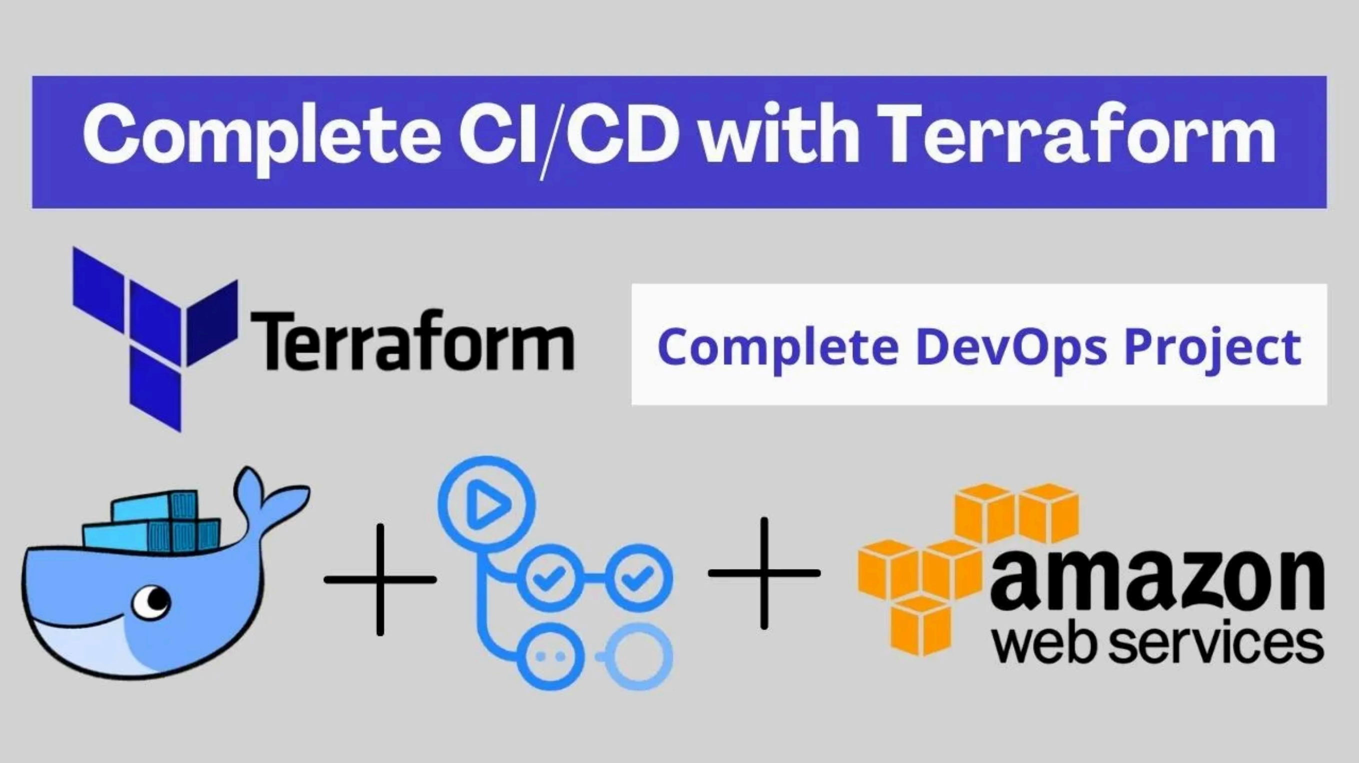
Deploy to AWS EC2 using Terraform and Github Actions CI/CD

Tushar Rajpoot · Follow

Published in FAUN — Developer Community 🐾 · 4 min read · Aug 25, 2022

51



Technologies:

- Terraform

- Github Actions
- Docker
- Node.js
- AWS EC2
- AWS S3
- AWS ECR

Tasks:

- Get access id, secret id from AWS
- Develop a simple nodejs app

```
const express = require("express")
const app = express()
```

```
app.get("/",(req,res)=>{
    res.send("Service is up and running")
})

app.listen(8080,()=>{
    console.log("Server is up")
})
```

- Write Dockerfile for Simple Application

```
FROM node:14
WORKDIR /user/app
COPY package.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD ["npm","start"]
```

- Generate SSH keys for connecting to the EC2 instance
- Create an S3 bucket for storing Terraform State file

