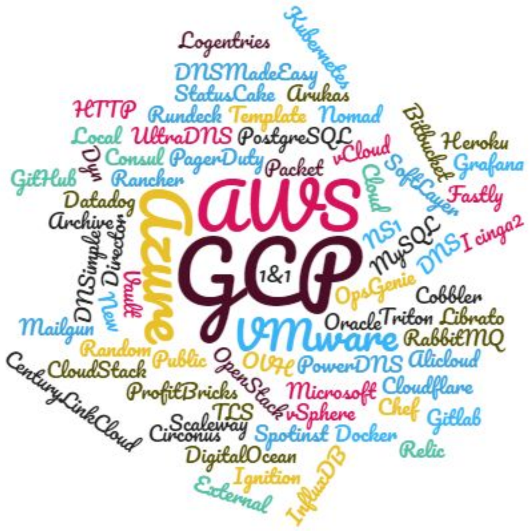
HashiCorp Terraform

Infrastructure-as-code with Terraform

DevOps Pro Camp

Introduction

Terraform is a tool from Hashicorp that allows you to spin up complex infrastructure effortlessly using popular public cloud providers as well as different PaaS. You define all the resources you want using special DSL, and Terraform makes sure that dependent resources in your template are all created in the proper order.



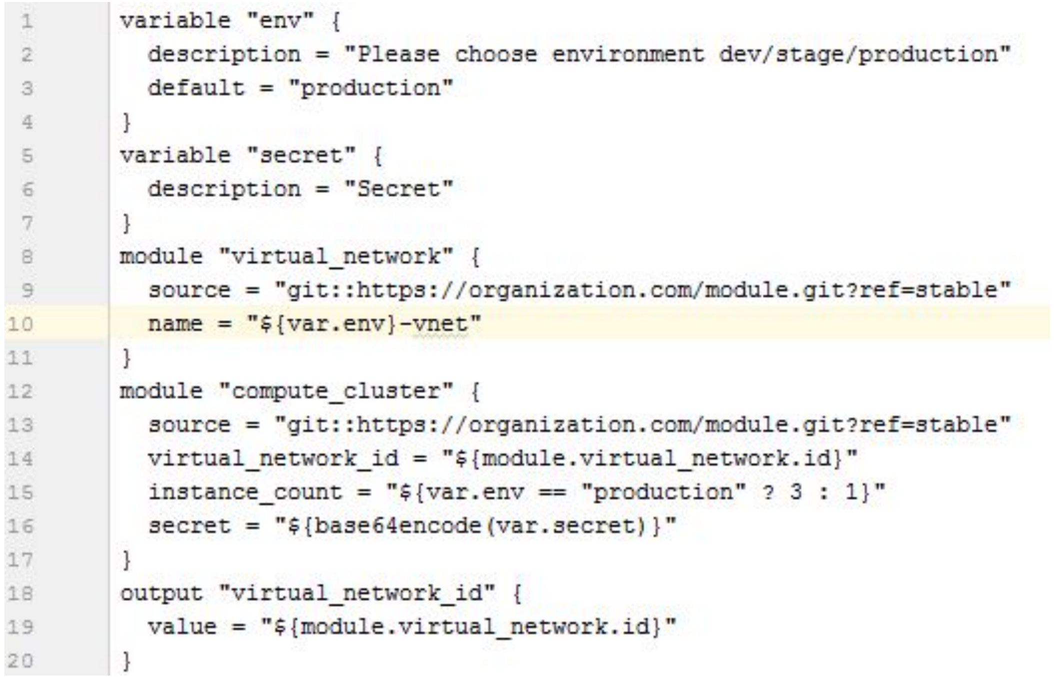
Use cases

- Development/demo environments, prototyping
- Modular infrastructure change management for IaaS/PaaS
- Multi-cloud deployments, infrastructure migration processes

Terraform features overview

HCL

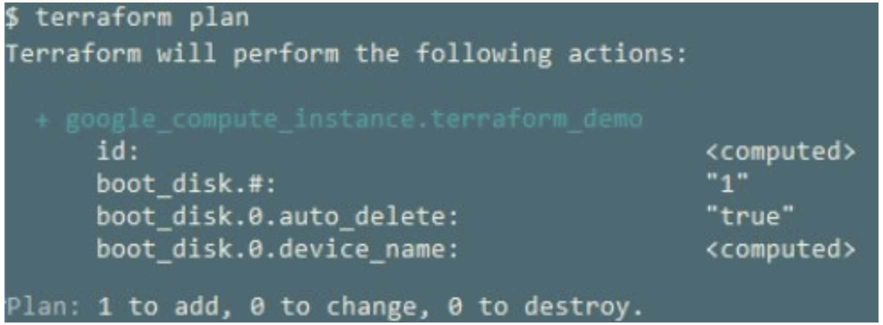
HashiCorp decided to move away from the hard-to-read definitions towards a human-friendly configuration view, and invented their own DSL - HCL.



```
variable "env" {
  description = "Please choose environment dev/stage/production"
  default = "production"
}
variable "secret" {
  description = "Secret"
}
module "virtual_network" {
  source = "git::https://organization.com/module.git?ref=stable"
  name = "${var.env}-vnet"
}
module "compute_cluster" {
  source = "git::https://organization.com/module.git?ref=stable"
  virtual_network_id = "${module.virtual_network.id}"
  instance_count = "${var.env == "production" ? 3 : 1}"
  secret = "${base64encode(var.secret)}"
}
output "virtual_network_id" {
  value = "${module.virtual_network.id}"
}
```

