AIMLCZG523– Assignment I

Group ID : #115

Abstract

This document presents an end‑to‑end MLOps pipeline for Heart Disease Prediction using the UCI dataset. It covers data preparation, model development, and experiment tracking with modern MLOps tools. The solution incorporates CI/CD automation, local Kubernetes Docker Desktop deployment. Monitoring and logging ensure reliability in production. The workflow demonstrates a scalable, reproducible approach to full‑lifecycle machine‑learning operation.

**Group Members Name with Student ID**

|  |  |  |
| --- | --- | --- |
| Sr.No | Student Name | Student ID |
| 1 | MAJJIGI JAMBANNA | 2024AA05721 |
| 2 | NIRANJAN KUMAR SHARMA | 2024AA05405 |
| 3 | SRIDEVI THAKKU RAMANAN | 2024AA05402 |
| 4 | TRIPTI | 2024AA05593 |
| 5 | HARSHAL KISHORE PHADAS | 2024AA05139 |

Contents

[1. Problem Statement 3](#_Toc218633738)

[2. Setup/installation instructions 4](#_Toc218633739)

[2.1. Prerequisites 4](#_Toc218633740)

[2.2. Local Python Environment Setup 4](#_Toc218633741)

[2.3. Data and Artifact Initialization 4](#_Toc218633742)

[2.4. Containerized Environment (Docker Desktop) 4](#_Toc218633743)

[2.5. Development Tools Configuration 4](#_Toc218633744)

[2.6. Quick set of commands repository and UI Access 5](#_Toc218633745)

[2.7. Validation Results 5](#_Toc218633746)

[2.8. Link to code repository 5](#_Toc218633747)

[3. Architecture diagram 6](#_Toc218633748)

[4. Well-documented code Jupyter Notebook 8](#_Toc218633749)

[5. Data Acquisition & Exploratory Data Analysis (EDA) 8](#_Toc218633750)

[6. Experiment Tracking 9](#_Toc218633751)

[7. CI/CD Pipeline 10](#_Toc218633752)

[7.1. Implementation Using Jenkins and Docker 10](#_Toc218633753)

[7.2. Jenkins Pipeline Stages 10](#_Toc218633754)

[8. Model Containerization 14](#_Toc218633755)

[9. Screenshots of test cases from API 15](#_Toc218633756)

[10. Monitoring & Logging 16](#_Toc218633757)

[11. Additional screenshot for reference 17](#_Toc218633758)

### Problem Statement

**MLOps Experimental Learning Assignment:** End-to-End ML Model Development, CI/CD, and Production Deployment Experimental Learning.

**Objective:**

Design, develop, and deploy a scalable and reproducible machine learning solution utilising modern MLOps best practices. The assignment emphasises practical automation, experiment tracking, CI/CD pipelines, containerization, cloud deployment, and monitoring—mirroring real-world production scenarios.

**Dataset:** Title: Heart Disease UCI Dataset

**Source**: UCI Machine Learning Repository

Note : CSV containing 14+ features (age, sex, blood pressure, cholesterol, etc.) and a binary target (presence/absence of heart disease).

**Problem Statement:** Build a machine learning classifier to predict the risk of heart disease based on patient health data, and deploy the solution as a cloud-ready, monitored API.

**Production-Readiness Requirements**

* All scripts must be executed from a clean setup using the requirements file.
* Model must serve correctly in an isolated environment (Docker; container build/test proof required).
* Pipeline must fail on code or test errors and give clear logs.

### Setup/installation instructions

Here are the Environment Setup Guidelines to set up the development and production environment for the Heart Disease MLOps assignment. Below setup ensures all components required for the MLOps lifecycle, Data acquisition, Training, Inference, and Monitoring are configured.

### Prerequisites

* **Python 3.9+** (Conda or Virtual environment recommended)
* **Docker & Docker Desktop** (For containerization)
* **Git** (For version control)

### Local Python Environment Setup

Create a dedicated environment and install dependencies to ensure consistency across development.

# Using Conda

conda create -n mlops\_heart python=3.9 -y

conda activate mlops\_heart

# Install required libraries

pip install -r requirements.txt

### Data and Artifact Initialization

Populated the local directory with the dataset and prepared for model training.

# Download the UCI Heart Disease dataset

python src/data\_downloader.py

# Run initial training to generate model artifacts and MLflow runs

python src/train.py

### Containerized Environment (Docker Desktop)

Set up the full MLOps stack (API, MLflow, Prometheus) [Dowlnload](https://www.docker.com/products/docker-desktop/) docker and installed.

# Build images and start all services

docker compose up --build -d

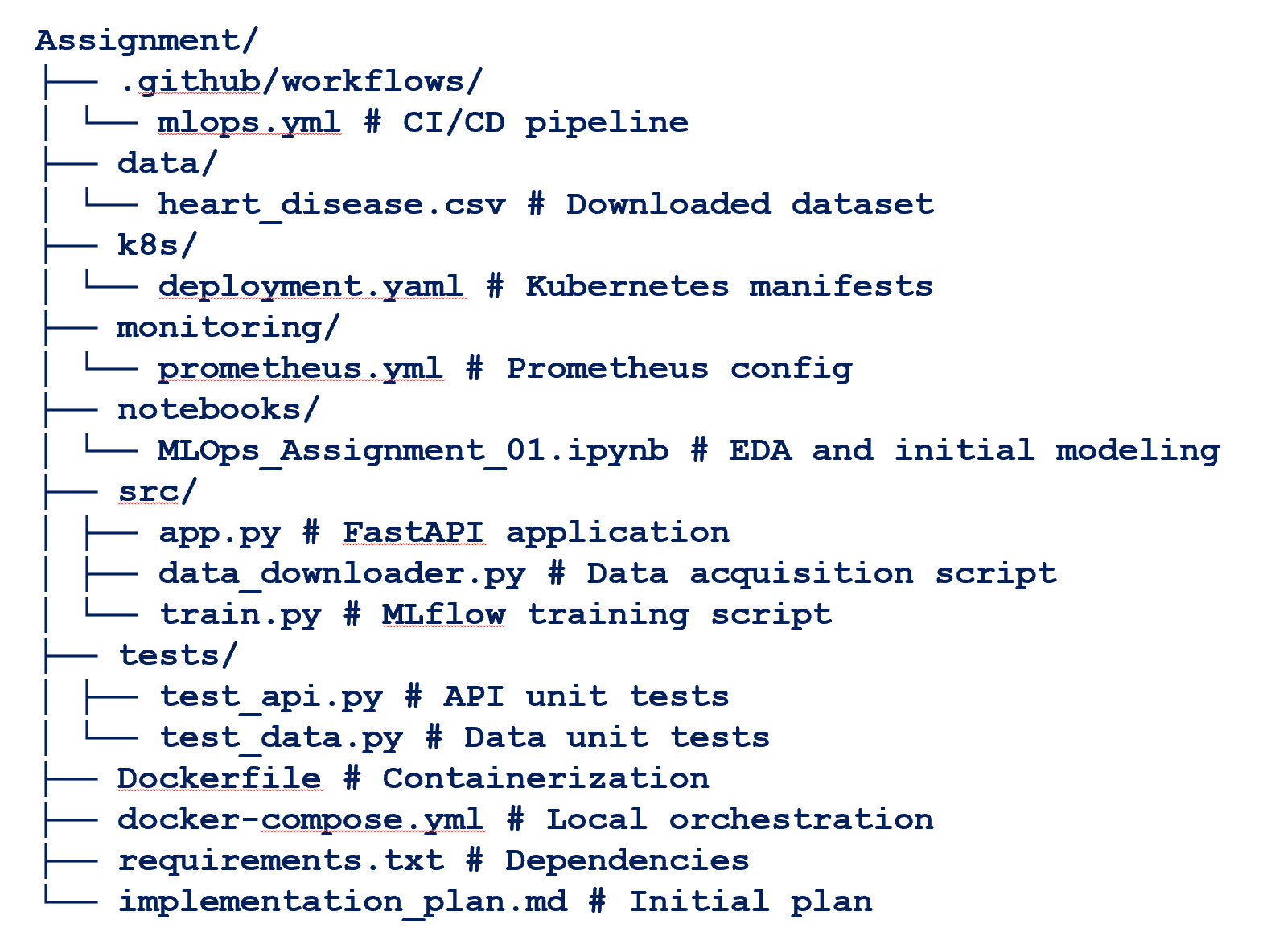
# Verify containers are running

docker ps

### Development Tools Configuration

* **MLflow**: Accessible at <http://127.0.0.1:5000> for experiment tracking.
* **Prometheus**: Accessible at <http://127.0.0.1:9090> for metrics monitoring.
* **API** : Access the API at API: <http://127.0.0.1:8000> for validating the API running state.
* **Testing**: Run python -m pytest tests/ to verify local logic or docker compose exec api python -m pytest tests/ for in-container validation

Below snapshot provides necessary files have been generated in the project directory.



