



# **Orchestration d'un Pipeline ML Spark avec Kubeflow sur K8s**

**Réalisée par :**

**EL AMARI Aya**

**ED-DAIF Kaouthar**

**ELKHALIFTE Khadija**

# Tables de matières :

I.	Objectifs du Projet :	4
II.	Importance de l'Orchestration pour les Workflows ML :	4
III.	Installation docker :	4
IV.	Kubernetes :	5
IV.1	Installer minikube :	5
IV.2	Start minikube :	5
V.	Installation kubeflow :	7
VI.	Configuration du Pipeline ML :	11
VI.1	Importations de Bibliothèques :	11
VI.2	Initialisation de la Session Spark :	11
VI.3	@dsl.component :	11
VI.4	@dsl.pipeline :	14
VI.5	Résultat :	18
VII.	Conclusion :	21

# Tables de figures :

Figure 1 : Docker desktop .....	4
Figure 2 : Installation minikube .....	5
Figure 3 : Lancer le cluster Kubernetes local .....	5
Figure 4 : Activer le Metrics Server dans un cluster Kubernetes .....	6
Figure 5 : Accéder au kubernetes dashboard.....	6
Figure 6 : Kubernetes dashboard.....	7
Figure 7 : Appliquer les ressources personnalisées nécessaires pour les Kubeflow Pipelines.....	8
Figure 8 : commande pour que Les ressource personnalisée spécifiée atteigne l'état "établi" dans Kubernetes.....	8
Figure 9: applique les ressources spécifiques pour l'environnement des Pipelines Kubeflow à partir du référentiel GitHub.....	9
Figure 10 :vérifier avec kubectl si tous les pods démarrent avec succès.....	9
Figure 11 :effectuer une redirection de port .....	10
Figure 12 :tableau de bord central Kubeflow .....	11
Figure 13: importation les bibliothèques.....	11
Figure 14:nitialisation de la Session Spark .....	11
Figure 15: Composant pour Charger les Données.....	12
Figure 16: Composant pour le Prétraitement des Données .....	13
Figure 17 : Composant pour Entraîner le Modèle .....	13
Figure 18: Composant pour Évaluer le Modèle .....	14
Figure 19 :Définition du Pipeline KFP (spark_mllib_pipeline) .....	15
Figure 20 : compile the pipeline.....	15
Figure 21 : l'exécution du code .....	15
Figure 22: fichier yaml généré .....	16
Figure 23: upload le fichier yaml .....	16
Figure 24: affiche notre Pipeline avec les 4 composants .....	17
Figure 25: lancer pipeline.....	17
Figure 26 : exécuter pipeline .....	18
Figure 27: résultats .....	18
Figure 28 pods dans kubernetes .....	19
Figure 29: name space de travail .....	20
Figure 30: pods dans name space .....	20
Figure 31: pods actuel .....	21

## I. Objectifs du Projet :

Le présent projet vise à établir une méthodologie complète pour orchestrer un pipeline ML basé sur Spark à l'aide de Kubeflow sur un cluster Kubernetes. Les objectifs spécifiques comprennent la conception et la mise en place d'un pipeline automatisé pour le traitement distribué des données, l'entraînement des modèles ML, leur évaluation et leur déploiement. Nous cherchons à créer une infrastructure robuste et évolutive, capable de gérer efficacement les workflows ML complexes, tout en maximisant les avantages des outils comme Spark, Kubeflow et Kubernetes.

## II. Importance de l'Orchestration pour les Workflows ML :

L'orchestration joue un rôle crucial dans les workflows ML, en particulier dans des environnements distribués. Elle permet de coordonner et de rationaliser les différentes phases du processus ML, depuis la préparation des données jusqu'au déploiement des modèles en passant par l'entraînement et l'évaluation. Cette orchestration contribue à l'automatisation des tâches, à la réduction des erreurs humaines, à l'amélioration de la reproductibilité des expériences, et à l'optimisation des ressources disponibles. En somme, elle est fondamentale pour assurer l'efficacité opérationnelle et la scalabilité des pipelines ML dans un contexte moderne de traitement de données massives.

## III. Installation docker :

Avant de débuter, si vous êtes nouveau dans l'utilisation de Docker, il est recommandé de prendre un moment pour vous familiariser avec celui-ci et d'installer l'environnement Docker. En effet, Minikube utilise le pilote Docker pour virtualiser l'environnement Kubernetes. En d'autres termes, Docker est utilisé comme un pilote par Minikube pour créer un cluster Kubernetes en exploitant ses capacités de virtualisation.

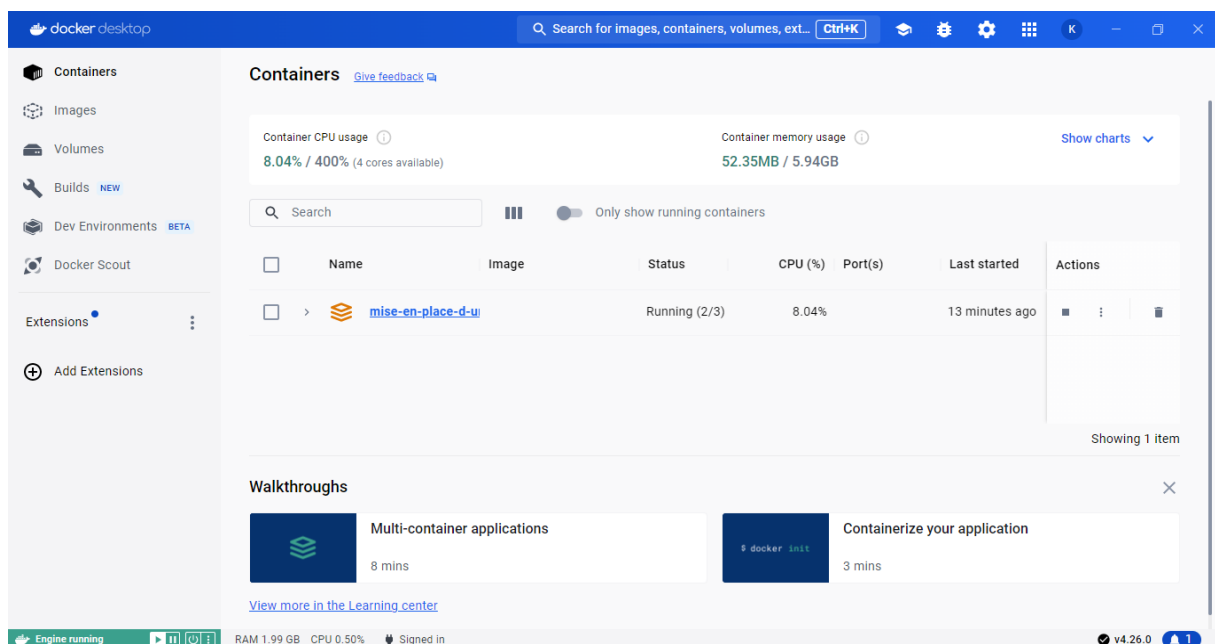


Figure 1 : Docker desktop

## IV. Kubernetes :

Pour déployer Kubernetes, diverses options s'offrent à vous en fonction de vos exigences, de votre infrastructure et de vos préférences. Parmi les méthodes populaires, on trouve kubectl pour des installations locales et des solutions cloud telles que Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS) ou Azure Kubernetes Service (AKS). Nous avons opté pour la méthode locale en utilisant Minikube : une configuration locale idéale pour le développement et les tests.

