BLACKBOXAI

Generated on: 7/11/2025 10:13:07 PM

User

📘 CI/CD Pipeline with Jenkins, Docker, and Kubernetes for Django Beginners

Tailored for First-Year College Students

✅ Section 1: What is CI/CD? (Beginner Friendly)

- CI = Continuous Integration: Automatically test and integrate code every time a developer pushes changes.
- CD = Continuous Deployment/Delivery: Automatically deliver tested code to a production/staging environment.

Why CI/CD?

- Speeds up development
- Reduces bugs in production
- Makes code collaboration easy

✅ Section 2: Tech Stack Overview

Layer	Tools/Tech	Purpose
Frontend	HTML, CSS, JS	User Interface
Backend	Django (Python)	Business logic, APIs
CI/CD Tool	Jenkins	Automate testing and deployment
Container	Docker	Package app and its dependencies
Orchestration	Kubernetes	Run and manage containers at scale
Deployment Manager	Helm	Templated K8s YAML deployment
Monitoring	Prometheus + Grafana	Watch app health and metrics

✅ Section 3: Project Goal

Build & Deploy a Django Login App with CI/CD using Jenkins → Docker → Kubernetes (with Helm)

✅ Section 4: Step-by-Step Breakdown

🔧 Step 1: Django Project Setup (Login App)

Create Project

```
 bash
django-admin startproject loginapp
cd loginapp
python manage.py startapp accounts
```

Basic Login View (accounts/views.py)

```
 python
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login


 def user_login(request):
    if request.method == "POST":
        username = request.POST["username"]
        password = request.POST["password"]
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
            return redirect("home")
    return render(request, "login.html")
```

🐳 Step 2: Dockerize the Django App

Dockerfile

```
 dockerfile
FROM python:3.11-slim

WORKDIR /app

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

COPY . .
```

```
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

docker-compose.yml (Optional)

```
yaml
version: '3'
services:
  web:
    build: .
    ports:
      - "8000:8000"
```

Build & Run Docker

```
bash
docker build -t django-login-app .
docker run -p 8000:8000 django-login-app
```

---

🤖 Step 3: Jenkins Setup for CI/CD

Install Jenkins Plugins

- Docker
- Git
- Pipeline

Jenkinsfile

```
groovy
pipeline {
    agent any


    stages {
        stage('Clone Repo') {
            steps {
                git 'https://github.com/yourname/django-login-app.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t login-app:latest .'
            }
        }
        stage('Push to DockerHub') {
            steps {
                withCredentials([string(credentialsId: 'dockerhub-pass', variable: 'PASS')]) {
                    sh 'echo $PASS | docker login -u yourname --password-stdin'
                    sh 'docker tag login-app:latest yourname/login-app:latest'
                    sh 'docker push yourname/login-app:latest'
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                sh 'kubectl apply -f k8s/'
            }
        }
    }
}
```

☸️ Step 4: Kubernetes Deployment

k8s/deployment.yaml

```
 yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: login-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: login
  template:
    metadata:
      labels:
        app: login
    spec:
      containers:
      - name: login-app
        image: yourname/login-app:latest
        ports:
        - containerPort: 8000
```

k8s/service.yaml

```
 yaml
apiVersion: v1
kind: Service
metadata:
  name: login-service
spec:
  type: LoadBalancer
  selector:
    app: login
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
```

📦 Step 5: Helm Chart (Simplified Deployment)

Helm Structure

```
login-chart/
  Chart.yaml
  values.yaml
  templates/
    deployment.yaml
    service.yaml
```

Command to Install:

```
 bash
helm install login-app ./login-chart
```

📈 Step 6: Monitoring (Prometheus + Grafana)

- Prometheus: Collect metrics from Django + K8s nodes.
- Grafana: Visualize data with dashboards.

Basic Monitoring Steps

- 1. Install Prometheus and Grafana via Helm:

```
 bash
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

Generated by BlackBox AI

```
helm install prometheus prometheus-community/kube-prometheus-stack
```

- 2. Access Grafana:

```
bash
kubectl port-forward svc/prometheus-grafana 3000:80
```

🐳 Step 4: Dockerize the Django App

🔹 Dockerfile

```
dockerfile

Use a lightweight Python image


FROM python:3.11-slim

Set working directory


WORKDIR /app

Copy and install dependencies


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

Copy project files


COPY . .

Run Django server


CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

🔹 Docker Commands

```
bash

Build Docker image


docker build -t django-login-app .

