**MINISTRY OF EDUCATION**

**FPT UNIVERSITY**



**HOANG THI HOAI THUONG**

**MLFLOW CLASSIFICATION PROJECT DOCUMENTATION**

**FIELD/MAJOR: ARTIFICIAL INTELLIGENCE**

**CLASS: MSE.20HCM**

**SUBJECT: AI in Production DevOps, DataOps, MLOps**

**INSTRUCTOR**

**Huynh Cong Viet Ngu**

HO CHI MINH CITY, 2025

**TABLE OF CONTENTS**

[MLflow Classification Project Documentation 3](#_Toc3504)

[1. Project Overview 3](#_Toc6158)

[2. Machine Learning Model 3](#_Toc12211)

[2.1 Model Description 3](#_Toc32244)

[2.2 Code Summary 3](#_Toc26446)

[2.3 Hyperparameters 4](#_Toc13276)

[2.4 MLflow Integration 5](#_Toc26002)

[2.5 Model Comparison 5](#_Toc14016)

[3. Flask Web Application 6](#_Toc19192)

[3.1 Code Summary (app.py) 6](#_Toc1781)

[3.2 Key Features 6](#_Toc31036)

[4. Deployment Setup 7](#_Toc16161)

[4.1 Docker Configuration 7](#_Toc10488)

[4.2 AWS Deployment Architecture 7](#_Toc17360)

[5. Deployment Steps on AWS 8](#_Toc2399)

[5.1 Prerequisites 8](#_Toc22751)

[5.2 Step-by-Step Deployment 8](#_Toc8836)

[6. Conclusion 13](#_Toc4887)

## **MLflow Classification Project Documentation**

### **1. Project Overview**

This project demonstrates the use of MLflow to manage a machine learning workflow for a binary classification task. It includes data generation, model training, hyperparameter tuning, model evaluation, model registration, and deployment via a Flask web application. The project is containerized using Docker and deployed on AWS EC2 instances with an RDS PostgreSQL database and S3 storage.

### **2. Machine Learning Model**

#### **2.1 Model Description**

* **Purpose**: The model performs binary classification on synthetic data generated using sklearn.datasets.make\_classification.
* **Algorithm**: Logistic Regression, a simple yet effective linear model for binary classification.
* **Functionality**: The model predicts a binary outcome (0 or 1) based on 20 input features.

#### **2.2 Code Summary**

The core script (classifier.py) performs the following:

* **Data Generation**: Creates a synthetic dataset with 1000 samples, 20 features, and 2 classes using make\_classification.
* **Data Splitting**: Splits data into 80% training and 20% testing sets.
* **Model Training**: Trains a Logistic Regression model with varying hyperparameters.
* **Hyperparameter Tuning**: Tests combinations of hyperparameters C and solver.
* **Evaluation**: Measures model performance using accuracy and F1-score.
* **Model Logging**: Logs parameters, metrics, and models to MLflow.
* **Model Registration**: Registers the model with the highest F1-score as "BestClassificationModel".

#### **2.3 Hyperparameters**

The hyperparameters tuned in the model are:

* **C** (Inverse of regularization strength):
  + Values: [0.1, 1.0, 10.0]
  + Meaning: Controls the trade-off between fitting the training data and keeping the model simple. Smaller values increase regularization, reducing overfitting but potentially underfitting.
* **solver** (Optimization algorithm):
  + Values: ['lbfgs', 'liblinear']
  + Meaning:
    - lbfgs: Uses Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm, suitable for larger datasets.
    - liblinear: Uses a coordinate descent algorithm, effective for smaller datasets or when L1 regularization is needed.

#### **2.4 MLflow Integration**

* **Purpose of MLflow**: MLflow is used to track experiments, log parameters and metrics, store models, and manage the model lifecycle.
* **Implementation**:
  + **Tracking URI**: Set to a SQLite database (sqlite:///mlflow.db) or PostgreSQL for persistence.
  + **Experiment Setup**: Creates an experiment named "classification\_experiment" to group runs.
  + **Logging**: Logs hyperparameters (C, solver), metrics (accuracy, f1\_score), and the trained model for each run.
  + **Model Registration**: The best model (highest F1-score) is registered in the MLflow Model Registry as "BestClassificationModel".
  + **Comparison**: Retrieves and displays all runs to compare performance metrics.

