Infrastructure Automation and Configuration with Terraform and Ansible

Version History

#	Version	Author	Reviewer
1			

Overview

The Proof of Concept (PoC) you've developed showcases a comprehensive solution for achieving high availability and efficient load distribution in a web application environment. By seamlessly integrating Terraform, Ansible, and AWS services, you've demonstrated a robust approach to deploy and manage your NGINX application.

In this PoC, Terraform serves as the foundation, enabling the automated provisioning of AWS resources. You've scripted the creation of three EC2 instances, strategically positioned across different availability zones. These instances are primed to handle incoming traffic and maintain the application's uptime, even in the face of hardware or software failures.

Ansible plays a pivotal role in configuration management, automating the setup of your NGINX application on the EC2 instances. Through an Ansible playbook, you've streamlined the installation and configuration process, ensuring consistent and accurate deployment across all instances.

Central to this PoC is the Elastic Load Balancer (ELB) orchestrated within AWS. By attaching the EC2 instances to the load balancer, you've established a resilient and highly available architecture. The ELB acts as a traffic manager, intelligently distributing incoming requests among the EC2 instances. This not only optimizes performance but also guarantees uninterrupted service by instantly routing traffic away from any faulty instance.

The key advantage of this PoC lies in its ability to provide a seamless user experience. Users can effortlessly access your NGINX application through the load balancer's DNS name. Behind the scenes, the load balancer's intelligent algorithms ensure that every request is directed to a healthy instance, mitigating any potential disruptions and bottlenecks.

Pre-requisites

- Ansible
- Terraform
- Open HTTP, HTTPS Port numbers in the security group

Procedure

Step1: Configure Instances by Using Terraform Script

- To perform this poc we need one ubuntu server and connect to that server
- Install packages ansible and terraform
  Terraform Installation: Install | Terraform | HashiCorp Developer
- Ansible Installation: Install and Configure Ansible on Ubuntu 22.04 Linux - Linux Shout (how2shout.com)
- We have to copy the ansible-demo zip file windows to linux server by using WinSCP tool https://3ktechnologies22-my.sharepoint.com/:u:/g/personal/vishak_a_3ktechnologies_com/Eby--gqp9BNPmvKRGpNuro4Bdgc3rd-X2wI-IvGdNE-RhA?e=xBdul9
- To unzip this file we have to install "sudo apt install unzip"
- sudo unzip ansible_demo.zip → to unzip the file
- we have to configure your account details
  sudo apt install awscli
  sudo aws configure
- Generate rsa public and private key by using below command
  ssh-keygen
- Your keys will be stored under this path copy the public key **cd /home/ubuntu/.ssh**
- Go to ansible home directory **cd /etc/ansible** and paste your private key path and save the file
  [defaults]
  private_key_path=/home/ubuntu/.ssh/id_rsa

- Go to this path ansible_demo/terraform_infra/modules/ec2/main.tf, Under main.tf file paste your **public key** and change your **Ami-id**, change your key pair name **key_name = "ubuntu.key"** run the following commands to install ec2-instances
  sudo chmod 777 main.tf
  vim main.tf
  sudo terraform init
  sudo terrafrom validate
  sudo terraform validate
  sudo terraform apply

Three EC2-Instances were created in your aws account and copy the 3-instance public ip address

Step2: Installing Nginx Application

- Go to ansible directory in that directory you will find out **inventory.yaml** file in that file replace the existing ip addresses with the last obtained ones in the file
- Check connection by pinging the inventory instances through ansible

  ansible -m ping virtualmachines -i inventory. yaml
- Execute the command to install and configure nginx.

  ansible-playbook -i inventory. yaml sample-nginx-playbook. yaml
- Try to access the instances through browser using their public ips, you can able to access your nginx application

Step3: Create Load Balancer Manually

- To create load balancer first we need to create **target group**, attach those three instances in target group what we created previously.
- In the security group we need to open **http** and **https** protocols.
- We can access the nginx application through DNS name of load balancer.

