Build an ML Pipeline (Part 2) — Model Registration and Serving with MLflow and KServe

Seamless Model Deployment: MLflow and KServe Collaboration

Vinayak Shanawad · Following

10 min read · Dec 4, 2023

🎯Goal

In my previous post, we looked at setting up the minikube cluster, installing Kubeflow pipelines, and creating the Kubeflow pipeline using the popular data science IRIS dataset.

Let's continue with the MLflow setup, register the model using the MLflow model registry, and serve the model using KServe.

Photo by Darya Jum on Unsplash

🤔 What is MLflow?

MLflow is an open-source platform for managing the end-to-end ML lifecycle. It provides a range of features to help data scientists and ML engineers streamline their workflow and collaborate effectively.

Some key features of MLflow include

1. **Hyperparameter Tuning:** MLflow integrates with popular hyperparameter optimization libraries like Hyperopt and Optuna, making it easy to perform hyperparameter tuning and track results.

2. **Experiment Tracking:** MLflow allows users to track experiments, including parameters, metrics, and artifacts (model files and other output). This helps with comparing and reproducing different runs.

3. **Model Versioning:** MLflow offers versioning for models, making it easy to track and manage different iterations of a model. This helps ensure reproducibility and simplifies model management.

4. **Model Registry:** The Model Registry in MLflow allows you to organize and manage model versions, including staging, transitioning, and rolling back models. It adds a layer of control to the model deployment process.

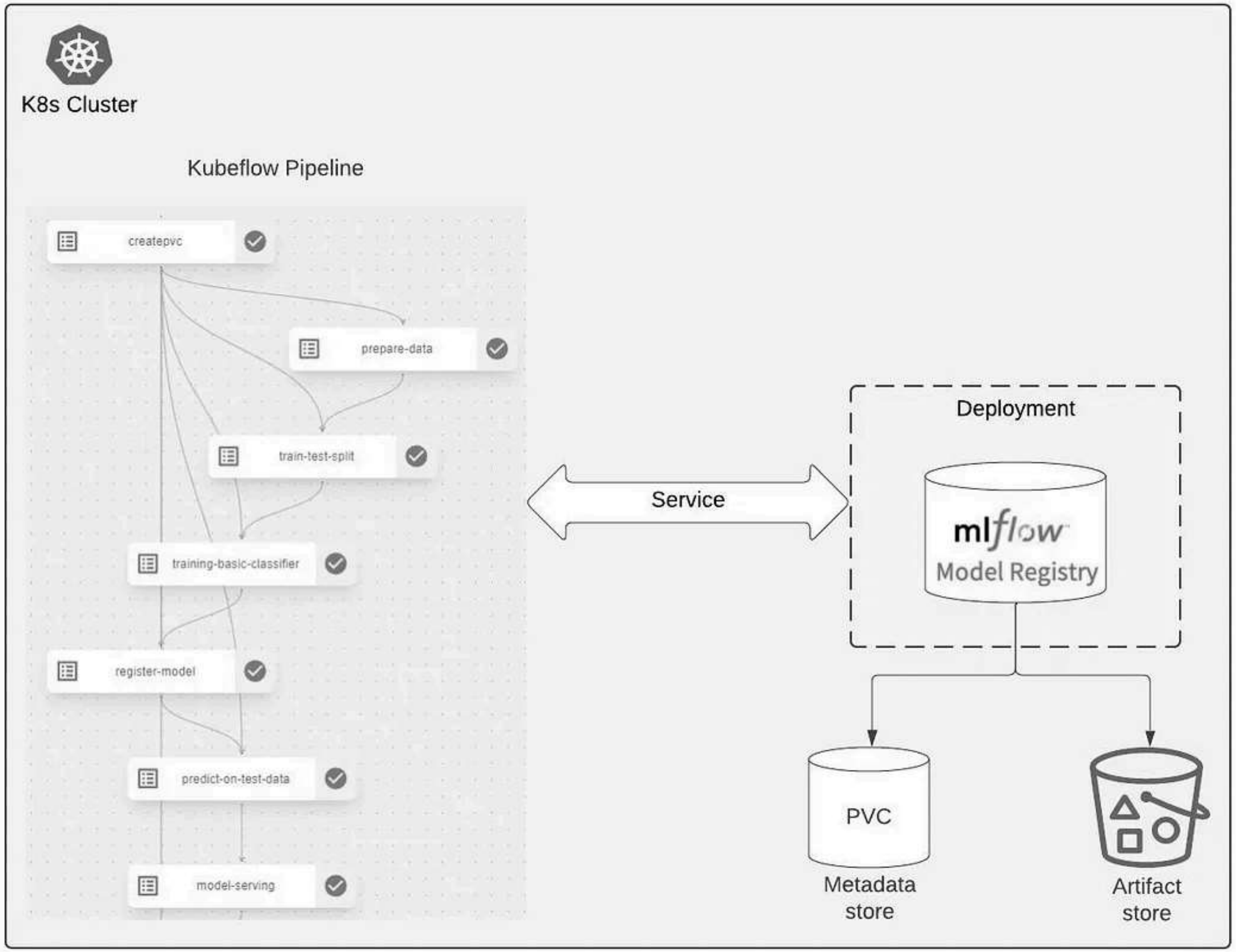
5. **Artifact Store:** MLflow supports various artifact stores, including local file systems, cloud storage (e.g., Amazon S3), and database-backed stores, for managing large model files and other artifacts.

