Assignment no. 02

Q. Define input variable, query data with output and store remote state of terraform.

Terraform supports a few different variable formats. Depending on the usage, the variables are generally divided into inputs and outputs.

- The input variables are used to define values that configure your infrastructure. These values can be used again and again without having to remember their every occurrence in the event it needs to be updated.

- Output variables, in contrast, are used to get information about the infrastructure after deployment. These can be useful for passing on information such as IP addresses for connecting to the server.

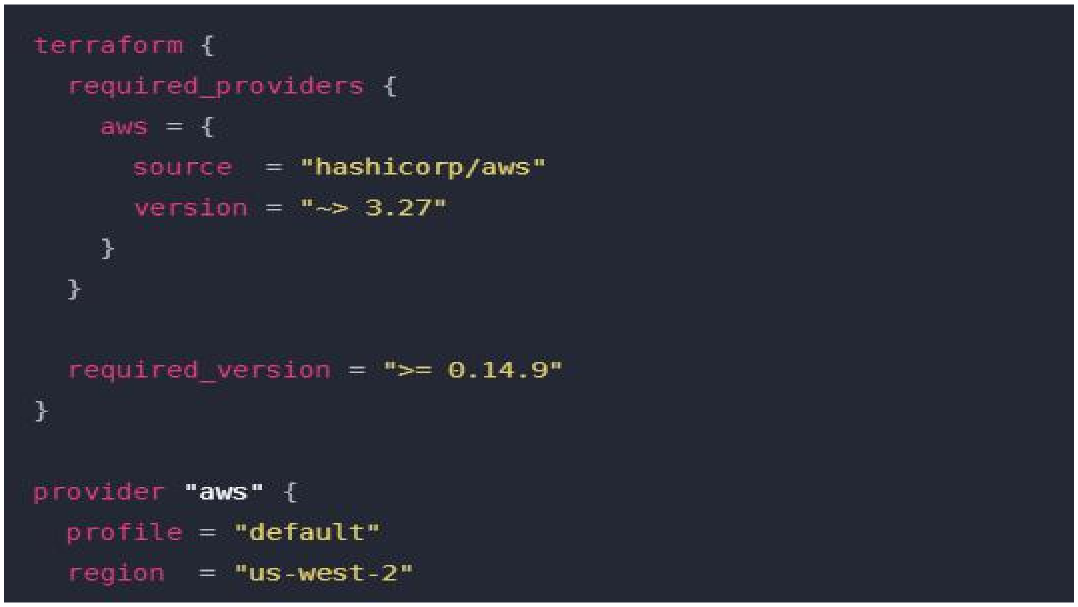
Input variables

Input variables are usually defined by stating a name, type and a default value. However, the type and default values are not strictly necessary. Terraform can deduct the type of the variable from the default or input value.

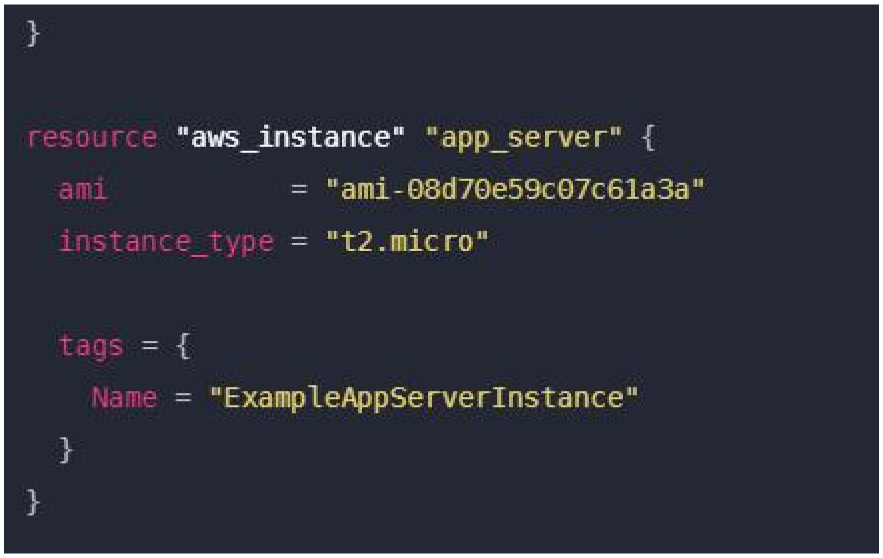
Variables can be predetermined in a file or included in the command-line options. As such, the simplest variable is just a name while the type and value are selected based on the input.
The input variables,use a couple of different types: strings, lists, maps, and boolean.

