Terraform 101

A talk by Charles

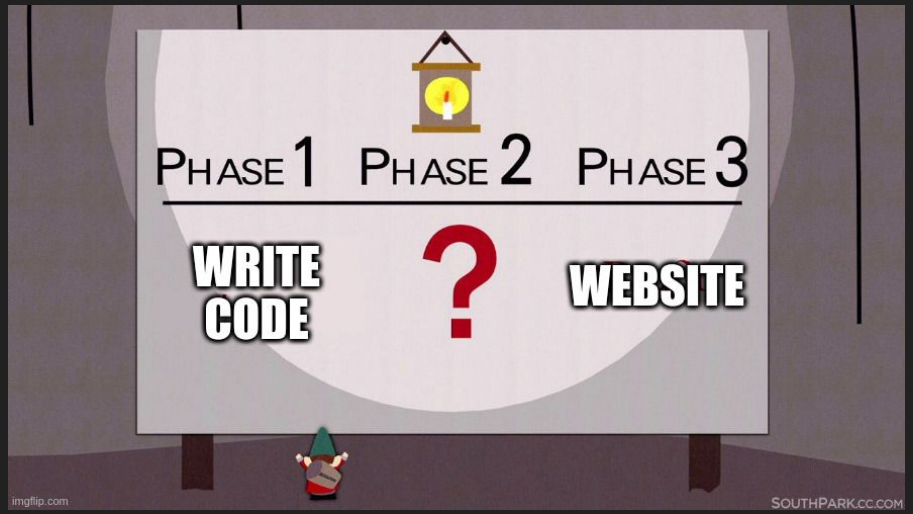
Slides and code: https://github.com/bushong1/presentations/tree/main/terraform-101

Agenda

- What’s Infrastructure?
- Terraform Intro and Anatomy
- Actually running it
- Good Stuff
- Helpful Tips + Brain Dump

Infrastructure?

Infrastructure is...?



- **Everything ?** could be “Infrastructure”
  - Automated tests
  - Static code to bucket
  - Docker image build
  - Container Deploy
  - Server Build
  - Database
  - Caching
  - Network
  - Security
  - ……

Let’s make a server on AWS

How do… computer?

- To create a server on AWS
  a. Know all the answers to the values on the right
  b. Pray you didn't get it wrong
  c. Get hacked because you did it wrong

```
"AdditionalInfo" : String,
"Affinity" : String,
"AvailabilityZone" : String,
"BlockDeviceMappings" : [ BlockDeviceMapping, ... ],
"CpuOptions" : CpuOptions,
"CreditSpecification" : CreditSpecification,
"DisableApiTermination" : Boolean,
"EbsOptimized" : Boolean,
"ElasticGpuSpecifications" : [ ElasticGpuSpecification, ... ],
"ElasticInferenceAccelerators" : [ ElasticInferenceAccelerator, ... ],
"EnclaveOptions" : EnclaveOptions,
"HibernationOptions" : HibernationOptions,
"HostId" : String,
"HostResourceGroupArn" : String,
"IamInstanceProfile" : String,
"ImageId" : String,
"InstanceInitiatedShutdownBehavior" : String,
"InstanceType" : String,
"Ipv6AddressCount" : Integer,
"Ipv6Addresses" : [ InstanceIpv6Address, ... ],
"KernelId" : String,
"KeyName" : String,
"LaunchTemplate" : LaunchTemplateSpecification,
"LicenseSpecifications" : [ LicenseSpecification, ... ],
"Monitoring" : Boolean,
"NetworkInterfaces" : [ NetworkInterface, ... ],
"PlacementGroupName" : String,
"PrivateIpAddress" : String,
"RamdiskId" : String,
"SecurityGroupIds" : [ String, ... ],
"SecurityGroups" : [ String, ... ],
"SourceDestCheck" : Boolean,
"SsmAssociations" : [ SsmAssociation, ... ],
"SubnetId" : String,
"Tags" : [ Tag, ... ],
"Tenancy" : String,
"UserData" : String,
"Volumes" : [ Volume, ... ]
```

