

**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"

```
(base) spartan@MLK-SCS-P0WGL9N2QF HW 11 New % /opt/anaconda3/bin/python shipping_microservice/consumer.py
Listening for messages on topic: order-confirmed
```

```
(base) spartan@MLK-SCS-P0WGL9N2QF HW 11 New % /opt/anaconda3/bin/python shipping_microservice/producer.py
Sending a sample message to topic: order-confirmed
Message sent successfully to topic order-confirmed partition 0 offset 1
Sample message sent.
```

- 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\_records



shipping\_db > shipping\_db > shipping\_records

Documents

2

Aggregations

Schema

Indexes

1

Validation



Type a query: { field: 'value' } or [Generate query](#)

+ ADD DATA

EXPORT DATA

UPDATE



DELETE

2

```
_id: ObjectId('6811cb39615cdf352a69259f')
itemId: "item456"
itemName: "Sample Product"
Quantity: 3
trackingId: "SHIP-6C1D8DE7"
status: "pending"
createdAt: 2025-04-30T00:03:21.229+00:00
```

```
_id: ObjectId('6811cb9bbff750230c1da327')
itemId: "item456"
itemName: "Sample Product"
Quantity: 3
trackingId: "SHIP-2BF74DCB"
status: "pending"
createdAt: 2025-04-30T00:04:59.940+00:00
```

```
shipping_microservice > database.py > save_shipping_record
1  from pymongo import MongoClient
2  from datetime import datetime
3
4  # Replace with your MongoDB connection string
5  MONGO_URI = "mongodb://localhost:27017/"
6  DATABASE_NAME = "shipping_db"
7  COLLECTION_NAME = "shipping_records"
8
9  client = MongoClient(MONGO_URI)
10 db = client[DATABASE_NAME]
11 shipping_collection = db[COLLECTION_NAME]
12
13 def save_shipping_record(item_id: str, item_name: str, quantity: int, tracking_id: str):
14     """Saves a new shipping record to the database."""
15     record = {
16         "itemId": item_id,
17         "itemName": item_name,
18         "Quantity": quantity,
19         "trackingId": tracking_id,
20         "status": "pending", # Default status
21         "createdAt": datetime.now()
22     }
23     try:
24         result = shipping_collection.insert_one(record)
25         print(f"Shipping record saved with ID: {result.inserted_id}")
26         return result.inserted_id
27     except Exception as e:
28         print(f"Error saving shipping record: {e}")
29         return None
30
31 if __name__ == "__main__":
32     # Example usage (for testing)
33     # Make sure your MongoDB server is running
34     print("Testing database connection and save function...")
35     test_record_id = save_shipping_record("item123", "Test Item", 2, "SHIP-TEST-001")
36     if test_record_id:
37         print(f"Test record saved successfully with ID: {test_record_id}")
38     else:
39         print("Failed to save test record.")
```

