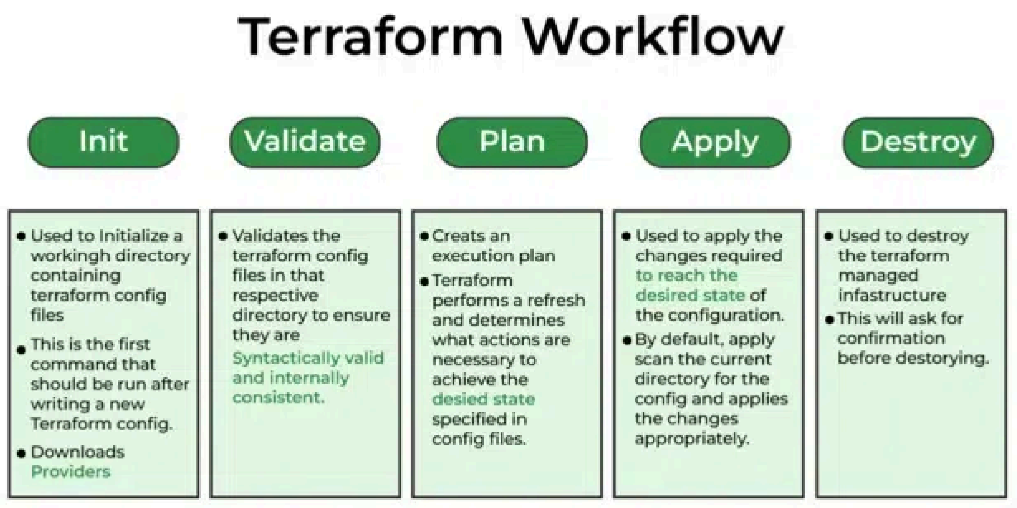
Day 2: Terraform Workflow in Detail

Terraform uses a standard workflow that helps manage infrastructure consistently:

Terraform Workflow:

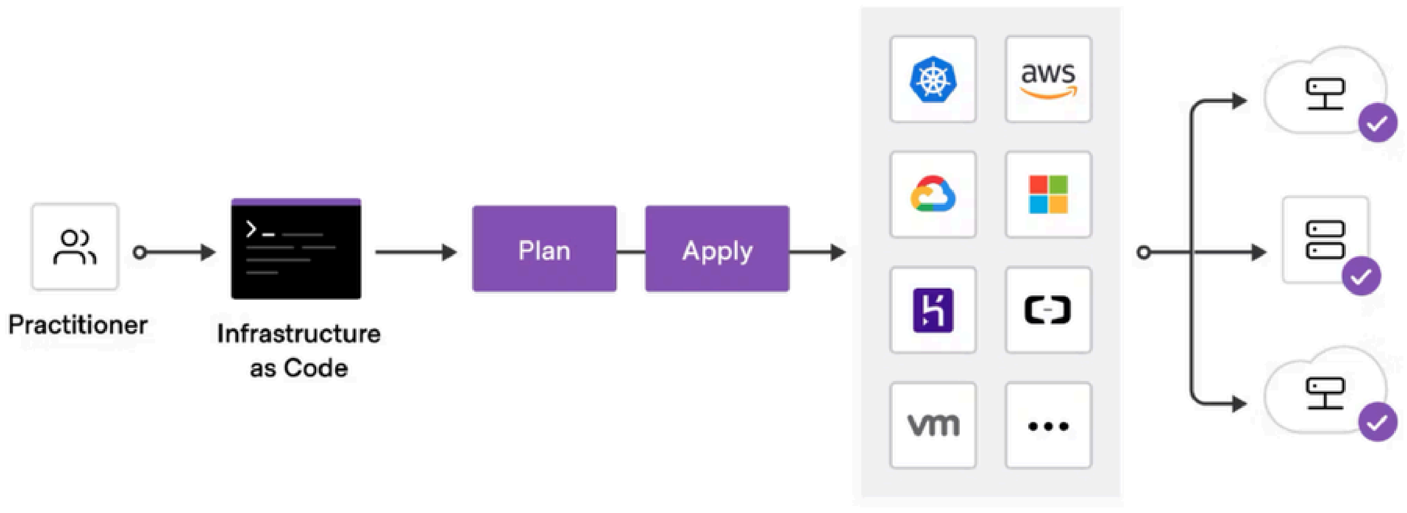
1. **Write**: You write infrastructure as code (IaC) in .tf files using HCL (HashiCorp Configuration Language).
2. **Initialize**: Prepare your directory with terraform init.
3. **Plan**: See what changes Terraform will make before applying them with terraform plan.
4. **Apply**: Make the actual changes to cloud resources with terraform apply.
5. **Destroy**: Remove all resources created by Terraform with terraform destroy.

