Building a CI/CD Pipeline for a Flask Application with Jenkins, Docker, and Kubernetes

Mohammed Talha Kalimi (DevOps Engineer)

LinkedIn: www.linkedin.com/in/mohammedtalhakalimi

Blog: https://medium.com/@kalimitalha8

Agenda

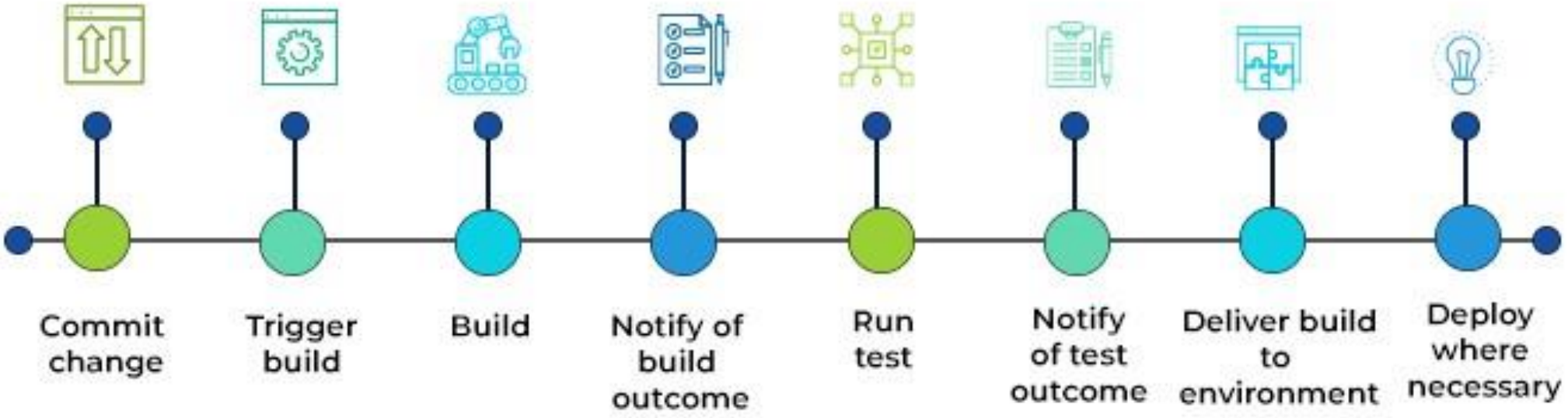
- Introduction
- Step 1: Create a Sample Flask Project
- Step 2: Configure Jenkins
- Step 3: Run the Pipeline
- Step 4: Verify Deployment
- Conclusion

Introduction

- In this Slides, we'll walk through the process of creating a simple Flask web application and setting up a CI/CD pipeline for it using Jenkins, Docker, and Kubernetes. This guide is perfect for developers and DevOps engineers looking to automate their deployment workflows and ensure a smooth delivery process.

TOOLBOX™

CI/CD PIPELINE



Step 1: Create a Sample Flask Project

- **Project Structure**
- First, we'll create a simple Flask web application. Here's the structure of our project:

```
sample-project/
├── app/
│   ├── app.py
│   ├── Dockerfile
├── Jenkinsfile
├── deployment.yaml
├── requirements.txt
└── README.md
```

app/app.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

requirements.txt

- List of dependencies for our Flask app.

- Makefile
- Flask==2.0.3

app/Dockerfile

Dockerfile

FROM python:3.9-slim

```
WORKDIR /app

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

COPY . .

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

deployment.yaml

- Kubernetes deployment and service configuration.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: sample-app
        image: <your-dockerhub-username>/sample-app:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: sample-app-service
spec:
  selector:
    app: sample-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Jenkinsfile

Jenkins pipeline configuration.

```
pipeline {
    agent any

stages {
        stage('Clone Repository') {
            steps {
                git 'https://github.com/username/sample-project.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                script {
                    dockerImage = docker.build("username/sample-
app:latest")
                }
            }
        }
        stage('Push Docker Image') {
            steps {
                script {

docker.withRegistry('https://index.docker.io/v1/', 'dockerhub-
credentials') {
                        dockerImage.push()
                    }
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                kubernetesDeploy configs: 'deployment.yaml',
kubeconfigId: 'kubeconfig-id'
            }
        }
    }
}
```

Step 2: Configure Jenkins

- **Install Plugins**
- Ensure you have the following Jenkins plugins installed:
- Docker Pipeline
- Kubernetes Continuous Deploy

Create Credentials

- **Docker Hub**: Add your Docker Hub credentials with the ID `dockerhub-credentials`.

- **Kubernetes**: Add your Kubernetes config file with the ID `kubeconfig-id`

Create a New Pipeline Job

- **Job Name**:

`sample-project-ci-cd`

- **Pipeline Script from SCM**:

Point to your GitHub repository containing the `Jenkinsfile`

Step 3: Run the Pipeline

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/username/sample-project.git
git push -u origin master
```

- **Push the Project to GitHub**
- First, initialize a new Git repository and push the project to GitHub.

Trigger the Pipeline

- Go to Jenkins and build the `sample-project-ci-cd` job. This will clone the repository, build the Docker image, push it to Docker Hub, and deploy it to your Kubernetes cluster.

Step 4: Verify Deployment

Check Kubernetes

Ensure that the pods are running and the service is correctly configured.

```
kubectl get pods
kubectl get services
```

Access the Application

- Use the external IP provided by the LoadBalancer service to access the application in your browser.

Conclusion

- By following these steps, you've successfully set up a CI/CD pipeline for a sample Flask application using Jenkins, Docker, and Kubernetes. This pipeline automates the process of building, testing, and deploying your application, making your workflow more efficient and reliable.

- Feel free to customize and expand this project to suit your needs. Happy coding!

Connect With Me

- LinkedIn: www.linkedin.com/in/mohammedtalhakalimi
- Twitter: https://x.com/KalimiTalha97
- Website: https://kalimitalha8.wixsite.com/my-site-3
- GitHub: https://github.com/TalhaKalimi
- Blog: https://medium.com/@kalimitalha8