### Quick set of commands repository and UI Access

|  |  |  |
| --- | --- | --- |
| Category | Instructions | Comments |
| Install Dependencies | pip install -r requirements.txt | Install all dependencies like pandas, numpy, scikit-learn, matplotlib,seaborn, mlflow, fastapi, uvicorn, pytest, httpx, pydantic,python-dotenv, flask, requests, pyyaml, prometheus-fastapi-instrumentator |
| Download Data | python src/data\_downloader.py | Download data from [Heart Disease - UCI Machine Learning Repository](https://archive.ics.uci.edu/dataset/45/heart+disease) and converted to .csv |
| Train the Model | python src/train.py | Train the model |
| Run API | uvicorn src.app:app --reload | Run the API which will start service at <http://127.0.0.1:8000> |
| Local Testing with Docker | docker compose up –build  Note for build : docker compose build | Launch the desktop docker instance UI to validate the training and testing the API and model test. |
| ML Flow user interface | mlflow ui | Use  to view the experiment history and compare different model runs and available  <http://127.0.0.1:5000> for experiment tracking. |
| Github commands leveraged | git status  git add .  git commit -m 'Production model version HD V1.0.01'  git push | Check the status and push the code |
| Jenkin | Configured the Jenkins for CI/CD | Configured Jenking and performed the Jenkin Job to build and deploy the code from [Github](https://github.com/jambannam/MLOPS/) |
| Starting the Prometheus grok | prometheus.exe --config.file="<systempath> \MTech\SEM III\MLOPS\Assignment\monitoring\prometheus.yml" | Start the Prometheus to monito the health of the service and access at <http://127.0.0.1:9090> |
| Unit testing / Testing | python -m pytest tests/ | To verify local logic or docker compose exec api python -m pytest tests/ for in-container validation.  Unit test also cam be performed through  $body = @{ age = 60.0; sex = 1.0; cp = 4.0; trestbps = 130.0; chol = 250.0; fbs = 0.0; restecg = 0.0; thalach = 150.0; exang = 0.0; oldpeak = 2.0; slope = 1.0; ca = 1.0; thal = 3.0} | ConvertTo-Json; Invoke-RestMethod -Method Post -Uri "<http://127.0.0.1:8000/predict>" -Body $body -ContentType "application/json" |

### Validation Results

* **Unit Tests** : Passed API status and data schema checks.
* **EDA** : Visualized clear correlations between features like thalach and age with the target.
* **Metrics** : Logistic regression, Random Forest achieved over ~80% accuracy on the test set.

### Link to code repository

|  |  |
| --- | --- |
| Repository | Details |
| mlops-heart-disease | <https://github.com/2024aa05593-trips/mlops-heart-disease> |

### Architecture diagram

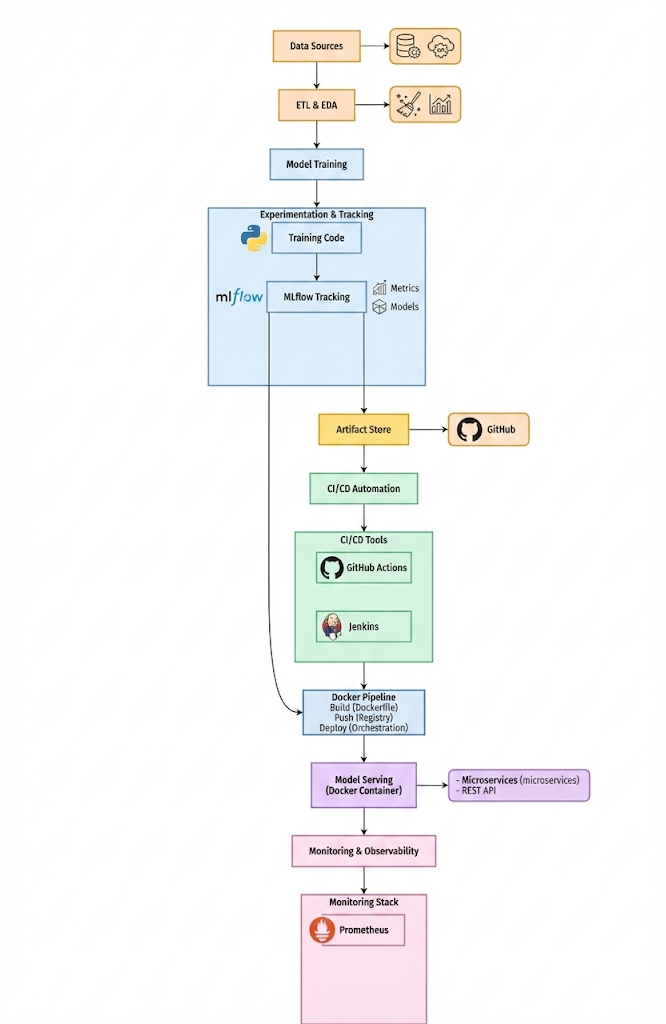


Fig 1 : Architecture diagram

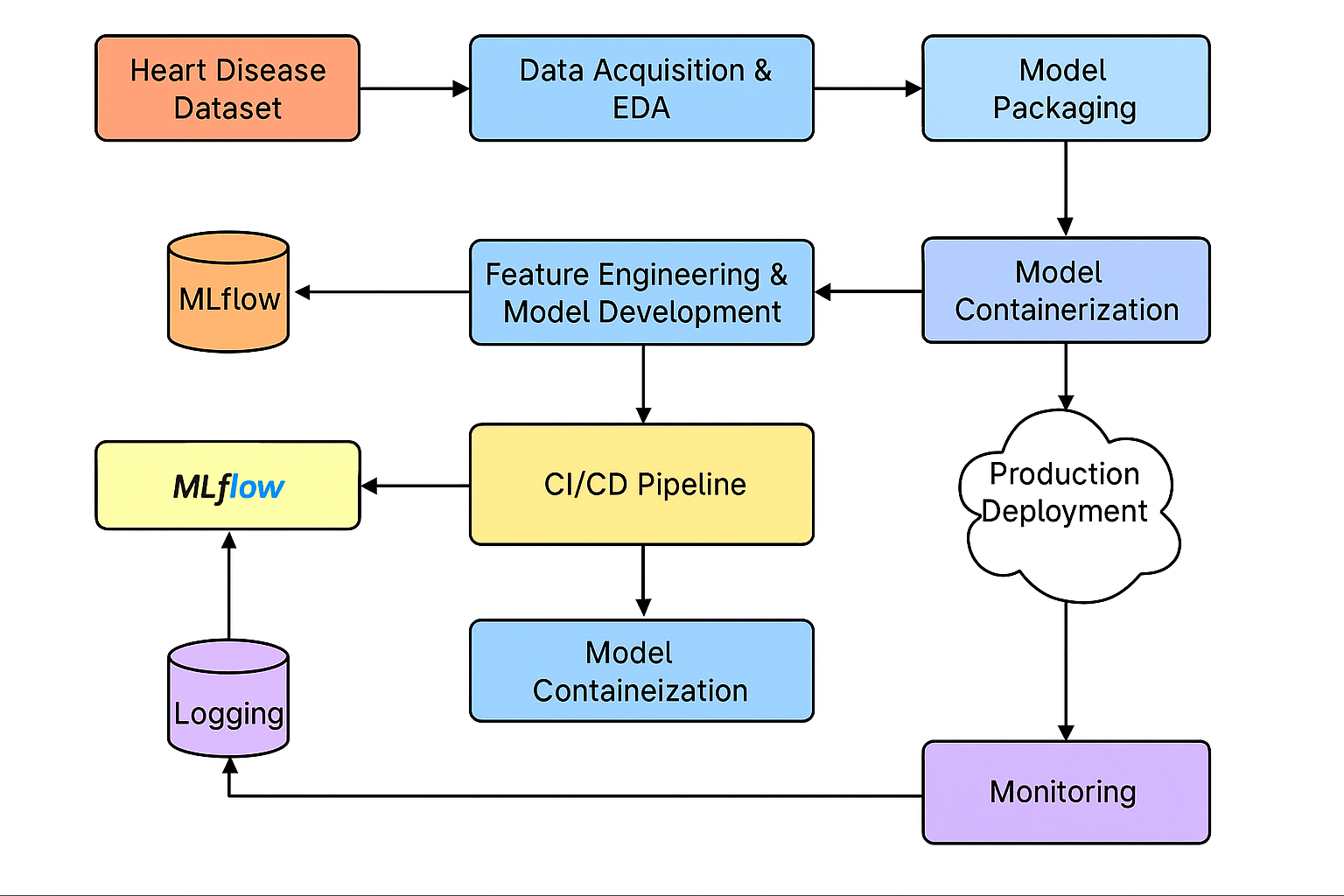


Fig 2 : End to end flow diagram

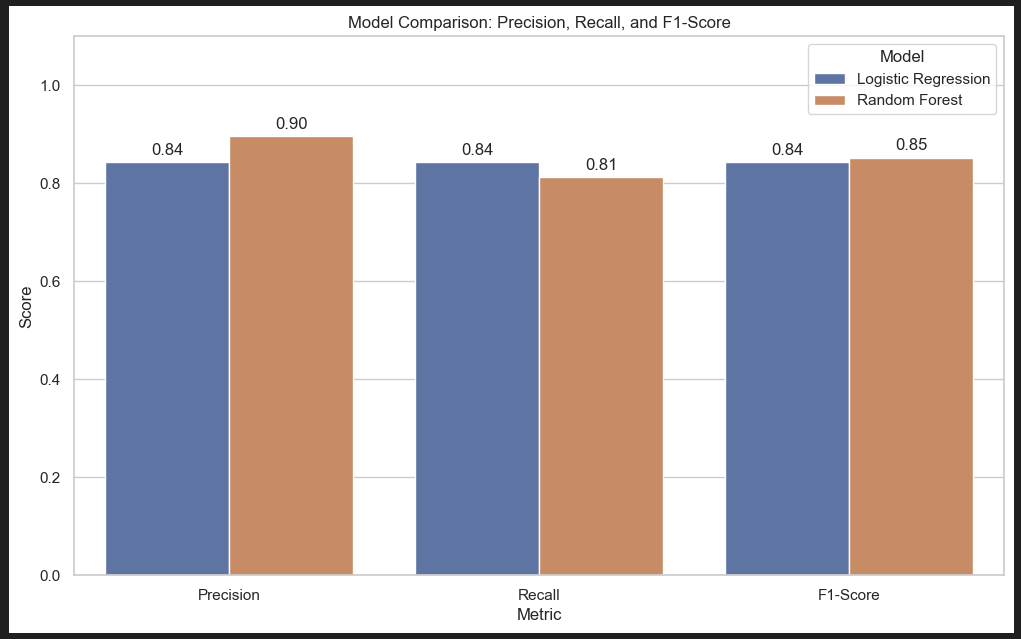
### Well-documented code Jupyter Notebook

