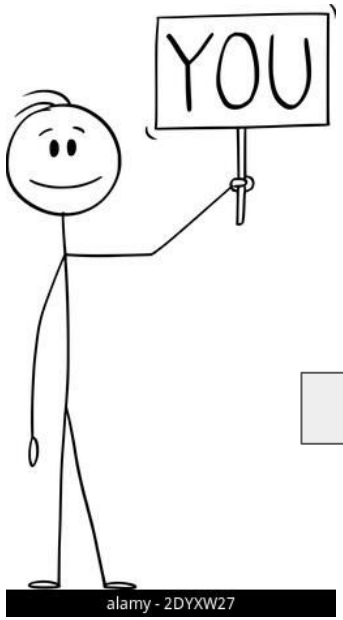
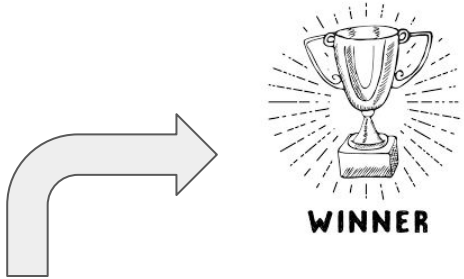
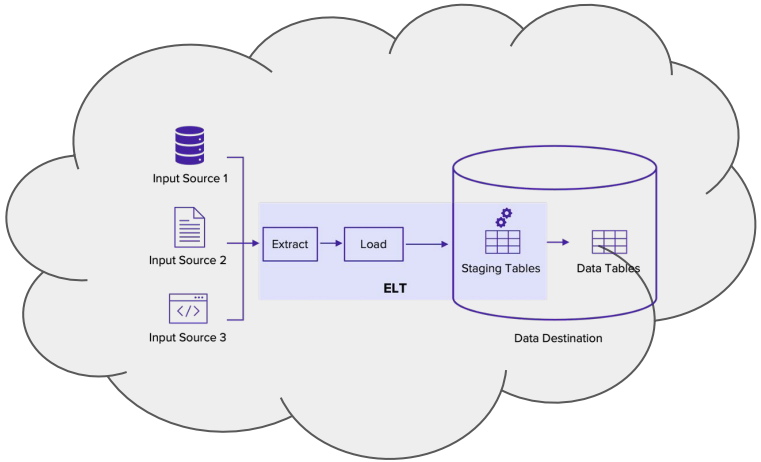
Terraform

Inspiration


YOU
alamy - 2DYXW27


WINNER


Input Source 1
Input Source 2
Input Source 3
Extract
Load
ELT
Staging Tables
Data Tables
Data Destination

But ...

The “but”

Everything runs only in one environment (prod)

- No separation between dev / test / prod.
- Any change = risk of breaking production.

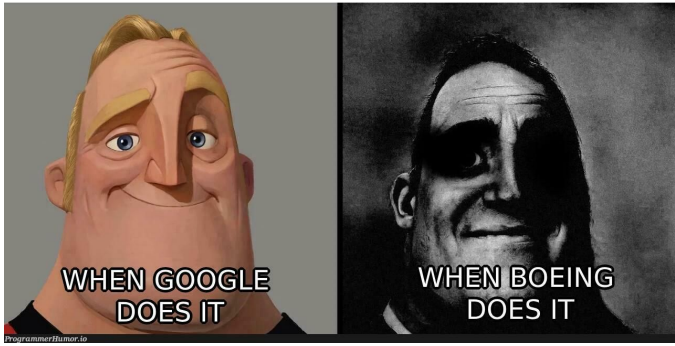
Infrastructure is “handmade”

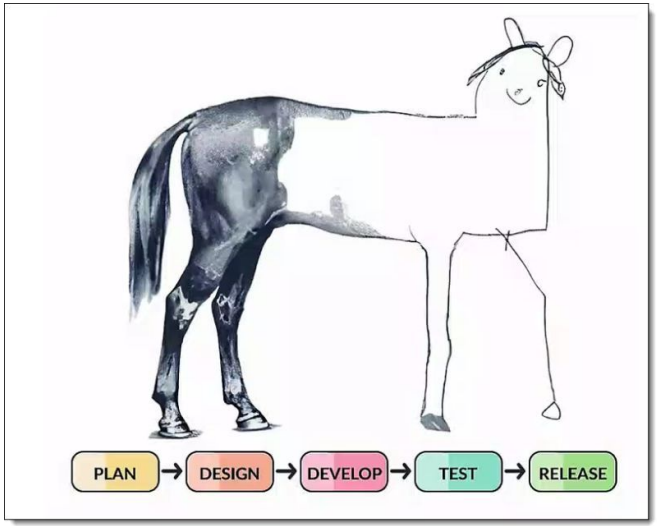
- Pipelines depend on resources created manually in GCP.
- Tracking who created what (and why) is messy.

