MLOps Assignment 1

Heart Disease MLOps Pipeline

Group Number: 80

BITS ID	Name	GitHub Username
2024AB05123	AISWARYA ANTONY	Aiswarya2020
2024AB05120	GAUTHAM KRISHNA M	GauthamKrM
2024AB05121	GOWRI ANILKUMAR	gowrianair
2024AB05127	MERIL SARA JOSE	Meril2003
2024AB05099	AMULYA K.M	Amulya2025123

1. SETUP AND INSTALLATION INSTRUCTIONS

1.1 Local Development

The local development setup begins with the installation of all required project dependencies. These dependencies are listed in the requirements.txt file and can be installed using the Python package manager.

```
pip install -r requirements.txt
```

After installing the dependencies, the machine learning model is trained. The training process includes downloading the dataset and training the model. The trained model is saved in the models/model directory.

```
python scripts/download_data.py
python src/train.py --n_estimators 100
```

Once the model training is completed, the API can be executed locally. Before starting the API, it must be ensured that the trained model exists in the models/model directory. The API is started using the following command:

```
python src/api.py
```

After execution, the API becomes accessible at http://0.0.0.0:8000.To validate the functionality of the application, automated tests are executed using the PyTest framework. These tests ensure that the individual components of the system function as expected.

```
pytest tests/
```

1.2 Kubernetes Deployment Using Minikube

The application can be deployed in a Kubernetes environment using Minikube. Prior to deployment, it is required that both kubectl and minikube are installed and running on the system.The Kubernetes configuration files are applied using the following command:

```
kubectl apply -f k8s/
```

This deployment creates multiple Kubernetes resources. These include the heart-disease-api Deployment, which runs the Docker image of the application, the heart-disease-service, which exposes the API using a LoadBalancer on port 80, and the Prometheus Deployment and Service used for monitoring purposes.

Since Minikube does not assign external IP addresses by default, LoadBalancer services are accessed using the minikube service command. The URLs for accessing the API service, Grafana service, and Prometheus service are obtained using the following commands:

```
minikube service heart-disease-service --url
minikube service grafana-service --url

minikube service prometheus-service --url
```

1.3 Monitoring via Prometheus

The application is instrumented for monitoring using the prometheus-fastapi-instrumentator library. This enables the collection of application-level metrics.

The metrics are exposed through the /metrics endpoint of the application. Prometheus scrapes these metrics based on the configuration defined in the k8s/monitoring.yaml file, enabling continuous monitoring of the application.

2. EDA AND MODELLING CHOICES

2.1 Exploratory Data Analysis (EDA) Summary

2.1.1 Dataset Quality and Structure
The dataset consists of 297 patient records with 14 clinical features and contains no missing values. All attributes were validated for numeric consistency during preprocessing, ensuring uniform and reliable input data for model training and CI/CD executions.

2.1.2 Target Class Distribution
The target variable shows a moderately balanced distribution, with 160 healthy cases (54%) and 137 heart disease cases (46%). This balance reduces the risk of majority-class bias and allows standard evaluation metrics such as accuracy, precision, recall, and ROC-AUC to remain meaningful without requiring additional class balancing techniques.

2.1.3 Statistical Characteristics
The average patient age is 54.5 years, with heart disease cases most concentrated in the 55–65 age range, indicating age as a significant clinical

factor. Certain clinical attributes, particularly cholesterol levels, exhibit wide variability, ranging from 126 mg/dl to 564 mg/dl, suggesting the presence of high-impact outliers that may influence model behavior.

2.1.4 Correlation and Predictive Features
Correlation analysis identifies thal, ca, and oldpeak as the strongest positive predictors of heart disease, with correlation values of 0.53, 0.46, and 0.42 respectively. In contrast, maximum heart rate (thalach) shows a negative correlation of -0.42, indicating that lower peak heart rates are often associated with higher disease risk.

2.2 Modeling Choices and Strategy

The modeling strategy is guided by insights from exploratory data analysis and the clinical importance of accurate heart disease prediction. Due to varying feature scales across clinical attributes, feature standardization using StandardScaler is applied for scale-sensitive models such as Logistic Regression and SVM. Categorical variables that are numerically encoded but represent distinct categories, such as chest pain type, are one-hot encoded to avoid unintended ordinal relationships.

