Engineer

Terraform

HashiCorp's tool that plans and applies infrastructure code

What it is

Terraform is HashiCorp's infrastructure as code tool. You describe cloud and on-premises resources in plain text files, keep those files in version control, and Terraform works out which real objects to create, change or destroy so that the running estate matches the files. The program itself is called as `terraform`, in lower case, which is how the name appears in every command and script.

The language is declarative. HashiCorp's documentation describes it as "describing an intended goal rather than the steps to reach that goal". A configuration says that a machine of a certain size exists in a certain network; it does not list the API calls that would produce one. Files carry the `.tf` extension, and there is also a JSON-based variant with the `.tf.json` extension. The native syntax is defined in terms of a syntax called HCL and has two main building blocks: a block, which is a container for other content, and an argument, which assigns a value to a name.

Three ideas carry the rest of the tool.

Providers are plugins. The configuration tells Terraform which plugins to install, and `terraform init` finds, downloads and installs them. HashiCorp says providers let Terraform work with virtually any platform or service that has an accessible API, and the publicly available ones are listed on the Terraform Registry. A provider defines the resource types for its platform, so the same core program can manage a DNS zone, a virtual machine and a monitoring rule at three different vendors.

Modules are directories. A module is a collection of `.tf` and `.tf.json` files kept together in one directory, and the root module is the working directory where Terraform is invoked.

State is memory. Terraform keeps track of the real infrastructure in a state file, which its introduction calls the source of truth for the environment. The state maps each resource in the configuration to the real object it manages, along with metadata. By default it is a local file named `terraform.tfstate`, and the previous version is kept beside it as `terraform.tfstate.backup`.

The licence has changed. On August 10, 2023, HashiCorp announced that it was moving its source code licence from the Mozilla Public License 2.0 to the Business Source License 1.1 for all future releases of its products, while its APIs, SDKs and almost all other libraries stay on MPL 2.0. OpenTofu describes itself as a Terraform fork, started by Gruntwork, Spacelift, Harness, Env0, Scalr and others, and developed under the Linux Foundation's stewardship. Its repository is published under MPL 2.0, and its site presents it as a replacement for Terraform that keeps existing workflows and configurations.

Use the concept in a decision

Put "What is Terraform?" to work in a real decision. We can help relate the concept to the system you are building or buying. The whole setup written down as files, so it can be recreated exactly and never lives only in one person's head.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

HashiCorp describes the core workflow in three steps. Write: you define resources, which may span several providers and services. Plan: Terraform works out what it would create, change or destroy and lays that out as an execution plan. Apply: once the plan is approved, it carries out those changes, ordering them by the dependencies between resources, which it derives from a resource graph.

The middle step is where most of the value sits. A plan is a preview that a person who did not write the change can read before anything happens. A reviewer sees more than the diff of the files: they see the consequences, two resources created, one updated and, now and then, one destroyed that nobody meant to touch. That last line is cheap to catch on a pull request and very expensive to discover afterwards.

The second reason is reach. Because the knowledge of each platform lives in its provider, one language, one workflow and one set of review habits cover every provider a team uses. A company on two clouds and a separate DNS host can describe all three in one repository and review all three the same way. Microsoft's Bicep makes a different trade in its Azure Resource Manager workflow: it keeps no state file of its own, because Azure stores the deployment state. That does not make Bicep an Azure-only language: its extensions also support resources outside Resource Manager, including Kubernetes and Microsoft Graph.

The third reason is the state. Since Terraform records which real object belongs to which block, it can tell an object it manages from one it has never seen, and it can compare the record with what is running. That record is also the price of the tool. It has to live somewhere the whole team can reach, it needs a lock so two runs do not write it at once, and it can contain secrets, so it needs access control as strict as theirs.

There are smaller costs as well. HCL is one more language to learn, and every provider has its own resource types and quirks. Teams that build a product on top of Terraform also have a licence to read: HashiCorp's licence FAQ has a question of its own on what counts as a competitive offering. OpenTofu offers the same workflow under MPL 2.0.

What it looks like in practice

HashiCorp's page on resource syntax uses this block as its example:

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

`resource` is the block type and `aws_instance` is the resource type, defined by the AWS provider. `"web"` is the local name: Terraform uses that label to track the resource in its state file, and it has no effect on the settings of the real machine. The two lines inside the block are arguments.