The attached zip file contains assignment implementation an **end‑to‑end MLOps pipeline** for the *Heart Disease UCI Dataset* involving data acquisition, preprocessing, exploratory analysis, model development, experiment tracking, packaging, and result visualization. Two models—**Logistic Regression** and **Random Forest**—were trained and evaluated using metrics such as accuracy, precision, recall, F1‑score, and ROC‑AUC. The **Random Forest** model performed best and was packaged for reproducibility. The work includes MLflow experiment tracking, visual performance comparisons, and the generation of an HTML report using Jupyter Notebook. Both PDF and HTML outputs document the complete workflow, results, and code submitted as part of the compressed file.

Contents of attached zip file are

1. MLOps\_Assignment\_01.ipynb
2. MLOps\_Assignment\_01.pdf

Model performance for 20% test and 80% train data.



### Data Acquisition & Exploratory Data Analysis (EDA)

This phase automates the ingestion of the **UCI Heart Disease dataset** via a Python-based pipeline. It handles end-to-end data preparation—from raw URL retrieval to structured CSV storage and uses statistical visualization to identify key clinical predictors for modeling.

* **Auto-Ingestion**: Standalone script (***src/data\_downloader.py***) fetches raw data from UCI repository.
* **Robust Cleaning**: Implemented median imputation to handle missing categorical and clinical values.
* **Binary Labeling**: Simplified the multi-level target into a binary classification (Disease vs. No Disease).
* **Predictor Insights**: Correlation heatmaps revealed thalach (max heart rate) and cp (chest pain) as top predictors.
* **Statistical Profiling**: Distribution analysis verified feature consistency across demographic groups (Age/Sex)

Distribution of data sets

|  |  |
| --- | --- |
|  |  |

### Experiment Tracking

Here are the key highlights for your **Experiment Tracking & Model Training** section.

* **Centralized Tracking**: Performed 11+ training runs under the Heart\_Disease\_Prediction experiment.
* **Metric logging**: Automatically captured Accuracy (~85%), F1-Score, and ROC-AUC for every iteration.
* **Artifact Archival**: Saved production-ready model.pkl and performance plots for each run. Reproducibility: Linked binary models to specific code versions, ensuring a fully auditable pipeline.

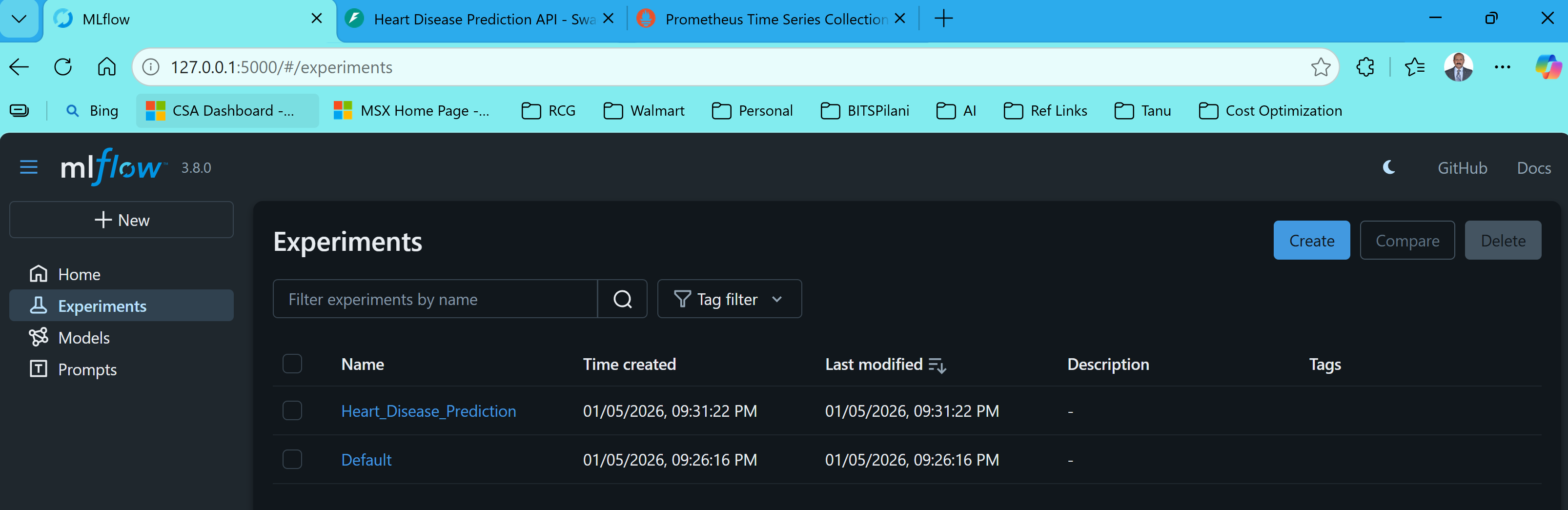


Fig : mlflow Experiments

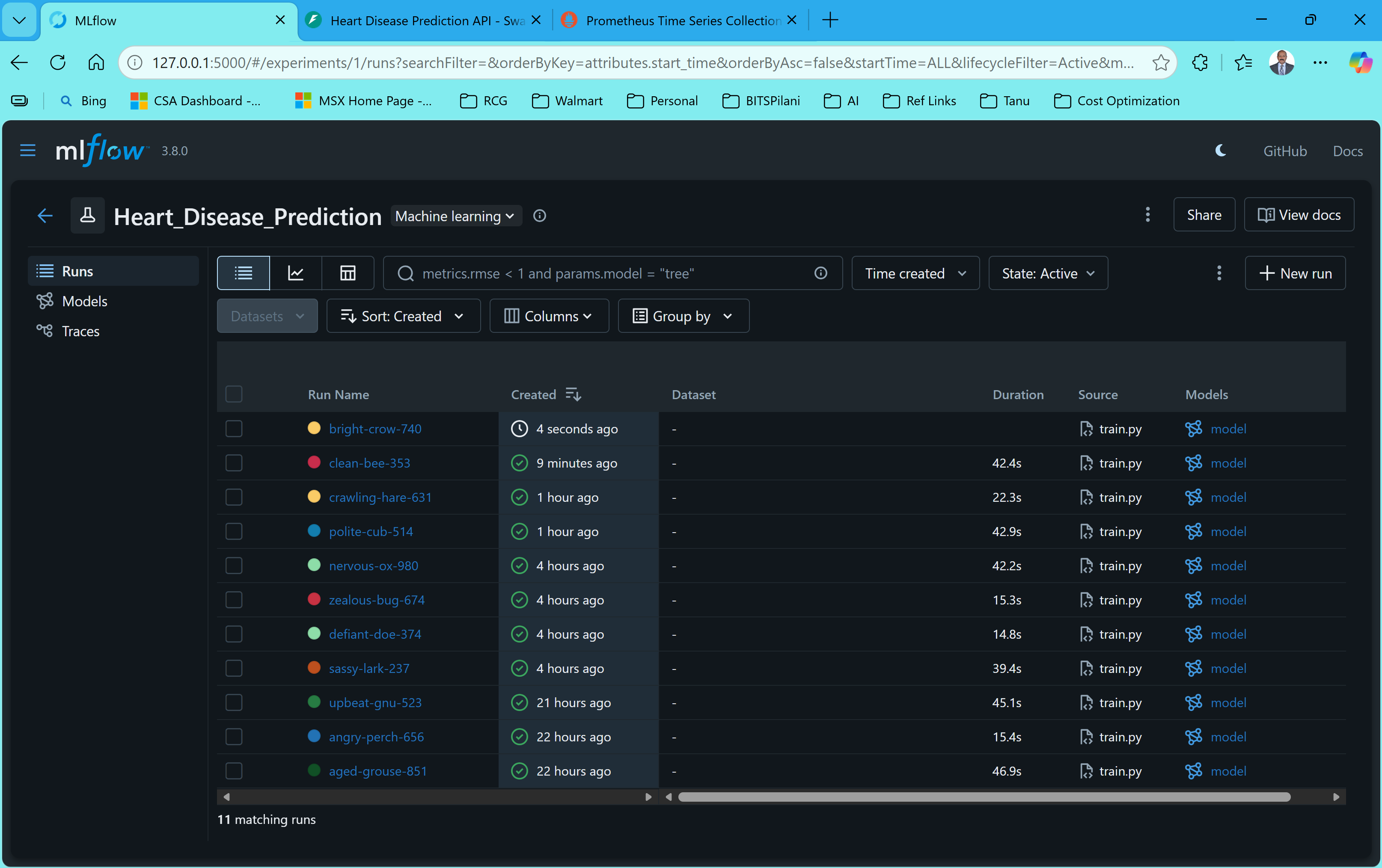


Fig : mlflow Disease Prediction

### CI/CD Pipeline

### Implementation Using Jenkins and Docker

The objective of this CI/CD pipeline is to automate the build, testing, containerization, and deployment process of the Heart Disease Prediction Machine Learning application, ensuring fast, reliable, and reproducible deployments with minimal manual intervention.