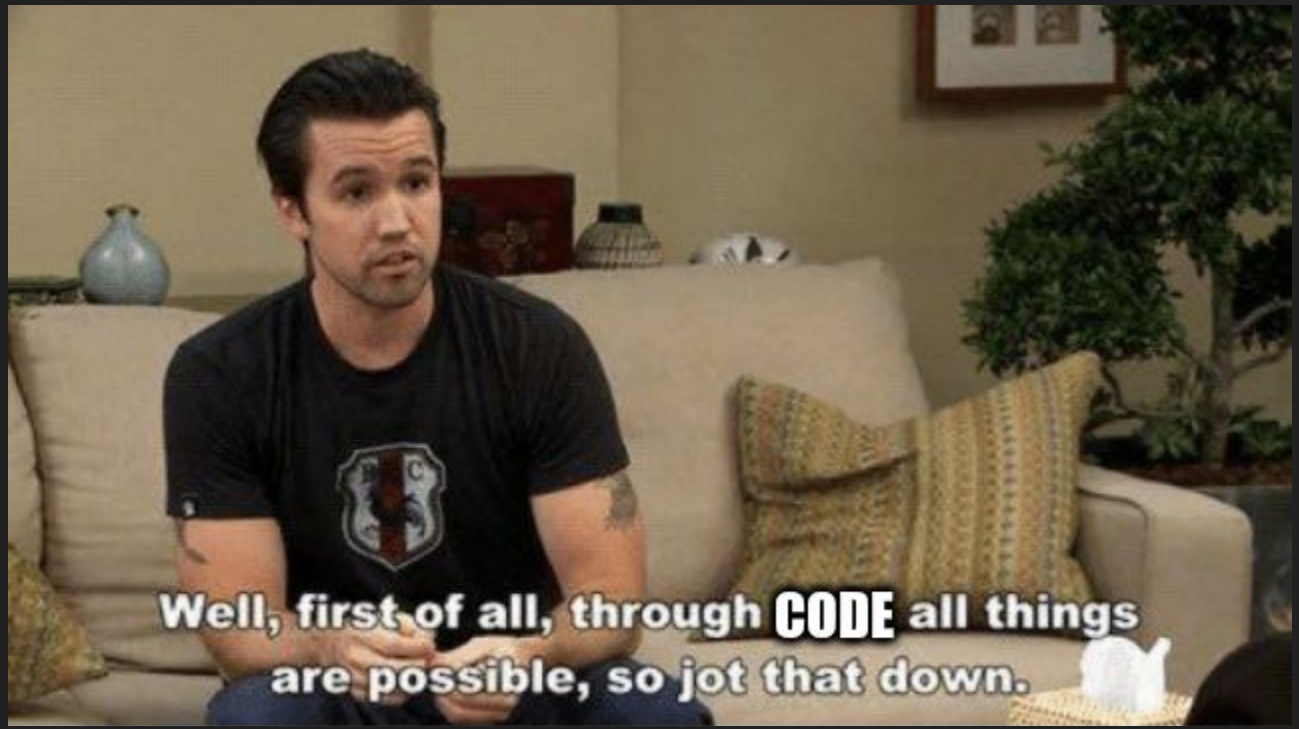
But maybe you suffer through it

...once

What if I wanna do it again?

- That’s hard, and I forget stuff…
  - Write documentation
- What if it changes?
  - Update the docs
- You won’t.
  - This is hard?