Write Terraform Scripts for provisioning EC2 instance

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>4.0"
    }
  }
  backend "s3" {
    key = "aws/ec2-deploy/terraform.tfstate"
  }
}

provider "aws" {
  region = var.region
}
```

```
resource "aws_instance" "servernode" {
  ami                    = "ami-052efd3df9dad4825"
  instance_type          = "t2.micro"
  key_name               = aws_key_pair.deployer.key_name
  vpc_security_group_ids = [aws_security_group.maingroup.id]
  iam_instance_profile   = aws_iam_instance_profile.ec2-profile.name
  connection {
    type        = "ssh"
    host        = self.public_ip
    user        = "ubuntu"
    private_key = var.private_key
    timeout     = "4m"
  }
  tags = {
    "name" = "DeployVM"
  }
}
resource "aws_iam_instance_profile" "ec2-profile" {
  name = "ec2-profile"
  role = "ECR-LOGIN-AUTO"
}
resource "aws_security_group" "maingroup" {
  egress = [
    {
      cidr_blocks      = ["0.0.0.0/0"]
      description      = ""
      from_port        = 0
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "-1"
      security_groups  = []
      self             = false
      to_port          = 0
    }
  ]
  ingress = [
    {
      cidr_blocks      = ["0.0.0.0/0", ]
      description      = ""
      from_port        = 22
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "tcp"
      security_groups  = []
      self             = false
      to_port          = 22
    },
    {
      cidr_blocks      = ["0.0.0.0/0", ]
      description      = ""
      from_port        = 80
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "tcp"
      security_groups  = []
      self             = false
      to_port          = 80
    }
  ]

}

resource "aws_key_pair" "deployer" {
  key_name   = var.key_name
  public_key = var.public_key
}

output "instance_public_ip" {
  value     = aws_instance.servernode.public_ip
  sensitive = true
}
```

- variables.tf file

```
variable "region" {
    default = "us-east-1"
}
variable "public_key" {

}
variable "private_key" {

}
variable "key_name" {

}
```

Write CI/CD pipeline

- Write Github Actions workflow: Set environment variables

```
env:
  AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
```

```
 AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
 TF_STATE_BUCKET_NAME: ${{ secrets.AWS_TF_STATE_BUCKET_NAME }}
 PRIVATE_SSH_KEY: ${{ secrets.AWS_SSH_KEY_PRIVATE }}
 PUBLIC_SSH_KEY: ${{ secrets.AWS_SSH_KEY_PUBLIC }}
 AWS_REGION: us-east-1
```

- Setup backend for S3 bucket with terraform init

```
- name: checkout repo
   uses: actions/checkout@v2
  - name: setup terraform
   uses: hashicorp/setup-terraform@v1
   with:
    terraform_wrapper: false
  - name: Terraform Init
   id: init
   run: terraform init -backend-
config="bucket=$TF_STATE_BUCKET_NAME" -backend-config="region=us-
east-1"
   working-directory: ./terraform
```

- Pass tf variables with Terraform plan

```
- name: Terraform Plan
 id: plan
 run: |-
  terraform plan \
  -var="region=us-east-1" \
  -var="bucket=$TF_STATE_BUCKET_NAME" \
  -var="public_key=$PUBLIC_SSH_KEY" \
  -var="private_key=$PRIVATE_SSH_KEY" \
  -var="key_name=deployer-key" \
  -out=PLAN
 working-directory: ./terraform
```

- Run terraform apply

```
- name: Terraform Apply
    id: apply
    run: |-
      terraform apply PLAN
    working-directory: ./terraform
```

- Set EC2 instance public ip as job output

```
- name: Set output
    id: set-dns
    run: |-
        echo "::set-output name=instance_public_dns::$(terraform
output instance_public_ip)"
    working-directory: ./terraform
```

- Authenticate ECR

```
- name: Login to AWS ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v1
```

- Set ec2 public ip as environment variable for later use

```
- run: echo SERVER_PUBLIC_IP=${{ needs.deploy-
infra.outputs.SERVER_PUBLIC_DNS }} >> $GITHUB_ENV
```

- Build, tag and push docker image to Amazon ECR

