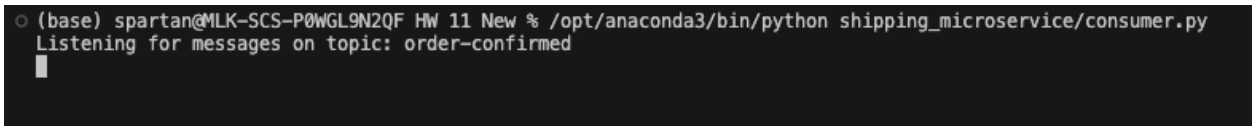
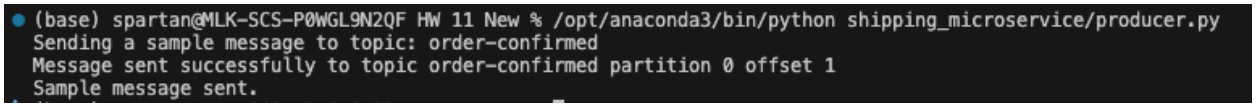
DATA-236 Sec 12 - Distributed Systems for Data Engineering
HOMEWORK 11
Nandhakumar Apparsamy
018190003

GitHub - https://github.com/Nandha951/DATA-236-HW-11-MLOps

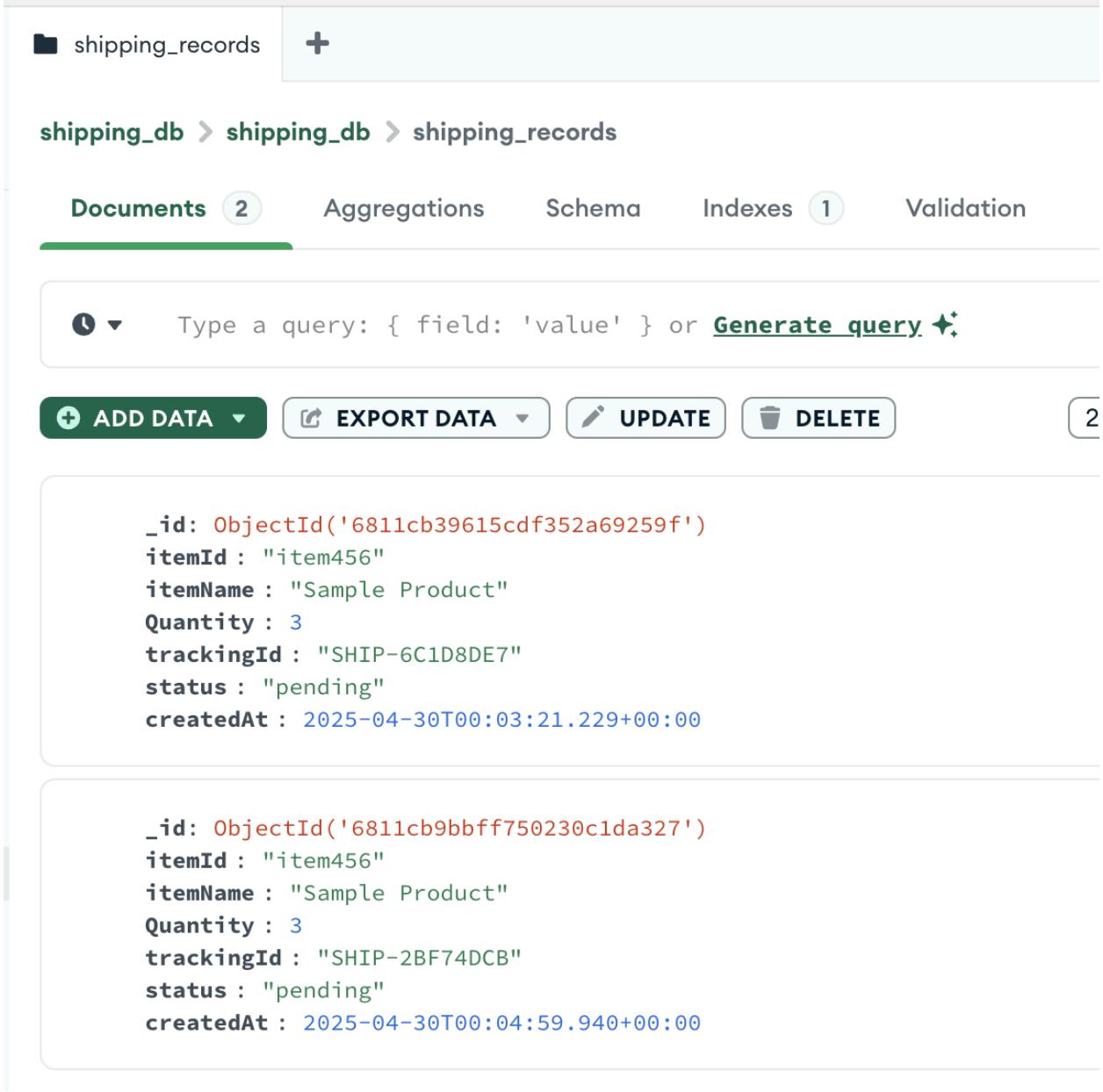
Q1. Shipping Microservice (5 Marks)
Build a shipping microservice that listens to the “order-confirmed” topic and creates a shipping record. (Refer the demo code for orders topic).
Requirements:
- Connect to Kafka as a consumer.
- Subscribe to the topic: "order-confirmed"