Output variables

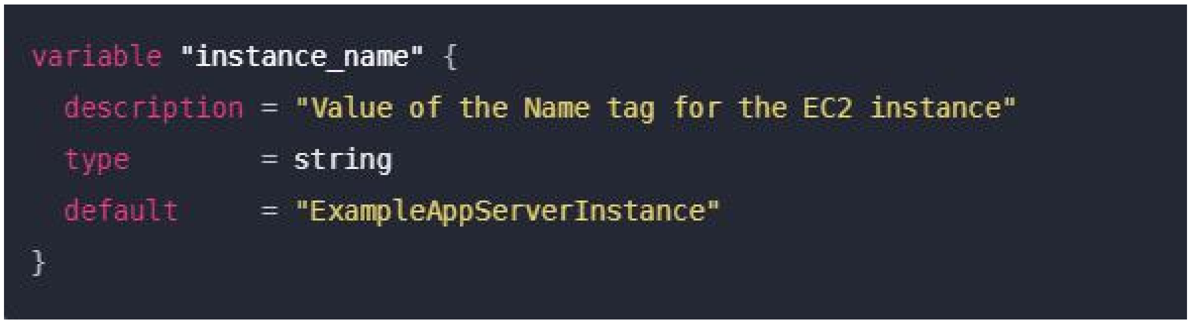
Output variables provide a convenient way to get useful information about your infrastructure. As you might have noticed, much of the server details are calculated at deployment and only become available afterwards. Using output variables you can extract any server-specific values including the calculated details.



```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
```



```
}

resource "aws_instance" "app_server" {
  ami           = "ami-08d70e59c07c61a3a"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

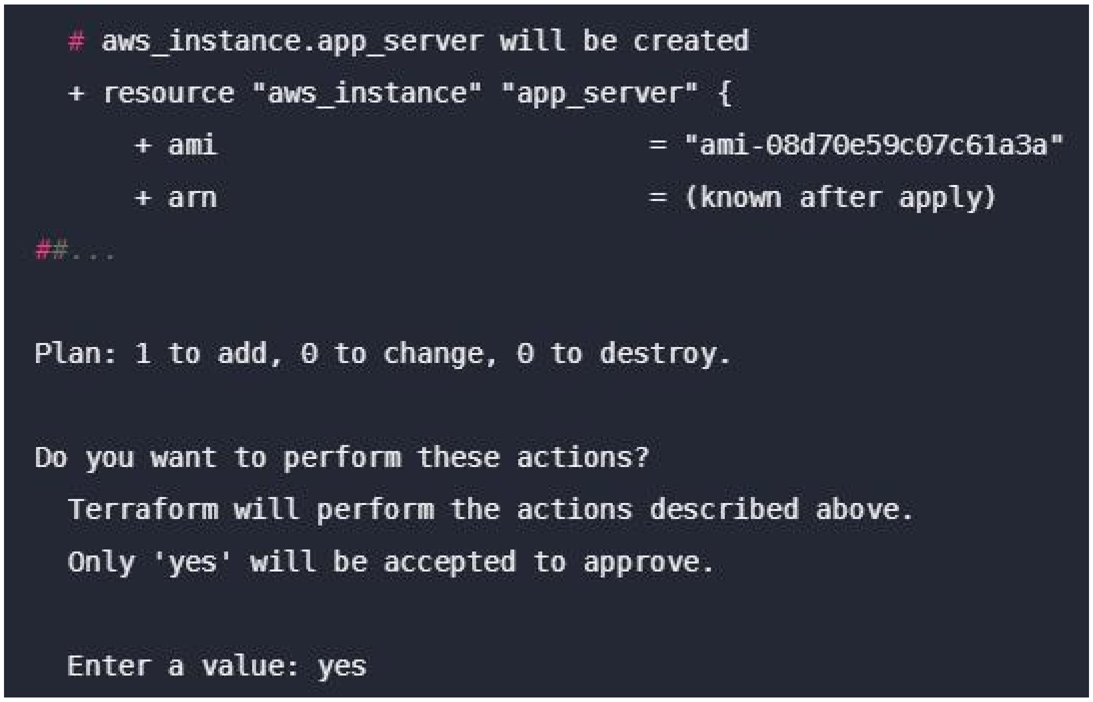


```
variable "instance_name" {
  description = "Value of the Name tag for the EC2 instance"
  type        = string
  default     = "ExampleAppServerInstance"
}
```

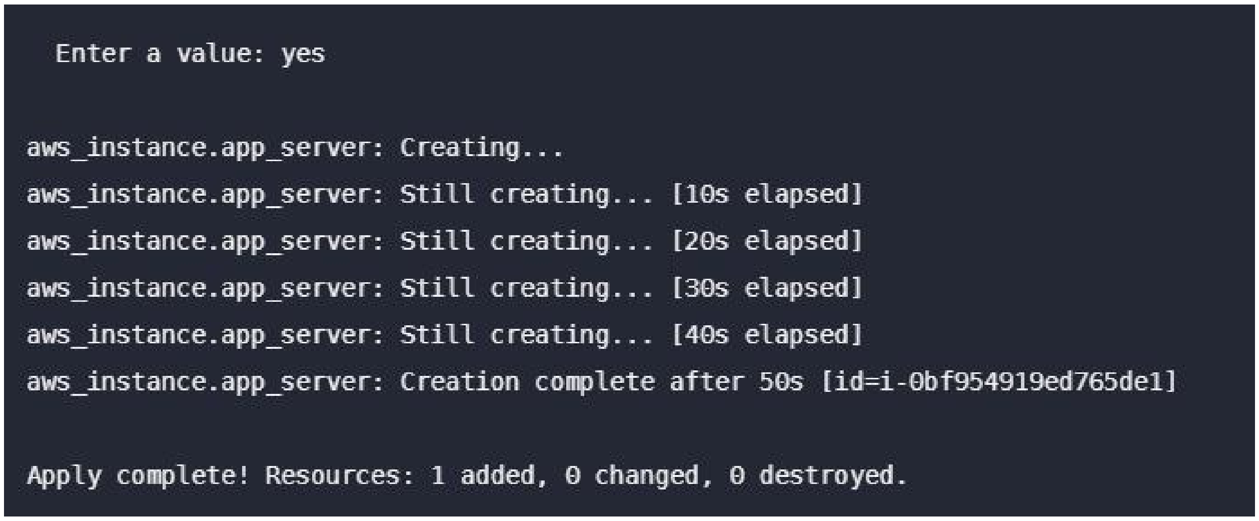
```
 resource "aws_instance" "app_server" {
   ami           = "ami-08d70e59c07c61a3a"
   instance_type = "t2.micro"

   tags = {
-    Name = "ExampleAppServerInstance"
+    Name = var.instance_name
   }
 }
```

Apply your configuration

```
$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:
```



```
  # aws_instance.app_server will be created
  + resource "aws_instance" "app_server" {
      + ami                          = "ami-08d70e59c07c61a3a"
      + arn                          = (known after apply)
##...

