Terraform and OpenTofu

Version Terraform 1.17 / OpenTofu 1.13 · verified 2026-11-02

Providers, plan/apply, state and locking, for_each, import/moved/removed, ephemeral secrets — and every Terraform vs OpenTofu difference.

Contents

Mental model 2

Providers and resources 2

Variables, locals and outputs 3

The plan/apply workflow and CI 4

State, backends and locking 5

Repetition and modules 6

Import, moved and removed 7

Secrets: sensitive vs ephemeral 8

Terraform vs OpenTofu 9

Further reading 11

Mental model

State is Terraform's record of which real object each resource address owns; your code is the desired state; a plan is the diff between the two after refreshing state from the cloud. Change any of the three outside the workflow — a console edit, two people applying at once, a renamed resource — and the plan stops meaning what you think it means. Most incidents are state problems, not HCL problems. Everything below applies to both `terraform` and `tofu` unless the last section says otherwise.

Providers and resources

Block	Purpose
`terraform { required_providers }`	Pin provider source and version
`provider "aws" { … }`	Configure a provider
`resource "TYPE" "NAME"`	An object Terraform manages
`data "TYPE" "NAME"`	Read an object it does not manage

A resource's address is `TYPE.NAME` (`aws_s3_bucket.logs`); state maps that address to a real ID. Data sources are read at plan time and never created or destroyed. `terraform init` downloads providers and records exact versions and checksums in `.terraform.lock.hcl`.

```
terraform {
  required_providers {
```

```
    aws = {
      source  = "hashicorp/aws"
      version = "~> 6.0"
    }
  }
}

provider "aws" { region = "eu-west-1" }

resource "aws_s3_bucket" "logs" {
  bucket = "acme-checkout-logs"
}
```

> **Tip:** Commit `.terraform.lock.hcl`. Without it, two machines can resolve different provider versions from the same `~>` constraint.

Variables, locals and outputs

Block	Role	Reference
`variable`	Input to a module	`var.env`
`locals`	Named expression	`local.prefix`
`output`	Value a module exposes	`module.x.out`

Set variables with `-var`, a `*.tfvars` file, or `TF_VAR_name` environment variables. Give each one a `type` and, where a bad value would be expensive, a `validation` block.

```
variable "env" {
  type = string
  validation {
    condition = contains(
      ["dev", "prod"], var.env)
    error_message = "env must be dev or prod."
  }
}

locals { prefix = "checkout-${var.env}" }

output "bucket" { value = aws_s3_bucket.logs.id }
```

The plan/apply workflow and CI

Command	Does
`init`	Providers, modules, backend
`fmt -check` / `validate`	Style and static checks
`plan -out=tfplan`	Diff, saved to a file
`apply tfplan`	Apply exactly that diff
`plan -detailed-exitcode`	Exit 2 when changes exist

In CI, plan on the pull request, review the plan output, then apply the saved plan file after merge — never a fresh, unreviewed plan. A scheduled `plan -detailed-exitcode` doubles as drift detection: exit code 2 means reality no longer matches the code.

```
terraform init -input=false
terraform fmt -check
terraform validate
terraform plan -input=false -out=tfplan
terraform apply -input=false tfplan
```

> **Warning:** A saved plan file contains variable values and resource attributes, including sensitive ones. Treat it as a secret and keep it out of public CI artifacts.

State, backends and locking

Concern	Answer
Where state lives	A backend: `s3`, `gcs`, `azurerm`, …
Concurrent applies	Locking; S3 uses `use_lockfile`
DynamoDB locking	Deprecated in Terraform's S3 backend
Many environments	Separate state per env
`terraform.workspace`	Same code, several states

Local state on a laptop cannot be shared or locked, so every team uses a remote backend. The S3 backend locks with a lock object next to the state when `use_lockfile = true` (it defaults to `false`). Workspaces reuse one backend and one set of credentials, which is why both tools say they are unfit for isolating environments that need separate access.

```
terraform {
  backend "s3" {
    bucket       = "acme-tf-state"
    key          = "checkout/prod.tfstate"
    region       = "eu-west-1"
    use_lockfile = true
  }
}
```

> **Gotcha:** In Terraform, a `backend` block cannot reference variables or locals — use `-backend-config` at `init`. OpenTofu allows variables and locals there, so code written for it may not run on Terraform.

