CHEST DISEASE CLASSIFICATION

USING CT SCAN IMAGES

TECHNOLOGY STACK

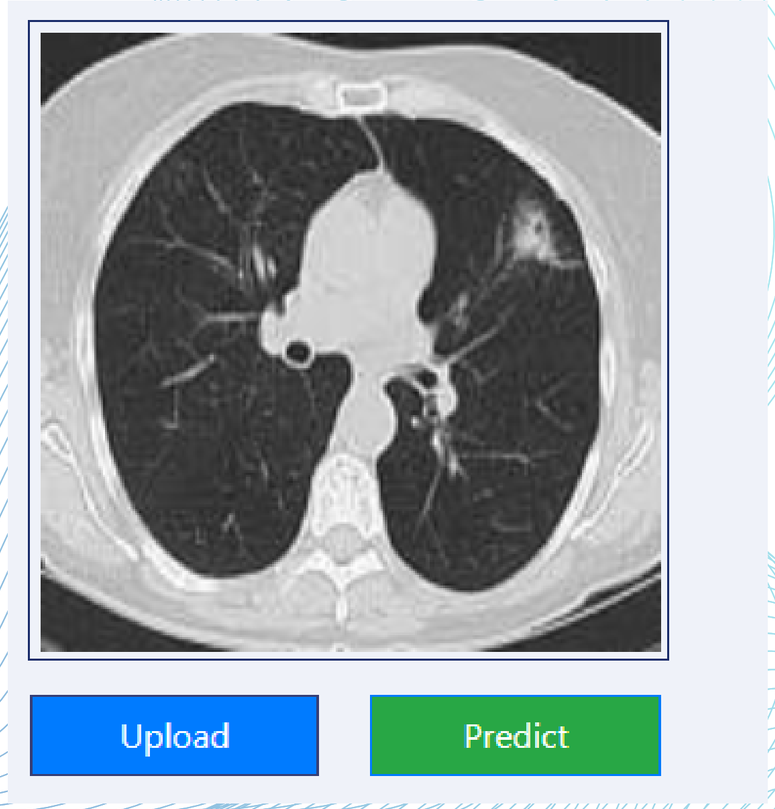
- **Languages & Tools: Python, TensorFlow, Keras, OpenCV, Jupyter Notebook, Flask, HTML, CSS, Docker**

- **CI/CD & Infrastructure: Jenkins, Docker, AWS EC2**

- **Version Control & Experiment Tracking: Git, DVC, MLflow, Dagshub**

DISCRIPTION

The "Chest Disease Classification from Chest CT Scan Images" project utilizes advanced deep learning techniques to classify various chest diseases from CT scan images. The project integrates a comprehensive pipeline from data ingestion to deployment, leveraging both on-premises and cloud resources..



Github-link

KEY COMPONENTS:

Data Pipeline:

- Data Version Control (DVC): Manages and versions datasets, ensuring consistency and reproducibility.
- Configuration Files: config.yaml and params.yaml define essential parameters for data processing and model training.

Model Training & Evaluation:

- Convolutional Neural Networks (CNNs): Used for building the classification model.
- Scripts & Source Code: Located in the src/cnnClassifier directory, including the core application logic (app.py).

KEY COMPONENTS:

Deployment & Monitoring:

- AWS EC2: Utilized for hosting and deploying the application, providing scalable cloud infrastructure.
- Jenkins: Facilitates continuous integration and deployment (CI/CD), automating the build, test, and deployment processes.

Experiment Tracking & Management:

- MLflow & Dagshub: Integrated for experiment tracking, enabling detailed logging of parameters, metrics, and artifacts.

