Automated Deep-Learning based Image Captioning System with Dev-Ops Driven Deployment

Software Production Engineering - Final Project
IIITB M.Tech 2nd Sem, 2025

Aman Bahuguna
(MT2024018)

Soumik Pal
(MT2024153)

DevOps is invaluable in today’s IT topology, because it bridges the gap between software development and IT operations, enabling faster and more reliable delivery of applications and services. By fostering collaboration, automating repetitive tasks like testing, deployment, and infrastructure management, DevOps reduces errors and accelerates release cycles. This leads to improved product quality, quicker feedback loops, and greater agility in responding to changing business needs, ultimately driving better customer satisfaction and competitive advantage.

In this project, we design and implement a comprehensive DevOps framework to fully automate the Software Development Life Cycle (SDLC) using cutting-edge DevOps tools. Focusing on the specialized domain of Machine Learning, we seamlessly integrate DevOps principles to unlock its benefits—thus pioneering a robust and efficient MLOps pipeline.

1. ML Component :

Problem Statement :

Given an image, generate a caption for it which would serve as the textual description of the image. This task combines techniques from both Computer Vision and Natural Language Processing, making it a multidisciplinary challenge.

Model Architecture chosen :

Typically, image captioning models follow an encoder-decoder architecture—where the image is first transformed into a meaningful feature representation by the encoder, and then the decoder translates this representation into a coherent descriptive sentence.

In our project, we have used a DenseNet CNN to extract image features and an LSTM to generate text by combining image and word embeddings.

Dataset Used : https://www.kaggle.com/datasets/adityajn105/flickr8k

We used a subset of the standard Flickr dataset, called Flickr8K that has ~8K images and 5 captions for each image, creating around 40K image, caption pairs. This was split into a training dataset, validation dataset and testing dataset (80% - 10% - 10% split).

Preprocessing :

To make the training, validation and testing process quicker, the feature extraction of all 8k images, using DenseNet201 model from tensorflow library, were done beforehand and stored as .pkl artifact in models folder. This was later loaded and used during training, validation and testing as a simple lookup.

The caption text preprocessing involved cleaning and normalizing sentences by lowercasing, removing special characters, extra spaces, and single letters, and adding start and end tags for model training.

Training the Model :

The model was trained over 12 epochs using a carefully selected learning rate and batch size to optimize performance. To ensure efficient training and prevent overfitting, model checkpointing was employed to save the best-performing model based on validation loss, allowing for recovery and reuse of the most effective weights. Additionally, early stopping was implemented with a patience of 3, meaning training would halt if the validation loss did not improve for three consecutive epochs—thereby saving time and computational resources while maintaining model generalization.

The model and tokenizer used were saved in the local models directory to be later reused during testing and inference.

Testing the Model :

The save model was loaded and tested on the test dataset to evaluate BLEU score. The BLEU score evaluates the quality of generated captions by measuring their similarity to reference captions based on overlapping n-grams.

Inference and Frontend App :

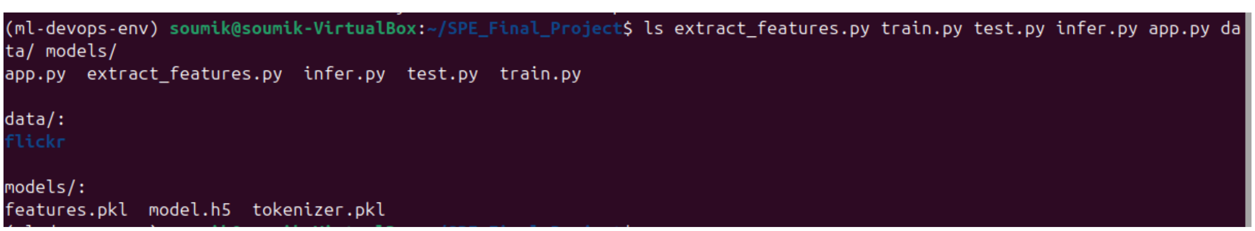
The inference script has no main method and instead exposes its method as an API which is called from app.py, a streamlit based frontend where a user can submit their image and the api call will return the generated caption which is then displayed to the user.

Project Structure based on ML Components :

Scripts :
- extract_features.py
- train.py
- test.py
- infer.py

Dataset :
- data/flickr/captions.txt
- data/flickr/images

Model and Artifacts :
- Features.pkl
- Tokenizer.pkl
- model.h5



The entire Machine Learning project was developed inside a Virtual Environment to ensure all dependencies could be replicated consistently.

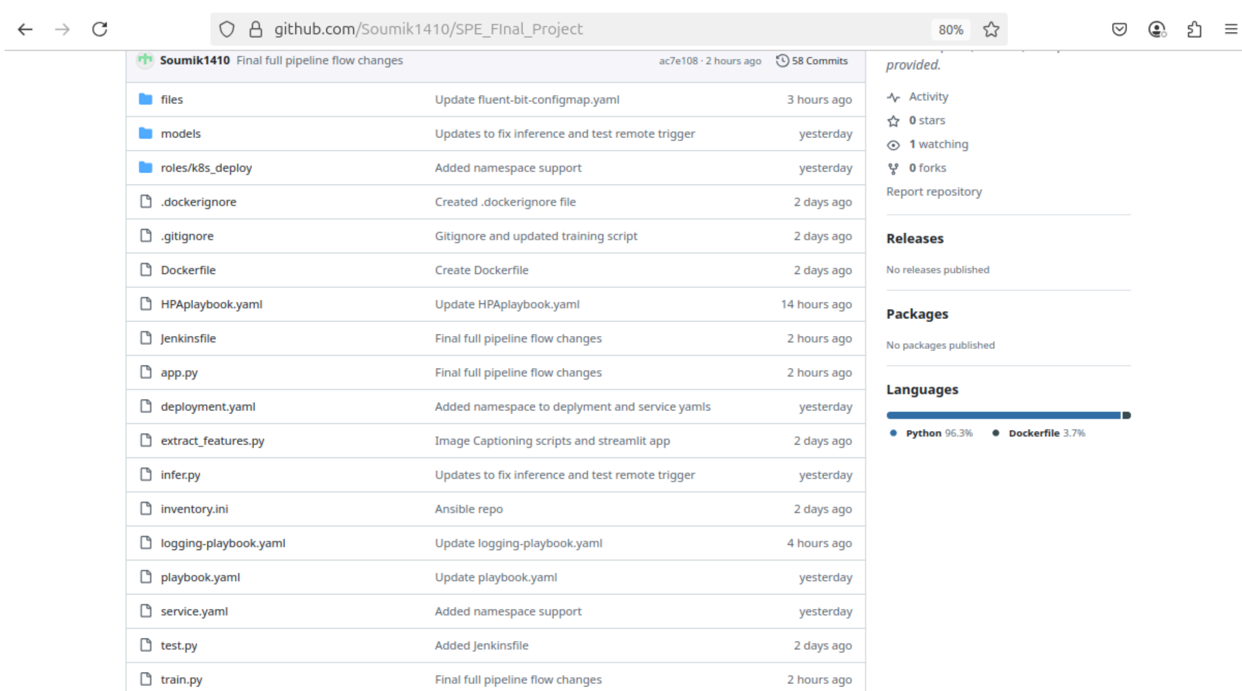
2. Code versioning - Git, Github

Git is a distributed version control system for tracking code changes, while GitHub is a cloud-based platform for hosting and collaborating on Git repositories.

In our project we are versioning all the scripts used in the ML pipeline such as extract_featues.py, train.py, test.py, infer.py & [app.py](app.py) along with the certain artifacts that are saved during training which will be later needed during testing and inference such as the tokenizer fit on training data captions for word to embedding mappings.

Along with these, other scripts used later in the pipeline such as Jenkinsfile, Dockerfile, Ansible playbooks, inventory file, Ansible Roles, Kubernetes Manifests to deploy the application, enable logging and monitoring and HPA are also versioned in this repository.

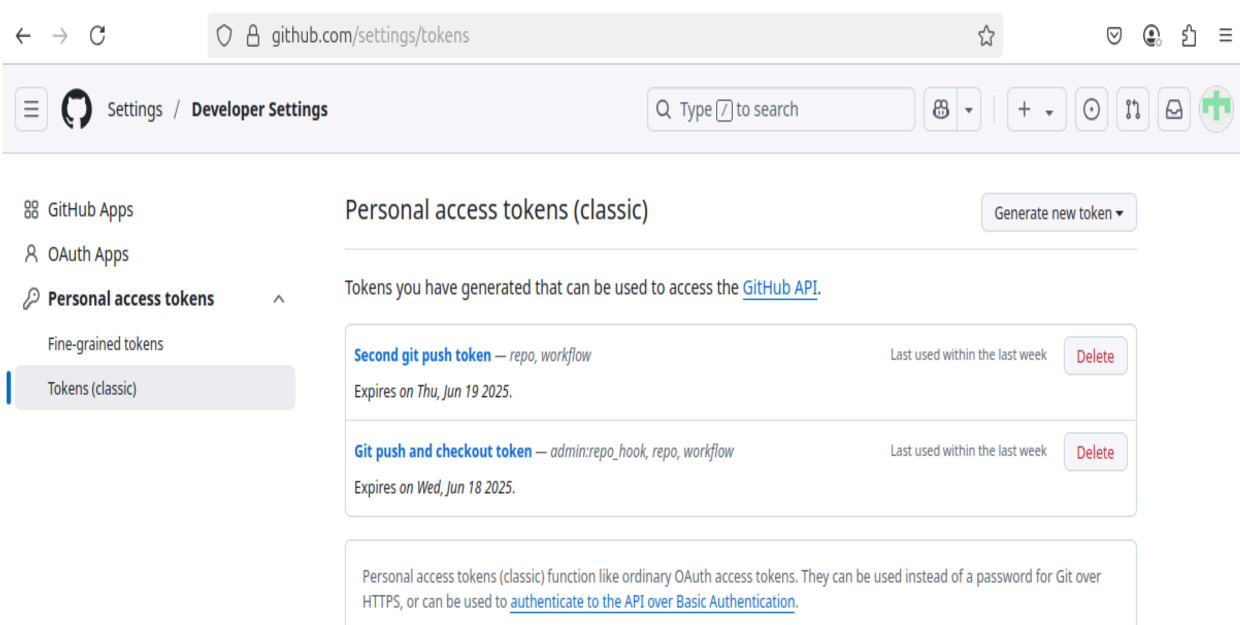
Final project repository :



Other large files present in the local development virtual environment such as the dataset and the trained models were not pushed to the Github repository by maintaining a .gitignore file

```
(ml-devops-env) soumik@soumik-VirtualBox:~/SPE_Final_Project$ cat .gitignore
__pycache__/
*.pyc
*.log
*.h5
data/flickr/Images/
data/flickr/captions.txt
mlruns/
.env

(ml-devops-env) soumik@soumik-VirtualBox:~/SPE_Final_Project$
```

Personal Access Tokens were created in Github for pushing commits from the local repository to the remote Github repository.



3. MLFlow - Experiment Tracking and Hyperparameter Versioning

When experimenting with different model architectures and different hyperparameters, it helps to track every experiment, the architecture & the different hyperparameters used for training, the metrics evaluated etc. so that the same model and results can be reproduced consistently and different metrics of different experiments can be easily compared.

MLFlow provides an easy way to do this, saving all these to a central server and provides a UI to view these experiments and corresponding saved parameters & metrics later. MLFlow also provides a registry to save and version models, but we opted to do this manually with checkpointing.

To efficiently track training experiments, the training loop was encapsulated within a code block that initiated an MLflow run each time a model was trained. During this process, key parameters such as batch size, optimizer, embedding dimension, LSTM units, dropout rate, and the number of training epochs were logged. At the end of each epoch, both the training loss and validation loss were recorded. Once the training was completed, the training loss versus validation loss curve was saved as an MLflow artifact, providing a visual representation of the model's performance throughout the training process.

```
# MLflow logging
with mlflow.start_run():
    # Log hyperparameters
    mlflow.log_param("batch_size", 64)
    mlflow.log_param("optimizer", "adam")
    mlflow.log_param("embedding_dim", 256)
    mlflow.log_param("lstm_units", 256)
    mlflow.log_param("dropout", 0.5)
    mlflow.log_param("epochs", 10)

    # Train and capture history
    history = caption_model.fit(
        train_gen,
        validation_data=val_gen,
        epochs=10,
        callbacks=[checkpoint, earlystop, reduce_lr],
        verbose=1
    )

    # Log metrics per epoch
    for epoch in range(len(history.history['loss'])):
        mlflow.log_metric("loss", history.history['loss'][epoch], step=epoch)
        mlflow.log_metric("val_loss", history.history['val_loss'][epoch], step=epoch)
```

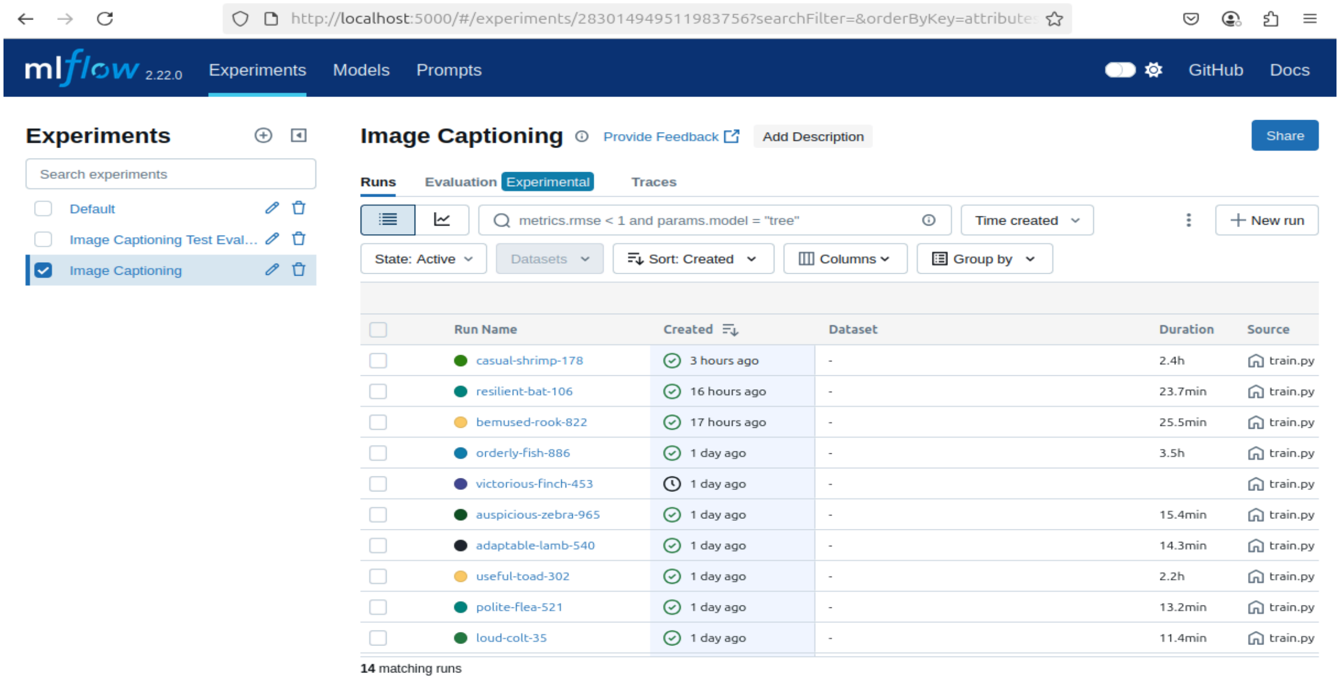
```
# Log metrics per epoch
for epoch in range(len(history.history['loss'])):
    mlflow.log_metric("loss", history.history['loss'][epoch], step=epoch)
    mlflow.log_metric("val_loss", history.history['val_loss'][epoch], step=epoch)

# Save training curve
plt.figure()
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training Curve')
plot_path = os.path.join(model_dir, 'training_curve.png')
plt.savefig(plot_path)
mlflow.log_artifact(plot_path)
```