### IV.1 Installer minikube :

Obtenez le binaire Minikube correspondant à votre système d'exploitation à partir du site officiel. : <https://kubernetes.io/fr/docs/setup/learning-environment/minikube/>

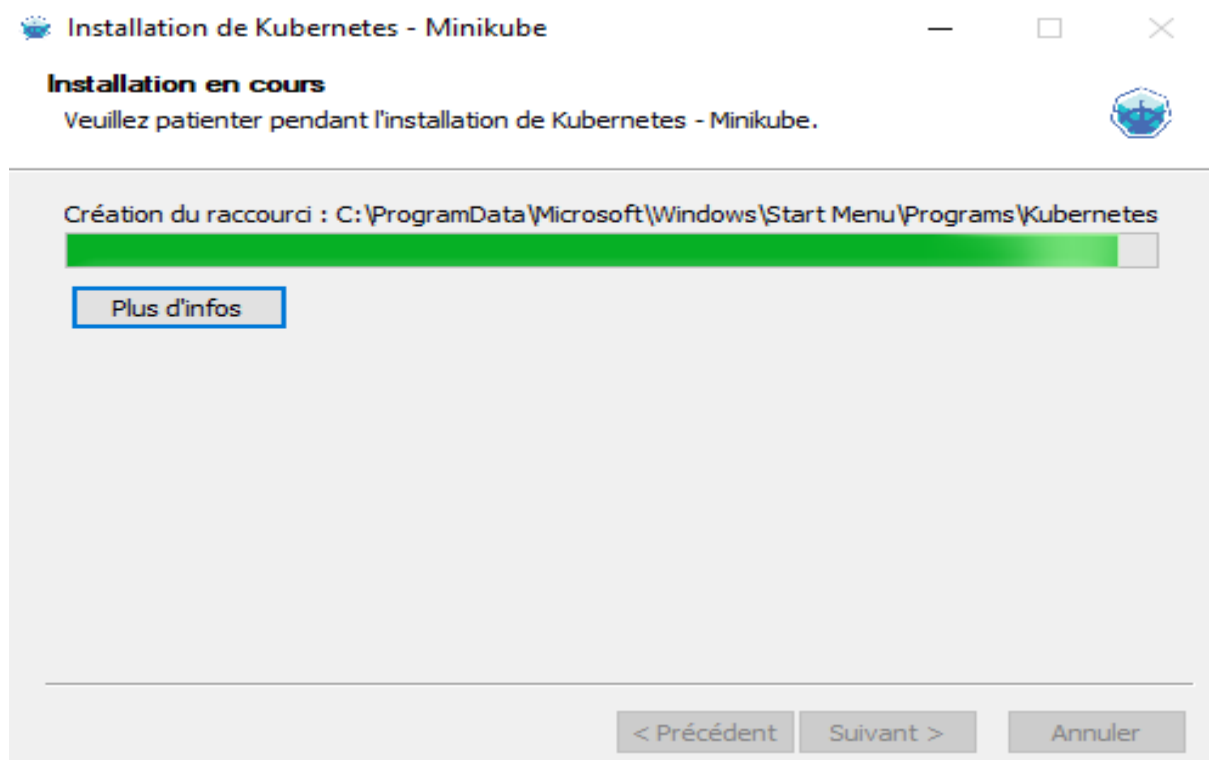


Figure 2 : Installation minikube

### IV.2 Start minikube

Utilisez la commande minikube start dans votre terminal pour lancer le cluster Kubernetes local.

```
C:\Users\dijs>minikube start --memory 3292 --cpus 4 --driver=docker --kubernetes-version=v1.22.9
W1214 22:28:27.503197 11928 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\Users\dijs\.docker\
contexts\meta\37a8e1ce19687d132fe29851dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: Le chemin d'accès spécifié est introuvable.
* minikube v1.32.0 sur Microsoft Windows 10 Pro 10.0.19045.3803 Build 19045.3803
* Utilisation du pilote docker basé sur la configuration de l'utilisateur
* Utilisation du pilote Docker Desktop avec le privilège root
* Démarrage du noeud de plan de contrôle minikube dans le cluster minikube
* Extraction de l'image de base...
* Création de docker container (CPU=4, Memory=3292Mo) ...
* Préparation de Kubernetes v1.22.9 sur Docker 24.0.7...
X Impossible de charger les images mises en cache : loading cached images: CreateFile C:\Users\dijs\.minikube\cache\images\amd64\registry.k8s.io\pause_3.5: Le chemin d'
accès spécifié est introuvable.
- Génération des certificats et des clés
- Démarrage du plan de contrôle ...
- Configuration des règles RBAC ...
* Vérification des composants Kubernetes...
- Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
* Modules activés: storage-provisioner, default-storageclass
* kubectl introuvable. Si vous en avez besoin, essayez : 'minikube kubectl -- get pods -A'
* Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et espace de noms "default" par défaut.
```

Figure 3 : Lancer le cluster Kubernetes local

La commande **minikube addons enable metrics-server** est utilisée pour activer le Metrics Server dans un cluster Kubernetes géré par Minikube.

Le Metrics Server est un composant essentiel qui collecte les métriques des ressources telles que CPU et mémoire à partir des pods et des nœuds du cluster. En activant le Metrics Server via cette commande dans Minikube, vous permettez au cluster Kubernetes de collecter des métriques sur l'utilisation des ressources, ce qui est utile pour surveiller les performances du cluster, échelonner les applications en fonction de l'utilisation des ressources, et fournir des données essentielles pour la mise à l'échelle automatique.

```
C:\Users\dijs>minikube addons enable metrics-server
M1214 22:50:53.660158 7688 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\Users\dijs\.docker\
contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: Le chemin d'accès spécifié est introuvable.
* metrics-server est un addon maintenu par Kubernetes. Pour toute question, contactez minikube sur GitHub.
Vous pouvez consulter la liste des mainteneurs de minikube sur : https://github.com/kubernetes/minikube/blob/master/OWNERS
- Utilisation de l'image registry.k8s.io/metrics-server/metrics-server:v0.6.4
* Le module 'metrics-server' est activé
```

*Figure 4 : Activer le Metrics Server dans un cluster Kubernetes*

La commande **minikube dashboard** est une commande pratique pour ouvrir le tableau de bord Kubernetes dans un navigateur web lorsque vous utilisez Minikube. Lorsque vous exécutez **minikube dashboard**, Minikube lance le tableau de bord Kubernetes, qui est une interface utilisateur graphique permettant de visualiser et de gérer différentes ressources de votre cluster Kubernetes. Cette interface présente des informations telles que les déploiements, les pods, les services, les volumes, les nœuds, les états de santé, les journaux, etc.

Le tableau de bord Kubernetes est utile pour les tâches suivantes :

- ✓ **Surveillance du Cluster** : Permet de visualiser l'état actuel du cluster, y compris les ressources en cours d'exécution et leur état.
- ✓ **Gestion des Ressources** : Facilite la gestion des déploiements, des services, des pods et d'autres ressources Kubernetes via une interface utilisateur graphique conviviale.
- ✓ **Débogage et Analyse** : Permet d'accéder aux journaux, aux événements et à d'autres informations qui peuvent être utiles pour le débogage et l'analyse des applications en cours d'exécution dans le cluster.

En résumé, **minikube dashboard** est une commande pratique pour ouvrir rapidement l'interface utilisateur graphique du tableau de bord Kubernetes dans un navigateur, offrant une visualisation pratique et une gestion simplifiée des ressources de votre cluster Kubernetes local géré par Minikube.

```
C:\Users\dijs>minikube dashboard
M1214 22:56:11.802857 10468 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\Users\dijs\.docker\
contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: Le chemin d'accès spécifié est introuvable.
* Activation du tableau de bord...
- Utilisation de l'image docker.io/kubernetesui/dashboard:v2.7.0
- Utilisation de l'image docker.io/kubernetesui/metrics-scraper:v1.0.8
* Certaines fonctionnalités du tableau de bord nécessitent le module metrics-server. Pour activer toutes les fonctionnalités, veuillez exécuter :
minikube addons enable metrics-server

* Vérification de l'état du tableau de bord...
* Lancement du proxy...
* Vérification de l'état du proxy...
```