#### **2.5 Model Comparison**

* The script compares models based on accuracy and f1\_score across all hyperparameter combinations.
* Output: A table showing run\_id, C, solver, accuracy, and f1\_score for each run.

### **3. Flask Web Application**

#### **3.1 Code Summary (app.py)**

The Flask application (app.py) serves as the interface to interact with the best registered model. Its functionalities include:

* **Model Loading**: Loads the registered model ("BestClassificationModel") from MLflow.
* **Input Validation**: Ensures input features are numerical, have exactly 20 values, and are free of NaN or infinite values.
* **Web Interface**:
  + **GET Request**: Renders an HTML form (index.html) for users to input 20 features.
  + **POST Request**: Processes input features, makes predictions, and displays results.
* **API Endpoint** (/api/predict):
  + Accepts JSON input with 20 features.
  + Returns JSON response with prediction (0 or 1) or error message.
* **Error Handling**: Displays errors for invalid inputs or model loading issues.

#### **3.2 Key Features**

* **User-Friendly Interface**: Allows users to input features via a web form and view predictions.
* **API Access**: Provides a RESTful API for programmatic access to predictions.
* **Robust Validation**: Ensures inputs are valid to prevent prediction errors.

### **4. Deployment Setup**

#### **4.1 Docker Configuration**

The project is containerized using a Dockerfile:

* **Base Image**: python:3.11-slim for a lightweight environment.
* **Steps**:
  + Copies requirements.txt and installs dependencies.
  + Copies app.py, classifier.py, templates/, and start.sh.
  + Runs classifier.py to train and register the model.
  + Sets executable permissions for start.sh.
  + Exposes ports 8001 (Flask) and 5000 (MLflow).
  + Executes start.sh to start the application.

#### **4.2 AWS Deployment Architecture**

The project is deployed across two EC2 instances, with an RDS PostgreSQL database and S3 for model storage:

* **EC2 Instance 1**: Runs the Flask application (app.py).
* **EC2 Instance 2**: Runs the MLflow tracking server.
* **RDS PostgreSQL**: Stores MLflow experiment and run data.
* **S3 Bucket**: Stores MLflow artifacts (e.g., trained models).

### **5. Deployment Steps on AWS**

#### **5.1 Prerequisites**

* AWS account with access to EC2, RDS, S3, and IAM.
* Docker installed locally to build and push images.
* AWS CLI configured with appropriate credentials.

#### **5.2 Step-by-Step Deployment**

##### **Step 1: Set Up S3 Bucket**

* Create an S3 bucket mlflow-classifier-artifacts for MLflow artifacts.
* Configure bucket policy to allow access from both EC2 instances:

JSON

{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Principal": {  
 "AWS": "<EC2-Instance-Role-ARN>"  
 },  
 "Action": ["s3:GetObject", "s3:PutObject", "s3:ListBucket"],  
 "Resource": [  
 "arn:aws:s3:::mlflow-classifier-artifacts",  
 "arn:aws:s3:::mlflow-classifier-artifacts/\*"  
 ]  
 }  
 ]  
}

* Note the bucket name for MLflow configuration.

##### **Step 2: Set Up RDS PostgreSQL**

* Launch an RDS instance with PostgreSQL:
  + Engine: PostgreSQL 17 for this project.
  + Instance type: db.t3.micro (for testing).
  + Database name: mlflow\_db.
  + Username: mlflowuser.
  + Password: xxxxxx.
* Configure security group to allow inbound traffic on port 5432 from both EC2 instances.
* Note the RDS endpoint (e.g., mlflow-classifier-db.creo6q2s4q75.ap-southeast-1.rds.amazonaws.com).

##### **Step 3: Create IAM Role for EC2**

* Create an IAM role (EC2MLflowRole) with permissions for:
  + S3 access (AmazonS3FullAccess or custom policy for mlflow-classifier-artifacts).
  + RDS access (if needed for monitoring).
* Attach the role to both EC2 instances.

##### **Step 4: Build and Push Docker Image**

* Create requirements.txt:

mlflow==2.16.0  
scikit-learn==1.5.1  
flask==3.0.3  
numpy==1.26.4  
pandas==2.2.2  
psycopg2-binary==2.9.9  
boto3==1.34.0

* Create start.sh:

Bash

#!/bin/bash  
mlflow server --backend-store-uri postgresql://mlflowuser:Mlflow123!@<RDS-ENDPOINT>:5432/mlflow\_db \  
 --default-artifact-root s3://mlflow-classifier-artifacts \  
 --host 0.0.0.0 --port 5000 & # Run MLflow server (EC2 Instance 2)  
python app.py # Run Flask app (EC2 Instance 1)

* Build Docker image:

Bash

docker build -t mlflow-classifier:latest .

