# ML Ops Assignment_1 Report

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


## Abstract

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



Group 5

# Contents

# 1. Group 5 Members

2024ab05206 - MOHAMED ELSAID MAHMOUD ALI ELORBANY

2024aa05590 - MOHAMMED RASHID

2024ab05010 - M YAMEEN KASHU

2024ab05007 - YASIR JAMAL SIDDIQUI

2024aa05755 – Deepan K G

# 2. Link to code repository

https://github.com/melorbany/mlops-heart-disease-uci

# 3. Link to Demo

http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/

# 4. Link To Video

https://drive.google.com/file/d/1pQBHrCbt_0SvijwPxOCogZ4qalTWDFqV/view?usp=sharing

# 5. Setup/install instructions

We used following products to implement the project:

- **Github**
  - Code repository
  - Implement CI/CD Pipeline
- **Docker**
  - Package trained model. Base image used: docker.io/library/python:3.13-slim
  - **Docker hub** as image repository.
- **AWS to host the API**

# 6. EDA and modelling choices

## 6.1 EDA Steps

We evaluated the data quality and identified patterns within the heart disease dataset.

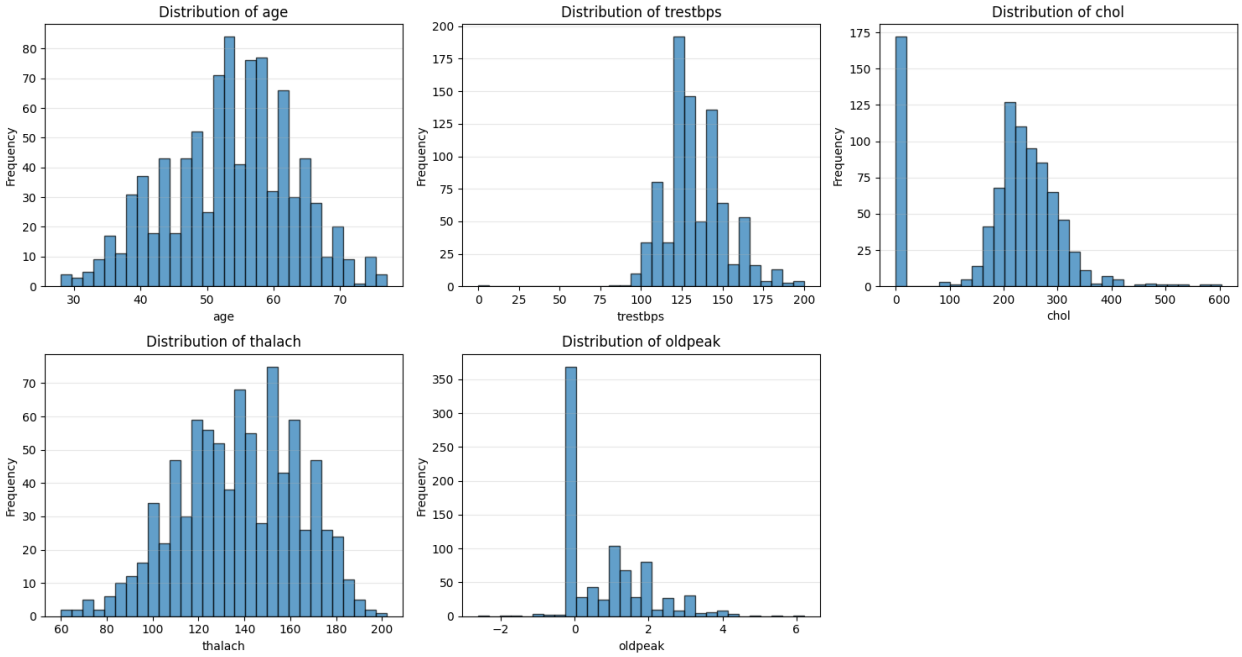
The following automated steps were performed:

- **Statistical Profiling:** Generated a summary report of the dataset's dimensions, identified missing values (treating "?" as null), and calculated descriptive statistics (mean, standard deviation, and range) for all numeric features.

