**Démarche de réalisation du projet**

**1. Préambule**

**1.1 Problématique**

La très jeune start-up de l'AgriTech, nommée "**Fruits**!", cherche à proposer des solutions innovantes pour la récolte des fruits.

La volonté de l’entreprise est de préserver la biodiversité des fruits en développant des robots cueilleurs intelligents qui permettraient des traitements spécifiques pour chaque espèce de fruits.

La start-up souhaite dans un premier temps se faire connaître en mettant à disposition du grand public une application mobile qui permettrait aux utilisateurs de prendre en photo un fruit et d'obtenir des informations sur ce fruit.

Pour la start-up, cette application permettrait de mettre en place une première version du moteur de classification des images de fruits.

De plus, le développement de l’application mobile permettra de construire une première version de l'architecture **Big Data** nécessaire.

**1.2 Objectifs dans ce projet**

1. Développer une première chaîne de traitement des données qui comprendra le **preprocessing** et une étape de **réduction de dimension**.
2. Tenir compte du fait que le volume de données va augmenter très rapidement après la livraison de ce projet, ce qui implique de :
   * Déployer le traitement des données dans un environnement **Big Data**
   * Développer les scripts en **Pyspark** pour effectuer du **calcul distribué**

**1.3 Déroulement des étapes du projet**

Le projet va être réalisé en 2 temps, dans deux environnements différents.

Nous allons dans un premier temps développer sur un nombre limité d'images à traiter pour limiter les temps de calcul en local.

Une fois les choix techniques validés, nous déploierons notre solution dans un environnement Big Data en mode distribué.

Le projet se divise donc en 2 parties:

1. Liste des choix techniques généraux retenus
2. Déploiement de la solution dans le cloud

# 2. Choix techniques généraux retenus

## 2.1 Calcul distribué

L’énoncé du projet nous impose de développer des scripts en **Pyspark** afin de prendre en compte l’augmentation très rapide du volume de donné après la livraison du projet.

**PySpark** est un moyen de communiquer avec **Spark** via le langage **Python**. **Spark**, quant à lui, est un outil qui permet de gérer et de coordonner l'exécution de tâches sur des données à travers un groupe d'ordinateurs. Spark (ou Apache Spark) est un framework open source de calcul distribué in-memory pour le traitement et l'analyse de données massives.

J’ai tout d’abord testé localement le script ce qui permet de **simuler le calcul distribué** afin de valider que notre solution fonctionne. J’ai donc ensuite créé une version cloud en **réalisant les opérations sur un cluster de machine**.

Dans les deux environnements (Local et Cloud) nous utiliserons donc **Spark** et nous l’exploiterons à travers des scripts python grâce à **PySpark**.

## 2.2 Transfert Learning

L'énoncé du projet nous demande également de réaliser une première chaîne de traitement des données qui comprendra le preprocessing et une étape de réduction de dimension.

Il est également précisé qu'il n'est pas nécessaire d'entraîner un modèle pour le moment.

Je décide de partir sur une solution de **transfert learning**.

Simplement, le **transfert learning** consiste à utiliser la connaissance déjà acquise par un modèle entraîné (ici **MobileNetV2**) pour l'adapter à notre problématique.

Nous allons fournir au modèle nos images, et nous allons récupérer l'avant dernière couche du modèle. En effet la dernière couche de modèle est une couche qui permet la classification des images ce que nous ne souhaitons pas dans ce projet.

L'avant dernière couche correspond à un **vecteur réduit** de dimension (1,1,1280).

Cela permettra de réaliser une première version du moteur pour la classification des images des fruits.

**MobileNetV2** a été retenu pour sa rapidité d'exécution, particulièrement adaptée pour le traitement d'un gros volume de données ainsi que la faible dimensionnalité du vecteur de caractéristique en sortie (1,1,1280).

# 3. Déploiement de la solution sur le cloud

Après avoir développé en local sur un nombre réduit d'images et vérifier que cela fonctionne, il est temps de déployer la solution à plus grande échelle sur un vrai cluster de machines.