Logistic Regression is selected as the baseline model due to its interpretability and strong linear relationships observed in the data. To capture non-linear interactions among clinical features, ensemble models such as Random Forest and Gradient Boosting are considered as advanced alternatives.Given the medical context, recall (sensitivity) is prioritized to minimize false negatives. Model robustness and generalizability are ensured using k-fold cross-validation, particularly due to the limited dataset size.

3.EXPERIMENT TRACKING SUMMARY

This project implements remote experiment tracking using MLflow hosted on DagsHub, rather than local tracking, to ensure persistence across CI/CD executions. The MLflow Tracking URI, username, and access token obtained from the DagsHub repository are securely stored as GitHub Secrets, enabling automated logging during pipeline runs without exposing credentials.

3.1 Trigger Points for Experiment Logging

Experiment tracking is automatically triggered through GitHub Actions on both pull requests and direct pushes to the main branch. During pull requests, validation checks are executed before allowing merge, ensuring that only verified code and model executions proceed. After merging into main, the workflow is re-triggered to execute the full pipeline, including dataset download, model training, experiment logging, Docker image creation, deployment on Minikube, and monitoring. This design ensures that experiment logging occurs during both pre-merge validation and post-merge execution stages.

3.2 MLflow Logging Behavior

As part of the CI/CD pipeline, a lightweight model training smoke test is executed in the build-and-test job, where the dataset is downloaded and the training script runs with a minimal estimator configuration. This run is logged to the remote MLflow server using credentials supplied through GitHub Secrets. After merge, a second job performs full model training, which is also logged remotely. As a result, both smoke test runs and final production training runs are recorded with their parameters, metrics, model artifacts, and generated plots.

3.3 Artifact Storage and Model Lineage

All experiment outputs are stored persistently in the MLflow artifact store hosted on DagsHub. These artifacts include trained model binaries, evaluation plots, execution logs, and intermediate files generated during training. This artifact storage establishes a complete model lineage, ensuring that every deployed model version can be traced back to its corresponding training run, configuration, and performance metrics.

3.4 Dashboard Visualization

Experiment history and performance metrics are visualized using the MLflow dashboard provided by DagsHub. Each pipeline execution automatically appears in the dashboard, displaying run statistics, parameters, metrics, and associated artifacts. Since both pull request smoke tests and post-merge training runs are logged, the dashboard always reflects the most recent execution, enabling easy comparison and reproducibility of experiments.

3.5 Validation and Deployment Continuity

Before deployment, the pipeline enforces automated validation through flake8 linting, pytest-based unit testing, and model training smoke tests. Only after all checks pass is a pull request merged into main. Following successful experiment logging, the pipeline proceeds with deployment automation by building and pushing a Docker image to Docker Hub, deploying the API on Minikube, restarting Kubernetes deployments, and serving predictions through running pods. Monitoring metrics are visualized using Grafana, ensuring that only validated and traceable models are deployed.