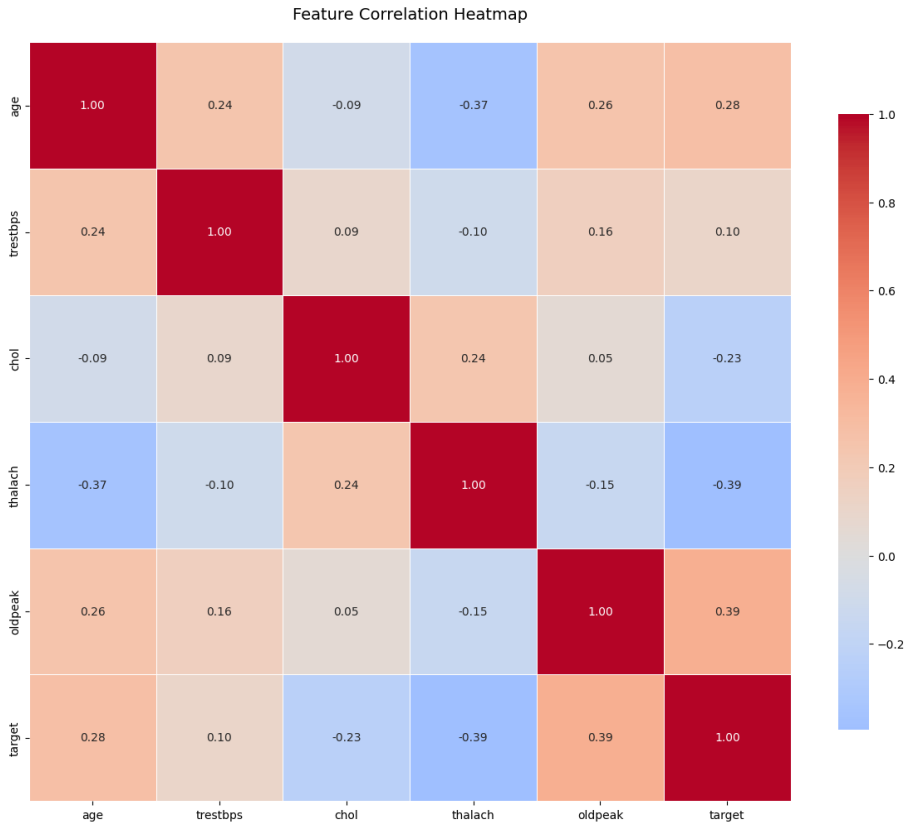
### EDA Summary

```
============================================================
HEART DISEASE DATASET - EDA SUMMARY
============================================================

Dataset shape: 918 rows, 14 columns

Target distribution:
  Class 0: 410 (44.7%)
  Class 1: 508 (55.3%)

Missing values per column:
  trestbps: 59
  chol: 29
  fbs: 90
  restecg: 2
  thalach: 55
  exang: 55
  oldpeak: 62
  slope: 307
  ca: 609
  thal: 484

Numeric features summary:
  age: mean=53.51, std=9.43, min=28.00, max=77.00
  trestbps: mean=132.14, std=19.06, min=0.00, max=200.00
  chol: mean=199.11, std=110.84, min=0.00, max=603.00
  thalach: mean=137.54, std=25.94, min=60.00, max=202.00
  oldpeak: mean=0.88, std=1.09, min=-2.60, max=6.20

============================================================```
```

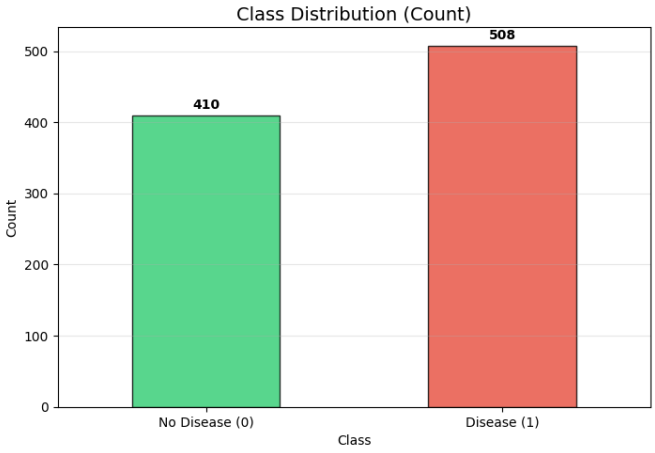
- **Univariate Distribution Analysis:** Developed a suite of histograms for all numeric variables to visualize data density and identify potential outliers.

