`TERRAFORM INFRASTRUCTURE` :

Terraform Scripts Documentation

Overview

This Terraform configuration sets up an Amazon EKS (Elastic Kubernetes Service) cluster with supporting infrastructure in AWS. The deployment is designed to ensure high availability, scalability, and security. Below is an overview of the key components:

1. Prerequisites

Before running the scripts, ensure you have the following:

- **Terraform Installed**: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
- **AWS CLI Installed**: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
- *AWS Access Credentials:**
  - Configure the AWS CLI with appropriate credentials using the below command:
    - **aws configure**
- **Docker Installed**: Required for building the Docker image.
- **Kubernetes Tools**: Install kubectl - https://kubernetes.io/docs/tasks/tools/

2. Files Overview

1. **eks.tf** : Defines the EKS cluster resources.
2. **eks_permissions.tf**: Configures the IAM roles and permissions required for the EKS cluster.
3. **iam_role.tf & iam_role_policy_attachments.tf**: Create and attach IAM roles and policies.
4. **outputs.tf**: Exposes outputs like EKS cluster name or VPC details.
5. **providers.tf**: Specifies the Terraform providers, such as AWS.
6. **security-groups.tf**: Manages network security group configurations.
7. **variables.tf**: Contains variable declarations used across Terraform scripts.
8. **vpc.tf**: Sets up the networking infrastructure, including VPC, subnets, and routing.

---

1. `AWS` – Cloud Provider Configuration

Purpose:

Specifies the cloud provider (AWS in this case) and necessary configurations like the region.

Steps:

1. Define the cloud provider using the `provider` block.
2. Specifies the AWS provider and sets the region to `us-east-2`. This tells Terraform to create resources in the Ohio region..

`provider.tf`:

```
provider "aws" {
profile = "default"
region     = var.aws_region
}
```

provider "kubernetes" { config_path = "~/.kube/config" # Ensure this path is correct }

```
## 2. `Variables` – Variable Definitions

### Purpose:
Defines all variables used in the project, such as AWS region, instance type, and key
pair names.
### Steps:
1. Define variables with a name, description, and optional default value.
2. Values can be overridden in the `terraform.tfvars` file or via command-line input.

### `variables.tf`:
```hcl
variable "aws_region" {
  description = "AWS Region"
  default     = "us-east-1"
}

variable "instance_type" {
  description = "Type of EC2 instance"
  default     = "t3.medium"
}

variable "vpc_id" {
  description = "The ID of the VPC where resources will be deployed"
}

