Methods for managing Terraform on a team

Otherwise known as, “Oh shit, how do we share this state file?????”

Alias: asenchi

- I am known as Curt Micol in real life
- In the past 19 years I have had the following titles (nothing changes but the name):
  - System Administrator
  - Operator
  - DevOps Engineer
  - Infrastructure Engineer
  - Site Reliability Engineer (SRE)
- I currently work at Heptio as a Senior Site Reliability Engineer
- Previously I have had the pleasure of working at EngineYard, Heroku, GitHub and Simple

HashiCorp Terraform

A simple resource

```
# variables.tf
variable "max" {
  default = 999
}
# main.tf
resource "random_integer" "test" {
  min = 1
  max = "${var.max}"
}
# outputs.tf
output "test_integer" {
  value = "${random_integer.test.result}"
}
```

input > filter > output

Truncated output of a `terraform plan`

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + random_integer.test
      id:     <computed>
      max:    "999"
      min:    "1"
      result: <computed>


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

Truncated output of a `terraform apply`

```
random_integer.test: Creating...
  max:    "" => "999"
  min:    "" => "1"
  result: "" => "<computed>"
random_integer.test: Creation complete after 0s (ID: 691)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

...

Outputs:

test_integer = 691
```

A dependency

```
# main.tf
resource "random_integer" "test" {
  min = 1
  max = "${var.max}"
}

resource "random_integer" "static" {
  min  = 1
  max  = 989
  seed = "${random_integer.test.result}"
}
```

Diff a state file change after adding resources

```
-    "serial": 2,
+    "serial": 3,
     "lineage": "7612e329-a630-8a91-9601-a1ad07759177",
     "modules": [
         {
@@ -16,6 +16,26 @@
                 }
             },
             "resources": {
+                "random_integer.static": {
+                    "type": "random_integer",
+                    "depends_on": [
+                        "random_integer.test"
+                    ],
...
```

Intent vs. Realized

Managing state is painful.

Terraform has a decent answer.

Remote state management

```
# terraform.tf
terraform {
  required_version = ">= 0.11.7"
  backend "s3" {
    bucket = "$S3_BUCKET"
    key = "my_module/terraform.tfstate"
    encrypt = "true"
    region = "us-west-1"
    dynamodb_table = "my_table"
    kms_key_id = "arn:aws:kms:us-west-1:$AWS_ACCOUNT_ID:key/$KMS_KEY_ID"
  }
}
```

Setting up remote state is painful.

- You need a bucket
- You need a bucket policy
- You need a KMS key, because everyone should encrypt S3 content (it's super cheap, do it)
- You need a dynamodb table for locking (there is no "I" in "team")
- You *should* have a role for accessing those
- You probably want a group to put users in so they can use that role
- You need policies to go with that role
- You need a way to manage all of this!

Managing *remote state is painful.

Terraform has a decent answer.

What happens if terraform fails to realize our intentions?

Break your resources into logical entities

TO THE TERMINAL!

Wrapping up


heptio