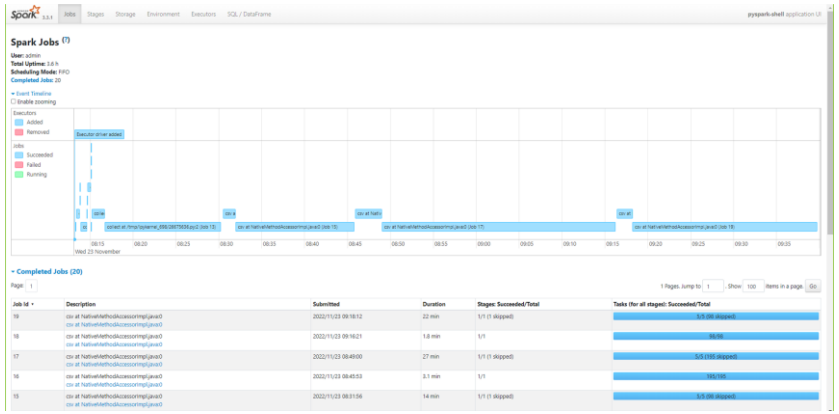


	Nb_Features
count	2518.0
mean	25088.0
std	0.0
min	25088.0
25%	25088.0
50%	25088.0
75%	25088.0
max	25088.0

3 Chaine de Traitement

Remarques & préconisations





VS

Questions ?

Merci !