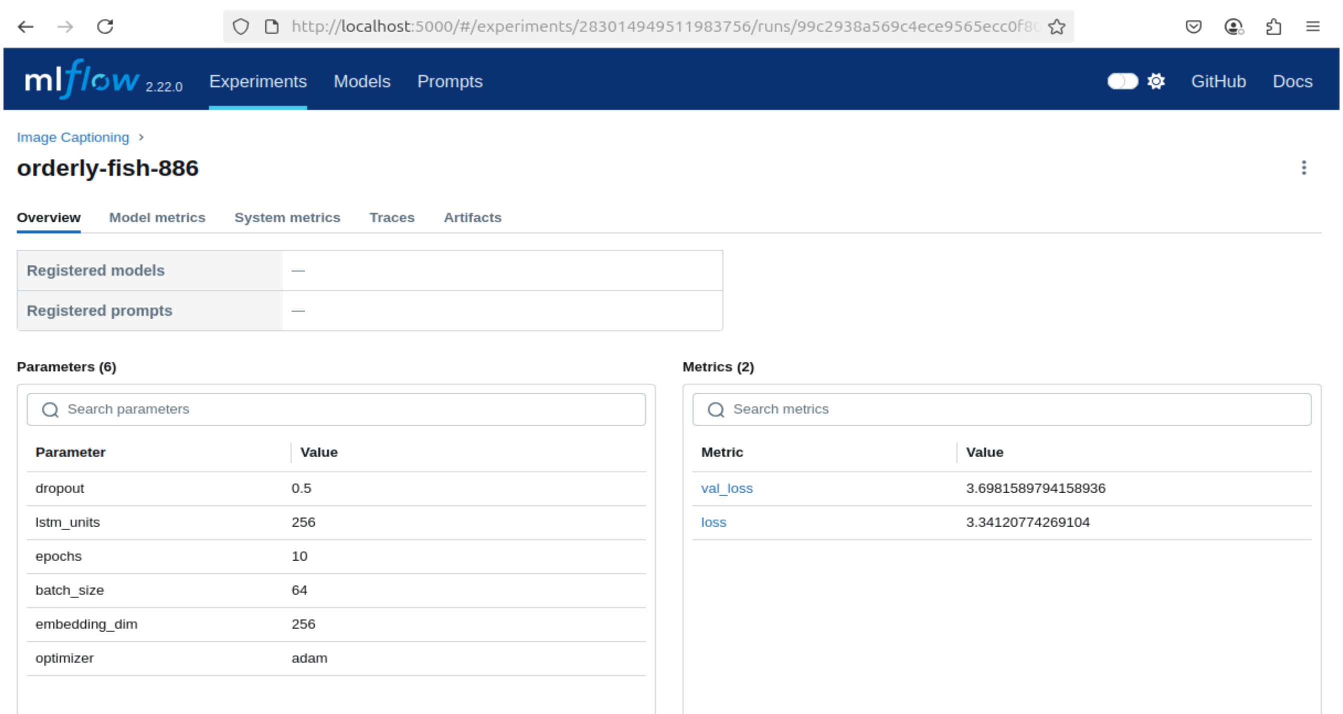
Similarly, in the test script, the BLEU score evaluated on the trained model loaded in the script is logged along with the length of the test dataset & the complete set of predictions made on the test set.

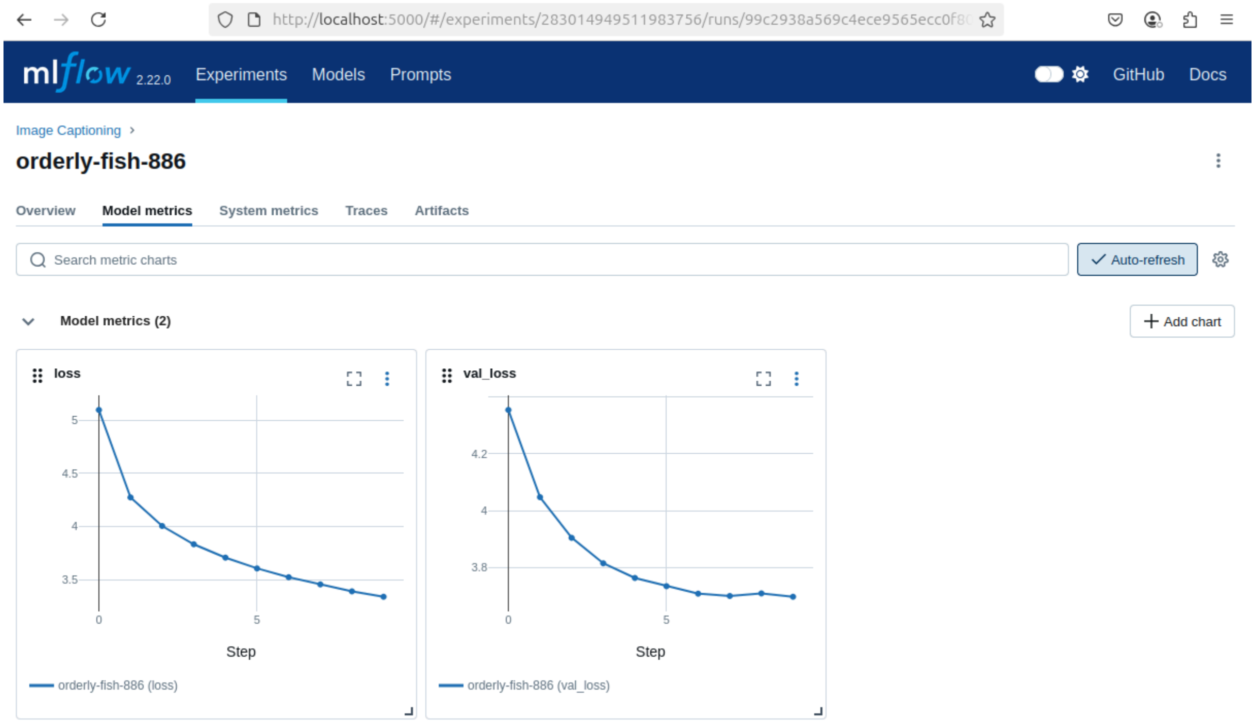
To view these saved metrics and parameters, we activate the UI with the command : mlflow ui

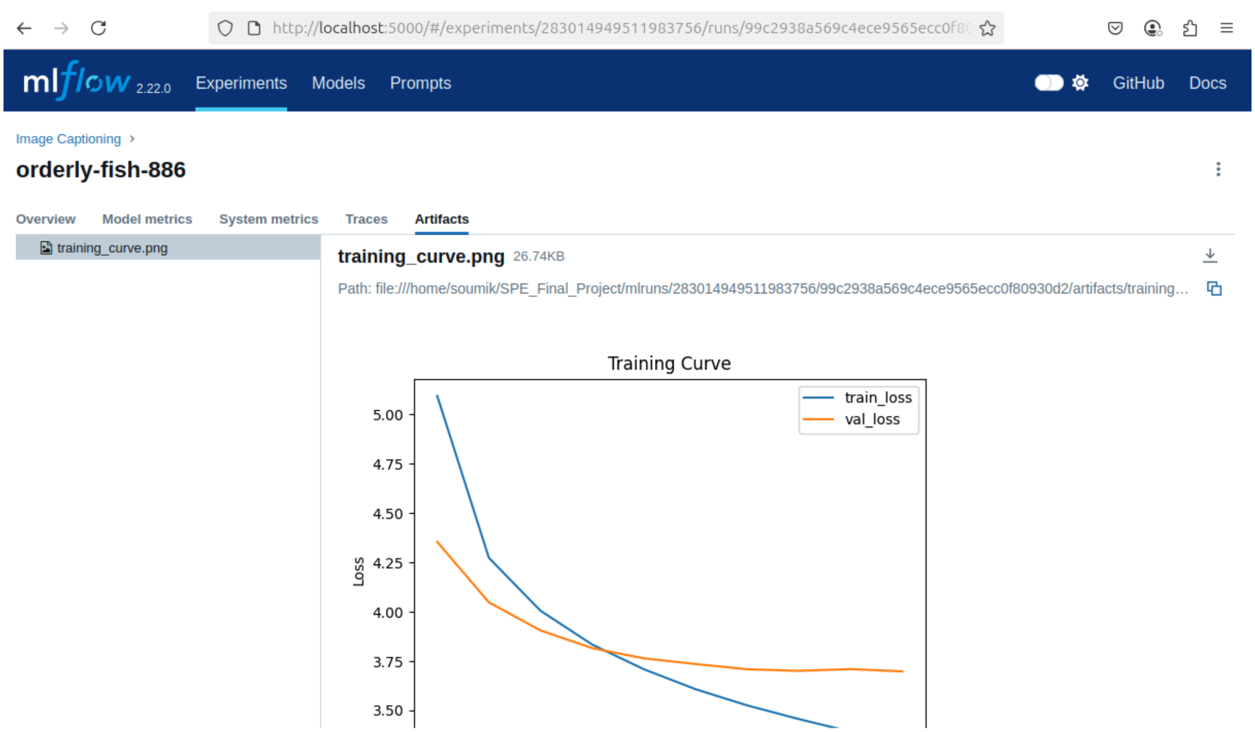
```
(ml-devops-env) soumik@soumik-VirtualBox:~/SPE_Final_Project$ mlflow ui
[2025-05-21 16:36:43 +0530] [2847600] [INFO] Starting gunicorn 23.0.0
[2025-05-21 16:36:43 +0530] [2847600] [INFO] Listening at: http://127.0.0.1:5000 (2847600)
[2025-05-21 16:36:43 +0530] [2847600] [INFO] Using worker: sync
[2025-05-21 16:36:43 +0530] [2847625] [INFO] Booting worker with pid: 2847625
[2025-05-21 16:36:43 +0530] [2847626] [INFO] Booting worker with pid: 2847626
[2025-05-21 16:36:44 +0530] [2847627] [INFO] Booting worker with pid: 2847627
[2025-05-21 16:36:44 +0530] [2847642] [INFO] Booting worker with pid: 2847642
```

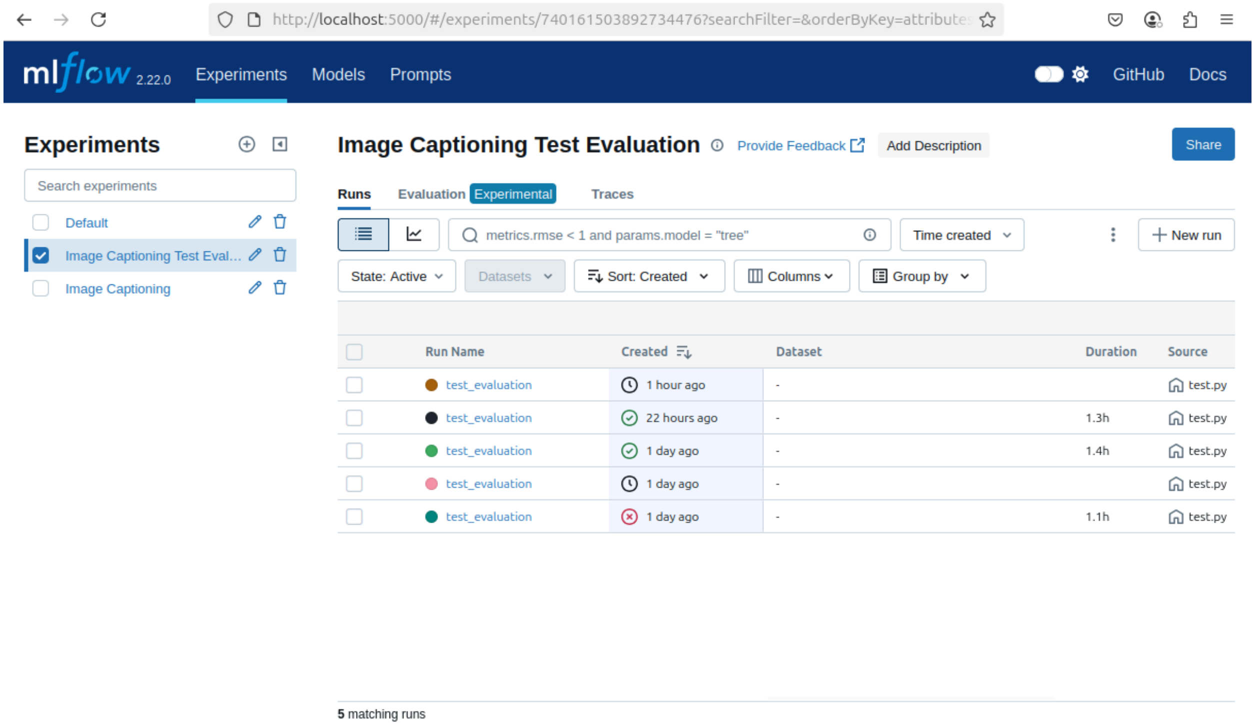
This boots up the UI on port 5000 on localhost. We can open this in a browser to view the different saved runs and their corresponding parameters and metrics.