```
- name: Build, tag, and push docker image to Amazon ECR
    env:
      REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      REPOSITORY: example-node-app
      IMAGE_TAG: ${{ github.sha }}
    run: |
      docker build -t $REGISTRY/$REPOSITORY:$IMAGE_TAG .
      docker push $REGISTRY/$REPOSITORY:$IMAGE_TAG
    working-directory: ./nodeapp
```

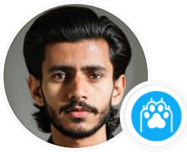
- Connect to EC2 using ssh and deploy docker container

```
- name: Deploy Docker Image to EC2
  env:
    REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    REPOSITORY: example-node-app
    IMAGE_TAG: ${{ github.sha }}
    AWS_DEFAULT_REGION: us-east-1
  uses: appleboy/ssh-action@master
  with:
    host: ${{ env.SERVER_PUBLIC_IP }}
    username: ubuntu
    key: ${{ env.PRIVATE_SSH_KEY }}
    envs:
PRIVATE_SSH_KEY,REGISTRY,REPOSITORY,IMAGE_TAG,AWS_ACCESS_KEY_ID,AWS_S
ECRET_ACCESS_KEY,AWS_REGION
    script: |-
      sudo apt update
      sudo apt install docker.io -y
      sudo apt install awscli -y
      sudo $(aws ecr get-login --no-include-email --region us-east-
1);
      sudo docker stop myappcontainer || true
      sudo docker rm myappcontainer || true
      sudo docker pull $REGISTRY/$REPOSITORY:$IMAGE_TAG
      sudo docker run -d --name myappcontainer -p 80:8080
$REGISTRY/$REPOSITORY:$IMAGE_TAG
```

If you find this helpful, please click the clap 👏 button below a few times to show your support for the author 👇

🚀 Join FAUN & get similar stories in your inbox each week

Terraform Github Actions Cicd DevOps AWS



Written by Tushar Rajpoot

Follow

151 Followers · Writer for FAUN — Developer Community 🐾

DevOps Engineer | Backend Developer

More from Tushar Rajpoot and FAUN — Developer Community 🐾

— Open Source

— automates — container deployment

— container scaling

— load balancing

— All top cloud providers are supported.

Platforms — Kubernetes playground

— Play with K8s.

Cloud Based :— GKE - Google Kubernetes Engine .

Tushar Rajpo... in FAUN — Developer Community...

Kubernetes Introduction

Kubernetes is an open-source container management tool that automates container...

6 min read · Sep 6, 2022

3 1

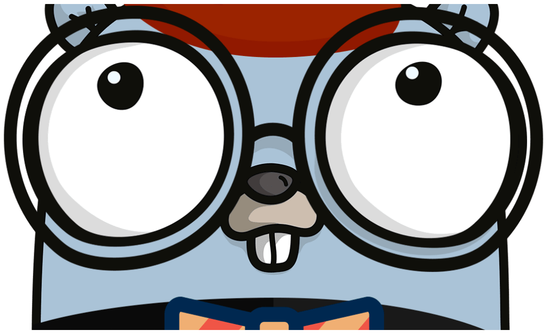
Prinux in FAUN — Developer Community 🐾

An App Capable Of Putting Postman To Trash

It's been more than a decade since the initial release of Postman, since then it did face...

3 min read · Jan 16, 2024

137



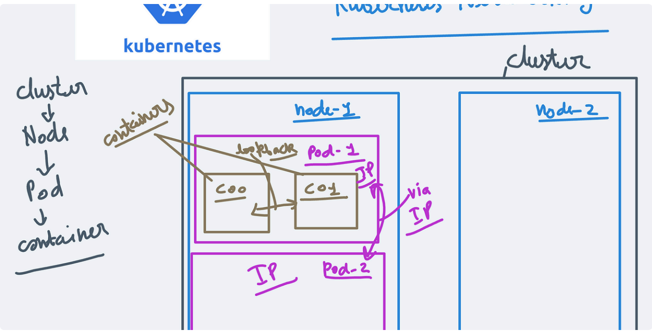
Janteshital in FAUN — Developer Community 🐾

Understanding go.mod and go.sum

Getting started with the modules in GoLang !!