Terraform uses a standard workflow that helps manage infrastructure consistently:



Terraform Workflow

Init	Validate	Plan	Apply	Destroy
• Used to Initialize a workingh directory containing terraform config files • This is the first command that should be run after writing a new Terraform config. • Downloads Providers	• Validates the terraform config files in that respective directory to ensure they are Syntactically valid and internally consistent.	• Creats an execution plan • Terraform performs a refresh and determines what actions are necessary to achieve the desied state specified in config files.	• Used to apply the changes required to reach the desired state of the configuration. • By default, apply scan the current directory for the config and applies the changes appropriately.	• Used to destroy the terraform managed infastructure • This will ask for confirmation before destorying.



1. Terraform Initialize

- In the project directory, run Terraform init to initialize Terraform.
- This command downloads the necessary provider plugins and sets up the backend configuration
- Prepares the directory to run Terraform commands.

Command:

```
terraform init
```

Example Output:

Initializing the backend...

Initializing provider plugins...

Terraform has been successfully initialized!

Use case tips:

- Always run init first when setting up or cloning a new Terraform project.
- Re-run it if you change your provider versions or backends.

2.terraform validate:

- The Terraform validate command is used to validate the syntax and configuration of your Terraform files without actually applying or modifying any infrastructure. It performs a static analysis of your code and checks for any errors or warnings in the configuration.

When to Use:

- After writing or editing .tf files.
- In CI/CD pipelines to catch configuration errors early.
- Before running terraform plan or terraform apply.

Command:

```
terraform validate
```

What It Does:

- Parses all .tf files in the directory.
- Ensures the syntax is correct.
- Checks for missing variables, providers, or wrong types.
- Validates internal references (like resources and modules).

Success Output:

```
Success! The configuration is valid.
```

Error Example:

Suppose you forget to close a bracket:

```
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-demo-bucket"
  acl    = "private"
```

When you run:

```
terraform validate
```

You’ll get:

Error: Missing '}' - expected closing brace

Notes:

- It doesn't check cloud-specific errors (e.g., invalid instance types) — that’s done by terraform plan.
- It only checks the syntax and internal logic.
- Still need to terraform init before running validate (because provider blocks need to be initialized).

Best Practice:

Use this command **every time you make a change** in .tf files — treat it like a linter or pre-commit check.

3. Terraform Plan:

Purpose: Shows what Terraform will do without making any changes (a dry run).

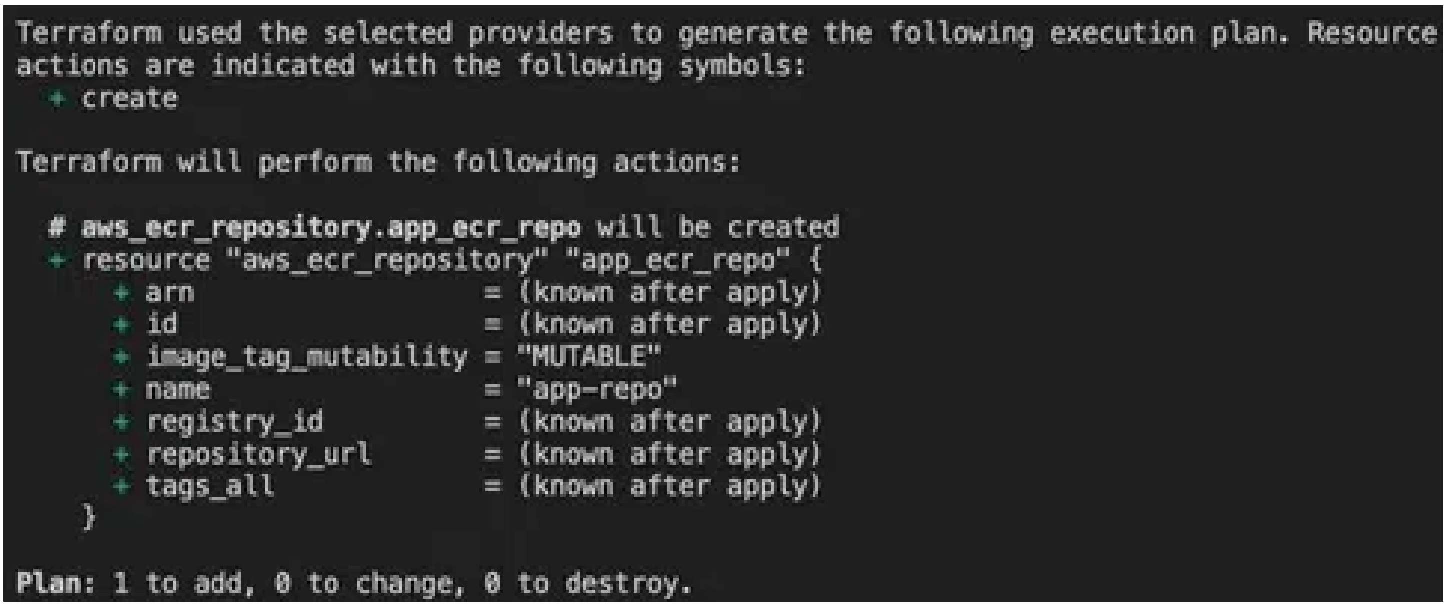
What it does:

- Reads the current state of infrastructure.
- Compares it with your .tf files.
- Shows the execution plan: what will be added, changed, or destroyed.

Command:

```
terraform plan
```

Example Output:



```
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_ecr_repository.app_ecr_repo will be created
  + resource "aws_ecr_repository" "app_ecr_repo" {
      + arn                  = (known after apply)
      + id                   = (known after apply)
      + image_tag_mutability = "MUTABLE"
      + name                 = "app-repo"
      + registry_id          = (known after apply)
      + repository_url       = (known after apply)
      + tags_all             = (known after apply)
    }

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

Output Symbols:

- + Create
- - Destroy
- ~ Modify

Use case tips:

- Use it before every apply to verify and confirm the expected changes.
- Great for code reviews and audits.

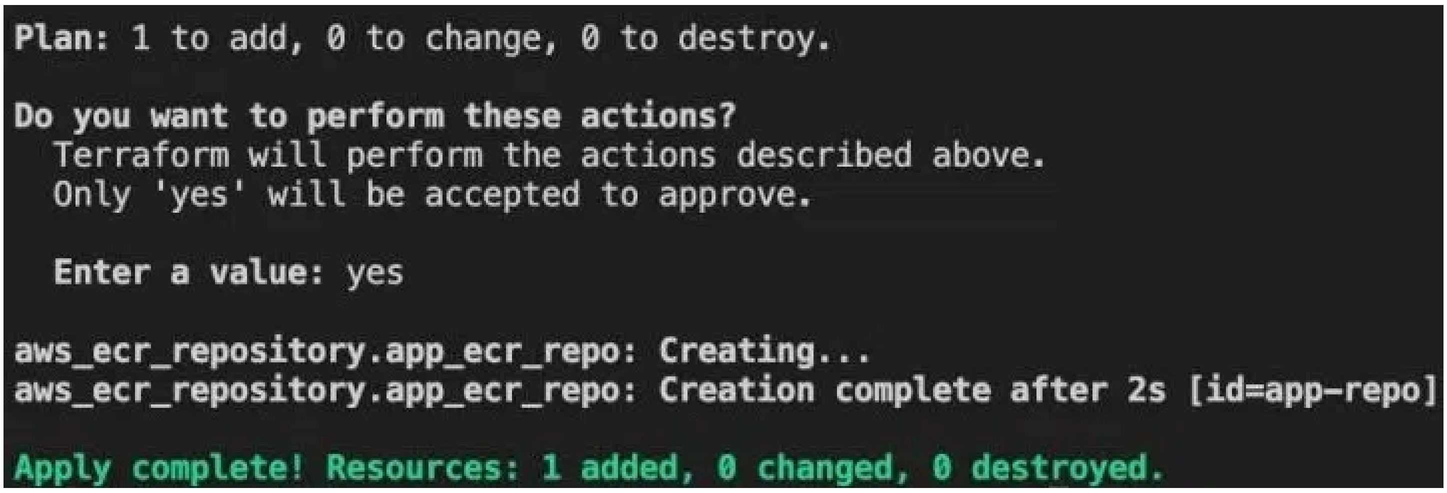
4. Terraform Apply:

Use Terraform apply to execute the plan and apply the changes to your infrastructure.

Terraform prompts for confirmation before proceeding. Once confirmed, it provisions the resources according to your configuration and updates the state file. Now run terraform apply command you will see this as output and with a resource that you have mentioned.

Command:

`terraform 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