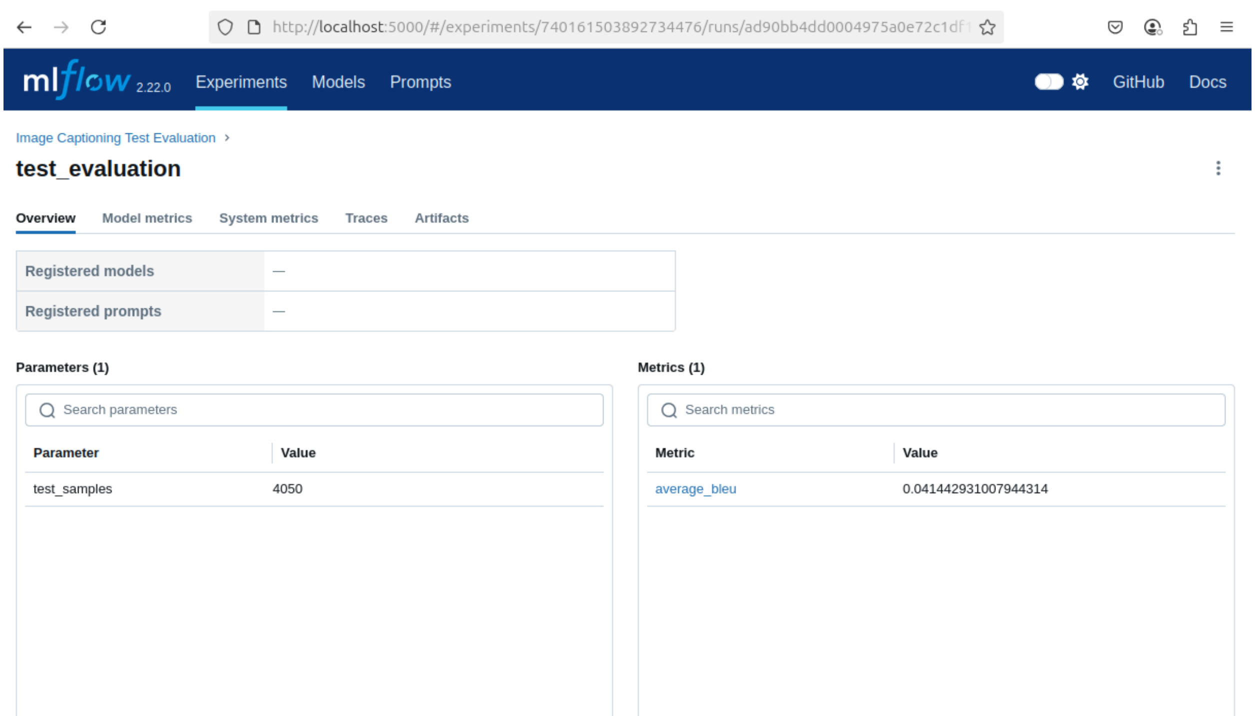

http://localhost:5000/#/experiments/283014949511983756?searchFilter=&orderByKey=attribute
mlflow 2.22.0
Experiments
Models
Prompts
GitHub
Docs
Experiments
Search experiments
Default
Image Captioning Test Eval…
Image Captioning
Image Captioning
Provide Feedback
Add Description
Share
Runs
Evaluation
Experimental
Traces
metrics.rmse < 1 and params.model = "tree"
Time created
New run
State: Active
Datasets
Sort: Created
Columns
Group by
Run Name
Created
Dataset
Duration
Source
casual-shrimp-178 3 hours ago - 2.4h train.py
resilient-bat-106 16 hours ago - 23.7min train.py
bemused-rook-822 17 hours ago - 25.5min train.py
orderly-fish-886 1 day ago - 3.5h train.py
victorious-finch-453 1 day ago - train.py
auspicious-zebra-965 1 day ago - 15.4min train.py
adaptable-lamb-540 1 day ago - 14.3min train.py
useful-toad-302 1 day ago - 2.2h train.py
polite-flea-521 1 day ago - 13.2min train.py
loud-colt-35 1 day ago - 11.4min train.py
14 matching runs

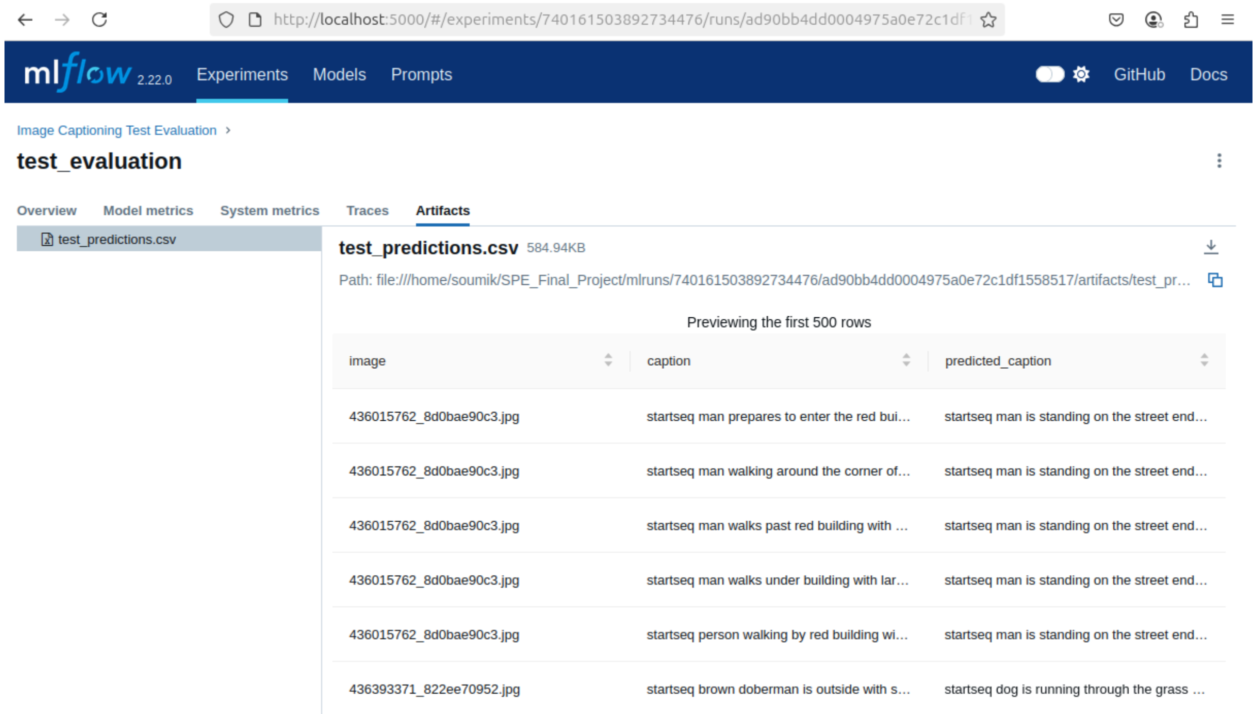

http://localhost:5000/#/experiments/283014949511983756/runs/99c2938a569c4ece9565ecc0f8
mlflow 2.22.0
Experiments
Models
Prompts
GitHub
Docs
Image Captioning >
orderly-fish-886
Overview
Model metrics
System metrics
Traces
Artifacts
Registered models —
Registered prompts —
Parameters (6)
Search parameters
Parameter Value
dropout 0.5
lstm_units 256
epochs 10
batch_size 64
embedding_dim 256
optimizer adam
Metrics (2)
Search metrics
Metric Value
val_loss 3.6981589794158936
loss 3.34120774269104


http://localhost:5000/#/experiments/283014949511983756/runs/99c2938a569c4ece9565ecc0f8
mlflow 2.22.0
Experiments
Models
Prompts
GitHub
Docs
Image Captioning ›
orderly-fish-886
Overview
Model metrics
System metrics
Traces
Artifacts
Search metric charts
Auto-refresh
Model metrics (2)
Add chart
loss
5
4.5
4
3.5
0
5
Step
orderly-fish-886 (loss)
val_loss
4.2
4
3.8
0
5
Step
orderly-fish-886 (val_loss)


http://localhost:5000/#/experiments/283014949511983756/runs/99c2938a569c4ece9565ecc0f8
mlflow 2.22.0
Experiments
Models
Prompts
GitHub
Docs
Image Captioning ›
orderly-fish-886
Overview
Model metrics
System metrics
Traces
Artifacts
training_curve.png
training_curve.png 26.74KB
Path: file:///home/soumik/SPE_Final_Project/mlruns/283014949511983756/99c2938a569c4ece9565ecc0f80930d2/artifacts/training...
Training Curve
train_loss
val_loss
Loss
5.00
4.75
4.50
4.25
4.00
3.75
3.50


http://localhost:5000/#/experiments/740161503892734476?searchFilter=&orderByKey=attribute
mlflow 2.22.0
Experiments
Models
Prompts
GitHub
Docs
Experiments
Search experiments
Default
Image Captioning Test Eval...
Image Captioning
Image Captioning Test Evaluation
Provide Feedback
Add Description
Share
Runs
Evaluation
Experimental
Traces
metrics.rmse < 1 and params.model = "tree"
Time created
New run
State: Active
Datasets
Sort: Created
Columns
Group by
Run Name
Created
Dataset
Duration
Source
test_evaluation
1 hour ago
-
test.py
test_evaluation
22 hours ago
-
1.3h
test.py
test_evaluation
1 day ago
-
1.4h
test.py
test_evaluation
1 day ago
-
test.py
test_evaluation
1 day ago
-
1.1h
test.py
5 matching runs


http://localhost:5000/#/experiments/740161503892734476/runs/ad90bb4dd0004975a0e72c1df
mlflow 2.22.0
Experiments
Models
Prompts
GitHub
Docs
Image Captioning Test Evaluation
test_evaluation
Overview
Model metrics
System metrics
Traces
Artifacts
Registered models
—
Registered prompts
—
Parameters (1)
Search parameters
Parameter
Value
test_samples
4050
Metrics (1)
Search metrics
Metric
Value
average_bleu
0.041442931007944314



4. Jenkins - Automation CI/CD server

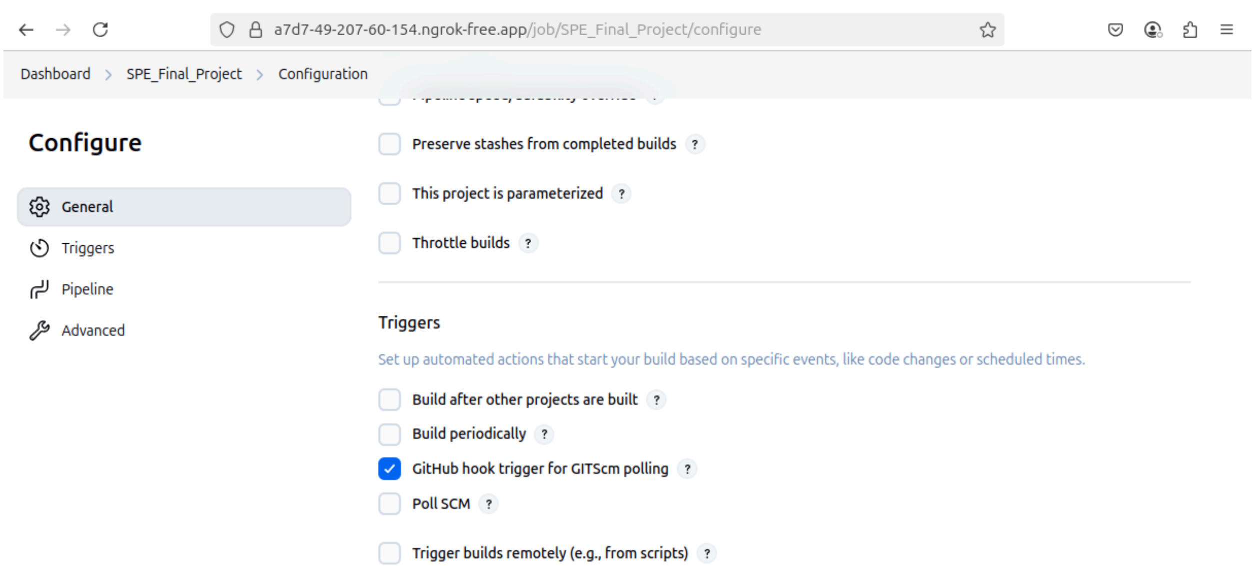
Jenkins is an open-source automation server widely used in DevOps for continuous integration and continuous delivery (CI/CD). It facilitates the automation of software development processes by allowing developers to automatically build, test, and deploy code changes.

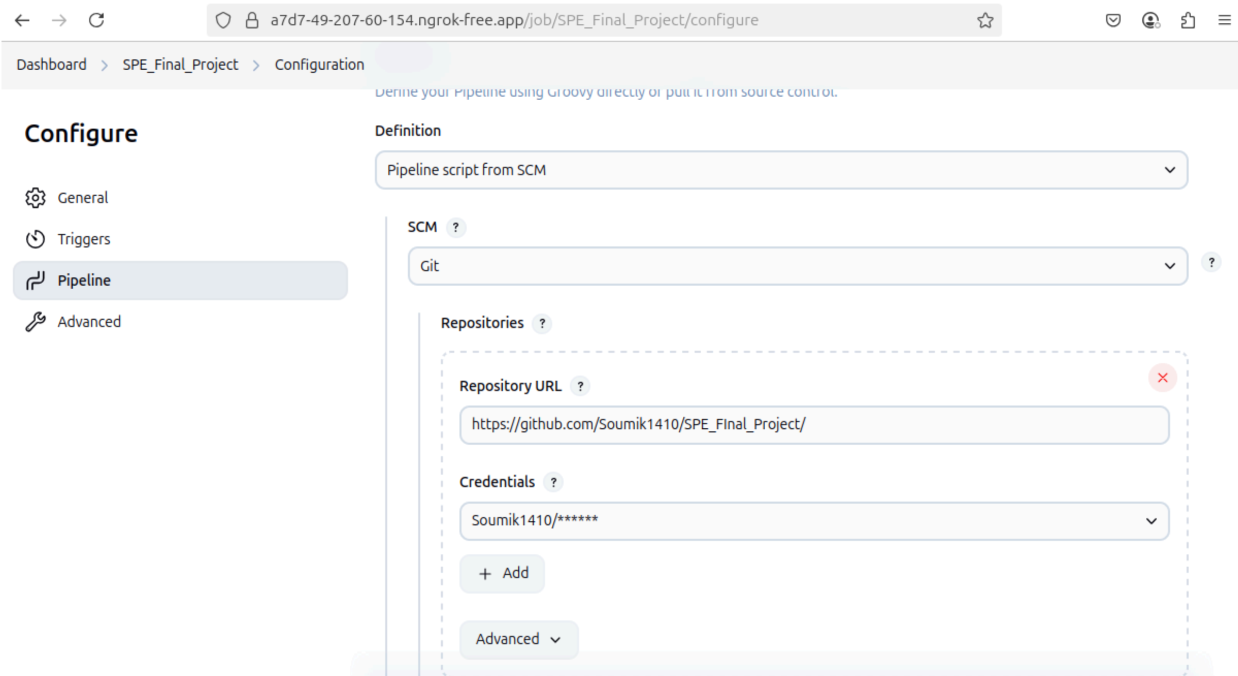
In our project, we developed a Jenkins pipeline that is triggered by webhooks sent by a SCM service (Github), and it performs the following actions automatically :

