HashiCorp

Terraform

How Terraform Works:

- Written in Golang
- Interfaces with the API of the “provider”
- Create
- Read
- Update
- Delete

```
resource "docker_image" "nodered_image" {
  name = "nodered/node-red:latest"
}
```

= *docker pull nodered/node-red:latest*

Core Terraform Workflow:

```
resource "docker_image" "image_id" {
  name = "nginx"
}

resource "docker_container" "container_id" {
  name  = "nginx"
  image = docker_image.image_id.latest
  ports {
    internal = "80"
    external = "8080"
  }
}
```

Write

```
  # docker_image.image_id will be created
  + resource "docker_image" "image_id" {
      + id     = (known after apply)
      + latest = (known after apply)
      + name   = "nginx"
    }

Plan: 2 to add, 0 to change, 0 to destroy.
```

Plan

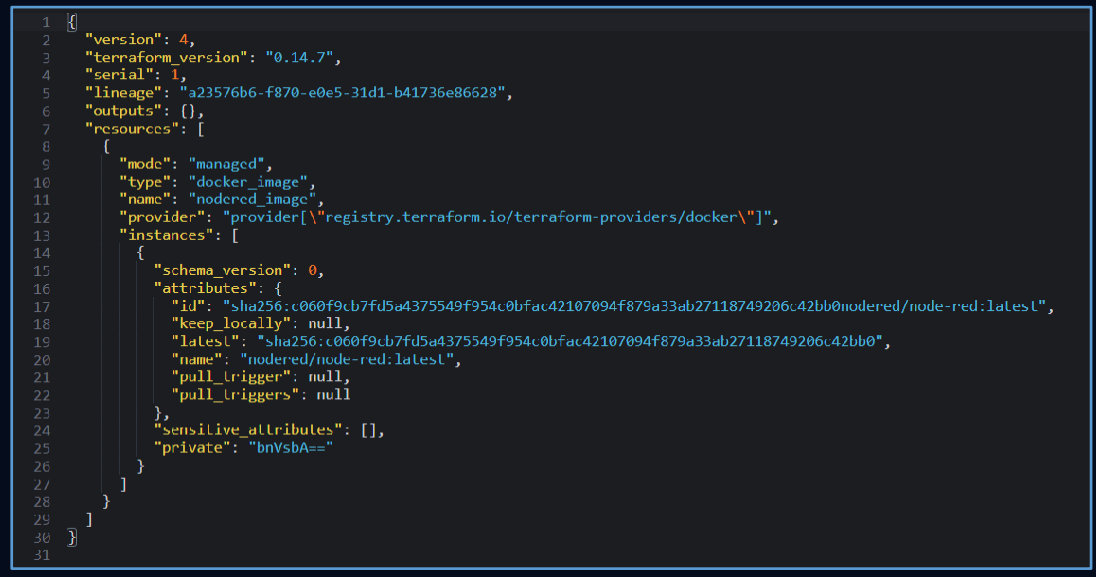
```
docker_image.image_id: Creating...
docker_image.image_id: Creation complete after 5s [id=sha256:f
docker_container.container_id: Creating...
docker_container.container_id: Creation complete after 1s [id=

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Apply

Terraform State

- Stores information about the current environment
- Is created based on the configuration files and any changes are committed to the infrastructure via the API
- Only knows about resources created by it. If those resources are missing, it can replace, but cannot see other resources.



IaC Workflow:

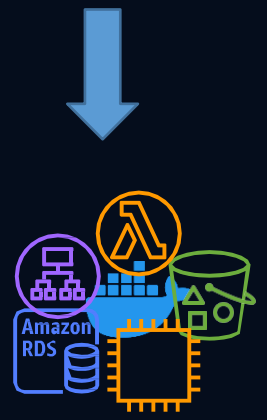
```
resource "docker_image" "image_id" {
  name = "nginx"
}

resource "docker_container" "container_id" {
  name  = "nginx"
  image = docker_image.image_id.latest
  ports {
    internal = "80"
    external = "8080"
  }
}
```

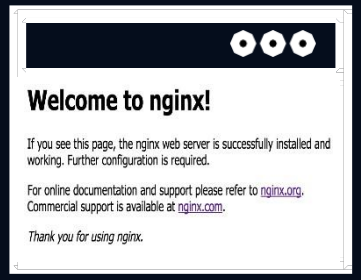
Terraform Code

Git Repository

CICD Tools



Infrastructure



Application

Idempotence

- Can run code as many times as you like while still maintaining the resources specified.
- One reason the “local-exec” provisioner isn’t recommended.
- Isn’t always true. You should ALWAYS verify your plan before applying infrastructure in production.

HashiCorp
Terraform

Declarative vs. Procedural

Declarative

- WHAT do you want the final deployment to look like?
- "I want a VPC and 2 EC2 Instances that are connected to an IGW for internet access"
- Requires "state"
- Process is more abstracted
- "Idempotent"
- Primary Terraform operation

Procedural (Imperative)

- HOW do you want to deploy resources?
- "Create the VPC first, then create the IGW, then create the EC2 instances"
- Not dependent on state
- More control over the process
- Running an operation twice will still perform the operation, regardless of its previous execution or the damage it can cause
- Terraform can perform Imperative tasks, but it is best practice to keep the code as declarative as possible

Terraform vs. Azure Tooling

HashiCorp Terraform

- Open source
- HCL Syntax
- Vendor Neutral
- Requires state management and storage
- Requires resources to run
- Requires logging infrastructure
- Breaking changes are generally more likely

CloudFormation

- Closed source
- JSON/YAML Syntax
- Only useful with Azure resources
- State is managed by Azure

The Docker Provider

Do Lab 6

https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs

Terraform Init Deeper Dive

https://developer.hashicorp.com/terraform/cli/commands/init

Terraform Dependency Lock

https://developer.hashicorp.com/terraform/language/files/dependency-lock

Terraform Apply

https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs/resources/image.html

Plan and Apply Deeper Dive

https://developer.hashicorp.com/terraform/cli/commands/apply

Referencing other Resources

https://developer.hashicorp.com/terraform/language/expressions#references-to-resource-attributes