Hard to reproduce setups

- If a pipeline fails, or someone new joins, recreating the setup is time-consuming.

TESTING IN PRODUCTION





What is dev/test/prod?

Dev (Development)

- Playground for building new features
- Small sample data, not sensitive
- Break things safely — experiments happen here

Test (Staging / QA)

- Looks like prod, but with test data
- Used for validation and integration testing
- Goal: catch issues before real users/data are affected

Prod (Production)

- Real data, real dashboards, real users
- Must be stable, secure, monitored
- Only well-tested code and configs go here

Terraform

Terraform is an open source tool created by HashiCorp that allows you to define your **infrastructure as code** using a simple, declarative language and to **deploy and manage that infrastructure** across a variety of public cloud providers (e.g., Amazon Web Services, Microsoft Azure, Google Cloud Platform)”

Infrastructure as a code???

Normally, people create resources in GCP by clicking around in the web console/CLI. That worksm but it’s manual, hard to repeat, and error-prone.

With **Infrastructure as Code**, you ***describe*** **your infrastructure** in “text files” (like “I need a BigQuery dataset, a GCS bucket, and a cloud function”).

You save that in Git, so others can review changes, track history, and collaborate.

The big benefit: you can **spin up the exact same environment** for dev, test, and prod

HCL

Terraform uses a language called **HCL — HashiCorp Configuration Language**.

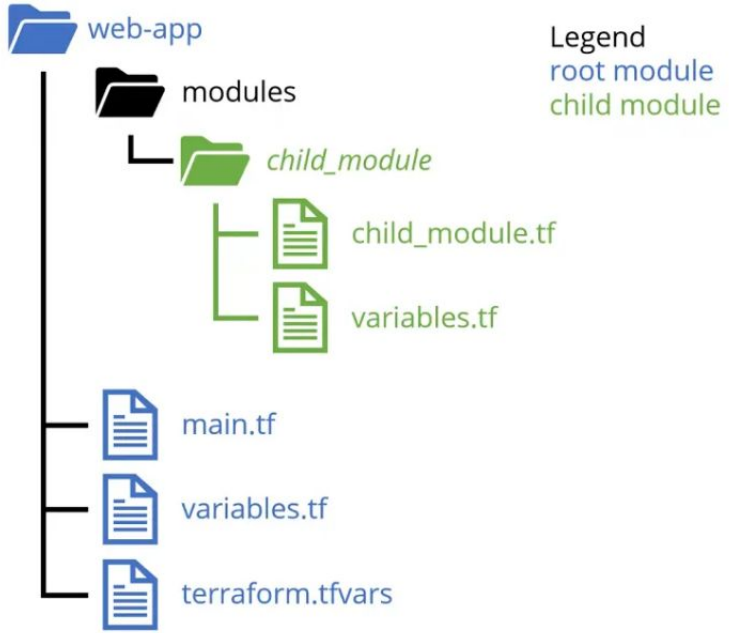
It looks a bit like JSON, but friendlier for humans to read and write.

