**MLFLOW Pipeline -**

Here's an example code snippet to create an MLOps pipeline with MLFlow that triggers a retraining pipeline if accuracy falls below a threshold:

import mlflow

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Set up MLFlow experiment

mlflow.set\_experiment("my\_experiment")

# Define parameters

num\_trees = 100

max\_depth = 5

threshold = 0.8

# Define function to train model and calculate accuracy

def train\_model(data):

# Split data into features and target

X = data.drop(columns=["target"])

y = data["target"]

# Train model

model = RandomForestClassifier(n\_estimators=num\_trees, max\_depth=max\_depth)

model.fit(X, y)

# Calculate accuracy

y\_pred = model.predict(X)

accuracy = accuracy\_score(y, y\_pred)

# Log metrics to MLFlow

mlflow.log\_metric("accuracy", accuracy)

mlflow.log\_param("num\_trees", num\_trees)

mlflow.log\_param("max\_depth", max\_depth)

# Return model and accuracy

return model, accuracy

# Define function to load data

def load\_data():

# Load data from file

data = pd.read\_csv("data.csv")

# Log data to MLFlow

mlflow.log\_param("data\_path", "data.csv")

# Return data

return data

# Define function to evaluate accuracy and trigger retraining if necessary

def evaluate\_accuracy(run\_id):

# Get accuracy from MLFlow

accuracy = mlflow.get\_metric\_history(run\_id, "accuracy")[-1].value

# Check if accuracy is below threshold

if accuracy < threshold:

# Log message to MLFlow

mlflow.log\_param("message", "Accuracy below threshold. Retraining model.")

# Load data

data = load\_data()

# Train model and log metrics to MLFlow

model, accuracy = train\_model(data)

# Log model to MLFlow

mlflow.sklearn.log\_model(model, "model")

# Load data

data = load\_data()

# Train initial model and log metrics to MLFlow

model, accuracy = train\_model(data)

# Log model to MLFlow

mlflow.sklearn.log\_model(model, "model")

# Evaluate accuracy and trigger retraining if necessary

evaluate\_accuracy(mlflow.active\_run().info.run\_id)

**Kubeflow Trigger if Accurancy fall below Threshould -**

import kfp

import kfp.dsl as dsl

import kfp.components as comp

import kfp.gcp as gcp

# Define parameters

num\_trees = 100

max\_depth = 5

threshold = 0.8

# Define pipeline components

@dsl.component

def train\_model\_op(data\_path):

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Load data

data = pd.read\_csv(data\_path)

# Split data into features and target

X = data.drop(columns=["target"])

y = data["target"]

# Train model

model = RandomForestClassifier(n\_estimators=num\_trees, max\_depth=max\_depth)

model.fit(X, y)

# Calculate accuracy

y\_pred = model.predict(X)

accuracy = accuracy\_score(y, y\_pred)

# Log accuracy to Kubeflow

print("accuracy: ", accuracy)

return model, accuracy

@dsl.component

def evaluate\_accuracy\_op(model, data\_path):

import pandas as pd

from sklearn.metrics import accuracy\_score

# Load data

data = pd.read\_csv(data\_path)

# Split data into features and target

X = data.drop(columns=["target"])

y = data["target"]

# Predict target using model

y\_pred = model.predict(X)

# Calculate accuracy

accuracy = accuracy\_score(y, y\_pred)

# Log accuracy to Kubeflow

print("accuracy: ", accuracy)

return accuracy

# Define pipeline

@dsl.pipeline(

name="mlops\_pipeline",

description="A pipeline that trains a random forest classifier and evaluates accuracy."

)

def mlops\_pipeline(data\_path):

# Train initial model

train\_model = train\_model\_op(data\_path)

model = train\_model.outputs["model"]

accuracy = train\_model.outputs["accuracy"]

# Evaluate accuracy and trigger retraining if necessary

with dsl.Condition(accuracy < threshold):

# Train new model

train\_model = train\_model\_op(data\_path)

model = train\_model.outputs["model"]

accuracy = train\_model.outputs["accuracy"]

# Evaluate final accuracy

evaluate\_accuracy = evaluate\_accuracy\_op(model, data\_path)

# Compile pipeline

pipeline\_func = kfp.compiler.compile(mlops\_pipeline, "mlops\_pipeline.tar.gz")

# Define Kubeflow client

client = kfp.Client()

# Define pipeline arguments

arguments = {"data\_path": "gs://my-bucket/data.csv"}

# Submit pipeline

experiment\_name = "my\_experiment"

run\_name = "my\_run"

run = client.create\_run\_from\_pipeline\_func(

pipeline\_func=pipeline\_func,

experiment\_name=experiment\_name,

run\_name=run\_name,

arguments=arguments,

)

Here are the steps involved in creating an MLOps pipeline with Kubeflow that triggers a retraining pipeline if accuracy falls below a threshold:

1. Define the pipeline parameters: You will need to define the parameters for your pipeline, such as the number of trees in your random forest classifier, the maximum depth of the trees, and the accuracy threshold.
2. Define the pipeline components: You will need to define the components of your pipeline, such as loading and cleaning the data, training the model, and evaluating the accuracy.
3. Define the pipeline: Using the Kubeflow DSL, define the steps of your pipeline, including loading the data, training the model, evaluating the accuracy, and triggering a retraining pipeline if necessary.
4. Compile the pipeline: Compile your pipeline into a package that can be executed by Kubeflow.
5. Submit the pipeline: Submit the compiled pipeline to Kubeflow using the Kubeflow client.
6. Monitor the pipeline: Monitor the progress of your pipeline and the accuracy of your model using Kubeflow's monitoring tools.
7. Retrain the model: If the accuracy falls below the threshold, trigger a retraining pipeline that uses the updated data to train a new model.
8. Update the production model: Once the new model has been trained and its accuracy is above the threshold, deploy the new model to production and update the serving endpoint.