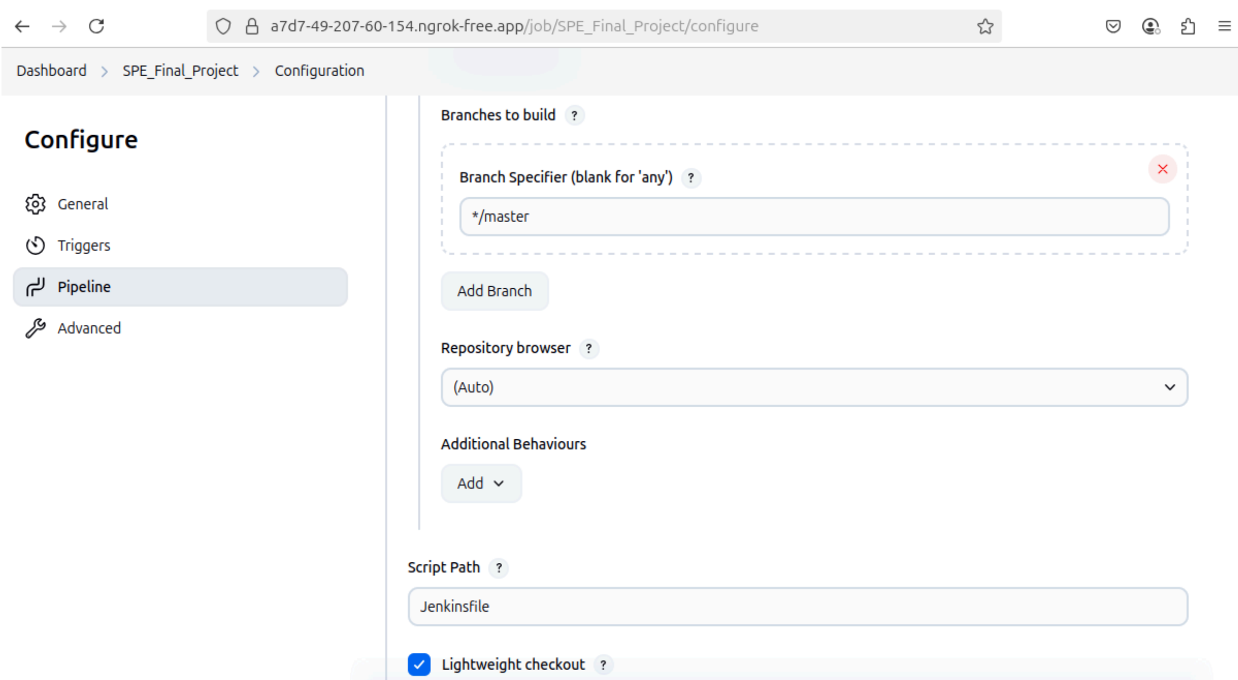
1. Checkout the latest code from the repository
2. Train the model

3. Execute the automated test script to evaluate the trained model
4. Generate a list of the environment's dependencies
5. Prune the list to include only the essential dependencies
6. Dockerize the application
7. Push the built image to Docker Hub
8. Trigger an Ansible playbook to pull the image and deploy it on the Kubernetes cluster

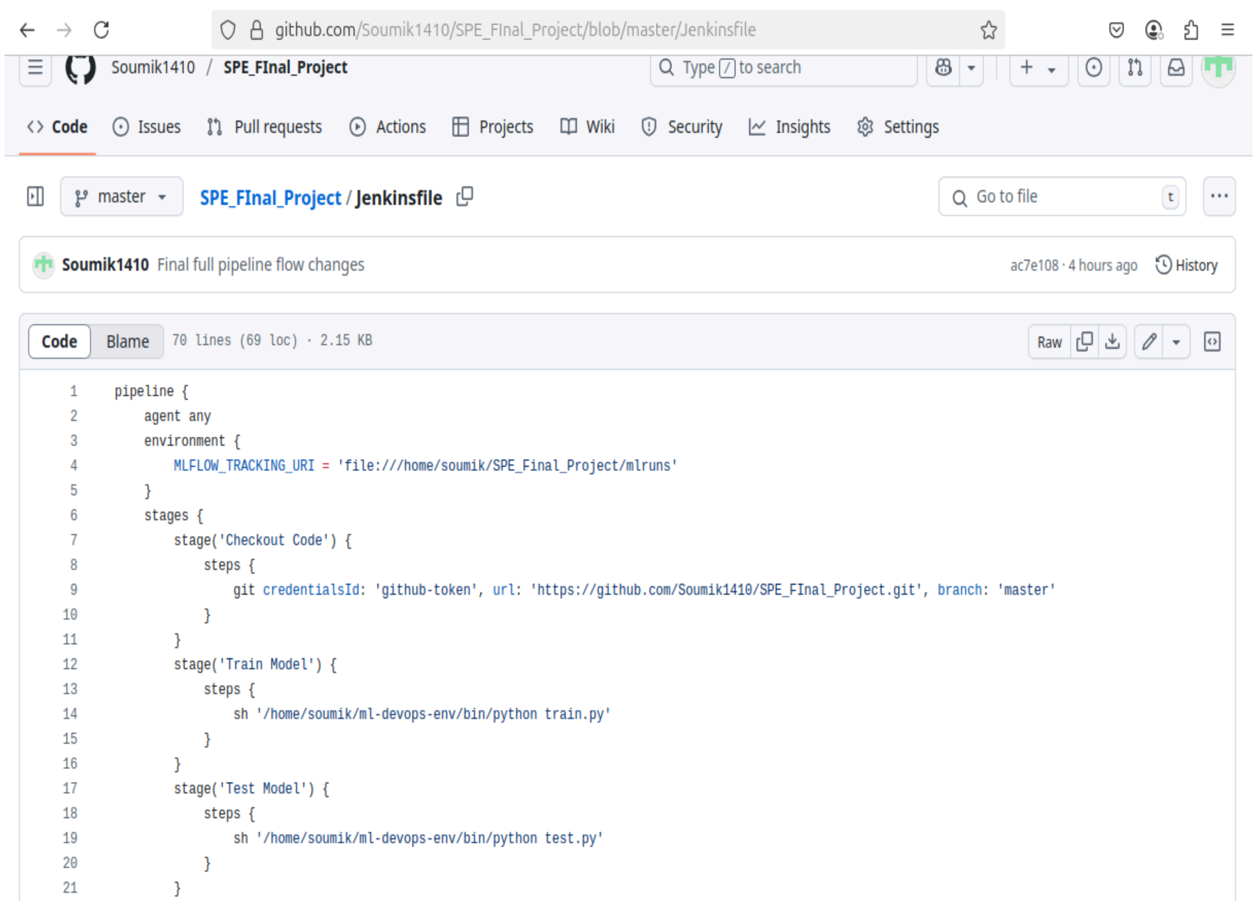
For this, we will go over the different stages of the pipeline as and when we discuss the other DevOps tools used. For now, we will show how the pipeline was set up to trigger from Github webhooks and the first 3 stages till automated testing.



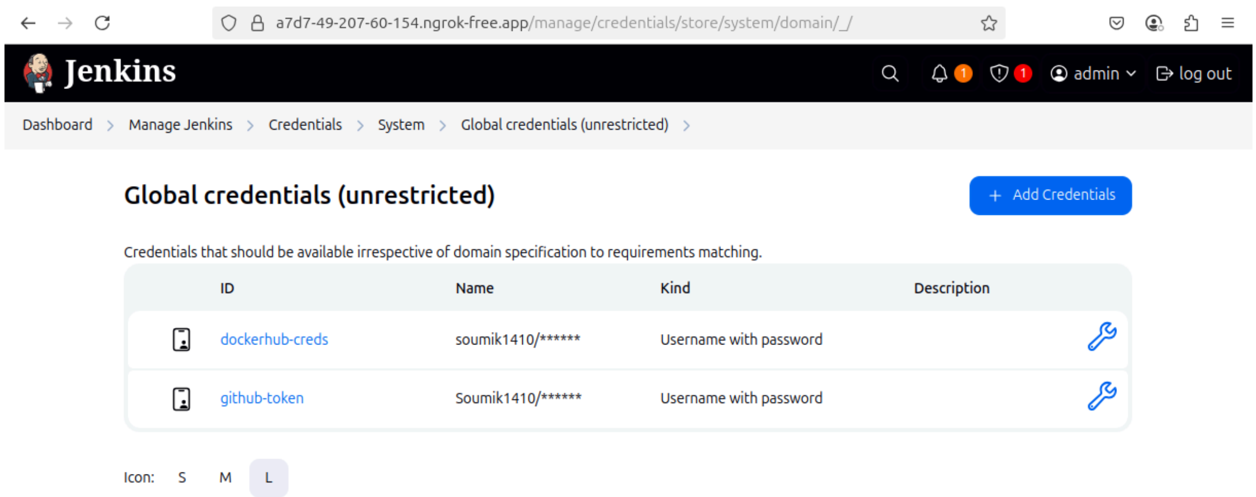




This creates a new Jenkins pipeline job that is triggered by webhooks sent from GitHub. It specifies that the source of the pipeline script resides in a Git repository, with details such as the repository URL, credentials for access, the branch where the pipeline script is located, and the filename of the script— in our case, Jenkinsfile. The Jenkinsfile itself defines the pipeline using declarative syntax, outlining each stage and the steps required within those stages to execute the tasks mentioned earlier.



For the first step i.e. checking out the latest code from a Github repository, the credentials needed were added to Jenkins credential manager. A Personal Access Token was created in Github, and that was stored in Jenkins as a username with password passing the PAT in place of the user' actual password.

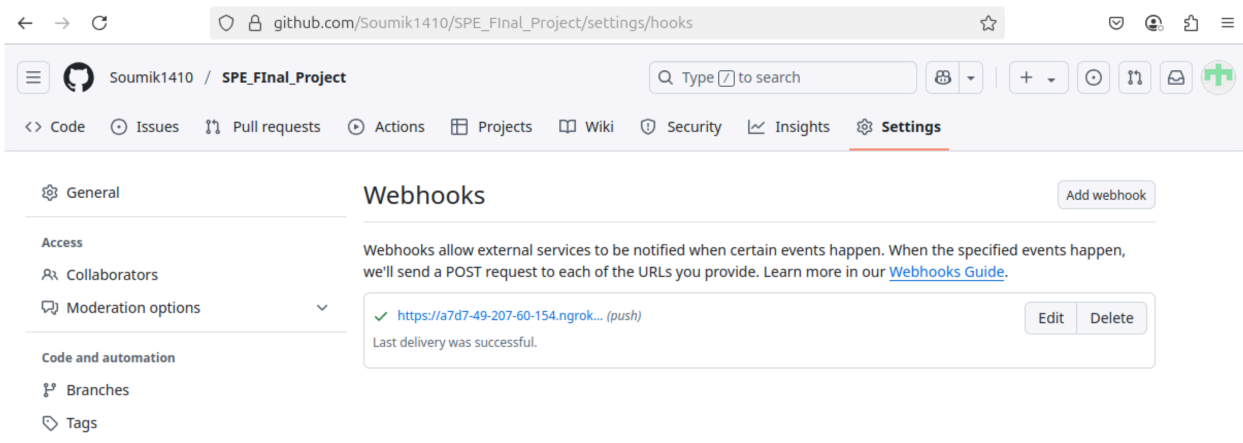


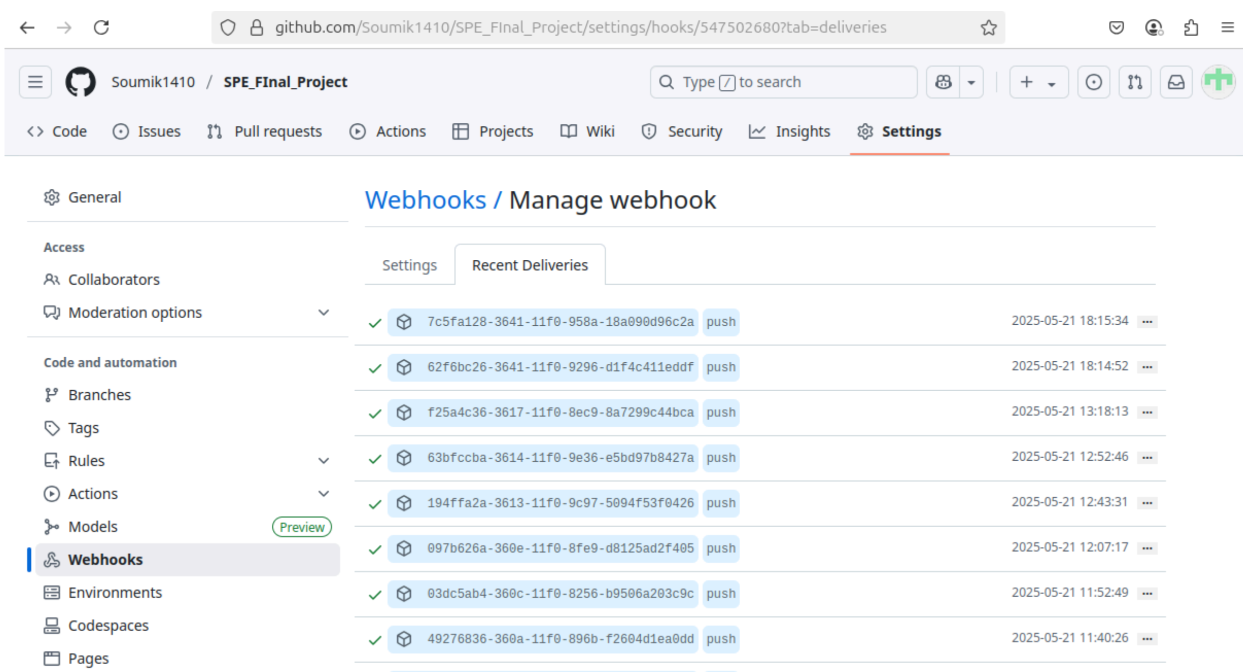
For the training and test stages, we needed all the libraries that were installed in the virtual environment used for development and as such first that environment is activated by using that specific environment's python interpreter instead of the system wise python interpreter located at /usr/bin/python.

MLFlow also normally stores all metrics and parameters and artifacts in a local folder 'mlruns' relative to the execution context of the train and test scripts. However, for these runs to show in the UI, they need to store them in a single location specified by the MLFLOW_TRACKING_URI environment variable.

Thus, the same is set in jenkins to ensure that different mlruns folders are not used to store the artifacts for different builds of the job, but instead they are all stored in the single specified location.

Now to trigger the jenkins pipeline whenever a push was made to the Github repository, a webhook was set up in Github that is delivered to jenkins everytime a push is made.



