

Building a Munki repo in 3 minutes with Terraform



GRAHAM GILBERT • JUNE 14 2019 • MACDEVOPS:YVR

Who?

- Systems Engineering @ Airbnb
- Serial open sourcer
- Automation obsessive
- Cancer survivor
- <https://graham.at/movember>



Hi everyone. My name is Graham, I'm on the systems engineering team at airbnb and I have a problem where I feel the need to give all of my software away. I also have a problem with automating all of the things. I kicked testicular cancer in the balls last year, so if you find this quick talk helpful in any way, please find your way to graham.at/movember



This is not a session on what Munki is. Today we are only going to concern ourselves with setting up the required infrastructure in AWS to support running a Munki server at scale.



This is however, a session on what Terraform is. Terraform allows you to stand up infrastructure in many places - Amazon Web Services, Google Compute Platform, Azure, Digital Ocean, VMware - you name where you might want to run infrastructure, there is probably a terraform provider for it.

Beachballs as code

```
beachball "my-lovely-beachball" {  
  size    = "small"  
  colours = "many"  
}
```

Let's say I have a beachball that I need to define as code. (ask someone to come up on stage as a volunteer). Ask the volunteer if they have a small beachball with many colors.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the output of a Terraform plan command.

```
$ terraform plan
+ beachball.my-lovely-beachball:
  id:      <computed>
  size:    "small"
  colours: "many"

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

So by asking **person** if they have the beachball, we have run our terraform plan. To correct our state, terraform knows it needs to create a beachball.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the output of a Terraform apply command. The output shows the creation of a resource named 'beachball.my-lovely-beachball' with attributes id, size, and colours. It also shows the completion of the apply operation with resource counts.

```
$ terraform apply -auto-approve
beachball.my-lovely-beachball: Creating...
  id:      "" => "<computed>"
  size:    "" => "small"
  colours: "" => "many"
beachball.my-lovely-beachball: Creation complete after 1s (ID: E1XCTQI020YA7A)

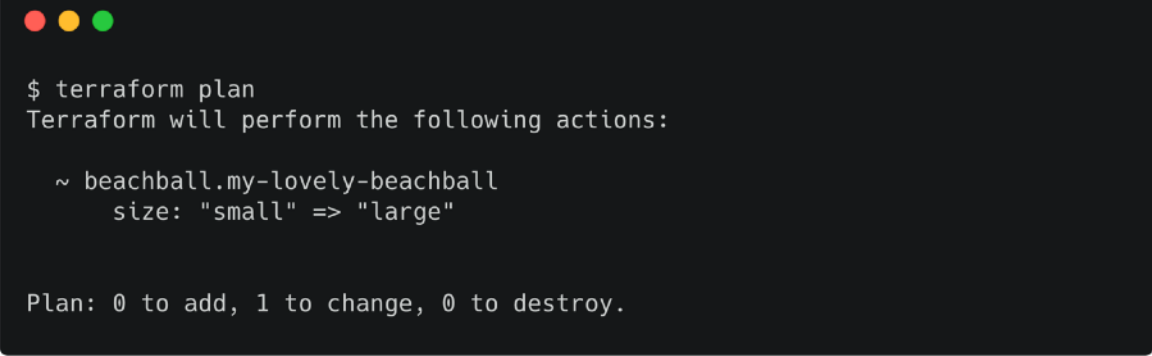
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

So now it's time to run our apply. (give person the beachball). Terraform is instructing beachball web services what kind of beachball it should create.

**We're gonna need a
bigger beachball**

```
beachball "my-lovely-beachball" {  
  size    = "large"  
  colours = "many"  
}
```

So we decide we want to make our beachball bigger. Ask the person what size beachball they have.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the output of the 'terraform plan' command.

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

~ beachball.my-lovely-beachball
  size: "small" => "large"

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

So by asking **person** what size ball they have, we can determine that they are divergent from our desired state, we know what we need to do to make the ball the size we want

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the output of a Terraform apply command. The output shows a resource being modified, with specific attribute changes listed, and a final summary of the apply operation.

```
$ terraform apply -auto-approve
beachball.my-lovely-beachball: Modifying... (ID: E1XCTQI020YA7A)
  size:      "small" => "large"
  colours:   "many"  => "many"
beachball.my-lovely-beachball: Creation complete after 0s (ID: E1XCTQI020YA7A)

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

Give person the bigger beachball. So now I'm telling beachball web services to make our correction. So in a nutshell that's how terraform works. It talks to APIs and then tells those APIs what to do. It doesn't create the resources directly, which is what makes it so powerful. For example, we manage parts of our firewall with terraform, as it has an api.

**But weren't we talking
about Munki?**

What do we need to create?



S3 Bucket

Store our packages, pkginfos, manifests and catalogs



CloudFront Distribution

Serve our Munki repo as fast as possible globally



Lambda@Edge

Code to provide Basic Authentication on our repo



Amazon Certificate Manager

Managed TLS for our CloudFront Distribution

So what infrastructure will we need to deploy for our Munki repo? {click} We definitely will need somewhere to serve our files from. We could have stood up a linux box running apache or nginx, but I'm lazy and amazon can scale storage a lot better than I can, so we'll use an s3 bucket. {click} Not all of our users are in one location. We have them all over the world so we should make use of a content delivery network to get the bits to our users as quick as possible. Even if your users are all in one spot, bandwidth from cloud front is cheaper than s3. {click} We want to keep our repo secure, so Lambda@Edge will run every time someone hits our cloud front distribution and injects basic authentication in. Finally, there's no point having basic authentication on our repo if it's all sent in the clear. ACM will handle all the certificate things for us, so we don't need to think about renewals or anything like that.

That sounds like a lot of work

If only there were some way to modularize Terraform...

Yep, that sounds like a lot. And standing up a Munki repo is something a lot of people want to do. It seems silly that everyone needs to configure the parts by themselves. If only there were some way to package up this group of resources

Modules to the rescue

```
module "munki-repo" {  
  source      = "grahamgilbert/munki-repo/aws"  
  version    = "0.1.0"  
  munki_s3_bucket = "macdevops2019-munki"  
  username    = "munki"  
  password    = "ilovemunki"  
  prefix      = "macdevops-2019"  
}
```

Terraform of course has modules - they allow us to package up common configuration items. In this case I've made a Munki repo module that is published on the terraform registry so others can easily stand up their Munki repo without having to worry about arn's or acm's

Building a Munki repo in 3 15 minutes with Terraform



GRAHAM GILBERT • JUNE 14 2019 • MACDEVOPS:YVR



Time to see if our demo worked

That's it!

- [@grahamgilbert](#)
- <https://grahamgilbert.com>
- <https://graham.at/movember>
- https://github.com/grahamgilbert/macdevops_2019