Terraform overview

A Practical Guide to Managing Linux Systems Using Terraform

Terraform
ORACLE Cloud Infrastructure
Microsoft Azure
amazon web services
Google Cloud

1. Introduction to Terraform

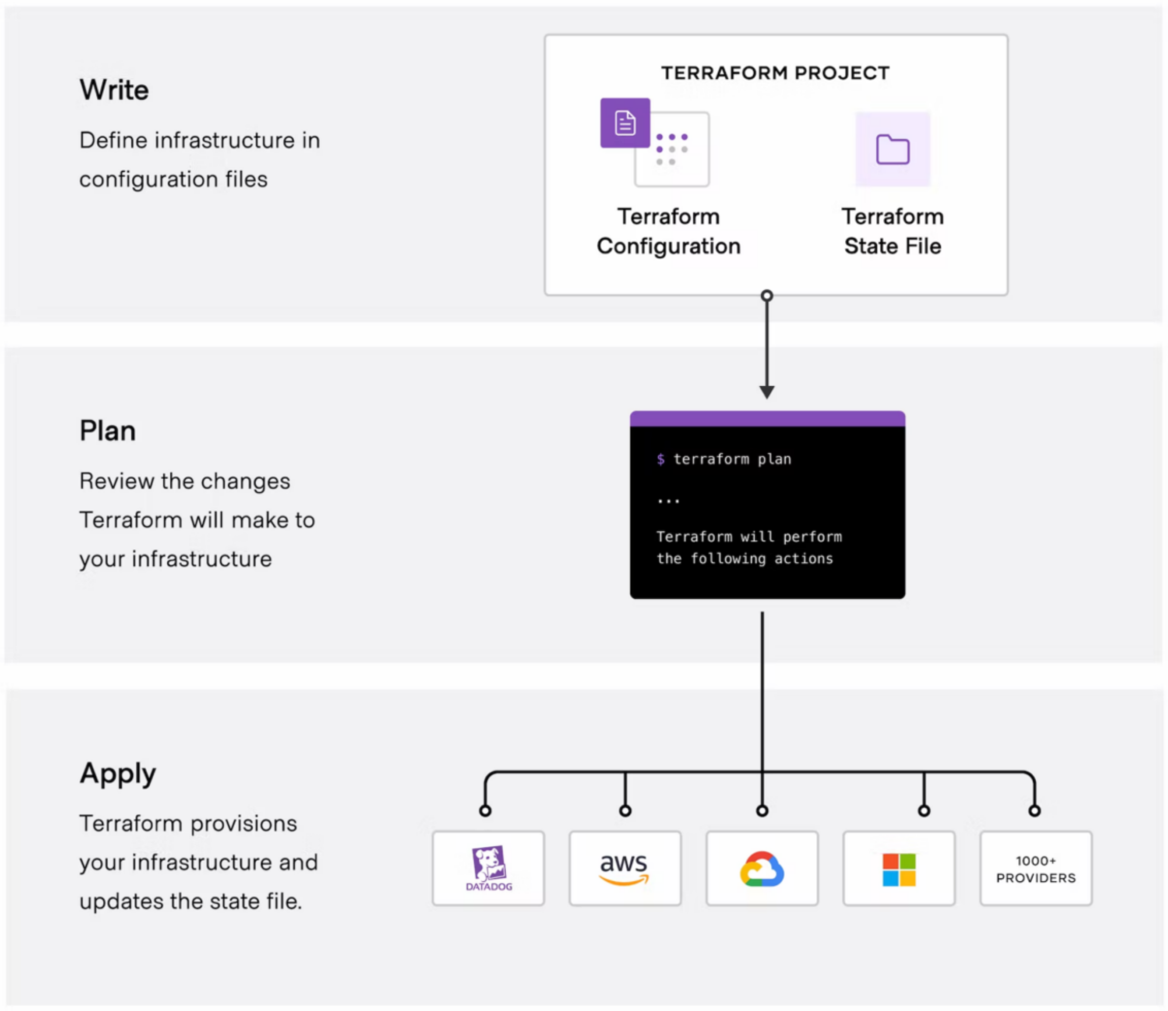
Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It enables users to define and provision infrastructure using a high-level configuration language known as HCL (HashiCorp Configuration Language). Terraform helps automate the deployment and management of infrastructure efficiently.