**Tools and Technologies Used**

* **GitHub** – Source Code Repository
* **Jenkins** – CI/CD Pipeline Orchestration
* **Docker & Docker Desktop** – Containerization and Application Execution
* **FastAPI** – REST API for model serving
* **Prometheus & Grafana** – Monitoring and Observability

**CI/CD Pipeline Workflow**

1. Developer commits and pushes latest changes to GitHub repository.
2. Jenkins automatically pulls the latest code from GitHub.
3. Jenkins builds a Docker image containing the FastAPI application and ML model.
4. Dependencies are installed and the environment is prepared automatically.
5. Jenkins deploys the Docker container and runs the application.
6. Prometheus and Grafana are configured to monitor the application.
7. Final running API becomes available at <http://localhost:8000>.

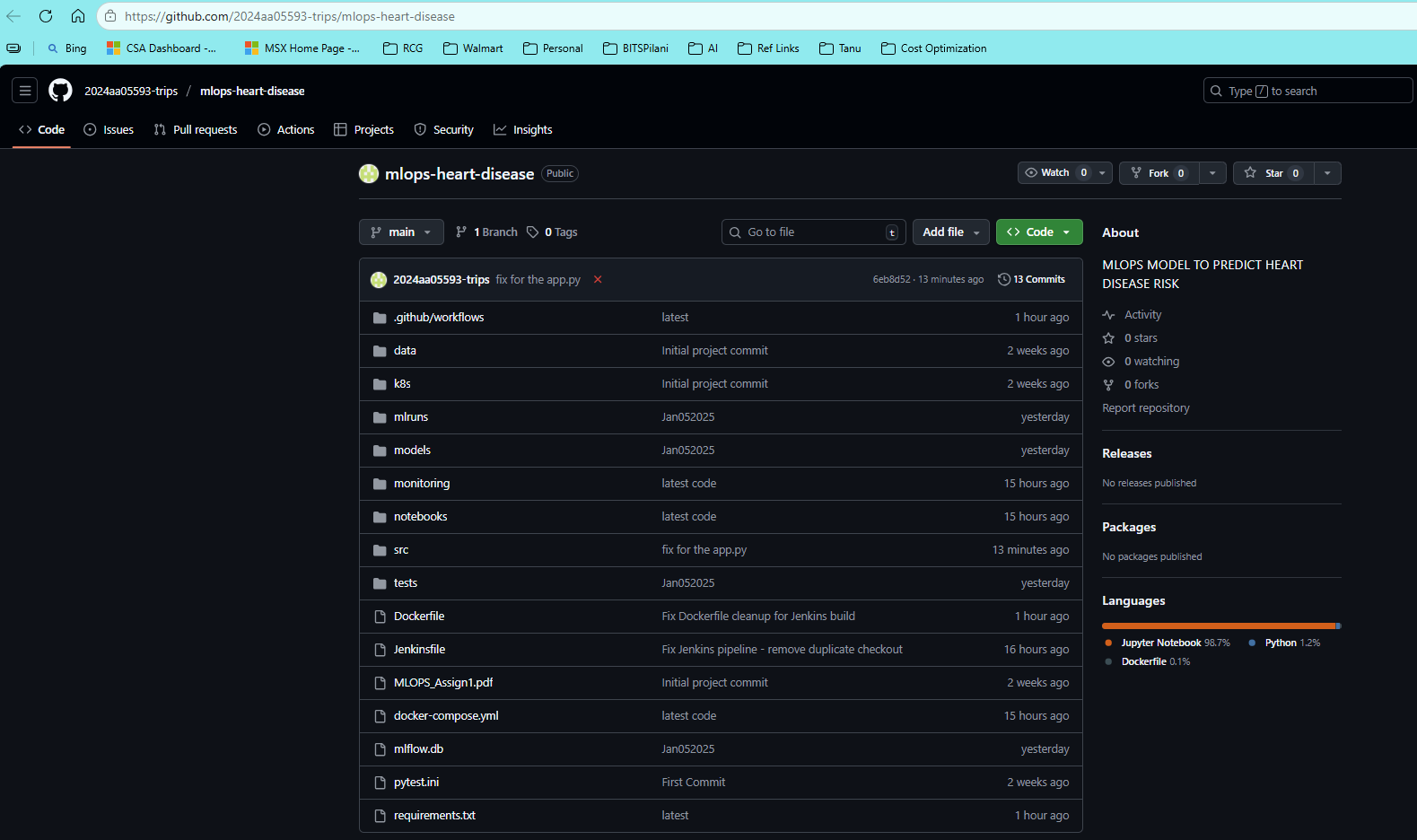
### Jenkins Pipeline Stages

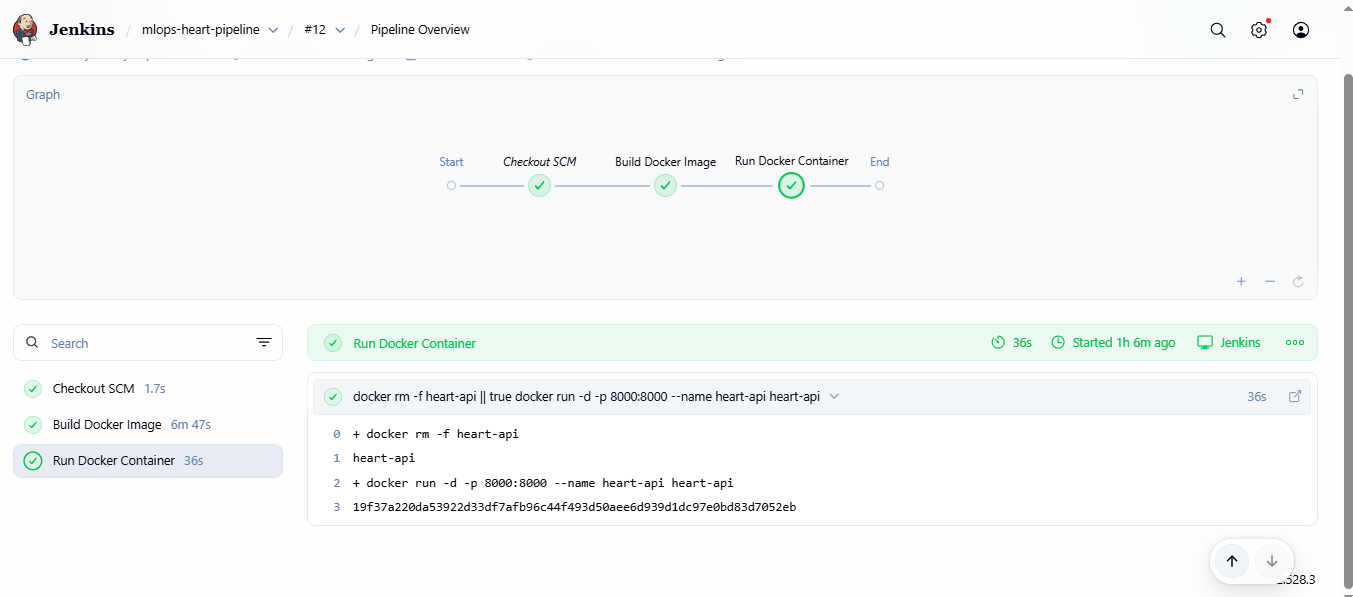
**1. Source Code Checkout (SCM)**

Jenkins fetches the latest code from GitHub automatically whenever the pipeline is triggered.  
**Outcome Achieved:**

* Automated centralized source fetching
* Eliminated manual download effort
* Ensured version-controlled builds

**Screenshot to Attach:** Jenkins console log showing SCM checkout completed successfully.





**2. Build Docker Image**

Jenkins builds the Docker image using the Dockerfile by executing:

docker build -t heart-api .

