Hunter Orbit - Beat Drift-CLI Automation tool Implementation

COSC595 Final Presentation

Introduction- Stakeholders

Client:

- Mark Wallis - Director of Hunter Orbit, Ltd

COSC595 Unit Coordinator

- Prof. Edmund Sadgrove

Team Members

Jason O.Aboh

Sumit Khokhar

Paldeep Kaur

Project Background

Beat Drift-CLI Automation Tool Implementation

- Configuring resources in AWS.
- Creating all resources in AWS using Terraform Script.
- Configuring and updating Terraform state files.
- Analysing drift in Terraform by visualising the terraform state files.
- Comparing resources collected in AWS with the resources collected in Terraform to compute drift.

Project Objectives

Beat Drift-CLI Automation Tool Implementation

- Configuring Resources in AWS.
- Crawling resources in Terraform.
- Detecting drift while creating any instance in AWS using Terraform provider.
- Comparing the json output while configuring resources in AWS with the json output while configuring resources in Terraform to detect drift in CLI.

Project Options

Beat Drift-CLI Automation Tool Implementation

- Tagging and monitoring resources in AWS.
- Using the correct region for configuration purpose.
- Generating AMI id for windows machine to create resources in AWS using Terraform provider.
- Validating Terraform state files while launching any resource in aws using Terraform provider.
- Analysing the change in Terraform state files while using different commands in Terraform provider like terraform refresh, terraform plan, terraform apply.
- Detecting drift in Terraform while comparing the configuration of different resources created with the actual API'S.
- Comparing the Resources collected in AWS with that of the resources configured in Terraform to compute the drift.

Project Plan & Rationale

- Configuring resources in AWS.
- Configuring resources in Terraform.
- Addressing the drift in the CLI tool while creating in AWS using Terraform provider.
- Addressing the drift by comparing the json output of the resources collected in AWS with that of the json output of the resources configured in Terraform.

Critical Evaluation of the solution

Benefits

- Beat drift CLI tool helps in automation process.
- Improves operational efficiency while using AWS and Terraform.
- Reducing the probability of errors while using AWS and Terraform resources.

Limitations

- Spot similarities and dissimilarities between
- The solutions handle independent tasks to help automate them, however; they will need to be run separately and need a CLI or script to combine all solutions.
- Due to the use of certain Python libraries that require specific version ranges, the python versions that the solutions run on might be limited.

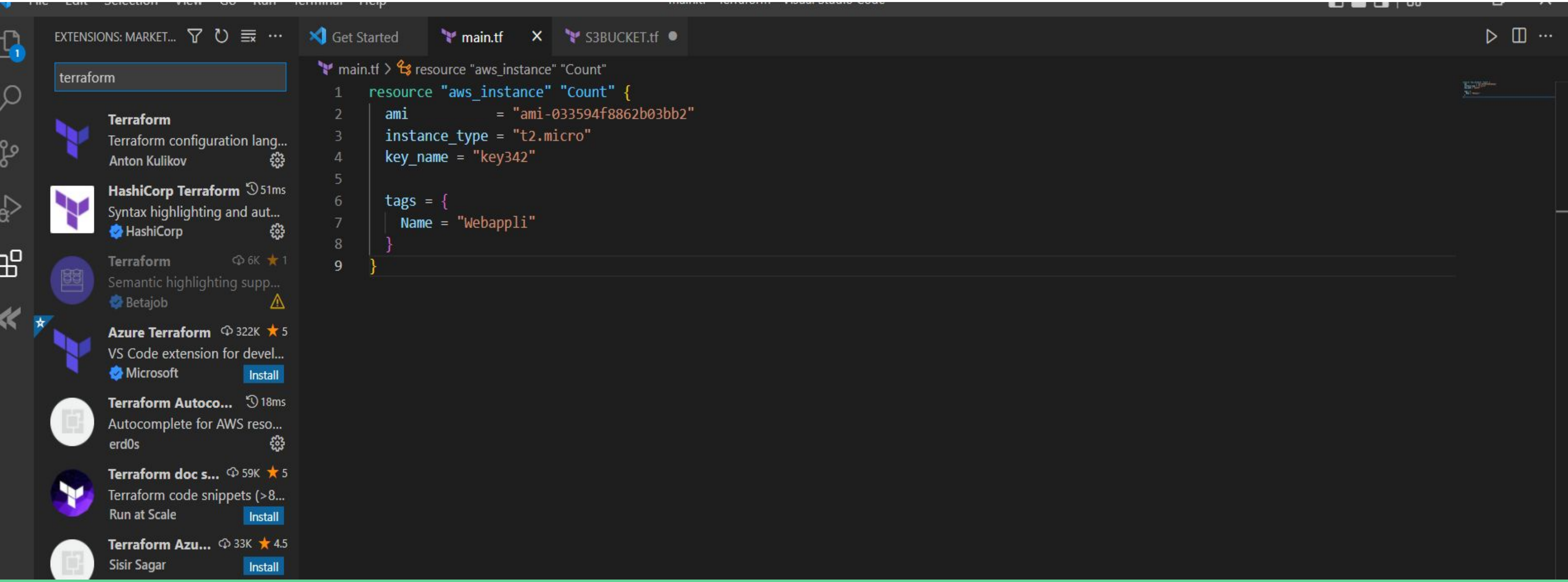
Summary of the project, lesson learned and future directions