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```



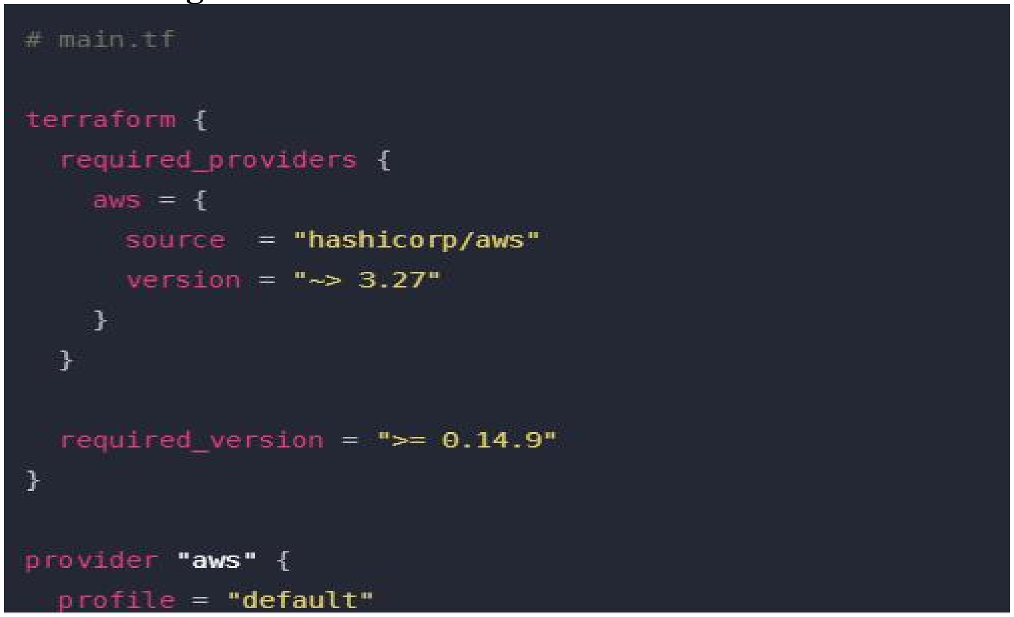
```
  Enter a value: yes

aws_instance.app_server: Creating...
aws_instance.app_server: Still creating... [10s elapsed]
aws_instance.app_server: Still creating... [20s elapsed]
aws_instance.app_server: Still creating... [30s elapsed]
aws_instance.app_server: Still creating... [40s elapsed]
aws_instance.app_server: Creation complete after 50s [id=i-0bf954919ed765de1]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Query Data

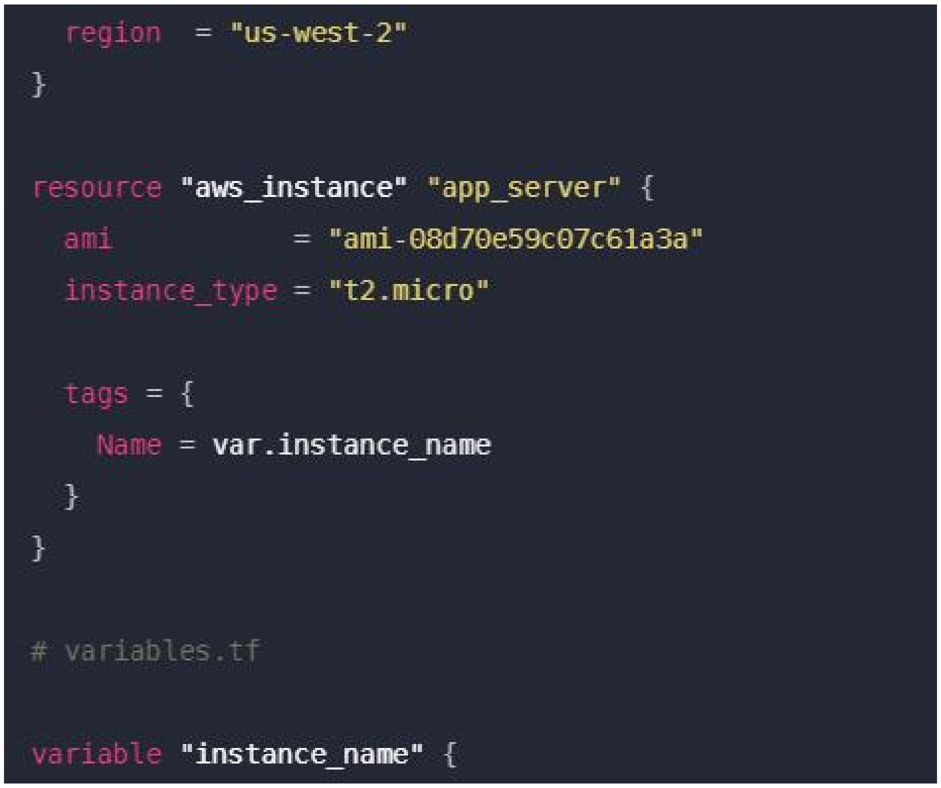
Cloud infrastructure, applications, and services emit data, which Terraform can query and act on using data sources. Terraform uses data sources to fetch information from cloud provider APIs, such as disk image IDs, or information about the rest of your infrastructure through the outputs of other Terraform configurations.

Data sources allow you to load data from APIs or other Terraform workspaces. You can use this data to make your project's configuration more flexible, and to connect workspaces that manage different parts of your infrastructure. You can also use data sources to connect and share data between workspaces in Terraform Cloud and Terraform Enterprise.

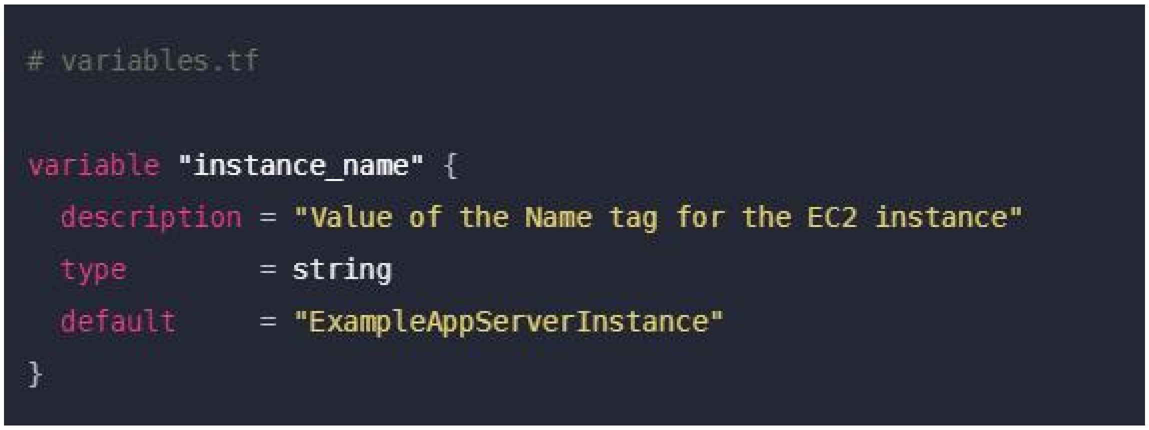
Initial configuration



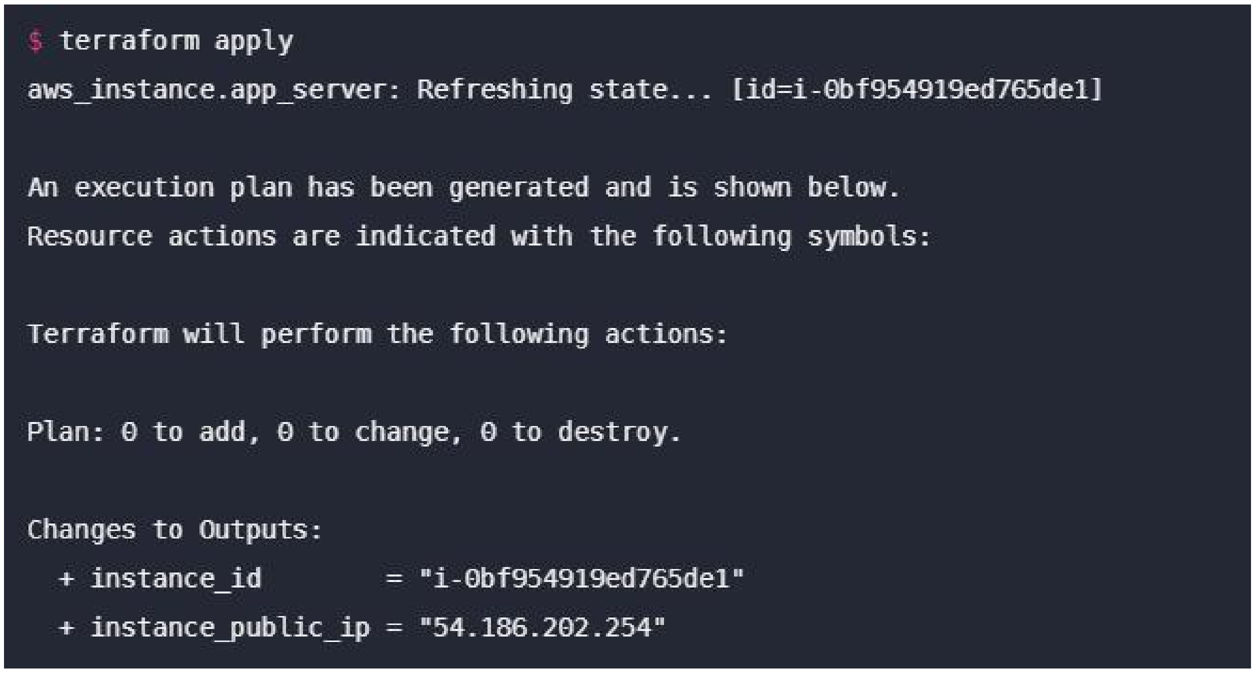
```
# main.tf

terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}

provider "aws" {
  profile = "default"
```



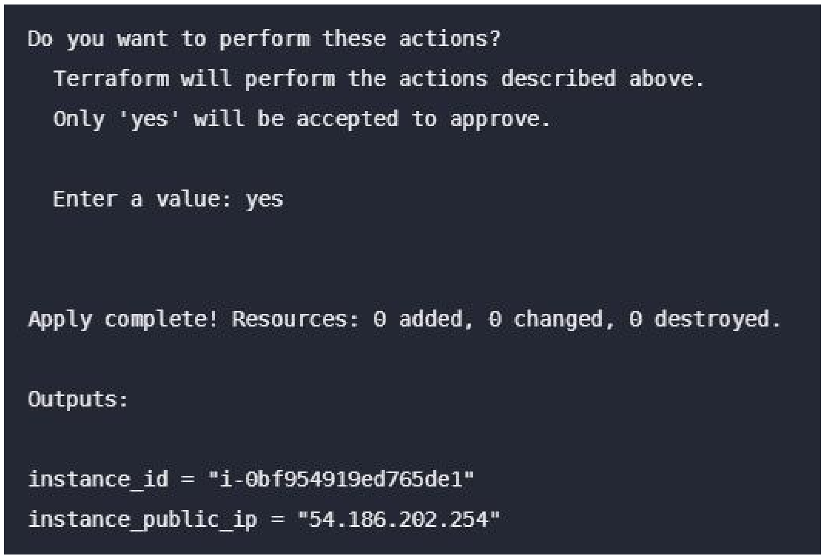
```
  region  = "us-west-2"
}

resource "aws_instance" "app_server" {
  ami           = "ami-08d70e59c07c61a3a"
  instance_type = "t2.micro"

  tags = {
    Name = var.instance_name
  }
}

# variables.tf

variable "instance_name" {
```



```
# variables.tf

variable "instance_name" {
  description = "Value of the Name tag for the EC2 instance"
  type        = string
  default     = "ExampleAppServerInstance"
}
```

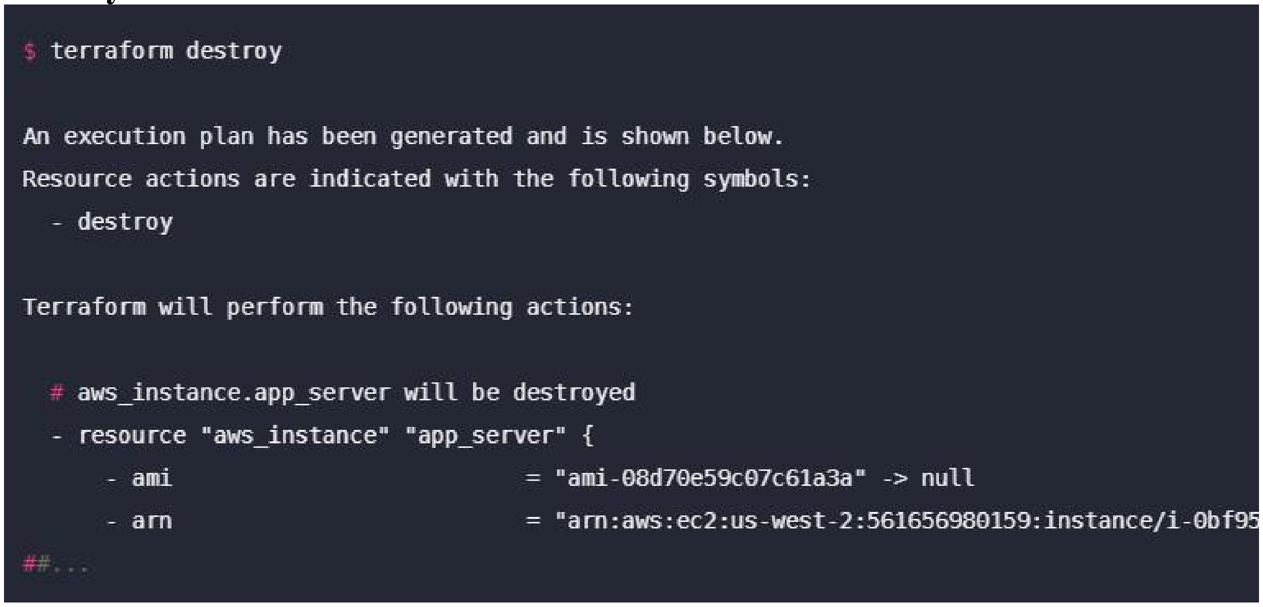


