


# DevOps Lab 1: Automated Web Application Deployment Pipeline

A complete CI/CD pipeline for deploying a containerized web application to AWS using modern DevOps tools and practices.

Status:  Active | Deployment: Automated | Infrastructure: AWS Singapore

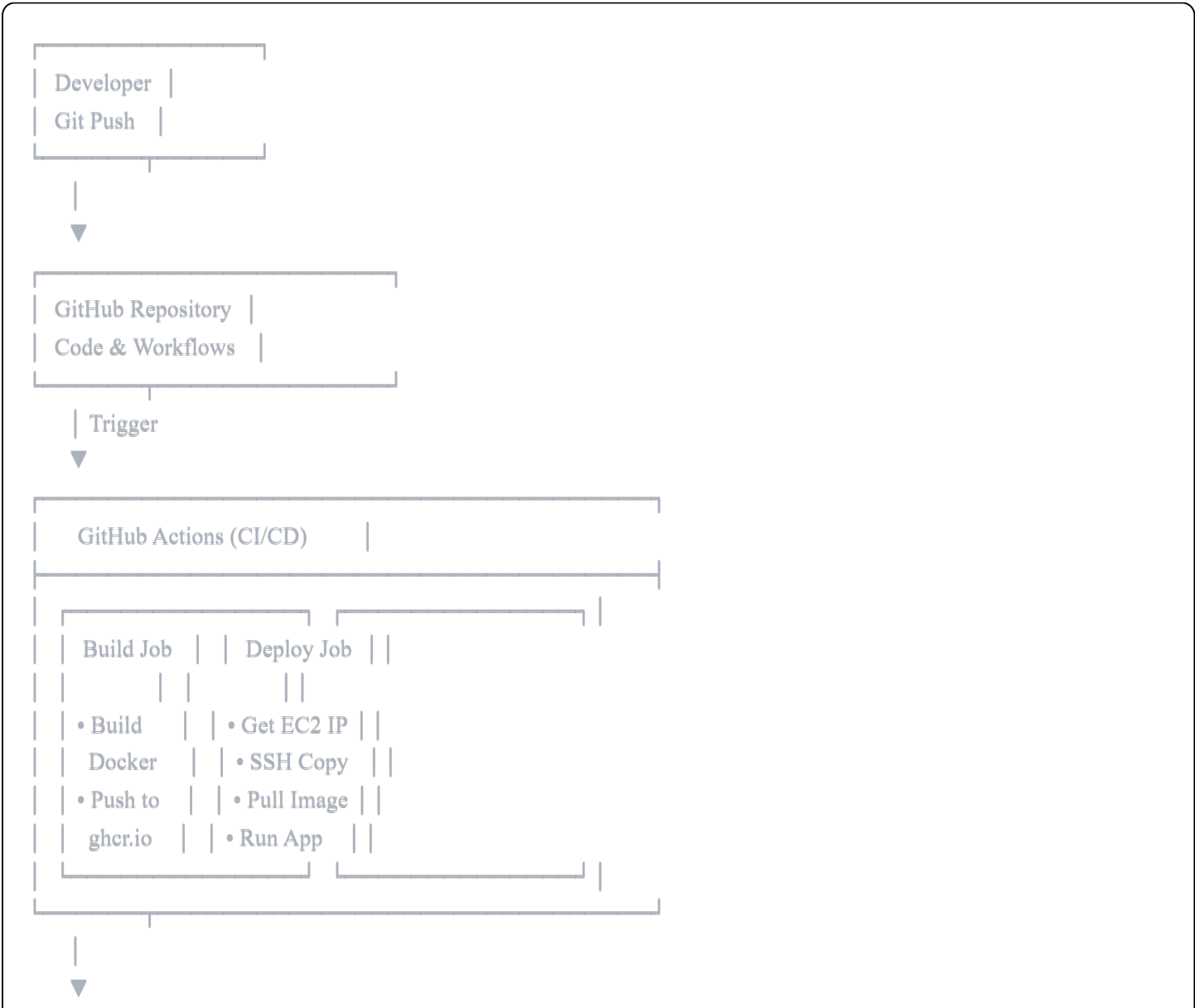
