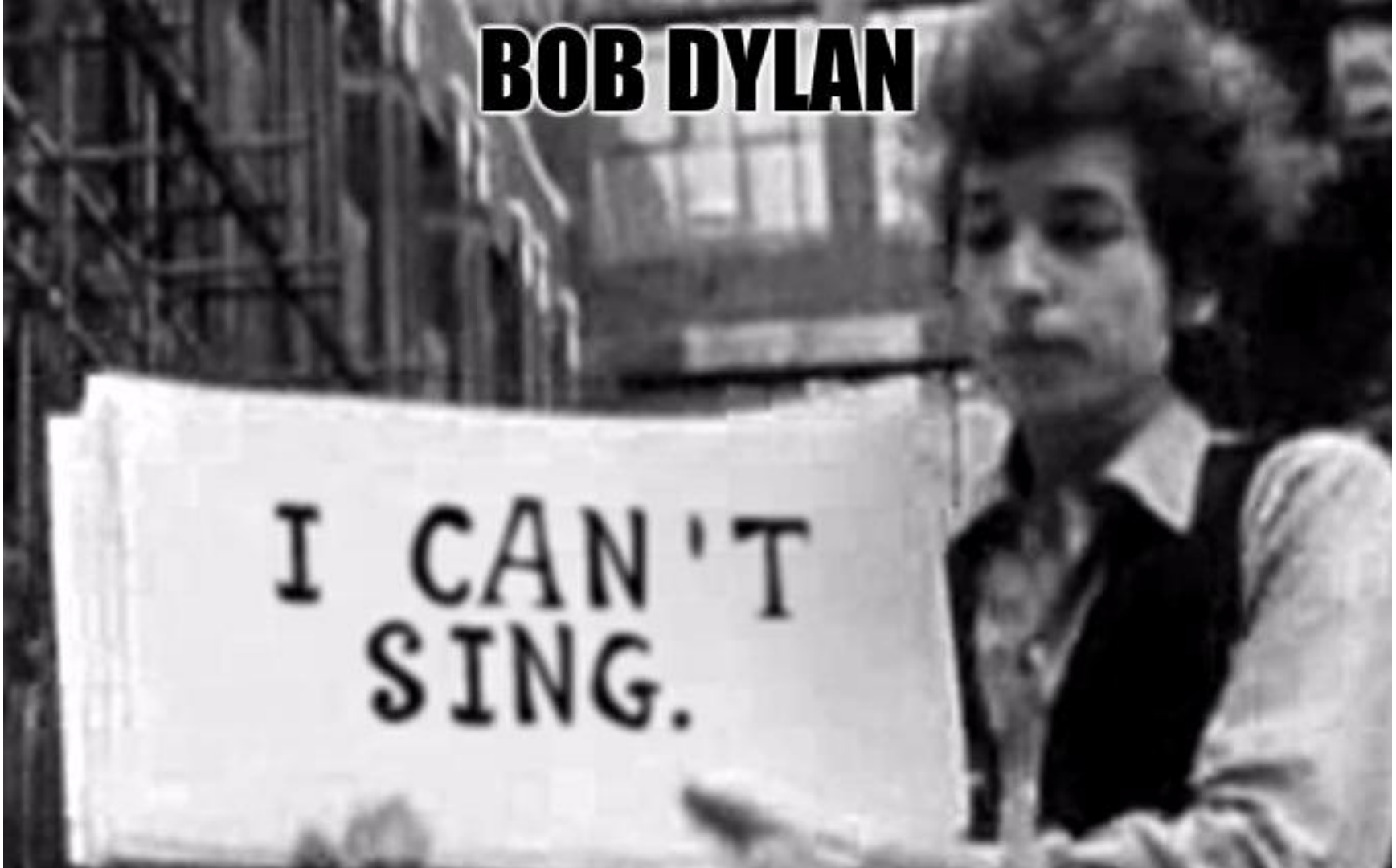
Terraform on AWS... Brownfield... multiregion

Just works? Worth it?

Presentation based on work performed
on contract at Twitter 2017-2019

alexi@alexikaruna.com


BOB DYLAN
I CAN'T
SING.