6. **Model Packaging and Deployment:** MLflow provides tools for packaging and deploying machine learning models as REST API endpoints, Docker containers, or other target environments. This simplifies the process of putting models into production.

7. Automatic Logging: MLflow can automatically log machine learning framework information, code versions, and environment details during each run. This ensures comprehensive tracking and reproducibility.

🛠️MLflow Setup

Let’s install MLflow tracking server on Minikube cluster and understand how Kubeflow pipelines can interact with MLflow tracking server.



MLflow Setup on Minikube (Image by Author)

Prerequisites

We need to create the service account that will be used in model deployment using KServe and secret that will be used in Kubeflow pipeline steps to register the models and load the model artifacts from S3, and MLflow tracking server to populate the model artifact details from s3 on MLflow UI.

Define the K8s manifest file to create a service account and secret.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: kubeflow
  name: mlflow-sa
  annotations:
    eks.amazonaws.com/role-arn: <role-arn-for-aws-s3-access>
    serving.kserve.io/s3-endpoint: s3.<region>.amazonaws.com
    serving.kserve.io/s3-usehttps: "1"
    serving.kserve.io/s3-region: "<region>"
    serving.kserve.io/s3-useanoncredential: "false"

---

apiVersion: v1
kind: Secret
metadata:
  namespace: kubeflow
  name: aws-credentials
  annotations:
     serving.kserve.io/s3-endpoint: s3.<region>.amazonaws.com
     serving.kserve.io/s3-usehttps: "1"
     serving.kserve.io/s3-region: "<region>"
     serving.kserve.io/s3-useanoncredential: "false"
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: <aws-access-key>
  AWS_SECRET_ACCESS_KEY: <aws-secret-key>
  AWS_DEFAULT_REGION: <region>

---

apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  namespace: kubeflow
  name: mlflow-sa
secrets:
- name: aws-credentials
```

🚀 MLflow deployment

Let's create a Docker image that builds the MLflow tracking server then build and push into Dockerhub. (You can use my docker image from Dockerhub)

```
# Defining base image
FROM python:3.8.2-slim

# Installing MLflow from PyPi
RUN pip install mlflow

# Defining start-up command
EXPOSE 5000
ENTRYPOINT ["mlflow", "server", "--host", "0.0.0.0", "--port", "5000"]
```

Define the K8s manifest file to create a simple PVC (`mlflow-pvc.yaml`) for the storage (size: 10 MB) of the metadata. But the cool thing is that MLflow supports lots of different options for these, including cloud services. We can use MySQL, PostgreSQL, etc as Metadata store. I am using S3 bucket on AWS as artifact store that contains all our model artifacts.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  namespace: kubeflow
  name: mlflow-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Mi
```

Let’s create PVC using the following command.

```
kubectl apply -f mlflow-pvc.yaml
```

Let’s define another K8s manifest file (`mlflow.yaml`) for MLflow setup on Minikube.

```yaml
# Creating MLflow deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: kubeflow
  name: mlflowserver
```

```yaml
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mlflowserver
  template:
    metadata:
      labels:
        app: mlflowserver
    spec:
      volumes:
        - name: mlflow-pvc
          persistentVolumeClaim:
            claimName: mlflow-pvc
      containers:
      - name: mlflowserver
        image: vinayaks117/mlflow-repo:v2.0
        imagePullPolicy: Always
        args:
        - --host=0.0.0.0
        - --port=5000
        - --backend-store-uri=/opt/mlflow/backend
        - --default-artifact-root=s3://kubeflow-mlflow/experiments
        - --workers=2
        env:
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: AWS_ACCESS_KEY_ID
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: AWS_SECRET_ACCESS_KEY
        - name: AWS_DEFAULT_REGION
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: AWS_DEFAULT_REGION
```

```
        ports:
        - name: http
          containerPort: 5000
          protocol: TCP
        volumeMounts:
        - name: mlflow-pvc
          mountPath: /opt/mlflow/backend
---
apiVersion: v1
kind: Service
metadata:
  namespace: kubeflow
  name: mlflowserver
spec:
  selector:
    app: mlflowserver
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 5000
```

The syntax here will seem quite familiar to you; just pay attention to the arguments we'll use when building our Docker image.

While you're likely familiar with the "host" and "port" arguments, the latter two might be new. They specify where MLflow should log our model metadata for the model registry and where to log the model artifacts. In this setup, I'm utilizing a simple Persistent Volume Claim (PVC).

`--backend-store-uri=/opt/mlflow/backend` is used to store the metadata (model parameters, evaluation metrics, etc) in PVC.

`--default-artifact-root=s3://kubeflow-mlflow/experiments` is used to store the artifacts (model artifacts) in S3.

> *Note: We are configuring AWS credentials in environment variables because we would like to populate the registered model artifacts from S3 in MLflow UI.*

K8s Service

Let's create an internal service so that we can interact with the MLflow tracking server via internal service from Kubeflow pipelines.