- Save record to db with tracking id and status
    - Generate a tracking Id: (e.g SHIP-XYZ)
    - Set the status to ‘Pending’

○ Save the record



shipping_microservice > database.py > save_shipping_record

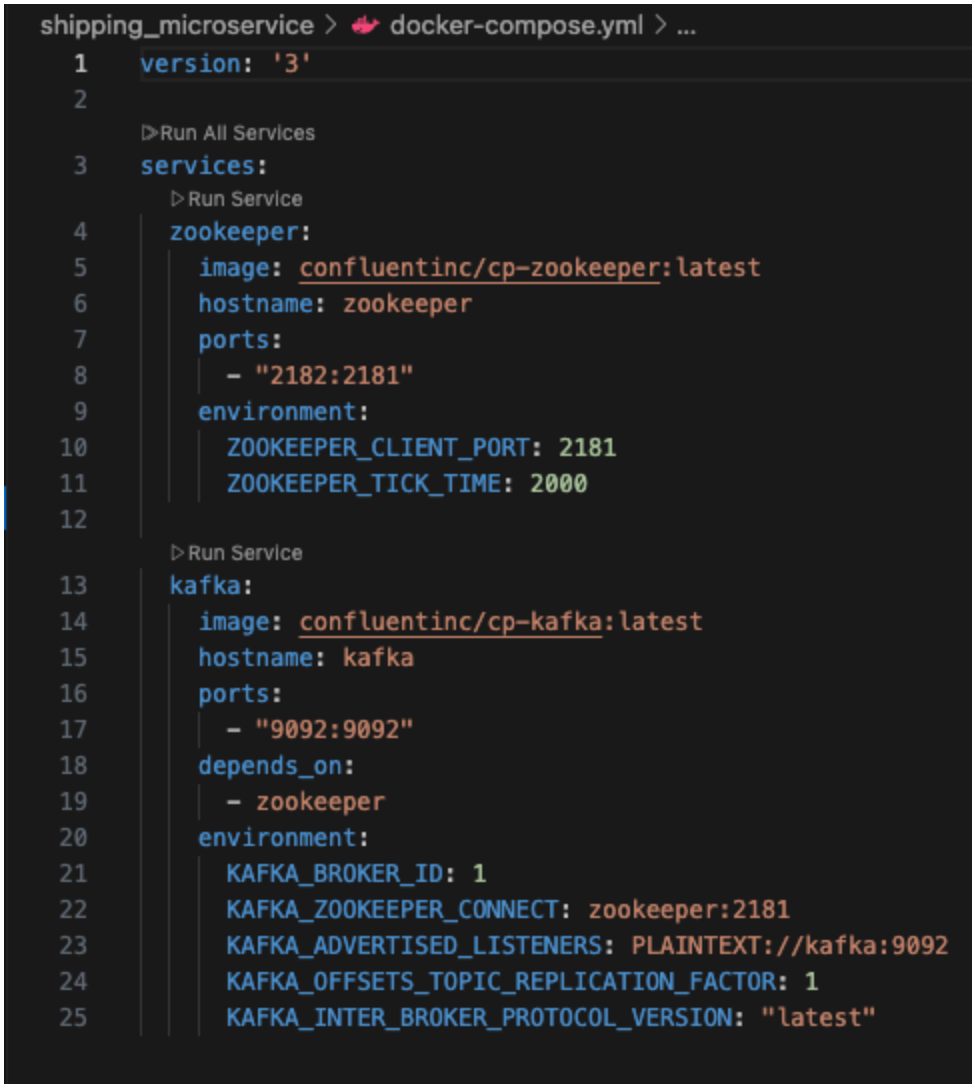
```python
from pymongo import MongoClient
from datetime import datetime

# Replace with your MongoDB connection string
MONGO_URI = "mongodb://localhost:27017/"
DATABASE_NAME = "shipping_db"
COLLECTION_NAME = "shipping_records"

client = MongoClient(MONGO_URI)
db = client[DATABASE_NAME]
shipping_collection = db[COLLECTION_NAME]

def save_shipping_record(item_id: str, item_name: str, quantity: int, tracking_id: str):
    """Saves a new shipping record to the database."""
    record = {
        "itemId": item_id,
        "itemName": item_name,
        "Quantity": quantity,
        "trackingId": tracking_id,
        "status": "pending", # Default status
        "createdAt": datetime.now()
    }
    try:
        result = shipping_collection.insert_one(record)
        print(f"Shipping record saved with ID: {result.inserted_id}")
        return result.inserted_id
    except Exception as e:
        print(f"Error saving shipping record: {e}")
        return None

if __name__ == "__main__":
    # Example usage (for testing)
    # Make sure your MongoDB server is running
    print("Testing database connection and save function...")
    test_record_id = save_shipping_record("item123", "Test Item", 2, "SHIP-TEST-001")
