Lesson 12 Demo 01

Managing Secrets and Credentials with Terraform

Objective: To manage secrets and credentials with Terraform for deploying and managing infrastructure as code on various cloud platforms

Tools required: Ubuntu OS and AWS management console

Prerequisites: Ensure you have created and implemented the AWS access key and secret key before starting this demo. Refer to Lesson 08, Assisted Practice 02, for detailed steps.

Steps to be followed:

1. Set up the directory and configure the AWS provider
2. Initialize Terraform and apply the changes

Step 1: Set up the directory and configure the AWS provider

1.1 Run the following command to create a directory:
mkdir mydir

```
shreemayeebhatt@ip-172-31-76-194:~$ mkdir mydir
shreemayeebhatt@ip-172-31-76-194:~$
```

1.2 Navigate inside the directory using the following command:
cd mydir

```
shreemayeebhatt@ip-172-31-76-194:~$ cd mydir
shreemayeebhatt@ip-172-31-76-194:~/mydir$
```

1.3 Run the following command to configure your AWS credentials:
aws configure

```
shreemayeebhatt@ip-172-31-76-194:~/mydir$ aws configure
AWS Access Key ID [None]: AKIA2IMLWKKJ3SXU2QMR
AWS Secret Access Key [None]: 2Rc/F6D0nOOjHnxWGyihUBiE4KZN2hRCJHzUfKe1
Default region name [None]:
Default output format [None]:
shreemayeebhatt@ip-172-31-76-194:~/mydir$
```

Note: Provide your AWS access key and secret key as shown in the screenshot

1.4 Create a Terraform file using the following command:
vi authenticate.tf

```
shreemayeebhatt@ip-172-31-76-194:~/mydir$ vi authenticate.tf
shreemayeebhatt@ip-172-31-76-194:~/mydir$
```

1.5 Enter the following script inside the **authenticate.tf** file to configure the AWS provider, specifying the region where resources will be created by default:

```
provider "aws" {
region = "us-east-1" #by default the resources would be created in north virginia
of aws account
}
```

```
provider "aws" {
 region = "us-east-1" #by default the resources would be created in north virginia of aws account

 }

~
```

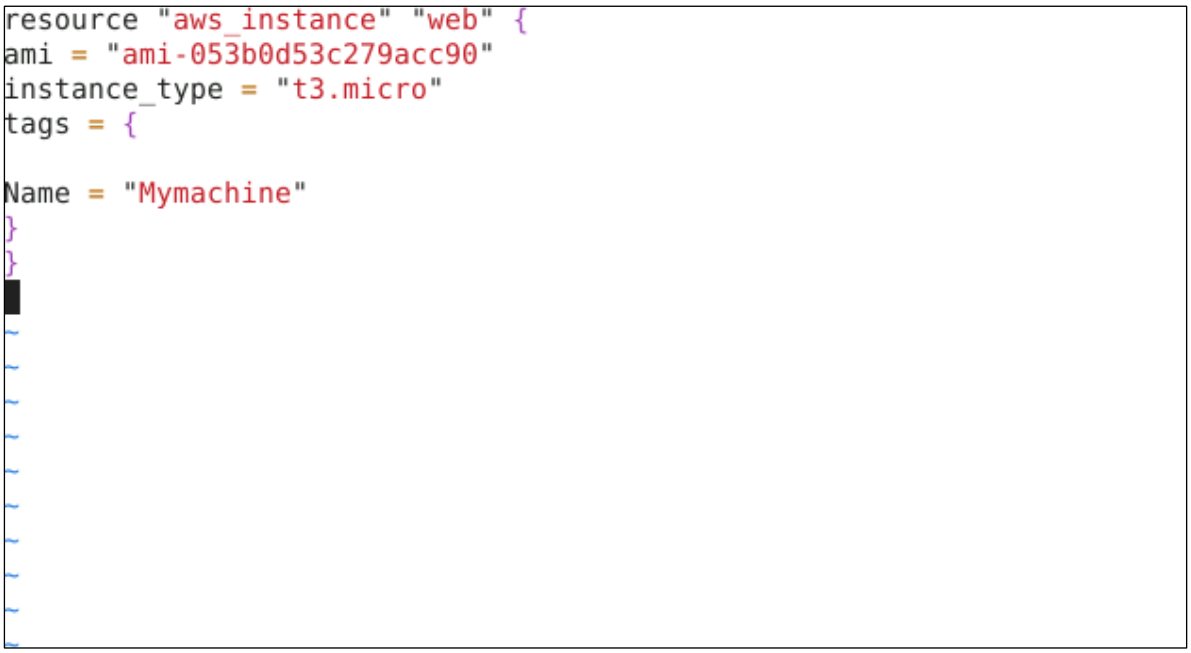
1.6 Create another Terraform file using the following command:
vi ec2.tf

```
shreemayeebhatt@ip-172-31-76-194:~/mydir$ vi ec2.tf
shreemayeebhatt@ip-172-31-76-194:~/mydir$
```