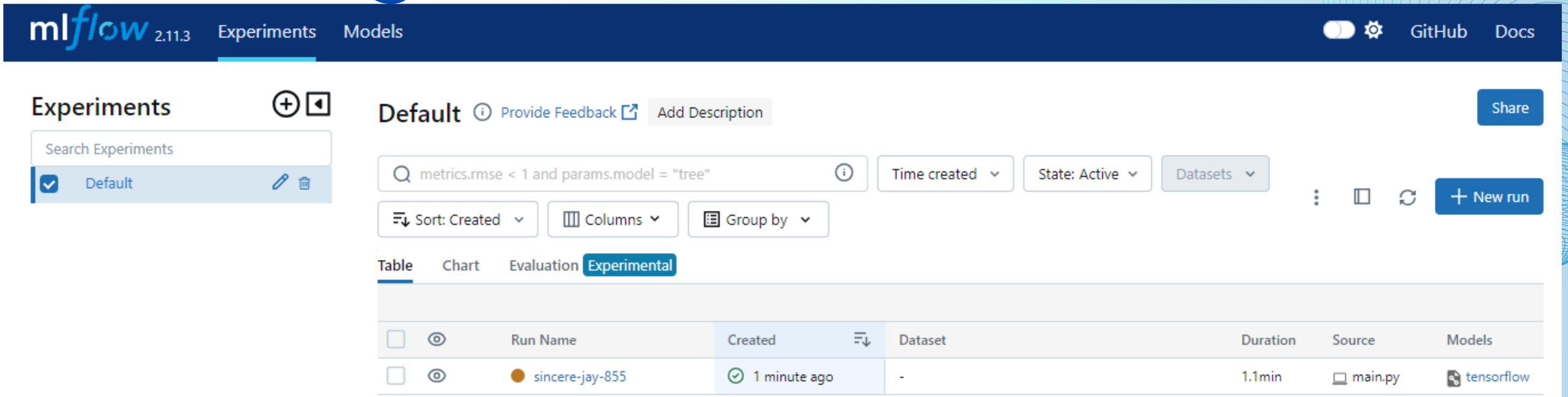
Web Application:

- Developed using Flask, the application provides a user-friendly interface for interacting with the model. The frontend is styled using CSS and HTML templates located in the static/css and templates directories

REPOSITORY STRUCTURE:

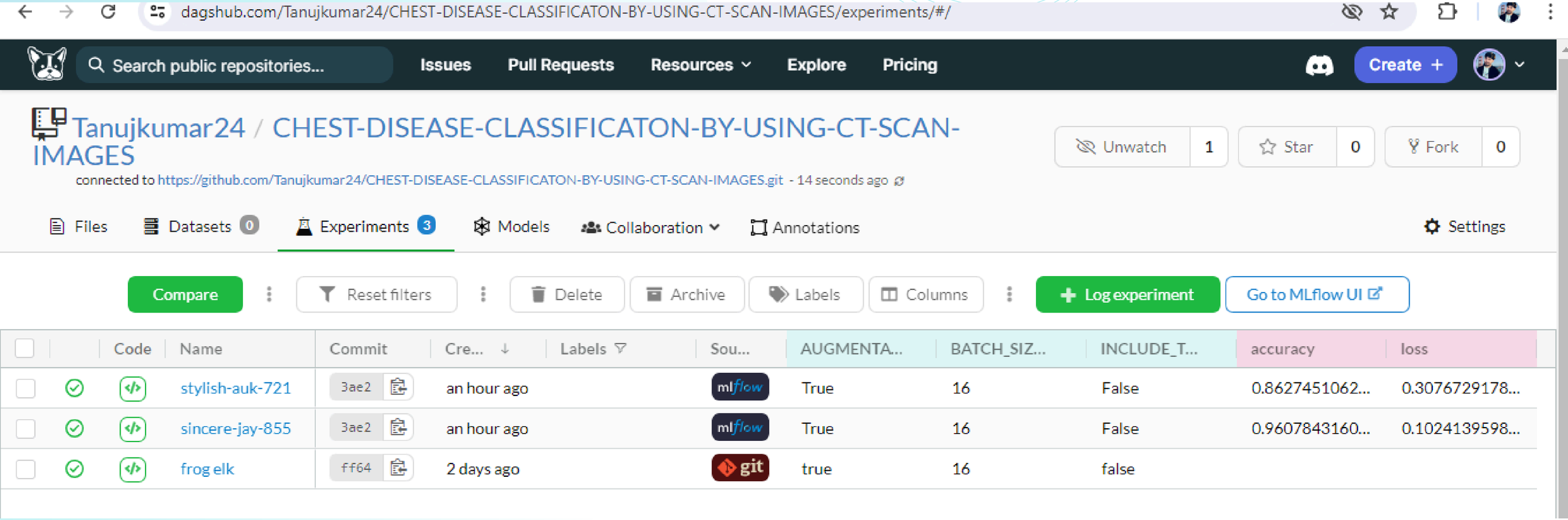
- .dvc: Contains DVC (Data Version Control) files for managing data and models, ensuring versioning and reproducibility.
- .github/workflows & .jenkins: Includes CI/CD pipeline configurations for automated testing and deployment using GitHub Actions and Jenkins.
- .vscode: Contains workspace settings and configurations.
- artifacts: Stores outputs and artifacts generated during the project, such as model predictions and logs.
- config: Houses configuration files (config.yaml, params.yaml) that define parameters and settings for data ingestion, model training, and evaluation.
- mlruns/0: Directory containing experiment data tracked using MLflow.
- research: Contains exploratory data analysis and research notebooks, such as the data ingestion notebook.
- scripts: Includes scripts for various tasks, such as data processing and model evaluation.
- src/cnnClassifier: The main source code directory for the CNN-based classifier, including app.py and other modules.
- static/css & templates: Frontend resources, including CSS files and HTML templates for the web application.
- Dockerfile & docker-compose.yaml: Configuration files for Docker, enabling containerization of the application for consistent environments across different stages of development and production.
- app.py & main.py: Core application scripts for running the web server and managing model predictions.
- dvc.yaml: DVC pipeline file that defines the sequence of stages in the data pipeline.
- requirements.txt: Lists all Python dependencies required for the project.
- setup.py: Script for installing the package and managing dependencies.
- README.md: Provides an overview of the project, including setup instructions and usage.