    if test_record_id:
        print(f"Test record saved successfully with ID: {test_record_id}")
    else:
        print("Failed to save test record.")
```

shipping_microservice > producer.py > send_order_confirmed_message

```python
from kafka import KafkaProducer
import json
import time

# Replace with your Kafka broker address
KAFKA_BROKER = "localhost:9092"
ORDER_CONFIRMED_TOPIC = "order-confirmed"

def send_order_confirmed_message(item_id: str, item_name: str, quantity: int):
    """Sends an order confirmed message to the Kafka topic."""
    producer = KafkaProducer(
        bootstrap_servers=[KAFKA_BROKER],
        value_serializer=lambda x: json.dumps(x).encode('utf-8')
    )

    message = {
        "itemId": item_id,
        "itemName": item_name,
        "Quantity": quantity
    }

    try:
        future = producer.send(ORDER_CONFIRMED_TOPIC, value=message)
        result = future.get(timeout=60)
        print(f"Message sent successfully to topic {result.topic} partition {result.partition} offset {result.offset}")
    except Exception as e:
        print(f"Error sending message: {e}")
    finally:
        producer.close()

if __name__ == "__main__":
    print(f"Sending a sample message to topic: {ORDER_CONFIRMED_TOPIC}")
    # Example message
    send_order_confirmed_message("item456", "Sample Product", 3)
    time.sleep(1) # Give some time for the message to be sent
    print("Sample message sent.")
```