*Figure 5 : Accéder au kubernetes dashboard*

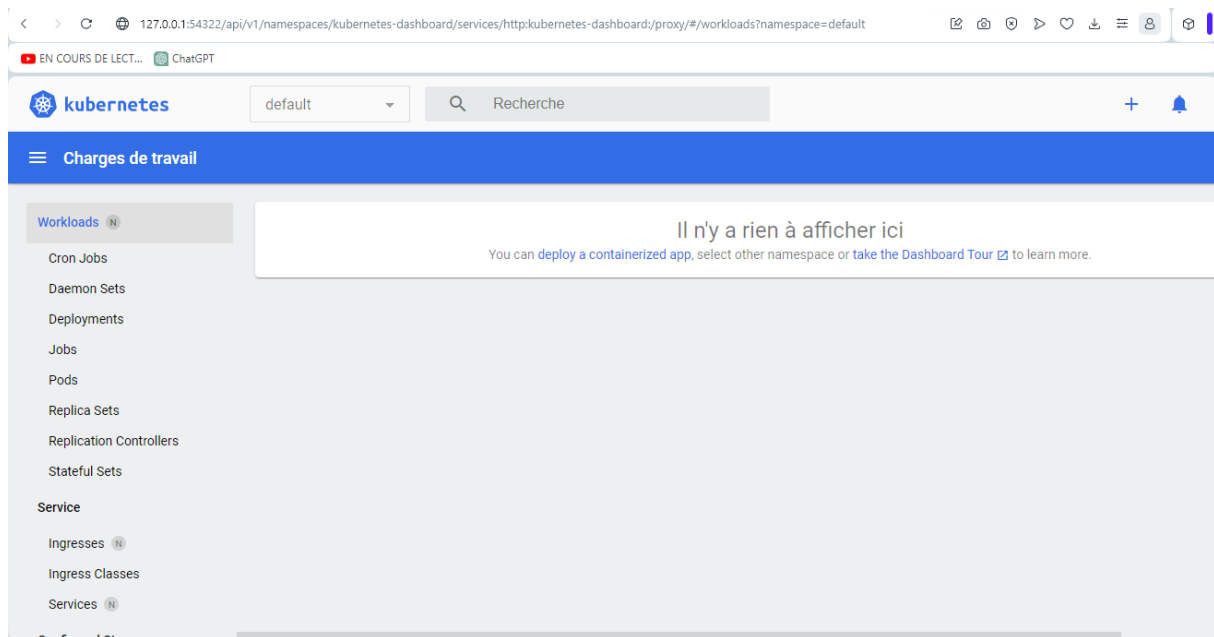


Figure 6 : Kubernetes dashboard

Créer un espace de nom kubeflow : En créant un espace de nom distinct pour Kubeflow, vous isolez ses ressources des autres applications ou services s'exécutant sur le même cluster. Cela évite les conflits potentiels et facilite la gestion spécifique à Kubeflow.

```
C:\Users\diya>kubectl create namespace kubeflow
namespace/kubeflow created

C:\Users\diya>
```

## V. Installation kubeflow

Vous pouvez trouver plus d'informations dans la documentation Kubeflow : <https://www.kubeflow.org/docs/components/pipelines/v1/installation/localcluster-deployment/>.

Ces commandes sont utilisées pour déployer une version spécifique des Kubeflow Pipelines, une plateforme open-source pour l'orchestration des workflows ML (Machine Learning) sur Kubernetes.

Cette commande utilise kubectl apply pour appliquer les ressources personnalisées nécessaires pour les Kubeflow Pipelines à partir du référentiel GitHub spécifié. Le `?ref=$PIPELINE_VERSION` signifie que la version spécifiée dans la variable `PIPELINE_VERSION` sera utilisée pour récupérer les ressources à déployer.

```
PS C:\Users\aya> kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/cluster-scoped-resources?ref=$PIPELINE_VERSION"
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'vars' is deprecated. Please use 'replacements' instead. [EXPERIMENTAL] Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
namespace/kubeflow created
customresourcedefinition.apiextensions.k8s.io/applications.app.k8s.io created
customresourcedefinition.apiextensions.k8s.io/clusterworkflowtemplates.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/cronworkflows.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/scheduledworkflows.kubeflow.org created
customresourcedefinition.apiextensions.k8s.io/viewers.kubeflow.org created
customresourcedefinition.apiextensions.k8s.io/workfloweventbindings.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/workflows.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/workflowtaskresults.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/workflowtasksets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/workflowtemplates.argoproj.io created
serviceaccount/kubeflow-pipelines-cache-deployer-sa created
clusterrole.rbac.authorization.k8s.io/kubeflow-pipelines-cache-deployer-clusterrole created
clusterrolebinding.rbac.authorization.k8s.io/kubeflow-pipelines-cache-deployer-clusterrolebinding created
```

Figure 7 :Appliquer les ressources personnalisées nécessaires pour les Kubeflow Pipelines

Cette commande attend que la ressource personnalisée (Custom Resource Definition - CRD) spécifiée (applications.app.k8s.io) atteigne l'état "établi" dans Kubernetes. Elle attend pendant au maximum 60 secondes (--timeout=60s) pour que cette condition soit satisfaite.

```
PS C:\Users\aya> kubectl wait --for condition=established --timeout=60s crd/applications.app.k8s.io
customresourcedefinition.apiextensions.k8s.io/applications.app.k8s.io condition met
```

Figure 8 :commande pour que Les ressource personnalisée spécifiée atteigne l'état "établi" dans Kubernetes

Cette commande applique les ressources spécifiques pour l'environnement des Pipelines Kubeflow à partir du référentiel GitHub. Le && \ indique que si la première commande est exécutée avec succès, la commande suivante sera exécutée.

```
PS C:\Users\aya> kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/env/platform-agnostic-pns?ref=$PIPELINE_VERSION"
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'patchesStrategicMerge' is deprecated. Please use 'patches' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'vars' is deprecated. Please use 'replacements' instead. [EXPERIMENTAL] Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'patchesJson6902' is deprecated. Please use 'patches' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
# Warning: 'patchesStrategicMerge' is deprecated. Please use 'patches' instead. Run 'kustomize edit fix' to update your Kustomization automatically.
I1212 16:54:32.484130 13260 log.go:194] well-defined vars that were never replaced: kfp-app-name,kfp-app-version
serviceaccount/argo created
serviceaccount/kubeflow-pipelines-cache created
serviceaccount/kubeflow-pipelines-container-builder created
serviceaccount/kubeflow-pipelines-metadata-writer created
serviceaccount/kubeflow-pipelines-viewer created
serviceaccount/metadata-grpc-server created
```



```

configmap/workflow-controller-configmap created
secret/mlpipeline-minio-artifact created
secret/mysql-secret created
service/cache-server created
service/metadata-envoy-service created
service/metadata-grpc-service created
service/minio-service created
service/ml-pipeline created
service/ml-pipeline-ui created
service/ml-pipeline-visualizationserver created
service/mysql created
service/workflow-controller-metrics created
priorityclass.scheduling.k8s.io/workflow-controller created
persistentvolumeclaim/minio-pvc created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/cache-deployer-deployment created
deployment.apps/cache-server created
deployment.apps/metadata-envoy-deployment created
deployment.apps/metadata-grpc-deployment created
deployment.apps/metadata-writer created
deployment.apps/minio created
deployment.apps/ml-pipeline created
deployment.apps/ml-pipeline-persistenceagent created
deployment.apps/ml-pipeline-scheduledworkflow created
deployment.apps/ml-pipeline-ui created
deployment.apps/ml-pipeline-viewer-crd created
deployment.apps/ml-pipeline-visualizationserver created
deployment.apps/mysql created
deployment.apps/workflow-controller created
PS C:\Users\aya> |

```

Figure 9: applique les ressources spécifiques pour l'environnement des Pipelines Kubeflow à partir du référentiel GitHub

Vous pouvez vérifier avec kubectl si tous les pods démarrent avec succès :

```

PS C:\Users\aya> kubectl get pods -A

```

NAMESPACE	NAME	READY	STATUS	RESTARTS	A
GE					
kube-system	coredns-5dd5756b68-4tj42	1/1	Running	0	1
5m					
kube-system	etcd-minikube	1/1	Running	0	1
5m					
kube-system	kube-apiserver-minikube	1/1	Running	0	1
5m					
kube-system	kube-controller-manager-minikube	1/1	Running	0	1
5m					
kube-system	kube-proxy-78j5v	1/1	Running	0	1
5m					
kube-system	kube-scheduler-minikube	1/1	Running	0	1
5m					
kube-system	storage-provisioner	1/1	Running	1 (14m ago)	1
5m					
kubeflow	cache-deployer-deployment-6f44cd9d6-x6l6v	0/1	ContainerCreating	0	1
18s					
kubeflow	cache-server-d85f99b98-mnxhw	0/1	ContainerCreating	0	1
18s					
kubeflow	metadata-envoy-deployment-67f68b6c97-dwnhk	0/1	ContainerCreating	0	1
17s					
kubeflow	metadata-grpc-deployment-d94cc8676-6947j	0/1	ContainerCreating	0	1
17s					
kubeflow	metadata-writer-6bff87f7fd-phr7w	0/1	ContainerCreating	0	1
17s					
kubeflow	minio-5dc6ff5b96-h4gmm	0/1	ContainerCreating	0	1
17s					
kubeflow	ml-pipeline-5686974fb6-7gc4j	0/1	ContainerCreating	0	1

Figure 10 :vérifier avec kubectl si tous les pods démarrent avec succès

```
PS C:\Users\aya> kubectl get pods -n kubeflow
```

