Migrating HealthCare.gov to Terraform: Lessons Learned

Christian Monaghan
@monaghan_a_gram
Cofounder, Nava PBC
What is Terraform?
A tool for building, changing, and versioning infrastructure
Manage cloud providers
Infrastructure as Code

- Declarative syntax
- Source control
- Variable support

```hcl
provider "aws" {
  access_key = "super_secret_key"
  secret_key = "another_secret_key"
  region     = "us-east-1"
}

resource "aws_instance" "example" {
  ami     = "ami-2757f631"
  instance_type = "t2.micro"
}

resource "aws_eip" "ip" {
  instance = "${aws_instance.example.id}""
Execution plans

- Developer reviews plan before proceeding

```
$ terraform apply

+ aws_eip.ip
  domain:
  instance:
  network_interface:
  private_ip:
  public_ip:
...

+ aws_instance.example
  ami:
  availability_zone:
  instance_state:
  instance_type:
  key_name:
  private_dns:
  private_ip:
  public_dns:
  public_ip:
  source_dest_check:
...
```

```
"<computed>"
"${aws_instance.example.id}"
"<computed>"
"<computed>"
"<computed>"

"ami-b374d5a5"
"<computed>"
"<computed>"
"t2.micro"
"<computed>"
"<computed>"
"<computed>"
"<computed>"
"<computed>"
"true"
```
Resource graph

- Resources created in dependency order

```yaml
# ...
aws_instance.example: Creating...
  ami: "" => "ami-b374d5a5"
  instance_type: "" => "t2.micro"
[..]
aws_instance.example: Still creating... (10s elapsed)
aws_instance.example: Creation complete
aws_eip.ip: Creating...
  allocation_id: "" => "<computed>"
  association_id: "" => "<computed>"
  domain: "" => "<computed>"
  instance: "" => "i-f3d77d69"
  network_interface: "" => "<computed>"
  private_ip: "" => "<computed>"
  public_ip: "" => "<computed>"
aws_eip.ip: Creation complete

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

- Resources created in dependency order
Our project history
AWS Cloudformation

JSON interface

3,000+ lines for 1 Virtual Private Cloud (VPC)

Managing dozens of VPCs
Custom tooling to interact with Cloudformation

YAML Config → Custom script → AWS Cloudformation
Challenges we faced with our existing tooling
Maintaining custom code :(

- Complex
- Not unit tested
- Limited documentation, quickly out of date
- Increasing bloat
- Hard to understand
- Hard to debug
Unable to incorporate manual changes

Past examples:

- Horizontally scale NATs (Network Address Translation)
- Adding a temporary second Elastic Load Balancer
- Scaling down from 3 availability zones to 1 availability zone
- Swap in new Elastic IPs
Uncertain client demands

- Must build atop partially provisioned vpc infrastructure
- Client frequently requesting custom architecture changes
- Client might make manual changes that would be unrecoverable in Cloudformation
Proliferating use cases

- Load testing resources
- Continuous Integration clusters
- Custom monitoring
- Graphite/Graphana
- Nessus scanning clusters
We were trying to shoehorn all these new use cases into our existing tooling
Engineering goal
Manage all infrastructure with a single tool that is flexible, extensible, fast, and well-supported
Choosing the right tool
Tools we considered

- CloudFormation
- ANSIBLE
- Puppet Labs
- SaltStack
- Terraform
- CHEF
Chef, Puppet, Ansible, SaltStack

- These are **configuration management** tools
- Install and manage software on **existing** machines
Why we chose Terraform

- Incorporate manual changes
- Declarative syntax, easy to read, understand, extend
- Supports multiple providers
- Separates planning and execution
- Well-supported, open-source
- Modular
Some Terraform basics
How it knows what to provision

- Desired state
- Actual state
- Changes required
Desired state looks like this

```hcl
provider "aws" {
  access_key = "ACCESS_KEY_HERE"
  secret_key = "SECRET_KEY_HERE"
  region     = "us-east-1"
}

resource "aws_instance" "example" {
  ami         = "ami-2757f631"
  instance_type = "t2.micro"
}
```
Actual state looks like this

```
$ terraform show
aws_instance.example:
  id = i-32cf65a8
  ami = ami-2757f631
  availability_zone = us-east-1a
  instance_state = running
  instance_type = t2.micro
  private_ip = 172.31.30.244
  public_dns = ec2-52-90-212-55.compute-1.amazonaws.com
  public_ip = 52.90.212.55
  subnet_id = subnet-1497024d
  vpc_security_group_ids.# = 1
  vpc_security_group_ids.3348721628 = sg-67652003
```
Prototyping
Greenfield approach

Define ➔ Diff ➔ Apply ➔ State Updated
Reverse engineering approach

Define → Diff → Apply → Import State
Refactor to use variables

Hardcoded

```hcl
resource "aws_vpc" "default" {
  cidr_block = "10.0.0.0/24"
}
tag {
  "Name" = "hcgov-sls-prod"
}
```