**Outcome Achieved:**

* Application is packaged with all dependencies
* Reproducible builds across systems
* Stable and isolated execution environment

**Screenshot to Attach:** Jenkins logs showing successful Docker build / Docker Desktop showing the heart-api image.



**3. Deploy and Run Application Container**

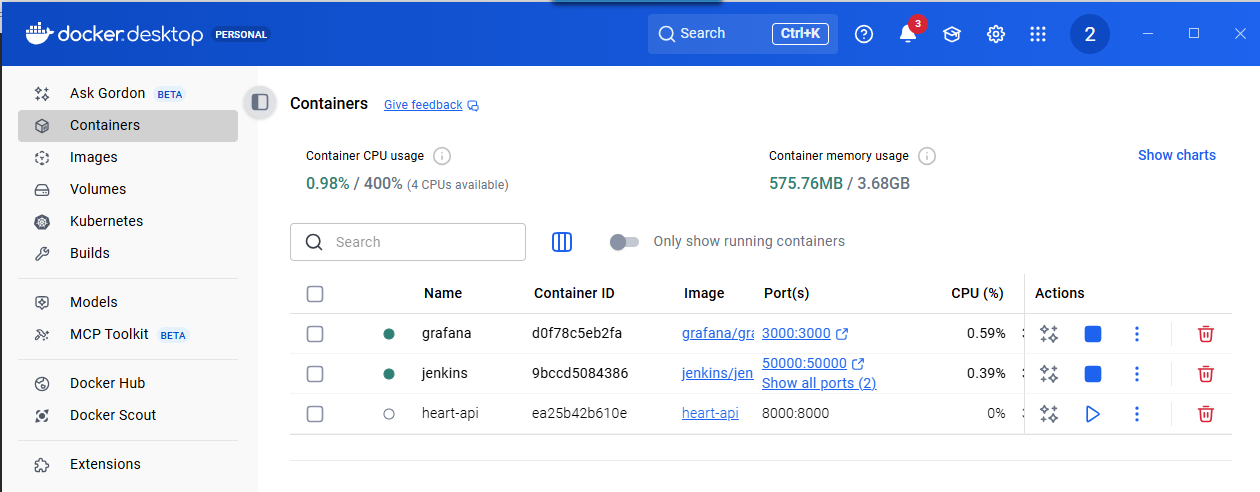
Jenkins deploys the application container using:

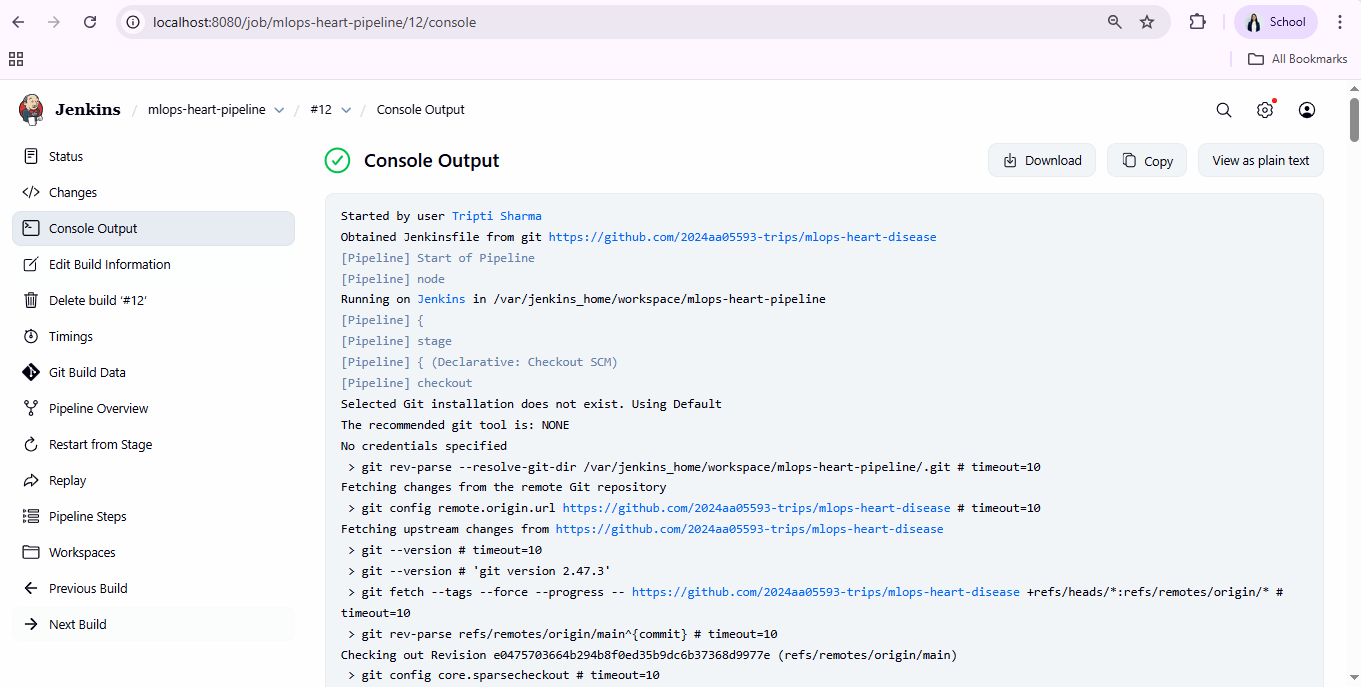
docker run -d -p 8000:8000 --name heart-api heart-api

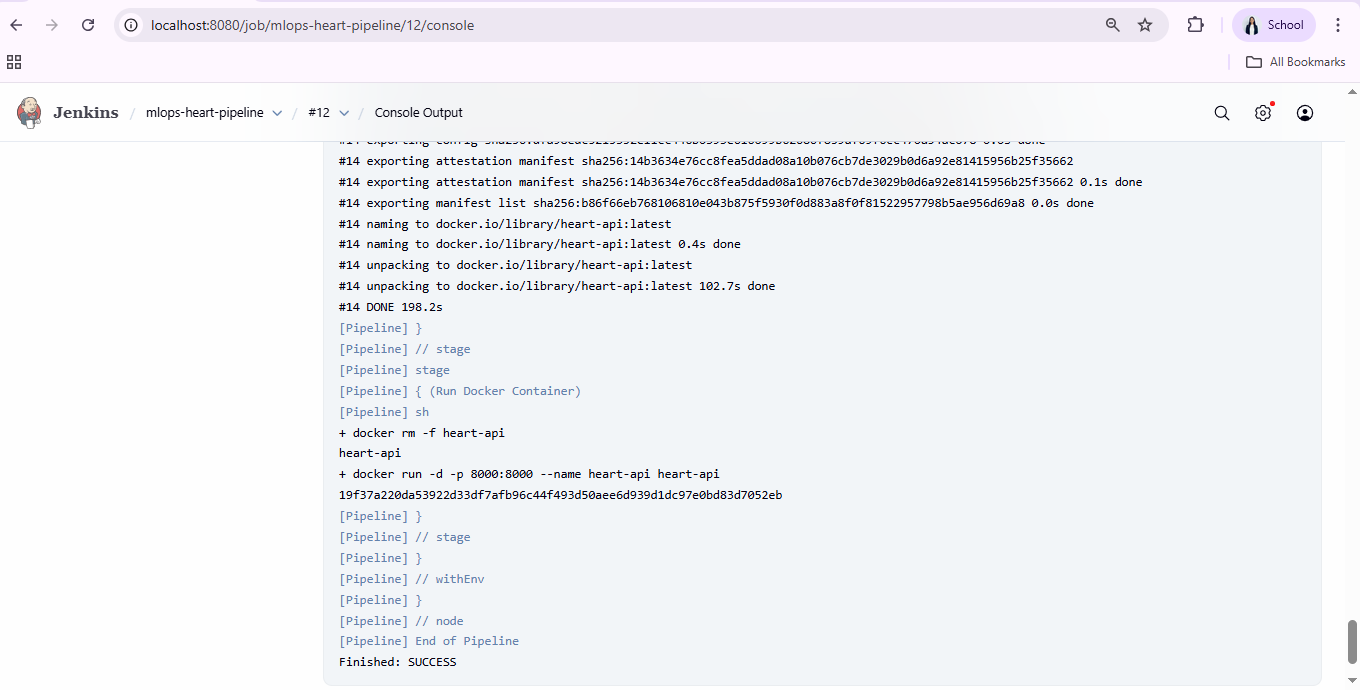
**Outcome Achieved:**

* Fully automated deployment
* FastAPI-based ML model served successfully
* Zero manual steps required during deployment

**Screenshot to Attach:** Running container in Docker Desktop / Jenkins success stage.







**Deployment Verification**

**1. API Health Check**

Opening the URL:

<http://localhost:8000>

Expected Output:

{"message": "Heart Disease Prediction API is running"}

**2. Swagger UI**

<http://localhost:8000/docs>

This interface allows us to test predictions interactively.

**Screenshot to Attach:** Swagger UI page.

**Monitoring and Observability**

Prometheus metrics are exposed at:

<http://localhost:8000/metrics>

Grafana dashboard is accessible at:

http://localhost:3000

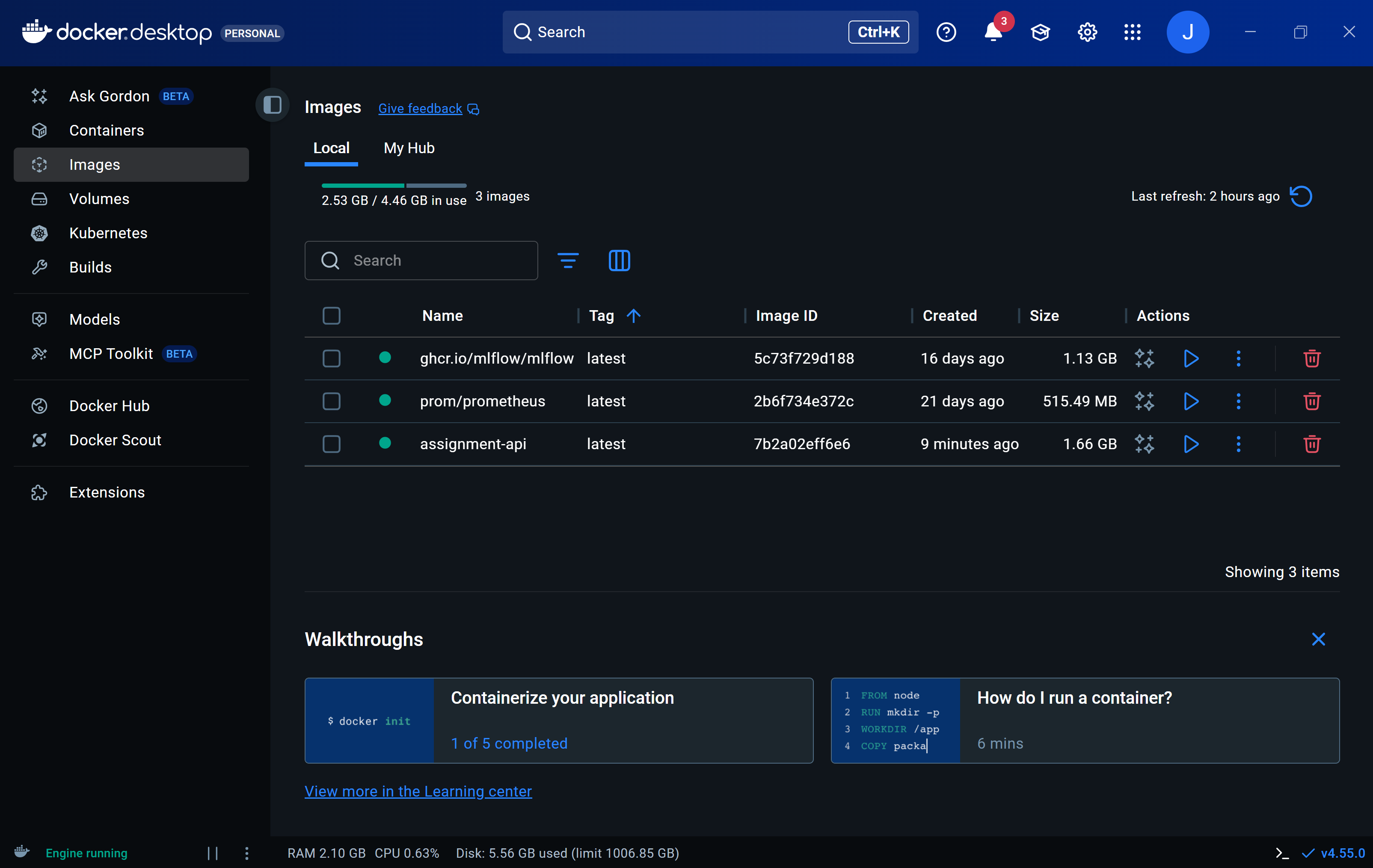
**Outcome Achieved:**

* API performance monitoring (latency, throughput, errors)
* Real-time health visibility of the service
* Production-style observability enabled

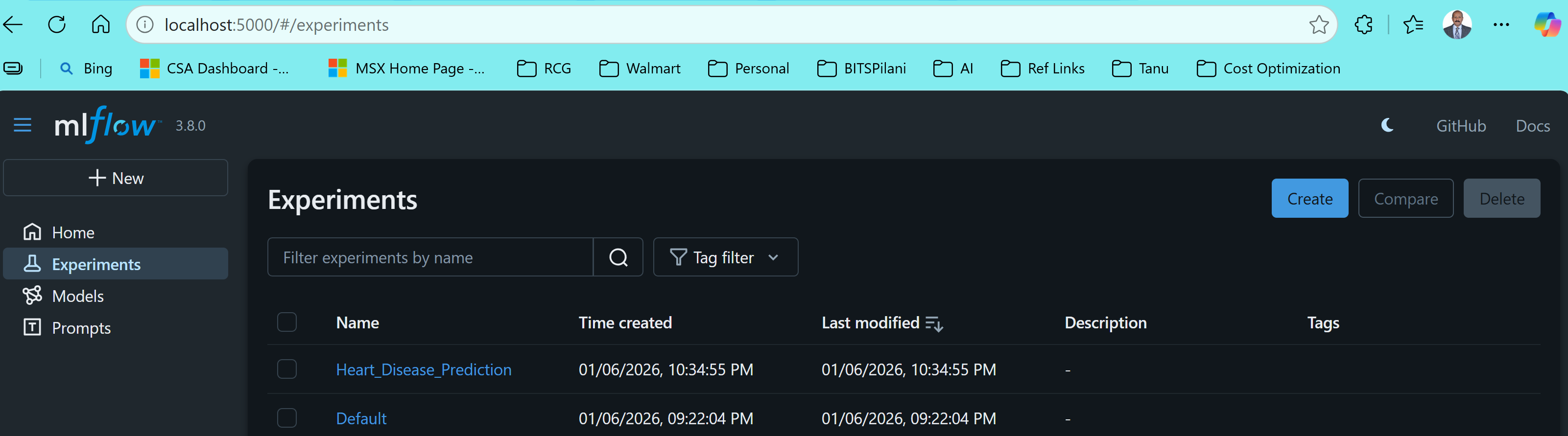
**Screenshots to Attach:**

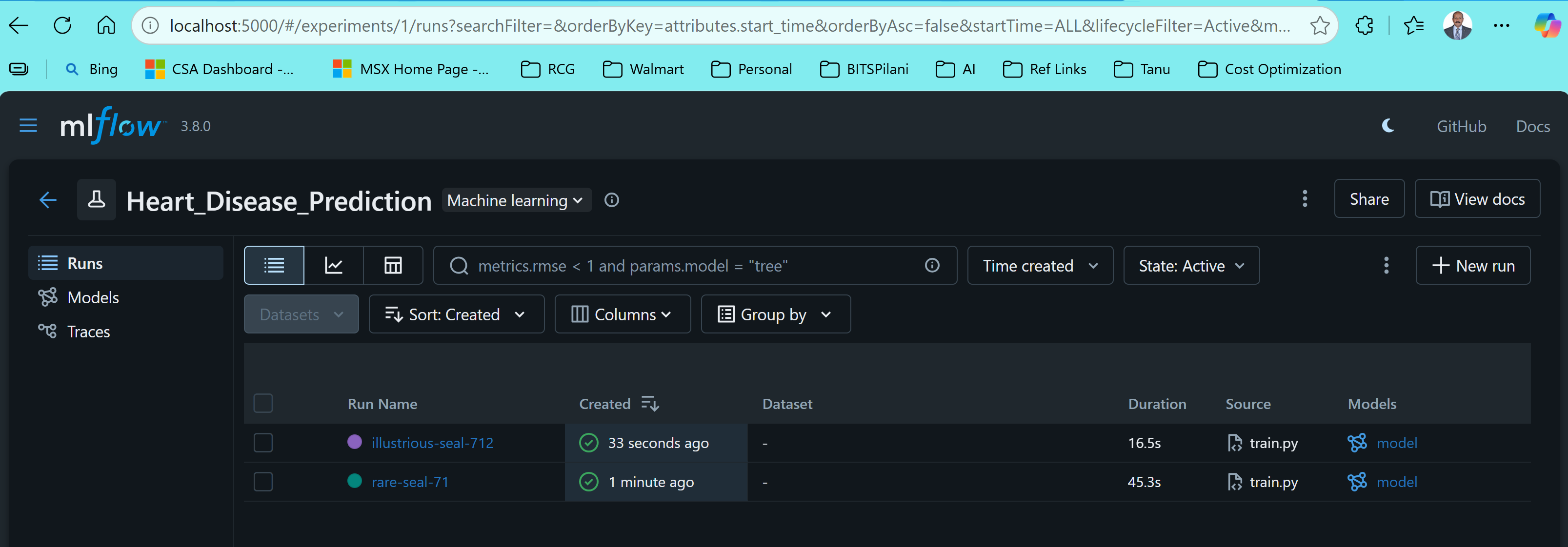
* Prometheus metrics page
* Grafana dashboard