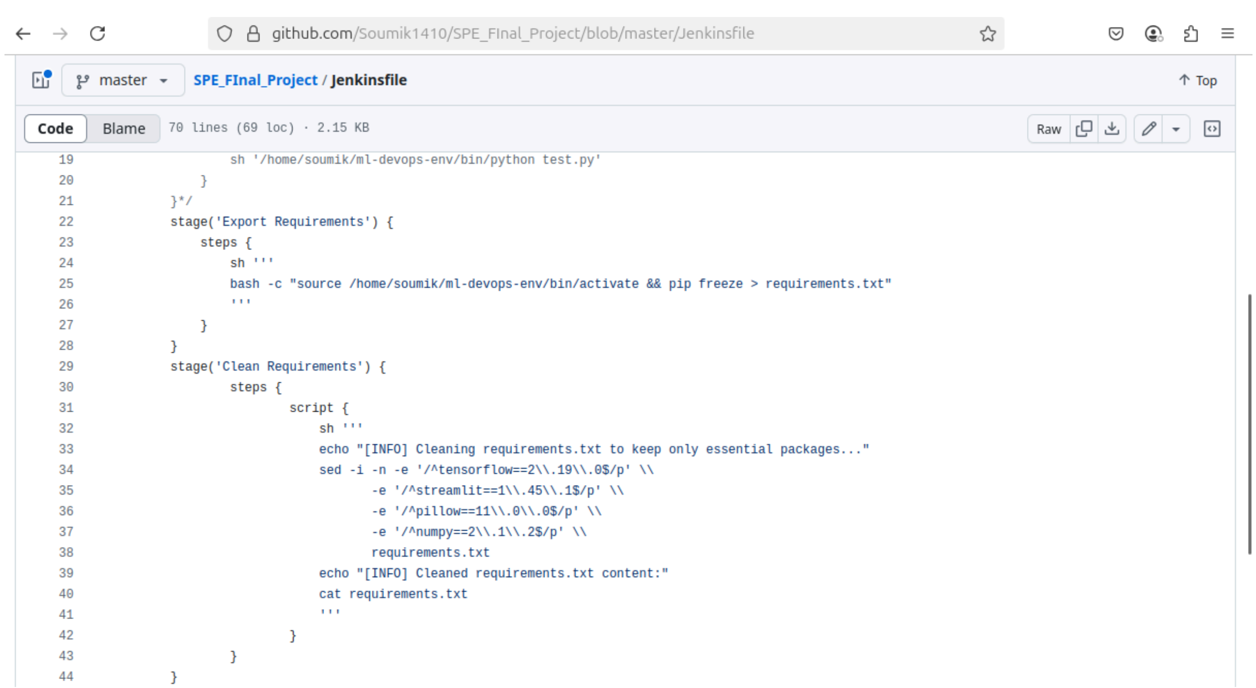


To enable the delivery of the webhook, the Jenkins instance running on my localhost needed to be exposed to the internet through a secure HTTP tunnel. This was achieved by using ngrok, which exposes port 8080 and generates a public URL. This https://<ngrok URL>/github-webhook, serves as the endpoint where the webhooks are delivered.

<u>4. Dockerizing the Application :</u>

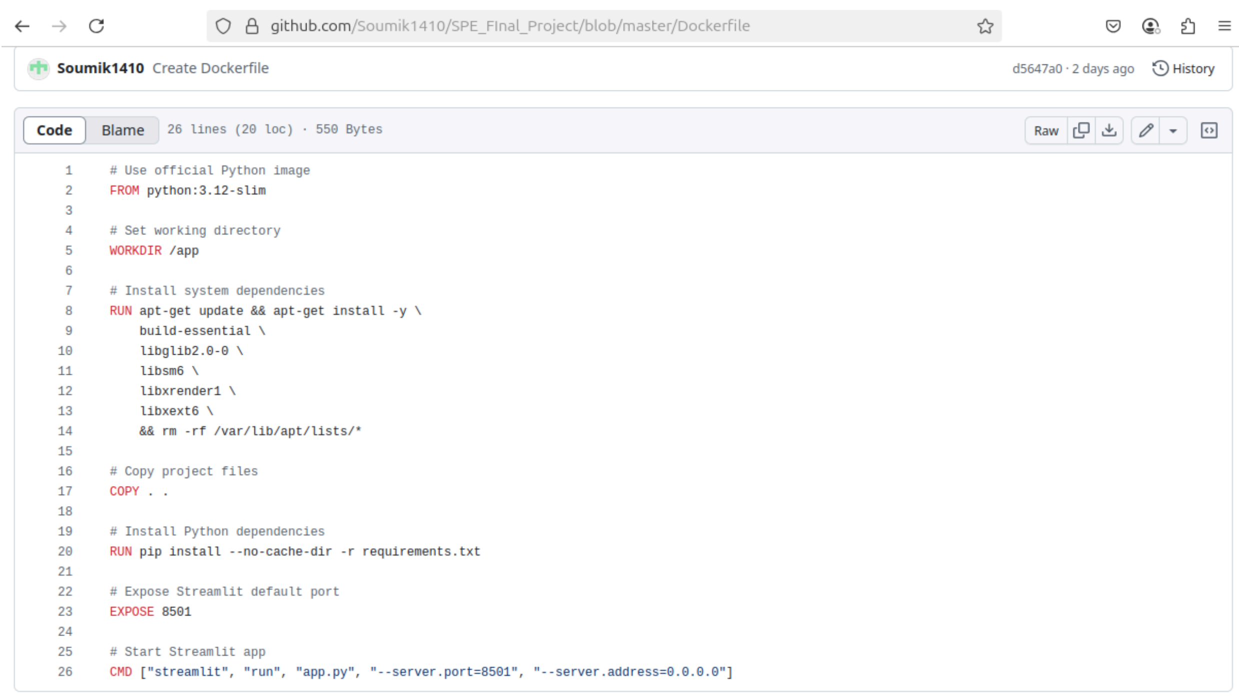
The next step in the project is to build an image that includes my inference script and the Streamlit-based frontend (defined in `app.py`). This image will be built on top of a base Python image, incorporating all necessary dependencies, the trained model, and any other saved artifacts required for deployment.

The next stage added to the Jenkins pipeline focuses on generating a requirements.txt file that lists all the environment's dependencies. However, since the environment is quite bloated due to the presence of training and testing scripts in the same repository, we need to prune the dependencies to include only the essential ones required to run app.py and infer.py. To achieve this, we added an additional stage in Jenkins to filter out all unnecessary dependencies from requirements.txt, leaving only Streamlit, TensorFlow, NumPy, Pandas, and their transitive dependencies. This helps reduce the overall size of the image.

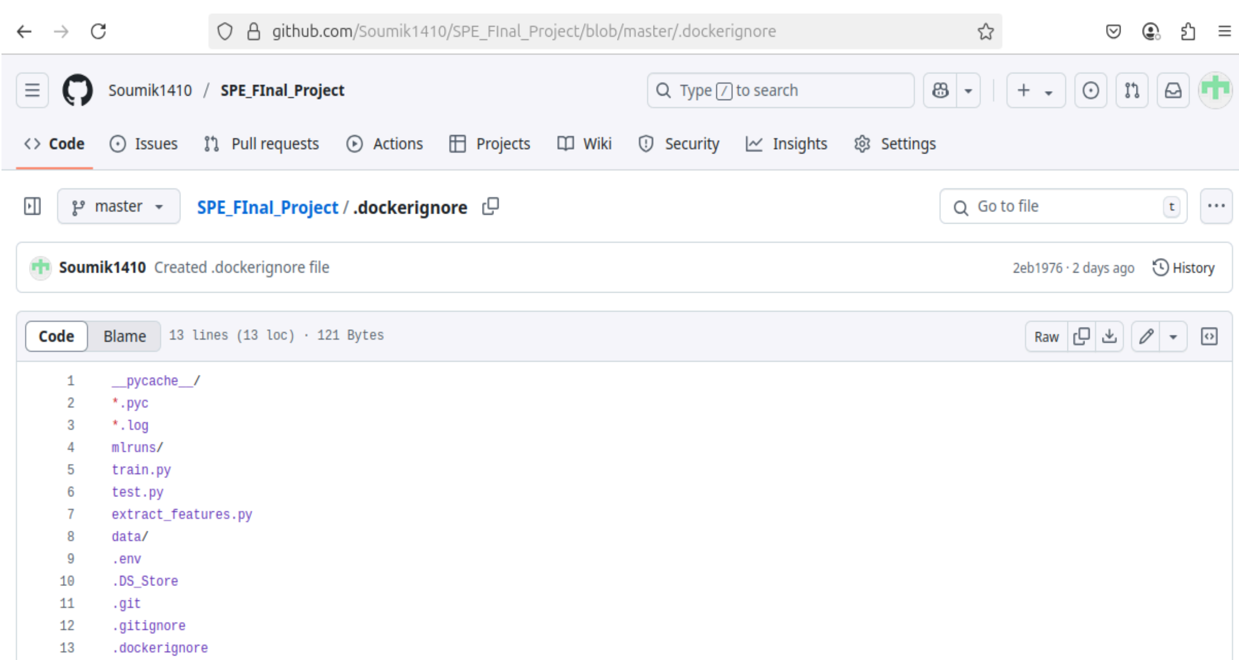



```
                sh '/home/soumik/ml-devops-env/bin/python test.py'
            }
        }*/
        stage('Export Requirements') {
            steps {
                sh '''
                bash -c "source /home/soumik/ml-devops-env/bin/activate && pip freeze > requirements.txt"
                '''
            }
        }
        stage('Clean Requirements') {
                steps {
                        script {
                            sh '''
                            echo "[INFO] Cleaning requirements.txt to keep only essential packages..."
                            sed -i -n -e '/^tensorflow==2\\.19\\.0$/p' \\
                                   -e '/^streamlit==1\\.45\\.1$/p' \\
                                   -e '/^pillow==11\\.0\\.0$/p' \\
                                   -e '/^numpy==2\\.1\\.2$/p' \\
                                   requirements.txt
                            echo "[INFO] Cleaned requirements.txt content:"
                            cat requirements.txt
                            '''
                        }
                }
        }
```

Then finally to create our docker image, we design our Dockerfile and .dockerignore files that Docker will use to build the image and also a new stage added to the Jenkins pipeline to trigger the docker build command.




```dockerfile
# Use official Python image
FROM python:3.12-slim

# Set working directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    libglib2.0-0 \
    libsm6 \
    libxrender1 \
    libxext6 \
    && rm -rf /var/lib/apt/lists/*

# Copy project files
COPY . .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose Streamlit default port
EXPOSE 8501

# Start Streamlit app
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```




