



Distributed Training Pipeline with Kubernetes & PyTorch

Rapport Tutoriel Technique – Projet fin du module

Master Intelligence artificielle / Module Déploiement des projets IA

Sperviseur :
Professeur Fahd Kalloubi

Auteurs :
AIT EL ARBI Ezzahra
EL AAMRANI Zahira
ERRAMI Hafssa

Année universitaire : 2025–2026

Table des matières

| | | |
|----------|--|-----------|
| 1 | Vue d'Ensemble | 5 |
| 1.1 | Objectifs | 5 |
| 1.2 | Cas d'Usage | 5 |
| 2 | Architecture du Système | 6 |
| 2.1 | Vue Globale | 6 |
| 2.2 | Technologies Utilisées | 7 |
| 3 | Fonctionnalités du Pipeline | 8 |
| 3.1 | Pipeline MLOps Complet | 8 |
| 3.1.1 | Data Ingestion | 8 |
| 3.1.2 | Feature Engineering | 8 |
| 3.1.3 | Distributed Training | 8 |
| 3.1.4 | Model Evaluation | 8 |
| 3.1.5 | Model Versioning | 8 |
| 3.1.6 | Automated Deployment | 8 |
| 3.2 | Optimisations | 9 |
| 4 | Prérequis | 10 |
| 4.1 | Système d'Exploitation | 10 |
| 4.2 | Ressources Matérielles | 10 |
| 4.3 | Logiciels Requis | 10 |
| 5 | Installation | 11 |
| 5.1 | Étape 1 : Environnement de Base | 11 |
| 5.1.1 | Installation sur Windows (WSL2) | 11 |
| 5.1.2 | Installation sur Linux | 11 |
| 5.2 | Étape 2 : Installation de Kubernetes (Minikube) | 11 |
| 5.2.1 | Installation de Minikube | 11 |
| 5.2.2 | Installation de kubectl | 12 |
| 5.2.3 | Vérification des installations | 12 |
| 5.3 | Étape 3 : Démarrage du Cluster Kubernetes | 12 |
| 5.4 | Étape 4 : Installation de Kubeflow Training Operator | 12 |
| 5.5 | Étape 5 : Installation des Dépendances Python | 13 |
| 6 | Utilisation du Pipeline | 14 |
| 6.1 | Clonage du Projet | 14 |
| 6.2 | Création des Fichiers de Configuration | 14 |
| 6.2.1 | Fichier <code>lightweight-pipeline.yaml</code> | 14 |

| | | |
|-----------|---|-----------|
| 6.2.2 | Fichier <code>quick-run.sh</code> | 14 |
| 6.3 | Lancement du Pipeline | 15 |
| 6.4 | Logs et Sorties | 15 |
| 7 | Pipeline MLOps Détailé | 16 |
| 7.1 | Data Ingestion | 16 |
| 7.2 | Feature Engineering | 16 |
| 7.3 | Distributed Training | 17 |
| 7.4 | Model Evaluation | 17 |
| 7.5 | Model Versioning | 17 |
| 8 | Résultats | 19 |
| 8.1 | Métriques de Performance | 19 |
| 8.1.1 | Configuration Expérimentale | 19 |
| 8.1.2 | Résultats Quantitatifs | 19 |
| 8.2 | Analyse Graphique des Performances | 19 |
| 8.3 | Temps d'Exécution du Pipeline | 20 |
| 8.4 | Visualisation de l'Utilisation des Ressources | 21 |
| 9 | Monitoring | 22 |
| 9.1 | Surveillance de l'Entraînement | 22 |
| 9.1.1 | Statut du PyTorchJob | 22 |
| 9.1.2 | Pods d'Entraînement | 22 |
| 9.1.3 | Logs en Temps Réel | 23 |
| 9.2 | Accès à la Console MinIO | 24 |
| 9.2.1 | Port-forwarding | 24 |
| 9.2.2 | Identifiants de Connexion | 24 |
| 9.2.3 | Navigation dans MinIO | 24 |
| 10 | Déploiement du Modèle | 26 |
| 10.1 | Déploiement du Service d'Inférence | 26 |
| 10.1.1 | Création du Manifeste Kubernetes | 26 |
| 10.1.2 | Déploiement sur le Cluster | 26 |
| 10.2 | Test de l'API d'Inférence | 26 |
| 10.2.1 | Exposition du Service | 26 |
| 10.2.2 | Requête de Test | 26 |
| 10.2.3 | Réponse Attendue | 27 |
| 11 | Troubleshooting | 28 |
| 11.1 | Problèmes Courants | 28 |
| 12 | Conclusion | 29 |