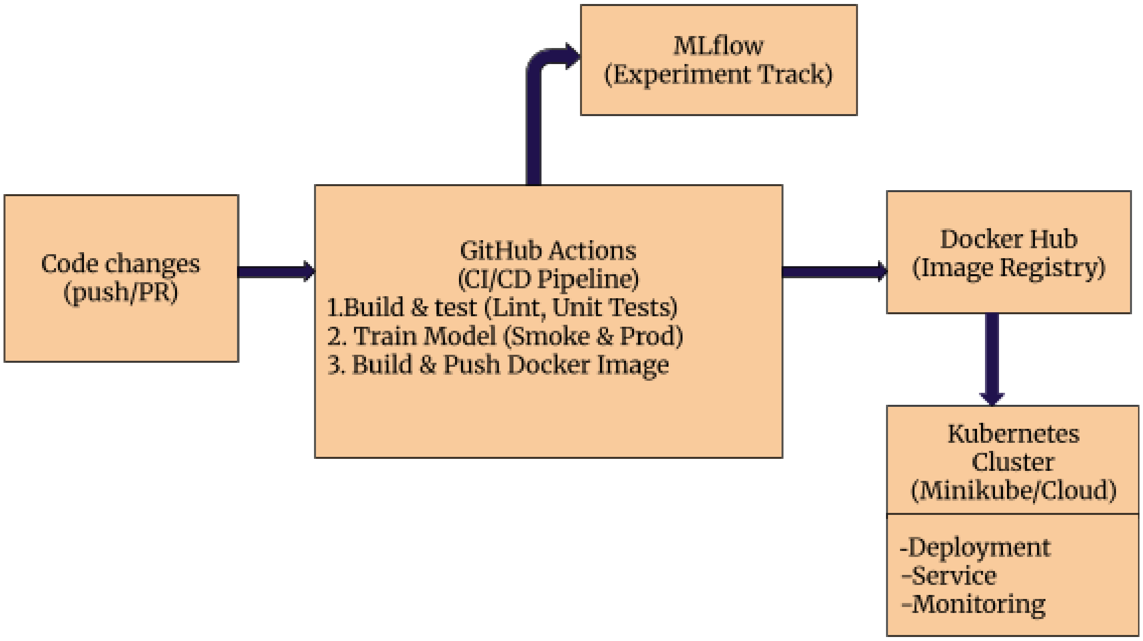
4. UNIT TEST SUMMARY

The project employs pytest as its unit testing framework, with all test cases organized under the tests/ directory. The testing suite validates both model training logic and API service reliability, ensuring correctness before deployment. Dependencies required for executing tests are installed from requirements.txt, using the command pip install -r requirements.txt. Local test execution is supported via pytest tests/ or python3 -m pytest tests/ -v from the project root.

The test suite is structured to maintain modularity and reusability. Shared fixtures such as sample datasets and mock models are defined in conftest.py. Core pipeline functionality is verified through test_train.py, which tests data loading, model evaluation metrics, and the overall training flow in train.py. Service layer validation is handled by test_api.py, which tests API endpoints and error handling logic within api.py.

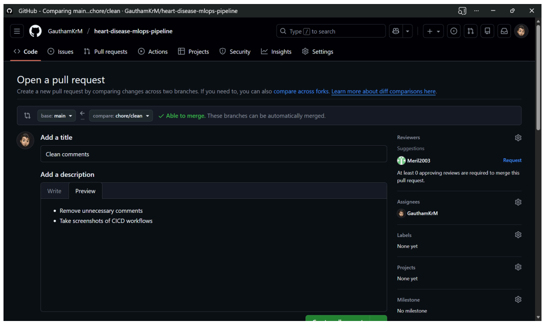
To enforce continuous validation, the testing framework is integrated into the GitHub Actions CI/CD workflow (mlops-pipeline.yml). Tests are automatically executed on every push and pull request targeting the main branch, preventing faulty code or models from progressing to deployment stages. This setup ensures automated regression testing, pipeline stability, and production readiness, following standard MLOps testing best practices.

5.ARCHITECTURE DIAGRAM



6.CI/CD AND DEPLOYMENT WORKFLOW SCREENSHOT

6.1 Raising a PR:



GitHub - Comparing main...chore/clean · GauthamKrM/heart-disease-mlops-pipeline

GauthamKrM / heart-disease-mlops-pipeline

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: main ← compare: chore/clean ✓ Able to merge. These branches can be automatically merged.

Add a title

Clean comments

Add a description

- Remove unnecessary comments
- Take screenshots of CICD workflows

Reviewers

Suggestions

Meril2003 Request

At least 0 approving reviews are required to merge this pull request.

Assignees

GauthamKrM

Labels

None yet

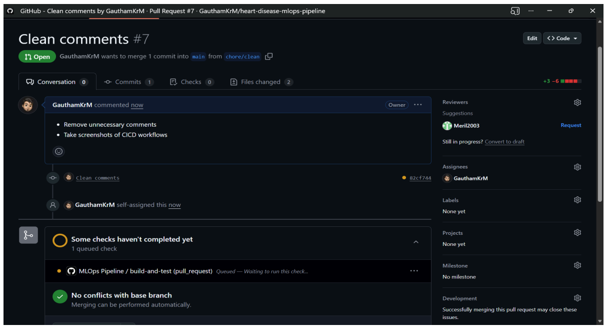
Projects

None yet

Milestone

No milestone

6.2 Checks running on PR - before merge: checks success:



GitHub - Clean comments by GauthamKrM · Pull Request #7 · GauthamKrM/heart-disease-mlops-pipeline

Clean comments #7

Open GauthamKrM wants to merge 1 commit into main from chore/clean

Conversation 0 Commits 1 Checks 0 Files changed 2 +3 −6

GauthamKrM commented now Owner

- Remove unnecessary comments
- Take screenshots of CICD workflows

Clean comments 82cf744

GauthamKrM self-assigned this now

Some checks haven't completed yet

1 queued check

MLOps Pipeline / build-and-test (pull_request) *Queued — Waiting to run this check...*

No conflicts with base branch

Merging can be performed automatically.