Je travaille sous Linux avec une version Ubuntu doncles commandes décrites ci-dessous sont donc réaliséesexclusivement dans cet environnement.

Plusieurs contraintes se posent :

1. Quel prestataire de Cloud choisir ?
2. Quelles solutions de ce prestataire adopter ?
3. Où stocker nos données ?
4. Comment configurer nos outils dans ce nouvel environnement ?

## 3.1 Choix du prestataire cloud : AWS

Le prestataire le plus connu et qui offre à ce jour l'offre la plus large dans le cloud computing est **Amazon Web Services** (AWS). Certaines de leurs offres sont parfaitement adaptées à notre problématique et c'est la raison pour laquelle j'utiliserai leurs services.

L'objectif premier est de pouvoir, grâce à AWS, louer de la puissance de calcul à la demande.  
L'idée étant de pouvoir, quel que soit la charge de travail, obtenir suffisamment de puissance de calcul pour pouvoir traiter nos images, même si le volume de données venait à fortement augmenter.

De plus, la capacité d'utiliser cette puissance de calcul à la demande permet de diminuer drastiquement les coûts si l'on compare les coûts d'une location de serveur complet sur une durée fixe (1 mois, 1 année par exemple).

## 3.2 Choix de la solution technique : EMR

La solution **PAAS** (Plateforme As A Service) avec le service **EMR** d'AWS est la plus adaptée à notre problématique car il permet une mise en œuvre qui soit à la fois plus rapide et plus efficace que la solution IAAS.

En effet, ce service EMR permet permet de louer des **instances EC2** avec des applications préinstallées. **Spark** y sera déjà installé et on a la possibilité de demander l'installation de **Tensorflow** ainsi que **JupyterHub. Il y a également la** Possibilité d'indiquer des **packages complémentaires** à installer à l'initialisation du serveur sur **l'ensemble des machines du cluster**.

## 3.3 Choix de la solution de stockage des données : Amazon S3

Amazon propose une solution très efficace pour la gestion du stockage des données : **Amazon S3**. S3 pour Amazon Simple Storage Service.

Nous ne stockerons pas nos données sur l'espace alloué par le serveur **EC2**, mais si nous ne prenons aucune mesure pour les sauvegarder ensuite sur un autre support, les données seront perdues lorsque le serveur sera résilié. On va donc utiliser Amazon S3 pour sauvegarder les données avant la résiliation du serveur et pour éviter d’être exposé à certaines problématiques si nos données venaient à saturer l'espace disponible de nos serveurs (ralentissements, disfonctionnements).

L'espace disque disponible est **illimité sur Amazon S3**, et il est **indépendant de nos serveurs EC2**. L'accès aux données est **très rapide** car nous restons dans l'environnement d'AWS et nous prenons soin de choisir la même région pour nos serveurs **EC2** et **S3**.

De plus, comme nous le verrons il est possible d'accéder aux données sur **S3**  
de la même manière que l'on **accède aux données sur un disque local**.  
Nous utiliserons simplement un **PATH au format s3://...** .

## 3.4 Configuration de l'environnement de travail

## La première étape est d'installer et de configurer [**AWS Cli**](https://aws.amazon.com/fr/cli/), il s'agit de l'**interface en ligne de commande d'AWS**. Elle nous permet d'**interagir avec les différents services d'AWS**, comme **S3** par exemple.

Pour pouvoir utiliser **AWS Cli**, il faut le configurer en créant préalablement  
un utilisateur à qui on donnera les autorisations dont nous aurons besoin.  
Dans ce projet il faut que l'utilisateur ait à minima un contrôle total sur le service S3.

La gestion des utilisateurs et de leurs droits s'effectue via le service **AMI** d'AWS.

## Une image contenant capture d’écran, texte, logiciel, Page web Description générée automatiquement

Une fois l'utilisateur créé et ses autorisations configurées nous créons une **paire de clés**  
qui nous permettra de nous **connecter sans à avoir à devoir saisir systématiquement notre login/mot de passe**.