Table des figures

| | | |
|-----|---|----|
| 2.1 | Architecture globale du pipeline distribué | 6 |
| 5.1 | Vérification des installations | 12 |
| 5.2 | Vérification du bon fonctionnement du cluster | 12 |
| 5.3 | Vérification training - operator | 13 |
| 6.1 | Logs du PyTorchJob et sortie attendue du pipeline | 15 |
| 8.1 | Évolution des métriques de performance (accuracy) | 20 |
| 8.2 | Évolution des métriques de performance (loss) | 20 |
| 9.1 | Statut du PyTorchJob | 22 |
| 9.2 | Pods d'Entraînement | 23 |
| 9.3 | Logs du PyTorchJob et sortie attendue du pipeline | 23 |
| 9.4 | minio home | 24 |
| 9.5 | Interface minio bucket models | 25 |

Liste des tableaux

| | | |
|-----|--------------------------------------|----|
| 2.1 | Stack technologique utilisée | 7 |
| 4.1 | Configuration matérielle requise | 10 |
| 8.1 | Résultats expérimentaux sur CIFAR-10 | 19 |
| 8.2 | Temps d'exécution du pipeline MLOps | 21 |

Chapitre 1

Vue d'Ensemble

Ce projet implémente un pipeline MLOps complet pour l'entraînement distribué de réseaux de neurones profonds sur Kubernetes. Il démontre les bonnes pratiques industrielles du Machine Learning en production.

1.1 Objectifs

- Entraînement distribué avec PyTorch DDP
- Orchestration avec Kubernetes
- Gestion des artefacts via MinIO
- Reproductibilité et scalabilité

1.2 Cas d'Usage

- Vision par ordinateur à grande échelle
- Environnements à ressources limitées
- Prototype MLOps académique ou industriel

Chapitre 2

Architecture du Système

2.1 Vue Globale

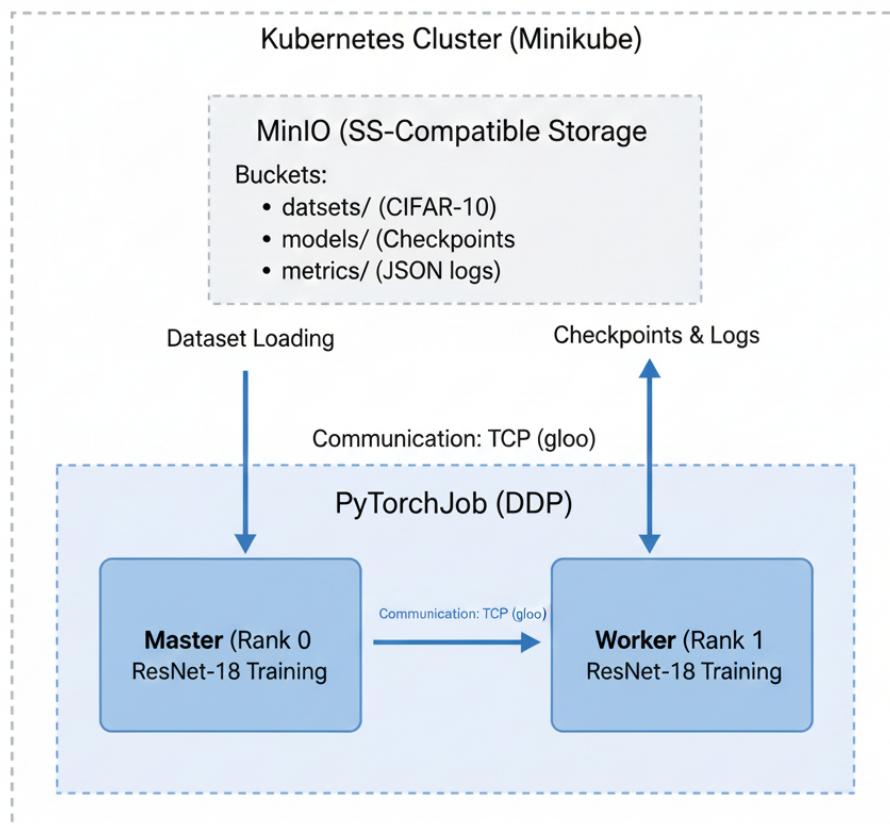


FIGURE 2.1 – Architecture globale du pipeline distribué

2.2 Technologies Utilisées

| Composant | Technologie | Version | Rôle |
|----------------------|----------------------------|---------|--------------------------|
| Orchestration | Kubernetes | 1.34+ | Gestion des conteneurs |
| Cluster | Minikube | 1.37+ | Cluster Kubernetes local |
| Training Framework | PyTorch | 2.0.0 | Deep Learning |
| Distributed Training | PyTorch DDP | – | Parallélisation |
| Job Operator | Kubeflow Training Operator | 1.8.1 | Gestion des PyTorchJob |
| Storage | MinIO | 2023-09 | Object storage (S3) |
| Dataset | CIFAR-10 | – | 60K images 32×32 |
| Model | ResNet-18 | – | CNN (11M paramètres) |