Goals of Talk

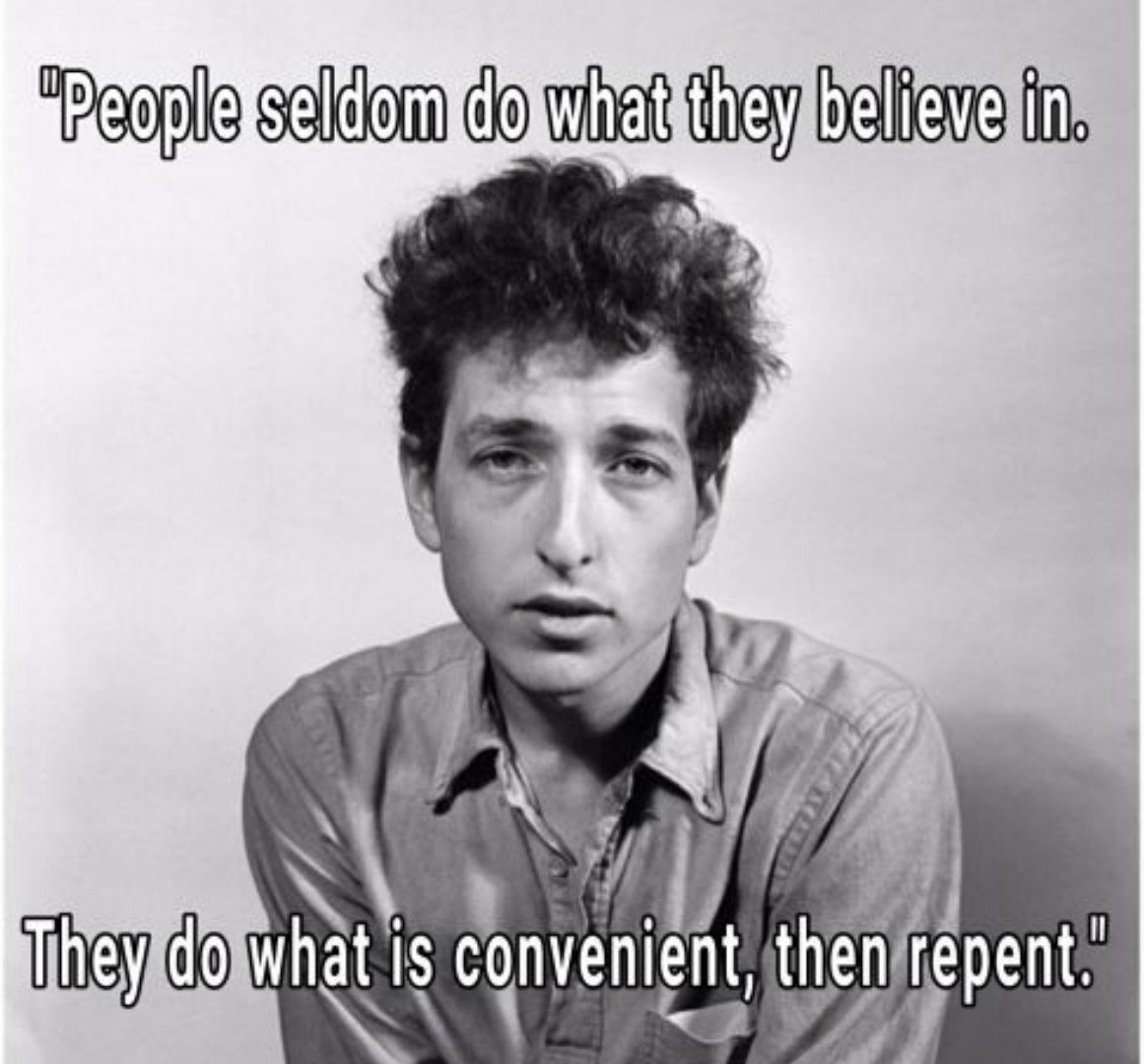
- Many blog posts/videos on intros to TF

- Less comprehensive resources on dealing with multiple regions, brownfield implementations and refactoring

- Goal today is to briefly introduce IAC/Terraform then talk about three main related things:
  - Multi-region project structure
  - Importance of modules in relation
  - Refactoring

Why (I chose) Infra As Code?

