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

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)**

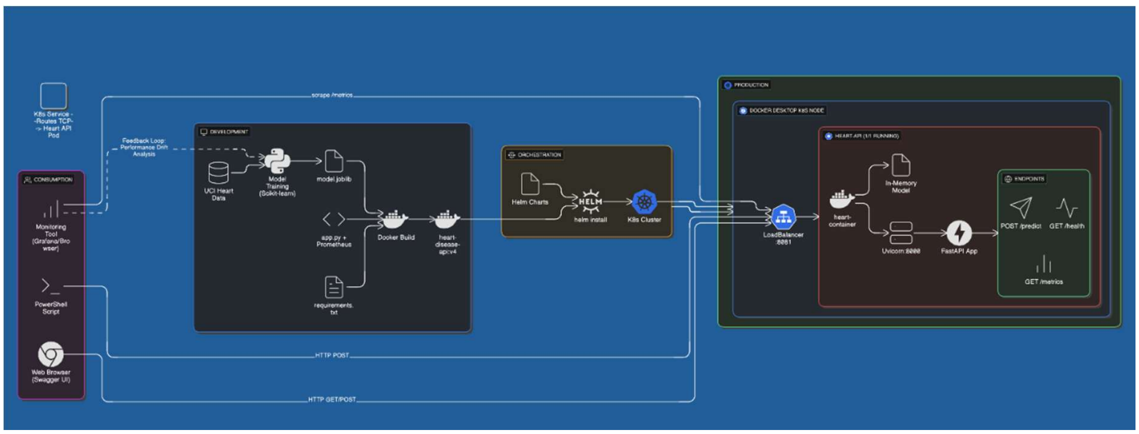


Diagram Flow Explanation:

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, Promet
│   └── 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
```

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	helm install heart-prediction
Testing	Automated Scripting	Invoke-RestMethod (PowerShell)
Web UI	OpenAPI Specification	http://localhost:8081/docs
Monitoring	Metrics Instrumentation	http://localhost:8081/metrics

8. Troubleshooting Log (Lessons Learned)

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

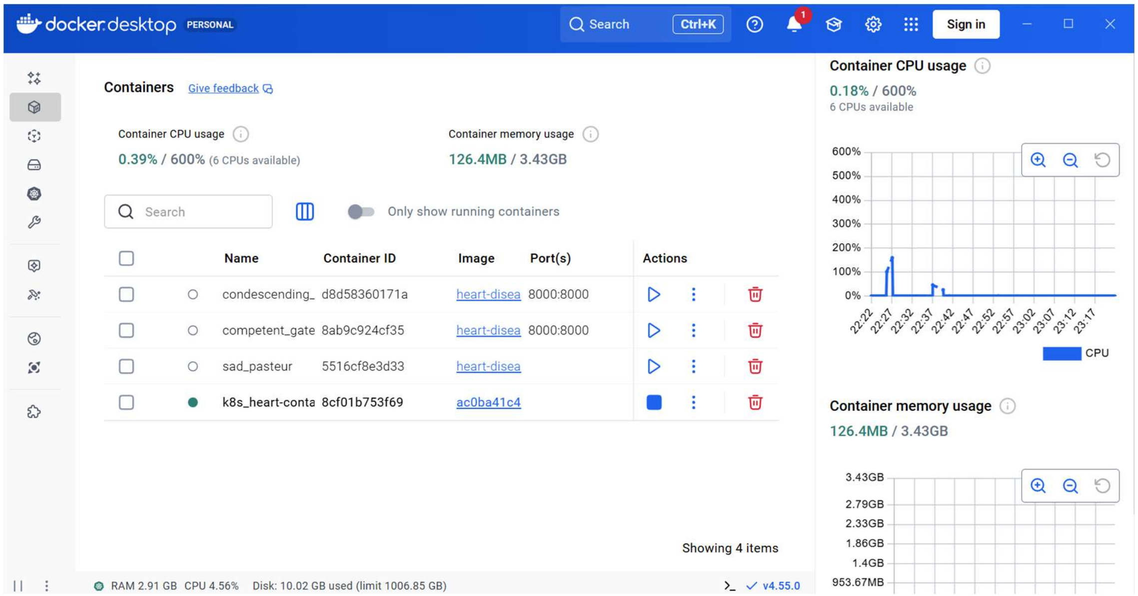
- **Port Conflict (Errno 98):** Resolved by identifying duplicate container definitions in deployment.yaml and ensuring only one process occupied Port 8000.