TABLE 2.1 – Stack technologique utilisée

Chapitre 3

Fonctionnalités du Pipeline

3.1 Pipeline MLOps Complet

3.1.1 Data Ingestion

- Téléchargement automatique du dataset CIFAR-10
- Stockage persistant dans MinIO pour réutilisation
- Support des datasets personnalisés

3.1.2 Feature Engineering

- Data augmentation (random flip, random crop)
- Normalisation basée sur les statistiques CIFAR-10
- Distribution intelligente des données entre les workers

3.1.3 Distributed Training

- Entraînement distribué avec PyTorch Distributed Data Parallel (DDP)
- Backend de communication : Gloo (optimisé CPU)
- Synchronisation automatique des gradients entre workers
- Architecture 1 Master + 1 Worker (extensible à N workers)

3.1.4 Model Evaluation

- Calcul des métriques à chaque epoch
- Train accuracy et Test accuracy
- Train loss et Test loss

3.1.5 Model Versioning

- Sauvegarde des checkpoints à chaque epoch
- Versioning basé sur des timestamps
- Maintien automatique du modèle *latest*

3.1.6 Automated Deployment

- Modèles prêts pour le déploiement

-
- Exposition possible via une API REST pour l'inférence

3.2 Optimisations

- **Efficacité mémoire** : batch size optimisé et gradient accumulation
- **Performance** : DataLoader multi-threaded, option `pin_memory`
- **Robustesse** : gestion des erreurs et mécanisme de retry automatique
- **Observabilité** : logs détaillés et métriques structurées

Chapitre 4

Prérequis

4.1 Système d'Exploitation

Le pipeline est compatible avec les systèmes d'exploitation suivants :

- **Windows** : Windows 10/11 avec WSL2 activé
- **Linux** : Ubuntu 20.04 ou version équivalente
- **macOS** : macOS 11 ou supérieur avec Docker Desktop

4.2 Ressources Matérielles

| Composant | Minimum | Recommandé |
|-----------|-----------|------------|
| CPU | 2 cœurs | 4 cœurs |
| RAM | 6 GB | 8 GB |
| Disque | 20 GB | 30 GB |
| GPU | Optionnel | Optionnel |

TABLE 4.1 – Configuration matérielle requise

4.3 Logiciels Requis

Les logiciels suivants doivent être installés avant le déploiement du pipeline :

- **Docker** 20.10 ou version ultérieure
- **Kubernetes** (via Minikube)
- **Python** 3.9 ou version ultérieure
- **Git** pour la gestion du code source

Chapitre 5

Installation

5.1 Étape 1 : Environnement de Base

5.1.1 Installation sur Windows (WSL2)

Dans PowerShell (exécuté en tant qu'administrateur) :

```
wsl --install
```

Après le redémarrage de Windows, ouvrir Ubuntu (WSL2) puis exécuter :

```
# Mise      jour du syst me
sudo apt update && sudo apt upgrade -y

# Installation de Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Fermer et rouvrir WSL pour appliquer les changements
exit
```

5.1.2 Installation sur Linux

```
# Installation de Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER
newgrp docker

# V rification
docker --version
```

5.2 Étape 2 : Installation de Kubernetes (Minikube)

5.2.1 Installation de Minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/
      minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

5.2.2 Installation de kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/
          release/stable.txt)/bin/linux/amd64/kubectl"
sudo install kubectl /usr/local/bin/
```

5.2.3 Vérification des installations

```
minikube version
kubectl version --client
```

```
[venv] ezzahra@DESKTOP-H1F0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow/downloaded_artifacts$ minikube version
minikube version: v1.37.0
commit: 65318f4cff9c12cc87ec9eb8f4cdd57b25047f3
[venv] ezzahra@DESKTOP-H1F0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow/downloaded_artifacts$ |
```

FIGURE 5.1 – Vérification des installations

5.3 Étape 3 : Démarrage du Cluster Kubernetes

```
# Creation du cluster Kubernetes local
minikube start --cpus=2 --memory=6144 --disk-size=20g --driver=
      docker
```

Vérification du bon fonctionnement du cluster :

```
kubectl cluster-info
kubectl get nodes
```

Sortie attendue :

```
[venv] ezzahra@DESKTOP-H1F0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow/downloaded_artifacts$ kubectl get nodes
NAME     STATUS   ROLES    AGE     VERSION
minikube Ready    control-plane   2d18h   v1.34.0
[venv] ezzahra@DESKTOP-H1F0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow/downloaded_artifacts$ |
```

FIGURE 5.2 – Vérification du bon fonctionnement du cluster

5.4 Étape 4 : Installation de Kubeflow Training Operator

```
# Cloner le repository
git clone --depth 1 --branch v1.8.1 https://github.com/kubeflow/
      training-operator.git
cd training-operator
```

```
# Installer l'opérateur
kubectl apply -k manifests/overlays/standalone
```

Attendre que l'opérateur soit prêt :

