Terraform

Everything sucks...but it might be the best tool for the job

Peter Hall
@peterkh
Peter Hall

Principal Engineer @ DigIO

Cloud Infrastructure ~ 8 years

Puppet, Ansible, custom scripts

And lots of CloudFormation

No really… so much CloudFormation
Agenda

- What is Terraform
- My history with Terraform
- The project
- Likes
- Dislikes
- Lessons learnt
Terraform

“write, plan and create infrastructure as code”

HashiCorp open source tool: https://terraform.io

Works across 100s of different services

```terraform
resource "digitalocean_droplet" "web" {  
  name   = "tf-web"  
  size   = "512mb"  
  image  = "centos-5-8-x32"  
  region = "sfo1"  
}

resource "dnsimple_record" "hello" {  
  domain = "example.com"  
  name   = "test"  
  value  = digitalocean_droplet.web.ipv4_address  
  type   = "A"  
}
```
My history with Terraform

- Experimented with v0.2 in 2014
- Continued with CloudFormation until September 2018
- New project, Google Cloud Platform based
- Terraform already in use by other projects
The project

- Large enterprise
- Adopting Google Cloud Platform and Kubernetes across the business
- Green fields project
- Terraform for GCP and configuring Kubernetes resources
- Security and compliance focused
- Multiple teams creating parts of the infrastructure
Features I loved about Terraform
### Modules

Container for multiple resources (i.e. Servers + Load balancer + Firewall Rules)

Module can contain other modules

Inputs / Outputs to modules explicitly defined

All *.tf files in a single directory define a module

Modules can be sourced from git, buckets or HTTP endpoints

```hcl
# webserver_cluster defines an ASG, ALB and Security Groups.
module "web_cluster" {
  source = "../modules/webserver_cluster"
  max_size = "9"
  min_size = "3"
  region   = "ap-southeast-2"
}

resource "dnsimple_record" "hello" {
  domain = "example.com"
  name   = "test"
  value  = module.web_cluster.lb_endpoint
  type   = "CNAME"
}
```
Providers

Lots of providers (well over 100) see: https://www.terraform.io/docs/providers/index.html

Defines service specific resource types

Use as many as required in a single module

Create your own provider as a plugin

```terraform
resource "aws_s3_bucket" "b1" {
  bucket = "my-unique-aws-bucket"
  acl    = "private"
  region = "ap-southeast-2"
}

resource "google_storage_bucket" "b2" {
  name    = "my-unique-gcp-bucket"
  location = "australia-southeast1"
}

resource "kubernetes_config_map" "my_bucket_map" {
  metadata {
    name = "my-bucket-map"
  }
  data = {
    bucket1 = aws_s3_bucket.b1.bucket
    bucket2 = google_storage_bucket.b2.name
  }
}
```
Data sources

Read-only resources

Gives Terraform ability to act based on information external to resources it manages

Defined per provider in addition to resource types

data "google_container_cluster" "gke" {
  name    = "my_cluster"
  region  = "australia-southeast1"
  project = "my_project"
}
Data source – example

data "google_client_config" "default" {}  
data "google_container_cluster" "gke" {   
    name = var.cluster_name   
    region = var.region   
    project = var.project_id  
}  

provider "kubernetes" {   
    load_config_file = false   
    version = "1.8"   
    host = "https://${data.google_container_cluster.gke.endpoint}"   
    token = data.google_client_config.default.access_token  
}
Reusability

- Shared modules library
- Variables provided at run time (.tfvars file, command line or env vars)
- Data sources to configure from environment
- Workspaces
Features I disliked about Terraform
Resource paths

Resources are stored / referenced with full module path

For example, an instance called “a” in my_module has the full path:

module.my_module.aws_instance.a

Need to update state with terraform state command to change path without resource recreation

This change will recreate all resources in my_local_module

module "my_module" {
  source = "modules/my_local_module"
}

module "my_new_module" {
  source = "modules/my_local_module"
}
Conditionals

Only way to do conditional resources in Terraform is via meta-arguments:

- `count`
- `for_each`

Set count to 1 or 0 to simulate if statement

No meta-arguments for modules

```resource "aws_s3_bucket" "b1" {
  count = var.enable_bucket ? 1 : 0
  bucket = "my-unique-aws-bucket"
  acl = "private"
  region = "ap-southeast-2"
}
```
Secrets

No standard method for flagging a field as sensitive

All values passed to resources end up in state file

Some providers implement resource specific encrypting of values in state file

Terraform documentation recommends using remote state with encryption on disk
Lessons learnt on the way
State file management

- Remote backend highly recommended
- Treat state file as critical piece of infrastructure
  - Backups
  - Versioning
  - Least privilege access control
  - Audit logging
- Isolate groups of infrastructure with separate state files
  - Blast radius, performance, different life cycles
Lock versions!

Lock Terraform version... to the specific patch version.

Lock all provider versions... to at least the minor version.

Terraform will only work against state files <= to its version (and upgrade them!)

Test new provider versions in staging environment first

```terraform

terraform {
  required_version = "= 0.12.6"

  backend "gcs" {
    bucket = "tf-state-prod"
    prefix = "terraform/network"
  }
}
```

State File:

```
{
  "version": 3,
  "terraform_version": "0.12.6",
  "serial": 2,
  "lineage": "3b5b0a68-0787-3665-aab6",
  "modules": [
    { ...
  }
}
```

This caused us issues with new versions of Terraform
Module layout

Think about your module layout early

Refactoring can be painful

Sweet spot for modules:

- Too few, lots of duplication
- Too many, unnecessary complexity

https://www.terraform.io/docs/modules/index.html#when-to-write-a-module

terraform/
production/
  main.tf  # includes gke-cluster and storage
non-production/
  main.tf  # same as production + build
modules/
  gke-cluster/
    main.tf  # Creates and configures cluster
build-infra/
  main.tf  # CI pipeline
storage/
  main.tf  # Databases and buckets
Well ok, not everything sucks

- Understand how Terraform expects code to be structured
- Good state management is crucial
- Refactoring is painful, spend some time up front on module layout
- Same tooling across clouds / services is great
- Data sources are awesome
- Shared modules save you time across projects

Overall good option for any project that integrates multiple services
Questions?

@peterkh

https://www.linkedin.com/in/peterkh/