Let's create an MLflow deployment and service using the following command.

```
kubectl apply -f mlflow.yaml
```

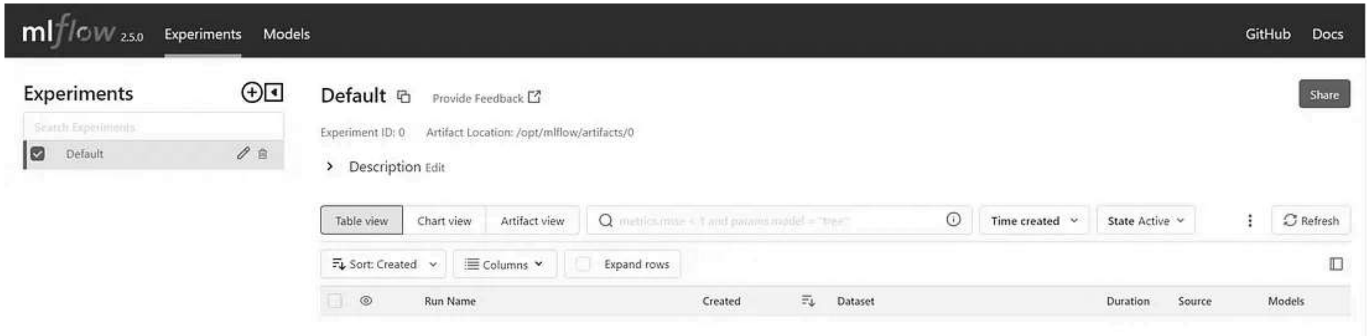
```
NAME                                              READY   STATUS    RESTARTS        AGE
cache-deployer-deployment-64dc947fc7-97gfx        1/1     Running   1 (4h34m ago)   9d
cache-server-7f7d6bfb55-m9cxj                     1/1     Running   1 (4h34m ago)   9d
controller-manager-dfbd6b98-xd7kk                 1/1     Running   2 (4h33m ago)   9d
metadata-envoy-deployment-6dcd4ddcb8-zbxgt        1/1     Running   1 (4h34m ago)   9d
metadata-grpc-deployment-5644fb9768-7t7nr         1/1     Running   9 (4h33m ago)   9d
metadata-writer-9c4488669-drcw7                   1/1     Running   6 (4h34m ago)   9d
minio-55464b6ddb-9ksjn                            1/1     Running   1 (4h34m ago)   9d
ml-pipeline-65cc95bd6b-6lrcz                      1/1     Running   2 (4h33m ago)   9d
ml-pipeline-persistenceagent-545d5c6786-mxfl8     1/1     Running   4 (4h34m ago)   9d
ml-pipeline-scheduledworkflow-8f9b7654d-jhw8l     1/1     Running   1 (4h34m ago)   9d
ml-pipeline-ui-7c4cf85598-gnmf9                   1/1     Running   1 (4h34m ago)   9d
ml-pipeline-viewer-crd-589c6c6569-4s7wt           1/1     Running   1 (4h34m ago)   9d
ml-pipeline-visualizationserver-9fcfbd447-f69qx   1/1     Running   1 (4h34m ago)   9d
mlflowserver-677f5d667-7nhqh                      1/1     Running   0               174m
mysql-7d8b8ff4f4-xkjtk                            1/1     Running   1 (4h34m ago)   9d
```

MLflow installation status (Image by Author)

Let's verify that the MLflow dashboard is accessible by port-forwarding:

```
kubectl port-forward -n kubeflow svc/mlflowserver 8081:5000
```

Then, open the MLflow dashboard at http://localhost:8081/



MLflow dashboard (Image by Author)

🛠 KServe Setup

Please refer to my previous article where I showed how to setup KServe on Minikube.