- **Service Discovery Gap:** Addressed empty "Endpoints" by synchronizing the selector labels between the Service and Deployment manifests.
- **Self-Healing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

9. Conclusion

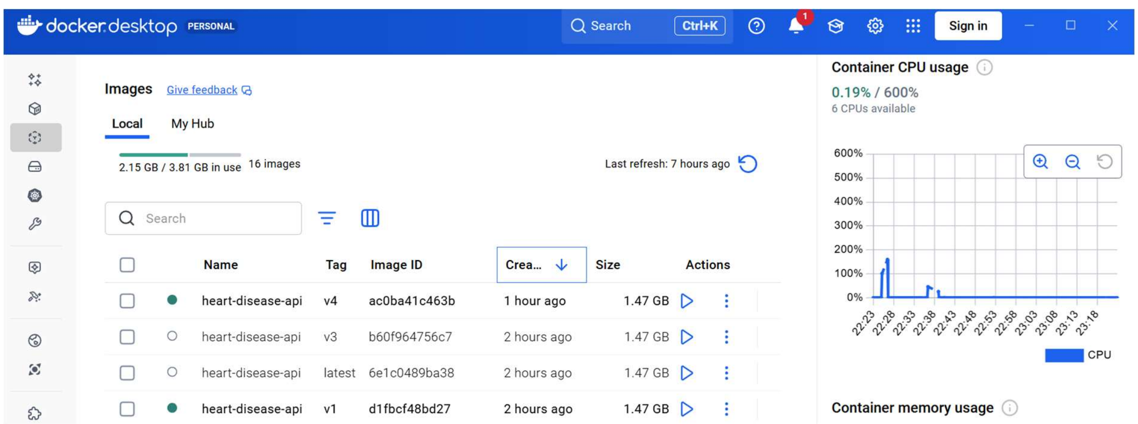
The implementation of the Heart Disease Prediction API demonstrates a robust MLOps architecture. By utilizing Kubernetes, the system ensures high availability and scalability, while the integration of Prometheus provides the necessary