```
kubectl wait --for=condition=ready pod -l app=training-operator -n
kubeflow --timeout=300s
```

Vérification :

```
kubectl get pods -n kubeflow
```

Sortie attendue :

| NAME | READY | STATUS | RESTARTS | AGE |
|-------------------------------|-------|---------|----------|-----|
| training-operator-xxxxx-xxxxx | 1/1 | Running | 0 | 1m |

```
(venv) ezzahra@DESKTOP-HIF0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow/downloaded_artifacts$ kubectl get pods -n kubeflow
NAME           READY   STATUS    RESTARTS   AGE
training-operator-57f54b87c4-sp5vn  1/1     Running   0          2d18h
(venv) ezzahra@DESKTOP-HIF0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow/downloaded_artifacts$ |
```

FIGURE 5.3 – Vérification training - operator

5.5 Étape 5 : Installation des Dépendances Python

```
# Cr ation de l'environnement virtuel
cd ~
mkdir mllops-distributed-project
cd mllops-distributed-project

python3 -m venv venv
source venv/bin/activate

# Installation des d pendances Python
pip install --upgrade pip
pip install torch torchvision minio requests pillow
```

Chapitre 6

Utilisation du Pipeline

6.1 Clonage du Projet

Se placer dans le répertoire de travail :

```
cd ~/mlops-distributed-project
```

Cloner le dépôt Git du projet :

```
git clone <votre-repo-url> .
```

Alternativement, les fichiers peuvent être créés manuellement comme décrit ci-dessous.

6.2 Création des Fichiers de Configuration

6.2.1 Fichier `lightweight-pipeline.yaml`

Créer le fichier de configuration Kubernetes :

```
nano lightweight-pipeline.yaml
```

Copier le contenu complet du pipeline dans ce fichier (voir fichier `lightweight-pipeline.yaml`).

Points clés du fichier :

- Namespace `mlops-light`
- Déploiement du service MinIO
- ConfigMap contenant le code d'entraînement
- PyTorchJob avec un Master et un Worker
- Ressources CPU et mémoire optimisées

6.2.2 Fichier `quick-run.sh`

Créer le script de lancement :

```
nano quick-run.sh  
chmod +x quick-run.sh
```

Copier le contenu du script de lancement dans ce fichier (voir `quick-run.sh`).

6.3 Lancement du Pipeline

Le pipeline complet peut être lancé à l'aide d'une seule commande :

```
./quick-run.sh
```

Ce script automatise les étapes suivantes :

- Nettoyage des namespaces Kubernetes précédents
- Déploiement du service MinIO
- Création automatique des buckets nécessaires
- Lancement du PyTorchJob distribué
- Affichage des logs d'entraînement en temps réel
- Upload le modèle final et les métriques vers Minio

6.4 Logs et Sorties

La figure suivante illustre un exemple de sortie attendue lors de l'exécution du pipeline, incluant les logs d'entraînement du PyTorchJob distribué.

```
(venv) ezzahra@DESKTOP-H1F0QH3:~/mlops-distributed-training-project/Project-without-mlflow$ ./quick-run2.sh
=====
Pipeline MLOps Distribué
1 Master + 2 Workers
=====

[1/6] Nettoyage des anciens déploiements...
namespace "mlops-light" deleted

[2/6] Déploiement de l'infrastructure...
namespace/mlops-light created
persistentvolumeclaim/minio-pvc created
deployment.apps/minio created
service/minio created
configmap/training-code-light created
persistentvolumeclaim/training-output created
pytorchjob.kubeflow.org/resnet-light created

[3/6] Attente de MinIO...
pod/minio-b89cc6486-rv5ts condition met
✓ MinIO prêt

[4/6] Configuration des buckets MinIO...
✓ Bucket 'datasets' créé
✓ Bucket 'models' créé
✓ Bucket 'metrics' créé

[5/6] Lancement de l'entraînement distribué...

PyTorchJob:
NAME      STATE    AGE
resnet-light  Running  7s

Pods (1 Master + 2 Workers attendus):
NAME          READY  STATUS    RESTARTS  AGE
minio-b89cc6486-rv5ts  1/1   Running   0          8s
resnet-light-master-0  1/1   Running   0          7s
resnet-light-worker-0  0/1   Init:0/1  0          7s
resnet-light-worker-1  0/1   Init:0/1  0          7s

[6/6] Attente du démarrage des pods (2-3 min)... 