Let’s look at the status of all Kubeflow pipeline, MLflow server, and KServe components from K8s cluster using `kubectl get pods -A` command.

```
NAMESPACE          NAME                                               READY   STATUS             RESTARTS        AGE
cert-manager       cert-manager-6d5bcc69b8-l4sv6                      1/1     Running            0               6m46s
cert-manager       cert-manager-cainjector-7d9d487b4c-j46zs           1/1     Running            0               6m46s
cert-manager       cert-manager-webhook-8569c487cd-dvj68              1/1     Running            0               6m46s
istio-system       istio-ingressgateway-655d674f87-wmfl8              1/1     Running            0               7m18s
istio-system       istiod-8d866bf74-8mkqd                             1/1     Running            0               7m43s
knative-serving    activator-558c6cd86f-kkph8                         1/1     Running            0               6m55s
knative-serving    autoscaler-74d5fff59f-7b2sc                        1/1     Running            0               6m55s
knative-serving    controller-67c76797fd-hqkgw                        1/1     Running            0               6m55s
knative-serving    net-istio-controller-84cb8b59fb-dr4t6              1/1     Running            0               6m52s
knative-serving    net-istio-webhook-8d785b78d-zxl74                  1/1     Running            0               6m52s
knative-serving    webhook-676958d654-kvgbx                           1/1     Running            0               6m55s
kserve             kserve-controller-manager-78c666cf85-x4w8j         2/2     Running            0               5m41s
kube-system        coredns-5d78c9869d-vz8dl                           1/1     Running            0               20m
kube-system        etcd-minikube                                      1/1     Running            0               20m
kube-system        kube-apiserver-minikube                            1/1     Running            0               21m
kube-system        kube-controller-manager-minikube                   1/1     Running            0               20m
kube-system        kube-proxy-t7hl7                                   1/1     Running            0               20m
kube-system        kube-scheduler-minikube                            1/1     Running            0               20m
kube-system        storage-provisioner                                1/1     Running            1 (20m ago)     20m
kubeflow           cache-deployer-deployment-64dc947fc7-fl6mq         1/1     Running            0               20m
kubeflow           cache-server-7f7d6bfb55-ng79h                      1/1     Running            0               20m
kubeflow           controller-manager-dfbd6b98-wcccn                  1/1     Running            0               20m
kubeflow           metadata-envoy-deployment-6dcd4ddcb8-2fmpk         1/1     Running            0               20m
kubeflow           metadata-grpc-deployment-5644fb9768-c4458          1/1     Running            7 (12m ago)     20m
kubeflow           metadata-writer-9c4488669-v8pgh                    1/1     Running            2 (9m57s ago)   20m
kubeflow           minio-55464b6ddb-rzxtt                             1/1     Running            0               20m
kubeflow           ml-pipeline-65cc95bd6b-pttx2                       1/1     Running            0               17m
kubeflow           ml-pipeline-persistenceagent-545d5c6786-w7rkn      1/1     Running            1 (12m ago)     20m
kubeflow           ml-pipeline-scheduledworkflow-8f9b7654d-vrchf      1/1     Running            0               20m
kubeflow           ml-pipeline-ui-7c4cf85598-xcpp4                    1/1     Running            0               20m
kubeflow           ml-pipeline-viewer-crd-589c6c6569-25b7t            1/1     Running            0               20m
kubeflow           ml-pipeline-visualizationserver-9fcfbd447-jcwdg    1/1     Running            0               20m
kubeflow           mlflowserver-556855dd6-625b6                       1/1     Running            0               2m5s
kubeflow           mysql-7d8b8ff4f4-wgd7c                             1/1     Running            0               20m
kubeflow           proxy-agent-8dc6b5fd8-f9x6s                        0/1     CrashLoopBackOff   6 (3m48s ago)   20m
kubeflow           workflow-controller-589ff7c479-qqh4b               1/1     Running            0               20m
```

Minikube Cluster Staus (Image by Author)

Let's go through the remaining steps (Model provisioning, Model evaluation, and Model deployment) from the ML pipeline as we discussed in my previous post.

5. Register a model in the MLflow model registry — AWS S3

```
@component(
    packages_to_install=["pandas", "numpy", "scikit-learn", "mlflow", "boto3"],
    base_image="python:3.9",
)
def register_model(data_path: str, aws_access_key_id: str, aws_secret_access_key
    import pandas as pd
    import numpy as np
    import pickle
    import os
    import mlflow
    from mlflow.models import infer_signature
    from sklearn import datasets

    with open(f'{data_path}/model.pkl','rb') as f:
        logistic_reg_model = pickle.load(f)

    # Infer the model signature
    X_test = np.load(f'{data_path}/X_test.npy', allow_pickle=True)
    y_pred = logistic_reg_model.predict(X_test)
    signature = infer_signature(X_test, y_pred)
```

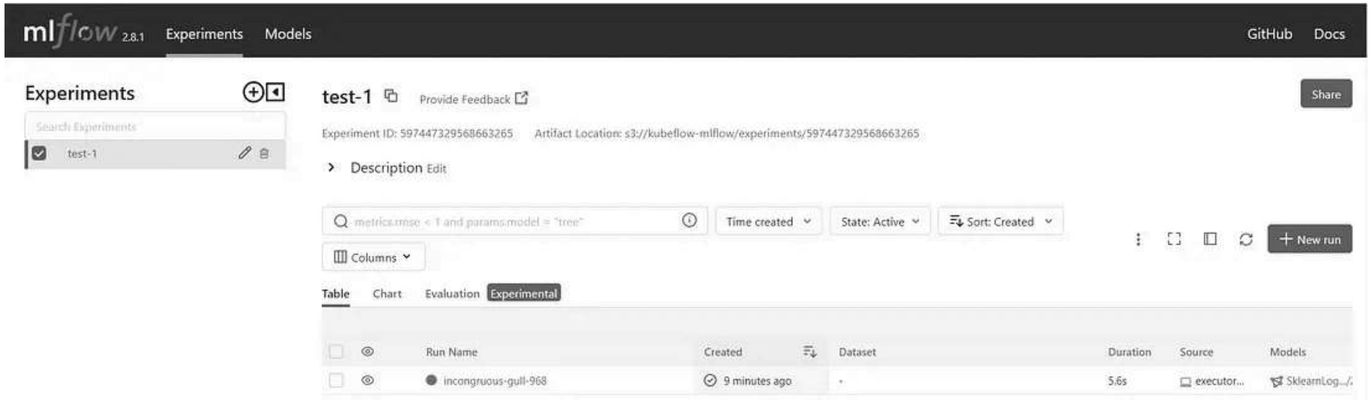
```python
    # Set AWS credentials in the environment
    os.environ["AWS_ACCESS_KEY_ID"] = aws_access_key_id
    os.environ["AWS_SECRET_ACCESS_KEY"] = aws_secret_access_key
    os.environ["AWS_DEFAULT_REGION"] = aws_default_region

    # log and register the model using MLflow scikit-learn API
    mlflow.set_tracking_uri("http://mlflowserver.kubeflow:5000")
    reg_model_name = "SklearnLogisticRegression"

    experiment_id = mlflow.create_experiment("test-1")

    with mlflow.start_run(experiment_id=experiment_id) as run:
        mlflow.log_param('max_iter', 500)

        # Log model artifact to S3
        artifact_path = "sklearn-model"
        mlflow.log_artifact(local_path=f'{data_path}/model.pkl', artifact_path=a

        model_info = mlflow.sklearn.log_model(
            sk_model=logistic_reg_model,
            artifact_path="sklearn-model",
            signature=signature,
            registered_model_name=reg_model_name,
        )

    model_uri = f"runs:/{run.info.run_id}/sklearn-model"

    # Register model linked to S3 artifact location
    mlflow.register_model(
        model_uri,
        reg_model_name
    )

    return {"artifact_path": artifact_path, "artifact_uri": run.info.artifact_ur
```