Key Features:

- **Declarative Language:** Define infrastructure in a readable format.
- **Supports Local and Remote Systems:** Can be used for both cloud and local environments.
- **Immutable Infrastructure:** Ensures consistency in infrastructure deployments.
- **Dependency Management:** Automatically understands and resolves dependencies between resources.

2. Terraform Workflow

Terraform operates in a structured workflow with the following key stages:


Write
Define infrastructure in configuration files
TERRAFORM PROJECT
Terraform Configuration
Terraform State File
Plan
Review the changes Terraform will make to your infrastructure
$ terraform plan
...
Terraform will perform the following actions
Apply
Terraform provisions your infrastructure and updates the state file.
DATADOG
aws
1000+ PROVIDERS

A. **Write:** Define infrastructure in .tf files.

B. **Initialize (terraform init)** – Sets up Terraform working directory.

C. **Plan (terraform plan)** – Displays changes before applying them.

D. **Apply (terraform apply)** – Creates or modifies infrastructure.

E. **Destroy (terraform destroy)** – Deletes infrastructure when no longer needed.

---

3. Terraform Basic Commands

- **terraform init →** Initializes the working directory.

- **terraform plan →** Shows what changes Terraform will make.
- **terraform apply →** Applies the configuration to create/update resources.
- **terraform destroy →** Deletes all managed infrastructure.

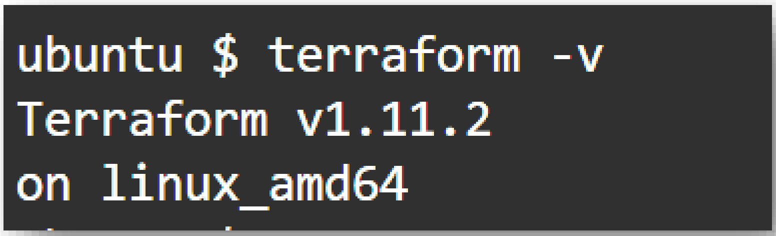
4. Terraform Configuration Basics

- **Providers →** Define which cloud provider (AWS, Azure, GCP) is used.
- **Resources →** Actual infrastructure components (VMs, databases, networks).
- **Variables →** Store reusable values (variable "region" { default = "us-east-1" }).
- **Outputs →** Display important information after deployment.
- **State File (terraform.tfstate) →** Stores the current state of infrastructure.

5. Installing Terraform on Linux

To install Terraform on a Linux system (Ubuntu/Debian-based):

- sudo apt update
- sudo apt install -y wget unzip
- wget https://releases.hashicorp.com/terraform/1.7.0/terraform_1.7.0_linux_amd64.zip
- unzip terraform_1.7.0_linux_amd64.zip
- sudo mv terraform /usr/local/bin/
- terraform --version



```
ubuntu $ terraform -v
Terraform v1.11.2
on linux_amd64
```

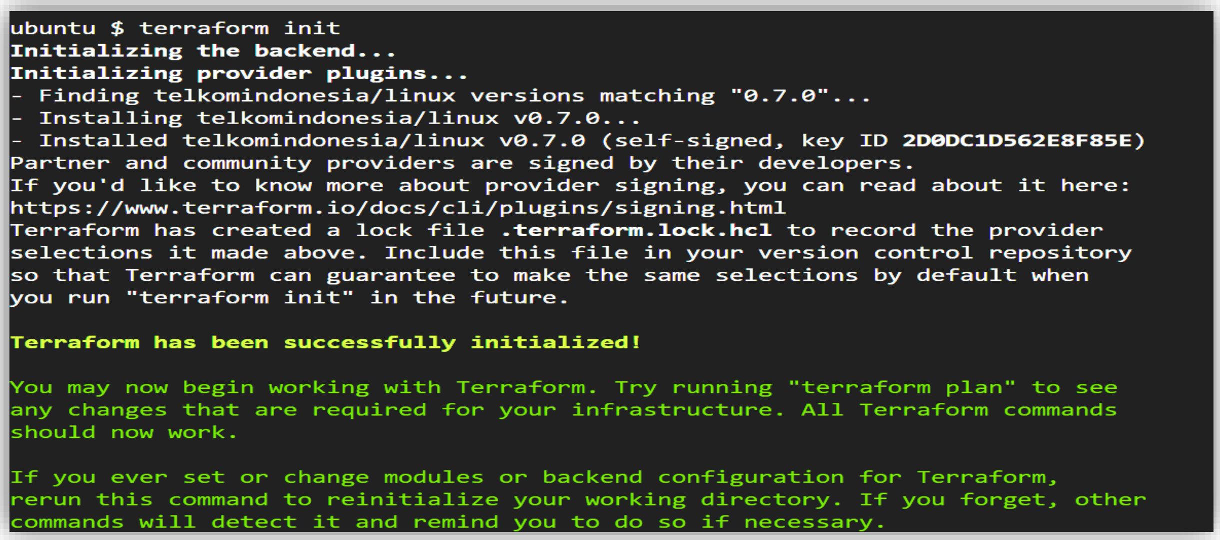
6. Basic Terraform Configuration for Local System