variable "subnet_id" {
  description = "The ID of the subnet where resources will be deployed"
}
```

3. `VPC` – VPC Configuration

Purpose:

Defines the AWS Virtual Private Cloud (VPC) and its components, such as subnets and internet gateways.

Steps:

1. Create a VPC with a CIDR block.
2. Attach an internet gateway to enable internet access.

`vpc.tf`:
```

```
resource "aws_vpc" "webapp_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support = true
  enable_dns_hostnames = true
  tags = {
    Name = "webapp_vpc"
  }
}

# Public Subnets
resource "aws_subnet" "webapp_public_subnet_a" {
  vpc_id                  = aws_vpc.webapp_vpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true
  tags = {
    Name = "webapp_public_subnet_a"
  }
}

resource "aws_subnet" "webapp_public_subnet_b" {
  vpc_id                  = aws_vpc.webapp_vpc.id
  cidr_block              = "10.0.2.0/24"
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true
  tags = {
    Name = "webapp_public_subnet_b"
  }
}

# Private Subnets
resource "aws_subnet" "webapp_private_subnet_a" {
  vpc_id                  = aws_vpc.webapp_vpc.id
  cidr_block              = "10.0.3.0/24"
  availability_zone       = "us-east-1a"
  tags = {
    Name = "webapp_private_subnet_a"
  }
}

resource "aws_subnet" "webapp_private_subnet_b" {
  vpc_id                  = aws_vpc.webapp_vpc.id
  cidr_block              = "10.0.4.0/24"
  availability_zone       = "us-east-1b"
  tags = {
    Name = "webapp_private_subnet_b"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "webapp_igw" {
  vpc_id = aws_vpc.webapp_vpc.id
```

```
}

# Public Route Table
resource "aws_route_table" "webapp_public_rt" {
  vpc_id = aws_vpc.webapp_vpc.id
}

resource "aws_route" "webapp_internet_access" {
  route_table_id         = aws_route_table.webapp_public_rt.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.webapp_igw.id
}

resource "aws_route_table_association" "webapp_public_subnet_a" {
  subnet_id      = aws_subnet.webapp_public_subnet_a.id
  route_table_id = aws_route_table.webapp_public_rt.id
}

resource "aws_route_table_association" "webapp_public_subnet_b" {
  subnet_id      = aws_subnet.webapp_public_subnet_b.id
  route_table_id = aws_route_table.webapp_public_rt.id
}

# Private Route Table
resource "aws_route_table" "webapp_private_rt" {
  vpc_id = aws_vpc.webapp_vpc.id
}

# Create a NAT Gateway for private subnets (optional, if you need outbound internet
access for private subnets)
resource "aws_eip" "webapp_nat_eip" {}

resource "aws_nat_gateway" "webapp_nat_gw" {
  allocation_id = aws_eip.webapp_nat_eip.id
  subnet_id     = aws_subnet.webapp_public_subnet_a.id
}

resource "aws_route" "webapp_nat_route" {
  route_table_id         = aws_route_table.webapp_private_rt.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = aws_nat_gateway.webapp_nat_gw.id
}

resource "aws_route_table_association" "webapp_private_subnet_a" {
  subnet_id      = aws_subnet.webapp_private_subnet_a.id
  route_table_id = aws_route_table.webapp_private_rt.id
}

resource "aws_route_table_association" "webapp_private_subnet_b" {
  subnet_id      = aws_subnet.webapp_private_subnet_b.id
  route_table_id = aws_route_table.webapp_private_rt.id
}
```

4. `Eks` – Eks cluster Configuration

Purpose:

Deploys an AWS EKS Cluster using the specified instance type & AMI.

`eks.tf` :

```
resource "aws_eks_cluster" "webapp_eks" {
  name     = "webapp-eks"
  role_arn = aws_iam_role.webapp_eks_cluster_role.arn
  version  = "1.31"
  vpc_config {
    subnet_ids         = [
      aws_subnet.webapp_public_subnet_a.id,
      aws_subnet.webapp_public_subnet_b.id,
      aws_subnet.webapp_private_subnet_a.id,
      aws_subnet.webapp_private_subnet_b.id,
    ]
    security_group_ids = [aws_security_group.webapp_eks_sg.id]  # Attach security
group to the cluster
  }
  tags = {
    Name = "webapp_eks"
  }
}

resource "aws_eks_node_group" "webapp_node_group" {
  cluster_name    = aws_eks_cluster.webapp_eks.name
  node_group_name = "webapp-nodes"
  node_role_arn   = aws_iam_role.webapp_eks_nodes_role.arn
  ami_type        = "AL2_x86_64"
  instance_types = ["t3.medium"]
  scaling_config {
    desired_size = 1
    min_size     = 1
    max_size     = 2
  }

  subnet_ids = [
    aws_subnet.webapp_public_subnet_a.id,
    aws_subnet.webapp_public_subnet_b.id,
    aws_subnet.webapp_private_subnet_a.id,
    aws_subnet.webapp_private_subnet_b.id,
  ]

  tags = {
    Name = "webapp-nodes"
  }
}

resource "null_resource" "update_kubeconfig" {
```

```
  depends_on = [aws_eks_cluster.webapp_eks]

  provisioner "local-exec" {
    command = "aws eks update-kubeconfig --name ${aws_eks_cluster.webapp_eks.name} --
region ${var.aws_region}"
  }
}

resource "null_resource" "wait_for_cluster" {
  depends_on = [aws_eks_cluster.webapp_eks]

  provisioner "local-exec" {
    command = "powershell -Command \"while ((aws eks describe-cluster --name
${aws_eks_cluster.webapp_eks.name} --query 'cluster.status' --output text) -ne
'ACTIVE') { Start-Sleep -Seconds 10 }\""
  }
}
```

5. `Security Groups` – EC2 Security Groups

Purpose:

Defines security groups for the EC2 instance, specifying inbound and outbound rules.

