DevOps Shack

Terraform + Ansible Project

Detailed Documentation for a Terraform + Ansible Project

This documentation covers a project that automates the creation of AWS EC2 instances using Terraform and configures them using Ansible to install Docker and run SonarQube in a Docker container. The project also includes instructions for setting up AWS CLI, Terraform, configuring SSH access, and automating deployment and provisioning.

1. Prerequisites

Before beginning the project, ensure that the following prerequisites are met:

- **AWS account** with programmatic access (AWS access key and secret key).
- **Ansible** and **Terraform** installed on your control node (local machine or remote server).
- **SSH key** for accessing the AWS EC2 instances.

2. Setting Up the Environment

Step 1: Install AWS CLI

AWS CLI is necessary to manage AWS resources from the command line. Below are the commands to install and configure AWS CLI:

```
# Install AWS CLI
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install unzip
unzip awscliv2.zip
sudo ./aws/install


# Configure AWS CLI

aws configure
```

When configuring AWS CLI, you will be prompted for your **AWS Access Key**, **Secret Key**, **Region**, and **Output format**.

Step 2: Install Terraform

Terraform is an open-source infrastructure as code (IaC) tool used to provision and manage cloud resources, such as AWS EC2 instances.

Install required dependencies

```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
```

Add HashiCorp's GPG key and repository

```
wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null
```

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
```

Update the package list and install Terraform

```
sudo apt update
```

```
sudo apt-get install terraform -y
```

Verify the installation by running:

```
terraform --version
```

Step 3: Create SSH Key Pair

You will need an SSH key to access the EC2 instances that are provisioned by Terraform.

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -C "your_email@example.com"
```

Save the private key (id_ed25519) securely, and the public key (id_ed25519.pub) will be copied to the EC2 instances using Terraform.

3. Terraform Configuration

File 1: main.tf (Terraform Infrastructure Definition)

This file contains the Terraform code to create EC2 instances, configure SSH access, and disable strict host key checking.

Key Sections of the Terraform File:

1. **Provider Block:** The provider block specifies the AWS region to use.

```
provider "aws" {
  region = "ap-south-1"
}
```

2. **EC2 Instance Resource:** This section defines the AWS EC2 instance properties such as AMI ID, instance type, subnet ID, key pair, security groups, and instance tags.

```
resource "aws_instance" "ec2_instance" {
  count         = var.number_of_instances
  ami           = var.ami_id
  subnet_id     = var.subnet_id
  instance_type = var.instance_type
  key_name      = var.ami_key_pair_name
  security_groups = ["sg-0f767baf3e3df0e07"]
  tags = {
    Name = "${var.instance_name}-${count.index + 1}"  # Unique name for each instance
  }
}
```

3. **Provisioning Block:** Using the null_resource type and provisioners (file and remote-exec), we upload the public SSH key to the EC2 instance and configure SSH access.

```
resource "null_resource" "configure_ssh" {
  count = var.number_of_instances

  connection {
    type        = "ssh"
    host        = aws_instance.ec2_instance[count.index].public_ip
    user        = "ubuntu"
    private_key = file("/home/ubuntu/T/DevOps.pem")
  }

  provisioner "file" {
    source      = "/home/ubuntu/.ssh/id_ed25519.pub"
    destination = "/home/ubuntu/id_ed25519.pub"
  }

  provisioner "remote-exec" {
    inline = [
      "mkdir -p ~/.ssh",
      "cat /home/ubuntu/id_ed25519.pub >> ~/.ssh/authorized_keys",
      "chmod 700 ~/.ssh",
      "chmod 600 ~/.ssh/authorized_keys"
    ]
  }
}
```

4. **Disabling Strict Host Key Checking:** This block disables strict host key checking on the newly created EC2 instances to avoid SSH prompts.

```
resource "null_resource" "disable_strict_host_key_checking" {
 count = var.number_of_instances

 connection {
  type        = "ssh"
  host        = aws_instance.ec2_instance[count.index].public_ip
  user        = "ubuntu"
  private_key = file("/home/ubuntu/PK/DevOps.pem")
 }

 provisioner "remote-exec" {
  inline = [
   "echo 'Host *' >> ~/.ssh/config",
   "echo '  StrictHostKeyChecking no' >> ~/.ssh/config",
   "echo '  UserKnownHostsFile=/dev/null' >> ~/.ssh/config",
   "echo '  LogLevel ERROR' >> ~/.ssh/config"
  ]
 }

 depends_on = [aws_instance.ec2_instance]
}
```