Experiment Tracking and Model Management with MLflow



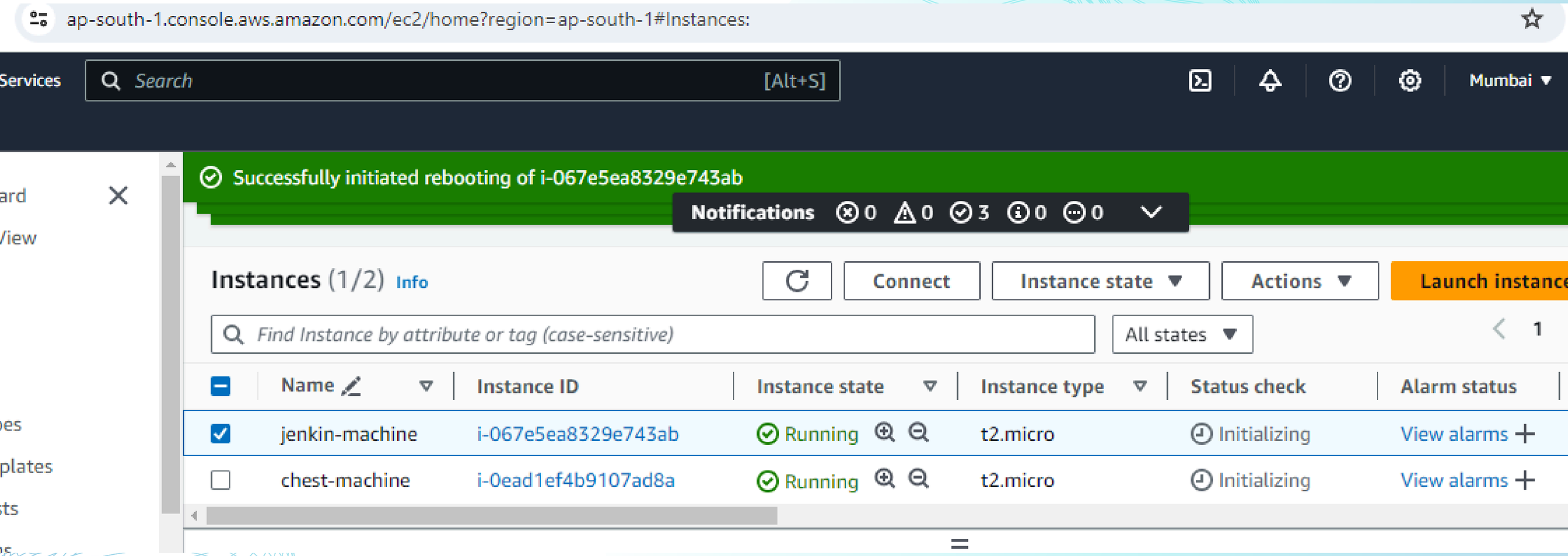
In the Chest Disease Classification project, MLflow was used to track experiment runs, log parameters and metrics, manage model versions, and streamline the deployment process. The primary reason for using MLflow was to ensure reproducibility and transparency in the machine learning workflow