Stateful rollout and planning

Terraform keeps the state of every resource and its metadata. When a new resource needs to be added or modified - Terraform allows the operations engineers to see the detailed plan of operations that will be performed with the specific resources before the actual execution. It allows quality assurance infrastructure procedures to be applied before the changes and decreases the risk of damaging the infrastructure.



```
$ terraform plan
Terraform will perform the following actions:

  + google_compute_instance.terraform_demo
      id:                                        <computed>
      boot_disk.#:                               "1"
      boot_disk.0.auto_delete:                   "true"
      boot_disk.0.device_name:                   <computed>

Plan: 1 to add, 0 to change, 0 to destroy.
```

Terraform provisioners

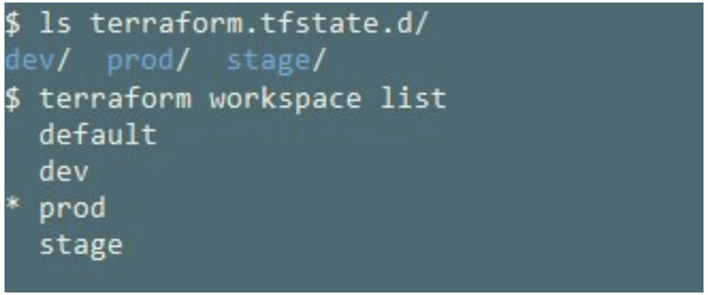
Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction.

- Execute remote command
- Copy file to remote machine
- Use chef/salt config management or habitat supervisor
- Locally execute a command getting data from remote PC
- Configure SSH or WINRM connection
- Provisioners dependency graph with null_resource provisioner

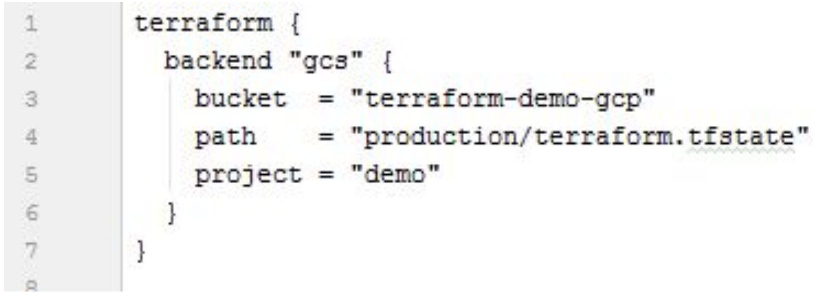
Terraform state storage

Terraform supports complex dependency management workflows, and it is backed by the Terraform state storage system. For each resource terraform stores metadata about its state and dependencies in a state file.

These state files can be local json-encoded *.tfstate* files, or you can use centralized storage for your state using Terraform Enterprise, Consul, S3, Azure and others for a centralized dependency and resource management.



```
$ ls terraform.tfstate.d/
dev/  prod/  stage/
$ terraform workspace list
  default
  dev
* prod
  stage
```

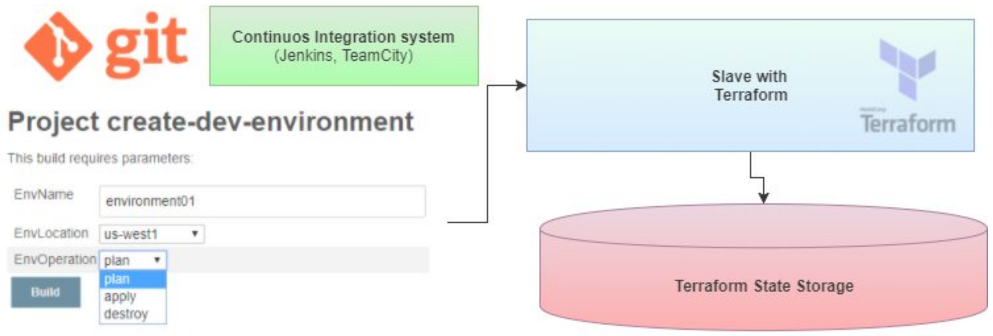


```
terraform {
  backend "gcs" {
    bucket  = "terraform-demo-gcp"
    path    = "production/terraform.tfstate"
    project = "demo"
  }
}
```

Terraform implementation cases

Development environments automation

- Store all the environment definition file in a version control system in own repository
- Use credentials management systems (CI inbuilt or external)
- Use parameterized jobs to work in specific workspaces to simplify environment tracking
- Use durable external storage for Terraform states