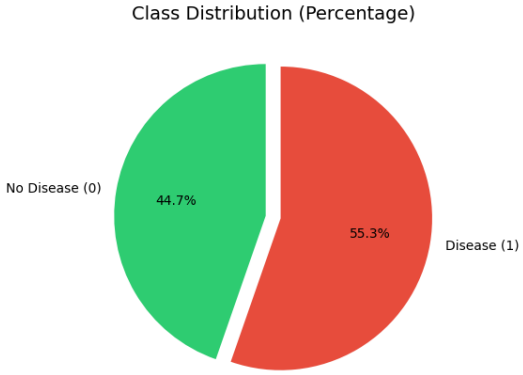



- **Correlation Mapping:** Conducted a multivariate analysis using a Pearson correlation heatmap to quantify the relationships between features and the target variable.



- **Class Balance Verification:** Evaluated the distribution of the target variable using bar and pie charts to ensure the dataset was not significantly skewed toward one class, which is critical for model training stability.



## 6.2 Modelling Choices

### *6.2.1 Preprocessing*

Based on the EDA phase, we took following preprocessing and feature engineering approaches:

- **For continuous variables**, we applied Median Imputation to handle missing data robustly and Standard Scaling to normalize feature scales, ensuring a mean of 0 and standard deviation of 1.
- **Qualitative variables** were processed using Mode Imputation followed by One-Hot Encoding. This converted categorical labels into a sparse binary format suitable for mathematical modeling.

### *6.2.2 Model Selection*

We evaluated two distinct algorithms to establish a performance baseline and explore feature interactions:

- **Logistic Regression:** Chosen as a high-interpretability baseline. It is effective for binary classification tasks where linear relationships between features (like age or blood pressure) and the target variable exist.
- **Random Forest Classifier:** An ensemble method used to capture non-linear patterns and complex feature interactions. We configured it with 200 estimators.

### *6.2.3 Validation Strategy*

To prevent overfitting and ensure the model generalizes well to unseen patient data, we employed a multi-layered validation approach:

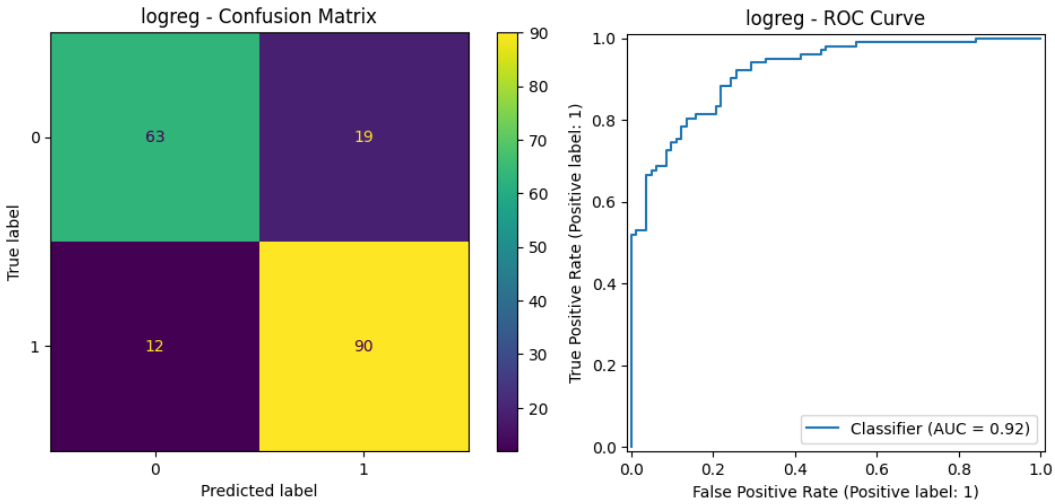
- **Stratified Train-Test Split:** We used a 20% hold-out test set, stratified by the target variable to maintain identical class distributions in both training and testing sets.
- **Stratified 5-Fold Cross-Validation:** During training, we performed K-Fold cross-validation on the training subset. This provided a "CV Mean" and "CV Standard Deviation," allowing us to assess the stability of the model across different data folds.

### *6.2.4 Evaluation Metrics*

Given the medical context of heart disease prediction, accuracy alone is insufficient. We prioritized the following metrics:

- **ROC-AUC (Primary Metric):** Used to evaluate the model's ability to distinguish between classes across all thresholds. This was our primary metric for selecting the "Best Model."

- **F1-Score:** To balance Precision and Recall, ensuring we minimize both False Positives and False Negatives.
- **Visual Diagnostics:** Automatically generated **Confusion Matrices** and **ROC Curves** for every experiment to visually inspect model performance.




## *6.2.5 Best Model Selection & Persistence*

The pipeline automatically compares the roc_auc of all trained models.

The winning model is:

1. Retrained on the full training set.
2. Serialized using joblib.
3. Logged as a production-ready artifact for deployment on AWS.