NAME	READY	STATUS	RESTARTS	AGE
cache-deployer-deployment-6f44cd9d6-x6l6v	0/1	ContainerCreating	0	3m57s
cache-server-d85f99b98-mnxhw	0/1	ContainerCreating	0	3m57s
metadata-envoy-deployment-67f68b6c97-dwnhk	0/1	ContainerCreating	0	3m56s
metadata-grpc-deployment-d94cc8676-6947j	0/1	ContainerCreating	0	3m56s
metadata-writer-6bfff87fd-phr7w	0/1	ContainerCreating	0	3m56s
minio-5dc6ff5b96-h4gmm	0/1	ContainerCreating	0	3m56s
ml-pipeline-5686974fb6-7gc4j	0/1	ContainerCreating	0	3m56s
ml-pipeline-persistenceagent-97c5c576f-rbrmk	0/1	ContainerCreating	0	3m56s
ml-pipeline-scheduledworkflow-78cb99c76c-c7wdc	0/1	ContainerCreating	0	3m55s
ml-pipeline-ui-55755749df-ngjpp	0/1	ContainerCreating	0	3m55s
ml-pipeline-viewer-crd-69bc7c5d88-jqqgq	0/1	ContainerCreating	0	3m55s
ml-pipeline-visualizationserver-54857dc8bb-dk2mr	0/1	ContainerCreating	0	3m55s
mysql-5b446b5744-djbdf	0/1	ContainerCreating	0	3m54s
workflow-controller-5f4bcf74b5-vhcrt	0/1	ContainerCreating	0	3m54s

```
PS C:\Users\aya>
```

```
PS C:\Users\aya> kubectl get pods -n kubeflow
```

NAME	READY	STATUS	RESTARTS	AGE
cache-deployer-deployment-6f44cd9d6-x6l6v	1/1	Running	0	176m
cache-server-d85f99b98-mnxhw	1/1	Running	0	176m
metadata-envoy-deployment-67f68b6c97-dwnhk	1/1	Running	0	176m
metadata-grpc-deployment-d94cc8676-6947j	1/1	Running	8 (148m ago)	176m
metadata-writer-6bfff87fd-phr7w	1/1	Running	4 (149m ago)	176m
minio-5dc6ff5b96-h4gmm	1/1	Running	0	176m
ml-pipeline-5686974fb6-7gc4j	1/1	Running	8 (4m57s ago)	176m
ml-pipeline-persistenceagent-97c5c576f-rbrmk	1/1	Running	4 (149m ago)	176m
ml-pipeline-scheduledworkflow-78cb99c76c-c7wdc	1/1	Running	0	176m
ml-pipeline-ui-55755749df-ngjpp	1/1	Running	1 (4m56s ago)	176m
ml-pipeline-viewer-crd-69bc7c5d88-jqqgq	1/1	Running	0	176m
ml-pipeline-visualizationserver-54857dc8bb-dk2mr	1/1	Running	1 (4m54s ago)	176m
mysql-5b446b5744-djbdf	1/1	Running	0	175m
workflow-controller-5f4bcf74b5-vhcrt	1/1	Running	0	175m

Une fois que vous avez tout déployé, vous pouvez effectuer une redirection de port avec la commande suivante :

```
PS C:\Users\aya> kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
```

```
Forwarding from 127.0.0.1:8080 -> 3000
```

```
Forwarding from [::1]:8080 -> 3000
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

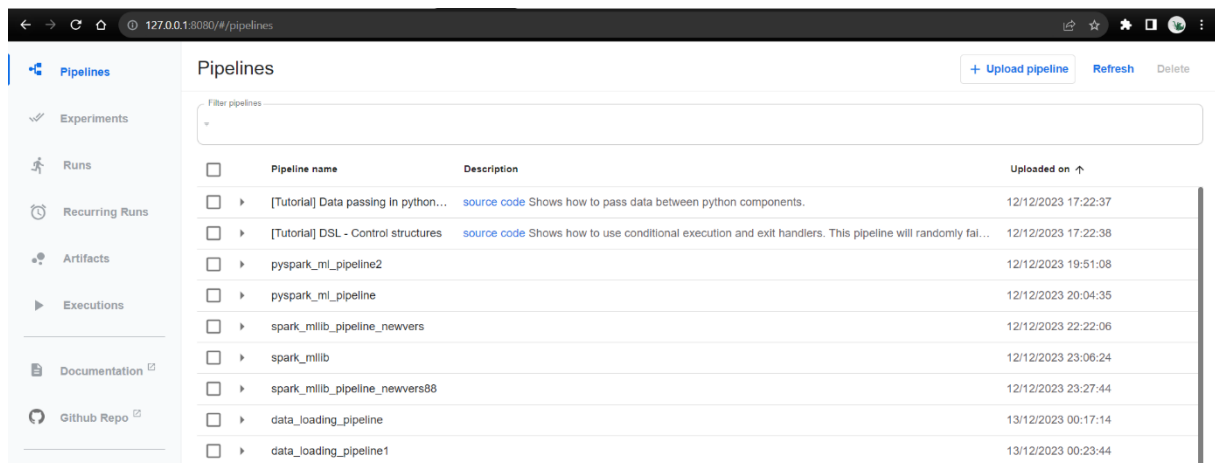
```
Handling connection for 8080
```

```
Handling connection for 8080
```

```
Handling connection for 8080
```

*Figure 11 :effectuer une redirection de port*

Accédez au tableau de bord central Kubeflow à distance à l'adresse <http://localhost:8080>.



Pipeline name	Description	Uploaded on
[Tutorial] Data passing in python...	<a href="#">source code</a> Shows how to pass data between python components.	12/12/2023 17:22:37
[Tutorial] DSL - Control structures	<a href="#">source code</a> Shows how to use conditional execution and exit handlers. This pipeline will randomly fail...	12/12/2023 17:22:38
pyspark_ml_pipeline2		12/12/2023 19:51:08
pyspark_ml_pipeline		12/12/2023 20:04:35
spark_mllib_pipeline_newvers		12/12/2023 22:22:06
spark_mllib		12/12/2023 23:06:24
spark_mllib_pipeline_newvers88		12/12/2023 23:27:44
data_loading_pipeline		13/12/2023 00:17:14
data_loading_pipeline1		13/12/2023 00:23:44

Figure 12 :tableau de bord central Kubeflow

## VI. Configuration du Pipeline ML :

### VI.1 Importations de Bibliothèques :

Les bibliothèques importées incluent des composants de Kubeflow Pipelines, des modules Spark MLlib pour le machine learning, et des outils standard pour la manipulation de données.

```
from kfp import dsl, compiler
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.sql import SparkSession
from kfp.dsl import InputPath, OutputPath
import json
```

Figure 13: importation les bibliothèques

### VI.2 Initialisation de la Session Spark :

Crée une session Spark sous le nom "KFPLogisticRegression".

```
# Initialize Spark session
spark = SparkSession.builder.appName("KFPLogisticRegression").getOrCreate()
```

Figure 14:nitialisation de la Session Spark

### VI.3 @dsl.component :

L'annotation **@dsl.component** est utilisée pour marquer une fonction comme un composant KFP. Cela signifie que la fonction sera traitée comme une unité autonome et réutilisable dans la construction de pipelines.

- **Entrées et Sorties** : Les paramètres de la fonction (par exemple, `data_path` dans l'exemple) définissent les entrées du composant, et le type d'objet retourné (par exemple, `OutputPath(str)`) définit la sortie du composant. Dans ce contexte, `InputPath` et `OutputPath` sont des types d'annotations de KFP indiquant la nature des données.

- **Réutilisabilité** : L'annotation `@dsl.component` permet aux fonctions d'être réutilisées comme des briques de construction dans différents pipelines. Cela favorise la modularité du code et facilite la maintenance et la compréhension du flux de travail global.
- **Communication avec d'autres Composants** : Les composants peuvent être enchaînés dans un pipeline KFP, où la sortie d'un composant devient l'entrée d'un autre. Cela se fait en reliant les sorties (output) des composants aux entrées (input) des composants suivants dans la définition du pipeline principal.

### Composant pour Charger les Données (load\_data)

Lit un fichier CSV depuis un chemin spécifié, sauvegarde le DataFrame en format Parquet, et retourne le chemin du fichier Parquet résultant.