MLflow Integration: Achievements



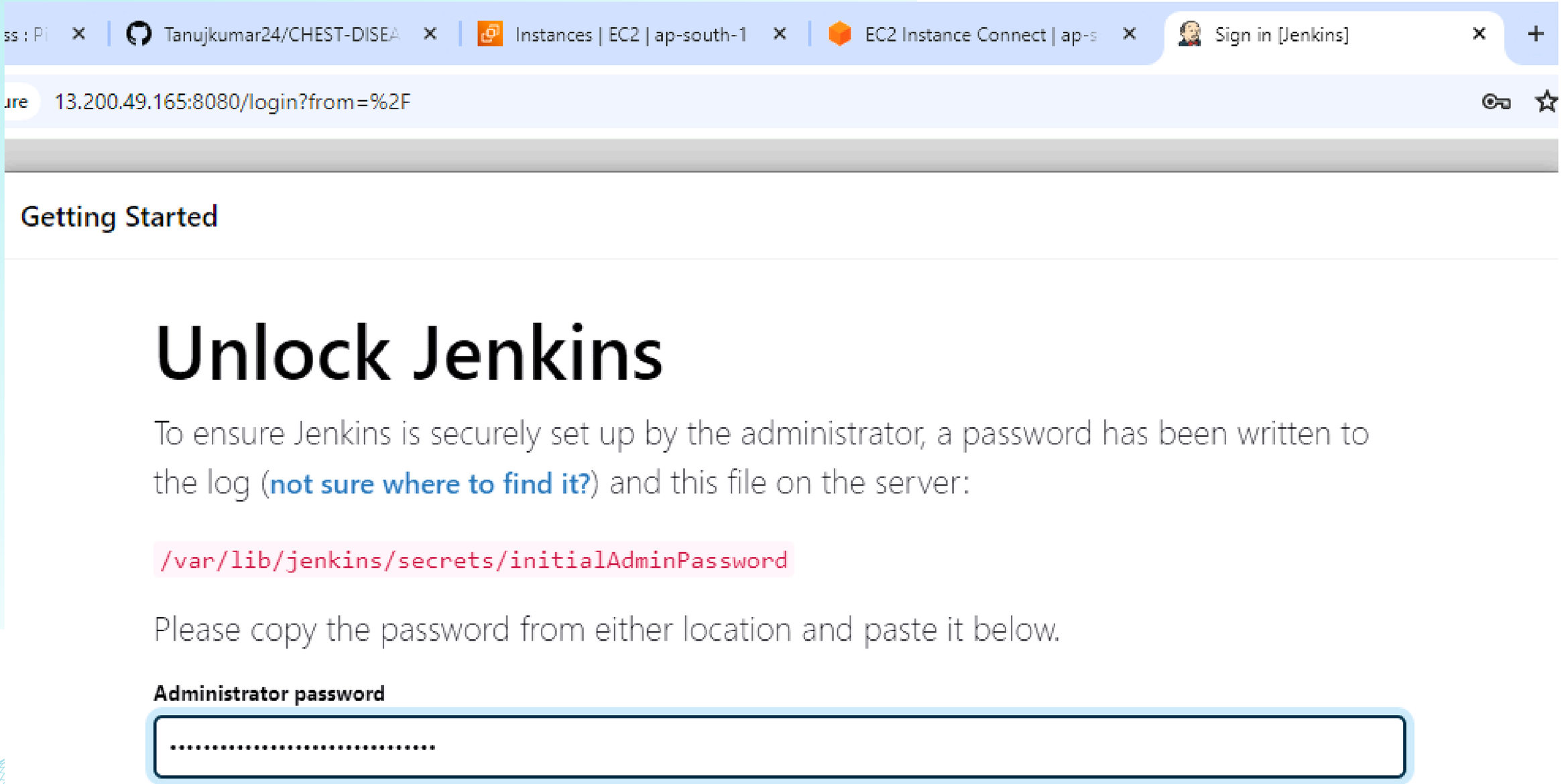
we achieved efficient experiment tracking, centralized model management, and improved collaboration. This setup enabled quick comparisons of different model versions and facilitated the deployment of the best-performing models. Overall, MLflow helped in maintaining a well-organized and scalable machine learning pipeline.

EC2 SETUP AND CONFIGURATION



- **Two EC2 Instances: Deployed two EC2 instances; one for running Jenkins and another for executing the application.**
- **Jenkins Instance Configuration: Installed Jenkins on the first EC2 instance and configured it for CI/CD.**
- **Application Instance Configuration: Set up the second EC2 instance to pull and run Docker images.**

JENKINS INTEGRATION AND CONFIGURATION



JENKINS PIPELINE SETUP AND MANAGEMENT

Dashboard > chest-project > Configuration

Configure

- General
- Advanced Project Options
- Pipeline

Repository browser ?

