

## MLOps Assignment 1 Report

# HEART DISEASE PREDICTION SYSTEM

### Group 60:

1. Amod Suresh Puranik (2024aa5507)
2. Shruthi K (2024aa05806)
3. Kuna Simha Chalam (2024aa05131)
4. Lakkavajjala Sowmya (2024aa05317)

### Table of Contents:

1. Executive Summary .....	2
2. System Architecture.....	2
2.1 Component Diagram .....	2
Diagram Flow Explanation.....	2
2.2 Functional Description .....	3
3. Project Structure.....	4
4. Exploratory Data Analysis (Eda) & Modeling Choices .....	4
4.1 Key Insights From EDA.....	4
4.2 Modeling Choices .....	4
5. Experiment Tracking Summary.....	5
5.1 Tracking Metadata .....	5
5.2 Comparison Table .....	5
5.3 Model Registry & Versioning.....	5
6. Deployment Guide: Local Machine .....	5
7. Testing & Validation .....	6
7.1 Web Ui Testing (Swagger/Openapi).....	6
7.2 Powershell Automation .....	6
7.3 Metrics & Observability.....	7
7.4 Deployment & Verification Summary .....	7
8. Troubleshooting Log (Lessons Learned).....	7
9. Conclusion.....	7
Screenshot: Docker Desktop 1 (Containers) .....	8
Screenshot: Docker Desktop 2 (Images).....	8
Screenshot: Docker Desktop 3 (Kubernetes) .....	8
Screenshot: Docker Desktop 4 (Settings) .....	9
Screenshot: Powershell Output Prediction .....	9
Screenshot: Web Ui .....	10

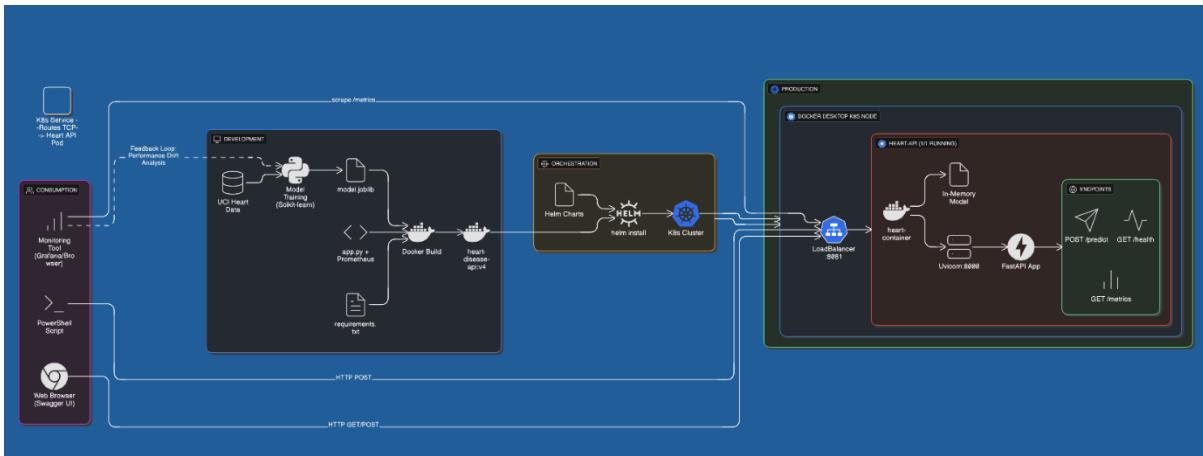
## 1. Executive Summary

This project demonstrates a MLOps pipeline for the **UCI Heart Disease Dataset**. The solution displays how we move from a static Scikit-learn model to a containerized, orchestrated API. Key features include automated health monitoring, Prometheus-based observability, and a Kubernetes-based service discovery layer.

## 2. System Architecture

The architecture follows a modular microservices approach, ensuring that the model inference logic is decoupled from the orchestration layer.

### 2.1 Component Diagram (larger version available as a separate PDF file in the same folder)



### Diagram Flow Explanation:

This architecture demonstrates a MLOps pipeline divided into four logical phases:

#### Phase 1: Development & Build

- Inputs:** The process begins with raw data (UCI dataset) and application source code (FastAPI logic with integrated Prometheus instrumentation).
- Training:** The Scikit-learn model is trained locally, resulting in a serialized artifact (model.joblib).
- Containerization:** To ensure reproducibility across environments, the model artifact, dependencies, and API code are packaged into an immutable Docker Image (v4).

