Full Deployment Flow Using Jenkins and Docker Compose

**1. Set Up SSH Trust (Jenkins Master ↔ Agent)**

**On master node:**

cd ~/.ssh

ssh-keygen # Leave passphrase blank

cat ~/.ssh/id\_rsa.pub # Copy this

**On agent node (EC2):**

cd ~/.ssh

vim authorized\_keys # Paste the copied public key

chmod 600 authorized\_keys

This lets Jenkins connect to the agent securely over SSH.

**2. Add Agent Node in Jenkins UI**

Navigate: Manage Jenkins → Nodes → New Node  
Set up:

| **Field** | **Value** |
| --- | --- |
| Node name | k8s |
| Type | Permanent Agent |
| Remote root directory | /home/ubuntu |
| Labels | k8s |
| Launch Method | SSH |
| Host | <EC2 public IP> of Agent Node |
| Credentials | ubuntu user + private key |
| Verification | No verification needed |

Make sure your EC2 allows inbound SSH and your Jenkins server can reach it.

**3. Create Shell Script for Docker Setup**

**File:** scripts/docker-setup.sh

#!/bin/bash

sudo apt-get update

sudo apt-get install docker.io -y

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker $USER

newgrp docker

docker --version

sudo apt-get install docker-compose -y

sudo apt-get install docker-compose-plugin -y || echo "Plugin not found, continuing..."

sudo curl -SL https://github.com/docker/compose/releases/download/v2.32.0/docker-compose-linux-x86\_64 -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

docker-compose --version

echo "Docker & Docker Compose installed successfully!"

sudo usermod -aG docker $USER

newgrp docker

echo "Created the new group and updated docker successfully!"

**Make executable:**

chmod +x docker-setup.sh

**Adding DockerHub Credentials in Jenkins**

**Steps:**

1. Go to **Manage Jenkins** → **Credentials**
2. Select **(global)** domain
3. Click **Add Credentials**
4. Fill in the following:

| **Field** | **Value** |
| --- | --- |
| Kind | Username and Password |
| Scope | Global |
| Username | *Your DockerHub username* |
| Password | *Your DockerHub personal access token* |
| ID | dockerhubCred |
| Description | DockerHub credentials for Jenkins pipeline |

1. Click **Create**

**4. Create docker-compose.yml to Orchestrate Services**

Ensure it's placed in your project root and matches this structure:

version: '3.4'

services:

mysql-service:

image: mysql:8.0

environment:

MYSQL\_ROOT\_PASSWORD: "root"

MYSQL\_DATABASE: "quantumsoft"

MYSQL\_PASSWORD: "root"

ports:

- "3306:3306"

volumes:

- mysql-data:/var/lib/mysql

networks:

- mynetwork

healthcheck:

test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]

interval: 10s

retries: 5

start\_period: 30s

backend:

image: spquantum/backend:q1

environment:

SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql-service:3306/quantumsoft

SPRING\_DATASOURCE\_USERNAME: root

SPRING\_DATASOURCE\_PASSWORD: root

ports:

- "8080:8080"

depends\_on:

mysql-service:

condition: service\_healthy

networks:

- mynetwork

frontend:

image: spquantum/frontend:q1

ports:

- "5502:80"

depends\_on:

- backend

networks:

- mynetwork

networks:

mynetwork:

volumes:

mysql-data:

**5. Create the Jenkins Pipeline (Groovy)**

pipeline {

agent { label "k8s" }

environment {

FRONTEND\_IMAGE = "spquantum/frontend:q1"

BACKEND\_IMAGE = "spquantum/backend:q1"

}

stages {

stage("Clone Code") {

steps {

git url: "https://github.com/sreyaspatil5/Kubernetes-three-tier-deployment.git", branch: "main"

}

}

stage("Setup Docker & Compose") {

steps {

sh "chmod +x scripts/docker-setup.sh"

sh "scripts/docker-setup.sh"

}

}

stage("Build Images") {

steps {

sh "docker build -t ${FRONTEND\_IMAGE} frontend"

sh "docker build -t ${BACKEND\_IMAGE} backend"

}

}

stage("Push To DockerHub") {

steps {

withCredentials([usernamePassword(

credentialsId: "dockerHubCred",

usernameVariable: "dockerHubUser",

passwordVariable: "dockerHubPass"

)]) {

sh "docker login -u ${env.dockerHubUser} -p ${env.dockerHubPass}"

sh "docker tag ${FRONTEND\_IMAGE} $dockerHubUser/frontend:q1"

sh "docker tag ${BACKEND\_IMAGE} $dockerHubUser/backend:q1"

sh "docker push $dockerHubUser/frontend:q1"

sh "docker push $dockerHubUser/backend:q1"

}

}

}

stage("Deploy Application") {

steps {

echo "This is deploying application"

sh "docker-compose up --build -d"

}

}

}

}

**6. Final Jenkins Setup**

Inside Jenkins:

* ✅ Create a new item → Pipeline
* ✅ Give description and labels
* ✅ Set GitHub project URL
* ✅ Enable: GitHub hook trigger for GITScm polling
* ✅ Add your pipeline script above
* ✅ Save and Build the job

**Enable GitHub Webhook Trigger for Jenkins Pipeline**

**1. Ensure Jenkins Has GitHub Integration**

Install or verify these plugins:

* **GitHub plugin**
* **GitHub Integration Plugin**
* **Git plugin**

Go to: Manage Jenkins → Plugins → Installed / Available

**2. Update Pipeline Config in Jenkins Job**

* Go to: your pipeline job → Configure
* Check these boxes:
  + GitHub project
    - URL: https://github.com/sreyaspatil5/Kubernetes-three-tier-deployment
  + Build Triggers → GitHub hook trigger for GITScm polling

This tells Jenkins to watch for webhook pings.

**3. Configure GitHub Webhook**

Inside your GitHub repo:

1. Go to **Settings → Webhooks**
2. Click **"Add webhook"**
3. Fill in:

| **Field** | **Value** |
| --- | --- |
| Payload URL | http://<your-jenkins-server>:8080/github-webhook/ |
| Content Type | application/json |
| Which events | Just the push event |
| Active | (checked) |

1. Click **Add Webhook**

This sends a POST request to Jenkins whenever code is pushed.

**4. Verify Connection**

* Push a small change to the repo
* In Jenkins → Job → Build History, check if a build was triggered
* In GitHub → Webhooks → "Recent Deliveries", check for a 200 OK

**Optional: Polling Fallback**

If webhooks fail (e.g., firewall issues), enable:

triggers {

pollSCM('\* \* \* \* \*') // Poll every minute (can adjust interval)

}

This goes inside your pipeline’s options or in declarative syntax within your Jenkinsfile.

**7. Monitor Build Logs & Check Containers**

docker ps # Check running containers

docker logs <id> # View logs

Access the app:

* Frontend: http://<EC2-IP>:5502
* Backend API: http://<EC2-IP>:8080