État actuel des pods:
NAME          READY  STATUS    RESTARTS  AGE
minio-b89cc6486-rv5ts  1/1   Running   0          39s
resnet-light-master-0  1/1   Running   0          38s
resnet-light-worker-0  1/1   Running   0          38s
resnet-light-worker-1  1/1   Running   0          38s

Vérification de la distribution:
- Master (Rank 0)
```

FIGURE 6.1 – Logs du PyTorchJob et sortie attendue du pipeline

Chapitre 7

Pipeline MLOps Détailé

Le pipeline MLOps mis en place est composé de cinq étapes principales, couvrant l'ensemble du cycle de vie du modèle, depuis l'ingestion des données jusqu'au versioning des modèles et des métriques.

7.1 Data Ingestion

Le téléchargement et la préparation du dataset CIFAR-10 sont effectués automatiquement à l'aide de `torchvision`.

```
# Téléchargement automatique de CIFAR-10
trainset = torchvision.datasets.CIFAR10(
    root='/data', train=True, download=True, transform=
        transform_train
)

testset = torchvision.datasets.CIFAR10(
    root='/data', train=False, download=True, transform=
        transform_test
)
```

Résultat : 50 000 images d'entraînement et 10 000 images de test.

7.2 Feature Engineering

Les données sont enrichies par des transformations et techniques d'augmentation afin d'améliorer la capacité de généralisation du modèle.

```
# Transformations et augmentation
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),           # Flip horizontal
    transforms.RandomCrop(32, padding=4),          # Crop al atoire
    transforms.ToTensor(),                        # Conversion en
                                                # tenseur
    transforms.Normalize(
        (0.4914, 0.4822, 0.4465),
        (0.2023, 0.1994, 0.2010)
)
```

```
) # Normalisation  
])
```

Résultat : DataLoaders configurés avec `DistributedSampler` pour un chargement distribué efficace.

7.3 Distributed Training

L'entraînement distribué est réalisé à l'aide de PyTorch Distributed Data Parallel (DDP), permettant la synchronisation automatique des gradients entre les workers.

```
# Configuration du training distribu  
if world_size > 1:  
    dist.init_process_group(backend='gloo', ...)  
    model = DDP(model)
```

L'entraînement est exécuté sur 10 epochs :

```
# Entrainement sur 10 epochs  
for epoch in range(10):  
    # Training loop avec synchronisation automatique  
    ...
```

Stratégie utilisée :

- PyTorch Distributed Data Parallel (DDP)
- Backend de communication : Gloo (optimisé CPU)
- Synchronisation automatique des gradients

7.4 Model Evaluation

L'évaluation du modèle est effectuée à chaque epoch afin de suivre les performances sur les données d'entraînement et de test.

```
# valuation      chaque epoch  
model.eval()  
with torch.no_grad():  
    for inputs, targets in testloader:  
        outputs = model(inputs)  
        loss = criterion(outputs, targets)  
        # Calcul accuracy  
        ...
```

Métriques calculées :

- Train Loss et Train Accuracy
- Test Loss et Test Accuracy

7.5 Model Versioning

Les modèles et métriques sont versionnés automatiquement afin d'assurer la traçabilité et la reproductibilité des expériences.

```
# Sauvegarde avec timestamp
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

# Checkpoint chaque epoch
torch.save(checkpoint, f'/output/checkpoint_epoch_{epoch+1}.pth')

# Upload vers MinIO
minio_client.fput_object(
    "models",
    f"resnet-cifar10/model_v{timestamp}.pth",
    final_model_path
)
```

Organisation du stockage dans MinIO :

```
models/
    resnet-cifar10/
        checkpoint_epoch_1.pth
        checkpoint_epoch_2.pth
        ...
        checkpoint_epoch_10.pth
        model_v20241228_143022.pth
        model_latest.pth