shipping_microservice > consumer.py > consume_order_confirmed

```python
from kafka import KafkaConsumer
import json
import uuid
from database import save_shipping_record

# Replace with your Kafka broker address
KAFKA_BROKER = "localhost:9092"
ORDER_CONFIRMED_TOPIC = "order-confirmed"

def consume_order_confirmed():
    """Consumes messages from the 'order-confirmed' topic and saves shipping records."""
    consumer = KafkaConsumer(
        ORDER_CONFIRMED_TOPIC,
        bootstrap_servers=[KAFKA_BROKER],
        auto_offset_reset='earliest',
        enable_auto_commit=True,
        group_id='shipping-service-group',
        value_deserializer=lambda x: json.loads(x.decode('utf-8'))
    )

    print(f"Listening for messages on topic: {ORDER_CONFIRMED_TOPIC}")

    try:
        for message in consumer:
            print(f"Received message: {message.value}")
            order_data = message.value

            # Extract necessary information from the order message
            # Assuming the order message structure contains itemId, itemName, and Quantity
            item_id = order_data.get("itemId")
            item_name = order_data.get("itemName")
            quantity = order_data.get("Quantity")

            if item_id and item_name and quantity is not None:
                # Generate tracking ID
                tracking_id = f"SHIP-{uuid.uuid4().hex[:8].upper()}"
                print(f"Generated tracking ID: {tracking_id}")

                # Save shipping record to database
                save_shipping_record(item_id, item_name, quantity, tracking_id)
            else:
                print(f"Skipping message due to missing data: {order_data}")

    except Exception as e:
        print(f"Error consuming messages: {e}")
    finally:
        consumer.close()
        print("Kafka consumer closed.")

if __name__ == "__main__":
    consume_order_confirmed()
```



NOTE: The shipping schema should consist of
itemId: String
itemName: String
Quantity: Number
trackingId: String
status: { type: String, default: "pending" }
createdAt: { type: Date, default: Date.now }

Q2. ML-OPS (5 Marks)

Create a mini-MLOps pipeline using your own model and deploy a FastAPI-based prediction endpoint.

● Train a Model - Use a simple dataset like Iris, Wine, or your own mini regression/classification problem, Save the model as a .pkl file using joblib

```
(base) spartan@MLK-SCS-P0WGL9N2QF mlops_project % /opt/anaconda3/bin/python train_model.py
Loading Iris dataset...
Dataset loaded.
Training Logistic Regression model...
Model training complete.
Saving model to models/model.pkl...
Model saved successfully.
```

train_model.py > ...

```python
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import joblib
import os

def train_and_save_model():
    """Trains a Logistic Regression model on the Iris dataset and saves it."""
    print("Loading Iris dataset...")
    iris = load_iris()
    X, y = iris.data, iris.target
    print("Dataset loaded.")

    print("Training Logistic Regression model...")
    model = LogisticRegression(max_iter=200)
    model.fit(X, y)
    print("Model training complete.")

    # Create a directory for the model if it doesn't exist
    model_dir = "models"
    os.makedirs(model_dir, exist_ok=True)
    model_path = os.path.join(model_dir, "model.pkl")

    print(f"Saving model to {model_path}...")
    joblib.dump(model, model_path)
    print("Model saved successfully.")

if __name__ == "__main__":
    train_and_save_model()
```

● Serve Using FastAPI - Build an API with /predict endpoint using FastAPI. Accept input as JSON and return the predicted output

```
(base) spartan@MLK-SCS-P0WGL9N2QF mlops_project % /opt/anaconda3/bin/python -m uvicorn app:app --reload

INFO:     Will watch for changes in these directories: ['/Users/spartan/SJSU/DATA 236/HW Assignment/HW 11 New/mlops_project']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [15009] using StatReload
Model loaded successfully.
INFO:     Started server process [15012]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:49879 - "POST /predict HTTP/1.1" 200 OK
```