## 7. Experiment tracking summary

We utilized **MLflow** to implement a robust experiment tracking system. For every training iteration, the pipeline automatically logged:

- **Hyperparameters and Configuration:** Full transparency into model settings and data split parameters.
- **Performance Metrics:** Systematic logging of ROC-AUC, F1-Score, and Cross-Validation results to ensure statistical significance.
- **Artifact Preservation:** Automated storage of diagnostic plots (Confusion Matrices, ROC Curves) and serialized model binaries, ensuring 1:1 traceability between the code version and the deployed model.

Following is one sample experiment tracking summary report (screenshots). We here only showing the important data that is captured:

# MLflow Experiment Summary

**Tracking DB:** `mlflow-db/mlflow.db`

## Experiments

| ID | Name | Status |
|---|---|---|
| 0 | Default | active |
| 1 | heart-disease-classification | active |

## Key Runs (latest)

| Run ID | Model | Status | Started (UTC) |
|---|---|---|---|
| 0eaebe4206b94b38b049157d8291633e | **Final model selection** | FINISHED | 2026-01-05 12:08 |
| 265fa5a7bbe14697a72b36857a016612 | Random Forest | FINISHED | 2026-01-05 12:08 |
| cb10886457b34c8ea968c4d3cd3a41d8 | Logistic Regression | FINISHED | 2026-01-05 12:08 |

## Selected Best Model

| Key | Value |
|---|---|
| Best model | **Logistic Regression** |
| Test ROC-AUC | **0.9192** |
| Model artifact | `models/heart_model.pkl` |

# Evaluation Metrics (Test Set)

## Random Forest

| Metric | Value |
|---|---|
| Accuracy | 0.848 |
| Precision | 0.849 |
| Recall | 0.882 |
| F1 | 0.865 |
| ROC-AUC | 0.910 |
| CV ROC-AUC (mean ± std) | 0.875 ± 0.046 |

## Logistic Regression

| Metric | Value |
|---|---|
| Accuracy | 0.832 |
| Precision | 0.826 |
| Recall | 0.882 |
| F1 | 0.853 |
| ROC-AUC | **0.919** |
| CV ROC-AUC (mean ± std) | 0.878 ± 0.042 |

## Dataset Summary

| Property | Value |
|---|---|
| Dataset | `heart_clean` |
| Rows | 918 |
| Features | 14 |
| Train / Test split | 80% / 20% |
| CV folds | 5 |

---

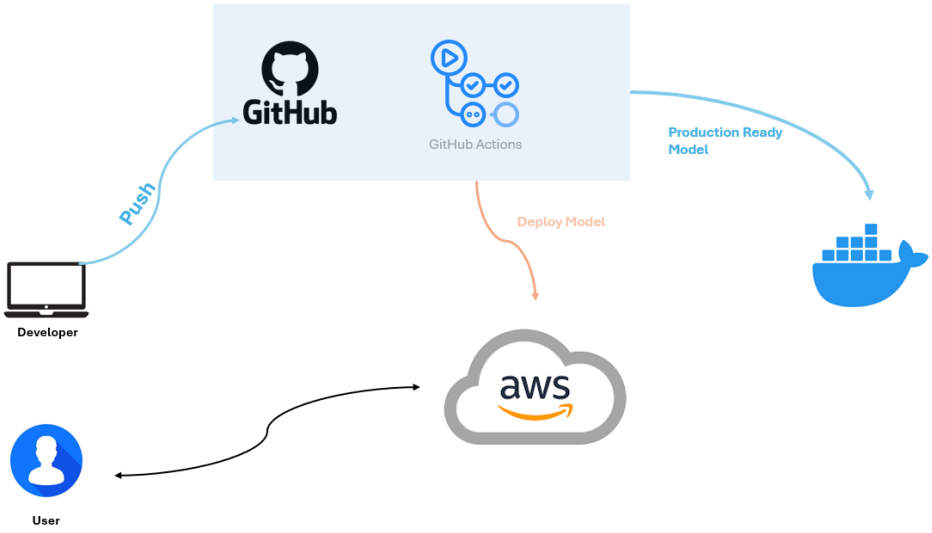
## Reproducibility & Metadata

| Key | Value |
|---|---|
| Git commit | `0f195eed139349235e67602fa222516a9bf04112` |
| Branch | `master` |
| Python | 3.13.11 |
| OS | Linux (Azure runner) |
| User | runner |

## Artifacts

| Run | Artifact URI |
|---|---|
| Final | `mlruns/1/0eaebe4206b94b38b049157d8291633e/artifacts` |
| RF | `mlruns/1/265fa5a7bbe14697a72b36857a016612/artifacts` |
| LogReg | `mlruns/1/cb10886457b34c8ea968c4d3cd3a41d8/artifacts` |

# 8. Architecture diagram




1. **The Development Trigger**

- **Developer & Push:** The process begins with the **Developer**, who writes code for the ML model, data processing scripts, or infrastructure configurations.
- **GitHub (Source Control):** Once the code is ready, the developer "pushes" it to a **GitHub Repository**. This push serves as the trigger for the entire automated pipeline.