Terraform can be used to manage local system configurations, such as creating directories, files, installing software, managing users, and configuring services.

Example 1: Creating a Directory on Local System

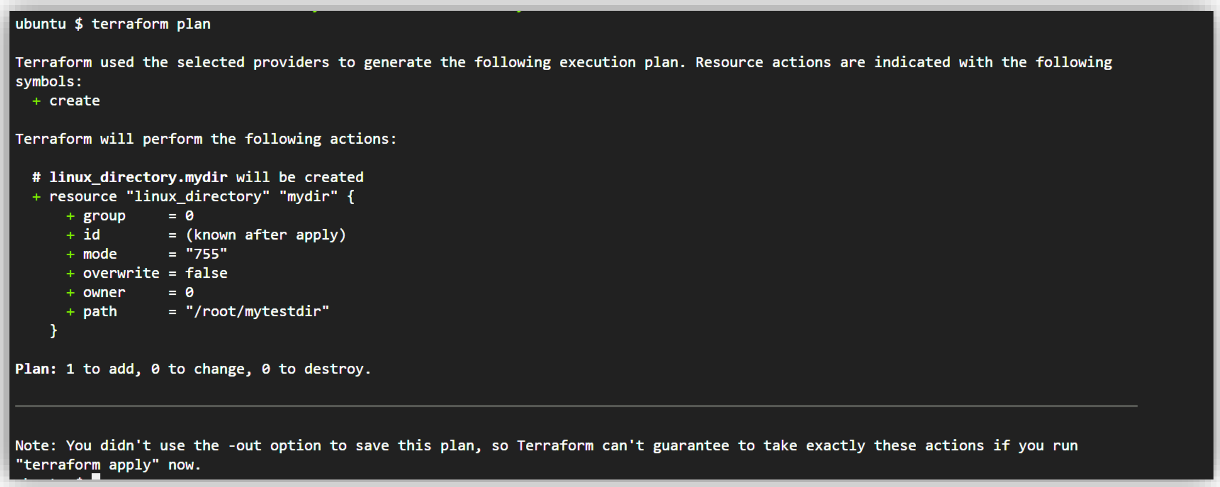
```
terraform {
  required_providers {
    linux = {
      source  = "TelkomIndonesia/linux"
      version = "0.7.0"
    }
  }
}

provider "linux" {
  host     = "localhost"
  port     = 22
  user     = "root"
  password = "1"
}

resource "linux_directory" "mydir" {
  path = "/root/mytestdir"
}
```