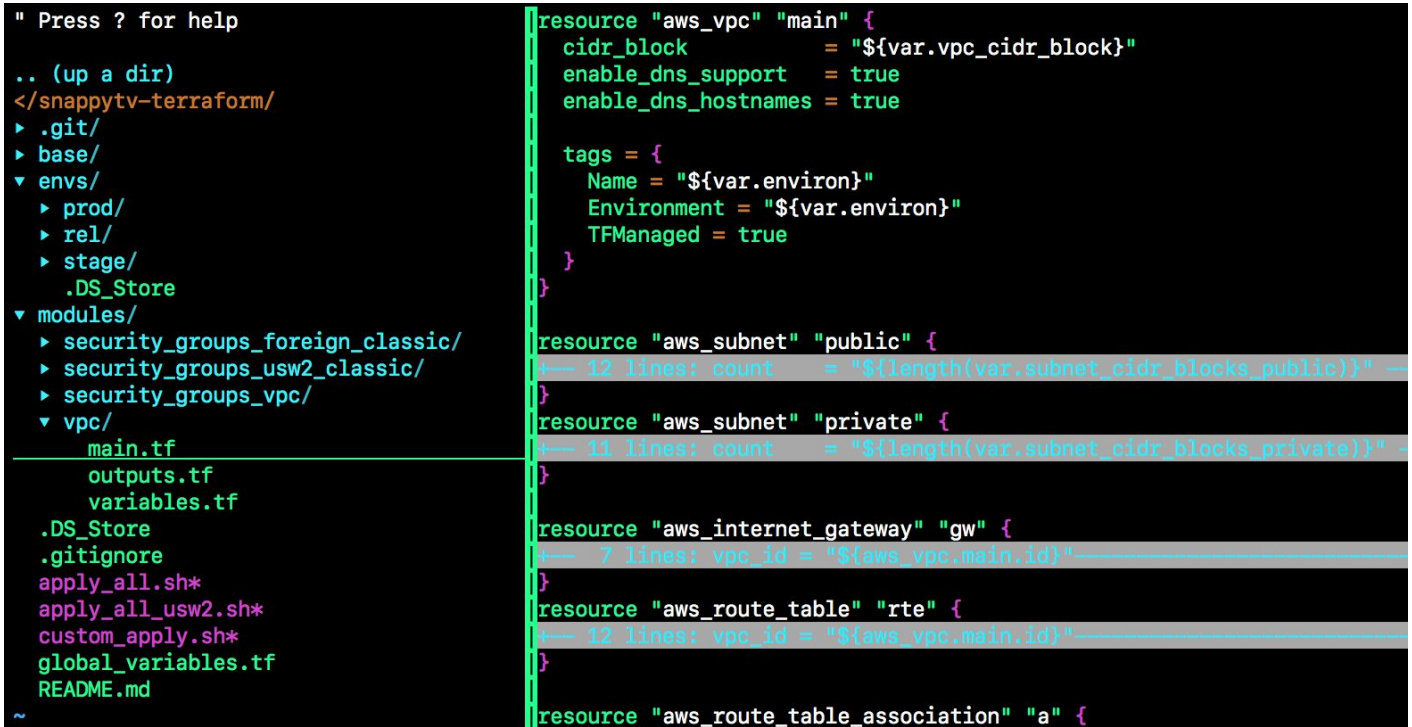
- Time/tediousness
- Human error
- Replicability
- Repeatability
- Stability
- ‘Code as documentation’


"People seldom do what they believe in.
They do what is convenient, then repent."

What is IAC/Terraform?

Framework to specify infrastructure resources via code.

Example: Delare a VPC on AWS



Why Infra As Code on EOL SnappyTV?

- Inherited amazing puppet repo for SnappyTV
- All instance creation, ELB's, security groups, etc have been managed manually by SRE.
- Need to upgrading our dynamic instances to AWS C5 size
- The need to spin up 27 vpc's (9 regions * 3 environs) systematically.
- The need to audit and segregate security groups and rules by environment (i.e. dev/staging/production)
- Seemed like reasonable tool to help with the above

Why Terraform?

- Puppet has some modules which are IAC but… we're still on Puppet 3 and these not avail
  I.e https://github.com/puppetlabs/puppetlabs-aws
- Provider agnostic - TF works with AWS (our case) but also GCP, and probably 100+ others.
  - Investment for company (perhaps), engineer (certainly)
- Seemed like the strongest and most used if IAC frameworks
- Seemed to be good docs and community

Challenges

- Terraform ‘provider’ which is the interface to the external service (i.e. AWS, GCP), in the case of AWS requires a ‘region’ field.

  Ie: provider "aws" { region = "ap-northeast-1" }

- SnappyTV in 9 AWS regions
- 3 environments per region: production, qa, development
- OMG… 27 individual TF state files!


FAIL

State File(s) and WHERE???!

- Local state?, git?, remote state

- Terraform remote state S3 backend (others avail)
  - Locking so only one dev can apply one tf repo at a time
  - Hands-off - once set up TF automagically handles reads and writes

Getting Started With TF and Scary Things

- Installing from TF site and docs straightforward

- Is quick to get to ‘hello world’ with a local statefile
- Most frightening thing is letting TF run the first time against a major brownfield infrastructure
- `terraform destroy` - destroys all managed infra!
- In the end TF can not destroy (at least in AWS) things that are dependent


LOOK
OUT !

Options to Manage Multiple (AWS) Regions/Envs:

Terraform Workspaces

- Allow you to apply same code to different environment or ‘place’
  - Non-starter for SnappyTV

Roll Your Own

- Manually treat each region/env combination as it’s own TF project… all within a tree with some shared resources
  -