2. **Orchestration & Automation**

- **GitHub Actions (The Engine):** This is the central orchestrator of your MLOps pipeline. As seen in your previous screenshot, GitHub Actions runs the sequential and parallel jobs like:
    - **CI:** Data fetching, linting, and security scans.
    - **CT:** Training the model and generating MLflow reports.
    - **Validation:** Running tests to ensure the model meets quality standards.

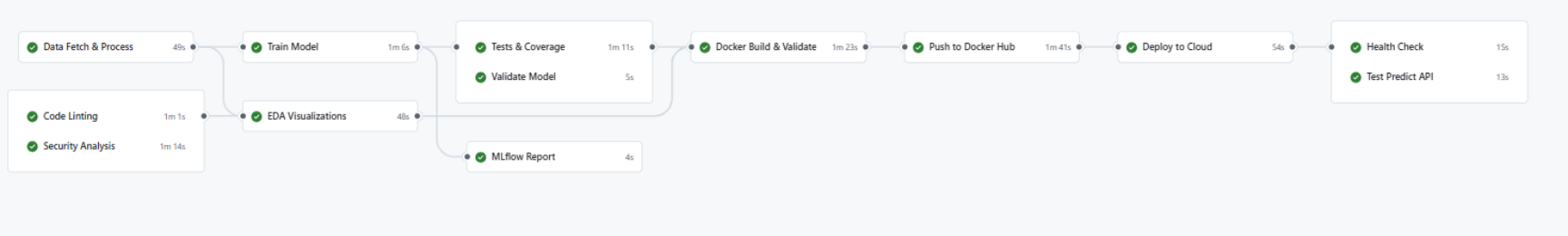
3. **Containerization & Artifact Management**

- **Docker (Production Ready Model):** Once the model is validated, GitHub Actions packages the model, its dependencies, and the API code into a **Docker Image**.
- **The Container:** By using Docker, you ensure the model runs exactly the same way in the cloud as it did during testing, eliminating "it works on my machine" issues. This image is typically pushed to a registry (like Docker Hub or AWS ECR).

4. **Cloud Deployment & Serving**

- **AWS (Production Environment):** GitHub Actions uses cloud credentials to "Deploy" the Docker container to **AWS**.
- **User Interaction:** The **User** (bottom left) interacts with the model via a Prediction API hosted on AWS. They send data to an endpoint and receive the model's prediction in return.

# 9. CI/CD and deployment workflow screenshots

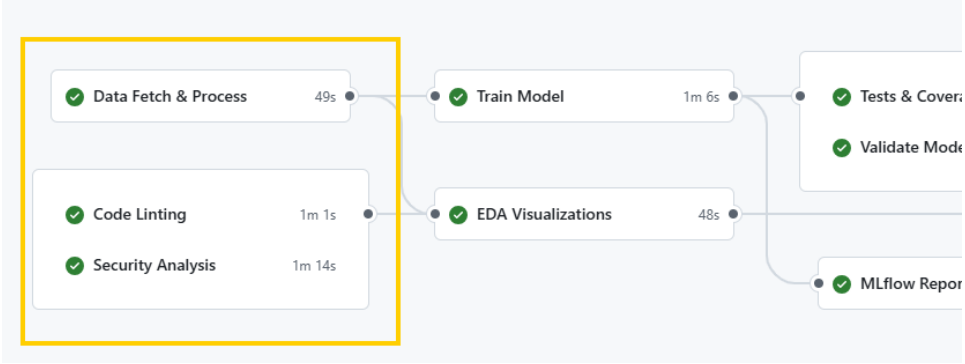



## 9.1 Pipeline Summary

The pipeline follows a **linear automation flow** with parallel processing in the early stages. It starts with data ingestion and security auditing, moves into model training and validation, and concludes with containerization, cloud deployment, and live API verification. The total automated sequence ensures that no code or model reaches the cloud without passing quality and performance "gates."