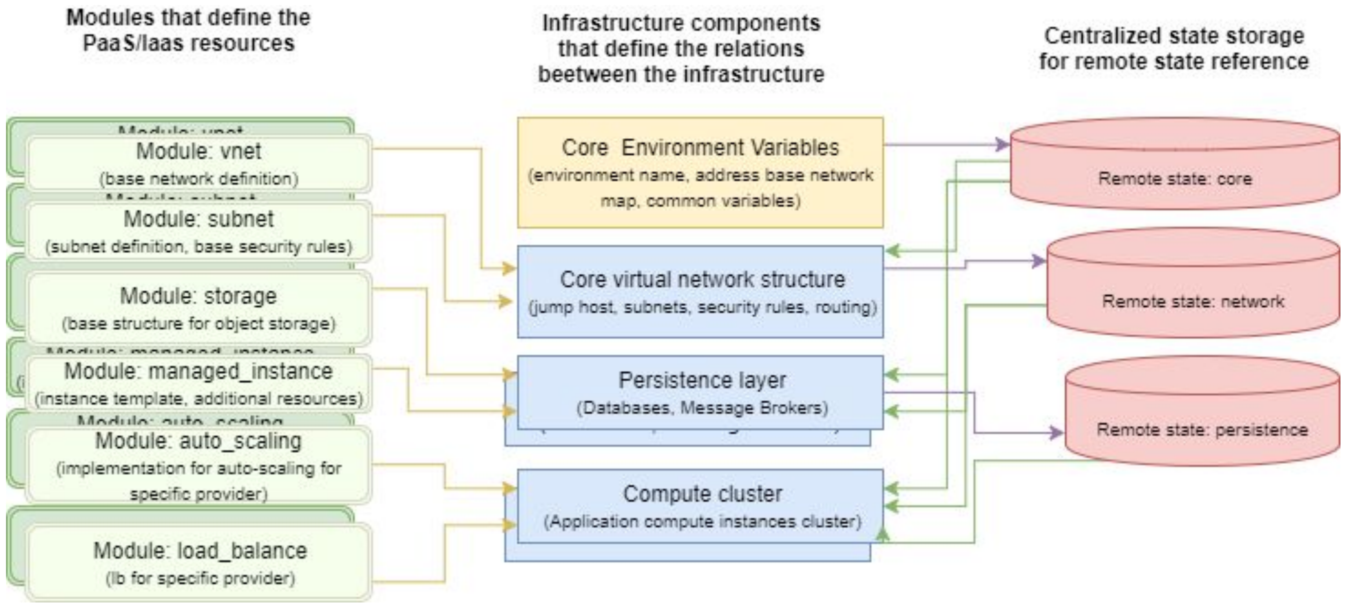
Project create-dev-environment

This build requires parameters:

EnvName	environment01
EnvLocation	us-west1
EnvOperation	plan

Complex infrastructure change management

Terraform allows you to build robust and predictable process for environment operations. Most of corporate deployments currently consist of a globally distributed infrastructure with complex dependencies, and often with a requirement to deploy resources on different cloud providers.


Modules that define the PaaS/Iaas resources
Infrastructure components that define the relations beetween the infrastructure
Centralized state storage for remote state reference
Module: vnet
(base network definition)
Module: subnet
(subnet definition, base security rules)
Module: storage
(base structure for object storage)
Module: managed_instance
(instance template, additional resources)
Module: auto_scaling
(implementation for auto-scaling for specific provider)
Module: load_balance
(lb for specific provider)
Core Environment Variables
(environment name, address base network map, common variables)
Core virtual network structure
(jump host, subnets, security rules, routing)
Persistence layer
(Databases, Message Brokers)
Compute cluster
(Application compute instances cluster)
Remote state: core
Remote state: network
Remote state: persistence

Change management suggestions

- Abstract from cloud provider by standardizing the components and modules dependencies and parameters
- Use modules with semantic versioning for infrastructure creation
- Use terraform remote state
- Isolate the environments, do not use global or hardcoded resources
- Separate the infrastructure components by role
- Constantly test your deployment in a Continuous Integration environment with new terraform versions and modules
- Use same versions of terraform across development and production environments

Security considerations

- Use single execution host
- Integrate with credentials management systems for sensitive information/secrets
- Use state files locking
- Constantly backup state files
- Provide logging for all terraform operations and environment when any terraform action is executed
- Always check the terraform plan before applying changes
- Separate development and production operations not only by environment/workspace, but use different variables, credentials, and even different Terraform host

Practical Tasks

In-class practice

Install and set terraform and AWS keys for eu-west-2

https://github.com/runonautomation/aws-tf-training

Network structure

Sample instance with a provisioner

Destroy

Flow:

Install Terraform

Modules overview

Plan

Apply

Remote storage state

Build

Change

Edit provisioners

Use outputs

Destroy

Thank you