### Model Containerization



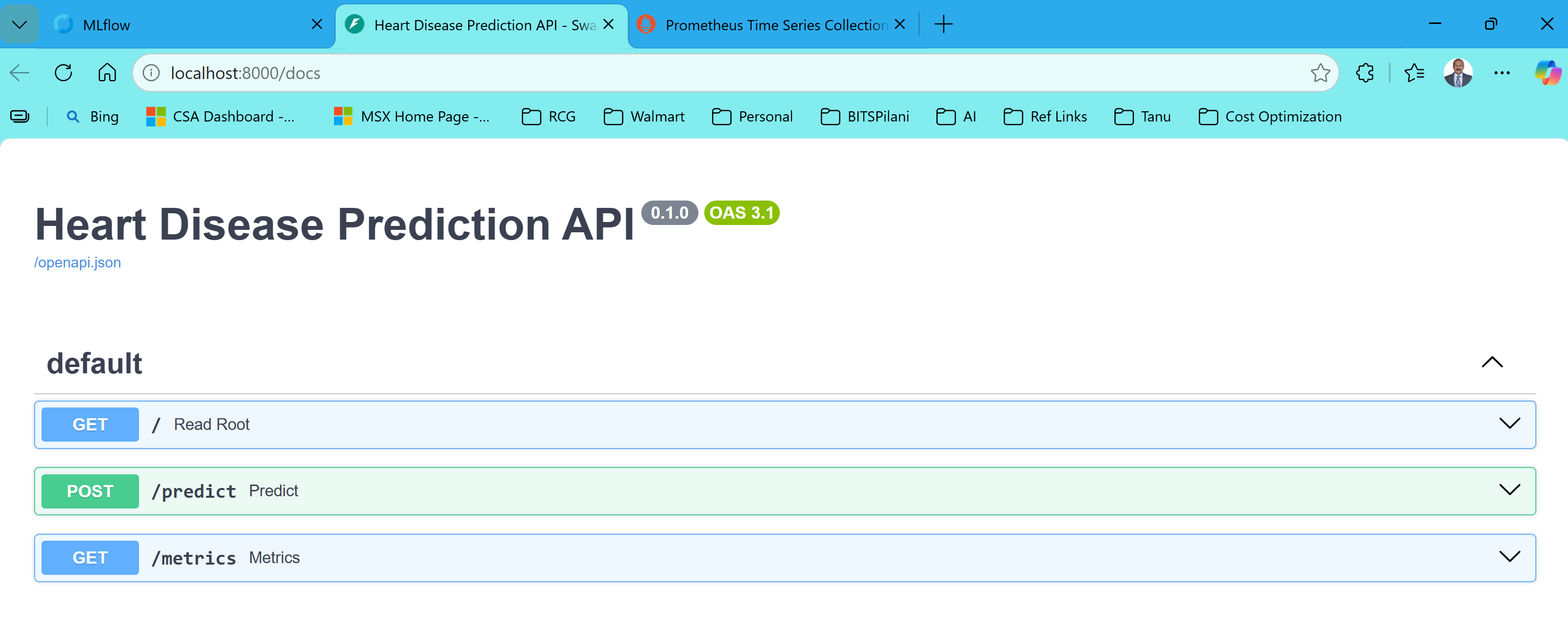
Experiment executed on docker image

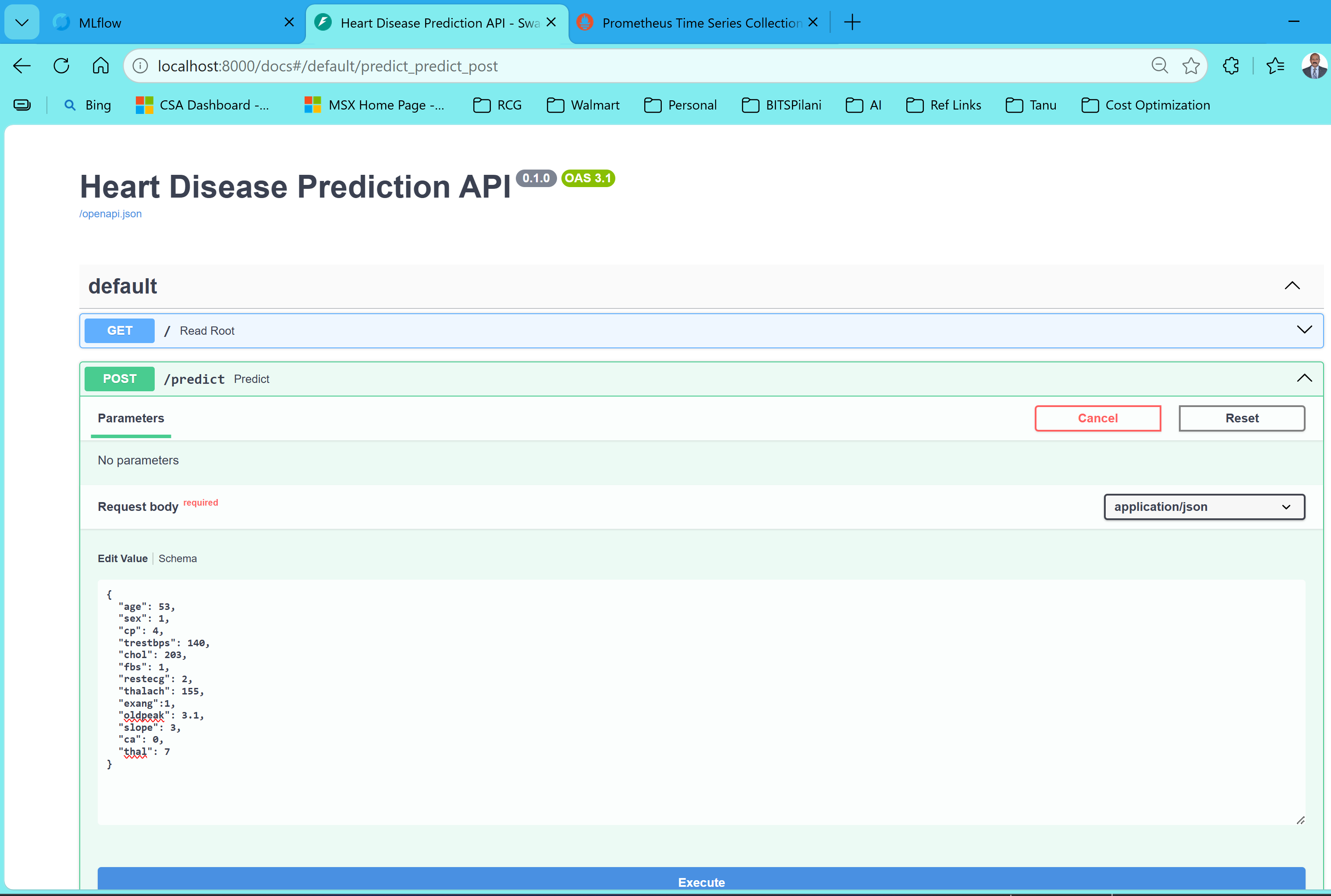


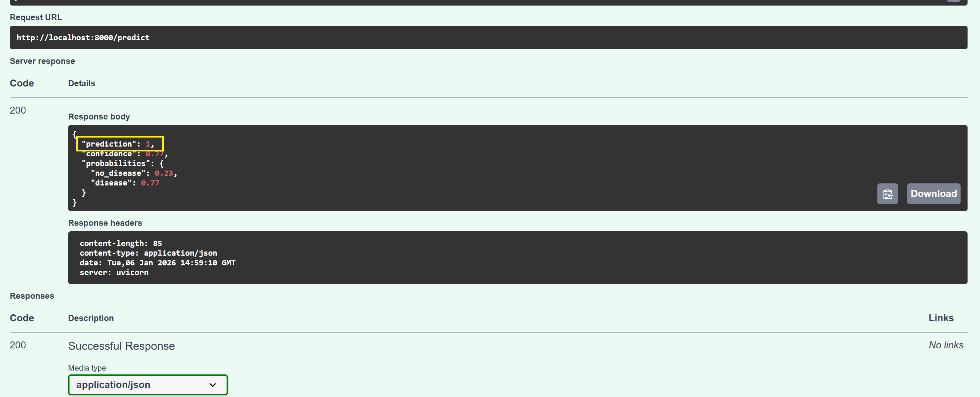


### Screenshots of test cases from API

Rest API is deployed and passed one of the patient details to retrieve the target. Ran couple of tests and predicted value was matching the target value.







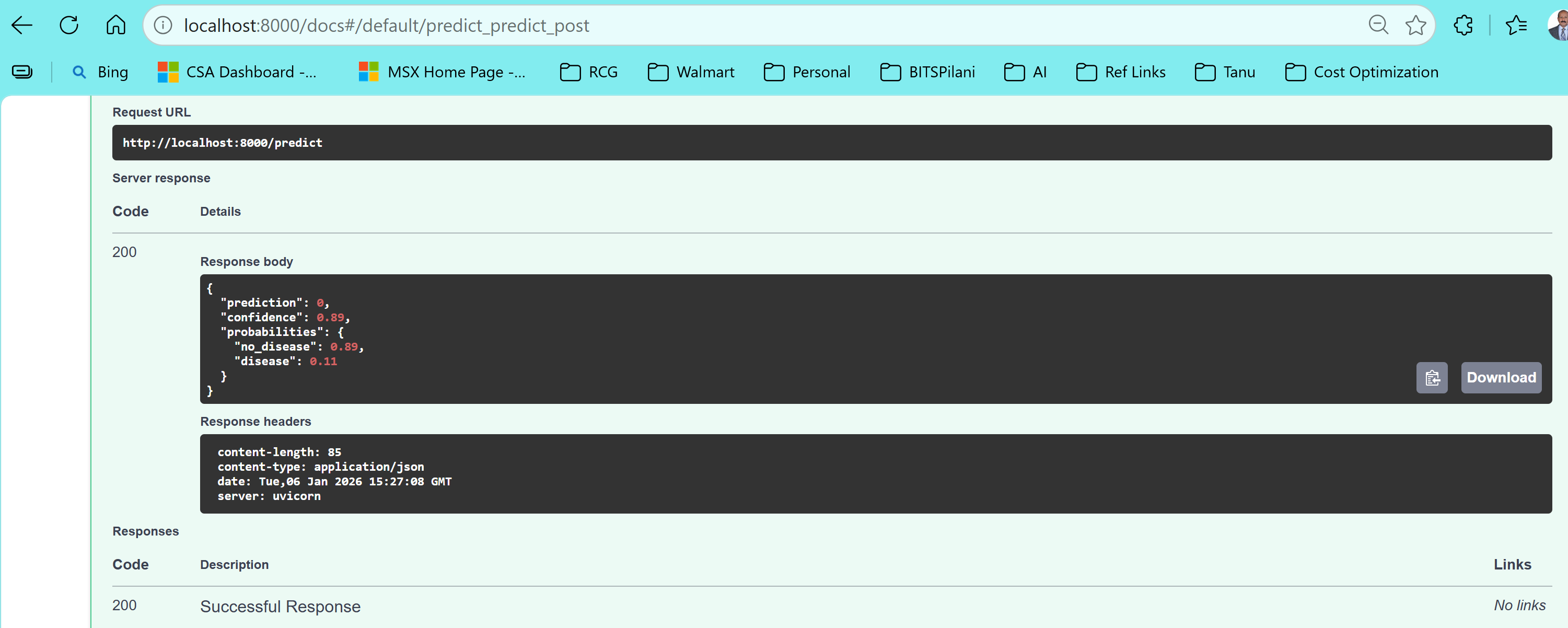
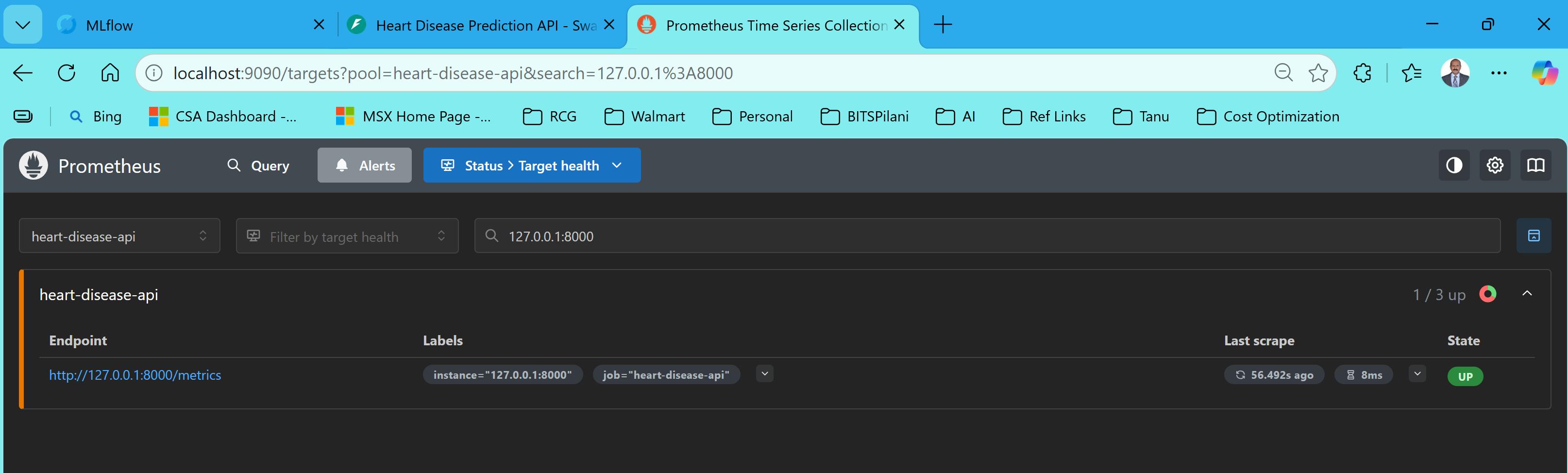