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Il faut également configurer l'**accès SSH** à nos futurs serveurs EC2.  
Ici aussi, via un système de clés qui nous dispense de devoir nous authentifier "à la main" à chaque connexion.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

## 3.5 Upload de nos données sur S3

La première étape consiste à **créer un bucket sur S3** dans lequel nous uploaderons les données du projet. On charge le contenu du dossier "**Test**" dans un répertoire "**Test**" sur notre bucket "**p8-data-juliesbo**".

Une image contenant texte, logiciel, Page web, Icône d’ordinateur

Description générée automatiquement

## 3.6 Configuration du serveur EMR

Pour créer un cluster, on se rend sur la page Amazon EMR (<https://eu-west-3.console.aws.amazon.com/emr/home?region=eu-west-3#/clusters>) et on clique sur le gros bouton bleu "Créer un cluster".

### **3.6.1 Étape 1 : Logiciels et étapes**

La première chose à faire est de donner un nom au cluster : **OC-P8-Fruits.** Il faut ensuite sélectionner les packages dont nous aurons besoin et nous laissons la version d'**EMR** par défaut, qui est la version la plus récente.

Nous cochons bien évidement **Spark**, **TensorFlow** pour importer notre modèle et réaliser le **transfert learning** et **JupyterHub** pour pouvoir travailler avec un **notebook Jupyter** via l'application.

#### Une image contenant texte, capture d’écran, logiciel, Page web Description générée automatiquement

### **3.6.2 Étape 2 : Matériel**

On va lancer un cluster Spark doté de trois instances m3.xlarge, c'est à dire un driver et deux workers.

Une image contenant texte, capture d’écran, nombre, Parallèle

Description générée automatiquementUne image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

### **3.6.3 Étape 3 : Paramètres de cluster généraux : Actions d'amorçage**

Nous allons à cette étape **choisir les packages manquants à installer** et qui nous serons utiles dans l'exécution de notre notebook. L'avantage de réaliser cette étape maintenant est que les packages installés le seront sur l'ensemble des machines du cluster.

Nous créons donc un fichier nommé "**bootstrap-emr.sh**" que nous uploadons sur S3 (je l’installe à la racine de mon **bucket "p8-data-juliesbo"**).

Une image contenant capture d’écran, texte, logiciel

Description générée automatiquement

Le fichier "**bootstrap-emr.sh**" ne contient que des commandes pip qui permettent d'installer les packages utilisés dans notre script.

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Pour cela dans la partie actions d'amorçage on chosit action personnalisée, puit on clique sur le bouton **configurer et ajouter**, on uploade dans le champ emplacement du script le fichier "**bootstrap-emr.sh**" et enfin sur **ajouter**.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

#### **3.6.4 Étape 4 : Modifier les paramètres du logiciel**

Pour paramétrer la persistance des notebooks créés et ouverts via JupyterHub afin d'enregistrer et ouvrir les notebooks directement sur **Amazon S3**. On doit entrer la configuration au format JSON au moment de la création du cluster.

### Une image contenant texte, capture d’écran, affichage, logiciel Description générée automatiquement

### **3.6.5 Étape 5 : Sécurité**

A cette étape nous sélectionnons la **paire de clés EC2** créé précédemment. Elle nous permettra de nous connecter en **ssh** à nos **instances EC2** sans avoir à entrer nos login/mot de passe. On laisse les autres paramètres par défaut.