```
$ terraform apply
aws_instance.app_server: Refreshing state... [id=i-0bf954919ed765de1]

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

Terraform will perform the following actions:

Plan: 0 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_id        = "i-0bf954919ed765de1"
  + instance_public_ip = "54.186.202.254"
```

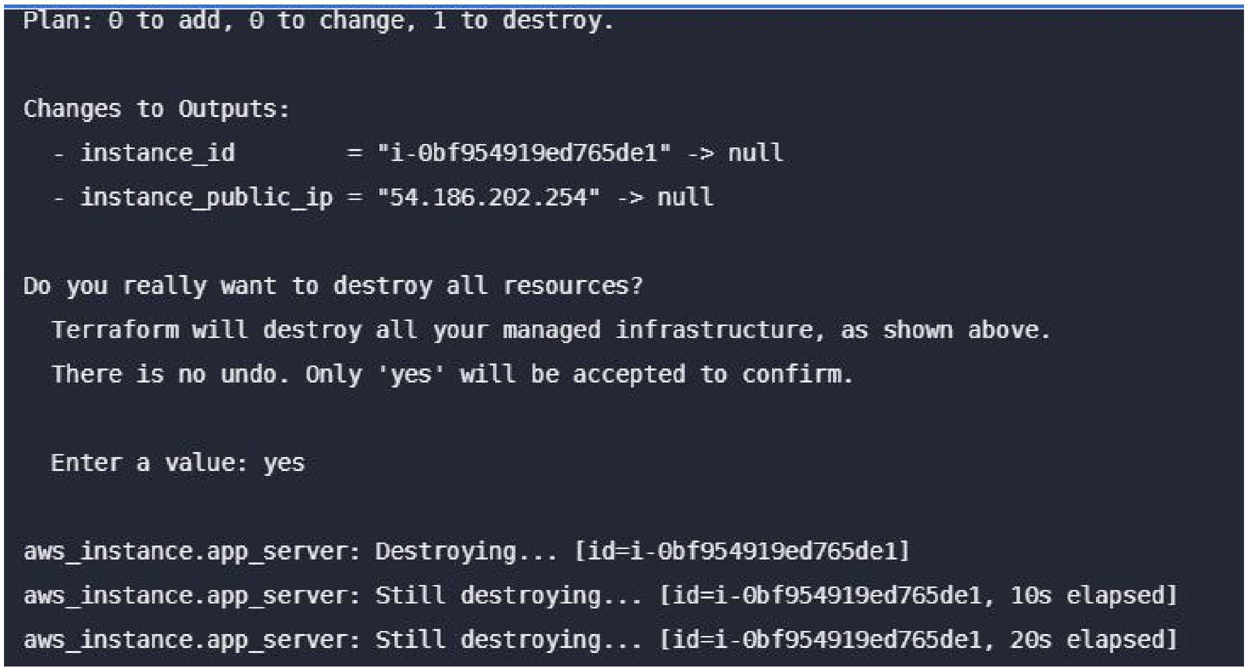


```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes


Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

instance_id = "i-0bf954919ed765de1"
instance_public_ip = "54.186.202.254"
```

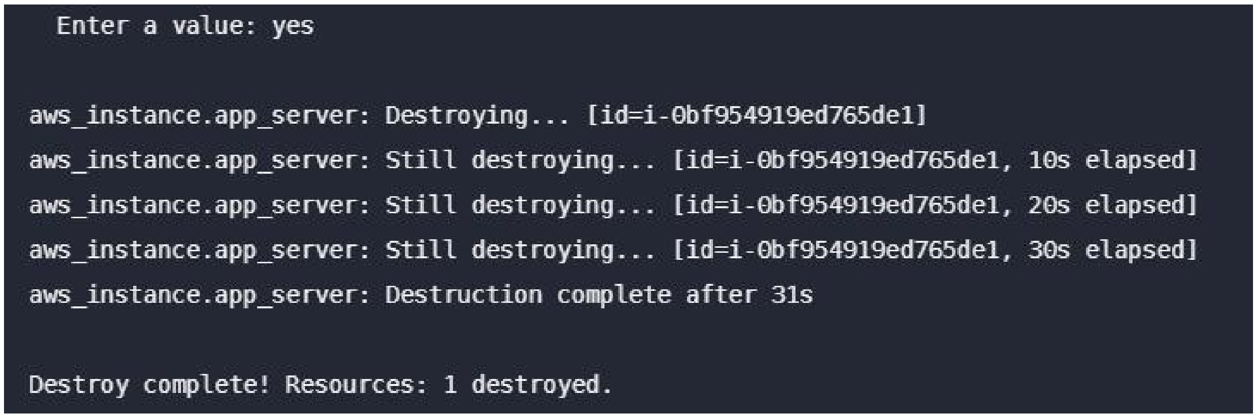
Destroy infrastructure