- **terraform init**



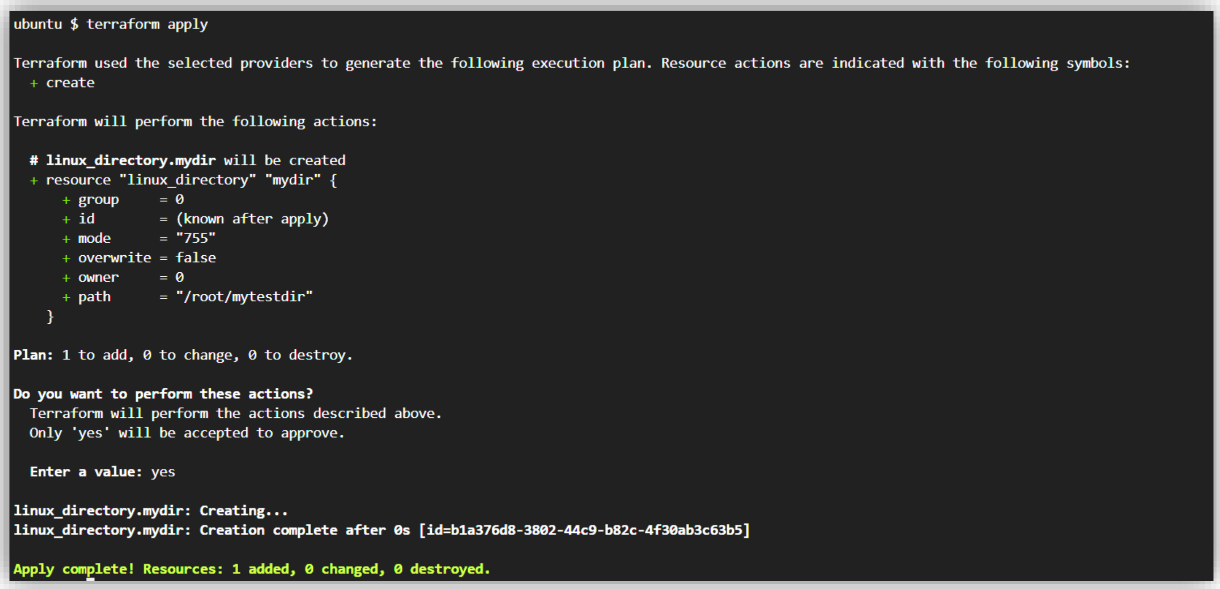
```
ubuntu $ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding telkomindonesia/linux versions matching "0.7.0"...
- Installing telkomindonesia/linux v0.7.0...
- Installed telkomindonesia/linux v0.7.0 (self-signed, key ID 2D0DC1D562E8F85E)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
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.
```

- **terraform plan**



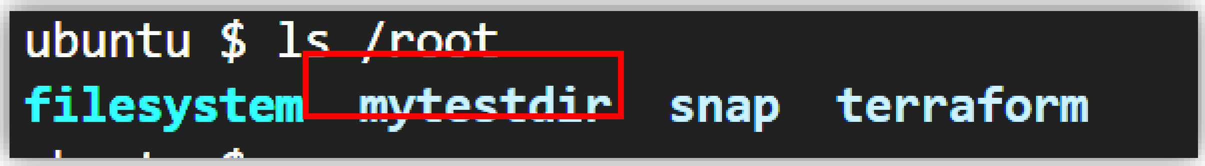
```
ubuntu $ terraform plan

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:

  # linux_directory.mydir will be created
  + resource "linux_directory" "mydir" {
      + group     = 0
      + id        = (known after apply)
      + mode      = "755"
      + overwrite = false
      + owner     = 0
      + path      = "/root/mytestdir"
    }

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

─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run
"terraform apply" now.
```

- **terraform plan**



```
ubuntu $ terraform apply

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:

  # linux_directory.mydir will be created
  + resource "linux_directory" "mydir" {
      + group     = 0
      + id        = (known after apply)
      + mode      = "755"
      + overwrite = false
      + owner     = 0
      + path      = "/root/mytestdir"
    }

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

linux_directory.mydir: Creating...
linux_directory.mydir: Creation complete after 0s [id=b1a376d8-3802-44c9-b82c-4f30ab3c63b5]

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

<u>Verify :- ls</u>

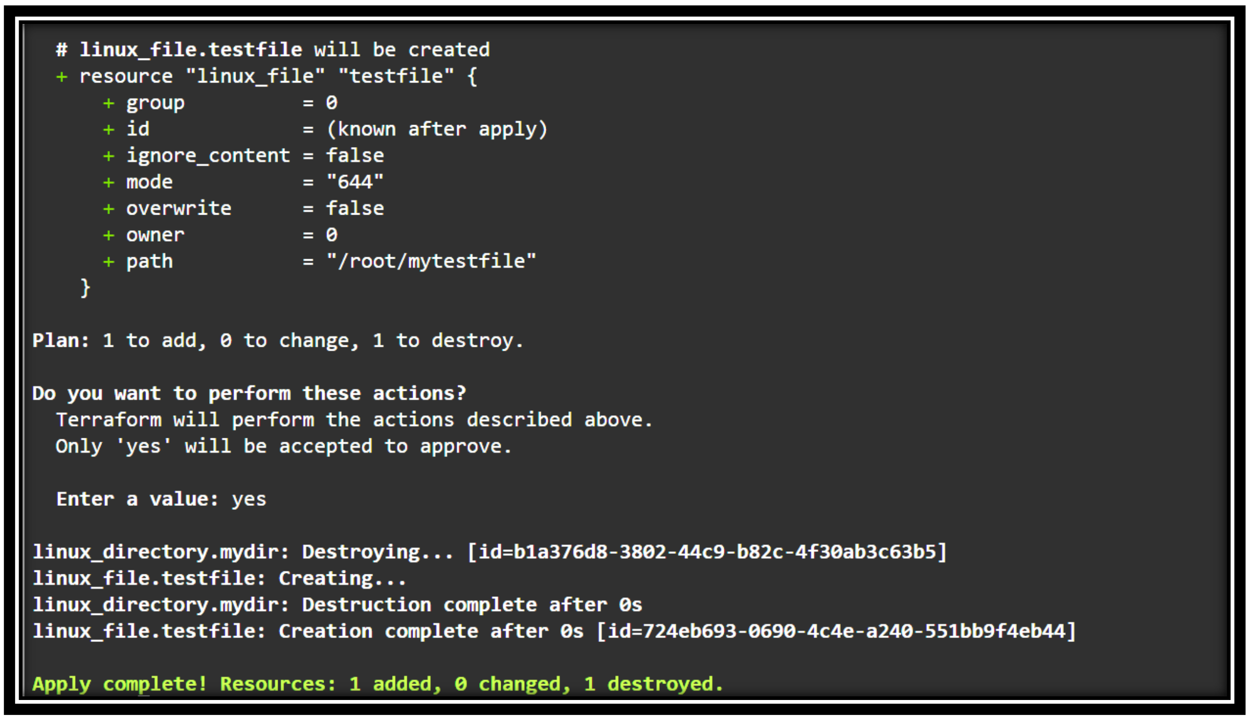