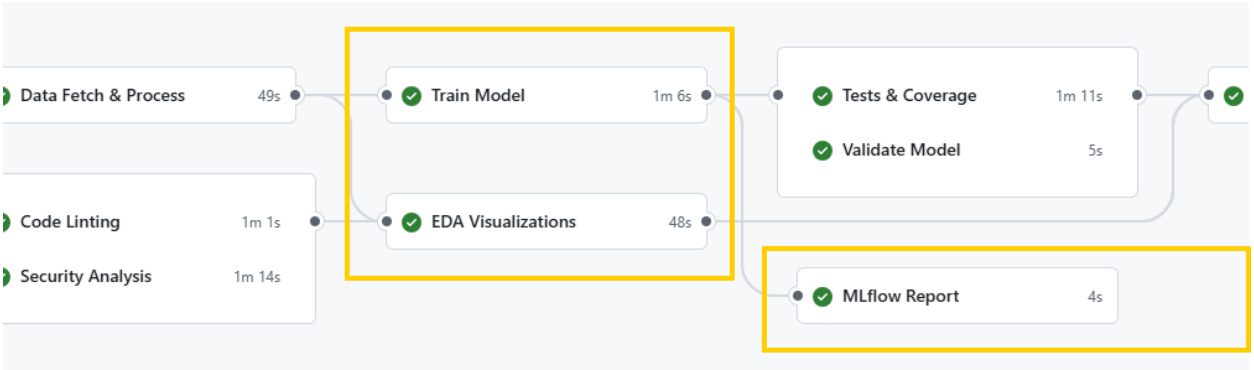
## 9.2 Pipeline phases

### *9.2.1 Data Engineering & CI (Continuous Integration)*



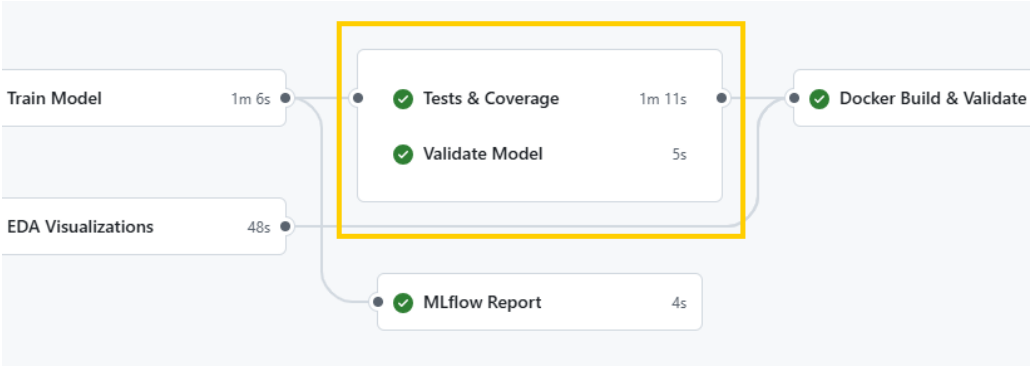

- **Pipeline Steps:** Data Fetch & Process, Code Linting, Security Analysis.
- **Role:** This is the "Data & Code Quality" phase. It ensures the inputs (data) and the logic (code) are clean, secure, and ready for compute-heavy tasks.

### *9.2.2 Continuous Training (CT) & Experimentation*



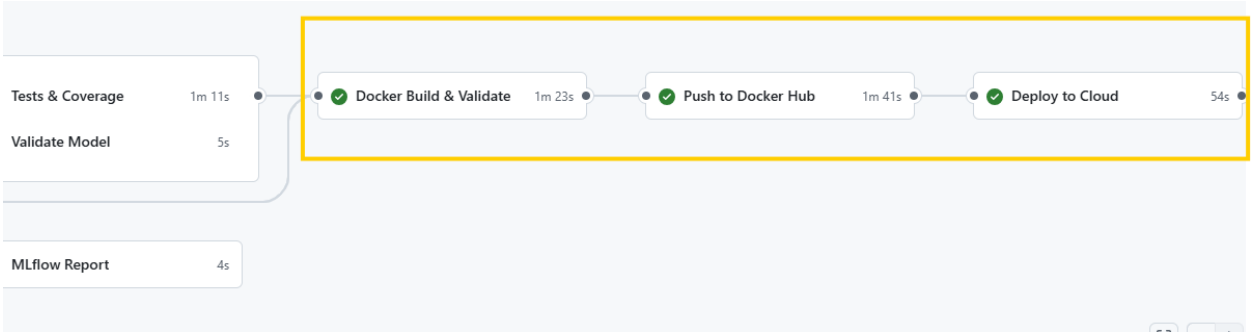

- **Pipeline Steps:** Train Model, EDA Visualizations, MLflow Report.
- **Role:** This phase handles the core ML workload. It trains the model while simultaneously generating visual insights (EDA) and logging all metadata (parameters/metrics) to a tracking server for reproducibility.