5. **Output Block:** The output block outputs the public IP addresses of the created EC2 instances.

```
output "vm_info" {
 value = { for idx, instance in aws_instance.ec2_instance : "${instance.tags.Name}" =>
instance.public_ip }
}
```

File 2: terraform.tfvars

This file contains the values of the variables used in main.tf.

```
instance_name       = "Test-instance"
instance_type       = "t2.medium"
subnet_id           = "subnet-0164395797ba54f93"
ami_id              = "ami-0f58b397bc5c1f2e8"
number_of_instances  = 2
ami_key_pair_name    = "DevOps"
```

File 3: variables.tf

This file defines the variables used in main.tf and sets default values.

```
variable "instance_name" {
 description = "Name of the instance to be created"
 default     = "Test-instance"
}

variable "instance_type" {
 description = "Type of instance to be created"
 default     = "t2.micro"
}
```

```
variable "subnet_id" {
  description = "The VPC subnet the instance(s) will be created in"
  default     = "subnet-0164395797ba54f93"
}

variable "ami_id" {
  description = "The AMI to use"
  default     = "ami-08e5424edfe926b43"
}

variable "number_of_instances" {
  description = "Number of instances to be created"
  default     = 1
}

variable "ami_key_pair_name" {
  description = "Key pair name for the instances"
  default     = "DevOps"
}
```

Running Terraform

1. **Initialize Terraform:**

```
terraform init
```

2. **Apply the Terraform Configuration:**

```
terraform apply -var-file="terraform.tfvars"
```

This command creates the EC2 instances as defined in the configuration files.

---

4. Ansible Configuration

Inventory File: inventory

The Ansible inventory file contains the public IP addresses of the EC2 instances created by Terraform. Ansible uses this file to know which servers to target for configuration.

```
[servers]

3.110.32.14

15.207.85.250


[servers:vars]

ansible_user=ubuntu
```

Ansible Playbook: playbook.yml

The Ansible playbook installs Docker on the EC2 instances and runs a SonarQube container.

Tasks in the Playbook:

1. **Update the apt package index:**

```
- name: Update apt package index
  apt:
    update_cache: yes
    cache_valid_time: 3600
```

This task ensures that the system's package list is up-to-date.

2. **Install Docker:**

```
- name: Install Docker
  apt:
    name: docker.io
    state: present
    update_cache: yes
```

This task installs Docker on the EC2 instance.

3. **Add user to the Docker group:**

```
- name: Add user to docker group
  user:
    name: "{{ ansible_user }}"
    groups: docker
    append: yes
```

This task ensures that the user can run Docker commands without needing root privileges.

4. **Ensure Docker is started and enabled:**

```
- name: Ensure Docker is started and enabled
  systemd:
    name: docker
    state: started
    enabled: yes
```

This task ensures that Docker is running and will start on system boot.

5. **Set Docker socket permissions:**

```
- name: Set Docker socket permissions
  file:
    path: /var/run/docker.sock
    mode: '0666'
```

This task sets the appropriate permissions for the Docker socket to allow non-root users to access it.

6. **Run the SonarQube container:**

```
- name: Run SonarQube container
  docker_container:
    name: sonarqube
    image: sonarqube:lts-community
    state: started
```

```
    ports:
      - "9000:9000"
```
This task runs SonarQube as a Docker container and exposes it on port 9000.

Running the Ansible Playbook

To execute the playbook and configure the EC2 instances:

```
ansible-playbook -i inventory playbook.yml
```

---

5. Conclusion

This project demonstrates how to use Terraform for provisioning AWS infrastructure and Ansible for configuration management. By following this documentation, you can:

- Use **Terraform** to automate the creation of AWS EC2 instances.
- Use **Ansible** to install Docker and configure applications (like SonarQube) on the provisioned EC2 instances.

The combination of **Terraform** for infrastructure as code (IaC) and **Ansible** for configuration management provides a powerful way to automate and manage cloud resources effectively.