```
$ terraform destroy

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_instance.app_server will be destroyed
  - resource "aws_instance" "app_server" {
      - ami                          = "ami-08d70e59c07c61a3a" -> null
      - arn                          = "arn:aws:ec2:us-west-2:561656980159:instance/i-0bf95
##...
```



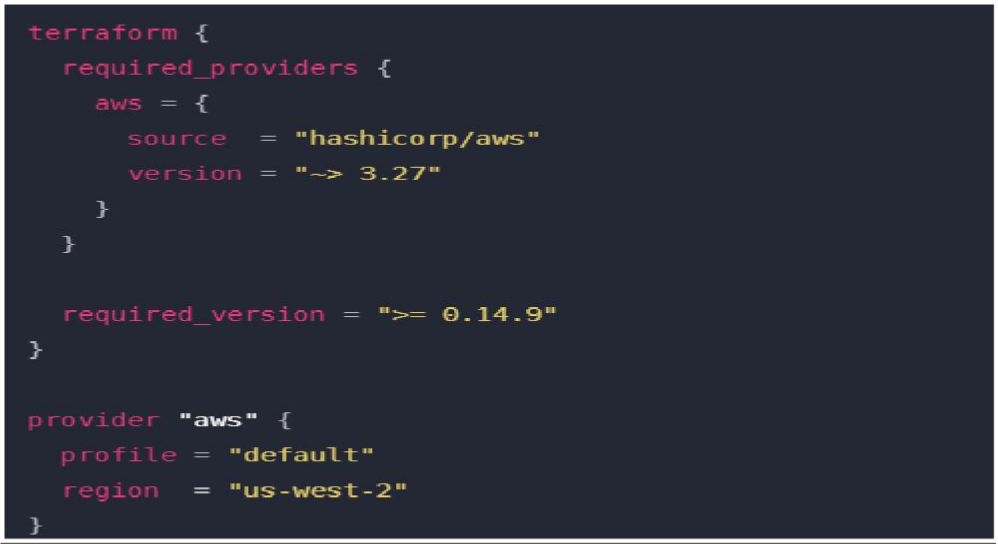
```
Plan: 0 to add, 0 to change, 1 to destroy.

Changes to Outputs:
  - instance_id        = "i-0bf954919ed765de1" -> null
  - instance_public_ip = "54.186.202.254" -> null

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.app_server: Destroying... [id=i-0bf954919ed765de1]
aws_instance.app_server: Still destroying... [id=i-0bf954919ed765de1, 10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0bf954919ed765de1, 20s elapsed]
```



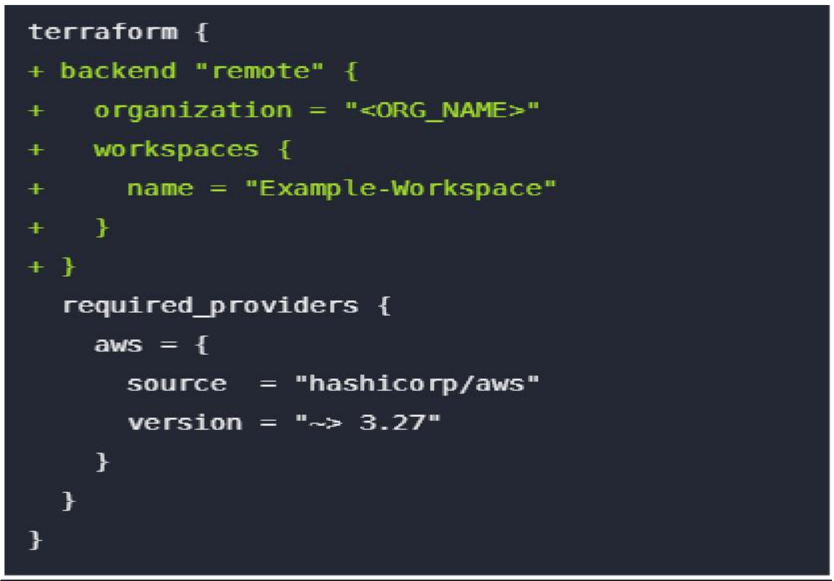
```
  Enter a value: yes

aws_instance.app_server: Destroying... [id=i-0bf954919ed765de1]
aws_instance.app_server: Still destroying... [id=i-0bf954919ed765de1, 10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0bf954919ed765de1, 20s elapsed]
aws_instance.app_server: Still destroying... [id=i-0bf954919ed765de1, 30s elapsed]
aws_instance.app_server: Destruction complete after 31s

Destroy complete! Resources: 1 destroyed.
```

Store Remote State

Remote backends allow Terraform to use a shared storage space for state data. The Terraform Cloud remote backend also allows teams to easily version, audit, and collaborate on infrastructure changes. Terraform Cloud also securely stores variables, including API tokens and access keys. It provides a safe, stable environment for long-running Terraform processes.

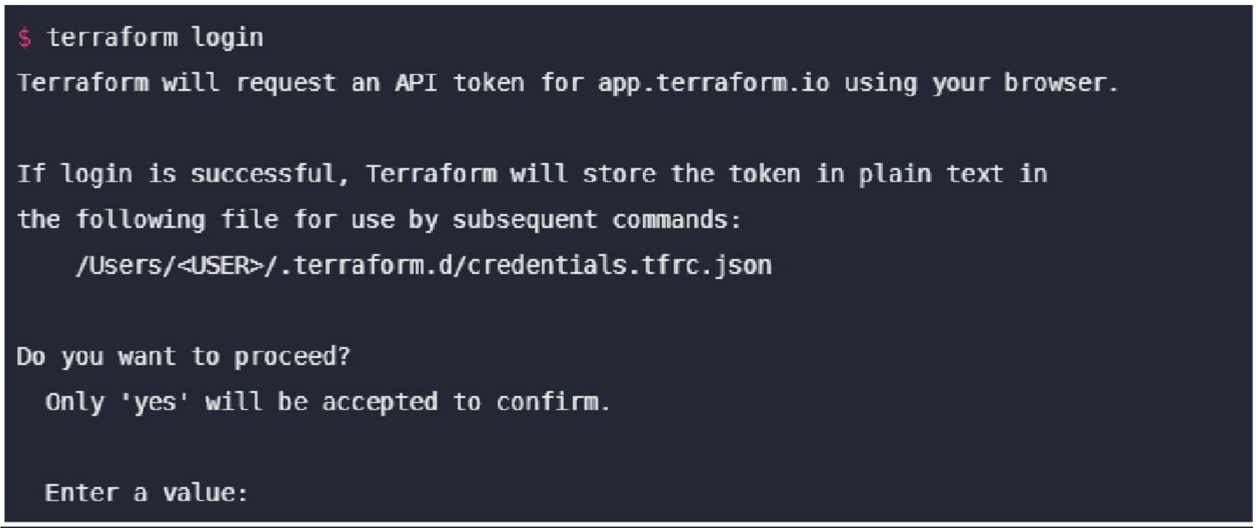