## Project Overview

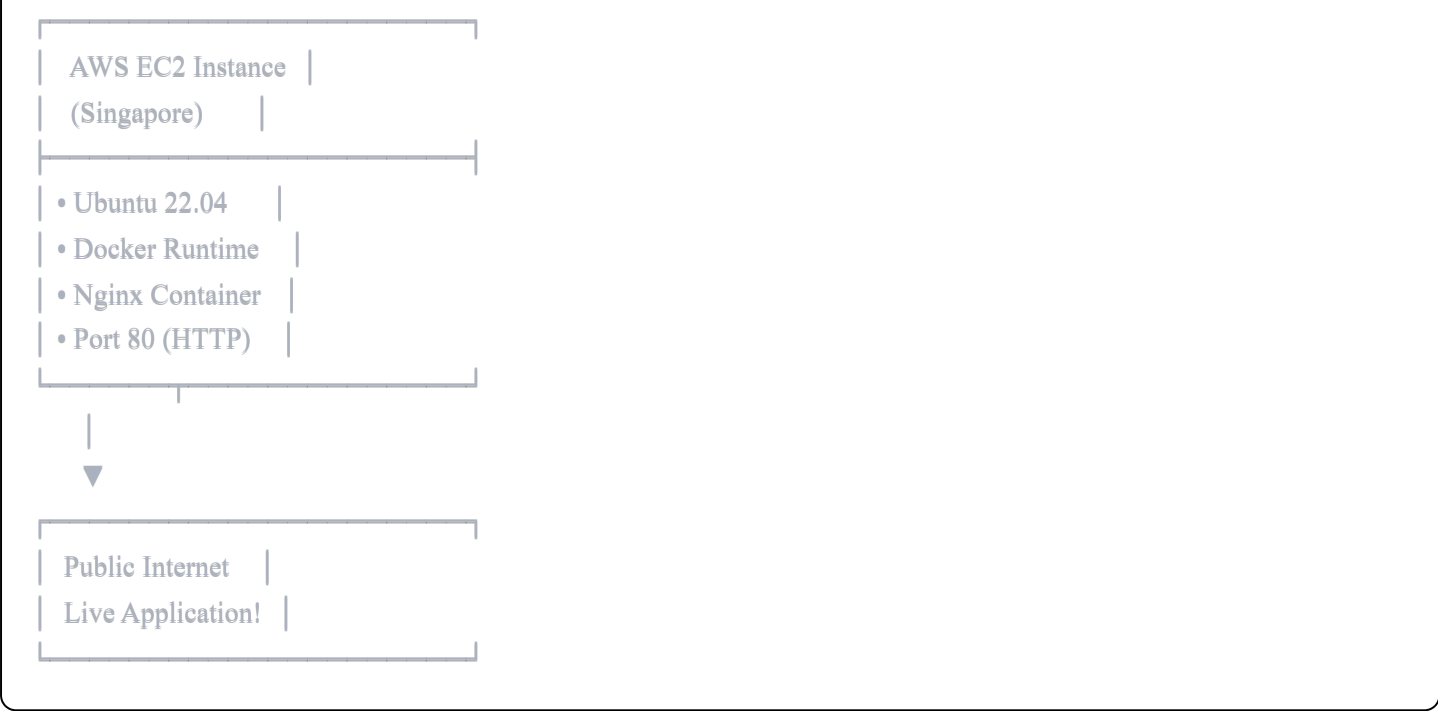
This project demonstrates a production-ready DevOps workflow that automates the entire deployment process from code commit to live application. Every push to the main branch automatically builds a Docker image, pushes it to GitHub Container Registry, and deploys it to an AWS EC2 instance.