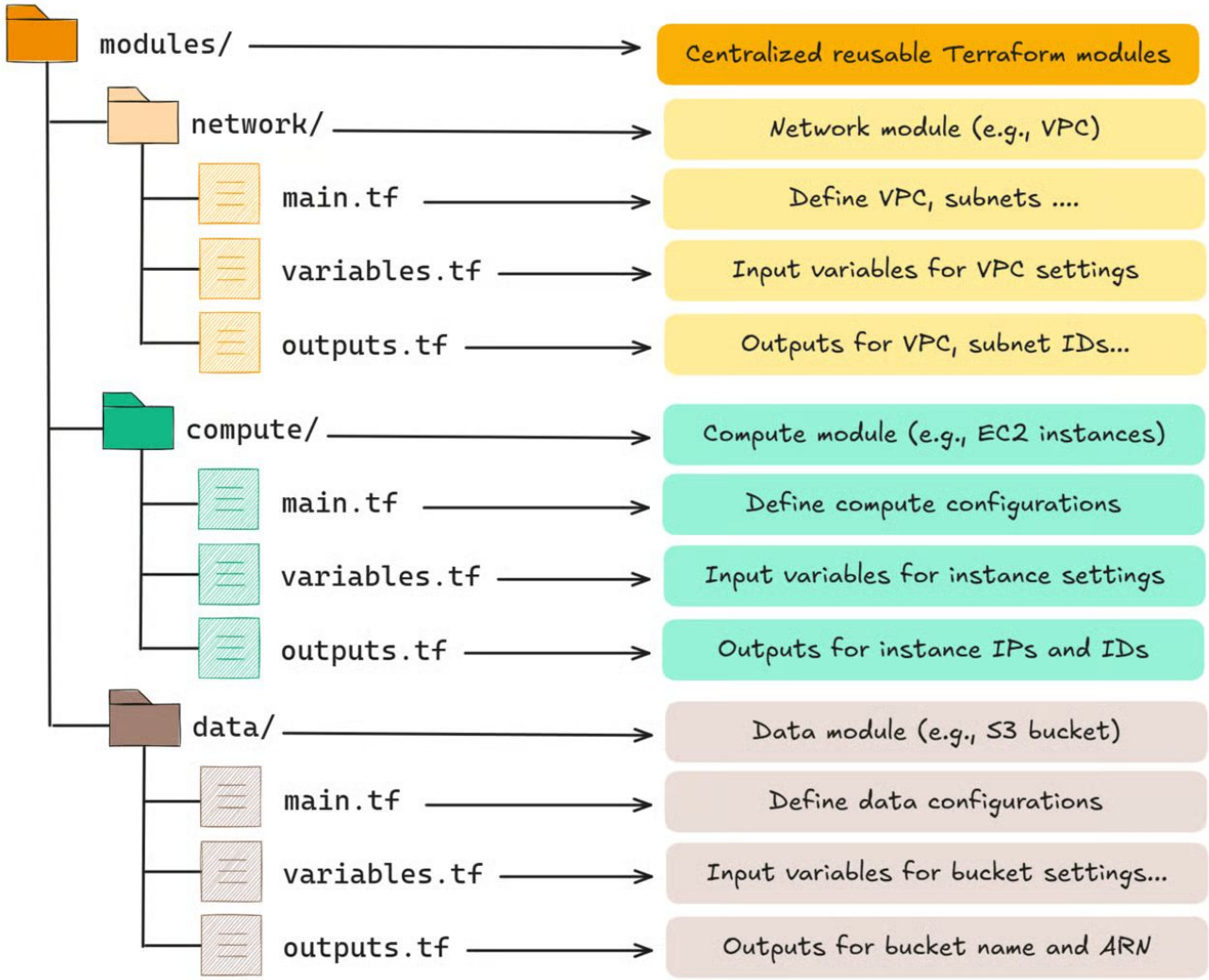
```
variable "project_id" {
  description = "GCP Project ID"
  type        = string
}

variable "bucket_name" {
  description = "Name of the storage bucket"
  type        = string
}

variable "function_name" {
  description = "Name of the function"
  type        = string
}
```

```
provider "google" {
  project = var.project_id
  region  = var.region
}

resource "google_storage_bucket" "raw_bucket" {
  name     = var.bucket_name
  location = var.region
}
```

Project structure

- main.tf The *blueprint* of your infrastructure
- variables.tf Defines *what inputs Terraform expects.*
  Like function parameters (types, defaults, descriptions)
- terraform.tfvars Provides the *actual values* for variables
  Like calling the function with arguments




modules/
Centralized reusable Terraform modules
network/
Network module (e.g., VPC)
main.tf
Define VPC, subnets ....
variables.tf
Input variables for VPC settings
outputs.tf
Outputs for VPC, subnet IDs...
compute/
Compute module (e.g., EC2 instances)
main.tf
Define compute configurations
variables.tf
Input variables for instance settings
outputs.tf
Outputs for instance IPs and IDs
data/
Data module (e.g., S3 bucket)
main.tf
Define data configurations
variables.tf
Input variables for bucket settings...
outputs.tf
Outputs for bucket name and ARN

1. terraform init

- Sets up Terraform in your project
- Downloads the provider plugins (e.g. Google Cloud)
- Run this once per project (or when adding new providers/modules)

2. terraform plan

- Dry run = “What would happen if I apply this?”
- Shows resources to add, change, or delete

3. terraform apply

- Actually creates/updates infrastructure
- Executes the plan

4. terraform destroy

- Tears down all resources managed by Terraform
- Useful for cleanup in dev/test
- Dangerous in prod 🚨 (removes everything in state)

State file

Every time you run `terraform apply`, Terraform needs to know:

- *What resources already exist?*
- *What's new in my code?*
- *What needs to change or be destroyed?*

Terraform keeps that information in a **state file**.

Market Share of Terraform

Current Customer(s)	Market Share (Est.)	Ranking
39 321	34.18%	#1

Top Competitors and Alternatives of Terraform

The top three of Terraform's competitors in the Configuration Management category are Ansible with 31.70%, Puppet with 13.61%, Chef with 6.64% market share.

Technology	Domains	Market Share (Est.)	Versus page
Ansible	36 458	31.70%	Terraform vs Ansible
Puppet	15 659	13.61%	Terraform vs Puppet
Chef	7 643	6.64%	Terraform vs Chef
AWS Config	3 653	3.18%	Terraform vs AWS Config

Words of wisdom

- terraform is not a data engineering tool, its a DevOps tool.

“If you understand Terraform conceptually, you can talk to DevOps engineers in their language, avoid surprises in prod, and design pipelines that scale safely. You don’t need to become DevOps, but you need to be a good neighbor to them”