Kubernetes services are accessible through DNS internally using the name `<service-name>.<namespace>` , earlier we deployed Kubernetes service `mlflowserver` exposes this container port as a service on port 5000 and connecting to `mlflowserver.kubeflow:5000` routes requests to the MLflow container on port 5000.

Hence we are using MLServer tracking URI as `http://mlflowserver.kubeflow:5000` in the above step.

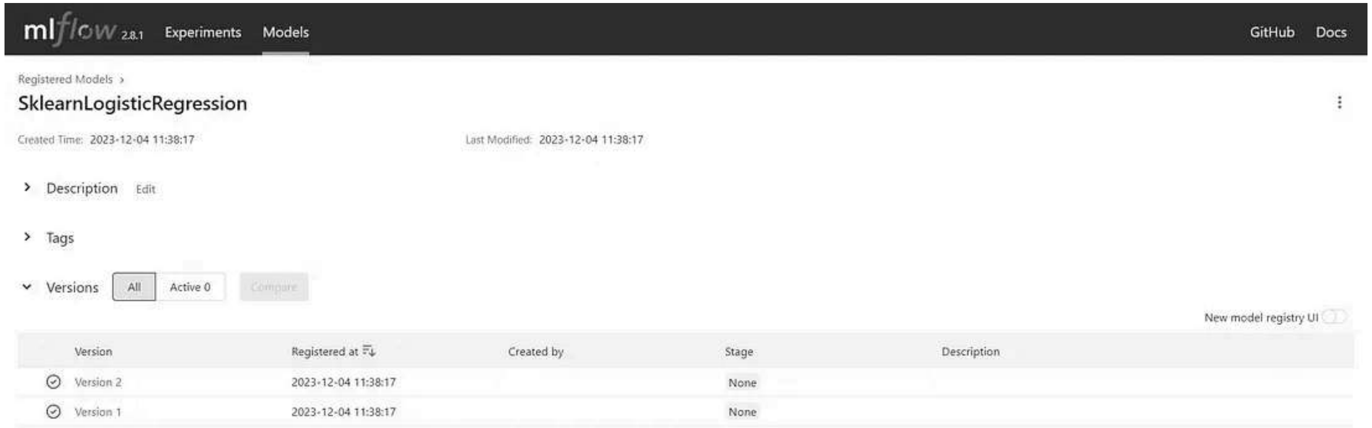
Notice that, we are setting AWS credentials in environment variables because Kubeflow pipeline steps run in separate pods from the base cluster and need those credentials to store model artifacts in S3.

We can observe model experiments are being tracked in MLflow UI after executing the above step in the ML pipeline.



