Présentation Projet Fruits!

Infrastructure Big Data AWS

Projet 11 Openclassrooms : Réalisez un traitement dans un environnement Big Data sur le Cloud

📊 Structure de la Présentation

15-20 minutes

01

Page de Titre

30 secondes

02

Contexte - La Start-up Fruits!

2 minutes

03

Mes Objectifs

2 minutes

04

Architecture AWS

2 minutes

05

Pipeline de Traitement

2 minutes

06

Le Broadcast - Concept

2 minutes

07

Le Broadcast - Code

1-2 minutes

08

La PCA

2 minutes

09

Configuration EMR

1-2 minutes

Contexte - La Start-up Fruits!

Fruits! est une jeune start-up dans l'AgriTech qui souhaite révolutionner la récolte des fruits. Leur vision : créer des robots cueilleurs intelligents.

Pour commencer, ils veulent lancer une application mobile grand public permettant de photographier un fruit et d'obtenir des informations instantanées - espèce, maturité, etc.

L'enjeu clé : Cette application doit traiter potentiellement des millions d'images. Mon rôle n'est PAS d'entraîner le modèle de classification, mais de construire l'infrastructure Big Data capable de gérer cette charge.

Scalabilité critique

Millions d'images

Performance

Temps de réponse < 2 secondes

Coûts maîtrisés

Budget limité de start-up

Conformité RGPD

Données utilisateurs en Europe

Mes Objectifs

Ma mission s'articule autour de 3 objectifs principaux :

1

Compléter le code existant

Un alternant avait commencé le travail mais il manquait deux éléments critiques : le broadcast des poids du modèle TensorFlow pour optimiser les performances et la réduction de dimensionnalité par PCA pour économiser le stockage.

2

Déployer sur AWS EMR

Mettre en production l'infrastructure sur AWS : configuration d'un cluster EMR avec Spark, utilisation de S3 pour le stockage, exécution distribuée avec PySpark.

3

Respecter les contraintes

Budget strict : moins de 10€ pour tout le projet, conformité RGPD : toutes les données en Union Européenne, architecture scalable : capable de monter en charge.

Architecture AWS

Voici l'architecture que j'ai mise en place :

IAM

Identity and Access Management

- Gestion sécurisée des accès
- Rôles spécifiques pour EMR et S3
- Principe du moindre privilège

