Automating a Minecraft Server Setup on AWS

Background

This tutorial will guide you through automating the provisioning, configuration, and deployment of a Minecraft server on AWS. We will use Terraform for infrastructure provisioning, Ansible for configuration management, and Docker for container management. By the end, you will have a fully automated pipeline that sets up an EC2 instance, installs Docker, deploys a Minecraft server container, and ensures the server restarts automatically on instance reboot.

Requirements

Before starting, ensure you have the following tools installed/available:

- An AWS Account (and associated credentials that allow you to create resources)
- AWS CLI (version 2.15+)
- Terraform (version 1.8+)
- Ansible (version 2.17+)

Setting Up and Installing the Necessary Tools

- **AWS CLI**

    1. **Install AWS CLI**: Follow the installation guide here
    2. **Configure AWS CLI**: Enter the following command in your terminal:

        ```
        aws configure
        ```

        You will be prompted to enter your AWS Access Key, Secret Access Key, region, and output format. If you are an IAM or AWS Account Root User, these are the only credentials you will need to set to access your AWS console. If you are using an AWS tool that generates session tokens, be sure to set that as well by typing the following command in your terminal (replace `value` with your session token):

        ```
        aws configure set aws_session_token <value>
        ```

- **Terraform**

    1. **Install Terraform**: Follow the installation guide here

- **Ansible**

    1. **Install Ansiblle**: Follow the installation guide here

Setting AWS Credentials as Environment Variables

- To allow Terraform and Ansible to interact with AWS (eg. authenticating the Terraform AWS provider), set your AWS credentials as environment variables as shown below (note: this step is only required for IAM or account root users).

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Pipeline Overview

1. **Provision Infrastructure with Terraform**: Create an EC2 instance and configure its network security settings.
2. **Configure Server with Ansible and Docker**: Install Docker on the EC2 and deploy the Minecraft server.
3. **Ensure Auto-Restart with Ansible**: Configure the server to restart automatically on instance reboot.

Pipeline Diagram

```
graph TD;
    A[Start] --> B[Provision EC2 with Terraform]
    B --> C[Install Docker with Ansible]
    C --> D[Deploy Minecraft Server with Docker]
    D --> E[Configure Auto-Restart with Ansible]
    E --> F[Connect to Minecraft Server]
```

(Graph is viewable on Github repository)

Execution Tutorial

After following the setup requirements above and cloning this github repository on your desired execution machine, we can automatically run the provisioning/configuration scripts by executing the `deploy.bash` file that's provided. All the commands that we need to deploy our project are located in this file. To execute this *bash* script, type the following commands in your execution machine's terminal (in the same directory where your `deploy.bash` file is located):

```
chmod +x deploy.bash
bash deploy.bash
```

Enter `yes` when prompted for connection to the server and then wait for the Ansible playbook to configure the instance.

Here is the full list of commands we will run in our *bash* script along with brief explanations of what each of them do:

```
#!/bin/bash

# Navigate to the Terraform directory and install the necessary providers
cd terraform-scripts
terraform init

# Format and validate the EC2 configuration file
terraform fmt
terraform validate

# Apply the EC2 instance configurations specified in main.tf
terraform apply -auto-approve

# Extract the public IP and private key associated with the newly created
EC2 instance
INSTANCE_IP=$(terraform output -raw instance_public_ip)
PRIVATE_KEY_PEM=$(terraform output -raw private_key_pem)

# Save the private key to a file and set permissions
echo "$PRIVATE_KEY_PEM" > ../ansible-scripts/minecraft_key.pem
chmod 400 ../ansible-scripts/minecraft_key.pem

# Update Ansible inventory with the new instance public IP
echo "[minecraft]" > ../ansible-scripts/inventory.ini
echo "$INSTANCE_IP ansible_user=ec2-user
ansible_ssh_private_key_file=minecraft_key.pem" >> ../ansible-
scripts/inventory.ini

# Wait for the instance to initialize (add a delay to ensure SSH is
available)
sleep 100

# Run Ansible playbook on inventory nodes specified (EC2 instance)
cd ../ansible-scripts
ansible-playbook -i inventory.ini playbook.yml

# Print Minecraft Server's Public IP address
echo "Minecraft Server IP address: $INSTANCE_IP"
```

Connect to the Minecraft Server with a Minecraft Client

1. Open Minecraft on any host machine and navigate to the **Multiplayer** Section.
2. Click on **Add Server**
3. Enter your EC2 instance's public IP address (located on your AWS EC2 instances dashboard) in the **Server Address** field.
4. Click **Done** and then select your server to join.

(Alternatively, to verify client connections can be made to your instance's public IP address (i.e., your Minecraft server address), run this command in your terminal: `nmap -sV -Pn -p T:25565 <instance_public_ip>`)

By following these steps, you have automated the process of setting up a Minecraft server on AWS. This approach ensures that the server is configured consistently and can be easily replicated or modified.

References

- AWS EC2 Documentation
- Terraform Documentation
- Building infrastructure with Terraform
- Ansible Documentation
- itzg/minecraft-server Docker Image
- How to Install Docker on EC2 and create a container
- How To Use Systemctl to Manage Systemd Services and Units