Terraform Notes PDF

Terraform is an infrastructure as code tool that lets you describe cloud resources in configuration files, preview changes with a plan and create them with apply. These notes explain providers, resources, variables, outputs, state, modules, the core workflow and safe practices, with small examples you can adapt to any supported cloud.

Key takeaways

- Terraform is declarative: you describe the end state and it works out the steps.
- The core workflow is write, init, plan, apply and, when needed, destroy.
- State records what Terraform manages, so protect it and store it remotely for teams.
- Variables, outputs and modules keep code reusable and clean.
- Always read the plan before applying, and never put secrets in code.

What is Terraform notes?

Terraform is an open-source infrastructure as code tool that provisions and manages cloud and on-premises resources from declarative configuration files, tracking what it created in a state file.

What is infrastructure as code?

Infrastructure as code (IaC) means defining servers, networks and services in text files that can be versioned, reviewed and repeated, instead of clicking in a console. Benefits are consistency across environments, a review trail, fast rebuilds and less manual error. Terraform is one popular IaC tool; others include cloud-specific templates.

What is the Terraform workflow?

1. **Write** configuration in `.tf` files.
2. `terraform init` downloads providers and sets up the working folder.
3. `terraform fmt` **and** `terraform validate` format and check the code.
4. `terraform plan` shows what will be created, changed or destroyed.
5. `terraform apply` makes the changes after confirmation.
6. `terraform destroy` removes managed resources.

Always read the plan; it is your safety check.

What are providers and resources?

A **provider** is a plugin that talks to a platform's API (for example AWS, Azure, Google Cloud or Kubernetes). A **resource** is one object you manage, such as a virtual machine or a storage bucket.

```
terraform {
  required_providers {
    aws = { source = "hashicorp/aws", version = "~> 5.0" }
  }
}

provider "aws" {
  region = var.region
}

resource "aws_s3_bucket" "logs" {
  bucket = "example-logs-${var.env}"
  tags   = { Environment = var.env }
}
```

Pin provider versions so that results are repeatable.

How do variables and outputs work?

Input variables make code reusable; **outputs** print or share values; **locals** name expressions.

```
variable "env" {
  type        = string
  description = "Environment name"
  default     = "dev"
}

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

Set values with a `.tfvars` file, environment variables or the command line. Mark sensitive values with `sensitive = true` so they are hidden in output.

What is state?

Terraform stores what it manages in a **state file**. It compares state, configuration and real infrastructure to decide what to change. State can contain sensitive data, so do not commit it to Git. For teams, use **remote state** in a shared backend with locking, which stops two people applying at once. Never edit state by hand; use `terraform state` commands when a change is needed.

What are data sources?

A **data source** reads information that already exists, such as an existing network or the latest machine image, without managing it. Use it to reference resources created elsewhere.

What are modules?

A module is a folder of Terraform files that you call from another configuration, like a reusable function. Use modules for repeated patterns such as a network or a web service. Keep modules small, give them clear inputs and outputs and version them.

How do dependencies work?

Terraform builds a graph from references: if a resource uses another resource's attribute, it creates the first one first. Use `depends_on` only when a dependency cannot be seen from references.

How do I handle several environments?

Use separate state per environment, either through separate folders or workspaces, and pass environment-specific values with variable files. Keep the same modules across environments so that differences are small and visible.

What are safe practices?

Store code in Git and review changes through pull requests. Run `plan` in a pipeline and require approval before `apply`. Use least-privilege credentials from the environment, never in code. Tag resources for ownership and cost tracking. Test changes in a non-production environment first, and use `prevent_destroy` on critical resources.

What are common mistakes?

Committing state or secrets, applying without reading the plan, changing resources by hand so that they drift from code, using unpinned versions and putting too much in one state file.

How do I practise safely?

Use a free or practice account, set a budget alert and begin with cheap resources such as a storage bucket or a small virtual machine. Run `plan` often, apply, inspect the result in the console and then run `destroy` so that nothing keeps costing money. Keep your practice code in Git and write a short readme of what each folder builds.

How does Terraform compare with configuration tools?

Terraform mainly provisions infrastructure, such as networks and servers. Configuration tools install and configure software on servers after they exist. Many teams use both, or replace the second step with container images. Know the difference and give an example of each.

What should I read next?

After these notes, study how to build a small network and a web server with modules, how to set up remote state with locking and how to run Terraform in a pipeline with a plan-then-approve step.

Read the provider documentation for the cloud you use, since arguments and behaviour differ between providers.

Frequently asked questions

What is Terraform?

Terraform is an infrastructure as code tool that provisions resources from declarative configuration files and tracks them in a state file.

What does terraform plan do?

It compares configuration, state and real infrastructure and shows what would be created, changed or destroyed without making changes.

Why is Terraform state important?

State records the resources Terraform manages. It can hold sensitive data, so protect it and use remote state with locking for teams.

What is a Terraform module?

A module is a reusable folder of configuration that you call from other configurations, like a function.

Is it safe to put passwords in Terraform files?

No. Use environment variables or a secrets manager, mark values sensitive and never commit secrets or state to Git.

Sources and further reading

- Terraform documentation
- AWS documentation

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550