Intro to IaC with Terraform

Infrastructure as Code

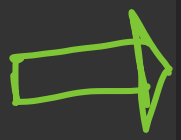
Advantages: Reproducibility and serves as documentation

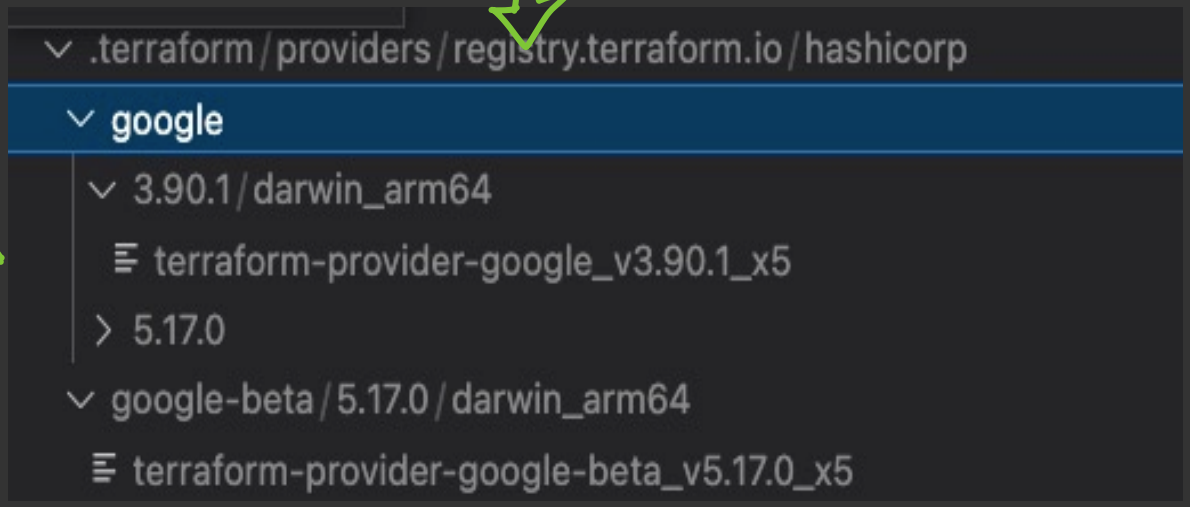
- As the industry shifted to the cloud and having services run at such a large scale, infrastructure became complex. With the ticketing ways, it's a slow process working with IT teams, and human beings are prone to making errors.
- Infra as Code helps teams become self sufficient, if we want to scale up a database, we just raise a PR for it, have it reviewed and we now have a bigger size DB !
- IaC also lets us apply best practices in software engineering like testing the code, testing the changes in different environments before the change is actually applied in production.

Guidelines while using terraform

- Set credentials / do an auth login for the cloud provider that we will be using
- Written in Hashicorp Configuration Language (HCL) in files with .tf extension
- The 'tf' binary comes with basic functionality for tf but not with code for any of the providers
- So when we begin working with any new project, we first define the provider block and then run the -tf init- command so tf downloads the code required for that provider

```
terraform {
  You, 3 days ago | 1 author (You)
  required_providers {
    You, 3 days ago | 1 author (You)
    google = {
      source  = "hashicorp/google"
      version = "~> 5.17"
    }
  }
}
```





Execution plans and refresh

The plan command is used to create an execution plan:

- It does a refresh and then determines which actions are necessary to achieve the desired state. This is a convenient way to check if the execution plan meets

```
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_firewall.allow_app_traffic will be created
  + resource "google_compute_firewall" "allow_app_traffic" {
      + creation_timestamp = (known after apply)
      + destination_ranges = (known after apply)
      + direction          = "INGRESS"
      + enable_logging     = (known after apply)
      + id                 = (known after apply)
      + min_upper          = 0
      + number             = true
      + numeric            = true
      + override_special   = "!#$%&*()-_=+[]{}<>:?"
      + result             = (sensitive value)
      + special            = true
      + upper              = true
    }

Plan: 16 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_public_ip = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Tf plan / apply

```
(base) dhruvparthasarathy@Dhruvs-MacBook-Air tf-gcp-infra-fork % tf refresh
random_id.db_name_suffix: Refreshing state... [id=Mk6mxw]
random_password.password: Refreshing state... [id=none]
google_compute_network.vpc: Refreshing state... [id=projects/webapp-415300/global/networks
google_compute_route.zero_for_webapp: Refreshing state... [id=projects/webapp-415300/globa
google_compute_firewall.deny_ssh: Refreshing state... [id=projects/webapp-415300/global/fi
google_compute_subnetwork.db: Refreshing state... [id=projects/webapp-415300/regions/us-ea
google_compute_global_address.private_ip_address: Refreshing state... [id=projects/webapp-
google_compute_firewall.deny_all_tcp: Refreshing state... [id=projects/webapp-415300/globa
google_compute_subnetwork.webapp: Refreshing state... [id=projects/webapp-415300/regions/u
google_compute_firewall.allow_app_traffic: Refreshing state... [id=projects/webapp-415300/
google_compute_firewall.deny_all_udp: Refreshing state... [id=projects/webapp-415300/globa
google_service_networking_connection.default: Refreshing state... [id=https%3A%2F%2Fwww.go
vicenetworking.googleapis.com]
google_sql_database_instance.instance: Refreshing state... [id=private-instance-324ea6c7]
google_sql_user.user: Refreshing state... [id=webapp//private-instance-324ea6c7]
google_sql_database.database: Refreshing state... [id=projects/webapp-415300/instances/pri
google_compute_instance.app_instance: Refreshing state... [id=projects/webapp-415300/zones

Outputs:

instance_public_ip = "34.23.210.166"
```

Tf refresh

Terraform State

Tf must store the state of our managed infra and config:

- State is used to
  - **Map real world infra** to our config
  - Keep track of metadata
  - **Improve performance** of large infra

When working with teams

- While for individual developers, it makes sense to keep the state information locally for **teams** it is a better practice to **maintain a single state in a centralized location** for all developers to access simultaneously
- **best practices** of programming come into picture here.
- Use version control system to manage conflicts
- use different environments to test changes before pushing to production

What not to push to Git

- .terraform ← has the downloaded binaries of the provider code
- *.testate ← state file
- *.tfstate.bkp ← backup of the state file.

Additional Features

The primary purpose of terraform is to declare resources. Everything else is syntactic sugar on top to make things flexible and convenient.

- A group of resources along with the relationship between them can be gathered together to make a module, which creates a larger unit of configuration
- The tf config will now consist of a root module, where evaluation begins along with a tree of child modules that are called within this root module

Variables have **types**

- If no type constraint is provided, a variable accepts an type
- However, mentioning a type helps us with getting **useful error messages** when something goes wrong
- Also remind users of the module of the type of values that need to be passed in

The general syntax for creating a resource in Terraform is

```
resource "<PROVIDER>_<TYPE>" "<NAME>" {
  [CONFIG ...]
}
```

Referencing Resources

Syntax: <PROVIDER>_<TYPE>.<NAME>.<ATTRIBUTE>