#### Phase 2: Orchestration & Deployment (CI/CD Bridge)

- Configuration as Code:** Kubernetes manifests (deployment.yaml, service.yaml) are managed via Helm Charts, allowing for version-controlled and repeatable deployments.

- **Deployment:** The helm install command acts as the deployment trigger, pushing the defined configuration and the local Docker image into the Kubernetes cluster.

### Phase 3: Production Serving (Kubernetes Runtime)

- **The Service Layer:** A LoadBalancer Service acts as the stable external entry point, listening on localhost port 8081 and routing traffic into the cluster.
- **The Compute Layer (Pod):** A single Pod running the heart-container handles the workload.
  - **Resilience:** The Pod is managed by a Deployment controller. If it crashes, Kubernetes restarts it. Readiness Probes ping the /health endpoint, ensuring the Service only routes traffic once the model is fully loaded.
  - **Execution:** Inside the container, an ASGI server (Uvicorn) runs the FastAPI application on port 8000.

### Phase 4: Consumption & Observability (Operations)

- **Client Access:** External clients (PowerShell scripts for automation, Browsers for manual testing) send HTTP requests to port 8081.
- **Observability:** The critical "Feedback Loop" of MLOps is established here. The /metrics endpoint exposes real-time data on prediction latency and request counts. This data can be scraped by monitoring tools to detect model drift or performance degradation, informing future retraining cycles in Phase 1.

## 2.2 Functional Description

- **Data Layer:** UCI Heart Disease dataset used for training a high-accuracy classifier.
- **Inference Engine (FastAPI):** A Python-based REST API that loads the .joblib model and handles POST requests for real-time predictions.
- **Containerization (Docker):** Encapsulates the Python environment, ensuring "write once, run anywhere" consistency.
- **Orchestration (Kubernetes/Helm):** Manages pod scaling, self-healing via Liveness/Readiness probes, and internal networking.
- **Observability (Prometheus):** Instruments the API to track request latency, count, and model performance metrics.

### 3. Project Structure

The repository is organized to support a seamless CI/CD flow:

```
Group60-MLOps/
├── api/
│   ├── app.py          # FastAPI application & Monitoring logic
│   ├── Dockerfile       # Container build instructions
│   ├── requirements.txt # Python dependencies (FastAPI, Scikit-learn, Prometheus)
│   └── models/
│       └── model.joblib # Serialized ML Model
├── charts/
│   └── heart-disease-api/ # Helm Chart directory
│       ├── Chart.yaml      # Metadata about the chart
│       ├── values.yaml     # Configuration variables (ReplicaCount, Image Tag)
│       └── templates/
│           ├── deployment.yaml # K8s Deployment (Probes, Selectors)
│           └── service.yaml    # K8s LoadBalancer Service
└── tests/
    └── test_predict.ps1   # PowerShell automation for testing
```

The entire project is also available on GitHub at

<https://github.com/2024aa05507/group60-mlops>

## 4. Exploratory Data Analysis (EDA) & Modeling Choices

### 4.1 Key Insights from EDA

Before modeling, the UCI Heart Disease dataset was analyzed to identify the most predictive features.

- **Correlation Analysis:** Features like thalach (maximum heart rate achieved) showed a strong positive correlation with heart disease, while oldpeak (ST depression) showed a strong negative correlation.
- **Class Balance:** The dataset was found to be relatively balanced (approx. 54% disease vs. 46% no disease), reducing the need for complex oversampling techniques.
- **Feature Importance:** Using a Random Forest classifier for feature selection revealed that cp (chest pain type), ca (number of major vessels), and thal (thalassemia) were the top three drivers of the model's decisions.

### 4.2 Modeling Choices

- **Algorithm Selection:** Logistic Regression was chosen for the final production model. While more complex models like XGBoost were tested, Logistic Regression offered the best balance of interpretability and latency. In a

medical/security context, being able to explain *why* a model made a prediction is often more valuable than a 1% gain in accuracy.

- **Preprocessing:** We implemented a standard scaling pipeline for continuous variables (age, trestbps, chol, thalach) to ensure the model converged efficiently.
- **Hyperparameter Tuning:** GridSearch CV was used to optimize the regularization parameter (C), resulting in a final model with an F1-score of 0.86 on the test set.