Dir/Project Structure

```
.. (up a dir)
/Users/akaruna/workspace/snappytv-terraform/
▸ .git/
▾ base/
  ▸ .terraform/
    global_variables.tf -> /Users/akaruna/workspace/snappytv-terraform/global_variables.tf
    s3.tf
    terraform.tfstate
    terraform.tfstate.backup
    variables.tf
▾ envs/
  ▾ prod/
    ▸ ap-northeast-1/
    ▸ ap-southeast-1/
    ▸ ap-southeast-2/
    ▸ eu-central-1/
    ▸ eu-west-1/
    ▸ sa-east-1/
    ▸ us-east-1/
    ▸ us-west-1/
    ▾ us-west-2/
      ▸ .terraform/
        global_variables.tf -> /Users/akaruna/workspace/snappytv-terraform/global_variables.tf
        security_groups.tf
        security_groups_stv_group_puppet.tf
        security_groups_stv_rds.tf
        variables.tf
        versions.tf
        vpc.tf
      .DS_Store
  ▸ rel/
  ▸ stage/
    .DS_Store
▾ modules/
  ▸ security_groups_foreign_classic/
  ▸ security_groups_usw2_classic/
  ▸ security_groups_vpc/
  ▸ vpc/
  .DS_Store
  .gitignore
  apply_all.sh*
  apply_all_usw2.sh*
  custom_apply.sh*
  global_variables.tf
  README.md
  .global_variables.tf.swp
~
```

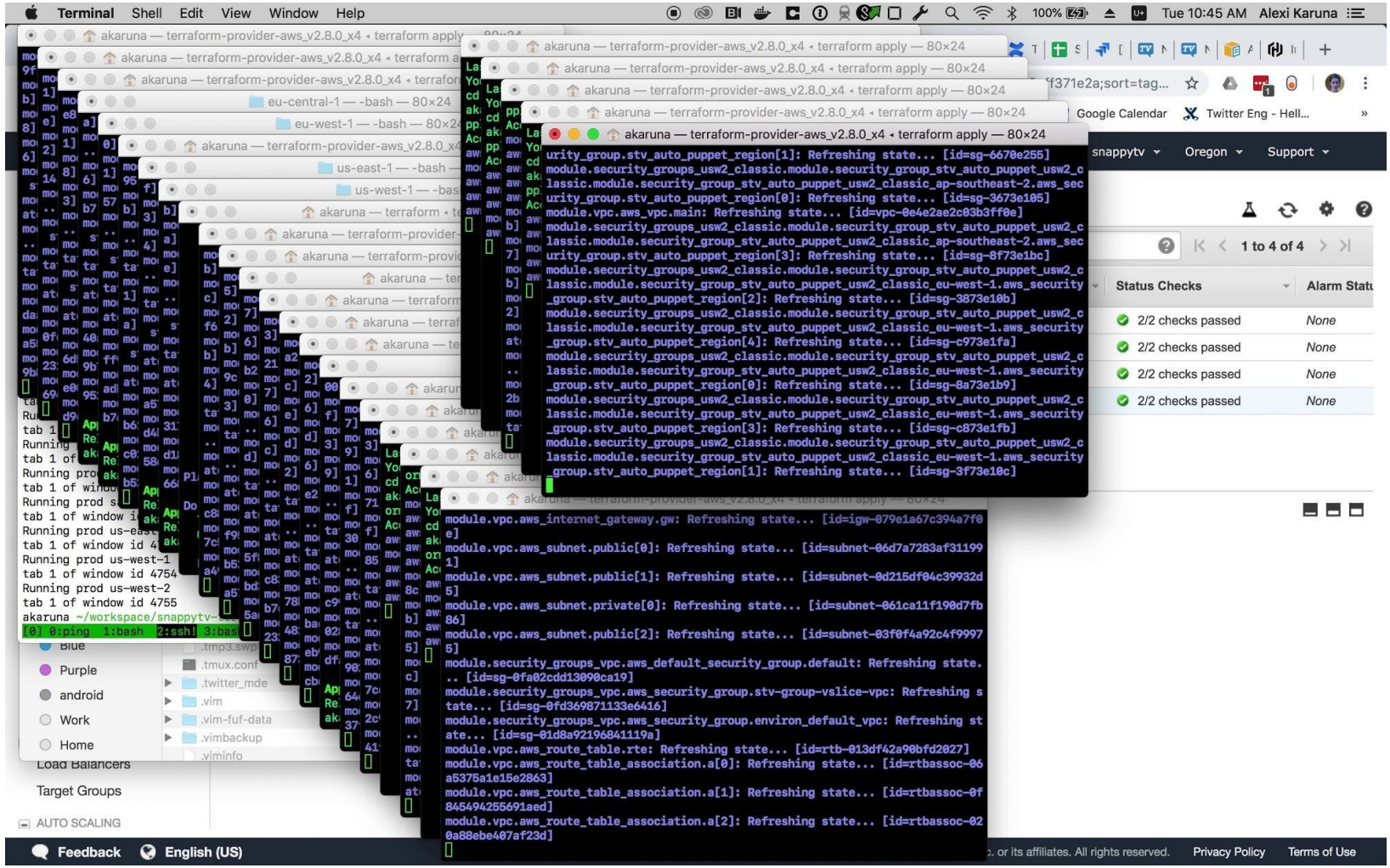
```
▸ base/
▾ envs/
  ▾ prod/
    ▾ ap-northeast-1/
      ▸ .terraform/
        global_variables.tf -> /Users/akaruna/workspace/snappytv
        security_groups.tf
        variables.tf
        versions.tf
        vpc.tf
    ▸ ap-southeast-1/
    ▸ ap-southeast-2/
    ▸ eu-central-1/
    ▸ eu-west-1/
    ▸ sa-east-1/
    ▸ us-east-1/
    ▸ us-west-1/
    ▸ us-west-2/
      .DS_Store
  ▸ rel/
  ▸ stage/
    .DS_Store
▸ modules/
```

Isn't Running TF 27x to Apply Change a PITA???