Une image contenant texte, Police, nombre, capture d’écran

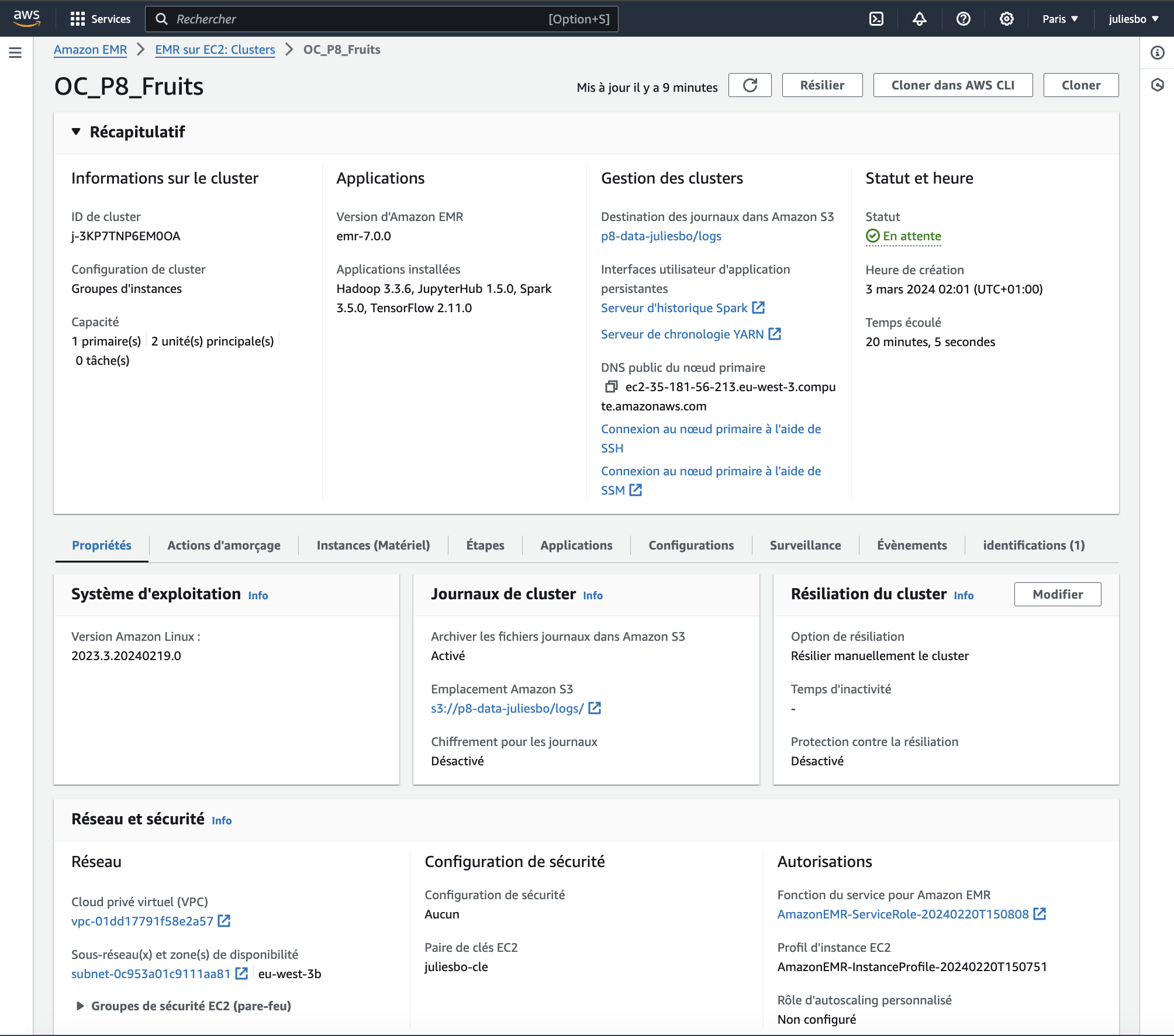
Description générée automatiquement

Et enfin, on clique sur "***Créer un cluster***"

## 3.7 Instanciation du serveur

Il ne nous reste plus qu'à attendre que le serveur soit prêt. Cette étape peut prendre entre **15 et 20 minutes**.

Lorsque le statut affiche en vert : "**En attente**" cela signifie que l'instanciation s'est bien déroulée et que notre serveur est prêt à être utilisé.



## 3.8 Création du tunnel SSH à l'instance EC2 (Maître)

Nous souhaitons maintenant pouvoir accéder à nos applications :

* **JupyterHub** pour l'exécution de notre notebook
* **Serveur d'historique Spark** pour le suivi de l'exécution des tâches de notre script lorsqu'il sera lancé

Cependant, ces applications ne sont accessibles que depuis le réseau local du driver,  
et pour y accéder nous devons **créer un tunnel SSH vers le driver**.