Run the container


docker run -p 8000:8000 django-login-app
```

🤖 Step 5: Jenkins for CI/CD

🔹 Install These Jenkins Plugins:

- Git
- Docker Pipeline
- Pipeline
- Kubernetes CLI Plugin

🔹 Jenkinsfile

```
groovy
pipeline {
  agent any

  stages {
    stage('Clone Code') {
      steps {
        git 'https://github.com/yourname/django-login-app.git'
      }
    }

    stage('Run Tests') {
      steps {
        sh 'python manage.py test'
      }
    }

    stage('Build Docker Image') {
      steps {
        sh 'docker build -t django-login-app .'
      }
    }
    stage('Push to DockerHub') {
      steps {
        withCredentials([string(credentialsId: 'dockerhub-pass', variable: 'PASS')]) {
          sh 'echo $PASS | docker login -u yourdockerusername --password-stdin'
          sh 'docker tag django-login-app yourdockerusername/django-login-app:latest'
          sh 'docker push yourdockerusername/django-login-app:latest'
        }
      }
    }


     stage('Deploy to Kubernetes') {
      steps {
        sh 'kubectl apply -f k8s/'
      }
    }
  }
}
```

🧪 Step 6: Add Selenium for UI Testing in Django

🔹 Install Selenium

```
 bash
pip install selenium
```

🔹 Basic Test Case (tests/test_login.py)

```
 python
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

driver = webdriver.Chrome()
driver.get("http://localhost:8000/login/")

username = driver.find_element(By.NAME, "username")
password = driver.find_element(By.NAME, "password")
submit = driver.find_element(By.XPATH, "//button[@type='submit']")

username.send_keys("admin")
password.send_keys("admin123")
submit.click()
```

```
 time.sleep(3)
driver.quit()
```

◆ Jenkins Integration (Optional)

Add this to Jenkins pipeline before building Docker image:

```
 groovy
stage('Run Selenium Tests') {
  steps {
    sh 'pytest tests/test_login.py'
  }
}
```

☸ Step 7: Kubernetes Deployment

◆ Deployment YAML (k8s/deployment.yaml)

```
 yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: django-login
spec:
  replicas: 2
  selector:
    matchLabels:
      app: login
  template:
    metadata:
      labels:
        app: login
    spec:
      containers:
      - name: login
        image: yourdockerusername/django-login-app:latest
        ports:
        - containerPort: 8000
```

◆ Service YAML (k8s/service.yaml)

```
 yaml
apiVersion: v1
kind: Service
metadata:
  name: login-service
spec:
  type: LoadBalancer
  selector:
    app: login
  ports:
    - port: 80
      targetPort: 8000
```

📦 Step 8: Helm for Easy Kubernetes Deployment

🔹 Helm Chart Directory Structure

```
 login-chart/
├── Chart.yaml
├── values.yaml
└── templates/
    ├── deployment.yaml
    └── service.yaml
```

🔹 Command to Deploy via Helm

```
 bash
helm install login-app ./login-chart
```

📈 Step 9: Monitoring with Prometheus + Grafana

🔹 Install via Helm

```
 bash
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update


 helm install monitoring prometheus-community/kube-prometheus-stack
```

🔹 Access Grafana Dashboard