```
ubuntu $ ls /root
filesystem  mytestdir  snap  terraform
```

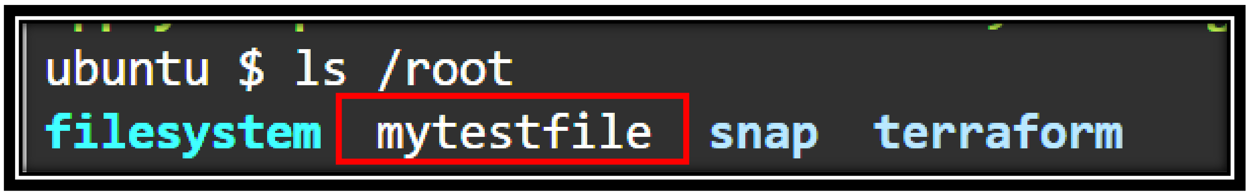
Example: Creating a File on Local System

```
terraform {
  required_providers {
    linux = {
      source  = "TelkomIndonesia/linux"
      version = "0.7.0"
    }
  }
}

provider "linux" {
  host     = "localhost"
  port     = 22
  user     = "root"
  password = "1"
}

resource "linux_file" "testfile" {
  path = "/root/mytestfile"
}
```

- **Out put :-**



```
  # linux_file.testfile will be created
  + resource "linux_file" "testfile" {
      + group          = 0
      + id             = (known after apply)
      + ignore_content = false
      + mode           = "644"
      + overwrite      = false
      + owner          = 0
      + path           = "/root/mytestfile"
    }

Plan: 1 to add, 0 to change, 1 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

linux_directory.mydir: Destroying... [id=b1a376d8-3802-44c9-b82c-4f30ab3c63b5]
linux_file.testfile: Creating...
linux_directory.mydir: Destruction complete after 0s
linux_file.testfile: Creation complete after 0s [id=724eb693-0690-4c4e-a240-551bb9f4eb44]

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



```
ubuntu $ ls /root
filesystem  mytestfile  snap  terraform
```

7. Running the tf Script:

• terraform init	# Initialize Terraform
• terraform plan	# Show expected changes
• terraform apply -auto-approve	# apply the changes

---

8. Destroying Terraform Infrastructure

To remove all resources created by Terraform, run:

- terraform destroy -auto-approve

---

9. Terraform State Management

- **Local State:** Stored in terraform.tfstate file in the working directory.
- **Version Control:** Helps track changes in system configurations.

---

10. Advantages of Terraform

✓ Automates local system configurations.

✓ Reduces manual setup and errors.

✓ Can be integrated with configuration management tools.

✓ Ensures repeatable and consistent setups.

Conclusion

Terraform simplifies system management using code, enabling efficient automation of tasks such as installing software, managing users, and configuring services. It is a powerful tool for DevOps engineers and system administrators working in local environments.