shipping\_microservice >  producer.py >  send\_order\_confirmed\_message

```
1  from kafka import KafkaProducer
2  import json
3  import time
4
5  # Replace with your Kafka broker address
6  KAFKA_BROKER = "localhost:9092"
7  ORDER_CONFIRMED_TOPIC = "order-confirmed"
8
9  def send_order_confirmed_message(item_id: str, item_name: str, quantity: int):
10     """Sends an order confirmed message to the Kafka topic."""
11     producer = KafkaProducer(
12         bootstrap_servers=[KAFKA_BROKER],
13         value_serializer=lambda x: json.dumps(x).encode('utf-8')
14     )
15
16     message = {
17         "itemId": item_id,
18         "itemName": item_name,
19         "Quantity": quantity
20     }
21
22     try:
23         future = producer.send(ORDER_CONFIRMED_TOPIC, value=message)
24         result = future.get(timeout=60)
25         print(f"Message sent successfully to topic {result.topic} partition {result.partition} offset {result.offset}")
26     except Exception as e:
27         print(f"Error sending message: {e}")
28     finally:
29         producer.close()
30
31 if __name__ == "__main__":
32     print(f"Sending a sample message to topic: {ORDER_CONFIRMED_TOPIC}")
33     # Example message
34     send_order_confirmed_message("item456", "Sample Product", 3)
35     time.sleep(1) # Give some time for the message to be sent
36     print("Sample message sent.")
```

```

shipping_microservice > consumer.py > consume_order_confirmed
1  from kafka import KafkaConsumer
2  import json
3  import uuid
4  from database import save_shipping_record
5
6  # Replace with your Kafka broker address
7  KAFKA_BROKER = "localhost:9092"
8  ORDER_CONFIRMED_TOPIC = "order-confirmed"
9
10 def consume_order_confirmed():
11     """Consumes messages from the 'order-confirmed' topic and saves shipping records."""
12     consumer = KafkaConsumer(
13         ORDER_CONFIRMED_TOPIC,
14         bootstrap_servers=[KAFKA_BROKER],
15         auto_offset_reset='earliest',
16         enable_auto_commit=True,
17         group_id='shipping-service-group',
18         value_deserializer=lambda x: json.loads(x.decode('utf-8'))
19     )
20
21     print(f"Listening for messages on topic: {ORDER_CONFIRMED_TOPIC}")
22
23     try:
24         for message in consumer:
25             print(f"Received message: {message.value}")
26             order_data = message.value
27
28             # Extract necessary information from the order message
29             # Assuming the order message structure contains itemId, itemName, and Quantity
30             item_id = order_data.get("itemId")
31             item_name = order_data.get("itemName")
32             quantity = order_data.get("Quantity")
33
34             if item_id and item_name and quantity is not None:
35                 # Generate tracking ID
36                 tracking_id = f"SHIP-{uuid.uuid4().hex[:8].upper()}"
37                 print(f"Generated tracking ID: {tracking_id}")
38
39                 # Save shipping record to database
40                 save_shipping_record(item_id, item_name, quantity, tracking_id)
41             else:
42                 print(f"Skipping message due to missing data: {order_data}")
43
44     except Exception as e:
45         print(f"Error consuming messages: {e}")
46     finally:
47         consumer.close()
48         print("Kafka consumer closed.")
49
50 if __name__ == "__main__":
51     consume_order_confirmed()

```

```
shipping_microservice > 🚀 docker-compose.yml > ...
1  version: '3'
2
   >Run All Services
3  services:
   >Run Service
4  zookeeper:
5    image: confluentinc/cp-zookeeper:latest
6    hostname: zookeeper
7    ports:
8      - "2182:2181"
9    environment:
10     ZOOKEEPER_CLIENT_PORT: 2181
11     ZOOKEEPER_TICK_TIME: 2000
12
   >Run Service
13 kafka:
14   image: confluentinc/cp-kafka:latest
15   hostname: kafka
16   ports:
17     - "9092:9092"
18   depends_on:
19     - zookeeper
20   environment:
21     KAFKA_BROKER_ID: 1
22     KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
23     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
24     KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
25     KAFKA_INTER_BROKER_PROTOCOL_VERSION: "latest"
```

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 > ...
1  from sklearn.datasets import load_iris
2  from sklearn.linear_model import LogisticRegression
3  import joblib
4  import os
5
6  def train_and_save_model():
7      """Trains a Logistic Regression model on the Iris dataset and saves it."""
8      print("Loading Iris dataset...")
9      iris = load_iris()
10     X, y = iris.data, iris.target
11     print("Dataset loaded.")
12
13     print("Training Logistic Regression model...")
14     model = LogisticRegression(max_iter=200)
15     model.fit(X, y)
16     print("Model training complete.")
17
18     # Create a directory for the model if it doesn't exist
19     model_dir = "models"
20     os.makedirs(model_dir, exist_ok=True)
21     model_path = os.path.join(model_dir, "model.pkl")
22
23     print(f"Saving model to {model_path}...")
24     joblib.dump(model, model_path)
25     print("Model saved successfully.")
26
27     if __name__ == "__main__":
28         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
```

 DATA 236 - HW 11 / New Request

POST

▼

http://127.0.0.1:8000/predict

ParamsAuthorizationHeaders (8)Body●ScriptsSettings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

```
1  {
2    "sepal_length": 5.1,
3    "sepal_width": 3.5,
4    "petal_length": 1.4,
5    "petal_width": 0.2
6  }
```

BodyCookiesHeaders (4)Test Results

200 OK • 15 ms •

{ } JSON ▼

▶ Preview

 Visualize ▼