transparency for AI Security and performance auditing which is critical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

SCREENSHOTS

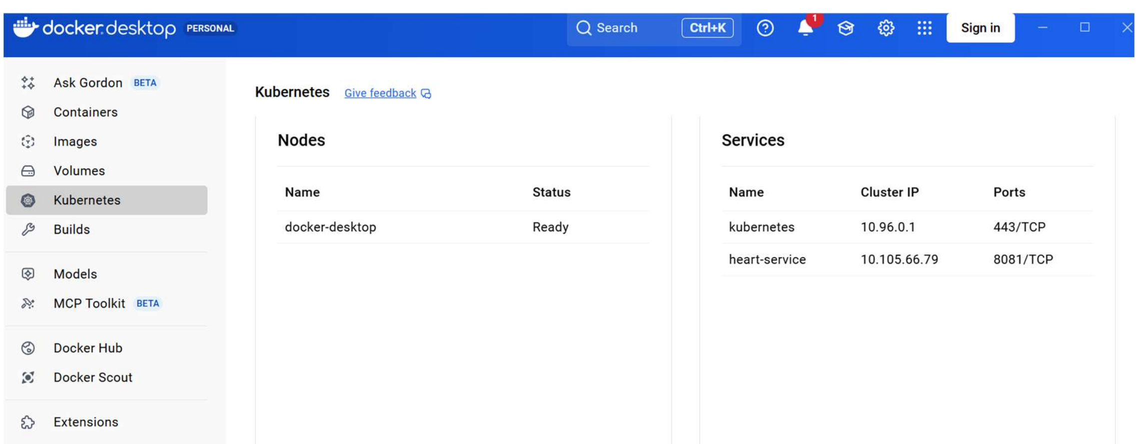
Screenshot: Docker Desktop 1 (Containers)



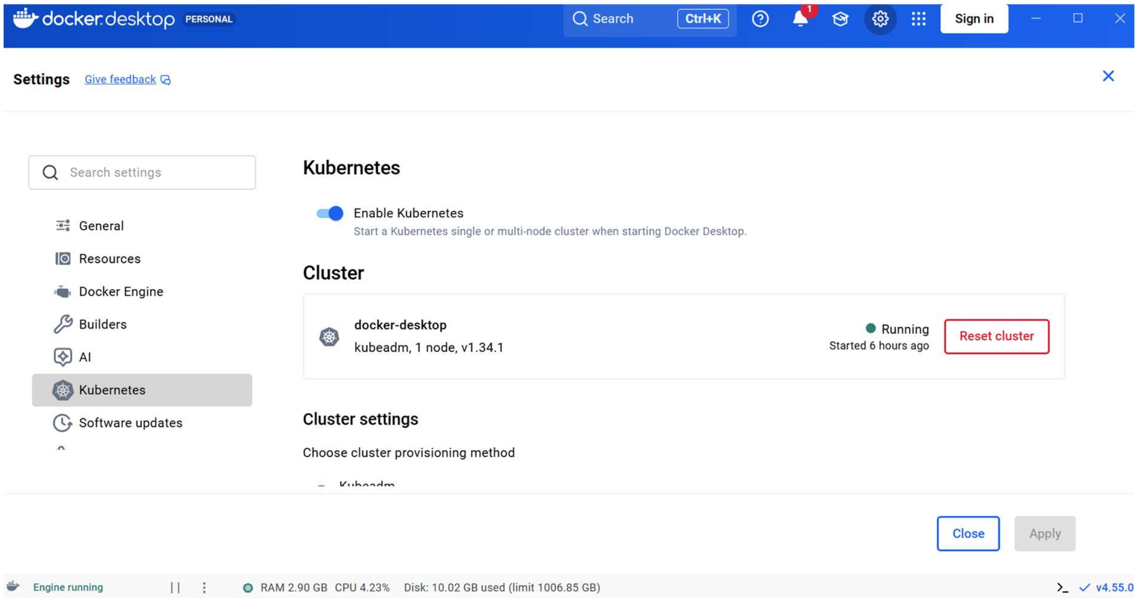
Screenshot: Docker Desktop 2 (Images)



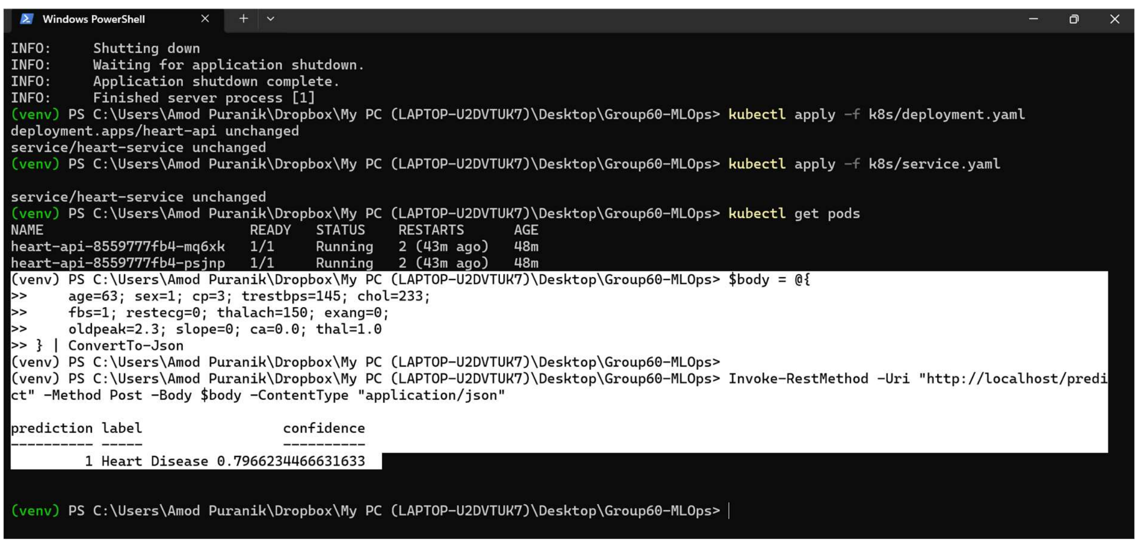
Screenshot: Docker Desktop 3 (Kubernetes)



Screenshot: Docker Desktop 4 (Settings)



Screenshot: PowerShell Output Prediction



Screenshot: Web UI