# Real-Time Fraud Detection: Microservice Deployment

## 1. FastAPI Microservice Code

This FastAPI application serves as a RESTful microservice for real-time fraud detection. It loads the pre-trained models (Random Forest, Isolation Forest, Autoencoder) and provides an endpoint (`/predict`) to analyze incoming transaction data.

Below is the complete FastAPI code:

### app.py

from fastapi import FastAPI, HTTPException  
from pydantic import BaseModel  
import numpy as np  
import joblib  
import tensorflow as tf  
from sklearn.preprocessing import MinMaxScaler  
import uvicorn  
  
# Load saved models  
rf\_model = joblib.load("models/random\_forest\_model.pkl")  
iso\_forest = joblib.load("models/isolation\_forest\_model.pkl")  
autoencoder = tf.keras.models.load\_model("models/autoencoder\_model")  
  
app = FastAPI(title="Real-Time Fraud Detection Service")  
  
# Define the input schema using Pydantic  
class Transaction(BaseModel):  
 HourOfDay: float  
 Amount\_scaled: float  
 RollingMeanAmount: float  
 V: list  
  
def combine\_fraud\_scores(rf\_probs, iso\_scores, ae\_errors, w\_rf=0.5, w\_if=0.3, w\_ae=0.2):  
 rf\_probs = np.asarray(rf\_probs)  
 iso\_scores = np.asarray(iso\_scores)  
 ae\_errors = np.asarray(ae\_errors)  
   
 scaler\_if = MinMaxScaler()  
 iso\_scores\_scaled = scaler\_if.fit\_transform(iso\_scores.reshape(-1, 1)).flatten()  
   
 scaler\_ae = MinMaxScaler()  
 ae\_errors\_scaled = scaler\_ae.fit\_transform(ae\_errors.reshape(-1, 1)).flatten()  
   
 final\_scores = (w\_rf \* rf\_probs) + (w\_if \* iso\_scores\_scaled) + (w\_ae \* ae\_errors\_scaled)  
 return final\_scores  
  
@app.post("/predict")  
def predict(transaction: Transaction):  
 try:  
 if len(transaction.V) != 28:  
 raise HTTPException(status\_code=400, detail="Expected 28 PCA component values in 'V' field.")  
   
 x\_input = np.array([[transaction.HourOfDay, transaction.Amount\_scaled, transaction.RollingMeanAmount] + transaction.V])  
   
 rf\_prob = rf\_model.predict\_proba(x\_input)[0, 1]  
 iso\_score = iso\_forest.decision\_function(x\_input)[0]  
 x\_pred = autoencoder.predict(x\_input)  
 ae\_error = np.mean((x\_input - x\_pred) \*\* 2)  
   
 final\_score = combine\_fraud\_scores([rf\_prob], [iso\_score], [ae\_error])[0]  
 fraud\_threshold = 0.6  
 fraud\_label = final\_score > fraud\_threshold  
   
 return {  
 "rf\_probability": rf\_prob,  
 "isolation\_score": iso\_score,  
 "autoencoder\_error": ae\_error,  
 "combined\_fraud\_score": final\_score,  
 "fraud\_prediction": bool(fraud\_label)  
 }  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=str(e))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 uvicorn.run(app, host="0.0.0.0", port=8000)

## 2. Docker Containerization

To deploy the FastAPI microservice in a Docker container, use the following Dockerfile:

### Dockerfile

FROM python:3.9-slim  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --upgrade pip && pip install -r requirements.txt  
COPY . .  
EXPOSE 8000  
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]

## 3. Kafka Integration for Streaming

To integrate the microservice with Kafka for real-time streaming, use the following consumer code:

### Kafka Consumer

from kafka import KafkaConsumer  
import json  
import requests  
  
consumer = KafkaConsumer(  
 'transactions\_topic',  
 bootstrap\_servers=['your\_kafka\_server:9092'],  
 auto\_offset\_reset='latest',  
 enable\_auto\_commit=True,  
 group\_id='fraud\_detection\_group',  
 value\_deserializer=lambda x: json.loads(x.decode('utf-8'))  
)  
  
api\_url = "http://localhost:8000/predict"  
  
for message in consumer:  
 transaction\_data = message.value  
 response = requests.post(api\_url, json=transaction\_data)  
 result = response.json()  
 print("Prediction:", result)

## 4. Latency Optimization

To ensure low-latency responses (~50ms per request), apply the following optimizations:

• Profile the pipeline using Python's `cProfile` or monitoring tools like Prometheus and Grafana.

• Optimize models using quantization techniques or TensorFlow Lite for reduced computation time.

• Deploy the microservice using Kubernetes to dynamically scale based on load.

• Implement asynchronous processing in FastAPI for non-blocking execution.

## 5. Summary

This document outlines a real-time fraud detection microservice using FastAPI, Docker, and Kafka. The microservice integrates trained fraud detection models, provides a RESTful API for predictions, and is designed for deployment in scalable environments. With Kafka integration, it can seamlessly process streaming transactions, offering real-time fraud detection.