S3

Simple Storage Service

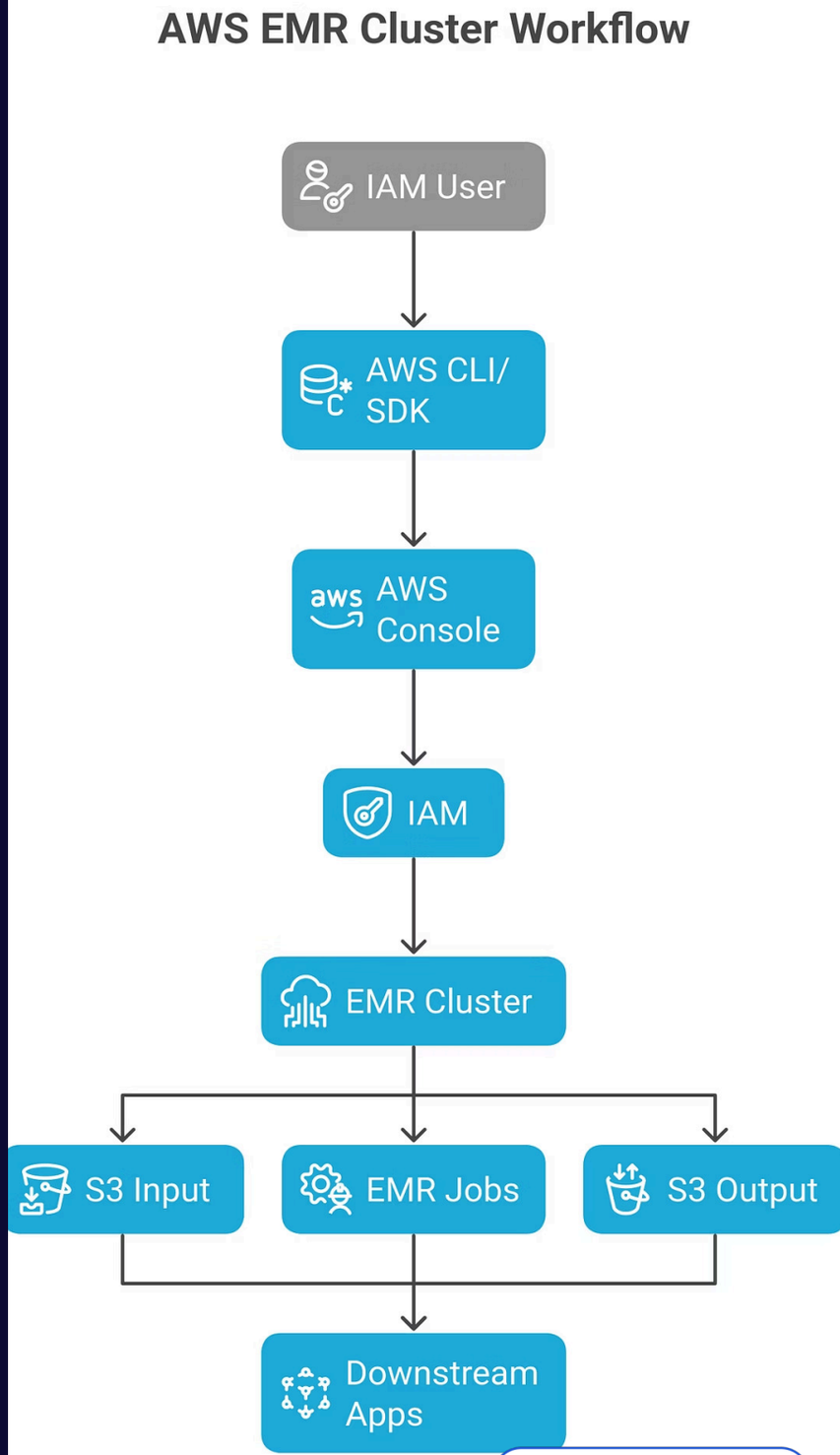
- Stockage des images brutes (dossier /Test)
- Stockage des résultats (dossier /Results)
- Stockage des résultats PCA (dossier /PCA_Results)

EMR

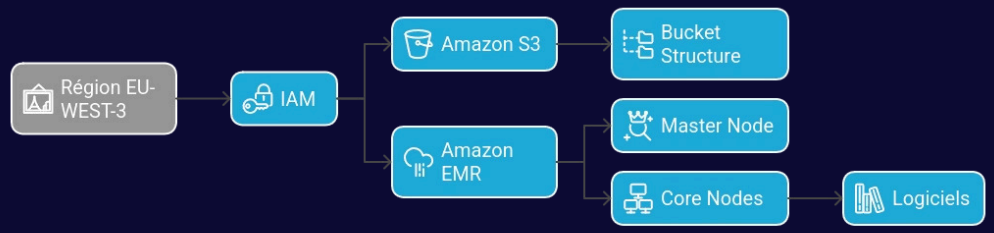
Elastic MapReduce

- Cluster de calcul avec Hadoop et Spark
- JupyterHub pour l'interface de développement
- 1 Master node + 2-4 Core nodes

Toutes ces ressources sont dans la région **eu-west-3 (Paris)** pour respecter le RGPD.