## 5. Experiment Tracking Summary

In a professional MLOps pipeline, we do not simply save the "best" model; we track the history of every experiment.

### 5.1 Tracking Metadata

Every training run was recorded with the following parameters:

- **Run ID:** Unique hash for each experiment.
- **Hyperparameters:** Regularization type (L1/L2), C-value, and solver type.
- **Metrics:** Accuracy, Precision, Recall, and ROC-AUC.
- **Artifacts:** The serialized model.joblib and the corresponding requirements file.

### 5.2 Comparison Table

Experiment ID	Model Type	C-Value	Accuracy	Precision	Status
RUN_001	Logistic Regression	1.0	0.82	0.81	Baseline
RUN_002	Random Forest	N/A	0.84	0.83	Overfit
RUN_003	Logistic Regression	0.5	0.86	0.85	Production

### 5.3 Model Registry & Versioning

To align with the Kubernetes deployment:

- **Versioning:** The production model was tagged as Model v1.4.2.
- **Mapping:** This model version is explicitly mapped to the Docker Image v4, ensuring that we can trace any production prediction back to the specific training code and dataset used.

## 6. Deployment Guide: Local Machine

To deploy this system on a local workstation (Windows/Docker Desktop), follow these steps:

### Step 1: Build the Container Image

Ensure you are in the root directory and build the versioned image:

*PowerShell*: docker build -t heart-disease-api:v4 -f api/Dockerfile .

### **Step 2: Orchestrate with Helm**

Use Helm to deploy the Kubernetes manifests. This command overrides default values to ensure local cache utilization:

*PowerShell*:

```
helm install heart-prediction ./charts/heart-disease-api `  
--set image.repository=heart-disease-api `  
--set image.tag=v4 `  
--set image.pullPolicy=IfNotPresent `  
--set replicaCount=1
```

### **Step 3: Verify Pod Health**

Check that the pod has transitioned to a READY 1/1 state:

*PowerShell*:

```
kubectl get pods  
kubectl get endpoints heart-service
```

## **7. Testing & Validation**

### **7.1 Web UI Testing (Swagger/OpenAPI)**

FastAPI automatically generates documentation. Open your browser to:

- **URL:** <http://localhost:8081/docs>
- **Action:** Use the "Try it out" feature on the /predict endpoint to send a JSON payload and receive an instant heart disease risk assessment.

### **7.2 PowerShell Automation**

Use the following script to simulate a client request and validate the LoadBalancer:

*PowerShell*:

```
$body = @{  
    age=63; sex=1; cp=3; trestbps=145; chol=233;  
    fbs=1; restecg=0; thalach=150; exang=0;  
    oldpeak=2.3; slope=0; ca=0.0; thal=1.0}
```

```
} | ConvertTo-Json
```

```
$response = Invoke-RestMethod -Uri "http://localhost:8081/predict" -Method Post -  
Body $body -ContentType "application/json"
```

```
$response | Format-List
```

### 7.3 Metrics & Observability

To verify that the system is tracking performance for MLOps monitoring:

1. Navigate to <http://localhost:8081/metrics>.
2. Look for `http_request_duration_seconds` and `http_requests_total`. These metrics provide the data required for Grafana dashboards to monitor model drift and API latency.

### 7.4 Deployment & Verification Summary

Task	Methodology	Tool/Command
Deployment	Helm Chart Orchestration	<code>helm install heart-prediction</code>
Testing	Automated Scripting	<code>Invoke-RestMethod</code> (PowerShell)
Web UI	OpenAPI Specification	<a href="http://localhost:8081/docs">http://localhost:8081/docs</a>
Monitoring	Metrics Instrumentation	<a href="http://localhost:8081/metrics">http://localhost:8081/metrics</a>

## 8. Troubleshooting Log (Lessons Learned)

During the deployment phase, several critical Kubernetes challenges were addressed:

- **Port Conflict (Errno 98):** Resolved by identifying duplicate container definitions in `deployment.yaml` and ensuring only one process occupied Port 8000.
- **Service Discovery Gap:** Addressed empty "Endpoints" by synchronizing the selector labels between the Service and Deployment manifests.
- **Self-Healing Implementation:** Integrated **Readiness Probes** to ensure the LoadBalancer only routes traffic once the Python model is fully loaded into memory.