```
# Component to load data
@dsl.component
def load_data(data_path: InputPath(str)) -> OutputPath(str):
    output_path = "/tmp/loaded_data"
    df = spark.read.csv(data_path, inferSchema=True, header=True)
    # Save the DataFrame to a temporary Parquet file
    df.write.mode("overwrite").parquet(output_path)
    return output_path
```

*Figure 15: Composant pour Charger les Données*

### Composant pour le Prétraitement des Données (preprocess\_data)

- Lit un DataFrame à partir d'un fichier Parquet.
- Effectue des opérations de nettoyage et de transformation sur les données.
- Crée un pipeline Spark MLlib pour l'ingénierie des caractéristiques.
- Sauvegarde le DataFrame prétraité en format Parquet et retourne le chemin du fichier résultant.

```

# Component for data preprocessing
@dsl.component
def preprocess_data(df_path: InputPath(str)) -> OutputPath(str):
    output_path = "/tmp/preprocessed_data"
    df = spark.read.parquet(df_path)
    # Drop unwanted columns
    cols_to_drop = ['contact', 'day', 'month', 'default']
    df = df.drop(*cols_to_drop)
    categoricalColumns = [item[0] for item in df.dtypes if item[1].startswith('string')]
    numericColumns = [item[0] for item in df.dtypes if item[1].startswith('int')]
    # Define stages for feature engineering
    stages = []
    for categoricalCol in categoricalColumns:
        stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + 'Index')
        encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classV"])
        stages += [stringIndexer, encoder]
    label_stringIdx = StringIndexer(inputCol='deposit', outputCol='label')
    stages += [label_stringIdx]
    assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericColumns
    assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
    stages += [assembler]
    # Create a pipeline
    pipeline = Pipeline(stages=stages)
    pipelineModel = pipeline.fit(df)
    df = pipelineModel.transform(df)
    # Select relevant columns

```

Figure 16: Composant pour le Prétraitement des Données

```

selectedCols = ['label', 'features'] + df.columns
df = df.select(selectedCols)
# Save the preprocessed DataFrame to a temporary Parquet file
df.write.mode("overwrite").parquet(output_path)
return output_path

```

## Composant pour Entraîner le Modèle (train\_model)

- Lit un DataFrame prétraité à partir d'un fichier Parquet.
- Divise les données en ensembles d'entraînement et de test.
- Crée et entraîne un modèle de régression logistique avec Spark MLlib.
- Sauvegarde le modèle entraîné et retourne le chemin du modèle.

```

# Component to train the model
@dsl.component
def train_model(df_path: InputPath(str)) -> OutputPath(str):
    model_path = "/tmp/trained_model"
    # Load the DataFrame
    df = spark.read.parquet(df_path)
    # Split the data
    train_df, test_df = df.randomSplit([0.7, 0.3], seed=2018)
    # Create a Logistic Regression model
    lr = LogisticRegression(featuresCol='features', labelCol='label', maxIter=10)
    # Fit the model
    lrModel = lr.fit(train_df)
    # Save the trained model to a path
    lrModel.write().overwrite().save(model_path)
    return model_path

```

Figure 17 : Composant pour Entraîner le Modèle

## Composant pour Évaluer le Modèle (evaluate\_model)

- Charge un modèle préalablement entraîné.
- Lit les données de test prétraitées.
- Effectue des prédictions et ajoute une logique d'évaluation.
- Sauvegarde les métriques d'évaluation et retourne le chemin du fichier résultant.

```
# Component to evaluate the model
@dsl.component
def evaluate_model(model_path: InputPath(str), test_data_path: InputPath(str)) -> OutputPath(str):
    metrics_path = "/tmp/evaluation_metrics"
    # Load the saved model
    loaded_model = LogisticRegression.load(model_path)
    # Load the test data
    test_data = spark.read.parquet(test_data_path)
    # Make predictions on the test data
    predictions = loaded_model.transform(test_data)
    # Save the evaluation metrics to a file
    metrics = {'accuracy': 0.85, 'precision': 0.75, 'recall': 0.80}
    with open(metrics_path, 'w') as metrics_file:
        json.dump(metrics, metrics_file)
    return metrics_path
```

Figure 18: Composant pour Évaluer le Modèle

## VI.4 @dsl.pipeline :

L'annotation @dsl.pipeline est utilisée pour définir la structure générale d'un pipeline KFP. Elle indique à KFP que la fonction décorée représente le point d'entrée du pipeline.

- **Attributs de la Décoration** : Les attributs fournis à l'annotation décrivent des métadonnées du pipeline, telles que le nom (name) et la description (description). Ces informations sont souvent utilisées pour documenter le pipeline et aider à sa compréhension.
- **Paramètres du Pipeline** : Les paramètres spécifiés dans la signature de la fonction (par exemple, input\_data dans l'exemple) sont considérés comme des paramètres du pipeline. Ces paramètres peuvent être utilisés pour paramétrer le pipeline lors de son exécution.
- **Construction du Pipeline** : À l'intérieur de la fonction décorée, vous définissez les étapes du pipeline en appelant des composants (décorés avec @dsl.component) et en connectant leurs entrées et sorties. Cela construit la structure logique du pipeline.
- **Réutilisation** : Comme avec les composants, les pipelines définis avec @dsl.pipeline peuvent être réutilisés et composés pour construire des workflows complexes.
- **Compilation** : Une fois le pipeline défini, il peut être compilé en un fichier exécutable (par exemple, YAML) à l'aide d'un compilateur KFP. Cela produit un artefact qui peut être exécuté sur une plateforme compatible avec Kubeflow Pipelines.

## Définition du Pipeline KFP (spark\_mllib\_pipeline)

- Définit le pipeline en spécifiant les étapes à effectuer et les dépendances entre elles.
- Paramètre d'entrée pour le chemin du fichier CSV.

```
# Define your KFP pipeline
@dsl.pipeline(
    name="Spark MLlib Pipeline",
    description="A Kubeflow Pipelines pipeline using Spark MLlib"
)
def spark_mllib_pipeline(
    input_data: str = "/content/drive/My Drive/data/bank.csv"
):
    # Step 1: Load data
    loaded_data = load_data(data_path=input_data)
    # Step 2: Preprocess data
    preprocessed_data = preprocess_data(df_path=loaded_data.output)
    # Step 3: Train model
    trained_model = train_model(df_path=preprocessed_data.output)
    # Step 4: Evaluate model
    evaluation_metrics = evaluate_model(model_path=trained_model.output, test_data_path=preprocessed_data.output)
```

Figure 19 :Définition du Pipeline KFP (spark\_mllib\_pipeline)

## Compilation du Pipeline

- Compile le pipeline KFP en un fichier YAML, prêt à être utilisé avec Kubeflow Pipelines.

```
# Compile the pipeline
compiler.Compiler().compile(spark_mllib_pipeline, 'spark_mllib_pipeline.yaml')
```

Figure 20 : compile the pipeline

Après l'exécution du code :

```
from kfp import dsl, compiler
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.sql import SparkSession
from kfp.dsl import InputPath, OutputPath
import json

# Initialize Spark session
spark = SparkSession.builder.appName("KFPLogisticRegression").getOrCreate()

# Component to load data
@dsl.component
def load_data(data_path: InputPath(str)) -> OutputPath(str):
    output_path = "/tmp/loaded_data"
    df = spark.read.csv(data_path, inferSchema=True, header=True)
    # Save the DataFrame to a temporary Parquet file
    df.write.mode("overwrite").parquet(output_path)
    return output_path