The project allowed us to gain better understanding of working with the command-line tools and scripts in order to accomplished the varied tasks that were required to list resources within AWS and terraform and then detect drift.

Lessons learned was that, integration of CLI tools being build independently takes a lot of time and more cooperation while building these tools; in a nutshell the functionalities were built however in the future; more focus will be on integration of the solutions and implementation of the proposed solution(s).

Software

- Visual Studio software used for the integration of terraform script
- To integrate Terraform Extensions used in Visual Studio : Hashicorp Terraform, Terraform



```
resource "aws_instance" "Count" {
  ami           = "ami-033594f8862b03bb2"
  instance_type = "t2.micro"
  key_name = "key342"

  tags = {
    Name = "Webappli"
  }
}
```

Detecting and Managing drift with Terraform CLI

➢ Terraform CLI: The CLI tool is built using the Python programming language. The core Terraform commands are wrapped and implemented as functions. The code base for the CLI tool is divided into two:

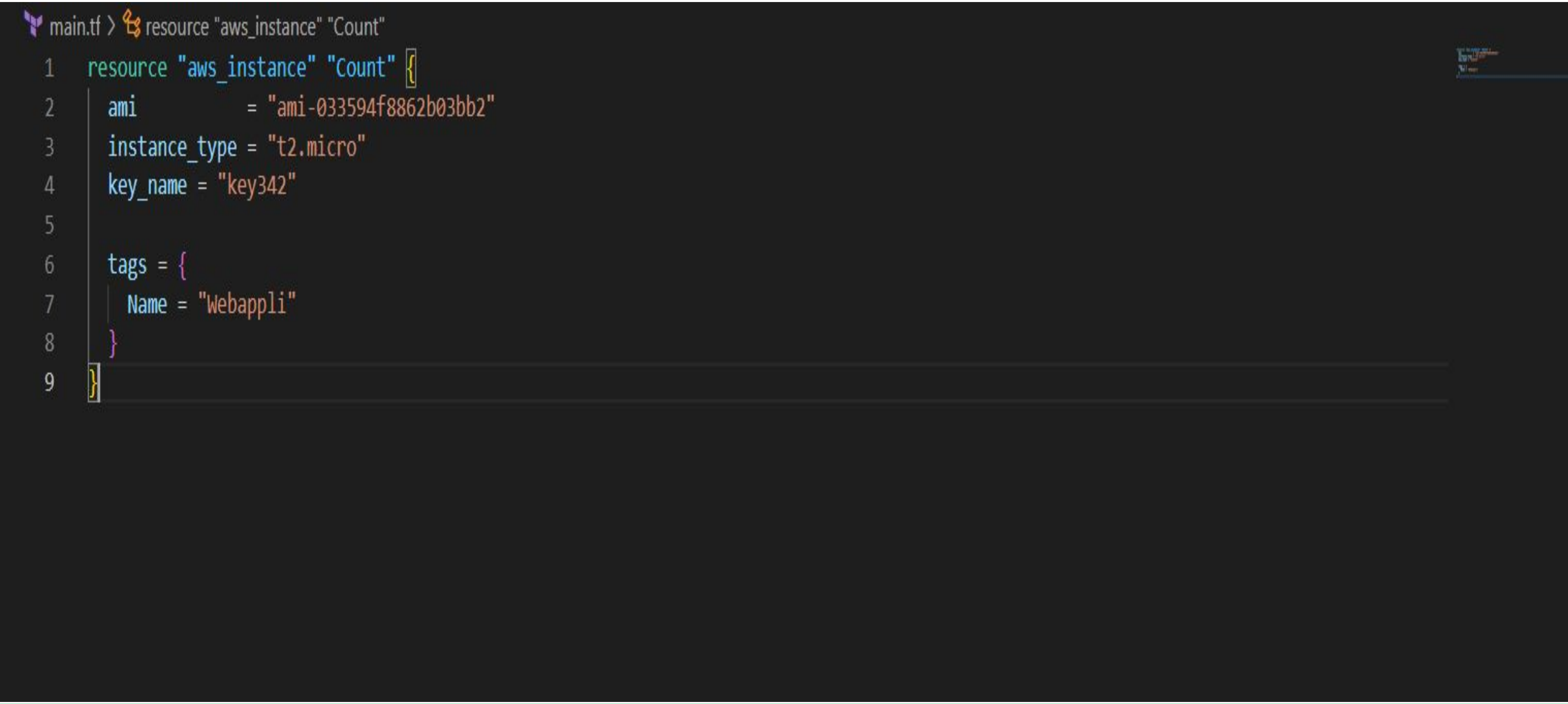
- Terraform code that contains all the necessary scripts to provision resources such as EC2 instances, S3 buckets, security groups etc. The main code for Terraform (terraform.tf). The script is used to create various AWS resources and the file that contains the variables (variables.tf).
- Python code that contains the scripts for the CLI tool and the necessary functions which are explained below. After setting the two codebases, the CLI tool can be run just like any other Python program.

Detecting and Managing drift with Terraform CLI

Steps through the main menu of terraform cli:

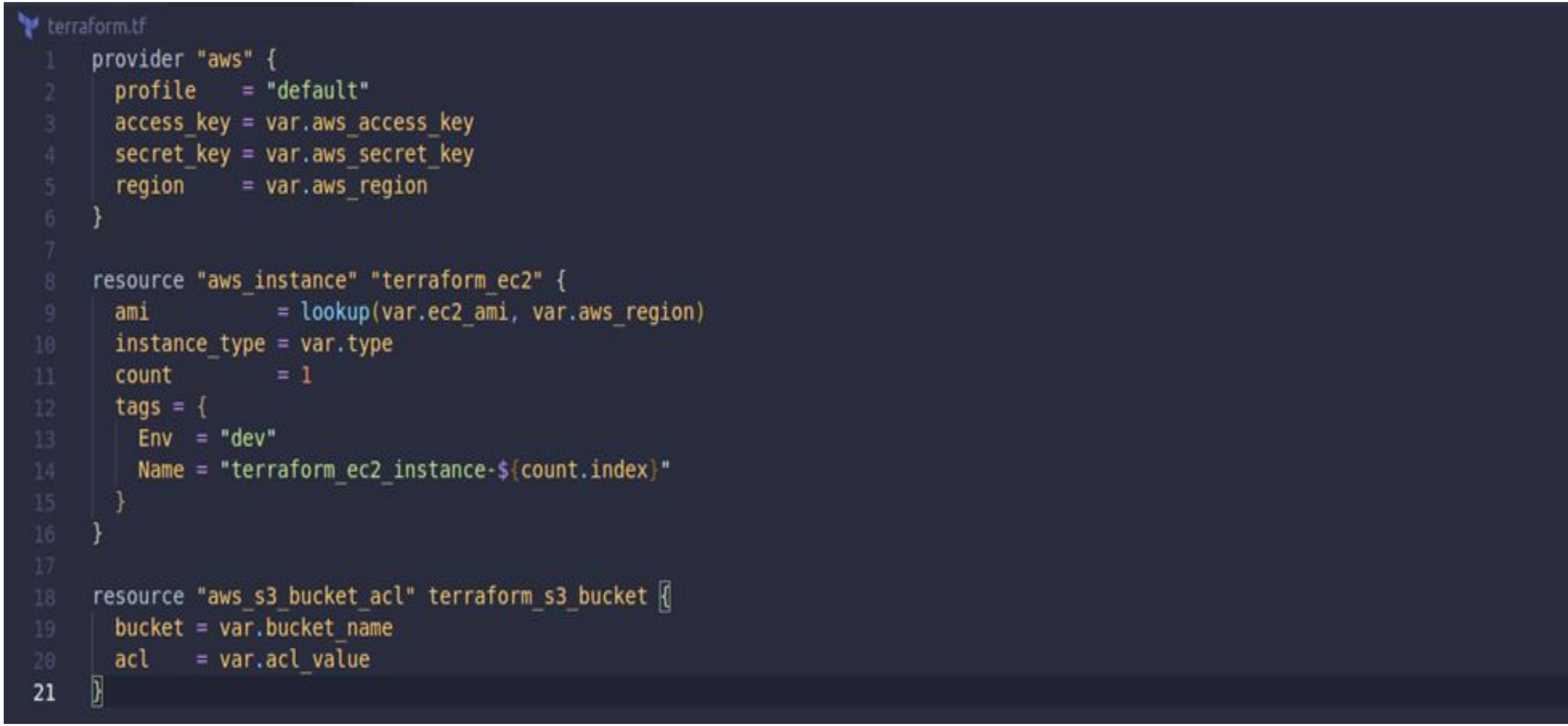
- Initialization of Terraform Directory
- Validation of Terraform Code
- Resources Provision
- Drift Detection
- Exiting the CLI tool

Creating EC2 instance in AWS using Terraform


main.tf > resource "aws_instance" "Count"

```
resource "aws_instance" "Count" {
  ami           = "ami-033594f8862b03bb2"
  instance_type = "t2.micro"
  key_name = "key342"

  tags = {
    Name = "Webappli"
  }
}
```