Architecture de l'Infrastructure Cloud



Détail de infrastructure AWS

	Amazon S3	Amazon EMR
Structure	📁 Bucket Structure: /Test/, /Results/, / PCA_Results/	🖥️ MASTER NODE: m5.xlarge, 4 vCPUs, 16 GB RAM; ⚙️ CORE NODES: m5.xlarge × 2-4, Spark Workers, HDFS Storage
Format	Parquet	Logiciels: Hadoop 3.3.6, Spark 3.5.0, JupyterHub, Tensorflow
Coût	~0.023$/GB/mois	~0.60€/heure

Configuration EMR

Voici la configuration de mon cluster EMR, optimisée pour le rapport performance/coût :

Master Node

Type : m5.xlarge (4 vCPUs, 16 GB RAM)

Rôle : coordination du cluster

Coût : ~0.19 €/heure

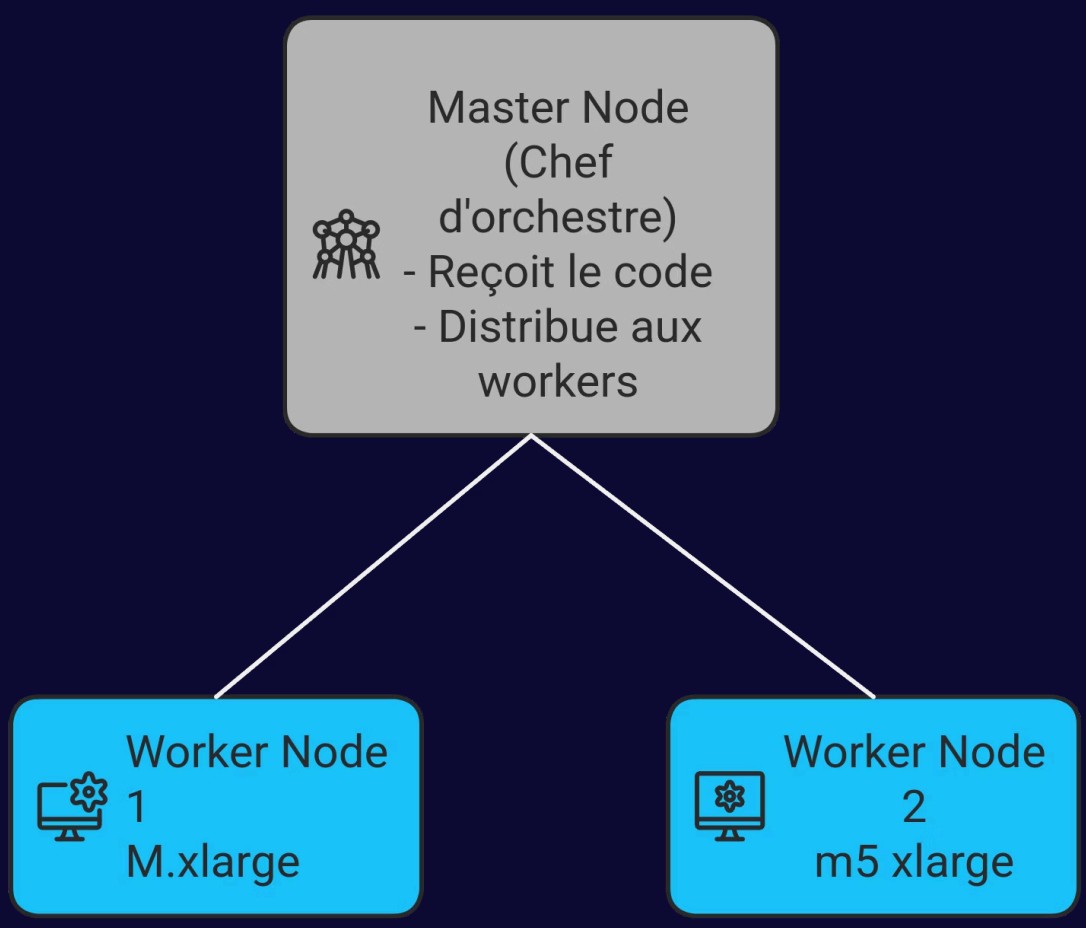
Core Nodes

Type : m5.xlarge

Quantité : 2 à 4 selon la charge

Rôle : calcul distribué et stockage HDFS

Coût : ~0.38 €/heure pour 2 nodes



- Traitement en parallèle des données
- 10;000 image en 5 minutes
- Scalable : AJout facile de machines

Coût total : environ 0.60 € par heure

Région : eu-west-3 (Paris) - conformité RGPD garantie

Contrôle des Coûts EMR

Un des plus grands défis avec AWS EMR, surtout pour une start-up au budget serré, est la maîtrise des coûts. La facturation à l'heure peut rapidement devenir une épine dans le pied si l'on n'est pas vigilant.

