**Job Ready 4 projects**

**For**

**AWS DevOps**

**Project 1: Automate EC2, Auto-Scaling, Load Balancer, and DNS Setup Using AWS CLI and Terraform**

**Project 2: Automate EC2 and Auto-Scaling with AWS Elastic Load Balancer (ELB) and Route 53 using AWS CodePipeline**

**Project 3: Deploy a Dockerized Application to AWS EKS (Elastic Kubernetes Service)**

**Project 4: Automate EKS Deployment with AWS CodePipeline**

Using AWS, leveraging EC2, Auto Scaling, Elastic Load Balancing, Route 53, EKS, Amazon ECR, and AWS CodePipeline for CI/CD automation.

**Details:**

**Project 1: Automate EC2, Auto-Scaling, Load Balancer, and DNS Setup Using AWS CLI and Terraform**

1. Install Terraform and AWS CLI

**sudo apt-get update && sudo apt-get install -y terraform awscli**

2. Configure AWS CLI

**aws configure**

3. Define Terraform Configuration

vim **main.tf**`

**provider "aws" {**

**region = "us-east-1"**

**}**

**resource "aws\_instance" "**my\_instance**" {**

**ami = "ami-12345678"** # Use the appropriate AMI ID

**instance\_type = "t2.micro"**

**key\_name = "MyKeyPair"**

**}**

**resource "aws\_autoscaling\_group" "asg" {**

**desired\_capacity = 2**

**max\_size = 10**

**min\_size = 1**

**vpc\_zone\_identifier = ["subnet-123456"]** # Replace with your subnet ID

**launch\_configuration = aws\_launch\_configuration.lc.id**

**}**

**resource "aws\_launch\_configuration" "lc" {**

**image\_id = "ami-12345678"**

**instance\_type = "t2.micro"**

**key\_name = "MyKeyPair"**

**}**

**resource "aws\_lb" "my\_lb" {**

**name = "my-load-balancer"**

**internal = false**

**load\_balancer\_type = "application"**

**subnets = ["subnet-123456", "subnet-654321"]**

**}**

**resource "aws\_route53\_record" "dns" {**

**zone\_id = "Z2EXAMPLE"** # Replace with your Route 53 hosted zone ID

**name = "**example.com**"**

**type = "A"**

**alias {**

**name = aws\_lb.my\_lb.dns\_name**

**zone\_id = aws\_lb.my\_lb.zone\_id**

**evaluate\_target\_health = false**

**}**

**}**

4. Deploy Infrastructure with Terraform

**terraform init**

**terraform apply**

**Project 2: Automate EC2 and Auto-Scaling with AWS Elastic Load Balancer (ELB) and Route 53 using AWS CodePipeline**

1. Set up Auto Scaling and Load Balancer Using Terraform

**Continue** from Project 1 to deploy an Auto Scaling Group and Elastic Load Balancer using Terraform.

2. Create an S3 Bucket for Application **Code Deployment**

**aws s3 mb** **s3://**my-app-bucket

3. Create a CodePipeline to Automate EC2 and ELB Setup

Set up a pipeline in AWS CodePipeline to deploy the application to EC2 instances when the code is pushed to an S3 bucket.

Use the following AWS **CLI** commands to create the pipeline:

**aws codepipeline create-pipeline --pipeline-name** MyPipeline --role-arn arn:aws:iam::123456789012:role/AWS-CodePipeline-Service --cli-input-json file://pipeline.json

Example `**pipeline.json**`:

**{**

**"pipeline": {**

**"name": "**MyPipeline**",**

**"roleArn": "arn:aws:iam::123456789012:role/AWS-CodePipeline-Service",**

**"artifactStore": {**

**"type": "S3",**

**"location": "my-app-bucket"**

**},**

**"stages": [**

**{**

**"name": "Source",**

**"actions": [**

**{**

**"name": "Source",**

**"actionTypeId": {**

**"category": "Source",**

**"owner": "AWS",**

**"provider": "S3",**

**"version": "1"**

**},**

**"outputArtifacts": [**

**{**

**"name": "SourceArtifact"**

**}**

**],**

**"configuration": {**

**"S3Bucket": "my-app-bucket",**

**"S3ObjectKey": "source.zip"**

**},**

**"runOrder": 1**

**}**

**]**

**},**

**{**

**"name": "Deploy",**

**"actions": [**

**{**

**"name": "DeployToEC2",**

**"actionTypeId": {**

**"category": "Deploy",**

**"owner": "AWS",**

**"provider": "CodeDeploy",**

**"version": "1"**

**},**

**"inputArtifacts": [**

**{**

**"name": "SourceArtifact"**

**}**

**],**

**"configuration": {**

**"ApplicationName": "MyApp",**

**"DeploymentGroupName": "MyAppDeploymentGroup"**

**},**

**"runOrder": 1**

**}**

**]**

**}**

**]**

**}**

**}**

**Project 3: Deploy a Dockerized Application to AWS EKS (Elastic Kubernetes Service)**

1. Set Up an Amazon ECR Repository

- Create an ECR repository to store your Docker images:

**aws ecr create-repository --repository-name** my-app-repo

2. Build and Push Docker Image to ECR

- Build and tag your Docker image:

**docker build -t** my-app **.**

**docker tag** my-app:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app-repo:latest

Push the image to the ECR repository:

**aws ecr get-login-password --region us-east-1 | docker login --username** AWS --password-stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com

**docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com**/my-app-repo:latest

3. Create an EKS Cluster

- Use `eksctl` to create an EKS cluster:

**eksctl create cluster** --name my-cluster --region us-east-1 --**nodes 3**

4. Deploy Application to EKS

- Create a Kubernetes deployment YAML file (`deployment.yaml`):

**vim** deployment.yaml

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: my-app-deployment**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: my-app**

**template:**

**metadata:**

**labels:**

**app: my-app**

**spec:**

**containers:**

**- name: my-app**

**image: 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app-repo:latest**

**ports:**

**- containerPort: 80**

- Apply the deployment to the EKS cluster:

**kubectl apply -f** deployment.yaml

5. Expose the Application with a Load Balancer

- Create a service to expose the deployment:

**vim** service.yaml

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: my-app-service**

**spec:**

**type: LoadBalancer**

**ports:**

**- port: 80**

**targetPort: 80**

**selector:**

**app: my-app**

- Apply the service:

**kubectl apply -f** service.yaml

6. Access the Application

- Get the external load balancer's DNS name:

**kubectl get services**

**Project 4: Automate EKS Deployment with AWS CodePipeline**

1. Create an S3 Bucket for Pipeline Artifacts

**aws s3 mb s3://**my-eks-pipeline-artifacts

2. Set Up a CodePipeline to Deploy the Application to EKS

Create a pipeline using AWS CodePipeline to automate the deployment of Docker images from ECR to the EKS cluster.

Use AWS CodeBuild to build the Docker image and push it to ECR, and use AWS CodePipeline to trigger the EKS deployment.

CodePipeline JSON:

**{**

**"pipeline": {**

**"name": "EKSDeploymentPipeline",**

**"roleArn": "arn:aws:iam::123456789012:role/AWS-CodePipeline-Service",**

**"artifactStore": {**

**"type": "S3",**

**"location": "my-eks-pipeline-artifacts"**

**},**

**"stages": [**

**{**

**"name": "Source",**

**"actions": [**

**{**

**"name": "SourceAction",**

**"actionTypeId": {**

**"category": "Source",**

**"owner": "AWS",**

**"provider": "S3",**

**"version": "1"**

**},**

**"outputArtifacts": [**

**{**

**"name": "SourceArtifact"**

**}**

**],**

**"configuration": {**

**"S3Bucket": "my-eks-source-bucket",**

**"S3ObjectKey": "source.zip"**

**},**

**"runOrder": 1**

**}**

**]**

**},**

**{**

**"name": "Build",**

**"actions": [**

**{**

**"name": "BuildAction",**

**"actionTypeId": {**

**"category": "Build",**

**"owner": "AWS",**

**"provider": "CodeBuild",**

**"version": "1"**

**},**

**"inputArtifacts": [**

**{**

**"name": "SourceArtifact"**

**}**

**],**

**"configuration": {**

**"ProjectName": "MyEKSBuildProject"**

**},**

**"runOrder": 1**

**}**

**]**

**},**

**{**

**"name": "Deploy",**

**"actions": [**

**{**

**"name": "DeployAction",**

**"actionTypeId": {**

**"category": "Deploy",**

**"owner": "AWS",**

**"provider": "ECS",**

**"version": "1"**

**},**

**"inputArtifacts": [**

**{**

**"name": "BuildArtifact"**

**}**

**],**

**"configuration": {**

**"ClusterName": "MyEKSCluster",**

**"ServiceName": "MyAppService"**

**},**

**"runOrder": 1**

**}**

**]**

**}**

**]**

**}**

**}**

This comprehensive AWS-based approach covers **automation** of EC2, Auto Scaling, Load Balancer setup, Kubernetes (EKS), and CI/CD with AWS CodePipeline.