Provisioning of Complete Resources in Terraform

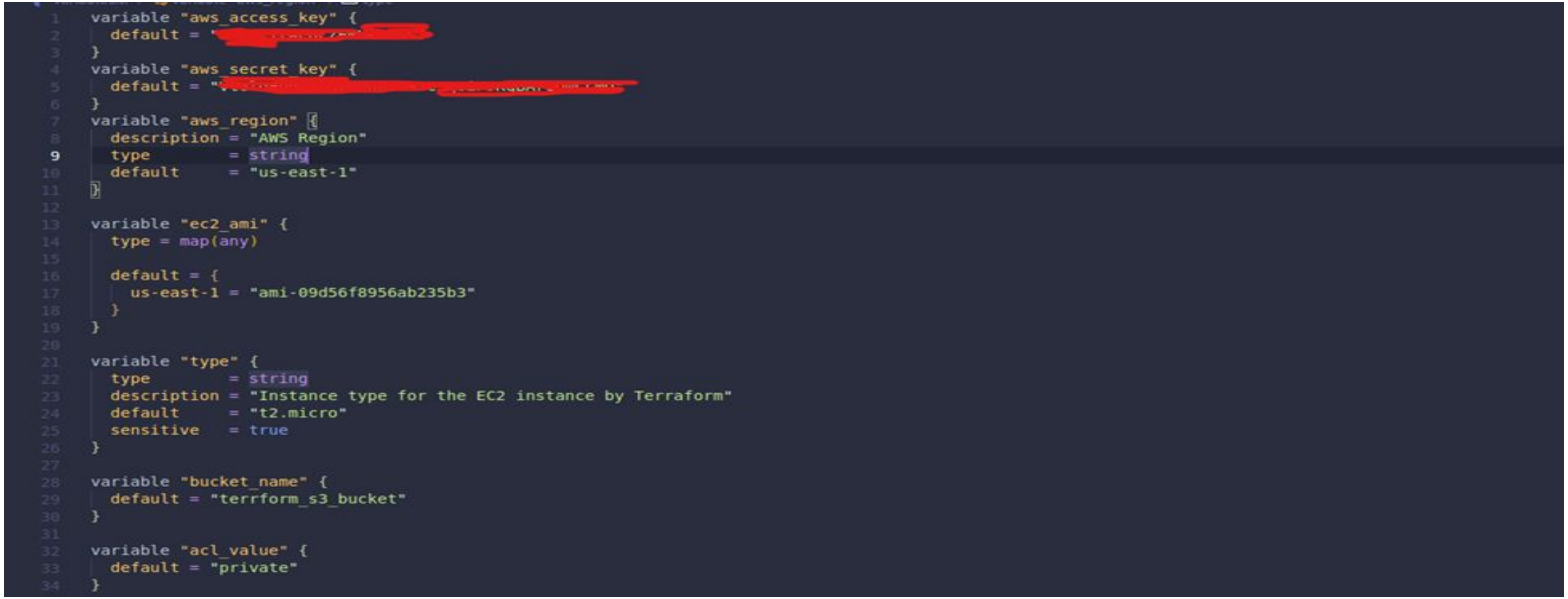


terraform.tf

```
provider "aws" {
  profile    = "default"
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
  region     = var.aws_region
}

resource "aws_instance" "terraform_ec2" {
  ami           = lookup(var.ec2_ami, var.aws_region)
  instance_type = var.type
  count         = 1
  tags = {
    Env  = "dev"
    Name = "terraform_ec2_instance-${count.index}"
  }
}

resource "aws_s3_bucket_acl" terraform_s3_bucket {
  bucket = var.bucket_name
  acl    = var.acl_value
}
```