aws_ecr_repository.app_ecr_repo: Creating...
aws_ecr_repository.app_ecr_repo: Creation complete after 2s [id=app-repo]

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

5. Terraform Destroy:

Destroys all the infrastructure created by Terraform.

What it does:

- Reads state file.
- Deletes all managed resources defined in your configuration.

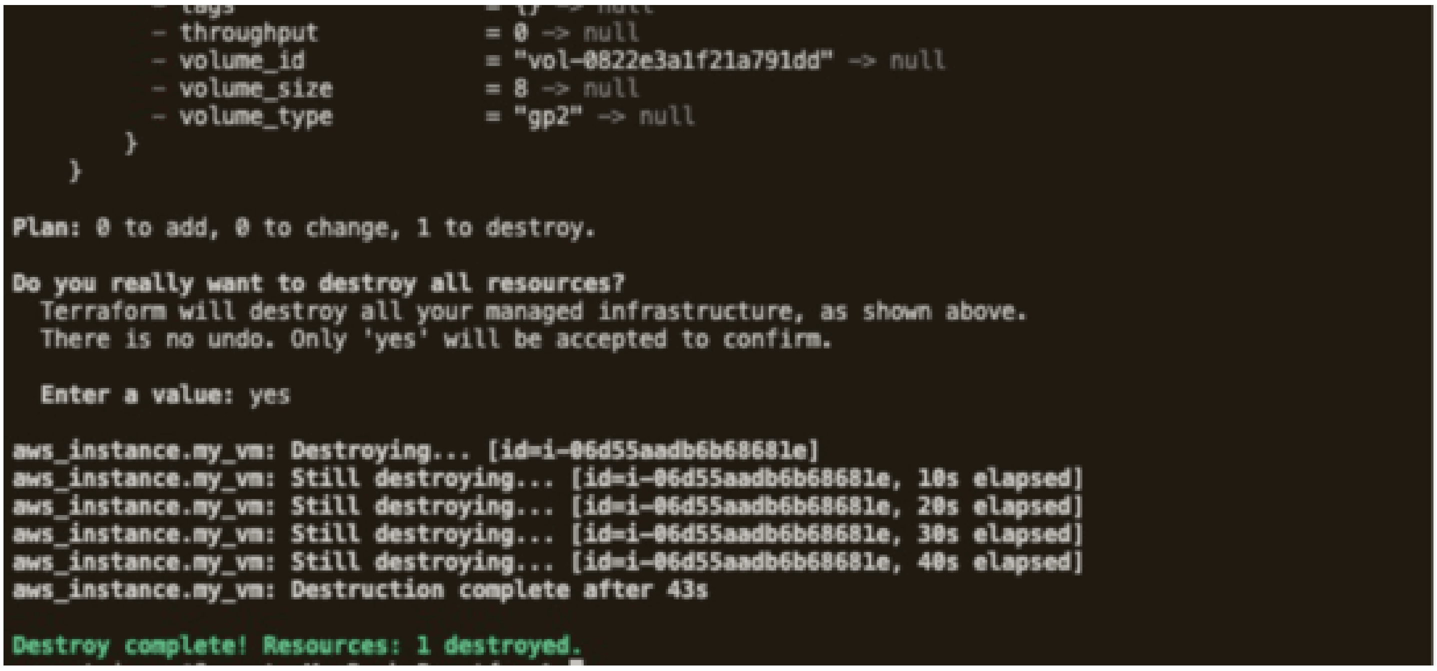
It’s important to exercise caution with this command as it permanently deletes resources.

Use case tips:

- Use carefully, especially in production environments.
- Commonly used for temporary environments (e.g., dev/test).

Command:

terraform destroy



```
          - tags                 = {} -> null
          - throughput           = 0 -> null
          - volume_id            = "vol-0822e3a1f21a791dd" -> null
          - volume_size          = 8 -> null
          - volume_type          = "gp2" -> null
        }
    }

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

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.my_vm: Destroying... [id=i-06d55aadb6b68681e]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 10s elapsed]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 20s elapsed]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 30s elapsed]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 40s elapsed]
aws_instance.my_vm: Destruction complete after 43s

Destroy complete! Resources: 1 destroyed.
```