* Push to Amazon ECR:
  + Create an ECR repository (mlflow-classifier).
  + Authenticate Docker to ECR:

Bash

aws ecr get-login-password --region ap-southeast-1 | docker login --username AWS --password-stdin <AWS-ACCOUNT-ID>.dkr.ecr.ap-southeast-1.amazonaws.com

* + Tag and push:

Bash

docker tag mlflow-classifier:latest <AWS-ACCOUNT-ID>.dkr.ecr.ap-southeast-1.amazonaws.com/mlflow-classifier:latest  
docker push <AWS-ACCOUNT-ID>.dkr.ecr.ap-southeast-1.amazonaws.com/mlflow-classifier:latest

##### **Step 5: Launch EC2 Instance 1 (Flask App)**

* Launch an EC2 instance:
  + AMI: Amazon Linux 2 or Ubuntu 20.04.
  + Instance type: t2.micro (for testing).
  + Attach IAM role EC2MLflowRole.
  + Security group: Allow inbound traffic on port 8001 (HTTP).
* Install Docker:

Bash

sudo apt update -y  
sudo apt-get install docker.io -y  
sudo service docker start  
sudo usermod -a -G docker ubuntu

* Pull and run Docker image:

Bash

aws ecr get-login-password --region ap-southeast-1 | docker login --username AWS --password-stdin <AWS-ACCOUNT-ID>.dkr.ecr.ap-southeast-1.amazonaws.com  
docker run -d -p 8001:8001 \  
 -e MLFLOW\_TRACKING\_URI=postgresql://mlflowuser:Mlflow123!@<RDS-ENDPOINT>:5432/mlflow\_db \  
 -e AWS\_ACCESS\_KEY\_ID=<YOUR-ACCESS-KEY> \  
 -e AWS\_SECRET\_ACCESS\_KEY=<YOUR-SECRET-KEY> \  
 <AWS-ACCOUNT-ID>.dkr.ecr.ap-southeast-1.amazonaws.com/mlflow-classifier:latest

* Update app.py to use the RDS tracking URI:

Python

mlflow.set\_tracking\_uri("postgresql://mlflowuser:Mlflow123!@<RDS-ENDPOINT>:5432/mlflow\_db")

##### **Step 6: Launch EC2 Instance 2 (MLflow Server)**

* Launch another EC2 instance with the same configuration as Instance 1.
* Security group: Allow inbound traffic on port 5000 (HTTP).
* Install Docker and pull the image as in Step 5.
* Run the MLflow server:

Bash

docker run -d -p 5000:5000 \  
 -e MLFLOW\_TRACKING\_URI=postgresql://mlflowuser:Mlflow123!@<RDS-ENDPOINT>:5432/mlflow\_db \  
 -e AWS\_ACCESS\_KEY\_ID=<YOUR-ACCESS-KEY> \  
 -e AWS\_SECRET\_ACCESS\_KEY=<YOUR-SECRET-KEY> \  
 <AWS-ACCOUNT-ID>.dkr.ecr.ap-southeast-1.amazonaws.com/mlflow-classifier:latest

##### **Step 7: Verify Deployment**

* Access the Flask app at **http://54.255.177.73:8001**
* Access the MLflow UI at **http://54.169.242.113:5000**
* Test predictions via the web interface or API:

Bash

curl -X POST http://<EC2-Instance-1-Public-IP>:8001/api/predict \  
-H "Content-Type: application/json" \  
-d '{"features": [0.1, 0.2, ..., 0.3]}' # 20 features

##### **Step 8: Monitor and Scale**

* Monitor EC2 instances using CloudWatch.
* Optionally, set up an Application Load Balancer for the Flask app to handle traffic.
* Back up RDS and S3 data periodically.

### **6. Conclusion**

This project demonstrates a complete machine learning pipeline using MLflow for experiment tracking and model management, integrated with a Flask web application for deployment. The AWS-based deployment leverages EC2 for compute, RDS for persistent storage, and S3 for artifact sharing, ensuring scalability and accessibility. The use of Docker simplifies deployment and ensures consistency across environments.