**DOCKERS ADVANCED PROJECTS WITH STEP BY STEP SOLUTIONS**

**Project 1: Dockerized CI/CD Pipeline using Jenkins and Kubernetes**

**Objective:** Automate **build, test, and deployment** using Jenkins, Docker, and Kubernetes.

**Tech Stack:**

* **Jenkins** (CI/CD automation)
* **Docker** (Containerization)
* **Kubernetes (K8s)** (Deployment & Scaling)
* **GitHub/GitLab** (Source Code Management)
* **Helm** (Kubernetes Package Management)

**Step 1: Install Jenkins on Docker**

bash

CopyEdit

docker run -p 8080:8080 -p 50000:50000 --name jenkins \

-v jenkins\_home:/var/jenkins\_home jenkins/jenkins:lts

🔹 **Access Jenkins at:** http://localhost:8080

**Step 2: Install Jenkins Plugins**

* Go to **Manage Jenkins → Plugins**
* Install:
  + **Docker Pipeline Plugin**
  + **Kubernetes Plugin**
  + **Git Plugin**

**Step 3: Create Jenkinsfile for CI/CD**

groovy

CopyEdit

pipeline {

agent any

stages {

stage('Clone Code') {

steps {

git 'https://github.com/user/repo.git'

}

}

stage('Build Docker Image') {

steps {

sh 'docker build -t myapp:latest .'

}

}

stage('Push to Docker Hub') {

steps {

withDockerRegistry([credentialsId: 'docker-hub-cred', url: '']) {

sh 'docker tag myapp:latest user/myapp:latest'

sh 'docker push user/myapp:latest'

}

}

}

stage('Deploy to Kubernetes') {

steps {

sh 'kubectl apply -f k8s-deployment.yaml'

}

}

}

}

**Step 4: Kubernetes Deployment YAML**

**k8s-deployment.yaml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: myapp

image: user/myapp:latest

ports:

- containerPort: 80

**Step 5: Run the Jenkins Pipeline**

1. Commit & Push Code to GitHub.
2. Jenkins triggers the **CI/CD pipeline**.
3. Docker image is **built & pushed** to Docker Hub.
4. Kubernetes **deploys the application**.

✅ **Outcome:** Fully automated **CI/CD pipeline using Jenkins, Docker & Kubernetes**.

**Project 2: Multi-Node Docker Swarm Cluster for High Availability**

**Objective:** Deploy a **highly available application** using **Docker Swarm**.

**Tech Stack:**

* **Docker Swarm** (Cluster Orchestration)
* **NGINX** (Load Balancer)
* **Redis + PostgreSQL** (Database Layer)

**Step 1: Initialize Docker Swarm**

On the manager node, run:

bash

CopyEdit

docker swarm init --advertise-addr <MANAGER-IP>

Copy the **join token** and add worker nodes:

bash

CopyEdit

docker swarm join --token <TOKEN> <MANAGER-IP>:2377

🔹 **Check cluster status:**

bash

CopyEdit

docker node ls

**Step 2: Create a Docker Compose File**

**docker-compose.yml**

yaml

CopyEdit

version: '3.8'

services:

nginx:

image: nginx:latest

ports:

- "80:80"

deploy:

replicas: 3

restart\_policy:

condition: on-failure

redis:

image: redis:alpine

deploy:

replicas: 2

restart\_policy:

condition: on-failure

postgres:

image: postgres:13

environment:

POSTGRES\_USER: admin

POSTGRES\_PASSWORD: password

deploy:

replicas: 1

restart\_policy:

condition: on-failure

**Step 3: Deploy the Application on Swarm**

bash

CopyEdit

docker stack deploy -c docker-compose.yml myapp

🔹 **Check service status:**

bash

CopyEdit

docker service ls

✅ **Outcome:** Application runs in a **multi-node Swarm cluster with auto-healing and load balancing**.