Steps:

1. Create security groups to allow specific inbound and outbound traffic (e.g., HTTP on port 80).
2. Attach the security group to the EC2 instance.

`security-groups.tf` :

```
# Security Group for EKS Cluster
resource "aws_security_group" "webapp_eks_sg" {
  vpc_id = aws_vpc.webapp_vpc.id

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
```

```
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 1025
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "webapp_eks_sg"
  }
}
```

6. `IAM` – IAM Roles and Policies

Purpose:

Manages AWS IAM roles and policies for assigning permissions to instances.

Steps:

1. Create an IAM role for EC2 instances.
2. Define policies for the role, such as assuming roles or specific permissions.

`iam_role.tf`:

```
resource "aws_iam_role" "webapp_role" {
  name = "webapp_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Service = "ec2.amazonaws.com" # Adjust this for your use case
        }
        Action = "sts:AssumeRole"
      },
    ]
  })
}

# New IAM Role for EKS Cluster
resource "aws_iam_role" "webapp_eks_cluster_role" {
  name = "webapp_eks_cluster_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
```

```
      {
        Effect = "Allow"
        Principal = {
          Service = "eks.amazonaws.com"
        }
        Action = "sts:AssumeRole"
      }
    ]
  })
}

# New IAM Role for EKS Nodes
resource "aws_iam_role" "webapp_eks_nodes_role" {
  name = "webapp_eks_nodes_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Service = "ec2.amazonaws.com"
        }
        Action = "sts:AssumeRole"
      }
    ]
  })
}
```

7. `Iam role policy` – role policy attachment

`iam_role_policy_attachment.tf`:

```
resource "aws_iam_role_policy_attachment" "secrets_manager_policy_attachment" {
  policy_arn = "arn:aws:iam::aws:policy/SecretsManagerReadWrite"
  role      = aws_iam_role.webapp_role.name
}

# Attach policies to EKS Cluster Role
resource "aws_iam_role_policy_attachment" "webapp_eks_cluster_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role     = aws_iam_role.webapp_eks_cluster_role.name
}

# Attach policies to EKS Nodes Role
resource "aws_iam_role_policy_attachment" "webapp_eks_node_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role     = aws_iam_role.webapp_eks_nodes_role.name
}
# Attach AmazonEKS_CNI_Policy to EKS Nodes Role
resource "aws_iam_role_policy_attachment" "webapp_eks_cni_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
```

```
  role       = aws_iam_role.webapp_eks_nodes_role.name
}

resource "aws_iam_role_policy_attachment" "webapp_ec2_container_registry_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
  role     = aws_iam_role.webapp_eks_nodes_role.name
}

}
```

8. `Output` – Output Values

Purpose:

Outputs useful information after applying Terraform, such as the instance's public IP.

Steps:

1. Use output blocks to display important data (e.g., the EC2 instance's public IP).
2. Access these values after running `terraform apply`.

`output.tf`:

```
output "eks_cluster_name" {
  description = "The name of the EKS cluster"
  value       = aws_eks_cluster.webapp_eks.name
}

output "eks_cluster_endpoint" {
  description = "The endpoint of the EKS cluster"
  value       = aws_eks_cluster.webapp_eks.endpoint
}

output "vpc_id" {
  description = "The ID of the VPC"
  value       = aws_vpc.webapp_vpc.id
}
```

Deploy the Infrastructure

Run the following commands in the directory containing the scripts:

1. Initialize the Terraform Working Directory

```
terraform init
```

2. Validate the Terraform Configuration

```
terraform validate
```

3. Plan the Deployment

```
terraform plan
```

4. Apply the Configuration

```
terraform apply
#Terraform will prompt for confirmation before applying the changes. Type yes to
proceed.
```

5. Destroy the Infrastructure

```
terraform destroy
# This command will also prompt for confirmation. Type yes to proceed with destroying
the resources.
```

DOCKERFILE

Dockerfile Documentation

This guide provides an explanation and usage instructions for the provided Dockerfile, which is designed to create a lightweight web application image using NGINX.