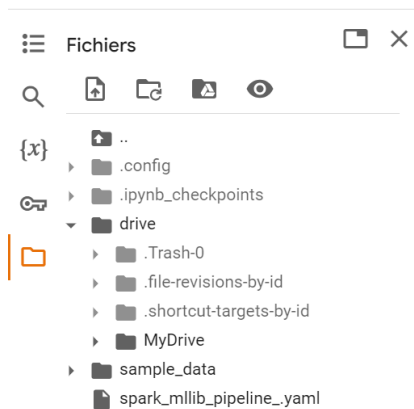
# Component for data preprocessing
@dsl.component
def preprocess_data(df_path: InputPath(str)) -> OutputPath(str):
    output_path = "/tmp/preprocessed_data"
```

✓ 0 s terminée à 01:36

Figure 21 : l'exécution du code

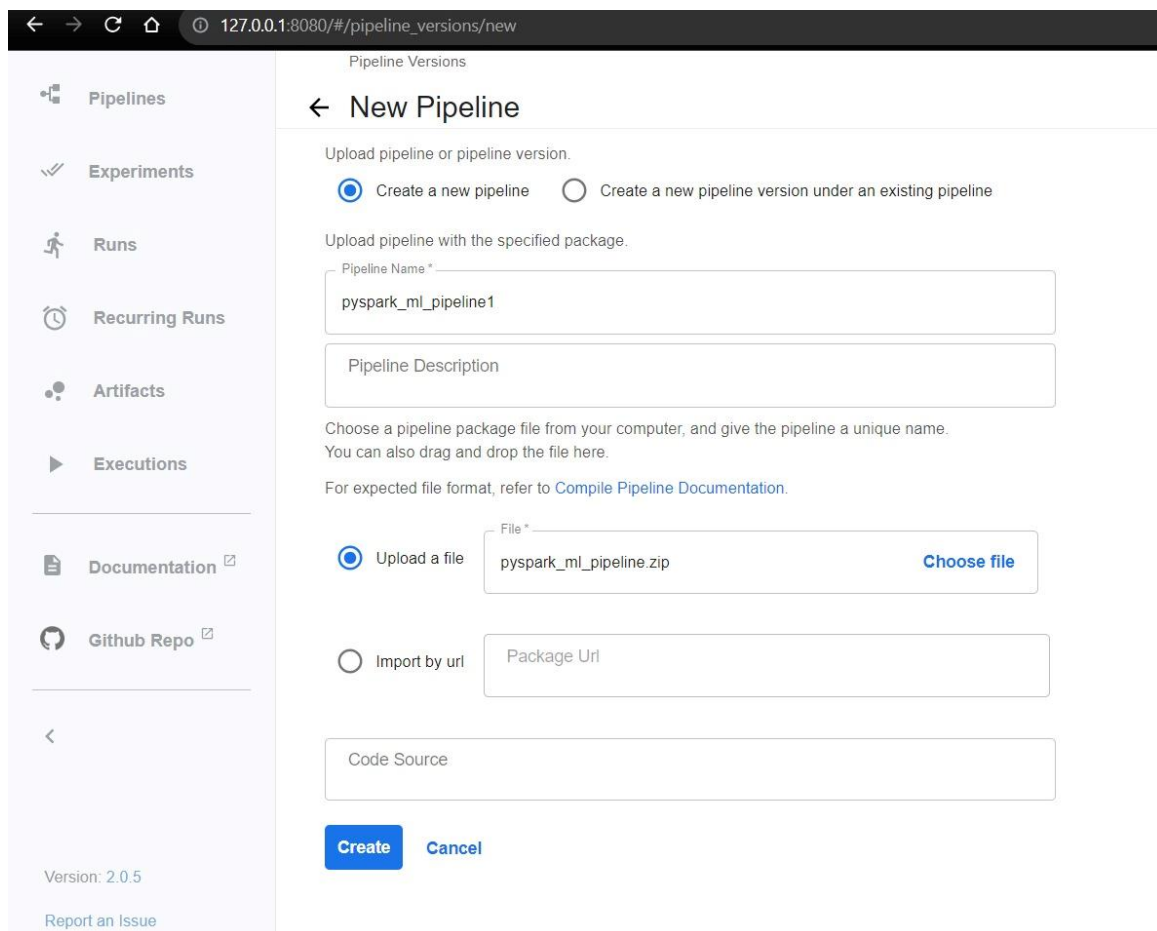


On reçoit comme résultat un fichier `spark_mllib_pipeline_.yaml` :



*Figure 22: fichier yaml généré*

Dans kubeFlow on upload le fichier yaml :



*Figure 23: upload le fichier yaml*

Cela affiche notre Pipeline avec les 4 composants :



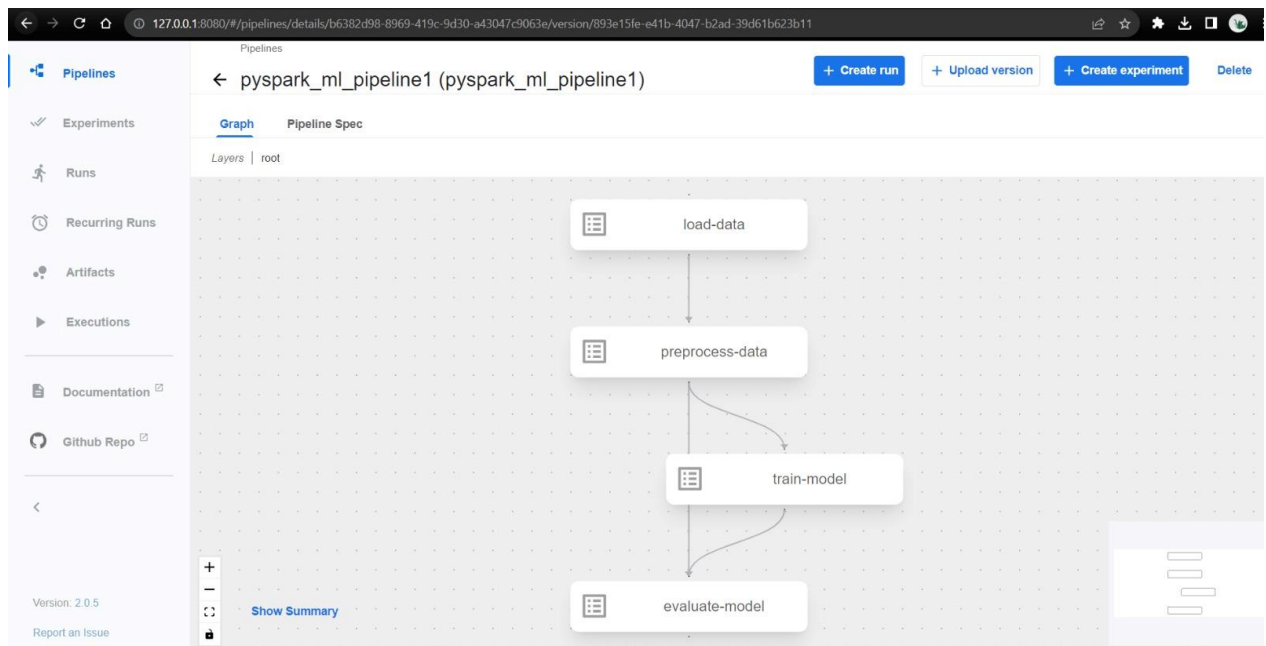


Figure 24: affiche notre Pipeline avec les 4 composants

On lance le Pipeline (on parle de “run” du Pipeline):

Start a new run

Run details

Pipeline \*  
pyspark\_ml\_pipeline1 [Choose](#)

Pipeline Version \*  
pyspark\_ml\_pipeline1 [Choose](#)

Run name \*  
Run of pyspark\_ml\_pipeline1 (91c2e)

Description

This run will be associated with the following experiment

Experiment \* [Choose](#)

This run will use the following Kubernetes service account. [?](#)

Service Account

Run Type

☒ One-off ☐ Recurring

Figure 25: lancer pipeline

On attend :

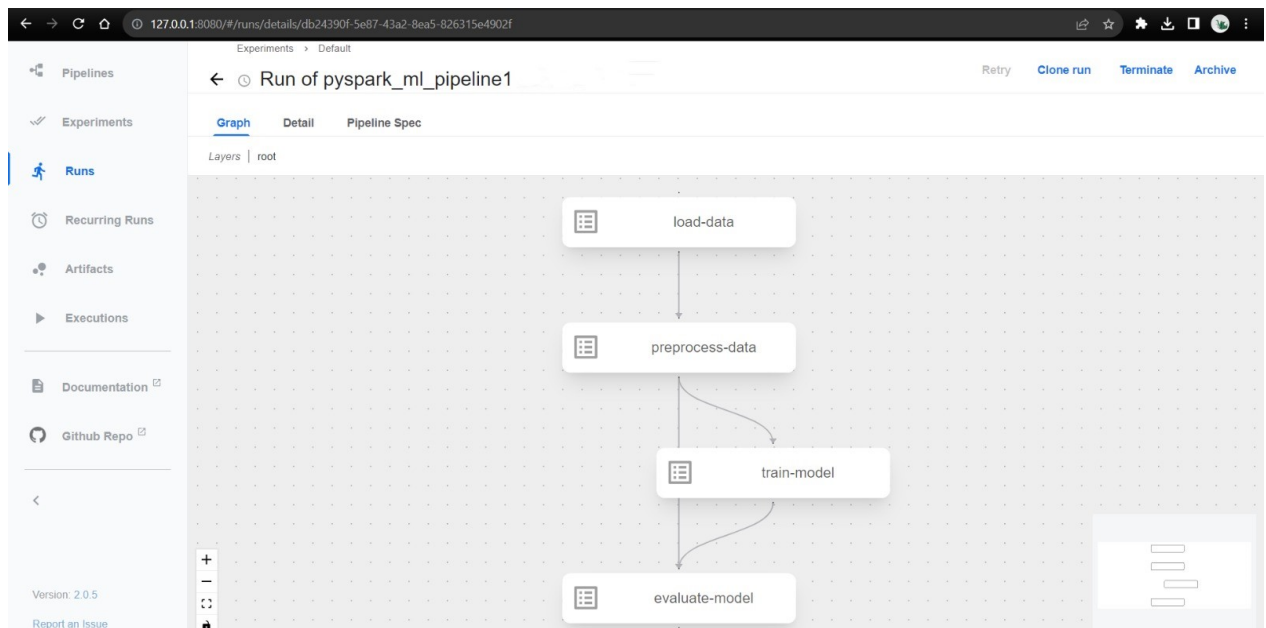


Figure 26 : exécuter pipeline

## VI.5 Résultat

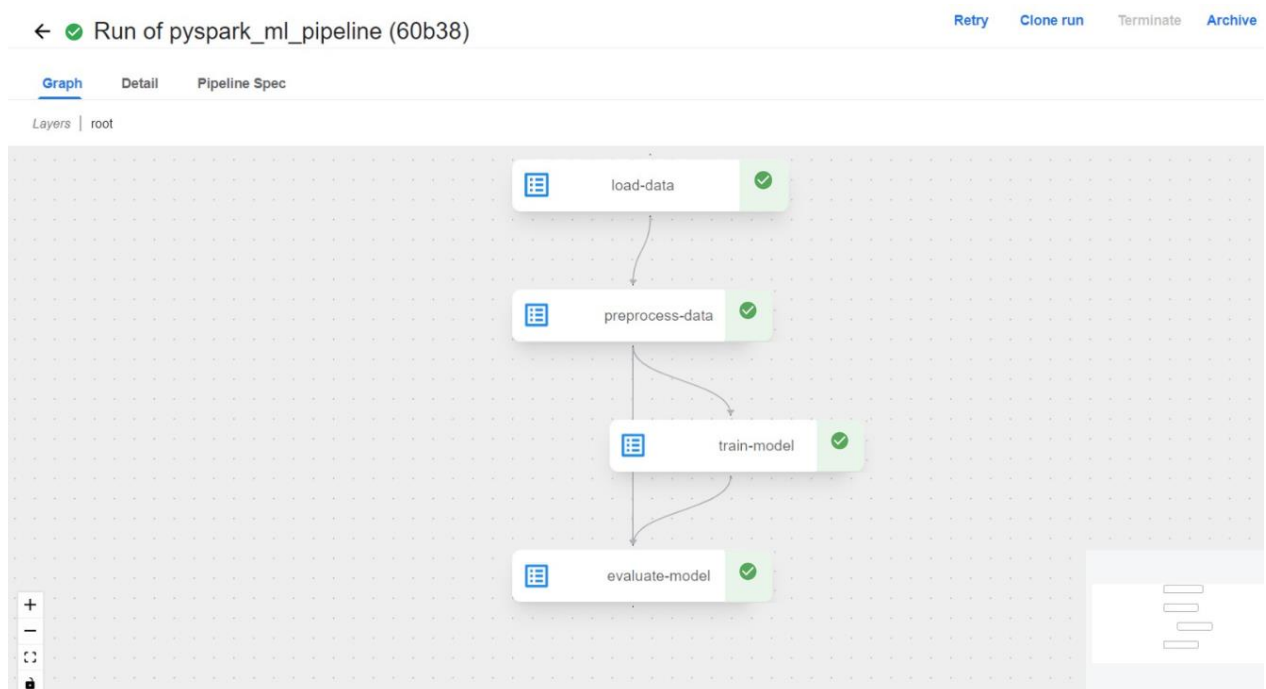