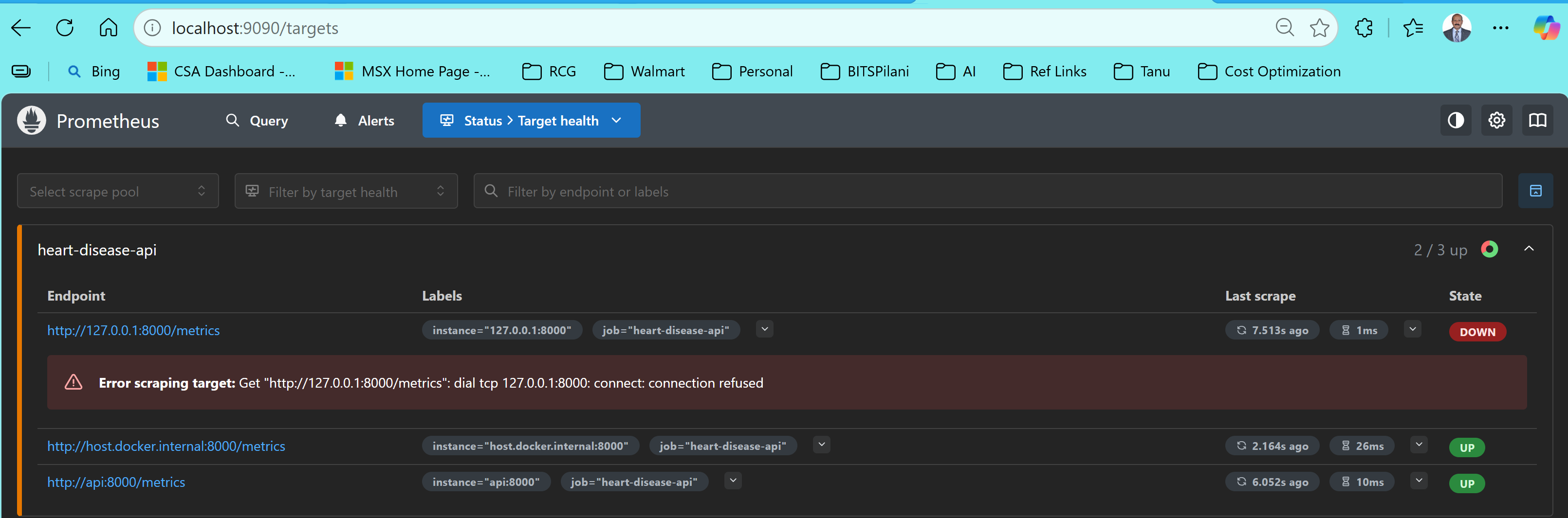
Fig : From docker API

### Monitoring & Logging

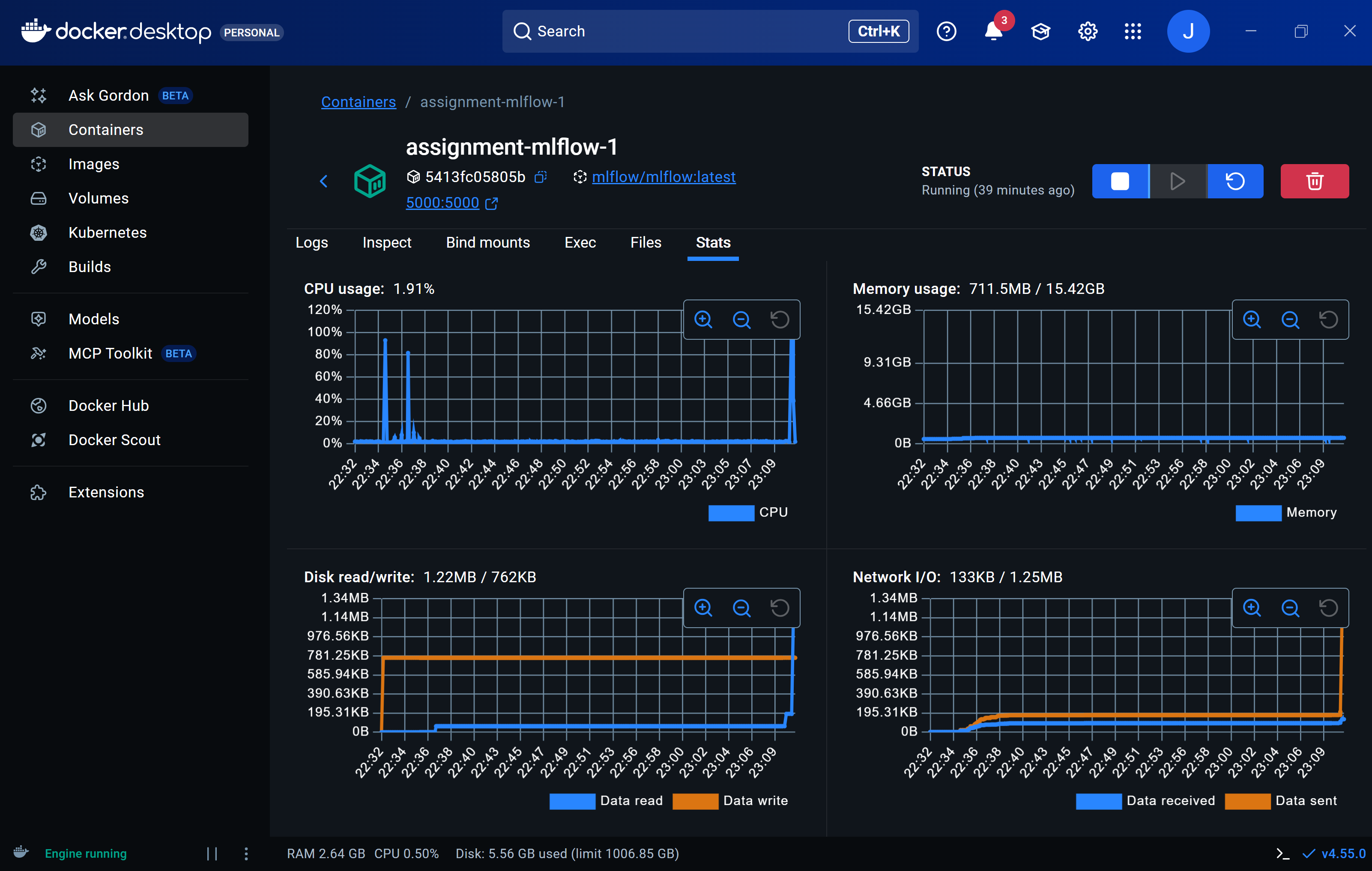
Service is running and status displays as “UP” service is down and status displays as “Down”



When docker is running



Note : Lat two are related to docker performance reports



### Additional screenshot for reference

* 1. Antigravity IDE leveraged for developing the code( .ipynb and .py scripts)