Step4: Create Load Balancer through Terraform Script

```
resource "random_string" "random" {
 length         = 8
 special        = true
 override_special = "/@£$"
}

resource "aws_security_group" "ansible-managed-node-sg" {
 name = "ansible-manged-node-sg"
 description = "basic sg for managed ansible nodes"

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

 // To Allow Port 80 Transport
 ingress {
  from_port = 80
  protocol = "tcp"
  to_port = 80
  cidr_blocks = ["0.0.0.0/0"]
 }
```

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

 lifecycle {
  create_before_destroy = true
 }
}

resource "aws_key_pair" "managed-node-key" {
 key_name   = "ubuntu-key"                # change your key_name
 public_key = "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCbgB5I6OBLvFLW/8bU2wkzyupE3P8iCuWx+W3temBISJcLZW8i/
i5A9YLC+g7/rrRwR/tfGpq0Y5JCX8u64FmzjvytaieSxhZfZB/xTESnEi7TIP6v2Ae6Q4Gj9djcpUMvutgXNVul1u
LsYSDxJp/Yl+uJ5g0GHr2XrHhoxXSBGSIZbT2wf2LNCyLgEM1GtVTdDvXOGxdVvafGapzIIGO/II/uwJ0rrsXo9O
Cdmd1hPA3ASpJBnECffc2b8I/vkO9sCbNTVEMU0KpzfTo2XxrFX0rAvj75I8NidiM0Eez/UHDnf/PNehorecIC
GpH96bRPRA16JQg+gLDxDH6oCb73EKTiNxOEFC5TDXd+iQC6FaPx3f1Klks1s6G8uwePbWLJ0Js6Vtp9HPPT
skAWVtFppqg8OTf1xOTe2V21wHKWqNYu/Bn8+JqpqWffkd4X25b7orfFaUILyt4jAnbmr8eM3kPCn/2za2lp
xxvdaxoQY0cvknSHRPvs6qq1DcYrJYM= ubuntu@APPSERVER" #create rsa key pair, put public key here
}

data "aws_availability_zones" "available" {
 state = "available"
}

resource "aws_instance" "managed_node" {
   count        = 3
   ami          = "ami-03f65b8614a860c29"     #change your ami-id of ubuntu server
   instance_type = "t2.micro"
   key_name     = aws_key_pair.managed-node-key.key_name

   tags = {
      Name = "ansible-managed-node-${random_string.random.result}"
 }


 vpc_security_group_ids = [
  aws_security_group.ansible-managed-node-sg.id
 ]
 availability_zone = data.aws_availability_zones.available.names[count.index]
```

```
}

resource "aws_security_group" "alb_security_group" {
  name        = "alb-security-group"
  description = "Security group for ALB"
  vpc_id      = "vpc-06a33deb8df68d299"            # Replace with your VPC ID

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    protocol = "tcp"
    to_port = 22
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port       = 0
    to_port         = 0
    protocol        = "-1"
    cidr_blocks     = ["0.0.0.0/0"]
  }
  lifecycle {
    create_before_destroy = true
  }

  # Add additional ingress or egress rules if needed
}

resource "aws_lb" "example_alb" {
  name               = "lb"
  internal           = false
  load_balancer_type = "application"
  subnets            = ["subnet-0a5ac6af0c4ee297a", "subnet-0893e4955428e5532", "subnet-
0ee944ff84e2a754d"]                     #Replace your subnet Ids
  security_groups    = [aws_security_group.alb_security_group.id]
}

resource "aws_lb_listener" "exa" {
  load_balancer_arn = aws_lb.example_alb.arn
  port              = 80
```

```
  protocol         = "HTTP"

  default_action {
    type           = "forward"
    target_group_arn = aws_lb_target_group.example_target_group.arn
  }
}

resource "aws_lb_target_group" "example_target_group" {
  name       = "example-target-group"
  port       = 80
  protocol    = "HTTP"
  vpc_id     = "vpc-06a33deb8df68d299"        # Replace your Vpc-id
  target_type = "instance"
}

resource "aws_lb_target_group_attachment" "example_attachment" {
  count           = 3
  target_group_arn  = aws_lb_target_group.example_target_group.arn
  target_id        = aws_instance.managed_node[count.index].id
}

output "alb_dns_name" {
  value = aws_lb.example_alb.dns_name
}
```

References

Install and Configure Ansible on Ubuntu 22.04 Linux - Linux Shout (how2shout.com)

Install | Terraform | HashiCorp Developer

Microsoft Teams Chat Files - OneDrive (sharepoint.com)