Reviewers

Suggestions

Meril2003 Request

Still in progress? Convert to draft

Assignees

GauthamKrM

Labels

None yet

Projects

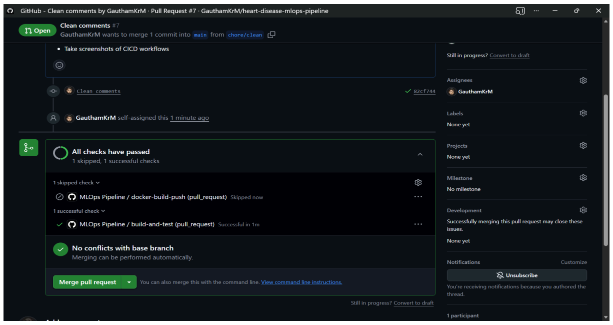
None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.



GitHub - Clean comments by GauthamKrM · Pull Request #7 · GauthamKrM/heart-disease-mlops-pipeline

Open Clean comments #7

GauthamKrM wants to merge 1 commit into main from chore/clean

- Take screenshots of CICD workflows

Clean comments ✓ 82cf744

GauthamKrM self-assigned this 1 minute ago

All checks have passed

1 skipped, 1 successful checks

1 skipped check

MLOps Pipeline / docker-build-push (pull_request) Skipped now

1 successful check

✓ MLOps Pipeline / build-and-test (pull_request) Successful in 1m

No conflicts with base branch

Merging can be performed automatically.

Merge pull request — You can also merge this with the command line. View command line instructions.

Still in progress? Convert to draft

Still in progress? Convert to draft

Assignees

GauthamKrM

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

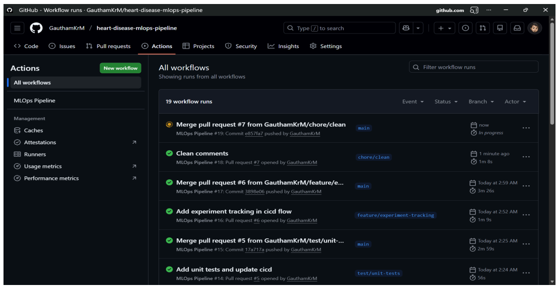
Notifications Customize

Unsubscribe

You're receiving notifications because you authored the thread.

1 participant

6.3 Actions triggered on merging the PR:



GitHub - Workflow runs · GauthamKrM/heart-disease-mlops-pipeline github.com

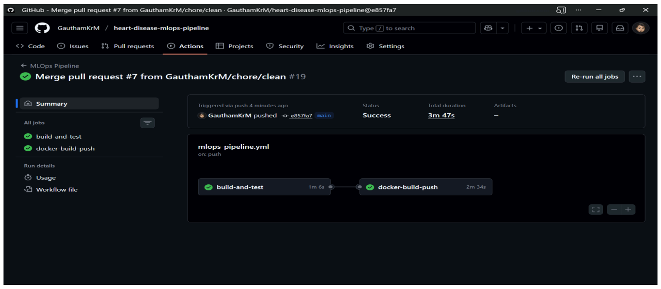
GauthamKrM / heart-disease-mlops-pipeline

All workflows

Showing runs from all workflows

19 workflow runs

- Merge pull request #7 from GauthamKrM/chore/clean — main — now, *In progress*
  MLOps Pipeline #19: Commit e857fa7 pushed by GauthamKrM
- Clean comments — chore/clean — 1 minute ago, 1m 8s
  MLOps Pipeline #18: Pull request #7 opened by GauthamKrM
- Merge pull request #6 from GauthamKrM/feature/e... — main — Today at 2:59 AM, 3m 26s
  MLOps Pipeline #17: Commit 3898e06 pushed by GauthamKrM
- Add experiment tracking in cicd flow — feature/experiment-tracking — Today at 2:52 AM, 1m 9s
  MLOps Pipeline #16: Pull request #6 opened by GauthamKrM
- Merge pull request #5 from GauthamKrM/test/unit-... — main — Today at 2:25 AM, 2m 59s
  MLOps Pipeline #15: Commit 17a717a pushed by GauthamKrM
- Add unit tests and update cicd — test/unit-tests — Today at 2:24 AM, 56s
  MLOps Pipeline #14: Pull request #5 opened by GauthamKrM