```
__pycache__/
*.pyc
*.log
mlruns/
train.py
test.py
extract_features.py
data/
.env
.DS_Store
.git
.gitignore
.dockerignore
```

github.com/Soumik1410/SPE_FInal_Project/blob/master/Jenkinsfile

master · SPE_FInal_Project / Jenkinsfile · Top

Code · Blame · 70 lines (69 loc) · 2.15 KB · Raw

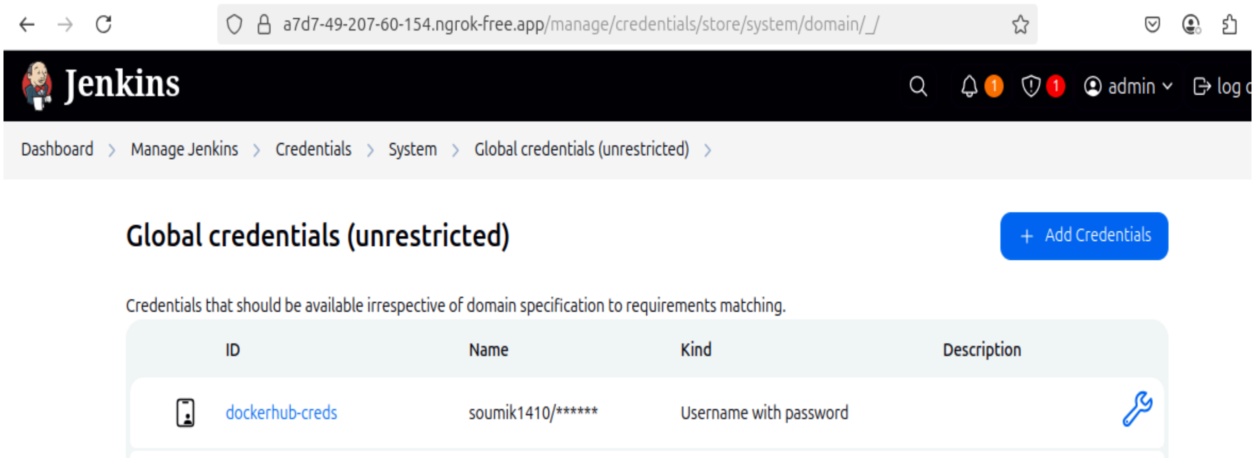
```
                    cat requirements.txt
                    '''
                }
            }
        }
        stage("Build Docker Image") {
                steps {
            script {
                sh 'docker build -t imagecaptioner_mt2024153:latest .'
                echo 'Docker Image successfully created.'
            }
        }
        }
```

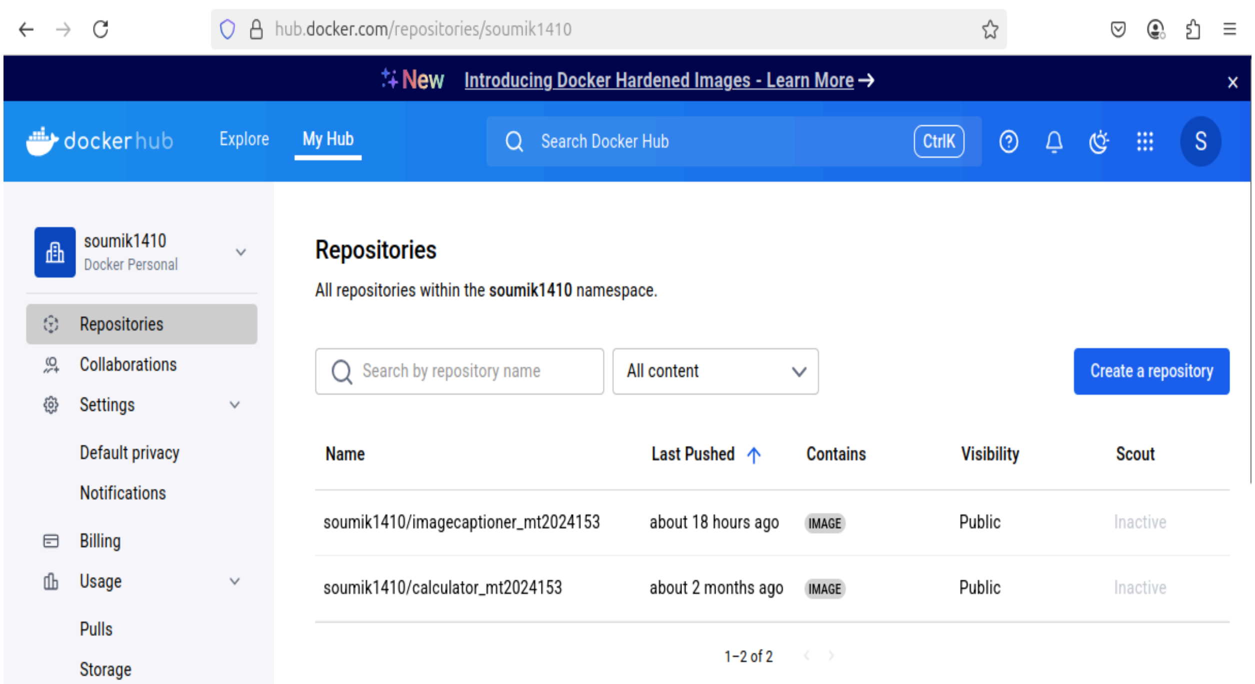
Docker will build the image according to the instructions given in the Dockerfile i.e. using python:3.12-slim as the base image, installing system dependencies, copying everything from jenkins workspace current directory to docker build (for the saved model and artifacts), then installing required python dependencies, exposing streamlit's default port, then finally running the streamlit point as entry command.

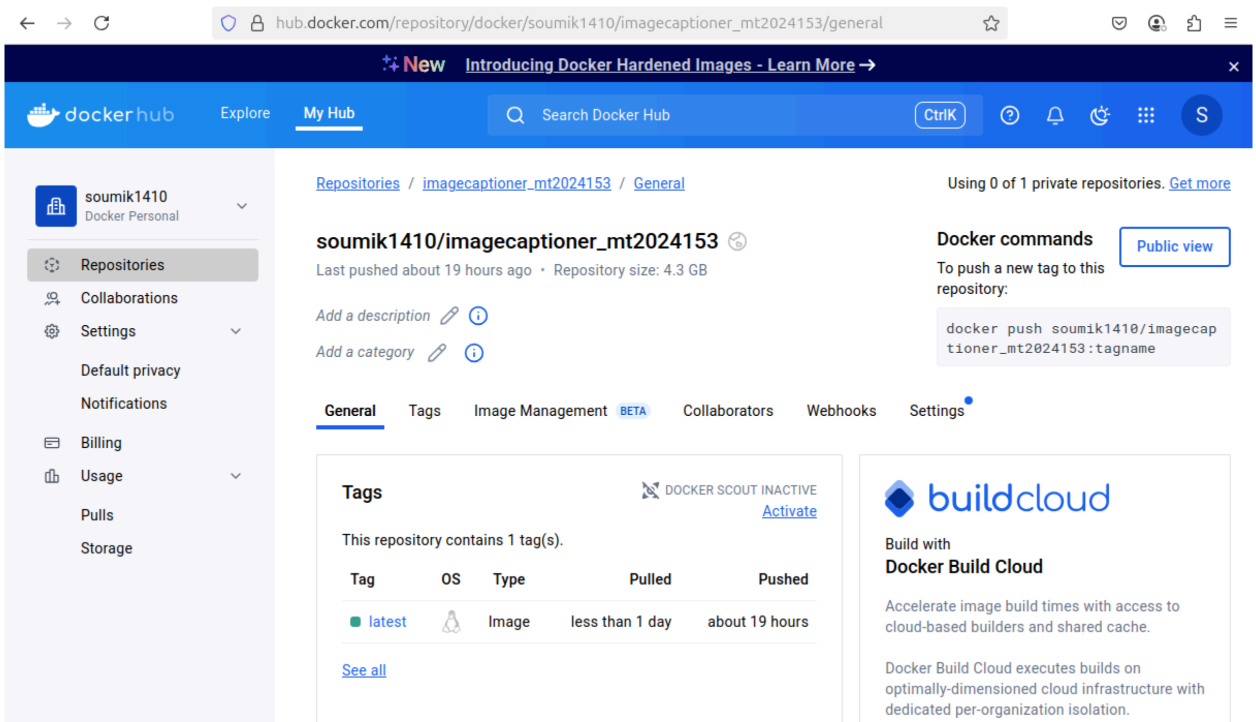
.dockerfile lists the files and folders to not be copied when executing the COPY . . command in Dockerfile

Once the build is complete, it is pushed to the Docker Hub repository. To facilitate this, a new stage is added to the Jenkins pipeline, and Docker Hub credentials are securely stored in Jenkins' credential manager for seamless authentication during the push process.

```
        stage('Pushing Docker Image to Hub') {
        steps {
            script {
                withDockerRegistry([credentialsId: 'dockerhub-creds', url: 'https://index.docker.io/v1/']) {
                        sh 'docker tag imagecaptioner_mt2024153:latest soumik1410/imagecaptioner_mt2024153:latest'
                sh 'docker push soumik1410/imagecaptioner_mt2024153:latest'
                }
                }
        }
        }
```


a7d7-49-207-60-154.ngrok-free.app/manage/credentials/store/system/domain/_/
Jenkins
admin
log o
Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >
Global credentials (unrestricted)
+ Add Credentials
Credentials that should be available irrespective of domain specification to requirements matching.
ID
Name
Kind
Description
dockerhub-creds
soumik1410/******
Username with password


hub.docker.com/repositories/soumik1410
New Introducing Docker Hardened Images - Learn More →
docker hub
Explore
My Hub
Search Docker Hub
CtrlK
S
soumik1410
Docker Personal
Repositories
Collaborations
Settings
Default privacy
Notifications
Billing
Usage
Pulls
Storage
Repositories
All repositories within the soumik1410 namespace.
Search by repository name
All content
Create a repository
Name
Last Pushed
Contains
Visibility
Scout
soumik1410/imagecaptioner_mt2024153
about 18 hours ago
IMAGE
Public
Inactive
soumik1410/calculator_mt2024153
about 2 months ago
IMAGE
Public
Inactive
1–2 of 2



5.Ansible - Deploying the application to a Kubernetes cluster (with Ansible Roles for modularity) :

Once the build has been successfully pushed to Docker Hub, the next step is to deploy it to the target environment. The deployment will be carried out on a Kubernetes cluster running locally using Minikube. The deployment tasks involve several key actions: pulling the latest build from Docker Hub, gracefully stopping the existing Minikube instance, starting up Minikube, setting the kubectl context to Minikube, creating the necessary namespace, and applying the Kubernetes manifests for both the application deployment and service.

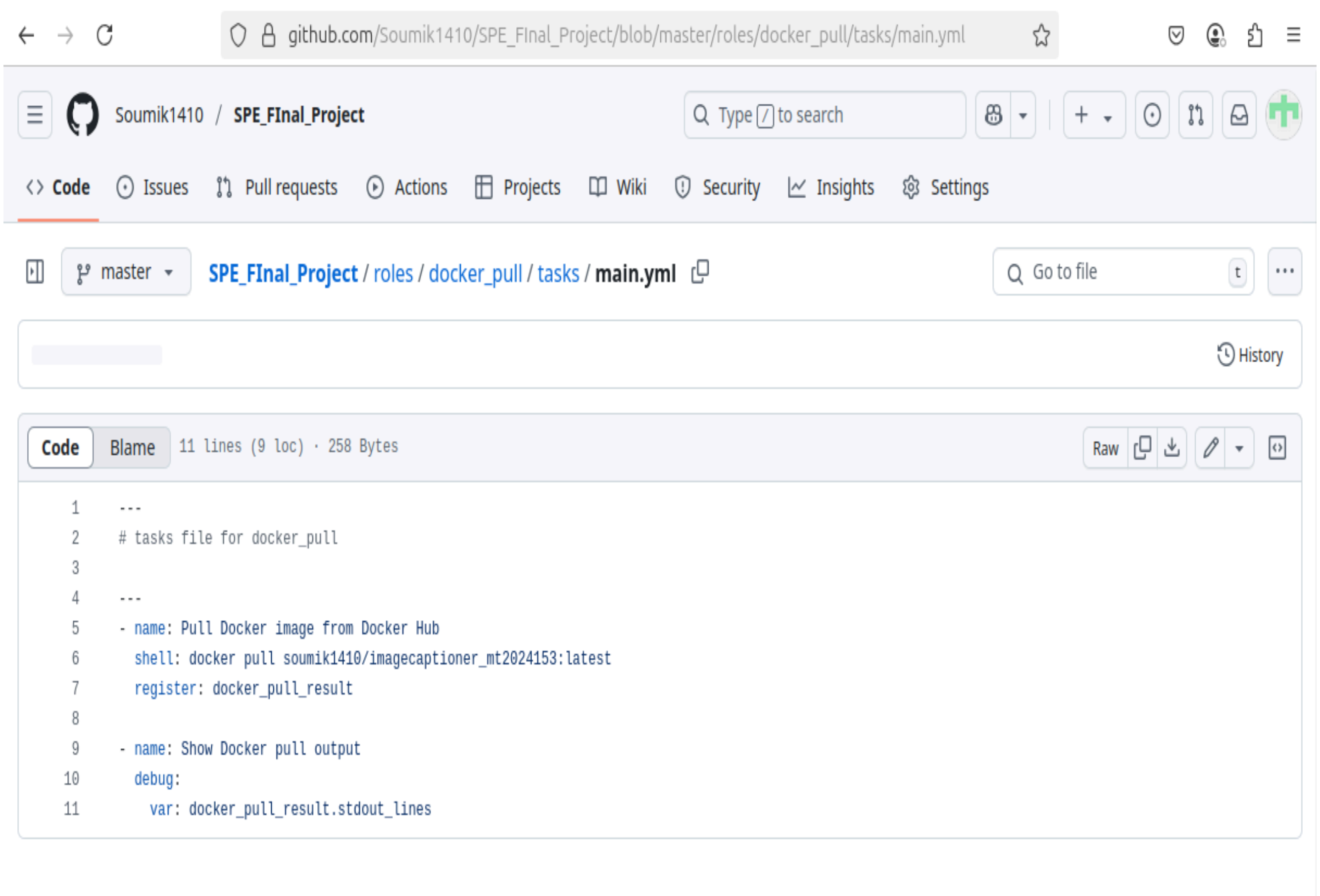
To automate these steps, we utilize an Ansible playbook that defines each task using the appropriate Ansible modules and specifies the desired states or actions. This ensures a smooth, repeatable deployment process on the local Minikube cluster.

We utilize Ansible roles to break up all the deployment tasks into 2 modules, 1 for pulling the latest docker image from Docker Hub and another for handling deployment of the application to the Kubernetes cluster.

The 2 roles created were docker_pull and k8s_deploy.

Folder structure for this included a roles folder with these 2 subdirectories and both followed a common structure with subfolders for tasks, handlers, tests etc. We have only used tasks in our project, modifying the main.yaml files for the 2 roles to perform the tasks mentioned earlier.

roles/docker_pull/tasks/main.yaml :



roles/k8s_deploy/tasks/main.yaml :

github.com/Soumik1410/SPE_FInal_Project/blob/master/roles/k8s_deploy/tasks/main.yml

Soumik1410 / **SPE_FInal_Project**

Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

master · SPE_FInal_Project / roles / k8s_deploy / tasks / **main.yml**

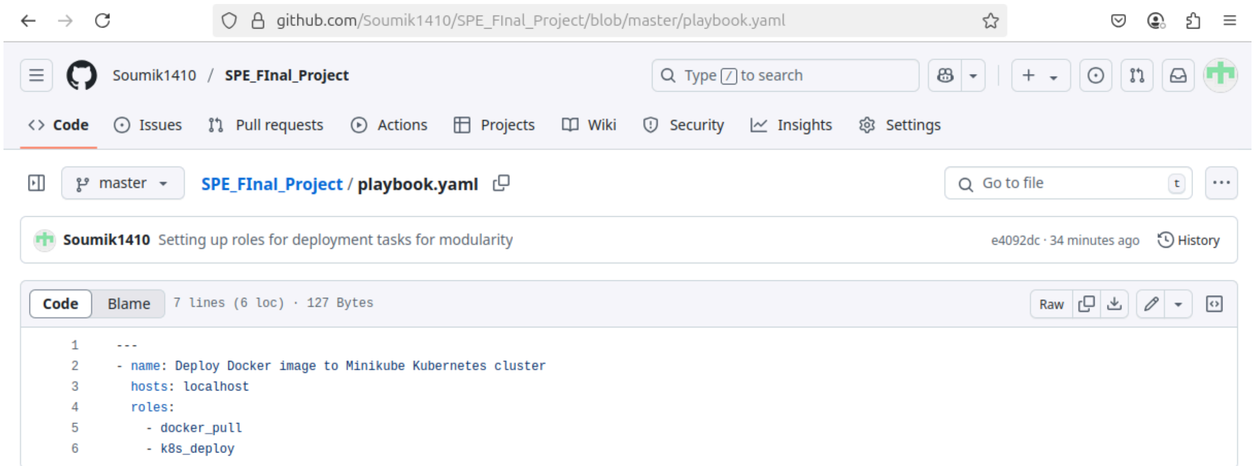
History

Code · Blame · 40 lines (32 loc) · 1.05 KB · Raw

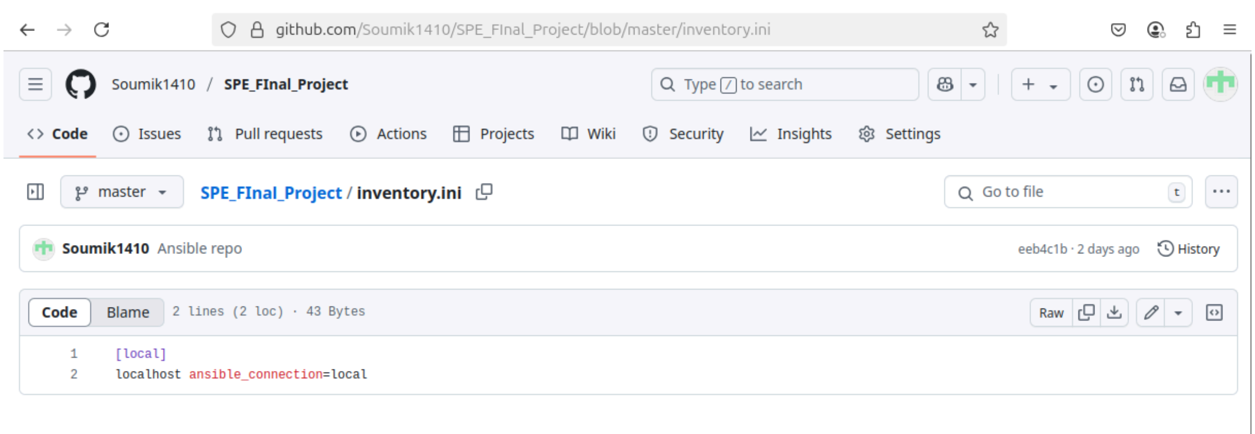
```yaml
---
# tasks file for roles/k8s_deploy