Par défaut, ce driver se situe derrière un firewall qui bloque l'accès en SSH.  
Pour ouvrir le port 22 qui correspond au port sur lequel écoute le serveur SSH, il faut modifier le **groupe de sécurité EC2 du driver**.

On se connecte ensuite en SSH au driver de notre cluster. Par défaut, ce driver se situe derrière un firewall qui bloque l'accès en SSH.Pour ouvrir le port 22 qui correspond au port sur lequel écoute le serveur SSH,il faut modifier le groupe de sécurité EC2 du driver. Sur la page de la consoleconsacrée à EC2, dans l'onglet "Réseau et sécurité", on clique sur "Groupes de sécurité".On a modifié aussi le groupe de sécurité d’ElasticMapReduce-Master.Dans l'onglet "Entrant", on ajoute une règle SSH dont la source est "N'importe où".

Une image contenant texte, logiciel, Page web, Site web

Description générée automatiquement

Une dernière étape est nécessaire pour accéder à nos applications, en demandant à notre navigateur d'emprunter le tunnel ssh. J'utilise pour cela **FoxyProxy.**

On active donc le **tunnel ssh avec la commande suivante :** ssh -i ~/.ssh/juliesbo-cle.pem -D 5555 [hadoop@ec2-35-181-56-213.eu-west-3.compute.amazonaws.com](mailto:hadoop@ec2-35-181-56-213.eu-west-3.compute.amazonaws.com). J'ai choisi d'utiliser le port **5555** au lieu du **8157.**

On demande ensuite à notre navigateur d'emprunter le tunnel ssh avec **FoxyProxy. On** a doncinstallé donc l'extension FoxyProxy dans notre navigateur et on a ajouté un proxy à l'adresse localhost:5555 en cochant l'optionSOCKS proxy.

On peut maintenant s'apercevoir que plusieurs applications nous sont accessibles.

Une image contenant texte, nombre, Police, logiciel

Description générée automatiquement

## 3.9 Connexion au notebook JupyterHub

Pour se connecter à **JupyterHub** en vue d'exécuter notre **notebook**, il faut commencer par cliquer sur l'application **JupyterHub** apparu depuis que nous avons configuré le **tunnel ssh** et **foxyproxy** sur notre navigateur.

On passe les éventuels avertissements de sécurité puis nous arrivons sur une page de connexion.

On se connecte avec les informations par défaut : login: **jovyan et** password: **jupyter**

Nous arrivons ensuite dans un dossier vierge de notebook. Grace à la **persistance** paramétrée à l'instanciation du cluster nous sommes actuellement dans l'arborescence de notre **bucket S3.** J’ai décidé d'**importer un notebook déjà rédigé en local directement sur S3** et je l'ouvre depuis **l'interface JupyterHub**.

Une image contenant capture d’écran, texte, logiciel, Page web

Description générée automatiquement

## 3.10 Exécution du code

J’ai exécuté ensuite le notebook en Pyspark depuis **JupyterHub hébergé sur notre cluster EMR.** Avant de commencer, je me suis assurer d'utiliser le **kernel pyspark**.

**En utilisant ce kernel, une session spark est créé à l'exécution de la première cellule**.  
Il n'est donc **plus nécessaire d'exécuter le code "spark = (SparkSession ..."** comme lors  
de l'exécution de notre notebook en local sur notre VM Ubuntu.

## 3.11 Suivi de l'avancement des tâches avec le Serveur d'Historique Spark

Il est possible de voir l'avancement des tâches en cours avec le **serveur d'historique Spark**.

Une image contenant texte, ligne, Police, capture d’écran

Description générée automatiquement

Une image contenant texte, Police, nombre, reçu

Description générée automatiquement

Le **serveur d'historique Spark** nous permet une vision beaucoup plus précise de l'exécution des différentes tâche sur les différentes machines du cluster.

On peut également constater que notre cluster de calcul a mis un tout petit peu **moins de 19 minutes** pour traiter les **22 688 images**.

## 3.12 Résiliation de l'instance EMR

Notre travail est maintenant terminé. Pour éviter une facturation inutile et comme le cluster de machines EMR est **facturé à la demande**, il faut maintenant **résilier le cluster**.