Initially but… this stuck and served my purposes well...

```
# /bin/bash

# this will open a new terminal window for each env * region and run `terraform apply`
for env in stage rel prod
do
  for region in ap-northeast-1 ap-southeast-1 ap-southeast-2 eu-central-1 eu-west-1 sa-east-1 us-east-1 us-west-1 us-west-2
  do
    echo "Running $env $region"
    osascript -e "tell application \"Terminal\" to do script \"cd ~/workspace/snappytv-terraform/envs/$env/$region && terraform apply\""
  done
done
```

- Reviewing and accepting 27x takes a minute but…
- You *want to* always review your diffs before changes
- Easier to review apples to apples vs one huge diff (even if were possible)
- Compared to manually… human error… digging to find problems… is major time saver and priceless

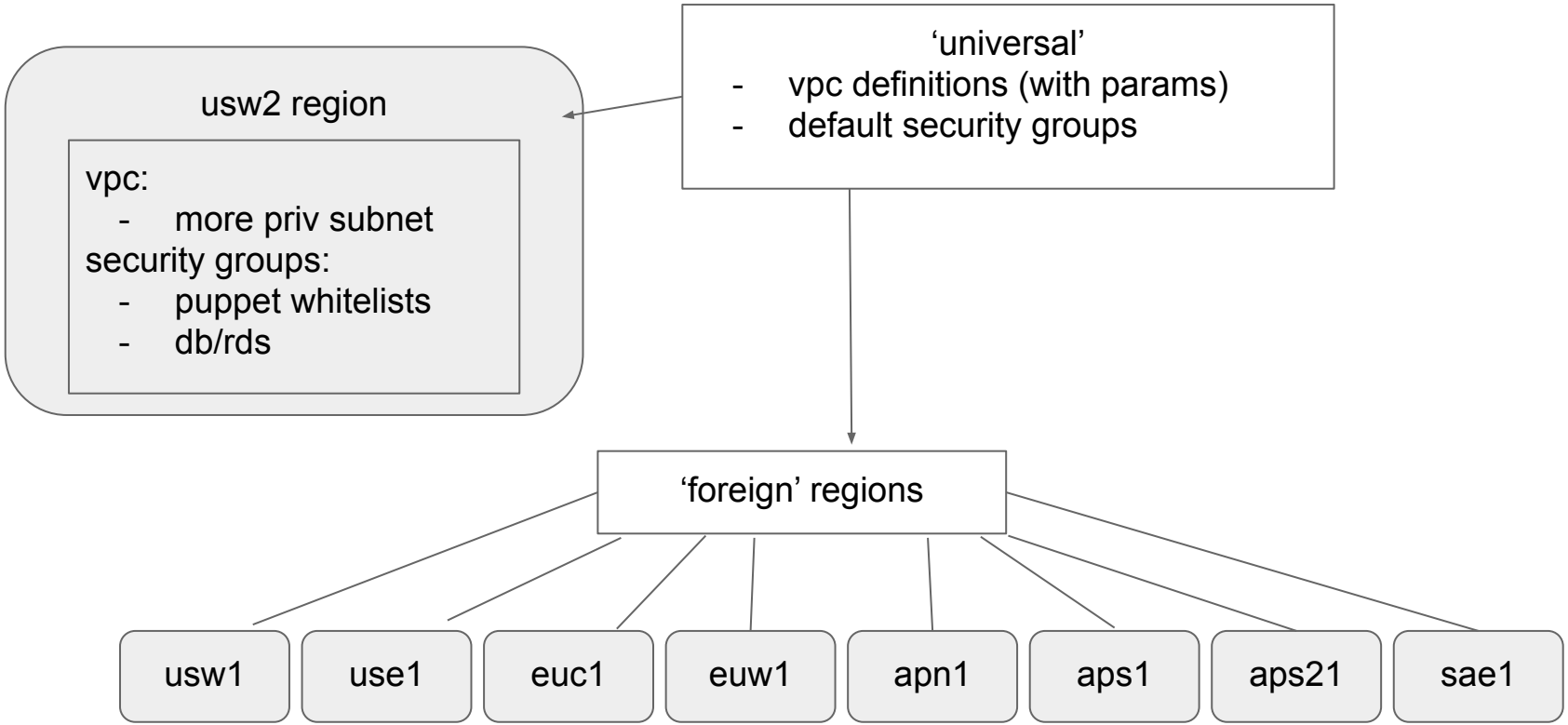

Terminal Shell Edit View Window Help
Tue 10:45 AM Alexi Karuna
akaruna — terraform-provider-aws_v2.8.0_x4 • terraform apply — 80×24
urity_group.stv_auto_puppet_region[1]: Refreshing state... [id=sg-6670e255]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_ap-southeast-2.aws_security_group.stv_auto_puppet_region[0]: Refreshing state... [id=sg-3673e105]
module.vpc.aws_vpc.main: Refreshing state... [id=vpc-0e4e2ae2c03b3ff0e]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_ap-southeast-2.aws_security_group.stv_auto_puppet_region[3]: Refreshing state... [id=sg-8f73e1bc]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_eu-west-1.aws_security_group.stv_auto_puppet_region[2]: Refreshing state... [id=sg-3873e10b]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_eu-west-1.aws_security_group.stv_auto_puppet_region[4]: Refreshing state... [id=sg-c973e1fa]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_eu-west-1.aws_security_group.stv_auto_puppet_region[0]: Refreshing state... [id=sg-8a73e1b9]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_eu-west-1.aws_security_group.stv_auto_puppet_region[3]: Refreshing state... [id=sg-c873e1fb]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_eu-west-1.aws_security_group.stv_auto_puppet_region[1]: Refreshing state... [id=sg-3f73e10c]
