Infrastructure as Code

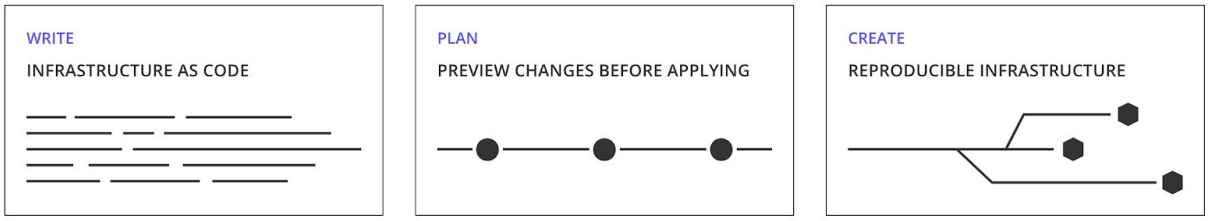
- Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.
- The IT infrastructure managed by this comprises both physical equipment such as bare-metal servers as well as virtual machines and associated configuration resources.
- The definitions may be in a version control system.
- It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.
- The value of IaC can be broken down into three measurable categories:
    - cost (reduction)
    - speed (faster execution)
    - risk (remove errors and security violations)

Immutable Infrastructure

- An immutable infrastructure is another infrastructure paradigm in which servers are never modified after they're deployed.
- If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones.
- After they're validated, they're put into use and the old ones are decommissioned.
- Advantages:
    - Known-good server state and fewer deployment failures
    - No configuration drift or snowflake servers
    - Consistent staging environments and easy horizontal scaling
    - Simple rollback and recovery processes

Terraform

- Allows Infrastructure as Code
- A tool to manage cloud services lifecycle
- Configuration can be written in HCL or JSON



- There’s a set of object types that can be defined with terraform in order to describe our deployment.
- Those objects are:
  - Provider
  - Resource
  - Data

Terraform Resource

- Resources are the most important element in the Terraform language. Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records.
- Resource declarations can include a number of advanced features, but only a small subset are required for initial use.
- A resource block declares a resource of a given type ("aws_instance") with a given local name ("web"). The name is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside of the scope of a module.

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

```
resource "aws_instance" "example" {
  # ...

  tags = {
    # Initial value for Name is overridden by our automatic scheduled
    # re-tagging process; changes to this are ignored by ignore_changes
    # below.
    Name = "placeholder"
  }

  lifecycle {
    ignore_changes = [
      tags["Name"],
    ]
  }
}
```

```
# alternative, aliased configuration
provider "google" {
  alias  = "europe"
  region = "europe-west1"
}

resource "google_compute_instance" "example" {
  # This "provider" meta-argument selects the google provider
  # configuration whose alias is "europe", rather than the
  # default configuration.
  provider = google.europe

  # ...
}
```

Terraform Data Source

- Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration. Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.
- Each provider may offer data sources alongside its set of resource types.

```
data "aws_ami" "example" {
  most_recent = true

  owners = ["self"]
  tags = {
    Name   = "app-server"
    Tested = "true"
  }
}
```

Example

Go to: https://github.com/avielb/tf-demo