Three commands take that file from text to a running machine:

```
terraform init
terraform plan -out=tfplan
terraform apply tfplan
```

`terraform init` prepares the working directory: it installs the provider plugins the configuration asks for and sets up the place where state will be kept. It is safe to run more than once, and its documentation says to commit the dependency lock file to version control.

`terraform plan -out=tfplan` computes the plan and writes it to a file in an opaque format. Planning does not apply the proposed infrastructure changes. `terraform apply tfplan` then performs the operations in that file without asking for confirmation, because HashiCorp treats passing a saved plan as the approval. For the same reason, the `-auto-approve` flag is ignored when a plan file is given.

In a team the split falls naturally. A CI job runs the plan on the pull request and keeps the file, a person reads the plan as well as the diff, and after the merge a second job applies that same file, so what runs is what was read. The apply documentation adds a condition that is easy to forget: make sure nobody can change the infrastructure outside the Terraform workflow, or the plan describes a world that has already moved.

On this site, the tech stack page puts Terraform alongside Ansible: both write down what the machines should be, so that an environment can be read instead of reconstructed. A portfolio entry applies the idea with a different tool: Azure infrastructure was provisioned as code in Bicep. The testing, staging and product environments are all built from one set of templates, and since the templates take parameters, adding or changing an environment is a pull request. Set beside this article, the entry shows the trade from earlier. Bicep has no state file of its own because Azure holds the record, while Terraform keeps its own state and in return works across more than one cloud.

Where it goes wrong

Most Terraform accidents share a shape: the configuration is valid, the run succeeds, and something real is gone or changed.

Renaming a resource destroys it. Change the label `"web"` to `"frontend"` and the resource has a new address. HashiCorp's refactoring guide says that by default Terraform interprets a changed address as an instruction to destroy the existing object and create a new one. For a disposable machine that costs a few minutes; for a database it costs the data. The fix is a `moved` block, which updates the address without destroying anything:

```
moved {
  from = aws_instance.web
  to   = aws_instance.frontend
}
```

Keep the block once it has done its work. HashiCorp strongly recommends retaining historical `moved` blocks and states that removing one is a breaking change.

Two people apply against two copies of local state. Out of the box, each workspace's state is a local file. That suits one person on one laptop. As soon as a colleague runs Terraform from another copy, the two state files disagree about what exists, and each run acts on its own version. The fix is to keep state in HCP Terraform or in remote storage with locking and access control, as HashiCorp recommends. Terraform then locks the state automatically for every operation that could write it, and does not continue if it cannot take the lock. Not all storage options support locking, so check before choosing one, and leave `-lock=false` alone: the flag exists, and the documentation recommends against it.

A stuck lock is forced open while someone is still working. When automatic unlocking fails, the lock stays behind, and the `force-unlock` command releases it with the lock ID that Terraform printed. The locking documentation tells the user to be very careful: forcing a lock that someone else holds can leave two writers on the same state, and the command is meant only for your own lock after automatic unlocking failed. The fix is to find out whose lock it is, and whether that run is still alive, before forcing anything.

The state file leaks secrets. HashiCorp warns that secrets can end up in the state, and that storing it in a system without secure access control can expose them. A state file in a world-readable bucket, or copied into a build artefact, is a

credentials leak. Keep it out of the repository, give it the same access control as the secrets it holds, and never edit it by hand; the `terraform state` command exists for the corrections that are needed.

Approval becomes a formality. The `-auto-approve` flag skips the interactive approval of the plan, and passing a saved plan file skips it too, because the file counts as the approval. The apply documentation warns that this covers changes that destroy things too, deleting resources among them. A CI job that applies on every merge without a reviewed plan has removed the one moment where a person could have seen the destroy coming. The fix is the saved-plan flow above, with a required human review between the plan job and the apply job. Where `-auto-approve` is used in automation, HashiCorp's own advice is to run a plan first and review it.

Further reading

- What is Terraform? — HashiCorp's introduction: write, plan, apply, providers and state.
- terraform apply command — saved plans, `-auto-approve`, and running the command in automation.
- State Locking — automatic locks, unsupported storage and the `force-unlock` warning.
- Refactor modules — why a changed address means destroy and create, and the `moved` block.
- HashiCorp adopts Business Source License — the August 2023 announcement of the move from MPL 2.0 to BSL 1.1.
- OpenTofu — the fork of Terraform maintained under the Linux Foundation.



Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Terraform?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/terraform/