Sample CI/CD Workflow with Amazon Linux 2023, basic flask application, github, github actions, and docker engine

1. Setting Up Amazon Linux 2023

Step 1: Launch an Amazon Linux 2023 Instance

a) Go to the AWS Management Console.
b) Navigate to EC2 and click Launch Instance.
c) Choose Amazon Linux 2023 (AL2023) as the AMI.
d) Select an instance type (e.g., t2.micro for free-tier).
e) Configure networking and add a key pair for SSH access.
f) Launch the instance and connect via SSH.

Step 2: Connect to the Instance

Use SSH to connect to your EC2 instance:

```
ssh -i your-key.pem ec2-user@your-ec2-public-ip
```

*Replace your-key.pem with your private key and your-ec2-public-ip with the instance's public IP.

Step 3: Update the System

Once logged in, update the system:

```
sudo dnf update -y
```

Step 4: Install Python and Dependencies

Install Python and necessary dependencies:

```
sudo dnf install python3 python3-pip python3-virtualenv -y
```

Verify installation:

```
python3 --version
pip3 --version
```

Step 5: Install Git

We need Git to work with GitHub:

```
sudo dnf install git -y
```

Verify installation:

```
git --version
```

Step 6: Install Docker

Docker will be required later for containerization:

```
sudo dnf install docker -y
```

Start and enable the Docker service:

```
sudo systemctl start docker
sudo systemctl enable docker
```

Add the current user to the docker group to run Docker without sudo:

```
sudo usermod -aG docker ec2-user
```

Log out and log back in for the group changes to take effect, then verify:

```
docker --version
```

2. Creating a Basic Flask Application

Now that the environment is ready, let's build a simple Flask app.

Step 1: Set Up a Virtual Environment

Create a project directory and move into it:

```
mkdir flask-app && cd flask-app
```

Create a virtual environment:

```
python3 -m venv venv
source venv/bin/activate
```

Step 2: Install Flask

```
pip install Flask
```

Step 3: Create a Simple Flask App

Create a file app.py and add the following code:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
  return "Hello, Amazon Linux 2023!"

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000)
```

Step 4: Run the Flask App

Run the app:

```
python app.py
```

*Visit http://your-ec2-public-ip:5000 in a browser to see the output.

3. Version Control with GitHub

Now let's push the Flask app to GitHub.

Step 1: Initialize Git

Inside the flask-app directory:

```
git init
git config --global user.name "YourName"
git config --global user.email "youremail@example.com"
```

Step 2: Create a .gitignore File

```
echo "venv/" > .gitignore
echo "__pycache__/" >> .gitignore
```

Step 3: Commit and Push the Code

Create a new GitHub repository from GitHub's website.

Add the remote repository:

```
git remote add origin https://github.com/yourusername/flask-app.git
```

Add, commit, and push the files:

```
git add .
git commit -m "Initial commit"
git branch -M main
git push -u origin main
```

4. GitHub Actions for CI/CD

Let's set up GitHub Actions to automate testing.

Step 1: Create a GitHub Actions Workflow

Inside the flask-app directory, create .github/workflows/main.yml:

```
name: Flask CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

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

      - name: Set up Python
```

```
    uses: actions/setup-python@v2
    with:
      python-version: "3.9"

  - name: Install dependencies
    run: |
      python -m venv venv
      source venv/bin/activate
      pip install Flask

  - name: Run tests
    run: python -m unittest discover
```

Push this workflow:

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

*GitHub will now automatically run this workflow when you push code.

5. Dockerizing the Flask Application

Step 1: Create a Dockerfile

Create a Dockerfile in the project root:

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

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

Step 2: Build and Run Docker Image

```
sudo usermod -aG docker $USER          #then exit to close the session, reconnect afterwards
docker build -t flask-app .
docker run -p 5000:5000 flask-app
```

5.1 Docker Commands

Check Docker version:

```
docker --version
```

Check system-wide Docker info:

```
docker info
```

List Available Images

```
docker images
```

Pull an Image from Docker Hub

```
docker pull ubuntu
```

Build a Docker Image

```
docker build -t myapp .
```

(-t assigns a name/tag to the image)

Tag an Image

```
docker tag myapp myrepo/myapp:v1
```

Remove an Image

```
docker rmi myapp
```

Run a Container

```
docker run -d -p 8080:80 myapp
```

(-d runs it in the background, -p maps ports)

List Running Containers

```
docker ps
```

List All Containers (Including Stopped Ones)

```
docker ps -a
```

Stop a Running Container

```
docker stop container_id
```

Start a Stopped Container

```
docker start container_id
```

Remove a Container

```
docker rm container_id
```

Run an Interactive Shell Inside a Running Container

```
docker exec -it container_id /bin/bash
```

View Container Logs

```
docker logs container_id
```

Follow Logs in Real-Time

```
docker logs -f container_id
```

Inspect a Running Container

```
docker inspect container_id
```

Check Resource Usage

```
docker stats
```

List Docker Volumes

```
docker volume ls
```

Create a Named Volume

```
docker volume create myvolume
```

List Docker Networks

```
docker network ls
```

Connect a Container to a Network

```
docker network connect mynetwork container_id
```

Stop All Running Containers

```
docker stop $(docker ps -q)
```

Remove All Stopped Containers

```
docker rm $(docker ps -a -q)
```

Remove All Images

```
docker rmi $(docker images -q)
```

Remove All Docker Volumes

```
docker volume rm $(docker volume ls -q)
```

5.2 Upload docker image to Docker Hub

a) Go to hub.docker.com, sign in or sign up

b) Back to the machine with docker container, use the following command to login

```
docker login
```

c) Tag the Docker Image

   a. List your local Docker images:

```
docker images
```

   b. You'll see an output like:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
flask-app	latest	a1b2c3d4e5f6	10 minutes ago	120MB

d) To push this image to Docker Hub, you need to tag it with your Docker Hub username:

```
docker tag flask-app your-dockerhub-username/flask-app:latest
```

e) For example, if your Docker Hub username is john123, run:

```
docker tag flask-app john123/flask-app:latest
```

f) Push the Image to Docker Hub

```
docker push your-dockerhub-username/flask-app:latest
```

g) Example:

```
docker push john123/flask-app:latest
```

h) Verify on Docker Hub, Go to Docker Hub.

i) Log in and check the Repositories tab to see your uploaded image.

j) Pull and Run from Any Machine, To pull the image on another machine:

```
docker pull your-dockerhub-username/flask-app:latest
```

k) Then, run it:

```
docker run -p 5000:5000 your-dockerhub-username/flask-app
```