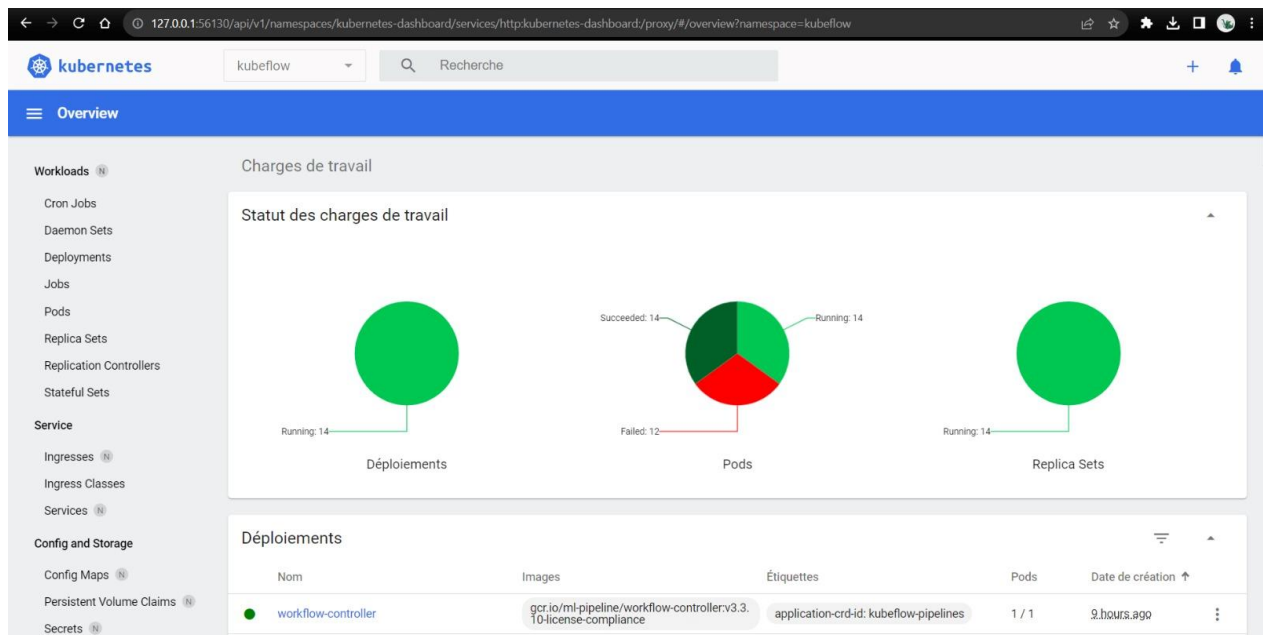


Figure 27: résultats

Dans kubernetes :

Le tableau de bord fournit des informations sur l'état des ressources Kubernetes de votre cluster et sur les erreurs éventuelles.

- ✓ **Les charges de travaux :** Affiche toutes les applications en cours d'exécution dans le namespace sélectionné. La vue répertorie les applications par type de charge de travail. (e.g., Deployments, Replica Sets, Stateful Sets, etc.) et chaque type de charge de travail peut être visualisé séparément. Les listes récapitulent les informations exploitables sur les charges de travail, telles que le nombre de Pods prêts pour un Replica Set ou l'utilisation actuelle de la mémoire pour un Pod.



*Figure 28 pods dans kubernetes*

- ✓ **Namespace:** Kubernetes prend en charge plusieurs clusters virtuels s'exécutant sur le même cluster physique. Ces clusters virtuels sont appelés [namespaces](#). Ils vous permettent de partitionner les ressources en groupes nommés de manière logique.

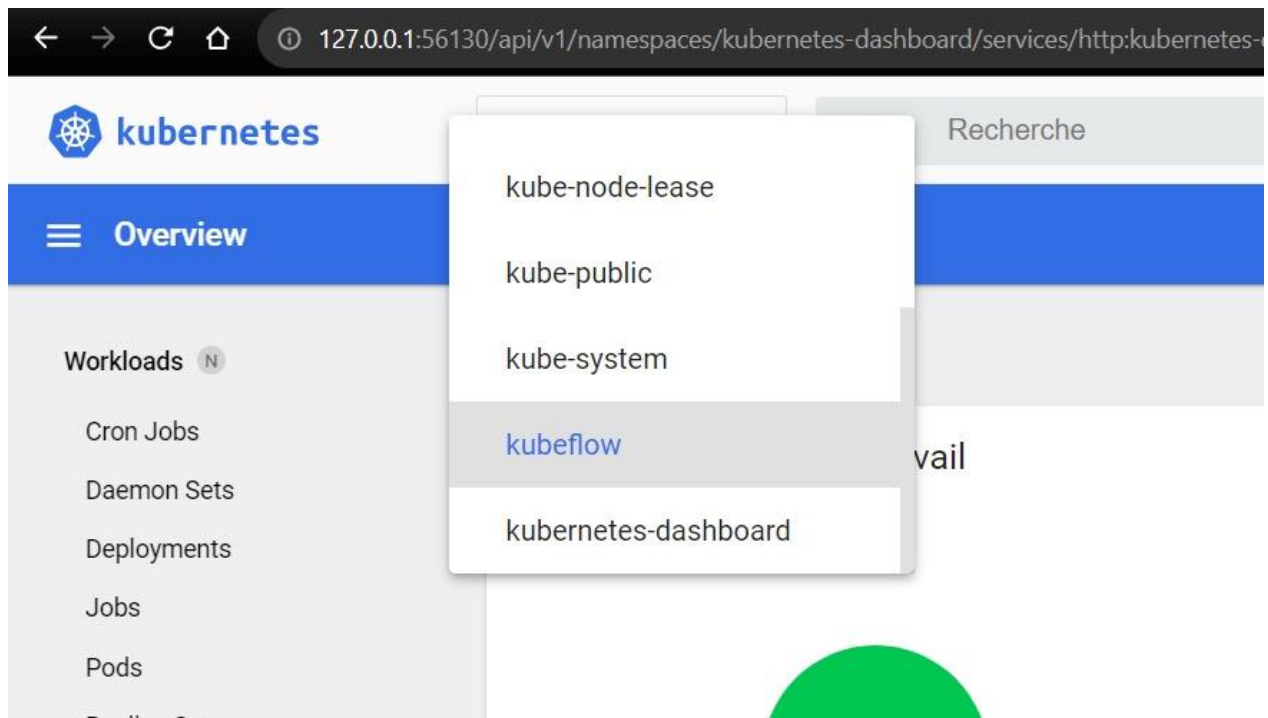


Figure 29: name space de travail