### *9.2.3 Model Evaluation & Validation*



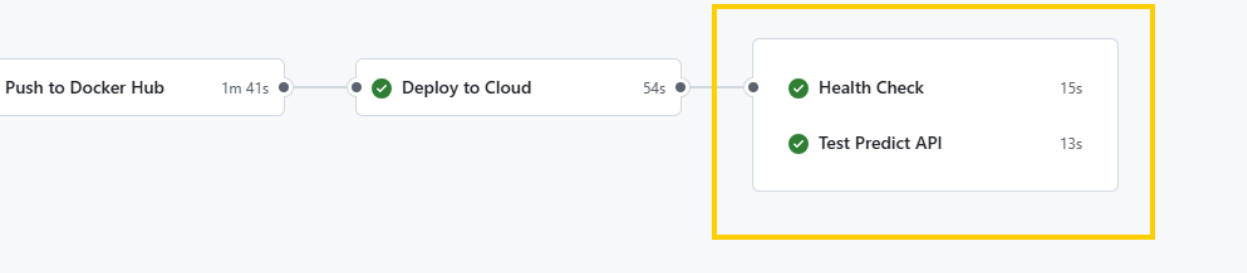

- **Pipeline Steps:** Tests & Coverage, Validate Model.
- **Role:** This acts as the **Quality Gate**. It ensures the code is robust (Tests) and the model performs better than a baseline (Validate) before any deployment resources are spent.

### *9.2.4 Continuous Deployment (CD)*

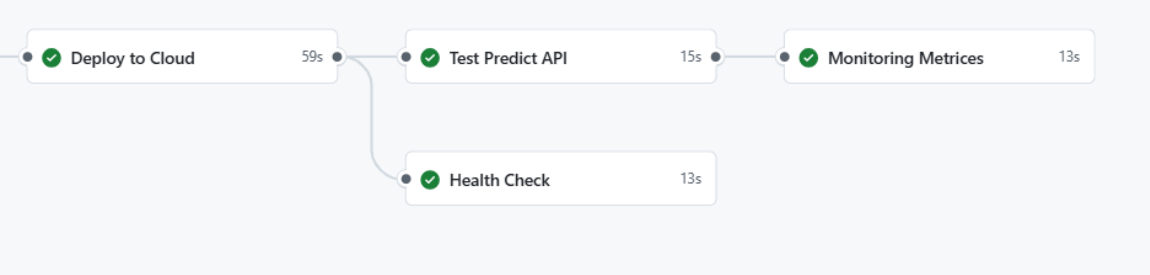



- **Pipeline Steps:** Docker Build & Validate, Push to Docker Hub, Deploy to Cloud.
- **Role:** This is the "Delivery" phase. It packages the model into a portable container (Docker) and pushes it to a production cloud environment, moving the model from a laboratory artifact to a live service.

## *9.2.5 Operations & Monitoring*




- **Pipeline Steps:** Health Check, Test Predict API.
- **Role:** This is the "Post-Deployment" phase. It provides immediate feedback that the live endpoint is not only "up" (Health) but also returning accurate mathematical predictions (Test Predict).

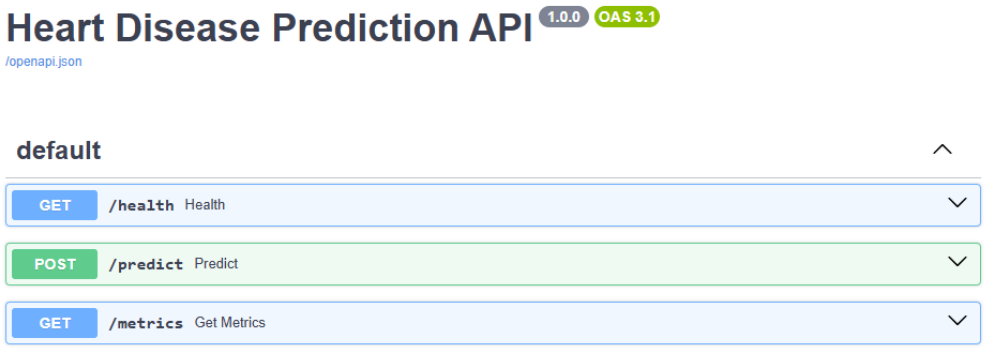



- **Pipeline Steps:** Monitoring
- **Role:** This is the "Post-Deployment" phase. It confirms that matrics API is working as expected.