Le Piège des Clusters Oubliés ⚠️

Un cluster EMR laissé allumé peut coûter **2-5€ par heure !**

- Une nuit oubliée : 20-50€
- Un week-end entier : plus de 100€
- Le budget projet peut être explosé en quelques heures seulement.

Ma Stratégie pour un Budget Maîtrisé

Développement Local (80% du temps)

- Installation de PySpark en local pour un environnement de test rapide et gratuit.
- Tests rigoureux sur un échantillon de 100 images.
- Validation complète et débuggage du code AVANT de passer sur EMR.

Sessions EMR Courtes (20% du temps)

- Sessions maximales de 1 à 2 heures pour des tests ciblés.
- Préparation minutieuse des tests pour optimiser le temps d'utilisation du cluster.
- Le cluster est arrêté immédiatement après chaque série de tests.

Pipeline de Traitement

Le pipeline de traitement se décompose en 4 étapes :

1

Chargement

Lecture des images depuis S3 en format binaryFile. Spark lit les dossiers de manière récursive. Extraction automatique des labels depuis les chemins.

2

Extraction de Features

Utilisation de MobileNetV2 pré-entraîné sur ImageNet. Transfer Learning : on n'entraîne pas, on extrait les features. Sortie : 1280 features par image.

3

Réduction de Dimensionnalité

PCA pour passer de 1280 à 50 dimensions. Conservation d'environ 95% de la variance. Standardisation préalable des données (important pour PCA).

4

Sauvegarde

Format Parquet optimisé. Résultats disponibles sur S3. Prêts pour l'entraînement du modèle de classification.

Point critique : Broadcast des poids du modèle (détail à venir)

Le Broadcast - Concept

Le broadcast est **LA technique d'optimisation cruciale** pour ce projet. Laissez-moi vous expliquer pourquoi.

Sans broadcast ❌

- Chaque worker doit charger le modèle MobileNetV2
- Le modèle fait 14 MB
- Avec 10 workers : 10 × 14 MB = 140 MB de transfert réseau
- Pire : le modèle est rechargé à CHAQUE nouvelle tâche
- Si on a 100 tâches, c'est 100 chargements !
- **Résultat :** 30 minutes de traitement

Avec broadcast ✅

- Le driver charge le modèle UNE SEULE FOIS
- Il le broadcaste efficacement à tous les workers
- Chaque worker garde le modèle EN MÉMOIRE
- Tous les batches réutilisent la même copie
- **Résultat :** 5 minutes de traitement
- **Gain : 6 fois plus rapide !**

Analogie : C'est comme si vous deviez distribuer un document à 30 élèves. Sans broadcast, chaque élève va individuellement à la bibliothèque le chercher. Avec broadcast, le prof fait 30 photocopies en une fois et les distribue. Beaucoup plus efficace !

Le Broadcast - Code

Voici comment j'ai implémenté le broadcast en pratique :

Sur le driver (machine principale) :

```
model = MobileNetV2(weights='imagenet')
weights = model.get_weights()
broadcast_weights = sc.broadcast(weights)
```

Sur chaque worker :

```
def model_fn():
    model = MobileNetV2(weights=None)
    model.set_weights(broadcast_weights.value)
    return model
```