4 min read · Jun 23, 2021

775 4



Tushar Rajpoot

Kubernetes Networking Visualized

Kubernetes Networking addresses four concerns:

2 min read · Sep 16, 2022

187 1

See all from Tushar Rajpoot

See all from FAUN — Developer Community 🐾

Recommended from Medium

Chloe McAree

Deploying Terraform using GitHub actions

When working with AWS services I have been using Terraform as my Infra As Code. In the...

5 min read · Oct 29, 2023

38

Dulanjana Lakmal

Automating Website Deployment to AWS EC2 Using GitHub Actions

GitHub Actions is a powerful automation tool that allows you to build, test, and deploy you...

3 min read · Dec 9, 2023

14

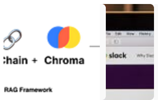
Lists

General Coding Knowledge

20 stories · 1076 saves

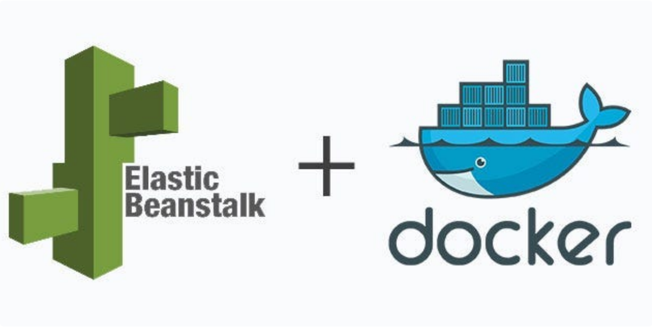
Natural Language Processing

1349 stories · 828 saves



Productivity

238 stories · 376 saves



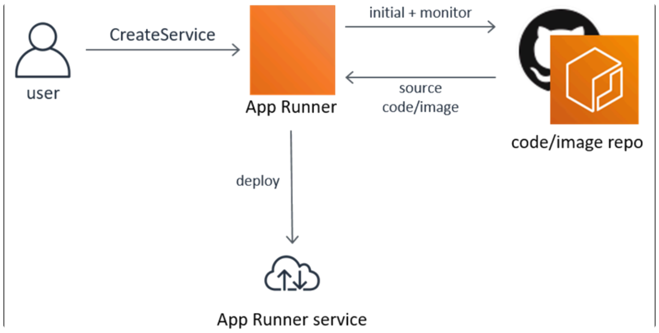
Neel Shah in AWS in Plain English

Deploying a Web App on AWS S3 Using GitHub Actions ☁️🐱🏍️

What is GitHub Actions?

3 min read · Oct 8, 2023

80 1



Steven Accardo

How to Deploy a Node.js Application to AWS Using Docker,...

In this article, I'll be walking you through how to create a simple Node.js and Express.js...

14 min read · Oct 30, 2023

61 1

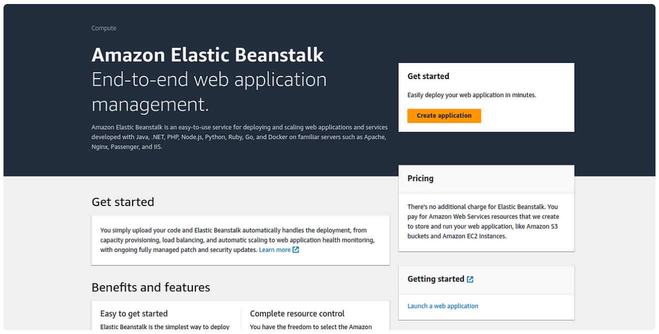
Ben Tran

Deploy Dockerized Application To AWS Elastic Beanstalk

In this tutorial, I will provide a step-by-step walkthrough on deploying a dockerized...

9 min read · Nov 1, 2023

142 1



Compute

Amazon Elastic Beanstalk

End-to-end web application management.

Amazon Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

Get started

Easily deploy your web application in minutes.

Create application

Get started

You simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and automatic scaling to web application health monitoring, with ongoing fully managed patch and security updates. Learn more

Pricing

There's no additional charge for Elastic Beanstalk. You pay for Amazon Web Services resources that we create to store and run your web application, like Amazon S3 buckets and Amazon EC2 instances.

Benefits and features

Easy to get started

Elastic Beanstalk is the simplest way to deploy

Complete resource control

You have the freedom to select the Amazon

Getting started

Launch a web application

Juan P. Lima

Deploy your NestJS API on AWS Elastic Beanstalk

Have you ever wondered how you can deploy your NestJS API on AWS Elastic Beanstalk?...

5 min read · Jan 3, 2024

1

See more recommendations