metrics/
    resnet-cifar10/
        metrics_v20241228_143022.json
```

Chapitre 8

Résultats

8.1 Métriques de Performance

8.1.1 Configuration Expérimentale

Les expériences ont été menées avec la configuration suivante :

- **Modèle** : ResNet-18 (11 millions de paramètres)
- **Dataset** : CIFAR-10 (60 000 images)
- **Nombre d'epochs** : 10
- **Batch size** : 128
- **Learning rate** : 0.1 avec stratégie de *cosine annealing*
- **Optimiseur** : SGD (momentum = 0.9, weight_decay = 5×10^{-4})
- **Workers** : 1 Master + 1 Worker

8.1.2 Résultats Quantitatifs

| Epoch | Train Loss | Train Acc | Test Loss | Test Acc |
|-------|------------|-----------|-----------|----------|
| 1 | 2.168 | 29.70% | 1.63 | 39.00% |
| 5 | 1.058 | 62.06% | 1.025 | 63.32% |
| 10 | 0.734 | 73.43% | 0.748 | 73.43% |

TABLE 8.1 – Résultats expérimentaux sur CIFAR-10

8.2 Analyse Graphique des Performances

La figure suivante illustre l'évolution des performances du modèle au cours de l'entraînement.

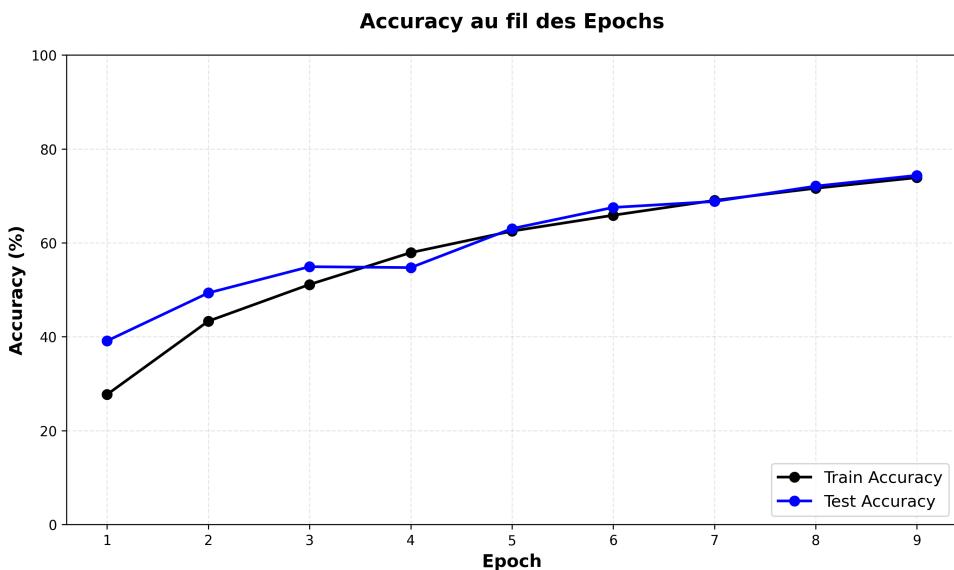


FIGURE 8.1 – Évolution des métriques de performance (accuracy)

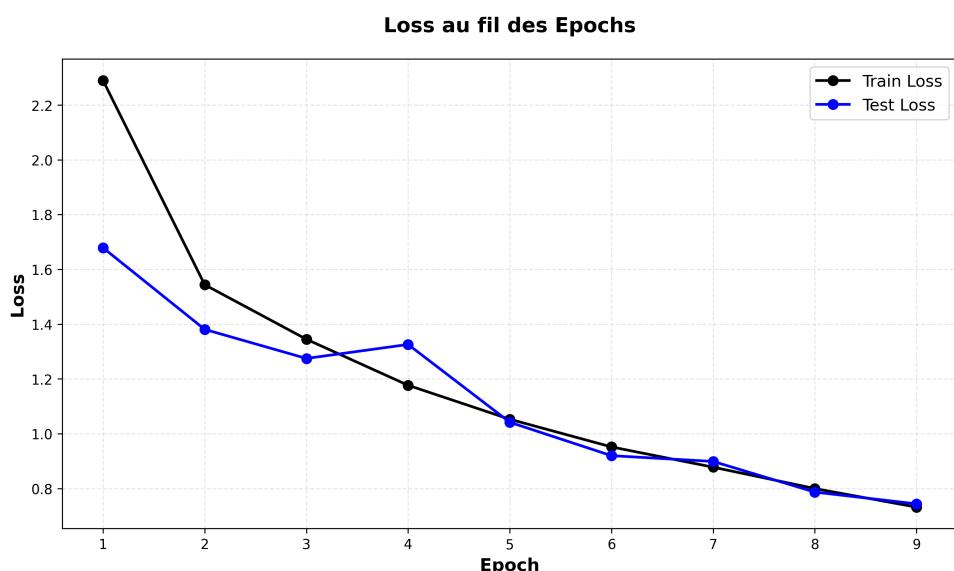


FIGURE 8.2 – Évolution des métriques de performance (loss)

8.3 Temps d'Exécution du Pipeline

Le tableau ci-dessous présente le temps d'exécution moyen observé pour chaque phase du pipeline.

| Phase | Durée | Description |
|---------------|---------------------|---|
| Setup | 2–3 min | Déploiement de MinIO et création des buckets |
| Data Download | 1–2 min | Téléchargement du dataset CIFAR-10 (première exécution) |
| Training | 4–5 h | Entraînement complet sur 10 epochs |
| Saving | 1–2 min | Sauvegarde et upload des modèles vers MinIO |
| Total | 4h :4min – 5h :7min | Exécution complète du pipeline |

TABLE 8.2 – Temps d'exécution du pipeline MLOps

8.4 Visualisation de l'Utilisation des Ressources

L'utilisation des ressources du cluster Kubernetes peut être surveillée en temps réel pendant l'entraînement à l'aide des commandes suivantes dans un nouveau terminal :

```
# Surveillance des ressources du cluster
kubectl top nodes
kubectl top pods -n mlops-light
```

Chapitre 9

Monitoring

Le monitoring permet de suivre en temps réel l'état du pipeline MLOps, l'exécution du PyTorchJob distribué ainsi que l'accès aux artefacts générés.

9.1 Surveillance de l'Entraînement

9.1.1 Statut du PyTorchJob

La commande suivante permet de vérifier l'état du job d'entraînement distribué :