```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
}
```



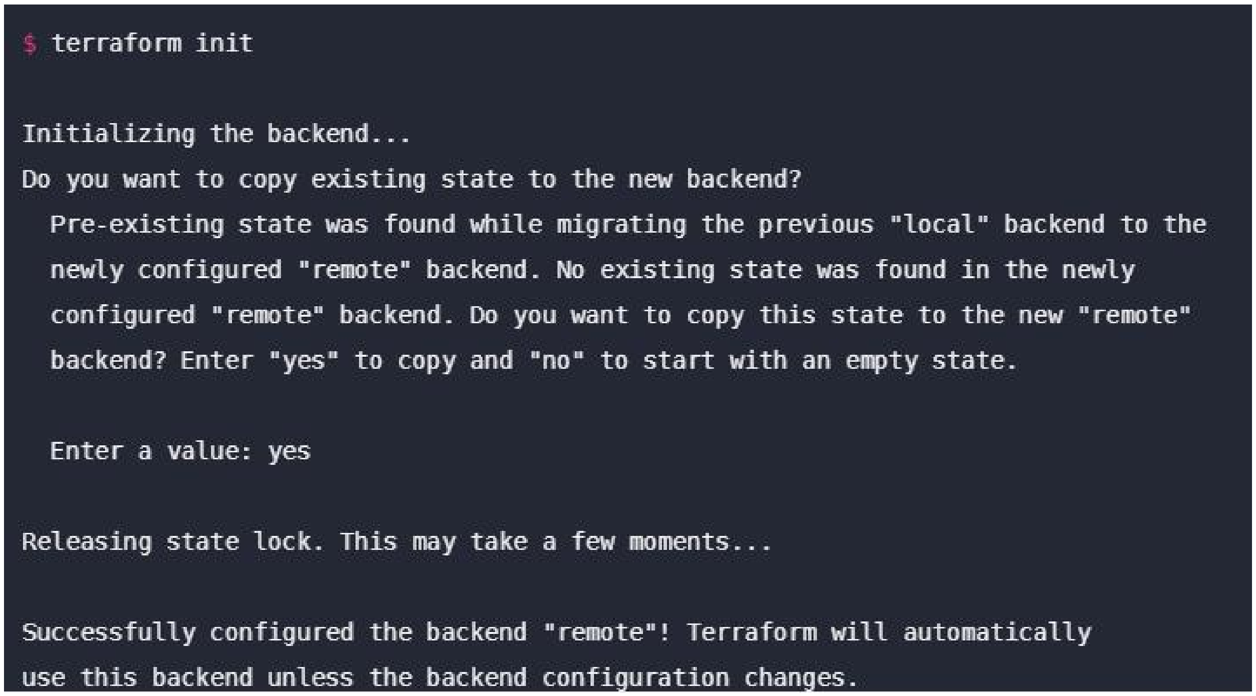
```
terraform {
+ backend "remote" {
+   organization = "<ORG_NAME>"
+   workspaces {
+     name = "Example-Workspace"
+   }
+ }
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }
}
```

Login to terraform

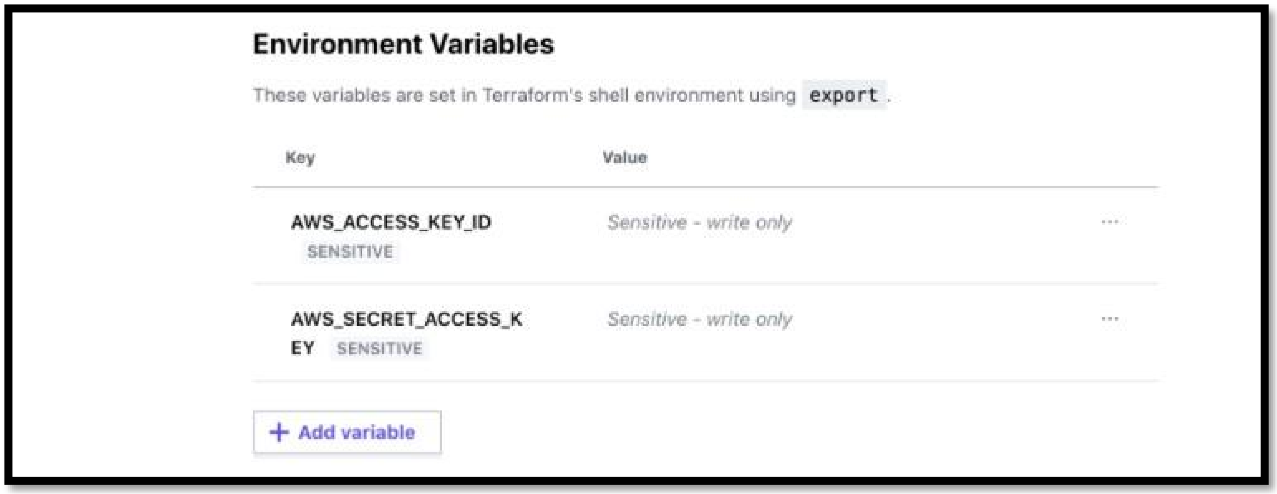


```
$ terraform login
Terraform will request an API token for app.terraform.io using your browser.

If login is successful, Terraform will store the token in plain text in
the following file for use by subsequent commands:
    /Users/<USER>/.terraform.d/credentials.tfrc.json

Do you want to proceed?
  Only 'yes' will be accepted to confirm.

  Enter a value:
```

Initialize Terraform



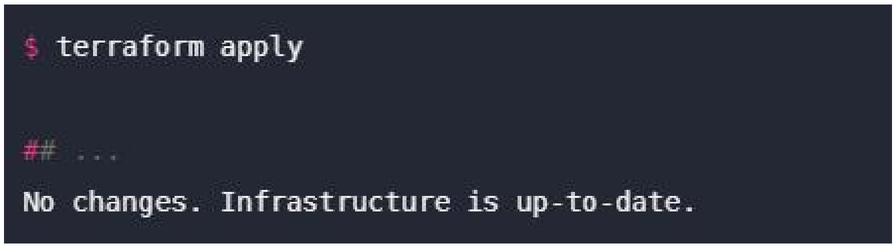
```
$ terraform init

Initializing the backend...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "local" backend to the
  newly configured "remote" backend. No existing state was found in the newly
  configured "remote" backend. Do you want to copy this state to the new "remote"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes

Releasing state lock. This may take a few moments...