Je réalise cette commande depuis l'interface AWS :

1. Commencez par **désactiver le tunnel ssh dans FoxyProxy**
2. Cliquez sur "**Résilier**"
3. Confirmez la résiliation
4. La résiliation prend environ **1 minute**
5. La résiliation est effectuée

## 3.12 Cloner le serveur EMR (si besoin)

Si nous devons de nouveau exécuter notre notebook dans les mêmes conditions, il nous suffit de **cloner notre cluster** et ainsi en obtenir une copie fonctionnelle sous 15/20 minutes, le temps de son instanciation.

Pour cela depuis l'interface AWS, cliquez sur "**Cloner**".

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

# 4. Les résultats sur S3

Les résultats du preprocessing sont disponibles sur S3 dans le dossier Results du bucket et ont été stockés au format parquet.

Une image contenant capture d’écran, texte, logiciel, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

**5. Conclusion**

Nous avons réalisé ce projet **en deux temps** en tenant compte des contraintes qui nous ont été imposées.

Nous avons **dans un premier temps développé notre solution en local** sur une machine virtuelle dans un environnement Linux Ubuntu.

La première phase a consisté à **installer l'environnement de travail Spark**. **Spark** a un paramètre qui nous permet de travailler en local et nous permet ainsi de **simuler du calcul partagé** en considérant **chaque cœur d'un processeur comme un worker indépendant**.  
Nous avons travaillé sur un plus **petit jeu de donnée**, l'idée était simplement de **valider le bon fonctionnement de la solution**.

Nous avons fait le choix de réaliser du **transfert learning** à partir du model **MobileNetV2**.  
Ce modèle a été retenu pour sa **légèreté** et sa **rapidité d'exécution** ainsi que pour la **faible dimension de son vecteur en sortie**.

Les résultats ont été enregistrés sur disque en plusieurs partitions au format "**parquet**".

**La solution a parfaitement fonctionné en mode local**.

La deuxième phase a consisté à créer un **réel cluster de calculs**. L'objectif était de pouvoir **anticiper une future augmentation de la charge de travail**.

Le meilleur choix retenu a été l'utilisation du prestataire de services **Amazon Web Services**  
qui nous permet de **louer à la demande de la puissance de calculs**, pour un **coût tout à fait acceptable**.  
Ce service se nomme **EC2** et se classe parmi les offres **Infrastructure As A Service** (IAAS).

Nous sommes allez plus loin en utilisant un service de plus haut niveau (**Plateforme As A Service** PAAS) en utilisant le service **EMR** qui nous permet d'un seul coup d'**instancier plusieurs serveur (un cluster)** sur lesquels nous avons pu demander l'installation et la configuration de plusieurs programmes et librairies nécessaires à notre projet comme **Spark**, **Hadoop**, **JupyterHub** ainsi que la librairie **TensorFlow**.

En plus d'être plus **rapide et efficace à mettre en place**, nous avons la **certitude du bon fonctionnement de la solution**, celle-ci ayant été préalablement validé par les ingénieurs d'Amazon.

Nous avons également pu installer, sans difficulté, **les packages nécessaires sur l'ensembles des machines du cluster**.

Enfin, avec très peu de modification, et plus simplement encore, nous avons pu **exécuter notre notebook comme nous l'avions fait localement**. Nous avons cette fois-ci exécuté le traitement sur **l'ensemble des images de notre dossier "Test"**.

Nous avons opté pour le service **Amazon S3** pour **stocker les données de notre projet**.  
S3 offre, pour un faible coût, toutes les conditions dont nous avons besoin pour stocker et exploiter de manière efficace nos données.  
L'espace alloué est potentiellement **illimité**, mais les coûts seront fonction de l'espace utilisé.

Il nous sera **facile de faire face à une monté de la charge de travail** en **redimensionnant** simplement notre cluster de machines (horizontalement et/ou verticalement au besoin), les coûts augmenteront en conséquence mais resteront nettement inférieurs aux coûts engendrés par l'achat de matériels ou par la location de serveurs dédiés.