Steps :

1. Create a dockerfile

Dockerfile

```
# Use NGINX as the base image
FROM nginx:latest

# Set the working directory
WORKDIR /usr/share/nginx/html

# Copy the static HTML content to the NGINX root directory
COPY index.html .

# Expose the default HTTP port
EXPOSE 80

# Run NGINX
CMD ["nginx", "-g", "daemon off;"]
```

2. Building the Docker Image

- Navigate to the directory containing the Dockerfile and index.html file.

  ```
  cd kuberetes-deployment-files
  ```

- Build the Docker image:

```
docker build -t static-webapp:latest .
```

3. Running the Docker Container

- Run the container:

```
docker run -d -p 80:80 static-webapp:latest
# This maps port 80 on the host machine to port 80 in the container.
```

4. Access the application in your browser:

http://localhost:80

5. Pushing the Image to a Repository

- Tag the image

```
docker tag static-webapp:latest chittimallanikhil:web-app:latest
```

- Login in to the DockerHub

```
docker login
# login with the credentials of the Docker Hub
```

- Push the image to Docker Hub:

```
docker push chittimallanikhil:web-app:latest
```

KUBERNETES DEPLOYMENT FILES

Kubernetes Deployment and Service Documentation

This guide explains the configuration and deployment of a Kubernetes application using a single manifest file that defines both a Deployment and a Service.

Overview

The provided manifest file deploys a web application with the following configurations:

Deployment: Manages application pods with two replicas. **Service**: Exposes the application externally using a LoadBalancer.

steps:

1. Create a manifest file and save it.

`webapp-deployment-service.yml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: chittimallanikhil/web-app
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
```

2. Apply the file using kubectl:

```
kubectl apply -f webapp-deployment-service.yml
```

3. Verify the Deployment and Service:

Check the pods:

```
kubectl get pods
```

Check the service:

```
kubectl get services
```

Note the external IP assigned to the service (available after a few moments):

```
kubectl get service webapp-service
```

4. Verification

Access the application using the external IP of the service on port 80:

http://localhost:80

PROMETHUES

Prometheus Monitoring Solution for Kubernetes

This guide provides step-by-step instructions for deploying Prometheus to monitor a Kubernetes cluster and its applications.

Prerequisites

- Kubernetes Cluster: A running Kubernetes cluster.
- Helm Installed: Install Helm if not already available.
- kubectl Installed: Ensure kubectl is configured to access the cluster.

Steps :

Step 1: Add Prometheus Helm Chart Repository

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
# This command adds the official Prometheus community repository and updates the Helm repository cache.
```

Step 2: Install Prometheus

Install Prometheus using the Helm chart:

```
helm install prometheus prometheus-community/prometheus
```

This installs Prometheus in the default namespace. To install it in a specific namespace, use:

```
helm install prometheus prometheus-community/prometheus -n <namespace> --create-namespace
```

Step 3: Verify the Installation

```
helm list -n default
```

Step 4: Verify Prometheus pods are running:

```
kubectl get pods -n default
```

Step 5: Access Prometheus UI

Forward the Prometheus server port to your local machine: (windows)

```
$POD_NAME = kubectl get pods --namespace default -l
"app.kubernetes.io/name=prometheus,app.kubernetes.io/instance=prometheus" -o
jsonpath="{.items[0].metadata.name}"

kubectl --namespace default port-forward $POD_NAME 9090
```

Access the Prometheus web interface in your browser:

http://localhost:9090