Basic End-to-End CI/CD Pipeline for a Microservices-Based Application

Building a fully automated CI/CD pipeline for a microservices-based application hosted on a cloud platform (AWS, Azure, or GCP). Using popular DevOps tools and practices to ensure efficient builds, testing, and deployments.

Tech Stack:

- **Version Control:** Git and GitHub/GitLab
- **Build Tools:** Maven/Gradle
- **CI/CD Platform:** Jenkins, GitHub Actions, GitLab CI, or CircleCI
- **Containerization:** Docker
- **Orchestration:** Kubernetes (hosted on EKS, AKS, or GKE)
- **Infrastructure as Code:** Terraform or Ansible
- **Monitoring:** Prometheus and Grafana
- **Cloud:** AWS (or your preferred provider)

Steps of Implementation:

Codebase: Creating a basic microservices application using your preferred language (e.g., Node.js, Java, Python) and, structuring it into separate repositories for each microservice.

- **Create the Simple Web App:** Simple HTML file with the content: **"Hi there! This is Aman, your newbie DevOps friend"**

Steps:

Log in to EC2 Instance:

Use SSH to connect to your EC2 instance.

```
ssh -i your-key.pem ubuntu@<your-ec2-public-ip>
```

Create the HTML File:

1. Created a directory for my web app:

   ```
   mkdir webapp && cd webapp
   ```
   ○
2. Created an `index.html` file (using vim/nano editor):

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Welcome</title>

</head>

<body>

    <h1>Hi there! This is Aman, your newbie DevOps friend</h1>

</body>

</html>
```

Verify the Web App Locally:

Started this simple web server using Python to serve the HTML file:

```
python3 -m http.server 8081
```

Visit `http://54.175.46.115:8081` in browser to see the message.

- ## Create the Microservices Application Structure

I organize the application into multiple microservices.

Plan:

Create a simple structure with two microservices:

1. **Frontend Service:** Displays the HTML page.
2. **Backend Service:** Returns a message like **"Welcome to my node.js application!"**.

Creating the Backend Service

I used **Node.js** for the backend service.

```
cd ~/webapp

mkdir backend

cd backend
```

Initializing a Node.js Project:

```
sudo apt install -y nodejs npm

npm init -y
```

○

Install the Required Packages: Installed the Express framework for creating a backend server:
`npm install express`

Create the Backend Code:

I created a file named `app.js` using vim/nano with following code I found on internet:

```
const express = require('express');

const app = express();

const PORT = 5000;
```

```
app.get('/', (req, res) => {

    res.json({ message: "Hello from the Backend!" });

});


app.listen(PORT, () => {

    console.log(`Backend service running on
http://localhost:${PORT}`);

});
```

Running the Backend Service:

```
node app.js
```

Tested it by visiting `http://54.175.46.115:5000` in my browser. You should see:

```
Welcome to My Node.js App!

This app is running on port 5000.

Try accessing <code>/api/message</code> for a sample API response.
```

○

Connecting Frontend to Backend

Modify the `index.html` file in the **frontend** service to call the backend.

Go to the `index.html` file and add a link:

```
<p>Backend Message: <a href="http://54.175.46.115:5000">Click here to
see backend message</a></p>
```

○

Restart the Python server to serve the updated frontend:

```
python3 -m http.server 8081
```

Structure the Repositories

Since each microservices ideally have their own repositories so we followed that rule.

Organize the Code:

Created a directory for the frontend and moved `index.html` into it:

```
mkdir ~/frontend

mv ~/my-simple-webapp/index.html ~/frontend/

cd ~/frontend

git init

git add .

git commit -m "Initial commit for frontend service"

git remote add origin https://github.com/amankc-neo/frontend-service.git

git push origin master
```

Backend Repository:

Move the backend code to its directory:

```
cd ~/webapp/backend

git init

git add .
git commit -m "Initial commit for backend service"

git remote add origin https://github.com/amankc-neo/backend-service.git

git push -u origin master
```

Push both repositories to GitHub, Create separate GitHub repositories for `frontend` and `backend`.

Containerization: Wrote Dockerfiles for each service and containerized them.

Dockerfile for frontend service:

```
# Use an official Python image as the base

FROM python:3.10-slim

# Set the working directory

WORKDIR /app

# Copy the HTML file to the container

COPY index.html /app/index.html

# Expose port 8081

EXPOSE 8081

# Command to start the Python HTTP server

CMD ["python3", "-m", "http.server", "8080"]
```

- cd ~/frontend

```
docker build -t aman2568/frontend-service:latest .
```

Prepared the image for the frontend service via Dockerfile.

Dockerfile for backend service:

```
# Use the official Node.js image as the base

FROM node:16

# Set the working directory

WORKDIR /app

# Copy package.json and package-lock.json first

COPY package*.json ./

# Install dependencies

RUN npm install

# Copy the application code

COPY . .