- name: Stop existing Minikube cluster if running
  shell: minikube stop
  ignore_errors: yes

- name: Delete existing Minikube cluster if exists
  shell: minikube delete
  ignore_errors: yes

- name: Start Minikube if not already running
  shell: |
    minikube status || minikube start --driver=docker
  register: minikube_status
  changed_when: "'Running' not in minikube_status.stdout"

- name: Set kubectl context to Minikube
  shell: minikube update-context
```

```yaml
- name: Create namespace if not present
  kubernetes.core.k8s:
    api_version: v1
    kind: Namespace
    name: image-caption-app-mt2024153
    state: present

- name: Apply Kubernetes deployment
  kubernetes.core.k8s:
    kubeconfig: ~/.kube/config
    state: present
    definition: "{{ lookup('file', 'deployment.yaml') }}"

- name: Apply Kubernetes service (if any)
  kubernetes.core.k8s:
    kubeconfig: ~/.kube/config
    state: present
    definition: "{{ lookup('file', 'service.yaml') }}"
  when: lookup('file', 'service.yaml', errors='ignore') is not none
```

The main playbook file simply lists these roles and a new stage is added to the Jenkins pipeline that executes this main playbook file

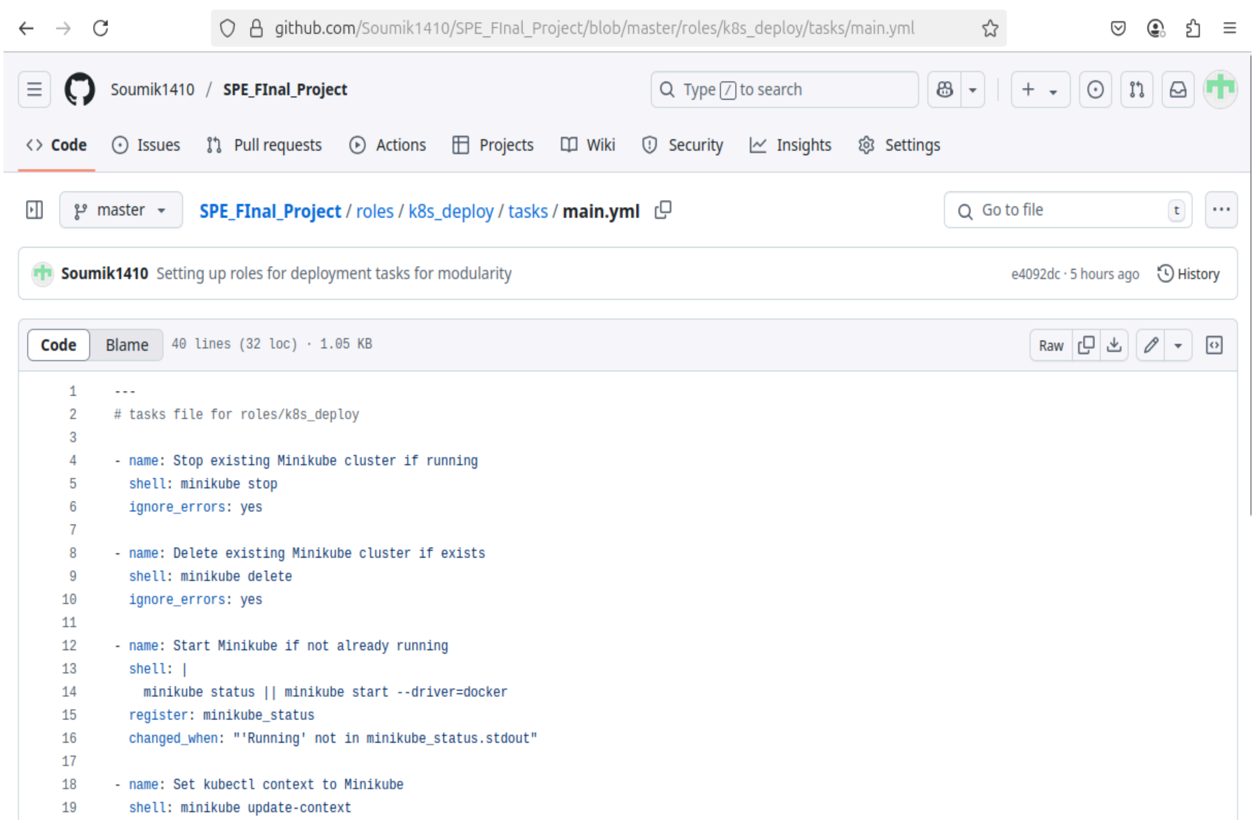


Ansible inventory file used for this deployment is simple as deployment target is localhost



Our deployment target is a kubernetes cluster deployed on my localhost node using minikube. The Ansible role k8s_deploy is used to set up the Kubernetes cluster and deploy our image captioning application.

First, it gracefully shuts down any existing minikube instance that might be running and starts up a new minikube instance, setting kubectl context to the kubernetes cluster deployed by the fresh minikube instance so that the cluster can be managed easily with kubectl commands without needing to specify cluster everytime.




```yaml
---
# tasks file for roles/k8s_deploy

- name: Stop existing Minikube cluster if running
  shell: minikube stop
  ignore_errors: yes

- name: Delete existing Minikube cluster if exists
  shell: minikube delete
  ignore_errors: yes

- name: Start Minikube if not already running
  shell: |
    minikube status || minikube start --driver=docker
  register: minikube_status
  changed_when: "'Running' not in minikube_status.stdout"

- name: Set kubectl context to Minikube
  shell: minikube update-context
```

Then, we use kubectl commands to create a custom namespace for our application called image-caption-app-mt2024153 and apply our deployment and service Kubernetes manifests which are used to specify the number of replicas, the build deployed and network access to our application.

Ansible module used for these is kubernetes.core.k8s which can be installed with ansible-galaxy collection install kubernetes.core

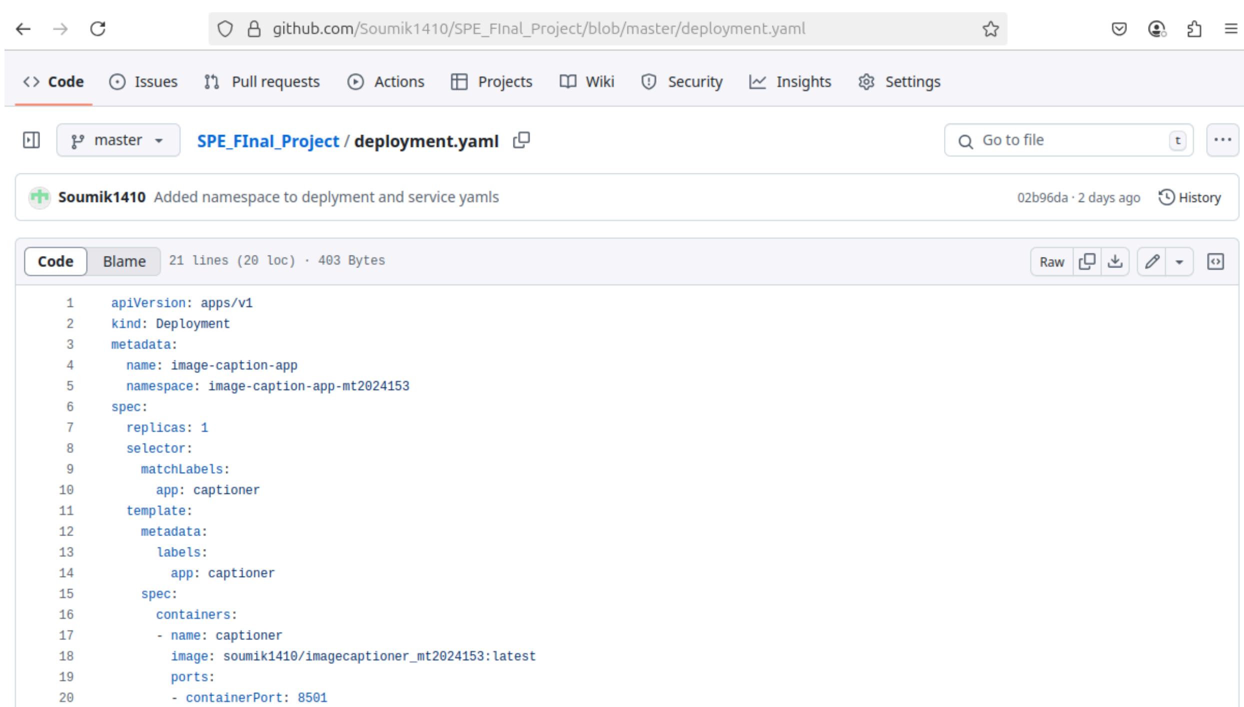
First, a separate namespace is created for our application called image-caption-app-mt2024153.

```yaml
- name: Create namespace if not present
  kubernetes.core.k8s:
    api_version: v1
    kind: Namespace
    name: image-caption-app-mt2024153
    state: present
```

Then, we apply a deployment.yaml and service.yaml in that namespace that deploys our application's docker image in a container inside a Kubernetes pod and sets up the network access to it. State:present with kubernetes.core.k8s module is equivalent to kubectl apply.

```yaml
- name: Apply Kubernetes deployment
  kubernetes.core.k8s:
    kubeconfig: ~/.kube/config
    state: present
    definition: "{{ lookup('file', 'deployment.yaml') }}"

- name: Apply Kubernetes service (if any)
  kubernetes.core.k8s:
    kubeconfig: ~/.kube/config
    state: present
    definition: "{{ lookup('file', 'service.yaml') }}"
  when: lookup('file', 'service.yaml', errors='ignore') is not none
```

deployment.yaml :

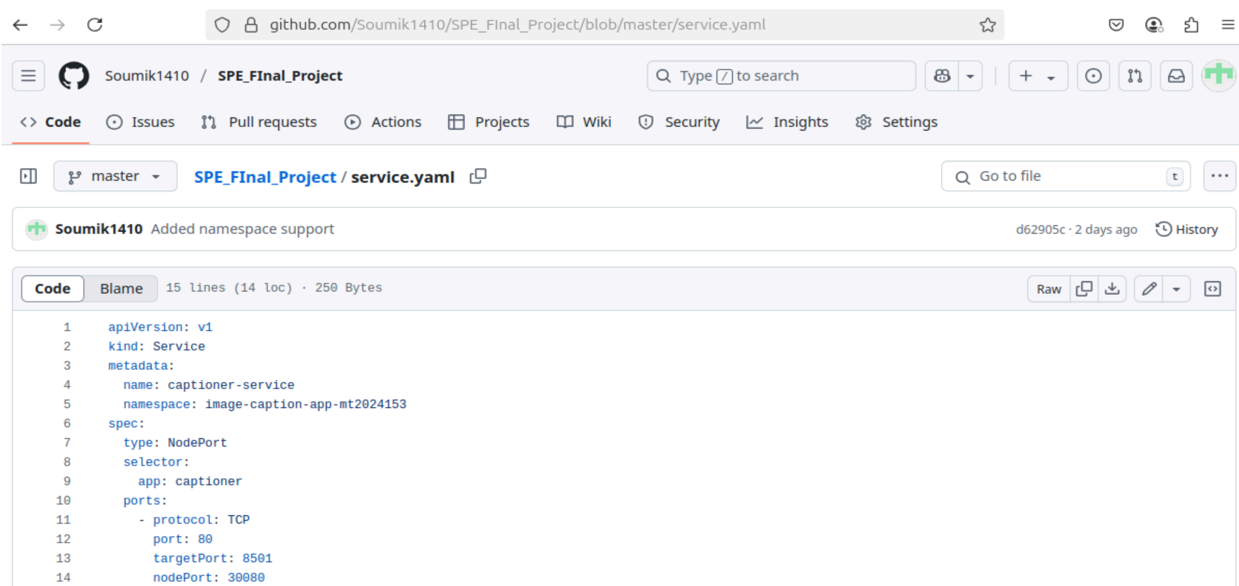



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: image-caption-app
  namespace: image-caption-app-mt2024153
spec:
  replicas: 1
  selector:
    matchLabels:
      app: captioner
  template:
    metadata:
      labels:
        app: captioner
    spec:
      containers:
      - name: captioner
        image: soumik1410/imagecaptioner_mt2024153:latest
        ports:
        - containerPort: 8501
```

This configuration defines a Kubernetes Deployment for the Image Captioning App, called image-caption-app, intended to run within the image-caption-app-mt2024153 namespace. It specifies a single replica (i.e., one Pod) for the application. Both the application and the Pod are labeled with captioner to facilitate identification, enabling users to easily determine which application is running on which Pod, as well as to manage the associated Pods within the Deployment.

The YAML further details that the Deployment will launch a single container within the Pod, which will run the Docker image for the Image Captioning App. This container will expose port 8501, the required port for accessing the Streamlit application hosted on the Pod.

service.yaml



This YAML specifies a Kubernetes Service called captioner-service, intended to run within the namespace image-caption-app-mt2024153. It is a NodePort service, which is used to allow external HTTP connections to containers deployed wiThis YAML defines a Kubernetes Service named captioner-service, which is deployed within the image-caption-app-mt2024153 namespace. The service is of type

NodePort, allowing external HTTP traffic to access containers running in the Kubernetes cluster. It uses the app: captioner label selector to identify and route traffic to the appropriate Pods.

The service maps node port 30080 to the container's port 8501 — the default port used by Streamlit. This setup allows external clients to connect to the Streamlit application running inside the Pods.