(Auto)

Additional Behaviours

Add

- **Pipeline Creation: Created a Jenkins pipeline to automate the build, test, and deployment process using a Jenkinsfile.**
- **Integration with GitHub: Configured Jenkins to trigger builds based on code commits from GitHub.**
- **Docker Build and Push: Automated the process of building Docker images and pushing them to Amazon ECR (Elastic Container Registry).**

Script Path ?

.jenkins/jenkinsfile

Lightweight checkout ?

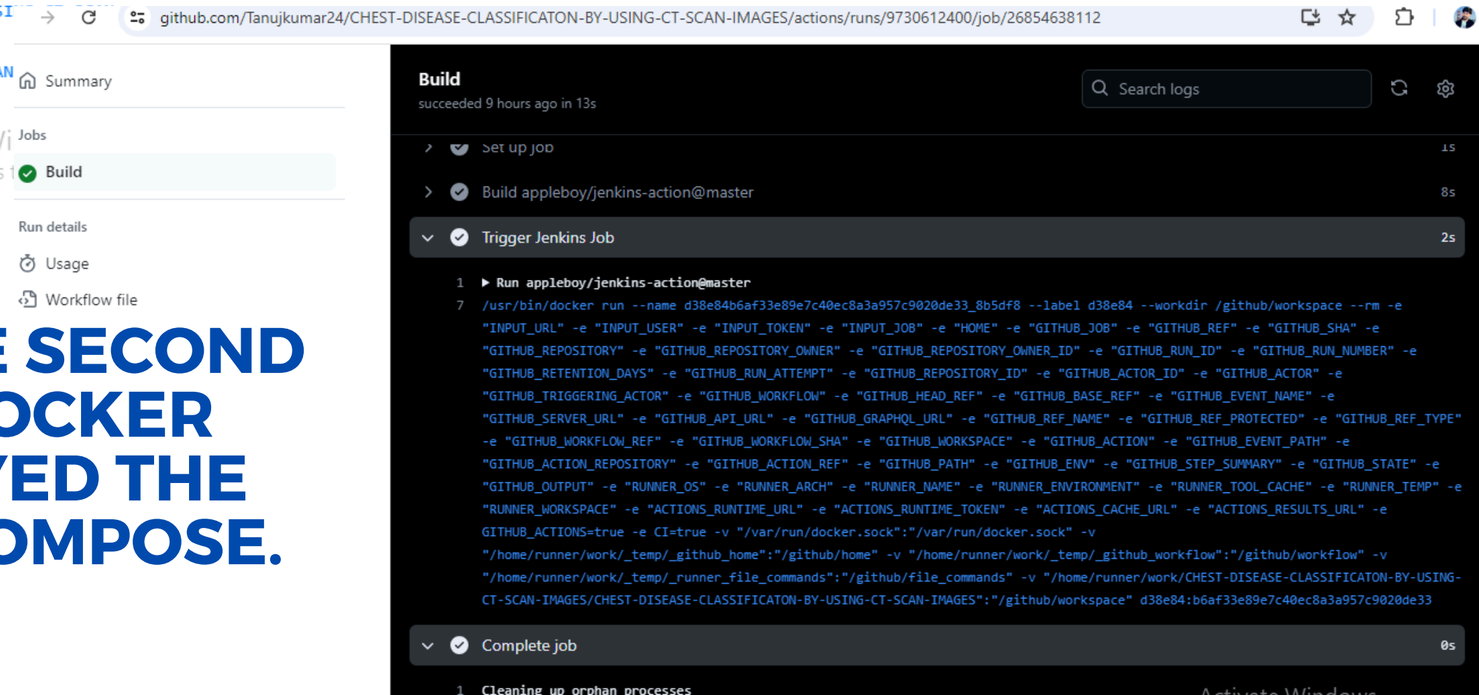
Pipeline Syntax

Save Apply

DEPLOYMENT WORKFLOW:

```
Started by user Tanujkumar
Obtained .jenkins/jenkinsfile from git https://github.com/Tanujkumar24/CHEST-DISEASE-CLASSIFICATON-BY-USING-CT-SCAN-
IMAGES.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/chest-project
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/Tanujkumar24/CHEST-DISEASE-CLASSIFICATON-BY-USING-CT-SCAN-IMAGES.git
 > git init /var/lib/jenkins/workspace/chest-project # timeout=10
Fetching upstream changes from https://github.com/Tanujkumar24/CHEST-DISEASE-CLASSIFICATON-BY-USING-CT-SCAN-IMAGES.git
 > git --version # timeout=10
 > git --version # 'git version 2.43.0'
 > git fetch --tags --force --progress -- https://github.com/Tanujkumar24/CHEST-DISEASE-CLASSIFICATON-BY-USI
IMAGES.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/Tanujkumar24/CHEST-DISEASE-CLASSIFICATON-BY-USING-CT-SCAN
timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
```

CONTINUOUS INTEGRATION: JENKINS PULLED CODE FROM GITHUB, BUILT DOCKER IMAGES, AND PUSHED THEM TO ECR.