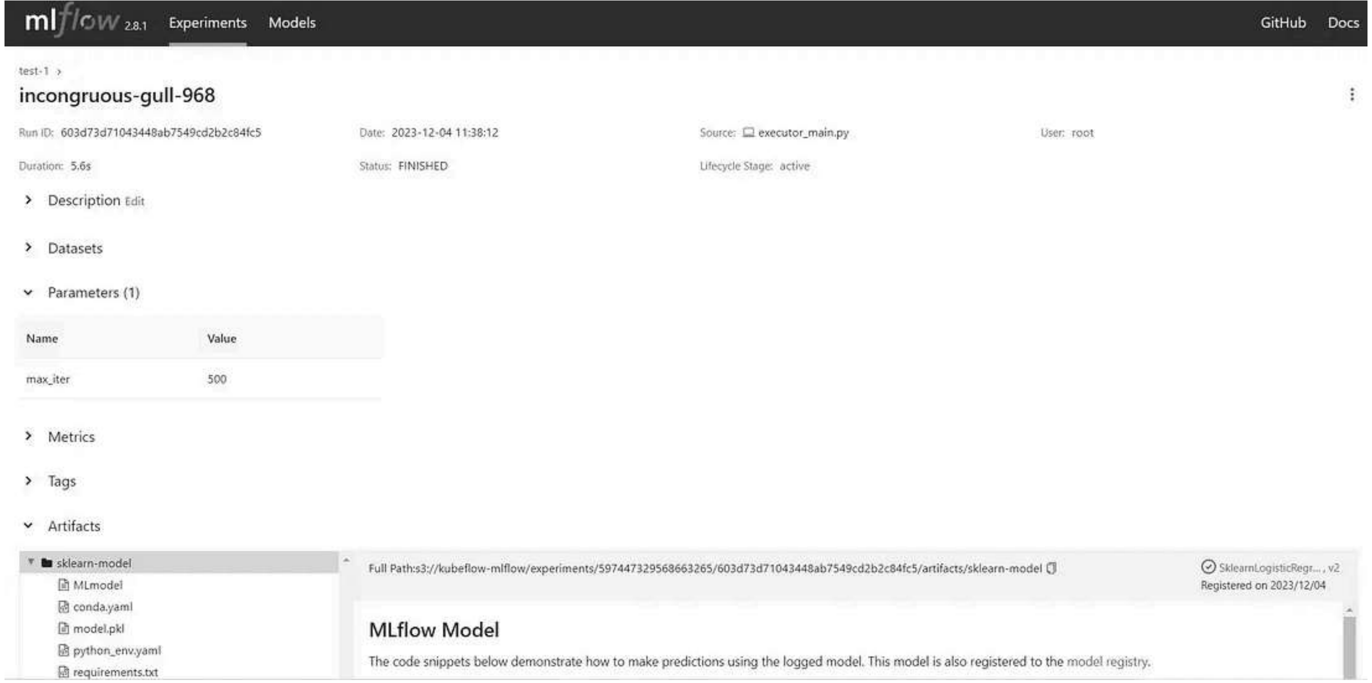
Model experiments (Image by Author)

Another approach to viewing the model registry is from the perspective of the Registered Models, not the Experiments.



Registered models (Image by Author)

The Model Versions are how a model lineage can be traced.



Metadata and Artifact details (Image by Author)

We can see that logged parameter `max_iter=500` from metadata store, logged and registered model artifacts from S3.

6. Model evaluation

Let's load the model from the Model Registry and score the test data. We can observe that we are setting AWS credentials in environment variables because we need load model artifacts in S3 and use it for prediction on test data.

```
@component(
    packages_to_install=["pandas", "numpy", "scikit-learn", "mlflow", "boto3"],
```

```
        base_image="python:3.9",
    )
    def predict_on_test_data(data_path: str, model_info: dict, aws_access_key_id: st
        import pandas as pd
        import numpy as np
        import pickle
        import os
        import mlflow

        # Set AWS credentials in the environment
        os.environ["AWS_ACCESS_KEY_ID"] = aws_access_key_id
        os.environ["AWS_SECRET_ACCESS_KEY"] = aws_secret_access_key
        os.environ["AWS_DEFAULT_REGION"] = aws_default_region

        artifact_path = model_info["artifact_path"]
        artifact_uri = model_info["artifact_uri"]

        mlflow.set_tracking_uri("http://mlflowserver.kubeflow:5000")
        model_uri = f"{artifact_uri}/{artifact_path}"
        logistic_reg_model = mlflow.sklearn.load_model(model_uri)

        X_test = np.load(f'{data_path}/X_test.npy',allow_pickle=True)
        y_pred = logistic_reg_model.predict(X_test)
        np.save(f'{data_path}/y_pred.npy', y_pred)

        X_test = np.load(f'{data_path}/X_test.npy',allow_pickle=True)
        y_pred_prob = logistic_reg_model.predict_proba(X_test)
        np.save(f'{data_path}/y_pred_prob.npy', y_pred_prob)

        return model_uri
```

7. Model deployment

We will utilize the KServe Python Client SDK that interacts with KServe control plane APIs for executing operations on a remote KServe cluster, such as creating, patching and deleting of a InferenceService instance.

Let's create a InferenceService using V1beta1InferenceService that is the Schema for the InferenceServices API in the following example.

```
@component(
    packages_to_install=["kserve"],
    base_image="python:3.9",
)
def model_serving(model_uri: str):
    from kubernetes import client
    from kserve import KServeClient
    from kserve import constants
    from kserve import utils
    from kserve import V1beta1InferenceService
    from kserve import V1beta1InferenceServiceSpec
    from kserve import V1beta1PredictorSpec
    from kserve import V1beta1SKLearnSpec
    import os

    namespace = utils.get_default_target_namespace()

    name='sklearn-iris-v2'
    kserve_version='v1beta1'
    api_version = constants.KSERVE_GROUP + '/' + kserve_version

    isvc = V1beta1InferenceService(api_version=api_version,
                                   kind=constants.KSERVE_KIND,
                                   metadata=client.V1ObjectMeta(
                                       name=name, namespace=namespace, annotatio
                                   spec=V1beta1InferenceServiceSpec(
                                   predictor=V1beta1PredictorSpec(
```

```
                                        service_account_name='mlflow-sa',
                                        sklearn=(V1beta1SKLearnSpec(storage_uri=model
```

Let’s look at the model deployment status and run some inference tests.