GitHub - Merge pull request #7 from GauthamKrM/chore/clean · GauthamKrM/heart-disease-mlops-pipeline@e857fa7

GauthamKrM / heart-disease-mlops-pipeline

← MLOps Pipeline

Merge pull request #7 from GauthamKrM/chore/clean #19

Triggered via push 4 minutes ago: GauthamKrM pushed e857fa7 main

Status: Success

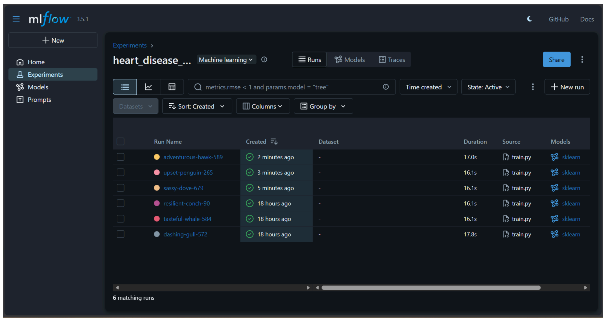
Total duration: 3m 47s

Artifacts: –

mlops-pipeline.yml

on: push

6.4 Mlflow - experiment tracking dashboard:



Experiments ›

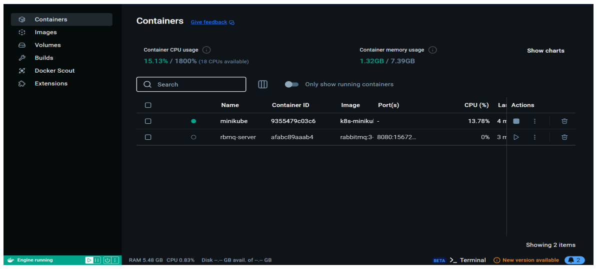
heart_disease_... Machine learning

metrics.rmse < 1 and params.model = "tree"

☐	Run Name	Created	Dataset	Duration	Source	Models
☐	adventurous-hawk-589	2 minutes ago	-	17.0s	train.py	sklearn
☐	upset-penguin-265	3 minutes ago	-	16.1s	train.py	sklearn
☐	sassy-dove-679	5 minutes ago	-	16.1s	train.py	sklearn
☐	resilient-conch-90	18 hours ago	-	16.1s	train.py	sklearn
☐	tasteful-whale-584	18 hours ago	-	16.1s	train.py	sklearn
☐	dashing-gull-572	18 hours ago	-	17.8s	train.py	sklearn

6 matching runs

6.5 Running minikube:



Containers

Container CPU usage
15.13% / 1800% (18 CPUs available)

Container memory usage
1.32GB / 7.39GB

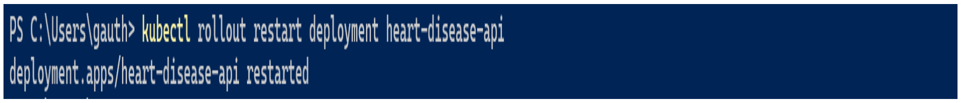
Only show running containers

☐		Name	Container ID	Image	Port(s)	CPU (%)	La:	Actions
☐	●	minikube	9355479c03c6	k8s-minikul	-	13.78%	4 m	
☐	○	rbmq-server	afabc89aaab4	rabbitmq:3-	8080:15672...	0%	3 m	

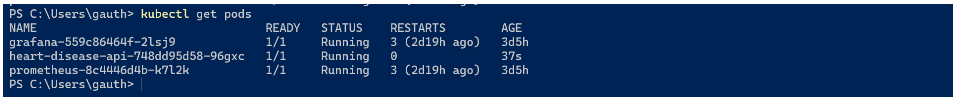
Showing 2 items

RAM 5.48 GB CPU 0.83% Disk --.-- GB avail. of --.-- GB

6.6 Restarting deployments for heart disease api:

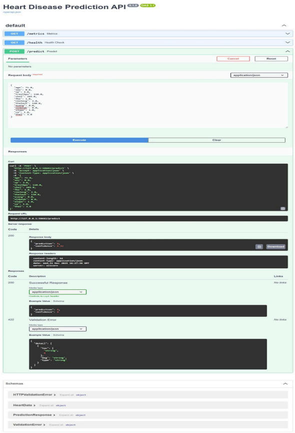


```
PS C:\Users\gauth> kubectl rollout restart deployment heart-disease-api
deployment.apps/heart-disease-api restarted
```



```
PS C:\Users\gauth> kubectl get pods
NAME                                  READY   STATUS    RESTARTS        AGE
grafana-559c86464f-2lsj9              1/1     Running   3 (2d19h ago)   3d5h
heart-disease-api-748dd95d58-96gxc    1/1     Running   0               37s
prometheus-8c4446d4b-k7l2k            1/1     Running   3 (2d19h ago)   3d5h
PS C:\Users\gauth>
```