HTTP DATA 236 - HW 11 / **New Request**

POST http://127.0.0.1:8000/predict

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

```
{
  "sepal_length": 5.1,
  "sepal_width": 3.5,
  "petal_length": 1.4,
  "petal_width": 0.2
}
```

Body Cookies Headers (4) Test Results 200 OK 15 ms

{ } JSON Preview Visualize

```
{
    "prediction": 0,
    "predicted_species": "setosa"
}
```

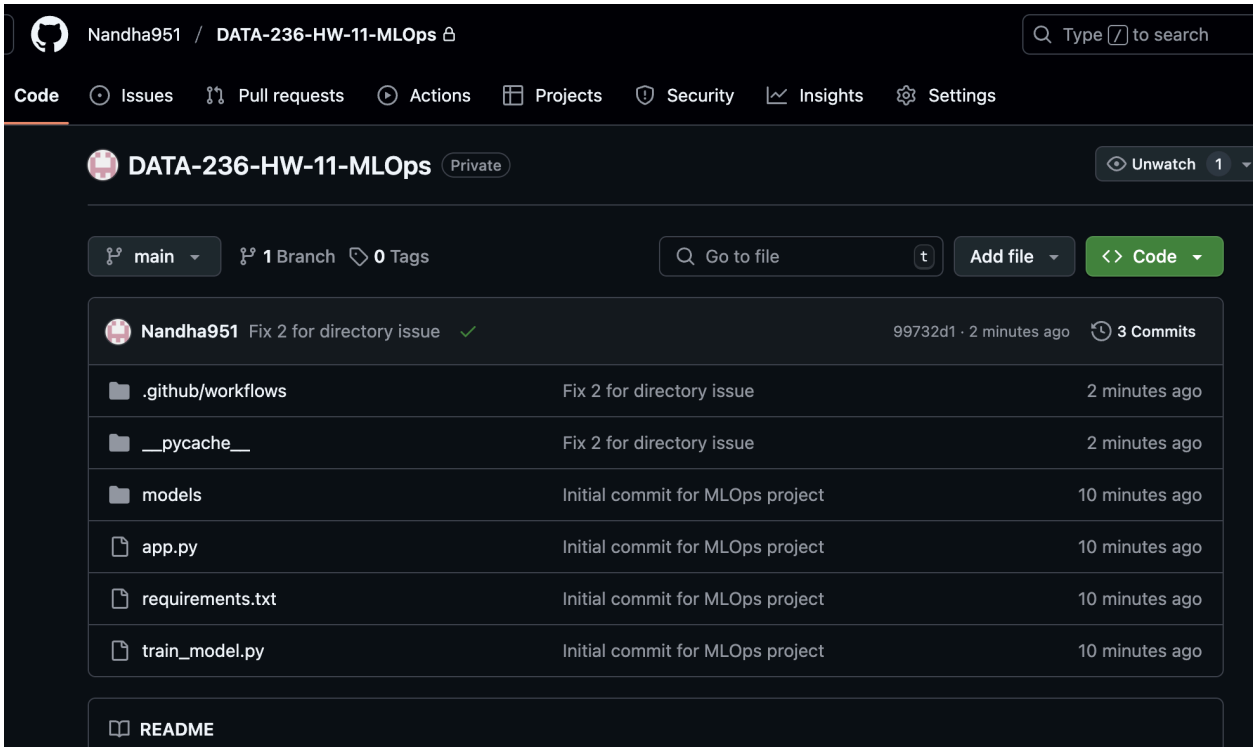
app.py > ...

```python
from fastapi import FastAPI
from pydantic import BaseModel
import joblib
import numpy as np
import os

# Define the path to the saved model
MODEL_PATH = os.path.join("models", "model.pkl")

# Load the trained model
try:
    model = joblib.load(MODEL_PATH)
    print("Model loaded successfully.")
except FileNotFoundError:
    print(f"Error: Model file not found at {MODEL_PATH}. Please run train_model.py first.")
    model = None # Set model to None if loading fails
except Exception as e:
    print(f"Error loading model: {e}")
    model = None

# Define the input data model based on Iris dataset features
class IrisFeatures(BaseModel):
    sepal_length: float
    sepal_width: float
    petal_length: float
    petal_width: float

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "MLOps FastAPI service is running. Go to /docs for API documentation."}

@app.post("/predict")
def predict_iris(features: IrisFeatures):
    """
    Predicts the Iris species based on input features.
    """
    if model is None:
        return {"error": "Model not loaded. Cannot make predictions."}

    # Convert input features to a numpy array
    data = np.array([[
        features.sepal_length,
        features.sepal_width,
        features.petal_length,
        features.petal_width
    ]])
```

