Demo Project: Configure Dynamic Inventory

Project Description

In this project, we will:

- Create **four EC2 instances** using Terraform.
- Configure Ansible to use the **AWS EC2 Plugin** to dynamically fetch and manage inventory.
- Remove the hardcoded IP addresses in the Ansible inventory file.
- Run an Ansible playbook targeting dynamically assigned EC2 instances.

Step 1: Create AWS EC2 Instance with Terraform

1. **Clone the Terraform repository**:

   `git clone https://gitlab.com/twn-devops-projects/ansible/terraform-learn.git`

   ```
   git clone https://gitlab.com/twn-devops-projects/ansible/terraform-learn.git
   cd terraform-learn
   git checkout feature/deploy-to-ec2-default-components
   ```

2. **Remove the existing Ansible provisioner** from `main.tf`:

```
resource "null_resource" "configure-server" {
  triggers = {
    trigger = aws_instance.myapp-server.public_ip
  }

  provisioner "local-exec" {
    working_dir = "/mnt/c/Users/eduar/devops_projects2/08-ansible/ansible-pr
    command = "ansible-playbook --inventory ${aws_instance.myapp-server.p
  }
}
```

3. **Add four EC2 instances to** `main.tf` :

   Note:

   - server-one and server-two are `instance_type = "t2.mirco"` and “dev-server”
   - server-three and server-four are `instance_type = "t2.small"` and “prod-server”

```
resource "aws_instance" "myapp-server" {
  ami = data.aws_ami.latest-amazon-linux-image.id
  instance_type = var.instance_type

  subnet_id = aws_subnet.myapp-subnet-1.id
  vpc_security_group_ids = [aws_default_security_group.default-sg.id]
  availability_zone = var.avail_zone

  associate_public_ip_address = true
  key_name = aws_key_pair.ssh-key.key_name

  tags = {
    Name: "dev-server"
  }
}

resource "aws_instance" "myapp-server-two" {
  ami = data.aws_ami.latest-amazon-linux-image.id
```

```
    instance_type = var.instance_type

    subnet_id = aws_subnet.myapp-subnet-1.id
    vpc_security_group_ids = [aws_default_security_group.default-sg.id]
    availability_zone = var.avail_zone

    associate_public_ip_address = true
    key_name = aws_key_pair.ssh-key.key_name

    tags = {
        Name: "dev-server"
    }
}

resource "aws_instance" "myapp-server-three" {
    ami = data.aws_ami.latest-amazon-linux-image.id
    instance_type = "t2.small"

    subnet_id = aws_subnet.myapp-subnet-1.id
    vpc_security_group_ids = [aws_default_security_group.default-sg.id]
    availability_zone = var.avail_zone

    associate_public_ip_address = true
    key_name = aws_key_pair.ssh-key.key_name

    tags = {
        Name: "prod-server"
    }
}

resource "aws_instance" "myapp-server-four" {
    ami = data.aws_ami.latest-amazon-linux-image.id
    instance_type = "t2.small"

    subnet_id = aws_subnet.myapp-subnet-1.id
    vpc_security_group_ids = [aws_default_security_group.default-sg.id]
```

```
    availability_zone = var.avail_zone

    associate_public_ip_address = true
    key_name = aws_key_pair.ssh-key.key_name

    tags = {
        Name: "prod-server"
    }
}
```

4. Comment out the private **SSH key** `ssh_key_private` **variable** from the `terraform.tfvars`:

```
vpc_cidr_blocks = "10.0.0.0/16"
subnet_cidr_block = "10.0.10.0/24"
avail_zone = "us-east-1a"
env_prefix = "dev"
my_ip = "167.57.247.97/32"
instance_type = "t2.micro"
public_key_location = "/home/eb/.ssh/id_rsa.pub"
# ssh_key_private = "/home/eb/.ssh/id_rsa"
image_name = "al2023-ami-2023.*-x86_64"
```

5. **Initialize and apply Terraform**:

```
terraform init
terraform apply --auto-approve
```

6. **Verify EC2 instances in AWS Console**.

Step 2: Configure Ansible for Dynamic Inventory

1. Comment out the "`Wait for SSH connection`" playbook, from `deploy-docker-ec2-new-user/yaml`
2. **Modify Ansible configuration** (`ansible.cfg`) to enable the AWS EC2 plugin:

```
[defaults]
host_key_checking = False
# inventory = hosts
inventory = inventory_aws_ec2.yaml
interpreter_python = /usr/bin/python3.9

enable_plugins = aws_ec2

remote_user = ec2-user
private_key_file = /home/eb/.ssh/id_rsa
```

We need this so we can connect to the AWS account and fetch the information of the server instances.