```bash
kubectl port-forward svc/monitoring-grafana 3000:80
```

Then open: `http://localhost:3000`

- Username: admin
- Password: admin (by default)

✅ Summary

You've now covered:

- Django development & Selenium testing
- Dockerizing your app
- Jenkins CI/CD pipelines
- Kubernetes deployments
- Helm templating
- Monitoring with Prometheus and Grafana

--- Excellent! Let's now extend the tutorial with Best Practices, Security Tips, and Real-World Industry CI/CD Structures. This will help you align with how top companies work while maintaining simplicity for learning.

📌 Step 10: Best Practices in CI/CD, Docker, Kubernetes & Django

🔹 General Best Practices

- Use Version Control (e.g., Git) for everything: code, config, infrastructure.
- Automate Everything using Jenkins, GitHub Actions, or GitLab CI.
- Keep Pipelines Fast: Split long builds/tests into smaller parallel stages.
- Fail Fast: Stop builds as soon as something fails.

🔹 Docker Best Practices

- Use lightweight base images (e.g., `python:3.11-slim`).

- Create a `.dockerignore` file to exclude unnecessary files.
- Avoid using `latest` tag in production; always use versioned images.
- Don't store secrets inside Dockerfiles or images.

◆ Django Best Practices

- Use `ALLOWED_HOSTS` and `DEBUG=False` in production.
- Store secrets (like API keys) in environment variables.
- Use Gunicorn + Nginx for serving Django apps in production.
- Apply Django security settings: CSRF, CORS, secure cookies, etc.

◆ Jenkins Best Practices

- Lock credentials using Jenkins Credentials Manager.
- Set up branch-specific pipelines using multibranch jobs.
- Keep Jenkinsfiles in your repo for transparency.
- Backup Jenkins configuration regularly.

◆ Kubernetes Best Practices

- Use readiness and liveness probes in deployments.
- Set resource limits (`cpu` , `memory`) for each container.
- Use namespaces to separate environments (dev, staging, prod).
- Always use RBAC (Role-Based Access Control) for access permissions.
- Avoid hard-coding secrets; use Kubernetes Secrets.

---

🔐 Step 11: Security Tips for CI/CD and Cloud Apps

◆ In Jenkins

- Use Role-based Access Plugin to control user roles.
- Don’t expose Jenkins to the internet directly.
- Use HTTPS and secure admin credentials.

◆ In Docker

- Scan images with tools like Trivy or Docker Scout.

- Use non-root users in Dockerfiles where possible.
- Always sign and verify images if using private registries.

🔹 In Kubernetes

- Use NetworkPolicies to control pod communication.
- Avoid running containers as `root`.
- Restrict access to `kubectl` using service accounts with minimal privileges.

🔹 For Django

- Never commit `.env` or `settings.py` files with secrets.
- Enable HTTPS (SSL) using Ingress + TLS.
- Use django-secure, whitenoise, and secure settings in production.

🏢 Step 12: Real-World CI/CD Structure in MNCs

Here's how real-world companies like Google, Infosys, Microsoft, or Zoho implement CI/CD pipelines:

🔹 Dev Team Workflow

- 1. Developer pushes code → GitHub/GitLab triggers Jenkins
- 2. Jenkins checks code → Runs unit tests + Selenium tests
- 3. Jenkins builds Docker image → Tags with Git SHA
- 4. Image pushed to private Docker registry (e.g., AWS ECR, GCR, Harbor)

🔹 Staging Pipeline

- 5. Jenkins deploys the image to staging namespace in Kubernetes
- 6. Runs Helm upgrade/install
- 7. QA/PMs test via staging URL

🔹 Production Pipeline

- 8. Once staging is approved, a manual approval stage (or Git tag) triggers production
- 9. Jenkins deploys to production namespace with different Helm `values.yaml`
- 10. Monitoring (Prometheus + Grafana) checks live traffic

- 11. Alerts go to Slack, PagerDuty, or email on failure

◆ Tools Used by MNCs

| Area | Common Tools | | ---------------- | ---------------------------------- | | Source Control | GitHub, GitLab, Bitbucket | | CI/CD | Jenkins, GitHub Actions, GitLab CI | | Containers | Docker | | Orchestration | Kubernetes + Helm | | Testing | Pytest, Selenium, Postman | | Monitoring | Prometheus, Grafana, ELK Stack | | Secrets Handling | Vault, AWS Secrets Manager, K8s |

✅ Final Thoughts

With this extended tutorial, you now understand:

- How to build and Dockerize a Django app
- How Jenkins automates CI/CD
- Deploying to Kubernetes with Helm
- Writing Selenium UI tests
- Monitoring and best practices used by real companies

- Create a PDF visual architecture diagram of the CI/CD flow?

merge this content and give the downloadable docx format

BLACKBOXAI

I don't know!