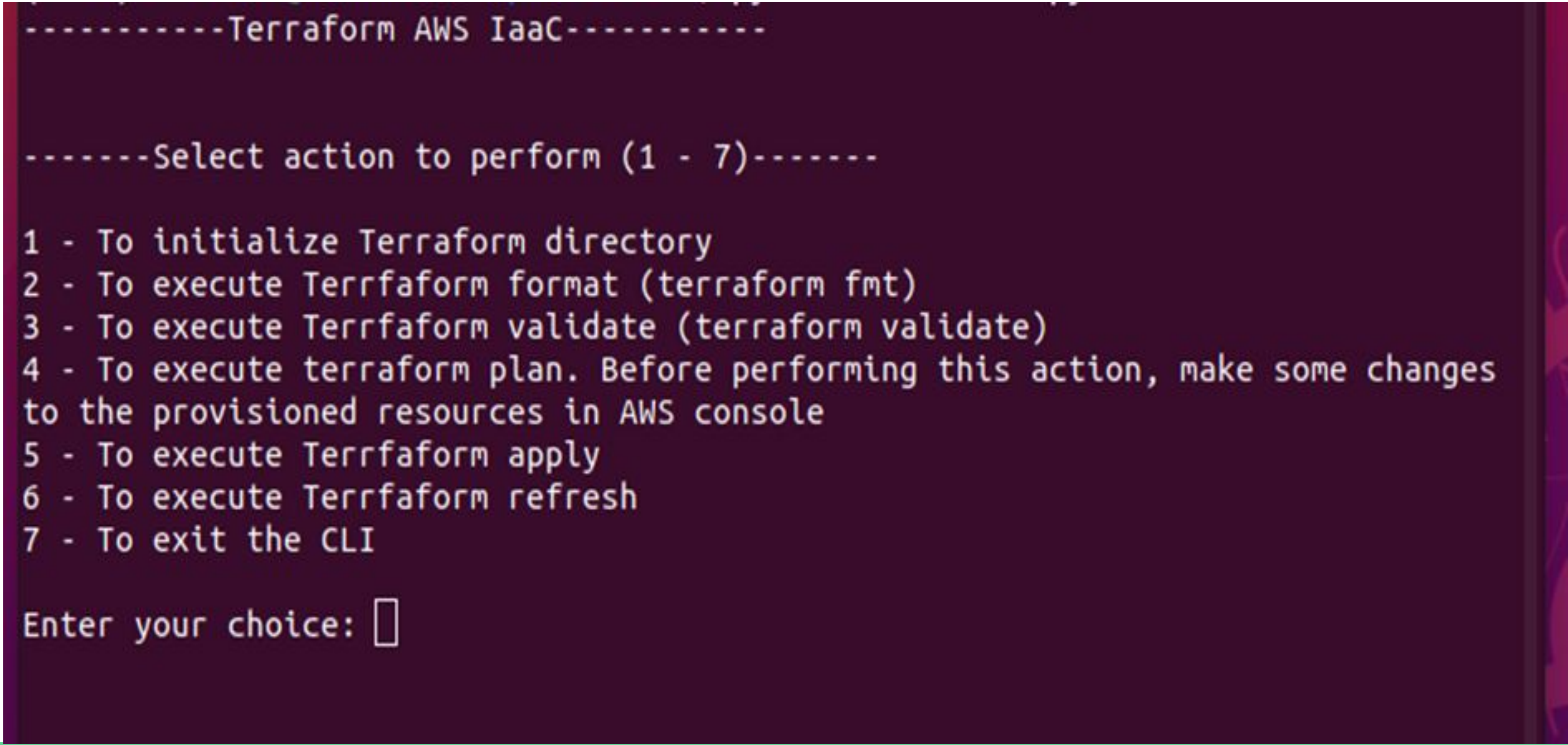
Terraform variable file

Terraform variables allow you to write configuration that is flexible and easier to re-use. Add a variable to define the instance name. **Create a new file called variables.tf with a block defining a new instance_name variable**.



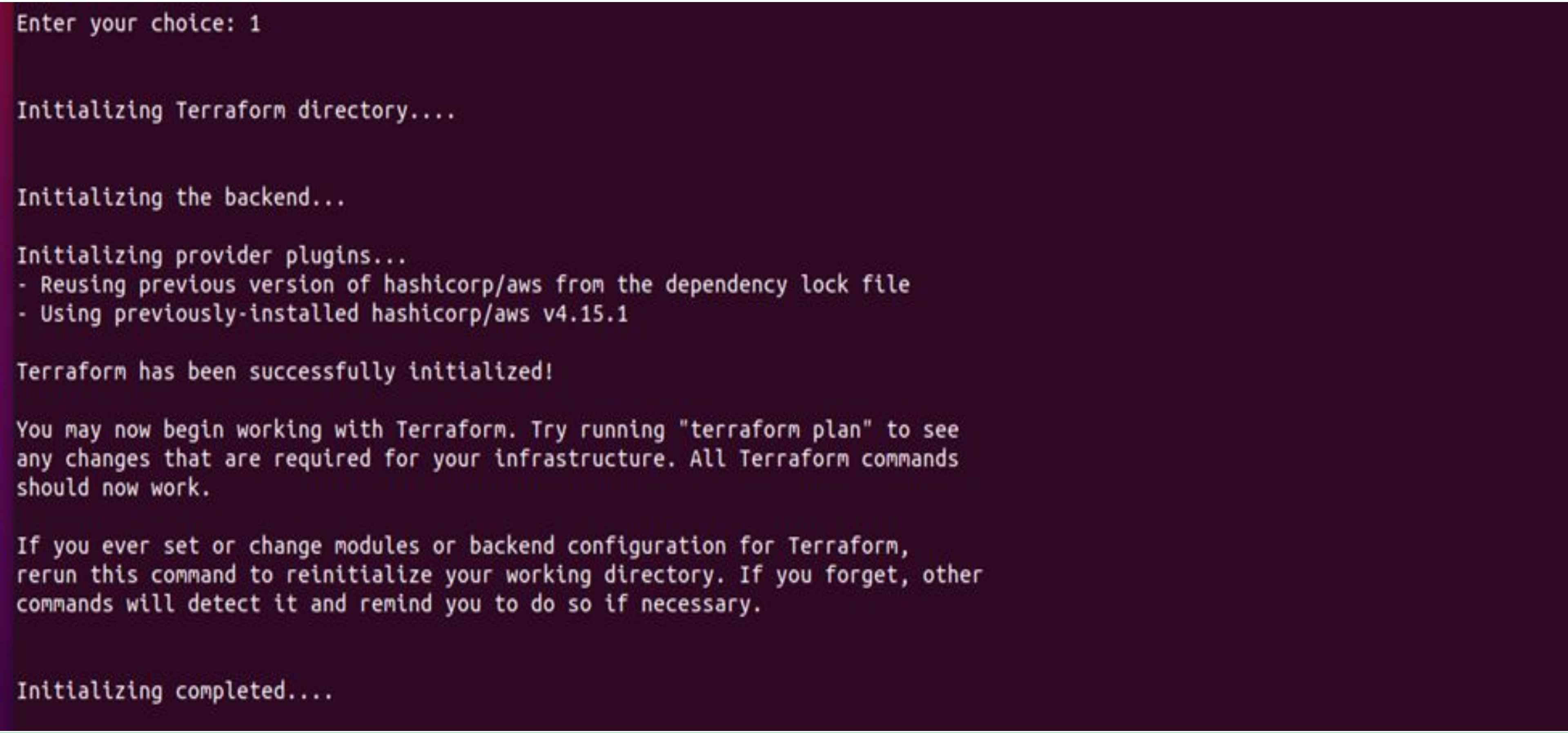
```
variable "aws_access_key" {
  default = "[redacted]
}
variable "aws_secret_key" {
  default = "[redacted]
}
variable "aws_region" {
  description = "AWS Region"
  type        = string
  default     = "us-east-1"
}

variable "ec2_ami" {
  type = map(any)

  default = {
    us-east-1 = "ami-09d56f8956ab235b3"
  }
}

variable "type" {
  type        = string
  description = "Instance type for the EC2 instance by Terraform"
  default     = "t2.micro"
  sensitive   = true
}

variable "bucket_name" {
  default = "terrform_s3_bucket"
}

variable "acl_value" {
  default = "private"
}
```

