Sessional 1 - Solution

Q1 — CI on GitHub: flake8 → pytest → build & push Docker image

High-level steps:
1. GitHub Actions for CI (runs on push to main).
2. Makefile to provide standard targets (lint, test, build, push).
3. Dockerfile to build the Flask app image.
4. GitHub Secrets for DOCKERHUB_USERNAME, DOCKERHUB_PASSWORD.
5. requirements.txt or pyproject.toml to install dependencies.

Example Makefile

```
PYTHON := python3
PIP := pip3
IMAGE_NAME := yourdockerhubuser/flask-app
TAG := $(shell git rev-parse --short HEAD)

.PHONY: install lint test build push

install:
	$(PIP) install -r requirements.txt

lint:
	flake8 src tests

test:
	pytest -q

build:
	docker build -t $(IMAGE_NAME):$(TAG) .

push:
	echo "$(DOCKERHUB_PASSWORD)" | docker login -u "$(DOCKERHUB_USERNAME)"
--password-stdin
	docker push $(IMAGE_NAME):$(TAG)
```

Example Dockerfile

```
FROM python:3.11-slim
```

```
WORKDIR /app
RUN apt-get update && apt-get install -y --no-install-recommends build-essential && rm -rf
/var/lib/apt/lists/*
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY src/ ./src
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENV PYTHONUNBUFFERED=1
EXPOSE 5000
CMD ["/entrypoint.sh"]
```

Example GitHub Actions Workflow

```
name: CI
on:
  push:
    branches: [ main ]
jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - name: Install deps
        run: pip install -r requirements.txt
      - name: Lint
        run: make lint
      - name: Test
        run: make test
      - name: Build Docker image
        run: docker build -t yourdockerhubuser/flask-app:${{ github.sha }} .
      - name: Push to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_PASSWORD }}
      - name: Push
        run: |
          docker tag yourdockerhubuser/flask-app:${{ github.sha }} yourdockerhubuser/flask-
  app:latest
          docker push yourdockerhubuser/flask-app:${{ github.sha }}
          docker push yourdockerhubuser/flask-app:latest
```

Q2 — Docker Compose for Flask app + MongoDB + Persistent Volume

Key requirement: MongoDB data must persist across container restarts — implement a named volume mounted to MongoDB's /data/db directory.

Example docker-compose.yml

```
version: "3.8"
services:
  mongo:
    image: mongo:6.0
    restart: unless-stopped
    volumes:
      - mongo-data:/data/db
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: examplepassword
    networks:
      - app-network

  web:
    image: yourdockerhubuser/flask-app:latest
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - mongo
    environment:
      MONGO_URI:
mongodb://admin:examplepassword@mongo:27017/yourdb?authSource=admin
      FLASK_ENV: production
    ports:
      - "5000:5000"
    networks:
      - app-network

volumes:
  mongo-data:

networks:
  app-network:
    driver: bridge
```

The named volume `mongo-data:/data/db` ensures persistence across container restarts.

Q3 — Local make checks + Jenkins pipeline after merge

Workflow:
1. Developer runs make lint and make test locally.
2. PR is created and optionally validated in GitHub Actions.
3. After merging to main, Jenkins triggers automatically to:
   - Pull latest code
   - Run lint/test
   - Build Docker image
   - Push image to registry
   - Deploy to target environment

Example Jenkinsfile

```
pipeline {
  agent any
  environment {
    IMAGE = "yourdockerhubuser/flask-app"
    TAG = "${env.GIT_COMMIT.take(8)}"
  }
  stages {
    stage('Checkout') {
      steps { checkout scm }
    }
    stage('Lint & Test') {
      steps {
        sh 'make lint'
        sh 'make test'
      }
    }
    stage('Build') {
      steps {
        sh "docker build -t ${IMAGE}:${TAG} ."
      }
    }
    stage('Push') {
      steps {
        withCredentials([usernamePassword(credentialsId: 'dockerhub-cred-id',
usernameVariable: 'DH_USER', passwordVariable: 'DH_PASS')]) {
          sh 'echo "$DH_PASS" | docker login -u "$DH_USER" --password-stdin'
          sh "docker push ${IMAGE}:${TAG}"
          sh "docker tag ${IMAGE}:${TAG} ${IMAGE}:latest && docker push ${IMAGE}:latest"
        }
      }
    }
    stage('Deploy') {
      steps {
        sh './scripts/deploy.sh ${IMAGE} ${TAG}'
      }
    }
  }
}
```

How Makefile and Jenkins interact:
- Jenkins uses the same make targets (lint, test, build, push) as developers.
- Developers ensure pre-merge code quality via local make.
- Jenkins ensures the merged code is validated and built reproducibly.