### Live Demo

 Application URL: <http://3.1.251.177/>

## Architecture





## 🔧 Technologies Used

Technology	Purpose	Version
Docker	Application Containerization	Latest
GitHub Actions	CI/CD Pipeline Automation	-
Terraform	Infrastructure as Code	1.6+
AWS EC2	Cloud Compute Platform	t3.micro
Bash	Deployment Automation	-
Nginx	Web Server	Alpine
Git	Version Control	-

## 📁 Project Structure

```
devops-lab1/
├── .github/
│   └── workflows/
│       └── deploy.yml    # GitHub Actions CI/CD pipeline
├── app/
│   └── index.html        # Web application
```

```
├── scripts/
│   ├── build-and-test.sh    # Local Docker build & test
│   └── deploy.sh           # EC2 deployment script
├── terraform/
│   ├── main.tf             # AWS infrastructure definition
│   ├── variables.tf        # Terraform variables
│   ├── key.tf              # SSH key pair configuration
│   └── terraform.tfvars    # Variable values (gitignored)
├── Dockerfile              # Container image definition
├── .dockerignore           # Docker build exclusions
├── .gitignore              # Git exclusions
└── README.md               # This file
```

---

## Features

### CI/CD Pipeline

- ✓ **Automated Builds:** Docker image built on every push
- ✓ **Container Registry:** Images stored in GitHub Container Registry
- ✓ **Automated Deployment:** Zero-touch deployment to AWS
- ✓ **Health Checks:** Automatic verification of deployment success
- ✓ **Fast Pipeline:** Code to production in under 2 minutes

### Infrastructure

- ✓ **Infrastructure as Code:** Entire AWS setup defined in Terraform
- ✓ **Reproducible:** Destroy and recreate infrastructure anytime
- ✓ **Version Controlled:** Infrastructure changes tracked in Git
- ✓ **Secure:** Security groups, SSH keys, and secrets management

### Application

- ✓ **Containerized:** Runs consistently anywhere Docker is available
  - ✓ **Lightweight:** Nginx Alpine base (~5MB)
  - ✓ **Production-Ready:** Health checks and monitoring
  - ✓ **Zero-Downtime:** New containers replace old automatically
-

## Prerequisites

### Required

- GitHub account
- AWS account (free tier eligible)
- Git installed locally
- Docker Desktop (for local testing)
- Terraform CLI (1.6+)

### Recommended

- Basic understanding of Linux commands
  - Familiarity with Docker concepts
  - AWS Console navigation knowledge
- 

## Quick Start

### 1. Clone the Repository

```
bash
git clone https://github.com/kaelcloud/devops-lab1.git
cd devops-lab1
```

### 2. Setup AWS Credentials

Create IAM user with these policies:

- `AmazonEC2FullAccess`
- `AmazonVPCFullAccess`

Generate access keys and add to GitHub repository secrets:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_REGION` (e.g., `ap-southeast-1`)

### 3. Generate SSH Key

```
bash  
ssh-keygen -t rsa -b 4096 -f ~/.ssh/devops-lab1-key -N ""
```

Add private key to GitHub secret as `EC2_SSH_KEY`.

Update `terraform/variables.tf` with your public key.

### 4. Deploy Infrastructure

```
bash  
  
cd terraform  
terraform init  
terraform plan  
terraform apply
```

Save the output IP address.

### 5. Make Repository Public

For free GitHub Container Registry access:

- Settings → Danger Zone → Change repository visibility → Public

### 6. Trigger Deployment

```
bash  
  
git add .  
git commit -m "Initial deployment"  
git push origin main
```

GitHub Actions will automatically build and deploy!

### 7. Access Your Application

Open browser: <http://3.1.251.177/>

---

## How It Works

On Every Git Push:

1. GitHub Actions Triggered

- Workflow file `.github/workflows/deploy.yml` executes

## 2. Build Job

- Checks out code
- Builds Docker image from `Dockerfile`
- Pushes image to `ghcr.io/kaelcloud/devops-lab1:latest`

## 3. Deploy Job

- Queries AWS for EC2 instance IP
- Waits for SSH availability
- Copies `deploy.sh` to EC2 via SCP
- Executes deployment script remotely
- Pulls latest Docker image
- Stops old container
- Starts new container
- Verifies with health check

## 4. Result

- Application updated with new code
- Zero downtime deployment
- Automatic rollback if health check fails

---

## Testing Locally

### Build and Test Docker Image

```
bash
./scripts/build-and-test.sh
```

Access at: `http://localhost:8080`

### Test Terraform Configuration

```
bash
cd terraform
terraform validate
terraform plan
```

## Manual Deployment Test

```
bash

# SSH to EC2
ssh -i ~/.ssh/devops-lab1-key ubuntu@3.1.251.177

# Check running containers
docker ps

# View application logs
docker logs devops-lab1-app

# Test application
curl localhost
```

---

## Configuration

### Change AWS Region

Edit `terraform/variables.tf`:

```
hcl

variable "aws_region" {
  default = "ap-southeast-1" # Change this
}
```

Don't forget to update GitHub secret `AWS_REGION` too!

