

Building GCP with Terraform



Google Cloud



Terraform

Greg Horie

Overview

- What is IaC?
- Procedural vs. Declarative
- What is Terraform?
- Basics Workflow
- HCL
- Providers
- State File
- Resources, Variables, Modules
- Demos!



What is IaC?

- IaC = Infrastructure-as-Code
- High-level coding languages used to automate IT infrastructure builds.
- **IaC Tool Adoption**

IaC Tool	Overall	Enterprise	SMB
Terraform	36%	36%	39%
Ansible	31%	33%	29%
Chef	29%	32%	21%
Puppet	27%	26%	18%
Salt	12%	14%	10%

* Taken from Flexera 2021 State of Cloud report.

- Not an apples-for-apples comparison.
- In practice, these tools are often used together to fully-automate builds.

Procedural vs. Declarative

- IaC tools can be categorized as Procedural or Declarative.
 - Based on how the tool applies changes to the underlying target systems.
- **Procedural**
 - Think script.
 - Step-by-step tasks to automate your build.
 - Ansible is procedural - Code step-by-step tasks to achieve a desired end state.
- **Declarative**
 - Think HTML.
 - Code describes the desired state, not the steps.
 - Hides the process and reveals relationships.
 - Terraform is declarative - Code specifies your desired end state.

What is Terraform?

- Terraform is an open source IaC tool.
- Enables you to build, change, and version cloud and on-prem resources.
- Original author - Mitchell Hashimoto.
 - Maintained by Hashicorp.
- Enabled through the use of "Providers".
 - Providers = plugins which are an abstraction over an API.
 - Many providers are available through the public Terraform Registry.
 - Support for all major cloud providers.
- Modules are also available to further abstract provider interactions.
 - Public modules are available and you can also build your own.



Terraform Installation

- <https://learn.hashicorp.com/tutorials/terraform/install-cli>

Quick Demo

- Using Terraform's local provider:

<https://registry.terraform.io/providers/hashicorp/local>

- Command line notes:

<https://github.com/netserf/netsig-presentation-building-gcp-with-terraform/terraform-cheat-sheet.txt>

Basic Workflow

1. Code

- Write your infrastructure configuration.

2. Initialize

- Initialized providers and modules.
- `terraform init`

3. Plan

- Displays the difference between desired and current state.
- Creates a plan to reconcile this gap.
- `terraform plan`

4. Apply

- Apply the changes to achieve the desired state.
- `terraform apply`

HCL

- HCL = Hashicorp Configuration Language
- Toolkit for creating structured configuration languages
- Goal is to be both human-readable and machine-friendly
- Example:

```
default_address = "127.0.0.1"  
default_message = upper("Incident: ${incident}")  
default_options = {  
  priority: "High",  
  color: "Red"  
}
```

Providers

- Provider = Terraform plugin that typically integrates with an external API.
- Acts as a translation layer
- Allows Terraform to communicate with an API to build external resources
 - e.g. VMs, databases, services, etc.



State File - terraform.tfstate

- Terraform stores information about your infrastructure in a state file
 - File name: `terraform.tfstate`
 - State keeps track of resources modified by your IaC config
- During `terraform plan`
 - Terraform compares config with state to plan changes to your infrastructure
- During `terraform apply`
 - Terraform makes updates to resources and writes metadata to the state file to reflect changes
- Terraform supports a number of state file operations:
 - `terraform show`
 - `terraform state list`
 - `terraform state -help`

Resources

- Resources are the most important element in Terraform
- Each resource block describes one or more infrastructure objects
 - e.g. Virtual networks, VM instances, or higher-level components such as DNS records.
- Example:

```
resource "google_compute_instance" "test_instance" {  
  name          = "test-instance"  
  machine_type  = "e2-medium"  
  zone          = "northamerica-northeast1-a"  
  
  boot_disk {  
    initialize_params {  
      image = "debian-cloud/debian-11"  
    }  
  }  
}
```

Variables

- Variables allow for customization of Terraform modules
- Makes your module reusable
- Variables can be set using files, CLI options and/or environment variables.
- Example:

```
variable "machine_type" {  
    type      = string  
    default = "e2-medium"  
}
```

Modules

- A module is a container for multiple resources that are used together
- Modules can abstract the underlying resource implementation
 - Promotes reuse
- The .tf files in your working directory form the root module
- The root module may call other modules
- Example:

```
module "cloud-storage_simple_bucket" {  
  source = "terraform-google-modules/cloud-storage/google//modules/simple_bucket"  
  version = "3.4.0"  
  name = "my-bucket"  
  project_id = "my-project"  
  location = "northamerica-northeast1"  
}
```

