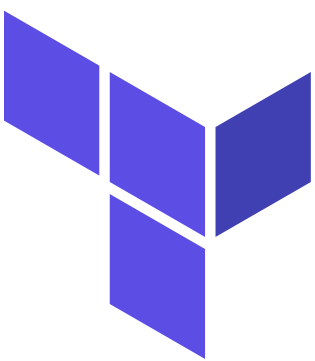
# title: Terraform best practices


Be smart. Think open source.



Write, Plan, and Create Infrastructure as Code

HashiCorp

# Terraform

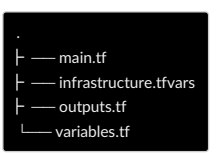
# Agenda

- Repository
- Workspaces
- Modules
- State

# Repository

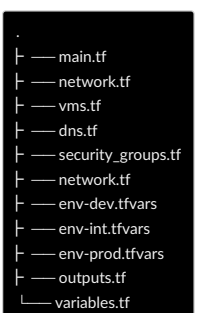
Split your files!

# Bad idea



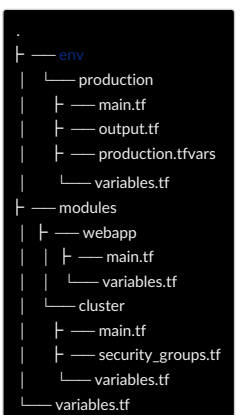

...and main.tf is over 2000 lines...

# Better approach




Better, but not much reusability

# Use modules, workspaces and other fancy things



# Workspaces

Each Terraform configuration has an associated backend that defines how operations are executed and where persistent data such as the Terraform state are stored.

The persistent data stored in the backend belongs to a workspace.

# Use workspaces

```
terraform workspace new production
terraform workspace new development
terraform workspace select development
terraform workspace list
```

# How to use it

The current workspace can be accessed with `${terraform.workspace}`

# How to use it

Variables should be restructured:

```
variable "token" {
  type = "map"
  default = {
    production  = "abcd1234"
    development = "1234abcd"
  }
}
```

# How to use it

...and can be accessed with

```
provider "github" {
  token = "${lookup(var.token, terraform_workspace)}"
}
```

# Modules

Modules in Terraform are self-contained packages of Terraform configurations that are managed as a group. Modules are used to create reusable components in Terraform as well as for basic code organization.

https://www.terraform.io/docs/modules/index.html

# Use a module

```
module "kubernetes" {
  source  = "coreos/kubernetes/azurerm"
  version = "1.8.9-tectonic.1"

  # insert the 7 required variables here
}
```

Uses https://registry.terraform.io/modules/coreos/kubernetes/azurerm/1.8.9-tectonic.1

# Build your own module

Just put everything that belongs together in a directory, use local variables and...

# Build your own module

...include it:

```
module "my-module" {
  source  = "./user-with-home-repo"
  username = "dummy"
}
```

# State

State is a necessary requirement for Terraform to function.

Reasons:

- Map resources in config to real-world objects
- Store metadata
- Perform better
- Syncing

# Backend Types (local)

```
terraform {
  backend "local" {
    path = "./states/terraform.tfstate"
  }
}
```

# Backend Types (remote, without locking)

NOT RECOMMENDED!

- artifactory
- etcd
- swift

# Backend Types (remote, with locking)

- azurerm
- consul
- etcdv3
- http (locking optional)
- s3 (locking via DynamoDB)
- terraform enterprise

# Backend Types (remote)

```
backend "etcdv3" {
  endpoints = ["etcd-1:2379", "etcd-2:2379", "etcd-3:2379"]
  lock      = true
  prefix    = "terraform-state/"
}
```

# Linting

`terraform validate` validates the syntax of terraform files.

# Styling

`terraform fmt` rewrites files in correct style and format. Configure your editor to do it automatically on save!

# Testing

Terraform provides unit and acceptance tests for providers in Go. To test your infrastructure you have to rely on third-party software.

# Integration Testing

- kitchen-terraform
- Goss

# Unit Testing

- Terratest
- Rspec-terraform

# Attribution / License

# Feel Free to Contact Us

www.adfinis-sygroup.ch

Tech Blog

GitHub

info@adfinis-sygroup.ch

Twitter