module.vpc.aws_internet_gateway.gw: Refreshing state... [id=igw-079e1a67c394a7f0e]
module.vpc.aws_subnet.public[0]: Refreshing state... [id=subnet-06d7a7283af311991]
module.vpc.aws_subnet.public[1]: Refreshing state... [id=subnet-0d215df04c39932d5]
module.vpc.aws_subnet.private[0]: Refreshing state... [id=subnet-061ca11f190d7fb86]
module.vpc.aws_subnet.public[2]: Refreshing state... [id=subnet-03f0f4a92c4f99975]
module.security_groups_vpc.aws_default_security_group.default: Refreshing state... [id=sg-0fa02cdd13090ca19]
module.security_groups_vpc.aws_security_group.stv-group-vslice-vpc: Refreshing state... [id=sg-0fd369871133e6416]
module.security_groups_vpc.aws_security_group.environ_default_vpc: Refreshing state... [id=sg-01d8a92196841119a]
module.vpc.aws_route_table.rte: Refreshing state... [id=rtb-013df42a90bfd2027]
module.vpc.aws_route_table_association.a[0]: Refreshing state... [id=rtbassoc-06a5375a1e15e2863]
module.vpc.aws_route_table_association.a[1]: Refreshing state... [id=rtbassoc-0f845494255691aed]
module.vpc.aws_route_table_association.a[2]: Refreshing state... [id=rtbassoc-020a88ebe407af23d]
Google Calendar
Twitter Eng - Hell...
snappytv
Oregon
Support
1 to 4 of 4
Status Checks
Alarm Statu
2/2 checks passed
None
Blue
Purple
android
Work
Home
Load Balancers
Target Groups
AUTO SCALING
Feedback
English (US)
Privacy Policy
Terms of Use

Structuring Code for Multiple 'Repos/regions'

Challenges…

- Deriving what is actually replicated vs unique
- Getting Terraform code (HCL - Hashicorp Configuration Language) to abstract it *maintainably by others*
- Learning what can/can't I do with HCL?