# Expose port 5000

EXPOSE 5000

# Command to run the application

CMD ["node", "app.js"]
```

- cd /webapp/backend

```
docker build -t aman2568/backend-service:latest .
```

Prepared the image for the backend service via Dockerfile.

Pushing both images to DockerHub:

```
docker login -u aman2568

docker push aman2568/frontend-service:latest

docker push aman2568/backend-service:latest
```

Continuous Integration Pipeline: Set up a CI pipeline to automate code building, and unit testing. Pipeline gets triggered on every commit to the master branch.

Github Actions Pipeline for Backend service:

```
name: Backend CI
On:
 push:
    branches:
      - master
  pull_request:

jobs:
  lint-test-build:
    runs-on: ubuntu-latest

    steps:
      # Checkout the code
      - name: Checkout Code
        uses: actions/checkout@v3

      # Set up Node.js
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 16
```

```
# Install dependencies
      - name: Install Dependencies
        run: npm install

      # Build Docker image
      - name: Build Docker Image
        run: docker build -t backend-service:ci .
```

Github Actions Pipeline for Frontend Service:

```
name: Frontend CI

on:
  push:
    branches:
      - master
  pull_request:

jobs:
  lint-test-build:
    runs-on: ubuntu-latest

    steps:
      # Checkout the code
      - name: Checkout Code
        uses: actions/checkout@v3
      # Set up Python
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: 3.10

      # Build Docker image
      - name: Build Docker Image
        run: docker build -t frontend-service:ci .
```

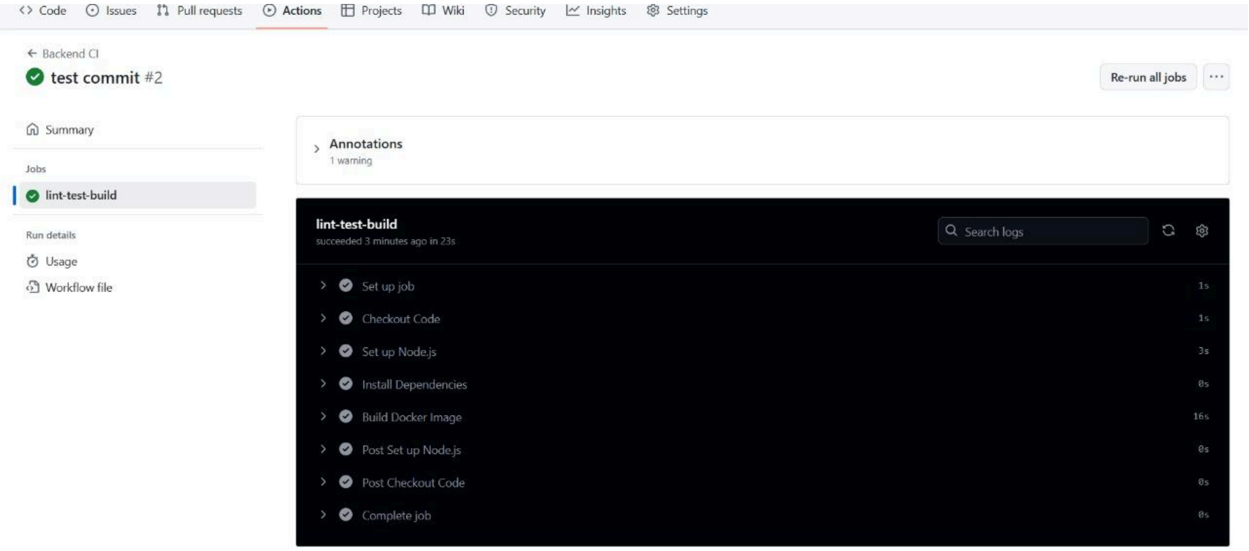
Pushing changes to GitHub:

```
cd /webapp/backend
git add .github/workflows/continuousIntegration.yml
git commit -m "Adding CI workflow for backend"
git push -u origin master

cd /frontend
git add .github/workflows/continuousIntegration.yml
git commit -m "Adding CI workflow for frontend"
git push -u origin master
```

Testing if the trigger automates workflows in Github Actions:

```
echo "// Test Commit" >> ~/webapp/backend/app.js
git add app.js
git commit -m "Test CI trigger"
git push -u origin master
```



Backend CI

test commit #2

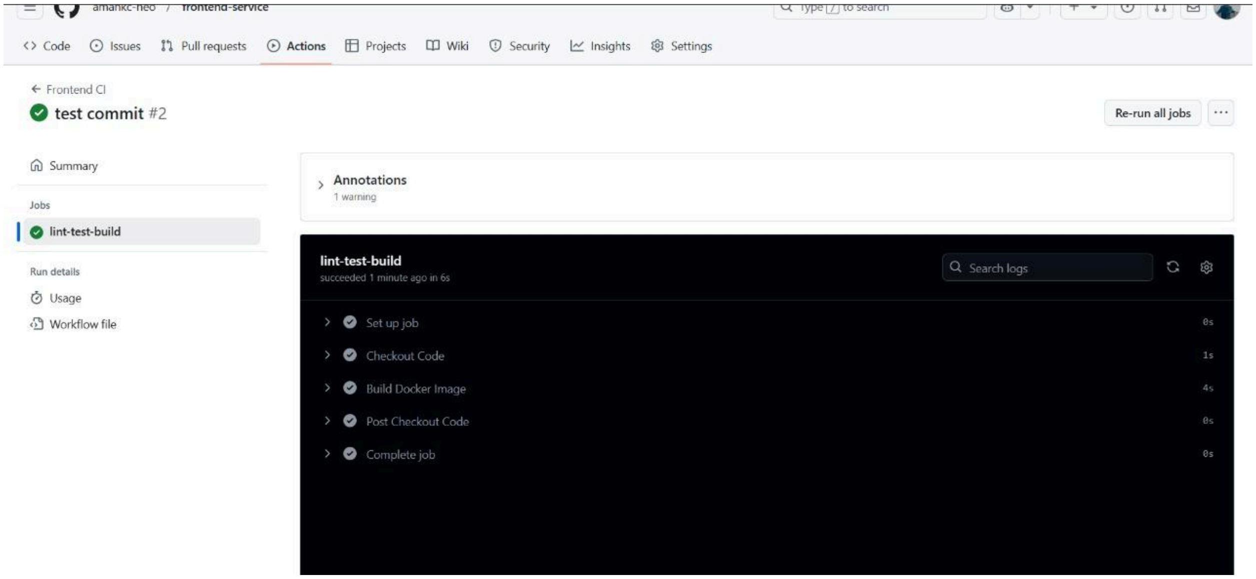
Annotations
1 warning

Jobs
lint-test-build

Run details

lint-test-build
succeeded 3 minutes ago in 23s

```
Set up job              1s
Checkout Code           1s
Set up Node.js          3s
Install Dependencies    0s
Build Docker Image      16s
Post Set up Node.js     0s
Post Checkout Code      0s
Complete job            0s
```



Frontend CI

test commit #2

Annotations
1 warning

Jobs
lint-test-build

Run details

lint-test-build
succeeded 1 minute ago in 6s

```
Set up job              0s
Checkout Code           1s
Build Docker Image      4s
Post Checkout Code      0s
Complete job            0s
```

Continuous Delivery Pipeline: Deploy the application on Kubernetes using Helm charts. Configure the pipeline for rolling updates or blue-green deployments.

Minikube (Local Development): Install Minikube

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Start Minikube: You can adjust cpus and memory according to your requirements but Kubernetes will need at least 2 cpus to run.

```
minikube start --cpus=2 --memory=4g
```

```
kubectl get nodes
```

Install Helm: Helm is a package manager for Kubernetes. It simplifies deploying applications to Kubernetes.

```
curl
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
bash
```

```
helm version
```

```
helm repo add stable https://charts.helm.sh/stable

helm repo update
```

Install a Test Chart (Optional): To verify if Helm is working:

```
helm install my-nginx stable/nginx-ingress

kubectl get pods
```

Configure `kubectl` to Connect to the Cluster: Install `kubectl` to manage Kubernetes resources:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/  &&  kubectl version --client
```

Configure Access:

For AWS EKS: Generate a kubeconfig file:

```
aws eks update-kubeconfig --region us-west-2 --name devops-cluster
```

For Minikube: Use Minikube's default kubeconfig:

```
kubectl config use-context minikube
```

Verify cluster access:

```
kubectl get nodes
```

Push Docker Images to a Container Registry

Our application images must be pushed to a container registry accessible by Kubernetes.

Log in to Docker Hub

Install Docker CLI (if not already installed):

```
sudo apt update

sudo apt install docker.io

sudo systemctl start docker

sudo systemctl enable docker


docker login
```

Tag and Push Docker Images

For the backend and frontend Docker images, push them to Docker Hub.

```
docker tag backend-service:latest aman2568/backend-service:latest

docker tag frontend-service:latest aman2568/frontend-service:latest
```

Push the Images:

```
docker push aman2568/backend-service:latest

docker push aman2568/frontend-service:latest
```

Verify Kubernetes Cluster and Image Accessibility

Create a basic deployment using one of your Docker images to verify connectivity:

```
kubectl create deployment test-backend
--image=aman2568/backend-service:latest
```

Verify that the Pod is running:

```
kubectl get pods

kubectl describe pod <pod-name>
```

Delete the test deployment:

```
kubectl delete deployment test-backend
```

1. **Monitoring and Logging:**Deploy Prometheus and Grafana to monitor application performance. Use a logging tool (e.g., ELK Stack or Fluentd) to aggregate logs.
2. **IaC:**Write Terraform scripts to provision Kubernetes clusters and cloud resources.
3. **Security:** Integrate security scanning tools like SonarQube or Trivy to detect vulnerabilities.