Variables

```hcl
variable "vpc_name" {
  default = "hcgov-sls-prototype"
}
variable "vpc_cidr" {
  default = "10.0.1.0/24"
}
resource "aws_vpc" "default" {
  cidr_block = "${var.vpc_cidr}"
  tags {
    "Name" = "${var.vpc_name}"
  }
}
```
Testing

1. Successfully provision a new VPC
2. Application functional
   a. Passes health checks
   b. Passes smoke testing
3. Infrastructure security scan
   a. AWS Trusted Advisor
End result

- A configuration file (.tf) that represents one complete vpc configuration
- A state file (.tfstate) that represents one existing vpc
Design
How can we design this for reuse?
Existing design

Variable inputs

Assemble building blocks

Building blocks
Implementation
Build new VPC's & cutover traffic
Learnings
Use shared modules sparingly
Use shared modules sparingly

Sharing modules within applications worked well
Use shared modules sparingly

Sharing modules **across** applications did not work well
Use shared modules sparingly

Change the **Elastic Load Balancer** module
Use shared modules sparingly
Use shared modules sparingly
Migrating infrastructure in place

It's possible, but time consuming
Importing existing state

- Native `terraform import` CLI utility
  - Only imports one resource at a time
  - Requires manually finding each resource id relevant to a particular vpc

- Third party open source `terraforming CLI`
  - Imports all resources in a region
  - Cannot narrow scope to a specific vpc
Lock resources to a particular terraform version

terraform {
  backend "s3" {
    bucket = "aws-xxx-xxx-xxx-xxx-us-east-1"
    key   = "hcgov-sls-prod/terraform/terraform.tfstate"
    region = "us-east-1"
    dynamodb_table = "tf_lock"
  }
  required_version = "~> 0.11.7"
}
Terraform needs to be managed in CI/CD

Otherwise:

- Risk losing internet connection in mid-apply
- No record of who changed what when
- Developers bump versions unintentionally
Semantically version modules with git tags

Good

```hcl
module "jump" {
  source = "git@github.com:CMSgov/terraform//jump?ref=1.0.5"
}
```

Bad

```hcl
module "jump" {
  source = "git@github.com:CMSgov/terraform//jump?ref=598a8ebe3e428b37e806668995d7ff5ac20f1d7a"
}
```
Terraform utilities
Export existing AWS resources to Terraform

[HTTP link to Terraform project]

Tags: ruby-gem, terraform, tfstate, aws

Project details:
- 977 commits
- 8 branches
- 34 releases
- 44 contributors

Latest commit: 1ba96b9 29 days ago

- bin: Rename exe/ to bin/
- contrib/zsh-completion: Add zsh-completion
- lib: Fix cross-account security group reference
tfenv

Terraform version manager inspired by rbenv

Terraform version manager

208 commits
1 branch
12 releases
18 contributors

genevera Merge pull request #91 from Piloxa/master

bin
run tfenv as a neighbour with full path (to keep vscode and whoever d...  a year ago
libexec
Fix for macOS > 10.12  3 months ago
test
Merge branch 'master' into version-from-sources 11 months ago
terraform fmt

Before

```hcl
resource "aws_cloudwatch_metric_alarm"
  alarm_name = "ec2-\${var.vpc_name}-web"
  count = "\${var.monitor_web_cpu}"
  comparison_operator = "GreaterThanOrEqual"
  namespace = "AWS/EC2"
  metric_name = "CPUUtilization"
  statistic = "Average"
  unit = "Percent"
  # Trigger alarm on > 90% CPU for 60s
  threshold = "60"
  period = "60"
  evaluation_periods = "2"
  # Alarms are defined
  alarm_actions = ["\${local.cloudwatch_ok_actions}
                   \${local.cloudwatch_ok_actions}
                   \{AutoScalingGroupName = "\${module.app_autoscaling_group_name}
                   \{DimensionName = "InstanceID",
                   "AutoScalingGroupARN"
                   \}
                   \}
```

After

```hcl
resource "aws_cloudwatch_metric_alarm"
  alarm_name = "ec2-\${var.vpc_name}-web"
  count = "\${var.monitor_web_cpu}"
  comparison_operator = "GreaterThanOrEqual"
  namespace = "AWS/EC2"
  metric_name = "CPUUtilization"
  statistic = "Average"
  unit = "Percent"
  # Trigger alarm on > 90% CPU for 60s
  threshold = "60"
  period = "60"
  evaluation_periods = "2"
  # Alarms are defined
  alarm_actions = ["\${local.cloudwatch_ok_actions}
                   \${local.cloudwatch_ok_actions}
                   \{AutoScalingGroupName = "\${module.app_autoscaling_group_name}
                   \{DimensionName = "InstanceID",
                   "AutoScalingGroupARN"
                   \}
                   \}
```
terraform-docs

Generate docs from terraform modules

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>subnet_ids</td>
<td>a comma-separated list of subnet IDs</td>
<td>-</td>
<td>yes</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpc_id</td>
<td>The VPC ID.</td>
</tr>
</tbody>
</table>
Thank you

@monaghan_a_gram