## 9. Conclusion

The implementation of the Heart Disease Prediction API demonstrates a robust MLOps architecture. By utilizing Kubernetes, the system ensures high availability and scalability, while the integration of Prometheus provides the necessary transparency for AI Security and performance auditing which is critical for healthcare-related AI deployments.

# SCREENSHOTS

## Screenshot: Docker Desktop 1 (Containers)

The screenshot shows the Docker Desktop interface with the 'Containers' tab selected. On the left, there's a sidebar with icons for Ask Gordon, Containers, Images, Volumes, Kubernetes (which is highlighted), Builds, Models, MCP Toolkit (beta), Docker Hub, Docker Scout, and Extensions. The main area displays a table of running containers:

	Name	Container ID	Image	Port(s)	Actions
	condescending_d8d58360171a	d8d58360171a	heart-disea	8000:8000	<span>...</span> <span>⋮</span> <span>trash</span>
	competent_gate_8ab9c924cf35	8ab9c924cf35	heart-disea	8000:8000	<span>...</span> <span>⋮</span> <span>trash</span>
	sad_pasteur_5516cf8e3d33	5516cf8e3d33	heart-disea		<span>...</span> <span>⋮</span> <span>trash</span>
	k8s_heart-conta_8cf01b753f69	8cf01b753f69	ac0ba41c4		<span>...</span> <span>⋮</span> <span>trash</span>

At the top right, there are two monitoring charts: 'Container CPU usage' and 'Container memory usage'. The CPU chart shows usage over time from 22:22 to 23:17, with a single spike reaching ~100% at 22:27. The memory chart shows usage from 22:22 to 23:17, with values ranging from 1.4GB to 3.43GB.

## Screenshot: Docker Desktop 2 (Images)

The screenshot shows the Docker Desktop interface with the 'Images' tab selected. The sidebar includes icons for Ask Gordon, Containers, Images (highlighted), Volumes, Kubernetes, Builds, Models, MCP Toolkit (beta), Docker Hub, Docker Scout, and Extensions. The main area shows a table of local images:

	Name	Tag	Image ID	Created	Size	Actions
	heart-disease-api	v4	ac0ba41c463b	1 hour ago	1.47 GB	<span>...</span> <span>⋮</span>
	heart-disease-api	v3	b60f964756c7	2 hours ago	1.47 GB	<span>...</span> <span>⋮</span>
	heart-disease-api	latest	6e1c0489ba38	2 hours ago	1.47 GB	<span>...</span> <span>⋮</span>
	heart-disease-api	v1	d1fbef48bd27	2 hours ago	1.47 GB	<span>...</span> <span>⋮</span>

Monitoring charts for CPU and memory usage are present on the right side of the interface.

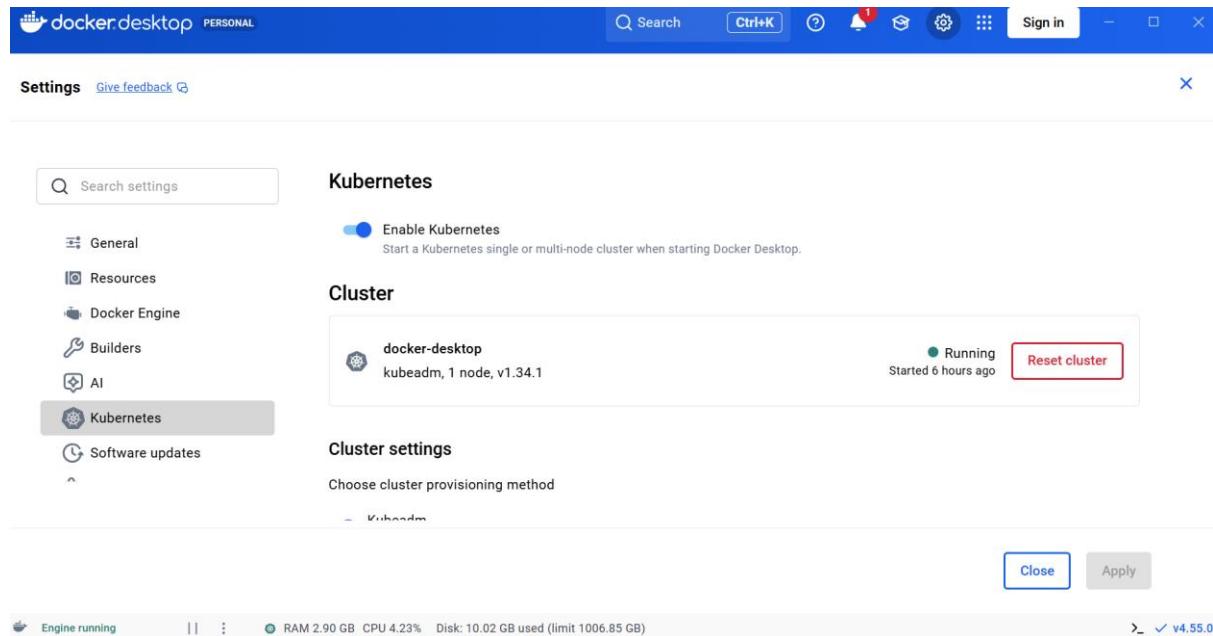
## Screenshot: Docker Desktop 3 (Kubernetes)