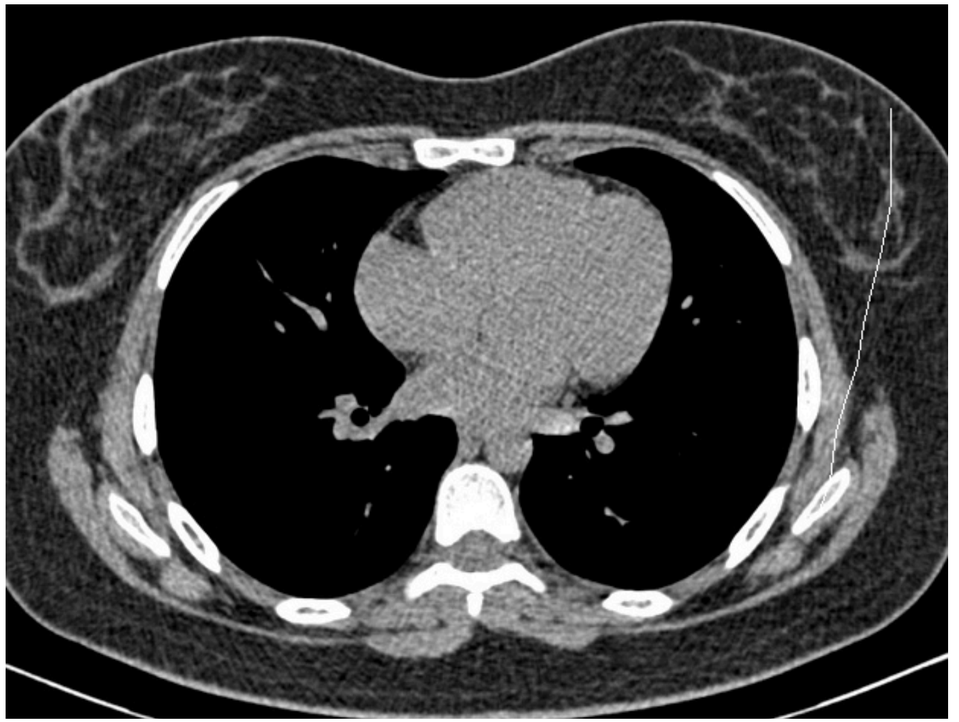


CONTINUOUS DEPLOYMENT: THE SECOND EC2 INSTANCE PULLED THE DOCKER IMAGE FROM ECR AND DEPLOYED THE APPLICATION USING DOCKER COMPOSE.

DATASET

we have used the CT SCAN Images of Normal Patient and patient suffering from Adenocarcinoma (a type of cancer that starts in glands that line the insides of the organs)

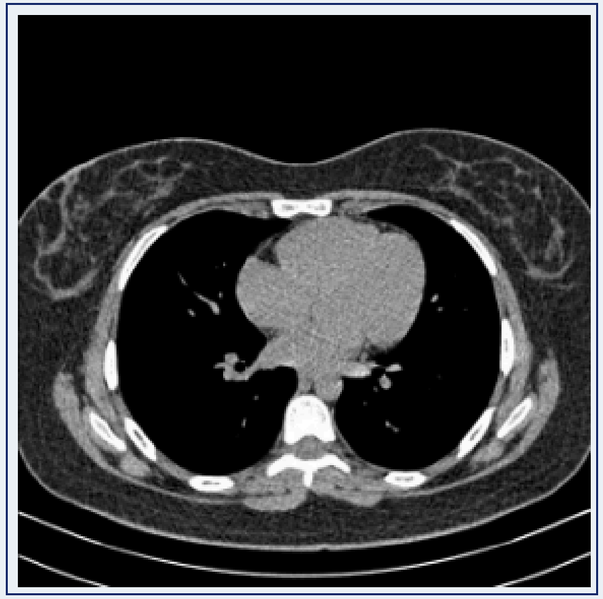
ADENOCARCINOMA



NORMAL

RESULTS

Chest Cancer Classification



Upload

Predict

Prediction Results


```
{
  "image": "Normal"
}
```