```
kubectl get pytorchjob -n mlops-light
```

Sortie attendue :

```
(venv) ezzahra@DESKTOP-HIF0QH3:~/mlops-distributed-training-project/Project-whitout-mlflow$ kubectl get pytorchjob -n mlops-light
NAME      STATE   AGE
resnet-light  Running  6h19m
```

FIGURE 9.1 – Statut du PyTorchJob

États possibles du PyTorchJob :

- **Created** : Job créé, pods en cours de démarrage
- **Running** : Entraînement en cours
- **Succeeded** : Entraînement terminé avec succès
- **Failed** : Échec de l'entraînement

9.1.2 Pods d'Entraînement

La commande suivante permet d'observer les pods actifs dans le namespace :

```
kubectl get pods -n mlops-light
```

Sortie attendue :

```
[6/6] Attente du démarrage des pods (2-3 min)...
```

```
Etat actuel des pods:
```

| NAME | READY | STATUS | RESTARTS | AGE |
|-----------------------|-------|---------|----------|-----|
| minio-b89cc6486-rv5ts | 1/1 | Running | 0 | 39s |
| resnet-light-master-0 | 1/1 | Running | 0 | 38s |
| resnet-light-worker-0 | 1/1 | Running | 0 | 38s |
| resnet-light-worker-1 | 1/1 | Running | 0 | 38s |

```
Vérification de la distribution:
```

- Master (Rank 0)
- Worker 1 (Rank 1)
- Worker 2 (Rank 2)

```
Attente que le Master soit prêt...
```

```
pod/resnet-light-master-0 condition met
```

FIGURE 9.2 – Pods d’Entraînement

9.1.3 Logs en Temps Réel

Logs du Master (Rank 0)

```
kubectl logs -f \
-l training.kubeflow.org/job-name=resnet-light,training.kubeflow.org/replica-type=master \
-n mlops-light
```

Logs du Worker (Rank 1)

```
kubectl logs -f \
-l training.kubeflow.org/job-name=resnet-light,training.kubeflow.org/replica-type=worker \
-n mlops-light
```

Exemple de logs d’entraînement :

```
=====
[Rank 0/2] 🚀 DÉMARRAGE ENTRAÎNEMENT DISTRIBUÉ
=====

[Rank 0] 📈 ÉTAPE 1/5: Data Ingestion
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to /data/cifar-10-python.tar.gz
100% [██████████] 170498071/170498071 [05:12<00:00, 545967.94it/s]
Extracting /data/cifar-10-python.tar.gz to /data
Files already downloaded and verified
[Rank 0] ✓ 50000 train, 10000 test

[Rank 0] 🏠 ÉTAPE 2/5: Feature Engineering
[Rank 0] ✓ DataLoaders configurés

[Rank 0] 🔍 ÉTAPE 3/5: Model Creation
[Rank 0] ✓ ResNet-18 créé

[Rank 0] ⚡ Entraînement 10 epochs...

Epoch 1/10 [0/196] Loss: 2.637 Acc: 7.03%
Epoch 1/10 [50/196] Loss: 2.920 Acc: 19.53%
```

FIGURE 9.3 – Logs du PyTorchJob et sortie attendue du pipeline

9.2 Accès à la Console MinIO

L'interface web de MinIO permet de visualiser et télécharger les modèles, checkpoints et métriques générés par le pipeline.

9.2.1 Port-forwarding