Model deployment status (Image by Author)

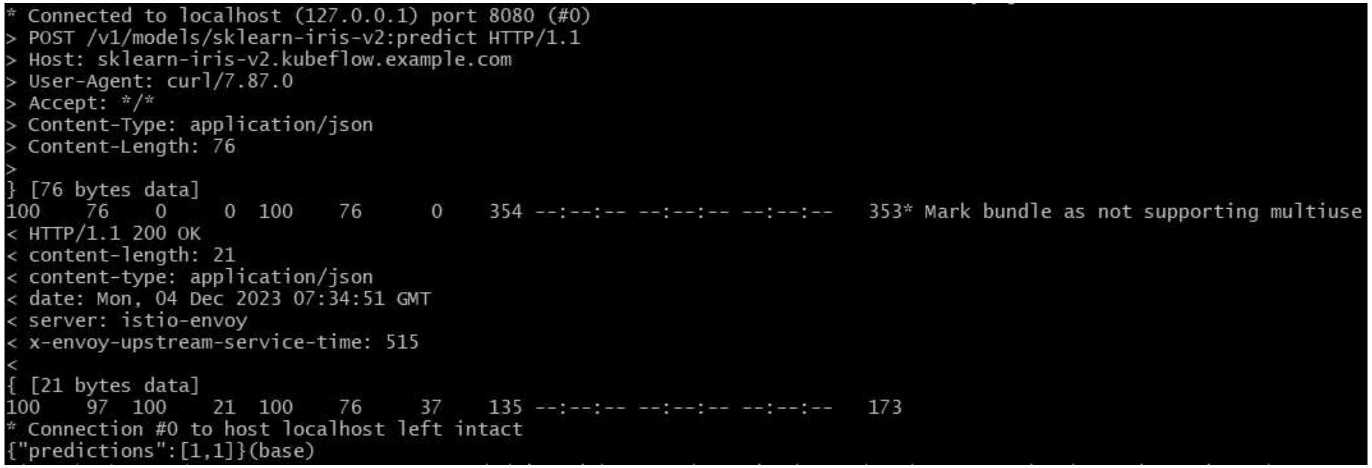
First, we need to do some port forwarding work so our model’s port is exposed to our local system with the command:

```
kubectl port-forward -n istio-system service/istio-ingressgateway 8080:80
```

We’ll use the `curl` command to send the input json file as input to the predict method on our `InferenceService` on KServe with the command:

```
curl -v -H "Host: sklearn-iris-v2.kubeflow.example.com" -H "Content-Type: applic
```

The response will look like:



```
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /v1/models/sklearn-iris-v2:predict HTTP/1.1
> Host: sklearn-iris-v2.kubeflow.example.com
> User-Agent: curl/7.87.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 76
>
} [76 bytes data]
100    76    0     0  100    76      0    354 --:--:-- --:--:-- --:--:--   353* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< content-length: 21
< content-type: application/json
< date: Mon, 04 Dec 2023 07:34:51 GMT
< server: istio-envoy
< x-envoy-upstream-service-time: 515
<
{ [21 bytes data]
100    97  100    21  100    76     37    135 --:--:-- --:--:-- --:--:--   173
* Connection #0 to host localhost left intact
{"predictions":[1,1]}(base)
```

Model inference result (Image by Author)

The Complete Pipeline

Now it's time to put together all components and define the complete pipeline.

```
from kubernetes import client, config
import base64

@pipeline(
    name="iris-pipeline",
)
def iris_pipeline(data_path: str):
    pvc1 = kubernetes.CreatePVC(
        # can also use pvc_name instead of pvc_name_suffix to use a pre-existing
        pvc_name_suffix='-iris-mlflow-pvc',
        access_modes=['ReadWriteMany'],
        size='1Mi',
```

```
        storage_class_name='standard'
    )

    # Load Kubernetes configuration
    config.load_kube_config()

    # Fetch the AWS credentials from the secret
    secret_name = "aws-credentials"
    secret_namespace = "kubeflow"
    secret_key_id = "AWS_ACCESS_KEY_ID"
    secret_key_access = "AWS_SECRET_ACCESS_KEY"
    secret_region = "AWS_DEFAULT_REGION"

    v1 = client.CoreV1Api()
    secret = v1.read_namespaced_secret(secret_name, namespace=secret_namespace)

    # Convert bytes to string
    aws_access_key_id = base64.b64decode(secret.data[secret_key_id]).decode('utf
    aws_secret_access_key = base64.b64decode(secret.data[secret_key_access]).dec
    aws_default_region = base64.b64decode(secret.data[secret_region]).decode('ut

    # Data preparation
    prepare_data_task = prepare_data(data_path=data_path)
    kubernetes.mount_pvc(prepare_data_task, pvc_name=pvc1.outputs['name'], mount

    # Split data into Train and Test set
    train_test_split_task = train_test_split(data_path=data_path)
    kubernetes.mount_pvc(train_test_split_task, pvc_name=pvc1.outputs['name'], m
    train_test_split_task.after(prepare_data_task)

    # Model training
    training_basic_classifier_task = training_basic_classifier(data_path=data_pa
    kubernetes.mount_pvc(training_basic_classifier_task, pvc_name=pvc1.outputs['
    training_basic_classifier_task.after(train_test_split_task)

    # Register a model in the MLflow model registry
    register_model_task = register_model(data_path=data_path, aws_access_key_id=
    kubernetes.mount_pvc(register_model_task, pvc_name=pvc1.outputs['name'], mou
    kubernetes.mount_pvc(register_model_task, pvc_name="mlflow-pvc", mount_path=
    register_model_task.after(training_basic_classifier_task)
```

```
    # Model evaluation
    predict_on_test_data_task = predict_on_test_data(data_path=data_path, model_
    kubernetes.mount_pvc(predict_on_test_data_task, pvc_name=pvc1.outputs['name'
    predict_on_test_data_task.after(register_model_task)

    # Model deployment
    model_serving_task = model_serving(model_uri=predict_on_test_data_task.outpu
    model_serving_task.after(predict_on_test_data_task)

    delete_pvc1 = kubernetes.DeletePVC(pvc_name=pvc1.outputs['name']).after(mode
```