1.7 Enter the following script inside the **ec2.tf** file to define an EC2 instance resource that Terraform will manage:

```
resource "aws_instance" "web" {
ami = "ami-053b0d53c279acc90"
instance_type = "t3.micro"
tags = {

Name = "Mymachine"
}
}
```



Step 2: Initialize Terraform and apply the changes

2.1 Initialize the Terraform configuration file using the following command:
terraform init

```
shreemayeebhatt@ip-172-31-76-194:~/mydir$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.47.0...
- Installed hashicorp/aws v5.47.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
shreemayeebhatt@ip-172-31-76-194:~/mydir$
```

2.2 Run the following command to preview changes before applying them to the infrastructure:
terraform plan

```
shreemayeebhatt@ip-172-31-76-194:~/mydir$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-053b0d53c279acc90"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
```

2.3 Run the following command to automatically apply planned changes to the infrastructure:
terraform apply --auto-approve

```
shreemayeebhatt@ip-172-31-76-194:~/mydir$ terraform apply --auto-approve

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-053b0d53c279acc90"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t3.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
```

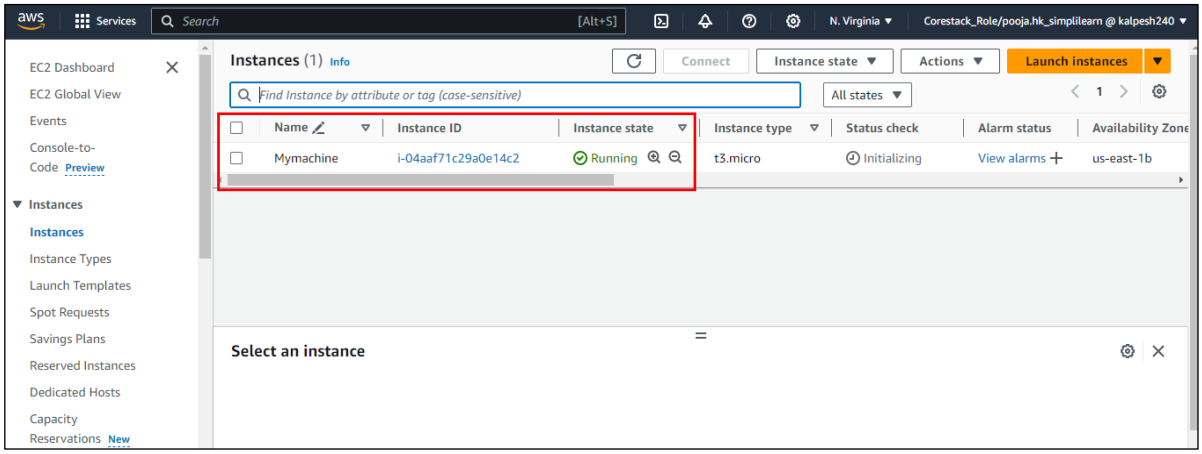
2.4 Verify the status to be successful in creation, as shown in the screenshot:

```
      + private_dns_name_options {
          + enable_resource_name_dns_a_record    = (known after apply)
          + enable_resource_name_dns_aaaa_record = (known after apply)
          + hostname_type                        = (known after apply)
        }

      + root_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + tags                  = (known after apply)
          + tags_all              = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Creation complete after 12s [id=i-04aaf71c29a0e14c2]
```

2.5 Navigate to your AWS console and check for the EC2 instance created, as shown:



Terraform securely uses the provided credentials to authenticate and create infrastructure on AWS. As you can see in the above screenshot, the successful creation of the EC2 instance in the AWS console demonstrates the capability of Terraform to securely manage and deploy resources.

By following these steps, you have successfully managed secrets and credentials with Terraform for deploying and managing infrastructure as code on various cloud platforms.