```
1  {
2    "prediction": 0,
3    "predicted_species": "setosa"
4  }
```



```
app.py > ...
1  from fastapi import FastAPI
2  from pydantic import BaseModel
3  import joblib
4  import numpy as np
5  import os
6
7  # Define the path to the saved model
8  MODEL_PATH = os.path.join("models", "model.pkl")
9
10 # Load the trained model
11 try:
12     model = joblib.load(MODEL_PATH)
13     print("Model loaded successfully.")
14 except FileNotFoundError:
15     print(f"Error: Model file not found at {MODEL_PATH}. Please run train_model.py first.")
16     model = None # Set model to None if loading fails
17 except Exception as e:
18     print(f"Error loading model: {e}")
19     model = None
20
21 # Define the input data model based on Iris dataset features
22 class IrisFeatures(BaseModel):
23     sepal_length: float
24     sepal_width: float
25     petal_length: float
26     petal_width: float
27
28 app = FastAPI()
29
30 @app.get("/")
31 def read_root():
32     return {"message": "MLOps FastAPI service is running. Go to /docs for API documentation."}
33
34 @app.post("/predict")
35 def predict_iris(features: IrisFeatures):
36     """
37     Predicts the Iris species based on input features.
38     """
39     if model is None:
40         return {"error": "Model not loaded. Cannot make predictions."}
41
42     # Convert input features to a numpy array
43     data = np.array([
44         features.sepal_length,
45         features.sepal_width,
46         features.petal_length,
47         features.petal_width
48     ])
```

```

app.py > ...
35 def predict_iris(features: IrisFeatures):
49
50     # Make prediction
51     prediction = model.predict(data).tolist() # Convert numpy array to list for JSON response
52
53     # Assuming Iris dataset target names for interpretation
54     # You might need to adjust this based on your specific model/dataset
55     iris_target_names = ["setosa", "versicolor", "virginica"]
56     predicted_species = iris_target_names[prediction[0]] if prediction and 0 <= prediction[0] < len(iris_target_names) else "unknown"
57
58
59     return {"prediction": prediction[0], "predicted_species": predicted_species}
60
61 if __name__ == "__main__":
62     import uvicorn
63     # To run the app, use: uvicorn app:app --reload
64     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.

Nandha951 / DATA-236-HW-11-MLOps

Code Issues Pull requests Actions Projects Security Insights Settings

**DATA-236-HW-11-MLOps** Private Unwatch 1

main 1 Branch 0 Tags

Go to file Add file Code

Commit	Files	Message	Time
Nandha951	Fix 2 for directory issue	99732d1 · 2 minutes ago	3 Commits
	.github/workflows	Fix 2 for directory issue	2 minutes ago
	__pycache__	Fix 2 for directory issue	2 minutes ago
	models	Initial commit for MLOps project	10 minutes ago
	app.py	Initial commit for MLOps project	10 minutes ago
	requirements.txt	Initial commit for MLOps project	10 minutes ago
	train_model.py	Initial commit for MLOps project	10 minutes ago

README

- 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

```
1  name: CI Pipeline
2
3  on:
4    push:
5      branches:
6        - main
7    pull_request:
8      branches:
9        - main
10
11  jobs:
12    build:
13      runs-on: ubuntu-latest
14
15      steps:
16        - name: Checkout code
17          uses: actions/checkout@v4
18
19        - name: Set up Python
20          uses: actions/setup-python@v5
21          with:
22            python-version: '3.x'
23
24        - name: Install dependencies
25          run: |
26            python -m pip install --upgrade pip
27            pip install -r requirements.txt
28
29        - name: Run model training sanity check
30          run: python train_model.py
```

The screenshot shows the GitHub Actions interface for a repository named 'DATA-236-HW-11-MLOps'. The 'Actions' tab is selected, displaying a list of workflow runs for the 'CI Pipeline' (ci.yml). The left sidebar shows the 'Actions' section with options like 'All workflows', 'CI Pipeline', 'Management', 'Caches', 'Attestations', 'Runners', 'Usage metrics', and 'Performance metrics'. The main content area shows three workflow runs:

Run Name	Status	Branch	Event	Status	Branch	Actor
Fix 2 for directory issue	Success	main	CI Pipeline #3: Commit 99732d1 pushed by Nandha951	1 minute ago	28s	...
Fix the directory issue in ci.yml	Failure	main	CI Pipeline #2: Commit e665ae8 pushed by Nandha951	6 minutes ago	9s	...
Initial commit for MLOps project	Failure	main	CI Pipeline #1: Commit fc1ea72 pushed by Nandha951	9 minutes ago	13s	...

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