Successfully configured the backend "remote"! Terraform will automatically
use this backend unless the backend configuration changes.
```



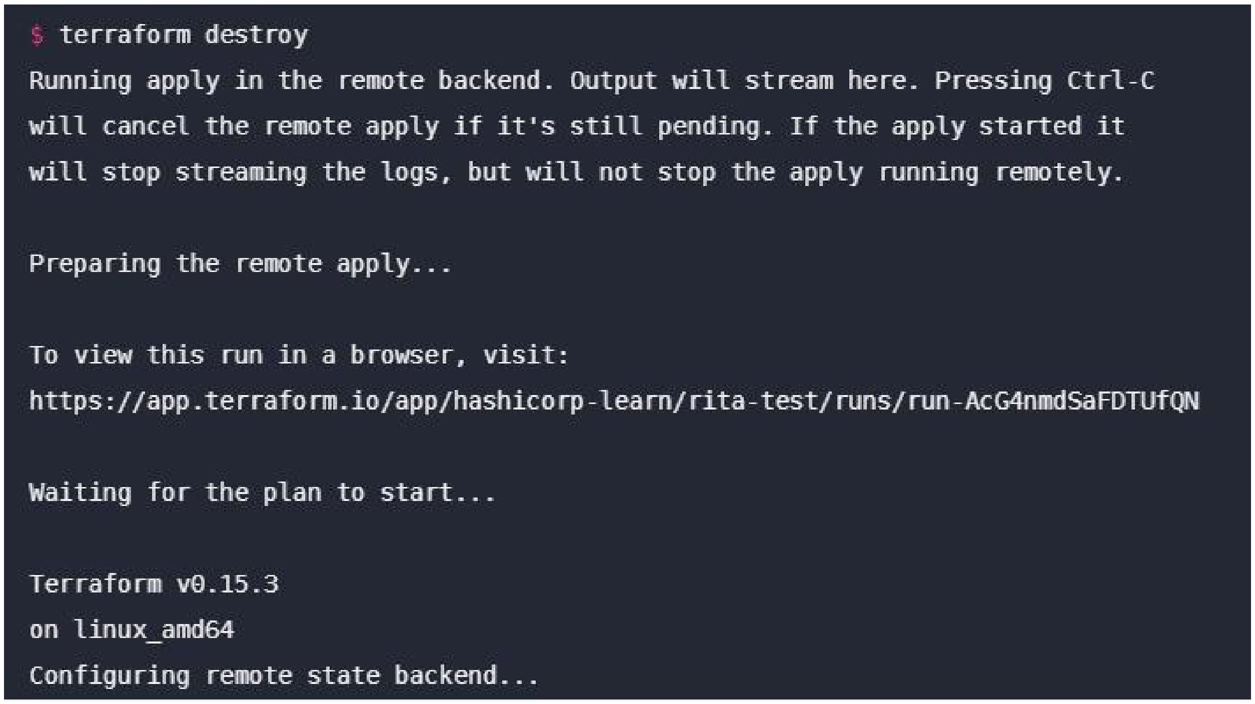
Environment Variables

These variables are set in Terraform's shell environment using `export`.

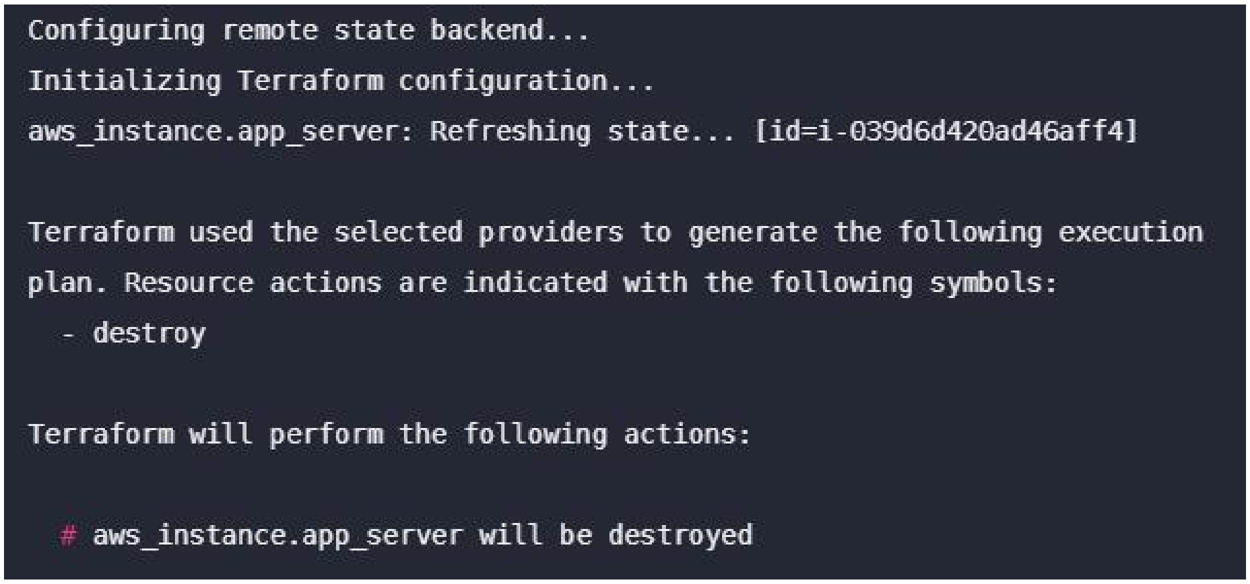
Key	Value
AWS_ACCESS_KEY_ID SENSITIVE	*Sensitive - write only*
AWS_SECRET_ACCESS_KEY SENSITIVE	*Sensitive - write only*



```
$ terraform apply

## ...
No changes. Infrastructure is up-to-date.
```



```
$ terraform destroy
Running apply in the remote backend. Output will stream here. Pressing Ctrl-C
will cancel the remote apply if it's still pending. If the apply started it
will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/hashicorp-learn/rita-test/runs/run-AcG4nmdSaFDTUfQN

Waiting for the plan to start...

Terraform v0.15.3
on linux_amd64
Configuring remote state backend...
```



```
Configuring remote state backend...
Initializing Terraform configuration...
aws_instance.app_server: Refreshing state... [id=i-039d6d420ad46aff4]

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

Terraform will perform the following actions:

  # aws_instance.app_server will be destroyed
```

Conclusion :
Hence, We Successfully Defined input variable, query data with output and store remote state of terraform.