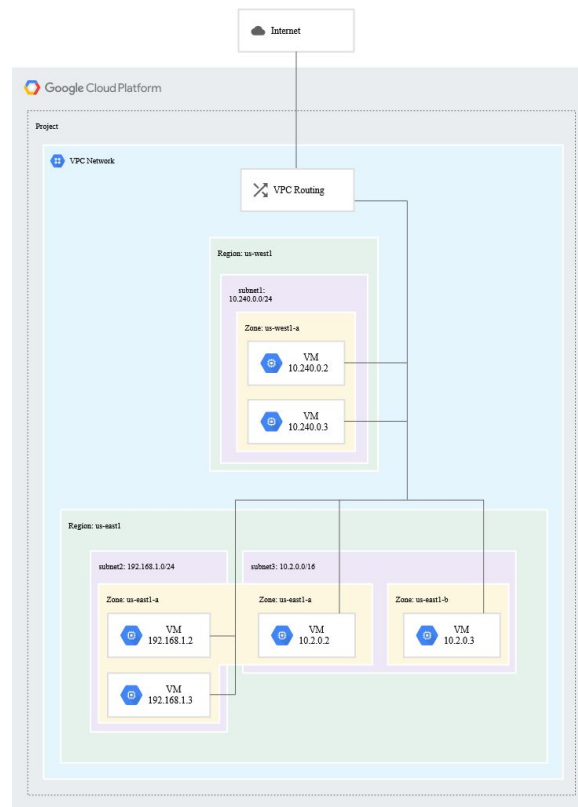
GCP Build Network

- Command line notes:

<https://github.com/netserf/netsig-presentation-building-gcp-with-terraform/terraform-cheat-sheet.txt>



Google Cloud



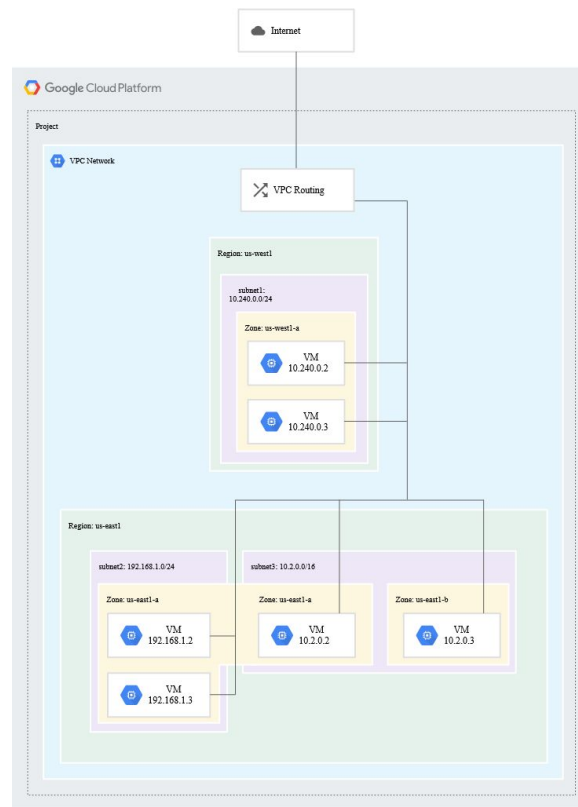
GCP Build Virtual Machines

- Command line notes:

<https://github.com/netserf/netsig-presentation-building-gcp-with-terraform/terraform-cheat-sheet.txt>



Google Cloud



Summary

- Terraform is a declarative infrastructure-as-code configuration language
- Used to standardize the build process in a variety of environments
- In particular, it is helpful in building cloud based infrastructure



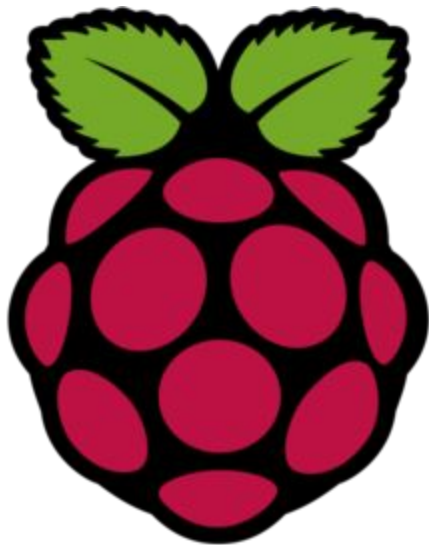
Possible Future Discussions

- Go deeper with GCP networking
 - IPv6, DNS, Load Balancing, VPNs, etc.
- Building AWS with Terraform
- Kubernetes / GKE
 - Service Mesh
- Monitoring / Log Management
 - e.g. Prometheus, Elastic Stack, etc.
- CI / CD
 - GitHub Actions
- PaaS Apps
 - e.g. GCP's AppEngine
- Other ideas welcome!



VicPiMakers and Others Slack

- Please let us know if you want an invite to this Slack group



Backup Slides