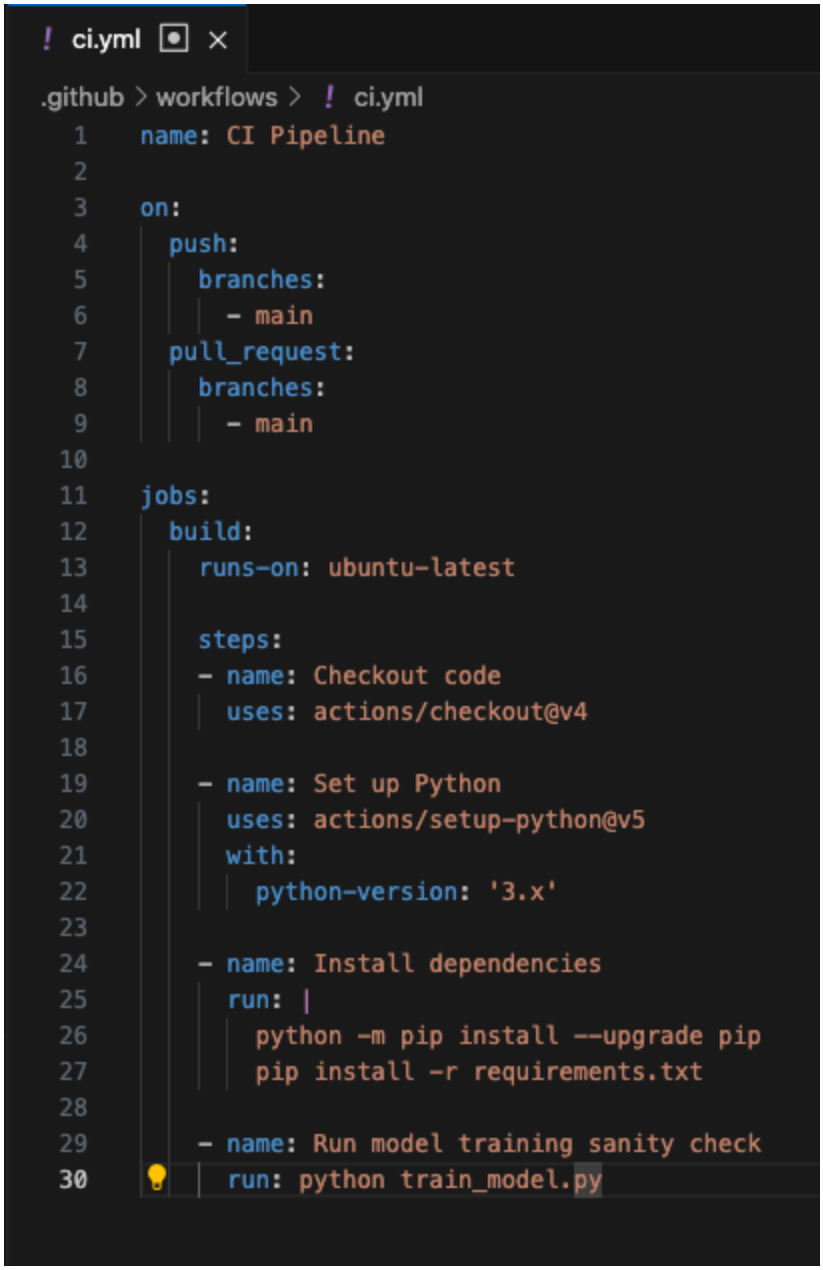
app.py > ...