```
kubectl port-forward -n mlops-light svc/minio 9001:9001
```

Ensuite, ouvrir le navigateur à l'adresse suivante :

<http://localhost:9001>

9.2.2 Identifiants de Connexion

- **Username** : minioadmin
- **Password** : minioadmin

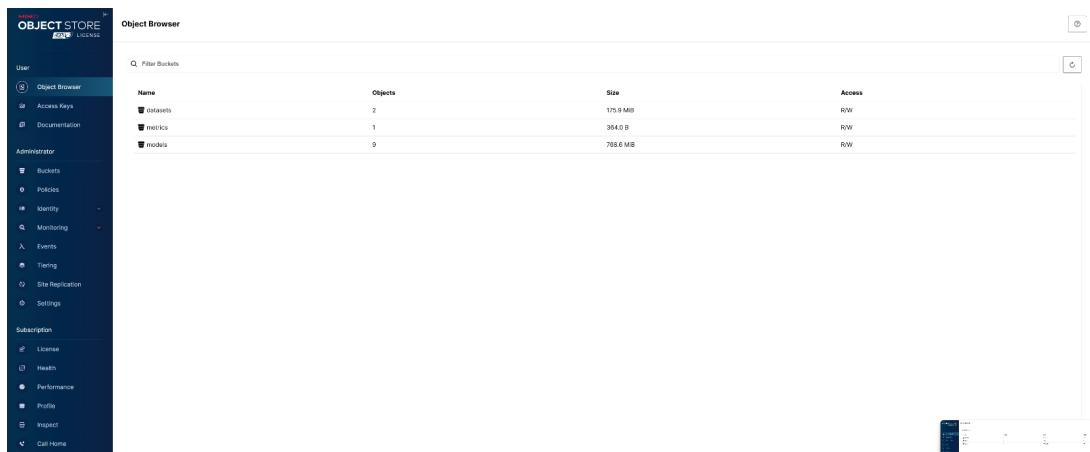


FIGURE 9.4 – minio home

9.2.3 Navigation dans MinIO

- Cliquer sur *Buckets*
- Sélectionner le bucket **models**
- Naviguer dans **resnet-cifar10/**
- Télécharger les checkpoints et les fichiers de métriques

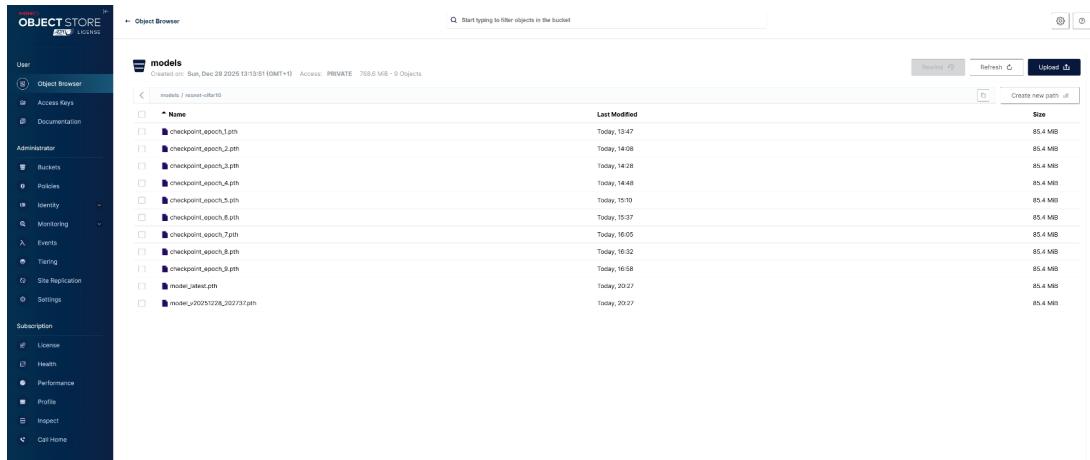


FIGURE 9.5 – Interface minio bucket models

Chapitre 10

Déploiement du Modèle

10.1 Déploiement du Service d’Inférence

10.1.1 Crédit du Manifeste Kubernetes

Créer le fichier de déploiement du service d’inférence :

```
nano deploy-inference.yaml
```

Puis copier le contenu complet du fichier `deploy-inference.yaml` définissant :

10.1.2 Déploiement sur le Cluster

```
kubectl apply -f deploy-inference.yaml
```

Attendre que le pod soit prêt :

```
kubectl wait --for=condition=ready pod \
-l app=inference -n mlops-light --timeout=300s
```

10.2 Test de l’API d’Inférence

10.2.1 Exposition du Service

Exposer le service d’inférence localement via un port-forwarding :

```
kubectl port-forward -n mlops-light svc/inference 8080:8080 &
```

10.2.2 Requête de Test

Tester l’API à l’aide d’une image :

```
curl -X POST -F "file=@test_image.png" http://localhost:8080/
predict
```

10.2.3 Réponse Attendue

```
{  
  "prediction": "cat",  
  "confidence": 0.9234  
}
```

Chapitre 11

Troubleshooting

11.1 Problèmes Courants

- Pods en Pending
- ImagePullBackOff
- CrashLoopBackOff
- MinIO inaccessible

(Pour plus de détails, voir le github repo : <https://github.com/Ezzahra-RB/Distributed-Training-Pipeline-with-Kubernetes-PyTorchJob>)

Chapitre 12

Conclusion

Ce projet démontre la faisabilité d'un pipeline MLOps distribué complet dans un environnement local à ressources limitées, en s'appuyant sur Kubernetes, PyTorch Distributed Data Parallel et MinIO. Tout en respectant les standards industriels de reproductibilité, de scalabilité et de robustesse. Il met en évidence l'importance de l'orchestration, de la reproductibilité et du versioning dans les projets d'Intelligence Artificielle modernes. Les résultats obtenus sur le dataset CIFAR-10 démontrent la pertinence de l'architecture proposée ainsi que la stabilité du processus d'entraînement. Enfin, ce travail constitue une base solide pour des extensions futures, notamment l'entraînement à grande échelle, l'intégration de GPU et l'automatisation avancée des pipelines MLOps.