Fetch AWS credentials from Secret: As discussed in prerequisites, we created a secret that holds AWS credentials that needs to be passed in Model registration and evaluation steps.

We created `pvc1` persistent volume claim with size of 1MB that can be used to store the prepared data hence we need to mount `pvc1` to data preparation step and remaining steps to access to prepared data from PVC.

We will log the parameters to metadata store that is PVC in this example and log the model artifacts to AWS S3 then return the dictionary that holds the `artifact_path` and `artifact_uri` details.

We can notice that `register_model_task.output` object passed to model evaluation step and that holds the artifact details from model registration step.

We connect pipeline steps and make sure those execute one after another using `after` function from Kubeflow. Finally, we delete the `pvc1` to cleanup the data.

🎉 Conclusion

We embarked on the journey to build a comprehensive end-to-end machine learning workflow using Kubernetes, Minikube, and a trio of powerful open-source technologies — Kubeflow Pipelines, MLflow, and KServe. The objective was to seamlessly transition from data processing to model deployment within a unified environment.

By the end of this comprehensive guide, hope you will be equipped to navigate through the intricacies of building, deploying, and monitoring machine learning models using the potent combination of Kubernetes and these cutting-edge open-source tools.

I hope you enjoyed this blog post, if you have any questions, feel free to contact me on LinkedIn and share your experience in the comment section.

The complete source code for this post is available in the following link.

AI-ML-Projects/ML-Pipeline-with-Kubeflow-MLflow-KServe at main · vinayak-shanawad/AI-ML-Projects

AWS SageMaker: Train, Deploy and Update a BERT model on Disaster Tweets Classification dataset …

github.com

Kubernetes · Kubeflow · Mlflow · Mlops

Written by Vinayak Shanawad

Following

156 Followers

Machine Learning Engineer | 3x Kaggle Expert | MLOps | LLMOps | Learning, improving and evolving.

More from Vinayak Shanawad

Vinayak Shanawad

Serving Hugging Face Transformers: Optimizing Custo...

Transforming Deployment: Seldon Core Meets Hugging Face

Sep 14, 2023 20 1

Kia Eisinga in Analytics Vidhya

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open...

Jan 27, 2020 2.7K 30

Confusion Matrix, Accuracy, Precision, Recall, F1 Score

Binary Classification Metric

Dec 10, 2019 1.1K 6

Build an ML Inference Data Pipeline using SageMaker and...

Automate and streamline our ML inference pipeline with SageMaker and Airflow

Mar 27, 2023 68 2

See all from Vinayak Shanawad

Recommended from Medium

Peeush Agarwal

End-to-End MLOps Pipeline using MLFlow (Part-2)

Koushik Dutta in Dev Genius

Airflow with Git-Sync in Kubernetes

Data Preprocessing, Model Training, and Hyperparameter Tuning

Oct 10 27 2

Install Airflow with Git-Sync in Kubernetes Cluster, so that once DAG is Pushed to Git...

Aug 8 4

Lists

Business

41 stories · 154 saves

Natural Language Processing

1798 stories · 1408 saves

Kevinnjagi

MLFlow Tutorial

Machine learning models are increasingly becoming a key part of decision-making in...

Oct 15 6

Anna

A Comprehensive Guide to MLflow: What It Is, Its Pros and Cons, and...

Machine learning (ML) development can be complex, involving many steps from data...

May 30 62

Panu Koskela in Nebius

Slurm vs Kubernetes: Which to choose for your ML workloads

Scaling your machine learning workloads will eventually require resource orchestration....

Analytics Insight

Top End-to-End Open-Source MLOps Tools for 2024

Top Open-Source MLOps Tools for Efficient Machine Learning Operations in 2024