**E-commerce Web Application using Jenkins Pipeline ,docker**

**Introduction**  
You are a DevOps Engineer at ShopEase Inc., a growing e-commerce startup that specializes in selling sustainable and eco-friendly products. As the company scales, the development team is frequently pushing updates to the backend application — built Python (or your chosen language) — which powers the entire shopping platform.

The manual process of building, testing, and deploying the application is time-consuming and error-prone. Recent incidents have shown that manual deployments often lead to version mismatches and delayed hotfixes, causing frequent downtime during peak shopping hours. This affects user experience and leads to revenue loss.

To tackle these issues, the company decides to automate the entire CI/CD process using Jenkins Pipelines. The goal is to ensure that every time a developer commits code to the GitHub repository, the following actions should happen automatically:

**Project structure**

**ecommerce-project/**

**|── backend/**

**│ ── app.py # Basic Flask App**

**│── requirements.txt # Python dependencies**

**│── Dockerfile # Backend Dockerfile**

**|── Jenkinsfile # Jenkins pipeline definition**

**└── README.md**

**backend/app.py**

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/products', methods=['GET'])

def get\_products():

products = [

{"id": 1, "name": "Laptop", "price": 800},

{"id": 2, "name": "Smartphone", "price": 600},

]

return jsonify(products)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=5000)

**Backend/Dockerfile**

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

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

**Create the Jenkins code for Ecommerce website.**

Set Up Jenkins Job

1. Create a Jenkins Job:
   * Go to Jenkins Dashboard → New Item → Pipeline.
2. Source Code Management:
   * Choose Git → Provide Repository URL.
3. Pipeline Definition:
   * Select Pipeline script from SCM.
   * Choose Jenkinsfile from the repository.
4. Add Docker Credentials:
   * Go to Jenkins → Manage Jenkins → Manage Credentials.
   * **Add Docker Hub credentials with ID dockerhub-credentials.**

**1) Write the Jenkinsfile that implements this pipeline, adhering to the specified stages and post-build conditions.Your goal is to create a Jenkins pipeline code in groove script with the following requirements:**

1. Environment Setup:
   * Define environment variables:
     + DOCKER\_IMAGE: The name of the Docker image should be my-ecommerce-backend.
     + DOCKER\_TAG: Dynamically set to the Jenkins build number.
     + REGISTRY: The Docker Hub repository should be dockerhub\_user/my-ecommerce.
2. Pipeline Stages:
   * Checkout Code:
     + Clone the repository from GitHub: https://github.com/your-user/ecommerce-project.git.
   * Build Docker Image:
     + Inside the backend directory, build a Docker image tagged as ${DOCKER\_IMAGE}:${DOCKER\_TAG}.
   * Push to Docker Hub:
     + Push the built Docker image to Docker Hub using the Jenkins credentials ID: dockerhub-credentials.
   * Deploy to Local Docker:
     + Stop any running container named ecommerce, remove it if it exists, and run a new container.
     + The container should map port 5000 on the host to 5000 inside the container.
     + The new container should use the image ${DOCKER\_IMAGE}:${DOCKER\_TAG} and be named ecommerce.
3. Post-Build Actions:
   * On Success, print the message "✅ Deployment Successful!".
   * On Failure, print the message "❌ Deployment Failed!"

**To check the running container**

* **docker ps**

**Test the Application:**

* **curl** [**http://localhost:5000/products**](http://localhost:5000/products)
* After running the pipeline job, go to Jenkins Dashboard > Your Project > Last Build.
* Click on Console Output.
* A successful pipeline should display messages like:

**Deployment Successful!**

**Finished: SUCCESS**

**If the pipeline failed, you'll see:**

**Deployment Failed! Finished: FAILURE**