Repetition and modules

Tool	Use when
`count = n`	N near-identical copies
`for_each = map/set`	Copies keyed by a stable name
`module "x" { source }`	Reusable group of resources
`dynamic` block	Repeated nested blocks

`count` instances are addressed by index (`[0]`), `for_each` instances by key (`["eu"]`). The keys must be known at plan time, so they cannot come from an attribute the cloud assigns on create. A module is any directory of `.tf` files; call it with inputs, read its outputs.

```
variable "buckets" {
  type = map(string) # name => region tag
}

resource "aws_s3_bucket" "this" {
  for_each = var.buckets
  bucket   = "acme-${each.key}"
  tags     = { region = each.value }
}
```

> **Gotcha:** Removing an item from the middle of a `count` list shifts every later index, so the plan replaces resources you did not touch. Use `for_each` for anything that has a natural name.

Import, moved and removed

Block	Effect on next apply
`import { to, id }`	Adopt an existing object
`moved { from, to }`	Rename an address, keep the object
`removed { from }`	Forget it; `destroy = false` keeps it

These blocks make refactoring reviewable: they appear in the plan, not in someone's shell history. `plan -generate-config-out=FILE` writes starter HCL for imported resources. Delete `moved` blocks only once every state that used the old address has applied them.

```
import {
  to = aws_s3_bucket.logs
  id = "acme-checkout-logs"
}

moved {
  from = aws_s3_bucket.this["eu"]
  to   = aws_s3_bucket.eu
}

removed {
  from = aws_s3_bucket.legacy
  lifecycle { destroy = false }
}
```

Secrets: sensitive vs ephemeral

Marker	Written to state and plan?
`sensitive = true`	Yes, only redacted in output
`ephemeral = true` (variable, output)	No
`ephemeral "TYPE" "NAME"` block	No
write-only argument (`*_wo`)	No

`sensitive` only redacts CLI output; the value is still written in plaintext to state and plan files. Ephemeral values exist for one run and may only flow into ephemeral contexts: provider blocks, other ephemeral values,

and write-only arguments. Bump the `*_wo_version` argument to make the provider send a new value.

```
ephemeral "random_password" "db" {
  length = 24
}

resource "aws_db_instance" "main" {
  # ... engine, size, username
  password_wo = (
    ephemeral.random_password.db.result
  )
  password_wo_version = 1
}
```

> **Warning:** Anyone who can read the state bucket can read every `sensitive` value in it. Lock down the bucket as tightly as the secrets themselves.

Terraform vs OpenTofu

Feature	Terraform 1.17	OpenTofu 1.13
State and plan encryption	No (protect the backend)	Built in
Variables in `backend`	No	Yes
`lifecycle { enabled }`	No	Yes
Stacks	Yes, in HCP Terraform	No

Feature	Terraform 1.17	OpenTofu 1.13
Native policy (`-policies`)	Yes	No
`.tofu` override files	No	Yes

Both share HCL, ephemeral values, write-only arguments and the refactoring blocks above. OpenTofu pulls providers from `registry.opentofu.org`, Terraform from `registry.terraform.io`. A `.tofu` file beats a `.tf` file of the same name, so one repo can carry OpenTofu-only code.

```
terraform {
  encryption { # OpenTofu only
    key_provider "aws_kms" "main" {
      kms_key_id = var.state_key_id
      region     = "eu-west-1"
      key_spec   = "AES_256"
    }
    method "aes_gcm" "main" {
      keys = key_provider.aws_kms.main
    }
    state { method = method.aes_gcm.main }
  }
}
```

> **Gotcha:** Once state encryption is on, OpenTofu cannot read the state without the key. Back up the unencrypted state and the key material before you enable it.

Further reading

- Terraform language documentation
- OpenTofu documentation
- Terraform — Manage sensitive data
- OpenTofu — State encryption
- OpenTofu — Migrating from Terraform