# End-to-End Fraud Detection Model Deployment on AWS using FastAPI and Docker

This document provides a summary of the deployment of a fraud detection LSTM model as a REST API   
using FastAPI, Docker, and AWS EC2. The screenshots included serve as evidence of the successful end-to-end deployment,   
testing, and cloud cost management process.

## 1. AWS EC2 Instance Setup

An AWS EC2 t3.micro instance was configured in the Asia Pacific (Sydney) region using Amazon Linux 2023.   
Security groups were set up to allow inbound HTTP (port 80) and SSH (port 22) traffic.   
The instance was used to host the containerized FastAPI application.  
A screenshot of a computer

AI-generated content may be incorrect.

## 2. SSH Connection and Docker Deployment

Deployment was performed through SSH access to the EC2 instance. Project files were transferred to the server,   
and the Docker image was built and run using the defined requirements and service scripts. The container exposed   
the FastAPI application on port 80 for public access.  
A screenshot of a computer screen

AI-generated content may be incorrect.

## 3. FastAPI Server Startup Logs

The Docker container logs confirm successful startup of the FastAPI service and correct model initialization.   
The LSTM model and preprocessing pipeline were loaded successfully, with parameters T=32 (timesteps) and F=17 (encoded features).

A screen shot of a computer program

AI-generated content may be incorrect.

## 4. API Health Endpoint Verification

The /health endpoint verified that the API was operational, confirming correct model artifacts, timesteps,   
feature dimensions, and decision threshold. Both PowerShell and browser tests returned successful JSON responses.

A computer screen with white text

AI-generated content may be incorrect.

## 5. Environment and Version Verification

Inside the container, installed library versions were confirmed to match the training environment.   
TensorFlow (2.20.0), scikit-learn (1.6.1), and FastAPI (0.119.0) were verified to ensure full compatibility   
between development and production.

## 6. Successful Fraud Prediction Request

A real transaction payload was sent via Swagger UI and PowerShell to the /predict endpoint.   
The API successfully encoded inputs, built the input sequence, and returned a JSON response containing   
the predicted fraud probability, binary decision, and threshold value.

## 7. Python Client Validation

A Python client was also used to send a POST request to the /predict endpoint. The response matched the   
results from Swagger UI and PowerShell, confirming consistent model inference across interfaces.

A close-up of a text

AI-generated content may be incorrect.

## 8. EC2 Instance Shutdown

After deployment validation, the EC2 instance was stopped to prevent further billing.   
This follows cloud cost management best practices, ensuring AWS compute charges apply only while the   
instance is active.

## 9. Conclusion

The fraud detection model was successfully deployed and tested on AWS EC2 using FastAPI and Docker.   
The project demonstrates real-time model inference, environment consistency, and cost-efficient cloud deployment.   
This workflow reflects key MLOps principles including version control, containerization, and scalable API hosting.