CLI Tool - Main Menu



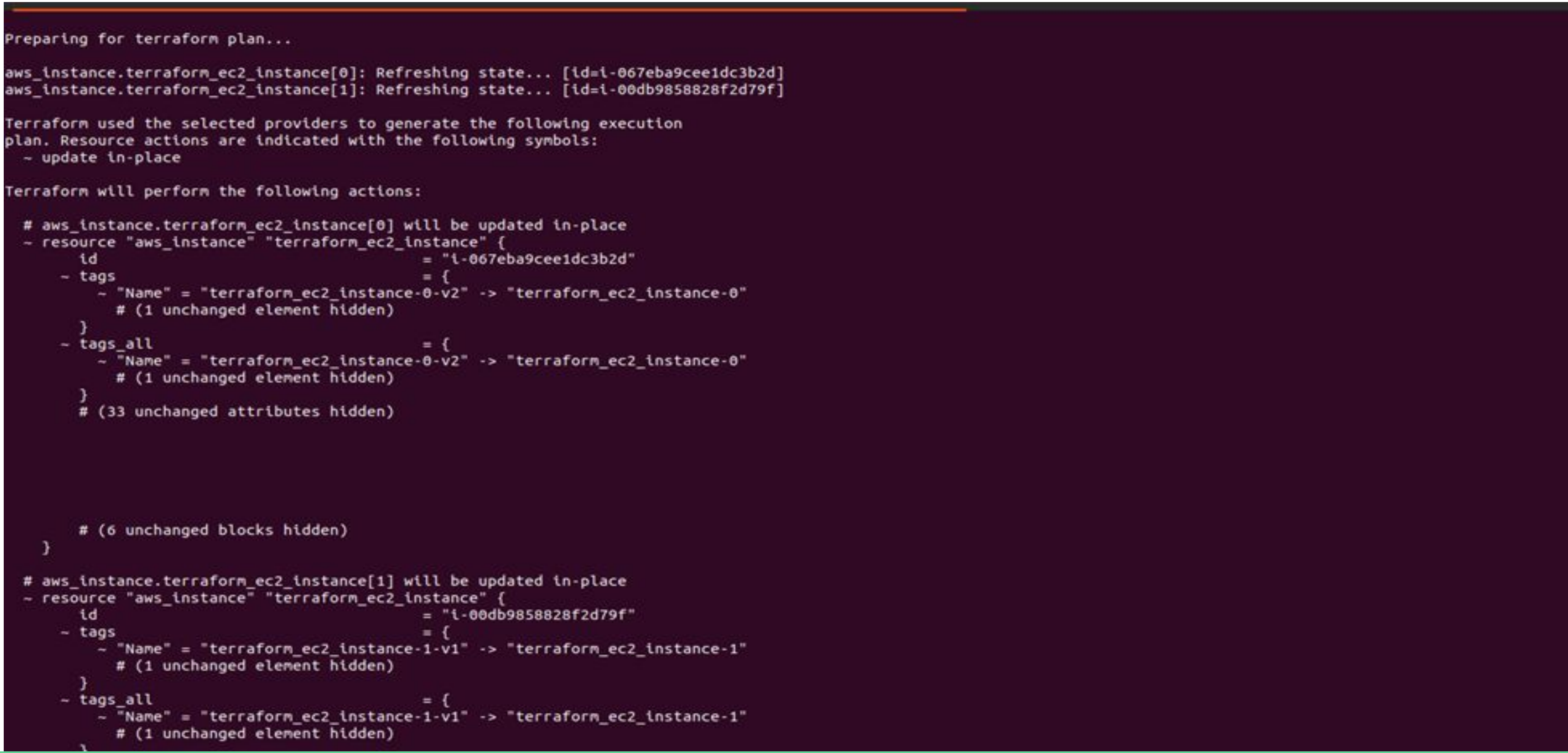
```
-----------Terraform AWS IaaC-----------



-------Select action to perform (1 - 7)-------

1 - To initialize Terraform directory
2 - To execute Terrfaform format (terraform fmt)
3 - To execute Terrfaform validate (terraform validate)
4 - To execute terraform plan. Before performing this action, make some changes
to the provisioned resources in AWS console
5 - To execute Terrfaform apply
6 - To execute Terrfaform refresh
7 - To exit the CLI

Enter your choice: 
```

Terraform Directory Initialisation



```
Enter your choice: 1


Initializing Terraform directory....


Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.15.1

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.


Initializing completed....
```

Drift in EC2 instance created in AWS with Terraform provider



```
Preparing for terraform plan...

aws_instance.terraform_ec2_instance[0]: Refreshing state... [id=i-067eba9cee1dc3b2d]
aws_instance.terraform_ec2_instance[1]: Refreshing state... [id=i-00db9858828f2d79f]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_instance.terraform_ec2_instance[0] will be updated in-place
  ~ resource "aws_instance" "terraform_ec2_instance" {
        id                                   = "i-067eba9cee1dc3b2d"
      ~ tags                                 = {
          ~ "Name" = "terraform_ec2_instance-0-v2" -> "terraform_ec2_instance-0"
            # (1 unchanged element hidden)
        }
      ~ tags_all                             = {
          ~ "Name" = "terraform_ec2_instance-0-v2" -> "terraform_ec2_instance-0"
            # (1 unchanged element hidden)
        }
        # (33 unchanged attributes hidden)






        # (6 unchanged blocks hidden)
    }

  # aws_instance.terraform_ec2_instance[1] will be updated in-place
  ~ resource "aws_instance" "terraform_ec2_instance" {
        id                                   = "i-00db9858828f2d79f"
      ~ tags                                 = {
          ~ "Name" = "terraform_ec2_instance-1-v1" -> "terraform_ec2_instance-1"
            # (1 unchanged element hidden)
        }
      ~ tags_all                             = {
          ~ "Name" = "terraform_ec2_instance-1-v1" -> "terraform_ec2_instance-1"
            # (1 unchanged element hidden)
```