Note: The below requirements are needed on the local controller node that executes the inventory.

- python > = 3.6
- boto3 > = 1.26.0
- botocore > = 1.29.0

3. **Create `inventory_aws_ec2.yaml` to dynamically fetch EC2 instances**:

```
---
plugin: aws_ec2
regions:
  - us-east-1
hostnames:
  - dns-name
keyed_groups:
  - key: tags
```

```
    prefix: tag
  - key: instance_type
    prefix: instance_type
```

4. **Test the Ansible inventory plugin**:

`ansible-inventory -i inventory_aws_ec2.yaml --list`

`ansible-inventory -i inventory_aws_ec2.yaml --graph`

Example Output:

```
ansible-inventory -i inventory_aws_ec2.yaml --graph
@all:
  |--@ungrouped:
  |--@aws_ec2:
  |  |--ec2-54-157-198-159.compute-1.amazonaws.com
  |  |--ec2-34-239-0-29.compute-1.amazonaws.com
  |  |--ec2-44-210-140-161.compute-1.amazonaws.com
  |  |--ec2-18-209-60-220.compute-1.amazonaws.com
  |--@tag_Name_dev_server:
  |  |--ec2-54-157-198-159.compute-1.amazonaws.com
  |  |--ec2-18-209-60-220.compute-1.amazonaws.com
  |--@instance_type_t2_micro:
  |  |--ec2-54-157-198-159.compute-1.amazonaws.com
  |  |--ec2-18-209-60-220.compute-1.amazonaws.com
  |--@tag_Name_prod_server:
  |  |--ec2-34-239-0-29.compute-1.amazonaws.com
  |  |--ec2-44-210-140-161.compute-1.amazonaws.com
  |--@instance_type_t2_small:
  |  |--ec2-34-239-0-29.compute-1.amazonaws.com
  |  |--ec2-44-210-140-161.compute-1.amazonaws.com
```

Step 3: Deploy Using Ansible

1. Modify `deploy-docker-new-user.yaml` to target the **dynamic inventory group**:

   Example:

```yaml
- name: Install Docker
  hosts: tag_Name_dev_server
  become: yes
  tasks:
  - name: Install Docker
    yum:
      name: docker
      update_cache: yes
      state: present
  - name: Start docker daemon
    systemd:
      name: docker
      state: started

- name: Create new linux user
  hosts: tag_Name_dev_server
  become: yes
  tasks:
  - name: Create new linux user
    user:
      name: eduardo
      groups: adm,docker

...
```

2. **Run the playbook**: `ansible-playbook deploy-docker-new-user.yaml`

```
TASK [Gathering Facts] *********************************************************************************************************************
ok: [ec2-18-209-60-220.compute-1.amazonaws.com]
ok: [ec2-54-157-198-159.compute-1.amazonaws.com]

TASK [Copy docker compose] *****************************************************************************************************************
ok: [ec2-18-209-60-220.compute-1.amazonaws.com]
ok: [ec2-54-157-198-159.compute-1.amazonaws.com]

TASK [Docker login] ************************************************************************************************************************
ok: [ec2-54-157-198-159.compute-1.amazonaws.com]
ok: [ec2-18-209-60-220.compute-1.amazonaws.com]

TASK [Start containers from compose] *******************************************************************************************************
[WARNING]: Docker compose: unknown None: /home/eduardo/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored,
please remove it to avoid potential confusion
ok: [ec2-18-209-60-220.compute-1.amazonaws.com]
ok: [ec2-54-157-198-159.compute-1.amazonaws.com]

PLAY RECAP *********************************************************************************************************************************
ec2-18-209-60-220.compute-1.amazonaws.com : ok=13   changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ec2-54-157-198-159.compute-1.amazonaws.com : ok=13   changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Step 4: Cleanup

- `terraform destroy --auto-approve`

Troubleshooting

Issue: Failed to connect to host via SSH

Error Message:

```
fatal: [ec2-18-209-60-220.compute-1.amazonaws.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh: ssh: connect to host ec2-18-209-60-220.compute-1.amazonaws.com port 22: Connection timed out", "unreachable": true}
```

Solution: Ensure the **Security Group allows SSH** from your IP.
