Reproducibility and Reusability

Objective:

Ensure reproducibility and reusability of machine learning models by building an automated pipeline using Docker and GitHub actions.

1. Git clone your repository.
   a. Change the directory.
2. **Create a Dockerfile:** The Dockerfile defines the environment for your project by specifying the base image, dependencies, and instructions for building the container.

JavaScript

```
FROM python:3.9-slim

WORKDIR /workspace

COPY . .

RUN apt-get update && apt-get install -y \
    build-essential \
    libssl-dev \
    libffi-dev \
    python3-dev

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 80

ENV NAME MLOpsLab

CMD ["python", "train.py"]
```

This Dockerfile contains the following steps:

   a. Base Image: We are using the `python:3.9-slim` base image, which is a lightweight Python environment.
   b. Working Directory: The container's working directory is set to `/workspace`.
   c. Copy Files: We copy all files from the current directory (your project directory) to the container.
   d. Install Dependencies: The container installs the dependencies listed in `requirements.txt`.
   e. Expose Port: If you're running a web service, this exposes port 80 (optional step).
   f. CMD: Specifies the command to run when the container starts. In this case, it runs `train.py` (a Python script for training your machine-learning model).

3. **Build the Docker Image:**
   As the Dockerfile and requirements.txt, we can build the Docker image.
   Navigate to the directory containing your Dockerfile and run the following command:

Unset

```
docker build -t mlops-lab-image .
```

Explanation:

   a. $docker\ build$: Command to build a Docker image.
   b. $-t\ mlops - lab - image$: Assigns a tag (name) to the image (mlops-lab-image).
   c. . - Refers to the current directory, which contains the Dockerfile.

4. **Run the Docker Container:**

After building the image, you can run the container. The CMD instruction in the Dockerfile specifies that the train.py script will execute when the container starts.

Unset

```
docker run mlops-lab-image
```

GitHub actions

In the cloned directory

1. **Create the .github/workflows Directory:**
   GitHub Actions look for workflow definitions in the .github/workflows folder. Start by creating this directory in your project's root folder:

Unset

```
mkdir -p .github/workflows
```

2. **Create a Workflow File**
   Inside the .github/workflows directory, create a file named ci.yml (or any other name like mlops-pipeline.yml). This file will define the steps that GitHub Actions will execute.

Unset

```
name: MLOps Pipeline CI

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Python 3.9
      uses: actions/setup-python@v2
      with:
        python-version: '3.9'  # Corrected syntax: colon after `with`

    - name: Install dependencies
      run: |
        pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run training script
      run: |
        python train.py
```

Explanation of Workflow Components:

- on: This specifies when the workflow will be triggered. In this case, it triggers on push and pull_request events on the main branch.
- jobs: Defines the different tasks the workflow will perform. In this case, there is a single job named build:

    - runs-on: Specifies the environment where the job will run. In this case, it's the latest Ubuntu image hosted by GitHub.
    - Steps:
        - actions/checkout@v2: This action checks out your repository code so the workflow can access it.
        - actions/setup-python@v2: This sets up Python 3.9 in the environment.
        - Install dependencies: Installs the required Python dependencies listed in requirements.txt.
        - Run training script: Executes your train.py script, which performs the machine learning tasks.
        - Run tests: Optionally, runs unit tests using pytest to ensure that the model training is successful and the code functions as expected.

3. **Commit the Workflow File:**
   Once the workflow file (ci.yml) is ready, add and commit the changes to your GitHub repository:

Unset

```
git add .github/workflows/ci.yml
git commit -m "Add GitHub Actions CI workflow"
git push origin main
```

4. **View Workflow Results:**
   After you push the changes to the main branch, GitHub Actions will automatically run the workflow defined in the ci.yml file.
    - Go to your repository on GitHub.
    - Click on the "Actions" tab. You will see a list of workflows that have run (or are currently running).
    - Click on the workflow run to see detailed logs and the status of each step.

   You will see something like this:
   Each job will be listed, and you can drill down into logs for each step to see if your pipeline is working as expected.

What Happens Behind the Scenes?

- GitHub Actions will automatically spin up a virtual machine with the environment specified in the workflow (Ubuntu, Windows, etc.).
- It checks out your code, installs dependencies, and runs the specified Python scripts.
- Any errors or failed steps will be logged and displayed in the Actions tab of your repository.

What Can You Automate with GitHub Actions?

- Experiment Tracking: You can automate the process of tracking model performance and versioning using MLflow integrated with your scripts.
- Continuous Integration (CI): Automatically check if your code passes tests (like pytest) every time there's a push.
- Docker Builds: Automatically build and push Docker images to a registry upon merging code changes.
- Deployment: Automate deployment to production environments (e.g., AWS, Azure) after successful runs.