### Change Instance Type

Edit `terraform/variables.tf`:

```
hcl

variable "instance_type" {
  default = "t3.micro" # Change to t3.small, t3.medium, etc
}
```

**Note:** Only t2.micro and t3.micro are free tier eligible.

### Modify Application

Edit `app/index.html` - changes auto-deploy on push!

---

## Monitoring & Logs

### GitHub Actions Logs

- Repository → Actions tab → Select workflow run
- View detailed logs for each step

### EC2 Application Logs

```
bash

# SSH to instance
ssh -i ~/.ssh/devops-lab1-key ubuntu@3.1.251.177

# View container logs
docker logs devops-lab1-app

# Follow logs in real-time
docker logs -f devops-lab1-app

# View system logs
sudo journalctl -u docker
```

### AWS CloudWatch (Optional)

- EC2 → Instances → Select instance → Monitoring tab
- View CPU, Network, and Disk metrics

---

## Troubleshooting

### Pipeline Fails at Build Stage

Symptoms: Docker build error

Solution:

```
bash
```

```
# Test locally
./scripts/build-and-test.sh

# Check Dockerfile syntax
docker build -t test .
```

## Pipeline Fails at Deploy Stage

**Symptoms:** SSH connection failed

**Solution:**

- Verify `EC2_SSH_KEY` secret is correct
- Check security group allows SSH (port 22)
- Ensure instance is running: `terraform refresh`

## Application Not Accessible

**Symptoms:** Cannot access <http://3.1.251.177>

**Solution:**

```
bash

# Check security group allows HTTP (port 80)
terraform show | grep "from_port.*80"

# SSH and check container
ssh -i ~/.ssh/devops-lab1-key ubuntu@3.1.251.177
docker ps
docker logs devops-lab1-app
```

## Terraform Apply Fails

**Symptoms:** Instance type not supported

**Solution:**

- Change to `t3.micro` in `variables.tf`
- Add `availability_zone = "ap-southeast-1a"` in `main.tf`

---

## Cleanup

To avoid AWS charges, destroy resources when not using:

```
bash
```

```
cd terraform
```

```
terraform destroy
```

Type  to confirm.

This removes:

- EC2 instance
- Elastic IP
- Security group
- SSH key pair

Can recreate anytime with !




---

## Cost Estimation

### Free Tier (First 12 Months)

- ☒ EC2 t3.micro: 750 hours/month (enough for 24/7)
- ☒ 30GB EBS storage
- ☒ 15GB data transfer out

### After Free Tier

-  EC2 t3.micro: ~~\$0.0104/hour~~ (\$7.50/month)
-  EBS 8GB: ~\$0.80/month
-  Elastic IP: Free while attached, \$0.005/hour if unused

**Total:** ~\$8-10/month after free tier expires.

---








## Key Skills Demonstrated

### DevOps Practices






- ☒ Continuous Integration/Continuous Deployment (CI/CD)
- ☒ Infrastructure as Code (IaC)
- ☒ Configuration Management

-  Automated Testing and Deployment

## Technical Competencies

-  Docker containerization and image management
-  AWS cloud platform (EC2, VPC, Security Groups)
-  Terraform for infrastructure provisioning
-  GitHub Actions for pipeline automation
-  Bash scripting for deployment automation
-  SSH key-based authentication
-  Git version control and workflows

## Best Practices

-  Secrets management (never commit credentials)
-  Security groups and firewall configuration
-  Health checks and monitoring
-  Zero-downtime deployments
-  Infrastructure versioning



## Author

### Muhammad Hazran Hafiz Ahmad

- GitHub: [@kaelcloud](#)
- LinkedIn: [Muhammad Hazran Hafiz Ahmad](#)
- Project Repository: [devops-lab1](#)

---

<div align="center">

★ If you found this project helpful, please star the repository! ★

Built with passion for DevOps excellence

[↑ Back to Top](#)

</div>