Once the deployment is complete we can verify the status using kubectl commands.

```
jenkins@soumik-VirtualBox:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

jenkins@soumik-VirtualBox:~$
```

```
jenkins@soumik-VirtualBox:~$ kubectl get ns
NAME                            STATUS   AGE
default                         Active   22m
image-caption-app-mt2024153     Active   22m
kube-node-lease                 Active   22m
kube-public                     Active   22m
kube-system                     Active   22m
jenkins@soumik-VirtualBox:~$
```

```
jenkins@soumik-VirtualBox:~$ kubectl get pods -n image-caption-app-mt2024153
NAME                                 READY   STATUS    RESTARTS   AGE
fluent-bit-c4zh6                     1/1     Running   0          17m
grafana-8559bcb694-2xpdc             1/1     Running   0          17m
image-caption-app-756c979f57-zmjnh   1/1     Running   0          18m
loki-87d578f98-c22dr                 1/1     Running   0          17m
jenkins@soumik-VirtualBox:~$
```

```
jenkins@soumik-VirtualBox:~$ kubectl describe pod image-caption-app-756c979f57-zmjnh -n image-caption-app-mt2024153
Name:             image-caption-app-756c979f57-zmjnh
Namespace:        image-caption-app-mt2024153
Priority:         0
Service Account:  default
Node:             minikube/192.168.58.2
Start Time:       Thu, 22 May 2025 15:42:58 +0530
Labels:           app=captioner
                  pod-template-hash=756c979f57
Annotations:      <none>
Status:           Running
IP:               10.244.0.5
IPs:
  IP:           10.244.0.5
Controlled By:  ReplicaSet/image-caption-app-756c979f57
Containers:
  captioner:
    Container ID:   docker://4019ff149aada3b8bed5b265fd142a395df318b46541688933f7e1dbd75f1169
    Image:          soumik1410/imagecaptioner_mt2024153:latest
    Image ID:       docker-pullable://soumik1410/imagecaptioner_mt2024153@sha256:f24895847a4e7558a26f4d598765c7b9f38460b
39b3a9b86261e72e32c6bb19b
    Port:           8501/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 22 May 2025 15:43:01 +0530
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:  500m
    Requests:
      cpu:        200m
```

```
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2j4qd (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-2j4qd:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Burstable
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  19m   default-scheduler  Successfully assigned image-caption-app-mt2024153/image-caption-app-756c97
9f57-zmjnh to minikube
  Normal  Pulling    19m   kubelet            Pulling image "soumik1410/imagecaptioner_mt2024153:latest"
  Normal  Pulled     19m   kubelet            Successfully pulled image "soumik1410/imagecaptioner_mt2024153:latest" in
2.137s (2.137s including waiting). Image size: 3039320995 bytes.
  Normal  Created    19m   kubelet            Created container: captioner
  Normal  Started    19m   kubelet            Started container captioner
jenkins@soumik-VirtualBox:~$
```

```
jenkins@soumik-VirtualBox:~$ kubectl get deployment -n image-caption-app-mt2024153
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
grafana             1/1     1            1           21m
image-caption-app   1/1     1            1           26m
loki                1/1     1            1           21m
jenkins@soumik-VirtualBox:~$ kubectl get svc -n image-caption-app-mt2024153
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
captioner-service   NodePort    10.104.28.183    <none>        80:30080/TCP     26m
grafana             NodePort    10.107.126.227   <none>        3000:30030/TCP   21m
loki                ClusterIP   10.99.136.160    <none>        3100/TCP         21m
jenkins@soumik-VirtualBox:~$
```

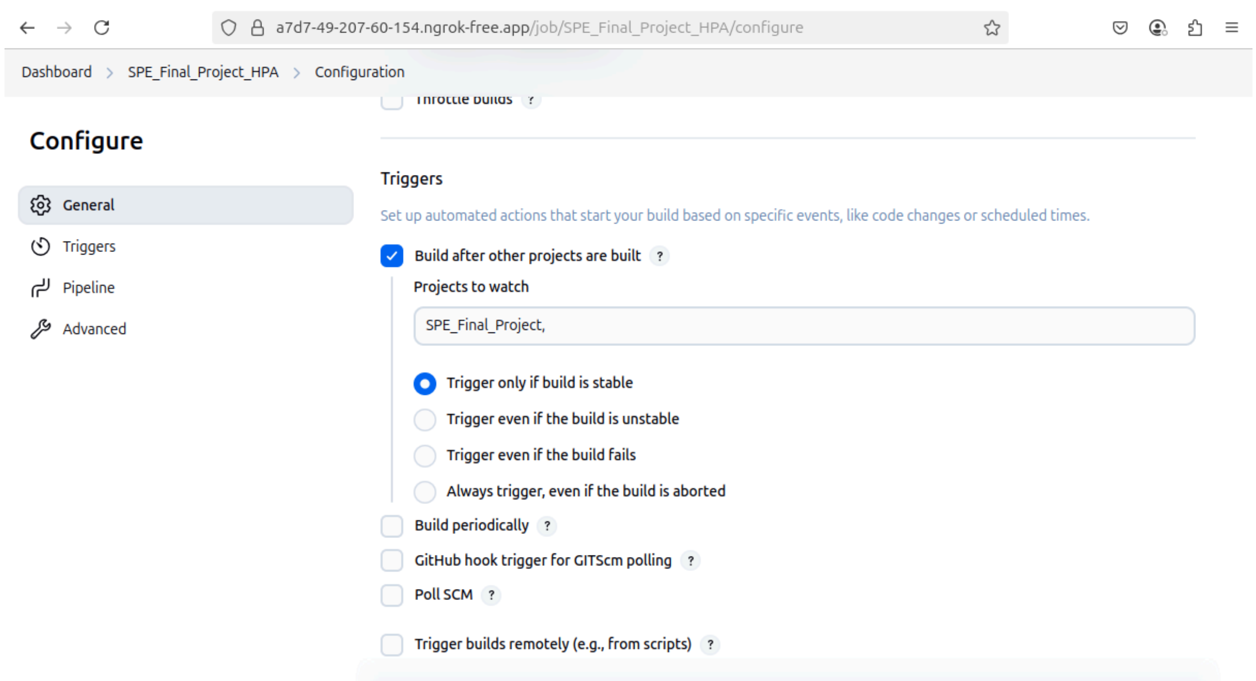
As we can see, all our deployment activities went successful. Here, we can see a couple more pods and deployments, these were brought up to introduce Horizontal Pod Autoscaling (HPA) and logging and monitoring in the pipeline.

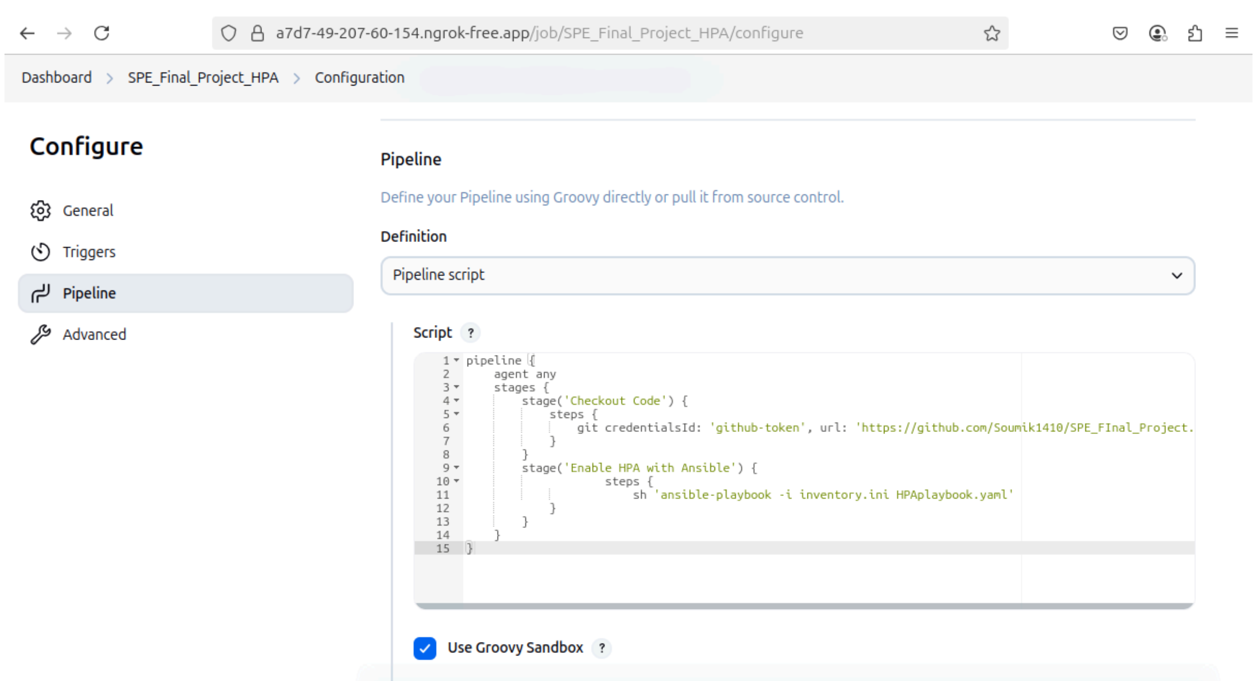
6. Horizontal Pod Autoscaler :

HPA (Horizontal Pod Autoscaler) is a Kubernetes resource that automatically scales the number of Pod replicas in a Deployment based on observed resource usage, such as CPU or memory, or on custom metrics.

For this to work in our cluster, we need to first bring up the metrics-server which measures metrics such as CPU usage or memory usage of pods. We need to set requests of CPU and memory usage for our image-caption-app Deployment. Then in our HPA configuration, we need to set the resource that we want to monitor, the target utilization (computed as a percentage of the requested value) and min and max replicas that we desire. Then based on current utilization vs target utilization, HPA will automatically scale up and scale down the number of replicas to ensure average resource utilization is close to the target threshold.

We define another Ansible playbook called HPAplaybook.yaml that lists all these tasks using the kubernetes.core.k8s module. This playbook is executed automatically once the application deployment is completed, by a Jenkins pipeline which checks out the latest version of this playbook from our Github repository and then executes this playbook. This is a separate pipeline from our first Jenkins pipeline that handled automated model training, testing, containerization and deployment of the application. This was done to modularize and independently test the HPA without needing to train the model time and time again. This second pipeline is automatically triggered on successful completion of the first pipeline.





HPAplaybook.yaml tasks :

Starting metrics server and waiting for it to be ready

github.com/Soumik1410/SPE_FInal_Project/blob/master/HPAplaybook.yaml

Soumik1410 Update HPAplaybook.yaml 02407db · 2 days ago History

Code Blame 84 lines (77 loc) · 2.56 KB Raw

```yaml
---
- name: Enable HPA on Minikube
  hosts: localhost
  gather_facts: no
  vars:
    deployment_name: image-caption-app
    namespace: image-caption-app-mt2024153
    container_name: captioner
    container_image: soumik1410/imagecaptioner_mt2024153:latest
    min_replicas: 1
    max_replicas: 5
    cpu_target_utilization: 80
    hpa_name: image-caption-app-mt2024153-hpa

  tasks:
    - name: Enable metrics-server addon in Minikube
      command: minikube addons enable metrics-server
      register: addon_result
      changed_when: "'enabled' in addon_result.stdout or 'already enabled' in addon_result.stdout"

    - name: Wait for metrics-server to be ready
      shell: kubectl rollout status deployment metrics-server -n kube-system
      retries: 5
      delay: 10
      register: rollout_status
      until: rollout_status.rc == 0
```

Patch our application deployment to have resource requests and limits

github.com/Soumik1410/SPE_FInal_Project/blob/master/HPAplaybook.yaml

master SPE_FInal_Project / HPAplaybook.yaml Top

Code Blame 84 lines (77 loc) · 2.56 KB Raw

```yaml
      until: rollout_status.rc == 0

    - name: Patch deployment with resource requests and limits
      kubernetes.core.k8s:
        state: patched
        kind: Deployment
        api_version: apps/v1
        namespace: "{{ namespace }}"
        name: "{{ deployment_name }}"
        definition:
          spec:
            template:
              spec:
                containers:
                  - name: "{{ container_name }}"
                    image: "{{ container_image }}"
                    resources:
                      requests:
                        cpu: "200m"
                      limits:
                        cpu: "500m"
```

Configure HPA

github.com/Soumik1410/SPE_FInal_Project/blob/master/HPAplaybook.yaml

master · SPE_FInal_Project / HPAplaybook.yaml

Code | Blame · 84 lines (77 loc) · 2.56 KB

```yaml
- name: Create or update HorizontalPodAutoscaler
  kubernetes.core.k8s:
    state: present
    definition:
      apiVersion: autoscaling/v2
      kind: HorizontalPodAutoscaler
      metadata:
        name: "{{ hpa_name }}"
        namespace: "{{ namespace }}"
      spec:
        scaleTargetRef:
          apiVersion: apps/v1
          kind: Deployment
          name: "{{ deployment_name }}"
        minReplicas: "{{ min_replicas }}"
        maxReplicas: "{{ max_replicas }}"
        metrics:
          - type: Resource
            resource:
              name: cpu
              target:
                type: Utilization
                averageUtilization: "{{ cpu_target_utilization }}"
```

Wait for HPA to start up and ready to start monitoring

```yaml
- name: Wait for HPA to be created
  shell: kubectl get hpa "{{ hpa_name }}" -n "{{ namespace }}"
  retries: 5
  delay: 5
  register: hpa_status
  until: hpa_status.rc == 0

- name: Show HPA status
  command: kubectl describe hpa "{{ hpa_name }}" -n "{{ namespace }}"
  register: hpa_describe

- debug:
    msg: "{{ hpa_describe.stdout }}"
```

Once this is executed, we can verify the status using kubectl commands.

```
jenkins@soumik-VirtualBox:~$ kubectl rollout status deployment metrics-server -n kube-system
deployment "metrics-server" successfully rolled out
jenkins@soumik-VirtualBox:~$ kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS        AGE
coredns-668d6bf9bc-zjgtz           1/1     Running   0               4h35m
etcd-minikube                      1/1     Running   0               4h36m
kube-apiserver-minikube            1/1     Running   0               4h35m
kube-controller-manager-minikube   1/1     Running   0               4h36m
kube-proxy-kb4st                   1/1     Running   0               4h35m
kube-scheduler-minikube            1/1     Running   0               4h36m
metrics-server-7fbb699795-5zxvg    1/1     Running   0               4h35m
storage-provisioner                1/1     Running   1 (4h35m ago)   4h35m
jenkins@soumik-VirtualBox:~$ kubectl top pods -n image-caption-app-mt2024153
NAME                                 CPU(cores)   MEMORY(bytes)
fluent-bit-c4zh6                     98m          12Mi
grafana-8559bcb694-2xpdc             2m           41Mi
image-caption-app-756c979f57-zmjnh   0m           7Mi
loki-87d578f98-c22dr                 33m          280Mi
jenkins@soumik-VirtualBox:~$
```

```
jenkins@soumik-VirtualBox:~$ kubectl describe deployment image-caption-app -n image-caption-app-mt2024153
Name:                   image-caption-app
Namespace:              image-caption-app-mt2024153
CreationTimestamp:      Thu, 22 May 2025 15:38:24 +0530
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               app=captioner
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=captioner
  Containers:
   captioner:
    Image:      soumik1410/imagecaptioner_mt2024153:latest
    Port:       8501/TCP
    Host Port:  0/TCP
    Limits:
      cpu:  500m
    Requests:
      cpu:         200m
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
```

```
jenkins@soumik-VirtualBox:~$ kubectl get hpa -n image-caption-app-mt2024153
NAME                              REFERENCE                        TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
image-caption-app-mt2024153-hpa   Deployment/image-caption-app    cpu: 0%/80%   1         5         1          4h36m
jenkins@soumik-VirtualBox:~$
```

7. Logging and Monitoring :