6.7 API demo:



Heart Disease Prediction API

/openapi.json

/metrics Metrics

/health Health Check

/predict Predict

Parameters

No parameters

Request body required application/json

```
{
  "age": 71.0,
  "sex": 0.0,
  "cp": 3.0,
  "trestbps": 110.0,
  "chol": 265.0,
  "fbs": 1.0,
  "restecg": 2.0,
  "thalach": 130.0,
  "exang": 0.0,
  "oldpeak": 0.0,
  "slope": 1.0,
  "ca": 1.0,
  "thal": 3.0
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:58682/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "age": 71.0,
  "sex": 0.0,
  "cp": 3.0,
  "trestbps": 110.0,
  "chol": 265.0,
  "fbs": 1.0,
  "restecg": 2.0,
  "thalach": 130.0,
  "exang": 0.0,
  "oldpeak": 0.0,
  "slope": 1.0,
  "ca": 1.0,
  "thal": 3.0
}'
```

Request URL

```
http://127.0.0.1:58682/predict
```

Server response

Code Details

200

Response body

```
{
  "prediction": 0,
  "confidence": 0.92
}
```

Response headers

```
 content-length: 34
 content-type: application/json
 date: Wed,31 Dec 2025 14:27:36 GMT
 server: uvicorn
```

Responses

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

```
{
  "prediction": 0,
  "confidence": 0
}
```

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

Schemas

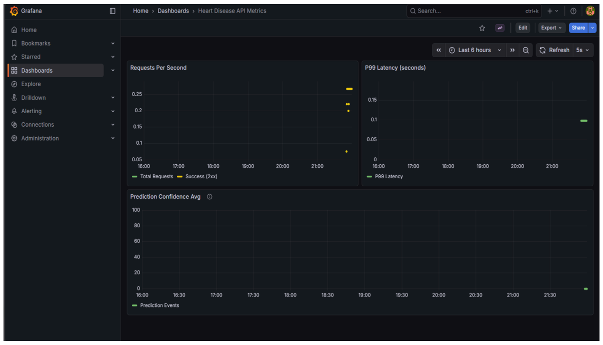
HTTPValidationError › Expand all object

HeartData › Expand all object

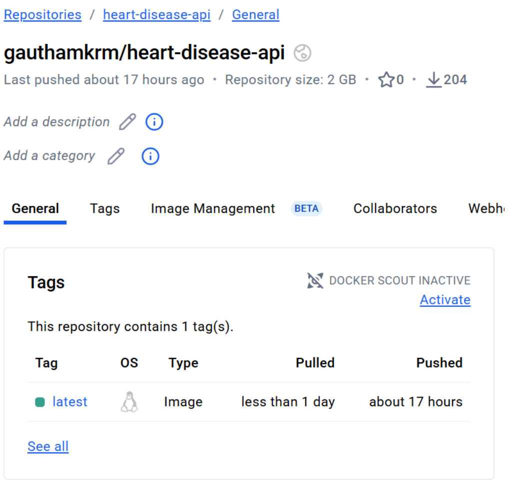
PredictionResponse › Expand all object

ValidationError › Expand all object

6.8 Grafana:



6.9 API image in dockerhub pushed from pipeline:



Repositories / heart-disease-api / General

gauthamkrm/heart-disease-api

Last pushed about 17 hours ago · Repository size: 2 GB · ☆0 · 204

Add a description

Add a category

General | Tags | Image Management BETA | Collaborators | Webh

Tags

DOCKER SCOUT INACTIVE
Activate

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	less than 1 day	about 17 hours

See all

7.IMPORTANT LINKS

GitHub Repository Link: *github-repo*

Workflow Screenshots: *screenshots*

CICD Workflow Screen Recording: *recording-link*