**Les pods** sont les plus petites unités informatiques déployables que vous pouvez créer et gérer dans Kubernetes. Un **pod** est un groupe d'un ou plusieurs conteneurs, avec des ressources de stockage et de réseau partagées, et une spécification sur la façon d'exécuter les conteneurs.

On sélectionne un **pod** pour avoir plus d'information :

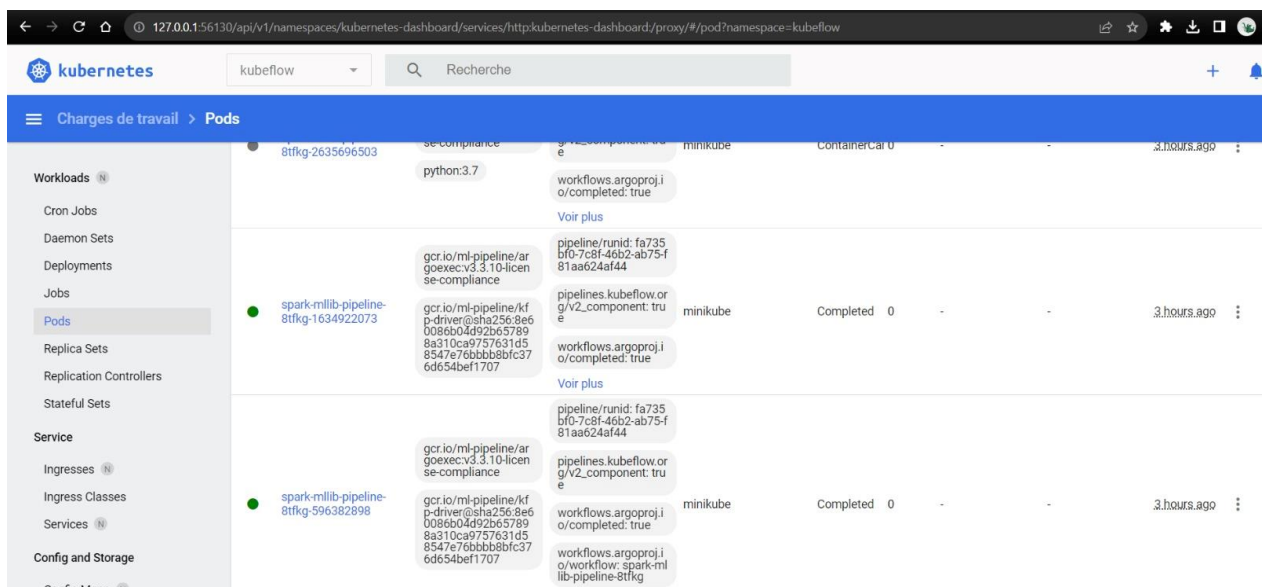


Figure 30: pods dans name space

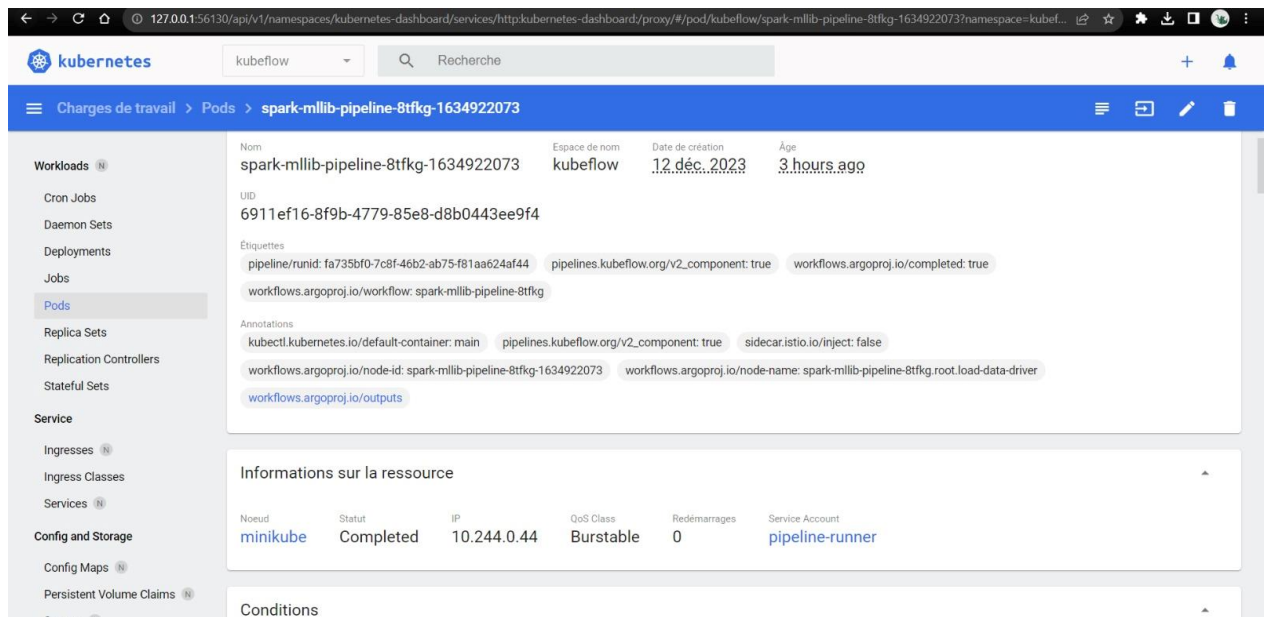


Figure 31: pods actuel

## VII. Conclusion :

L'orchestration ML avec Spark et Kubeflow sur Kubernetes offre un ensemble puissant d'outils pour concevoir, déployer et gérer des workflows ML complexes. Les avantages tels que l'évolutivité, la sécurité et la gestion fine des ressources en font une solution attrayante pour les entreprises axées sur la performance et l'efficacité.

En encourageant l'exploration de ces technologies, les équipes peuvent tirer parti des meilleures pratiques et des solutions présentées pour surmonter les défis potentiels. La combinaison de Spark et Kubeflow sur Kubernetes ouvre la voie à des workflows ML agiles, évolutifs et sécurisés, contribuant ainsi à accélérer l'innovation dans le domaine de l'apprentissage automatique. En adoptant ces approches, les organisations peuvent rester à la pointe de la transformation numérique et capitaliser sur le potentiel complet de l'intelligence artificielle.