Repeated vs Unique Settings



Modules: Handling Region/Env Similarities

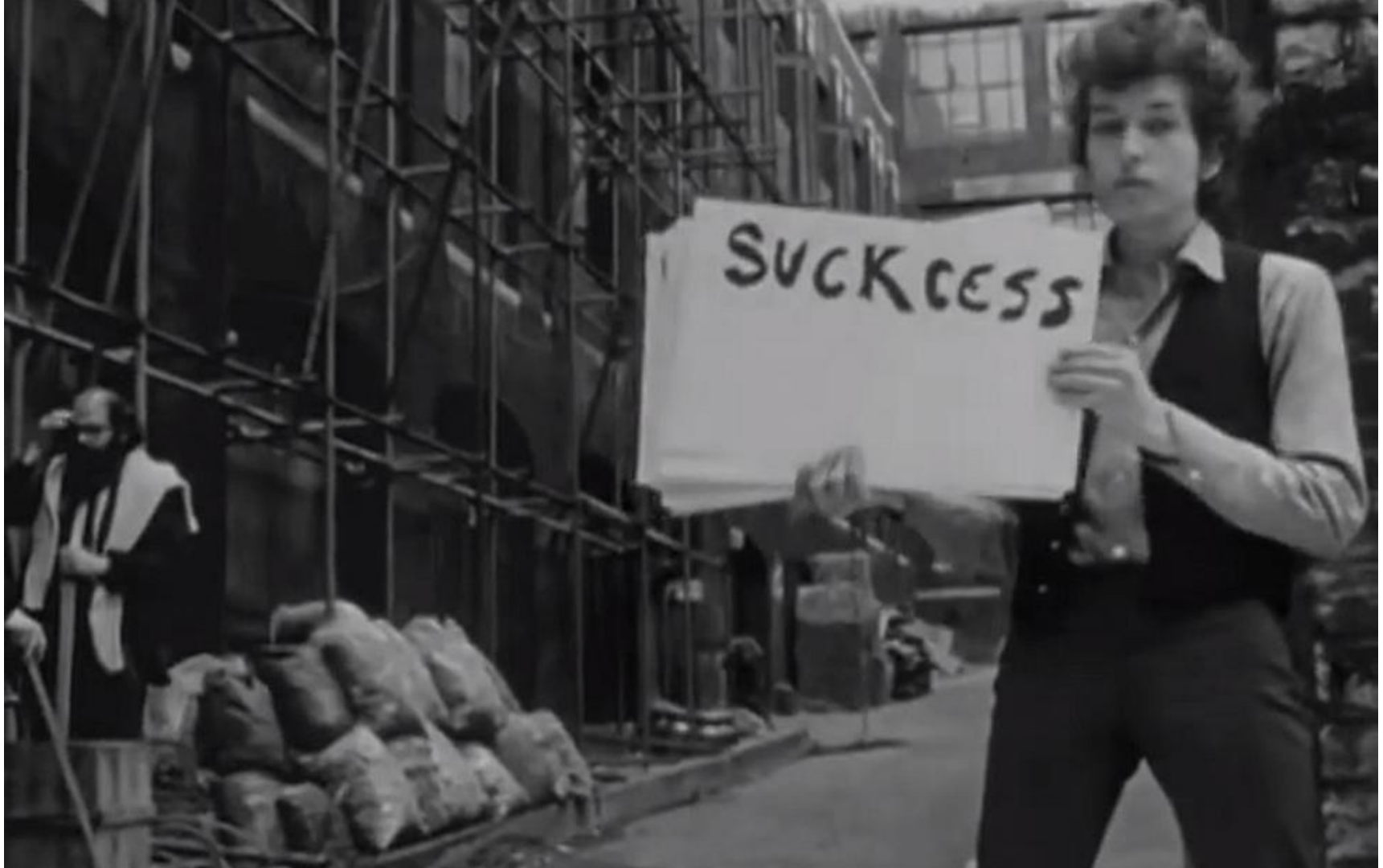
- ‘My us-west-2 region is different than my other regions’
- ‘Each env uses different security groups but the base code is the same’

'Foreign' vs Usw2 VPC Config (ENV)

```
module "vpc" {
  source = "../../../modules/vpc"

  environ = var.environ

  # us-west-1 has az's 1-3 but can not create vpc subnets in us-west-1b
  availability_zones         = ["us-west-1a", "us-west-1c"]
  vpc_cidr_block             = "172.18.48.0/20"
  subnet_cidr_blocks_public  = ["172.18.48.0/22", "172.18.52.0/22", "172.18.56.0/22"]
  subnet_cidr_blocks_private = ["172.18.60.0/22"]
}

~
envs/prod/us-west-1/vpc.tf
module "vpc" {
  source = "../../../modules/vpc"

  environ                    = var.environ
  availability_zones         = ["us-west-2a", "us-west-2b", "us-west-2c"]
  vpc_cidr_block             = "172.18.0.0/19"
  subnet_cidr_blocks_public  = ["172.18.0.0/21", "172.18.8.0/21", "172.18.16.0/21"]
  subnet_cidr_blocks_private = ["172.18.24.0/23", "172.18.26.0/23", "172.18.28.0/23"]
}

~
~
~
envs/prod/us-west-2/vpc.tf
```

VPC Module (with params)

```
resource "aws_vpc" "main" {
+-- 9 lines: cidr_block           = "${var.vpc_cidr_block}"----------
}

resource "aws_subnet" "public" {
  count      = "${length(var.subnet_cidr_blocks_public)}"

  vpc_id     = "${aws_vpc.main.id}"
  cidr_block = "${element(var.subnet_cidr_blocks_public, count.index)}"
  availability_zone = "${element(var.availability_zones, count.index)}"
  map_public_ip_on_launch = true

  tags = {
    Name = "${var.environ} public ${count.index+1}"
    Environment = "${var.environ}"
    TFManaged = true
  }
}
resource "aws_subnet" "private" {
  count      = "${length(var.subnet_cidr_blocks_private)}"

  vpc_id     = "${aws_vpc.main.id}"
  cidr_block = "${element(var.subnet_cidr_blocks_private, count.index)}"
  availability_zone = "${element(var.availability_zones, count.index)}"

  tags = {
    Name = "${var.environ} private ${count.index+1}"
    Environment = "${var.environ}"
    TFManaged = true
  }
```