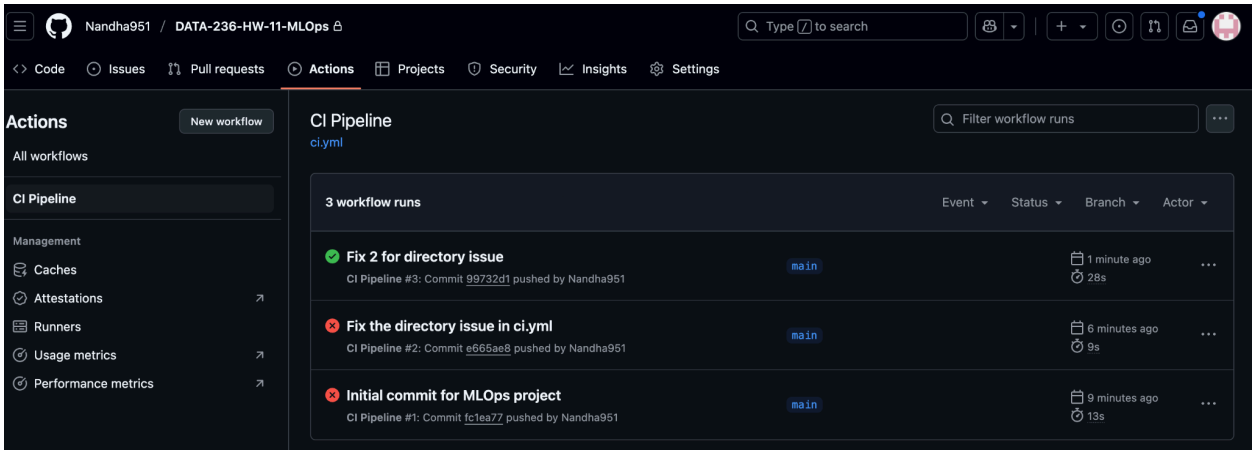
```python
def predict_iris(features: IrisFeatures):

    # Make prediction
    prediction = model.predict(data).tolist() # Convert numpy array to list for JSON response

    # Assuming Iris dataset target names for interpretation
    # You might need to adjust this based on your specific model/dataset
    iris_target_names = ["setosa", "versicolor", "virginica"]
    predicted_species = iris_target_names[prediction[0]] if prediction and 0 <= prediction[0] < len(iris_target_names) else "unknown"


    return {"prediction": prediction[0], "predicted_species": predicted_species}

if __name__ == "__main__":
    import uvicorn
    # To run the app, use: uvicorn app:app --reload
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

● Set Up GitHub Repo- Upload your project files: app.py, model.pkl, requirements.txt, etc.



● Add CI Workflow - Create .github/workflows/ci.yml. It should install dependencies, run the FastAPI script or sanity check model load.


! ci.yml
.github > workflows > ! ci.yml
name: CI Pipeline
on:
push:
branches:
- main
pull_request:
branches:
- main
jobs:
build:
runs-on: ubuntu-latest
steps:
- name: Checkout code
uses: actions/checkout@v4
- name: Set up Python
uses: actions/setup-python@v5
with:
python-version: '3.x'
- name: Install dependencies
run: |
python -m pip install --upgrade pip
pip install -r requirements.txt
- name: Run model training sanity check
run: python train_model.py



Submit the screenshot of your CI workflow.yml and the actions page of your GitHub repo with the pipeline successful.