weights=None

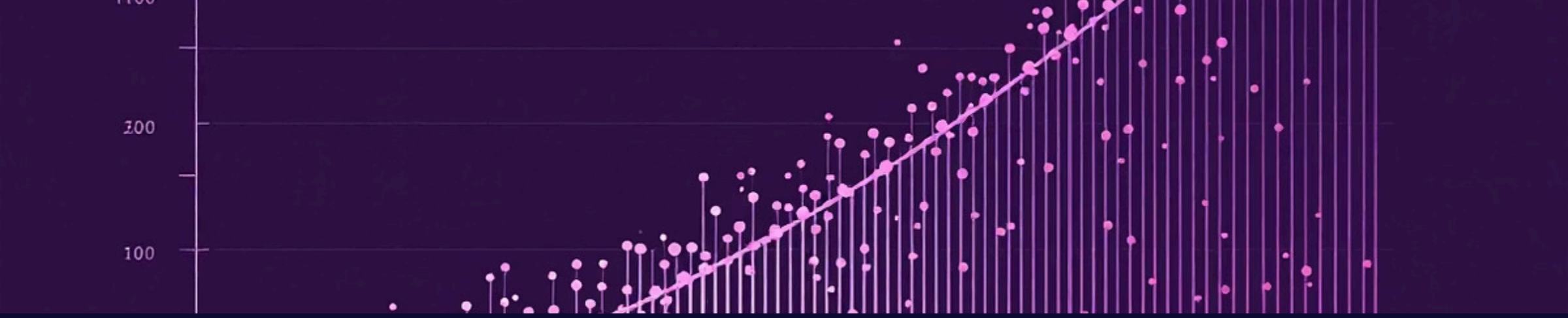
Pas de téléchargement répété

.value

Accède à la variable broadcastée

Pandas UDF Scalar Iterator

Le modèle est chargé une seule fois par worker, puis réutilisé



La PCA

Passons à la réduction de dimensionnalité par PCA.

Le problème

- MobileNetV2 génère 1280 features par image
- Pour 10,000 images : 50 MB
- Pour 1,000,000 images : 5 GB !
- Coûteux en stockage S3
- Lent à traiter par la suite

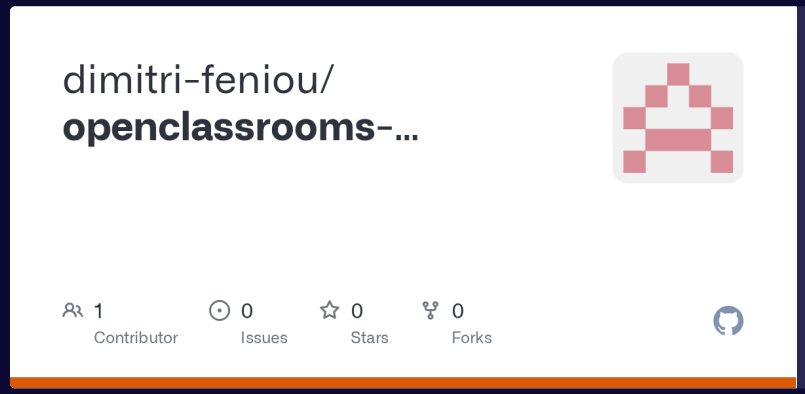
La solution PCA

- Réduction de 1280 à 50 dimensions
- Conservation de ~95% de la variance (excellente préservation de l'information)
- Réduction de 96% de la taille des données
- Pour 1,000,000 d'images : 200 MB au lieu de 5 GB
- **Gain : 25 fois moins d'espace !**

Processus :

1. Standardisation des features (moyenne 0, écart-type 1)
2. Application de la PCA avec 350 composantes
3. Analyse de la variance expliquée
4. Sauvegarde des résultats

Présentation Vidéo du projet



GitHub

openclassrooms-projet11-aws-big-data/Presentation_vidéo_aws.m...

Contribute to dimitri-feniou/openclassrooms-projet11-aws-big-data development by creating an account on GitHub.

Conclusion & Bilan du Projet

Points forts

- Scalabilité prouvée (testée jusqu'à 10 000 images, capable de millions)
- Architecture testée et Spark distribution automatique
- Conformité RGPD (100% des données en région EU Paris)
- Coûts maîtrisés (budget de 4€ respecté)
- Performances optimales (Broadcast 6x plus rapide, PCA 96% de réduction)

Points d'amélioration

- Complexité initiale (configuration EMR + IAM + S3)
- Risque de coûts (vigilance constante requise)
- Temps de démarrage (5-10 minutes pour instancier le cluster)

Alternatives à considérer

- Databricks (interface plus intuitive)
- AWS Glue (serverless)
- Google Dataproc