# 10. Live API Details

## 10.1 Production URL

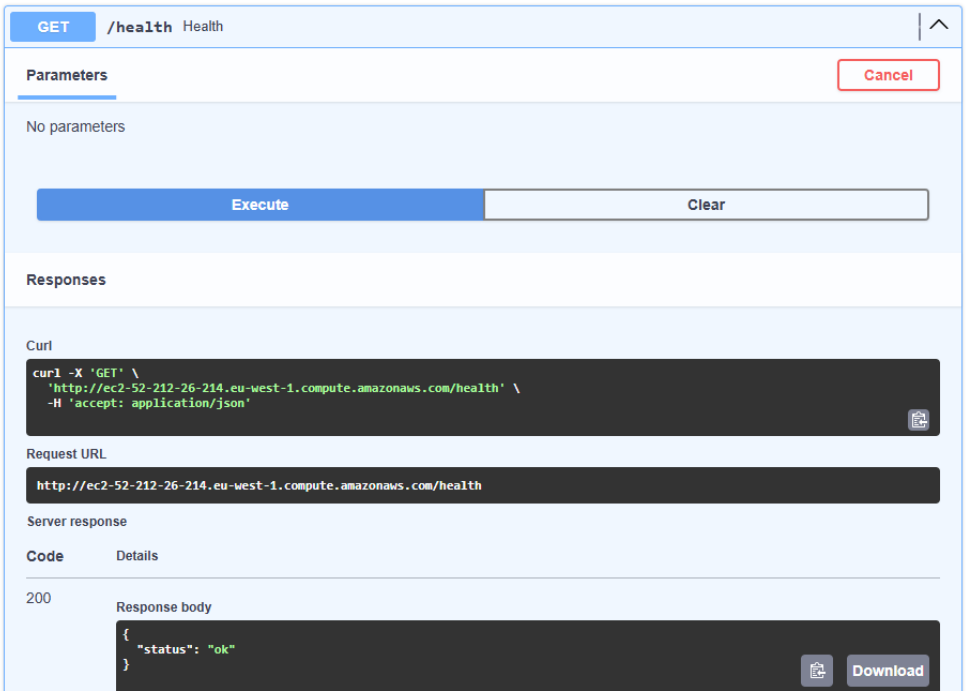
http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/




## 10.2 Tests

### *10.2.1 Health Check*

curl -X 'GET' 'http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/health' -H 'accept: application/son'



### *10.2.2 Predict API*

**Target = 1 scenario**

Curl

```
curl -X 'POST' \
  'http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "age": 65,
  "sex": 1,
  "cp": 3,
  "trestbps": 145,
  "chol": 233,
  "fbs": 1,
  "restecg": 0,
  "thalach": 150,
  "exang": 0,
  "oldpeak": 2.5,
  "slope": 0,
  "ca": 0,
  "thal": 1
}'
```

Request URL

```
http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/predict
```

Server response

| Code | Details |
|---|---|
| 200 | Response body |

```
{
  "prediction": 1,
  "probability": 0.5847428590787366
}
```

Download

**Target = 0 scenario**

Curl

```
curl -X 'POST' \
  'http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "age": 25,
  "sex": 1,
  "cp": 3,
  "trestbps": 90,
  "chol": 120,
  "fbs": 1,
  "restecg": 0,
  "thalach": 90,
  "exang": 0,
  "oldpeak": 1.5,
  "slope": 0,
  "ca": 0,
  "thal": 1
}'
```

Request URL

```
http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/predict
```

Server response

| Code | Details |
|---|---|
| 200 | Response body |

```
{
  "prediction": 0,
  "probability": 0.463930893450522
}
```

## *10.2.3 Get Metrics API*

Curl

```
curl -X 'GET' \
  'http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/metrics' \
  -H 'accept: application/json'
```

Request URL

```
http://ec2-52-212-26-214.eu-west-1.compute.amazonaws.com/metrics
```

Server response

| Code | Details |
|---|---|
| 200 | Response body |

```
{
  "total_requests": 4,
  "total_predictions": 4,
  "positive_predictions": 3,
  "negative_predictions": 1
}
```

Download