SUCKCESS

Bringing Existing Security Groups into TF

- Import a group:

  `terraform import aws_security_group.elb_sg sg-983db2a8`

- Must write code for what imported
  - Terraform outputs the code from the import command
- Many imports, just write a bash script

Rearranging/Refactoring in Terraform Code

- You will refactor!
- Don’t be afraid of ‘just starting’ where you are
- After moving code, modules or renaming the output of `plan` or `apply` can be frightening… but…
- Fixing things is actually (generally) easier than you think...
- `terraform mv` is your friend
- Utilize the diff in the output and feed it into `terraform mv`
- Shell script for large moves (find tix)

Refactoring Example

So with current refactor, testing to move one node, this is the delete and recreate for tf plan output:

```
Plan: 54 to add, 0 to change, 45 to destroy. # we are truly creating 9 new sgs, dont want to destroy and recreate the rest!

  +
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_ap-northeast-1.aws_security_group.stv_auto_puppet_region[0]

  -
module.security_groups_usw2_classic.module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic_ap-northeast-1.aws_security_group.stv_auto_puppet_region[0]
```

The 'mv' Command Looks Like...

`terraform state move ['-' item] ['+' item]`

```
akaruna ~/workspace/snappytv-terraform/envs/stage/us-west-2 (master) $ terraform state mv terraform
state mv
module.security_groups_usw2_classic.module.security_groups_usw2_classic.module.security_group_stv_aut
o_puppet_usw2_classic_ap-northeast-1.aws_security_group.stv_auto_puppet_region[0]
module.security_groups_usw2_classic.module.security_group_stv_auto_puppet_usw2_classic.module.securit
y_group_stv_auto_puppet_usw2_classic_ap-northeast-1.aws_security_group.stv_auto_puppet_region[0]
```

*** bash scriptable for major moves!!! ***

Conclusion

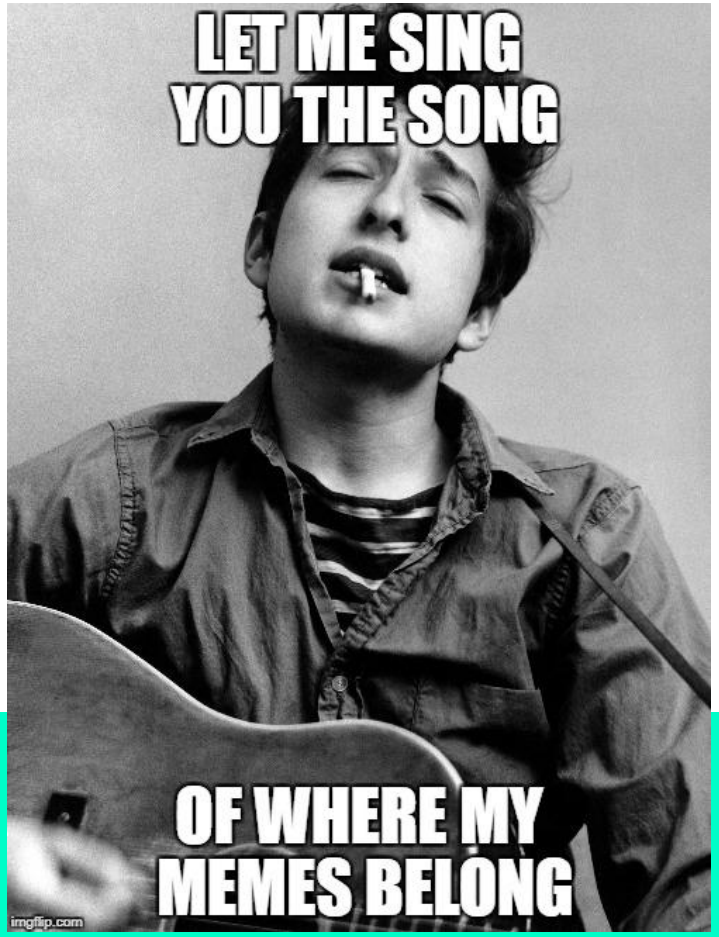
- Able to implement TF with provided docs and groups
- HCL can feel kludgy but is workable
- Thinking in terms of maintainability
- Has served SnappyTV solidly without issue for 6m+
- TF 0.12 out after implementation, good improvements
- I’d personally spend more time on:
  - Getting familiar with HCL 0.12+
  - Making more data driven (i.e. more like hiera in puppet)

Resources

This is a great post which makes clear some of the pitfalls with being a Terraform newbie and also how to structure multiple AWS environments. Much of my implementation pulls from this:

https://charity.wtf/2016/03/30/terraform-vpc-and-why-you-want-a-tfstate-file-per-env/

Thank You!



Presentation based on work performed
on contract at Twitter 2017-1029

alexi@alexikaruna.com