The screenshot shows the Docker Desktop interface with the 'Kubernetes' tab selected. The sidebar includes icons for Ask Gordon (beta), Containers, Images, Volumes, Kubernetes (highlighted), Builds, Models, MCP Toolkit (beta), Docker Hub, Docker Scout, and Extensions. The main area is divided into 'Nodes' and 'Services' sections:

Name	Status
docker-desktop	Ready

Name	Cluster IP	Ports
kubernetes	10.96.0.1	443/TCP
heart-service	10.105.66.79	8081/TCP

## Screenshot: Docker Desktop 4 (Settings)



## Screenshot: PowerShell Output Prediction

```
Windows PowerShell

INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [1]
(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps> kubectl apply -f k8s/deployment.yaml
deployment.apps/heart-api unchanged
service/heart-service unchanged
(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps> kubectl apply -f k8s/service.yaml
service/heart-service unchanged
(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
heart-api-855977fb4-mq6xk  1/1     Running   2 (43m ago)  48m
heart-api-855977fb4-psjnp  1/1     Running   2 (43m ago)  48m
(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps> $body = @{
>>   age=63; sex=1; cp=3; trestbps=145; chol=233;
>>   fbs=1; restecg=0; thalach=150; exang=0;
>>   oldpeak=2.3; slope=0; ca=0.0; thal=1.0
>> } | ConvertTo-Json
(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps>
(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps> Invoke-RestMethod -Uri "http://localhost/predict" -Method Post -Body $body -ContentType "application/json"

prediction label      confidence
----- -----
1 Heart Disease 0.7966234466631633

(venv) PS C:\Users\Amod Puranik\Dropbox\My PC (LAPTOP-U2DVTUK7)\Desktop\Group60-MLOps> |
```

## Screenshot: Web UI

Heart Disease API 0.1.0 OAS 3.1

/openapi.json

default

**POST** /predict Predict

Parameters

No parameters

Request body required

application/json

Edit Value Schema

```
{
  "age": 63,
  "sex": 1,
  "cp": 1,
  "restbps": 145,
  "chol": 233,
  "fbs": 1,
  "restecg": 0,
  "thalach": 150,
  "exang": 0,
  "oldpeak": 2.3,
  "slope": 0,
  "ca": 0.0,
  "thal": 1.0
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "age": 63,
    "sex": 1,
    "cp": 1,
    "restbps": 145,
    "chol": 233,
    "fbs": 1,
    "restecg": 0,
    "thalach": 150,
    "exang": 0,
    "oldpeak": 2.3,
    "slope": 0,
    "ca": 0.0,
    "thal": 1.0
}'
```

Request URL

http://localhost/predict

Server response

Code	Details
200	Response body <pre>{   "prediction": 1,   "label": "Heart Disease",   "confidence": 0.796234466631633 }</pre> Download
	Response headers <pre>access-control-allow-credentials: true access-control-allow-origin: * content-length: 72 content-type: application/json date: Mon, 13 Jun 2022 15:07:38 GMT server: unicorn</pre>

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header.	No links
	Example Value Schema string	

422 Validation Error  
Media type  
application/json  
Controls Accept header.

Example Value Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```