Enter: AWS API



But there’s a lot of API...

```
accessanalyzer
acm
acm-pca
alexaforbusiness
amp
amplify
amplifybackend
apigateway
apigatewaymanagementapi
apigatewayv2
appconfig
appflow
appintegrations
application-autoscaling
application-insights
appmesh
appstream
appsync
athena
auditmanager
autoscaling
autoscaling-plans
backup
batch
braket
budgets
ce
chime
cli-dev
cloud9
clouddirectory
cloudformation
cloudfront
cloudhsm
cloudhsmv2
cloudsearch
cloudsearchdomain
cloudtrail
cloudwatch
codeartifact
codebuild
codecommit
codeguruprofiler
codeguru-reviewer
codepipeline
codestar
codestar-connections
codestar-notifications
cognito-identity
cognito-idp
cognito-sync
comprehend
comprehendmedical
compute-optimizer
configservice
configure
connect
connect-contact-lens
connectparticipant
cur
customer-profiles
databrew
dataexchange
datapipeline
datasync
dax
ddb
deploy
detective
devicefarm
devops-guru
directconnect
discovery
dlm
dms
docdb
ds
dynamodb
dynamodbstreams
ebs
ec2
ec2-instance-connect
ecr
ecr-public
ecs
efs
eks
elasticache
elasticbeanstalk
elastic-inference
elastictranscoder
elb
elbv2
emr
emr-containers
es
events
firehose
fms
forecast
forecastquery
frauddetector
fsx
gamelift
glacier
globalaccelerator
glue
greengrass
greengrassv2
groundstation
guardduty
health
healthlake
help
history
honeycode
iam
identitystore
imagebuilder
importexport
inspector
iot
iot1click-devices
iot1click-projects
iotanalytics
iot-data
iotdeviceadvisor
iotevents
iotevents-data
iotfleethub
iot-jobs-data
iotsecuretunneling
iotsitewise
iotthingsgraph
iotwireless
ivs
kafka
kendra
kinesis
kinesisanalytics
kinesisanalyticsv2
kinesisvideo
kinesis-video-archived-media
kinesis-video-media
kinesis-video-signaling
kms
lakeformation
lambda
lex-models
lex-runtime
license-manager
lightsail
location
logs
lookoutvision
machinelearning
macie
macie2
managedblockchain
marketplace-catalog
marketplacecommerceanalytics
marketplace-entitlement
mediaconnect
mediaconvert
medialive
mediapackage
mediapackage-vod
mediastore
mediastore-data
mediatailor
meteringmarketplace
mgh
migrationhub-config
mobile
mq
mturk
mwaa
neptune
network-firewall
networkmanager
opsworks
opsworks-cm
opsworkscm
organizations
outposts
personalize
personalize-events
personalize-runtime
pi
pinpoint
pinpoint-email
pinpoint-sms-voice
polly
pricing
qldb
qldb-session
quicksight
ram
rds
rds-data
redshift
redshift-data
rekognition
resource-groups
resourcegroupstaggingapi
robomaker
route53
route53domains
route53resolver
s3
s3api
s3control
s3outposts
sagemaker
sagemaker-a2i-runtime
sagemaker-edge
sagemaker-featurestore-runtime
sagemaker-runtime
savingsplans
schemas
sdb
secretsmanager
securityhub
serverlessrepo
servicecatalog
servicecatalog-appregistry
servicediscovery
service-quotas
ses
sesv2
shield
signer
sms
snowball
sns
sqs
ssm
sso
sso-admin
sso-oidc
stepfunctions
storagegateway
sts
support
swf
synthetics
textract
timestream-query
timestream-write
transcribe
transfer
translate
waf
waf-regional
wafv2
wellarchitected
workdocs
worklink
workmail
workmailmessageflow
workspaces
xray
```

Creating your infrastructure could be done...

- by hand in the GUI
  - Easy to start, hard to maintain
- by hand with the CLI
  - Easy to forget how
- with a homemade script using the API
  - Hard to link stuff together, track state

- **using a tool purpose made to solve these problems**


Terraform

Terraform

- API Wrapper
  - It does what AWS lets it
- State Manager
  - “What do I have? What do I need? What changed?”
- Modular, Declarative Language
  - YAML-esque
- Fast
  - ...-er than alternatives

Let’s get to the code

Anatomy of Terraform

```
locals {
  app_name = "hello-world"
}
data "aws_ami" "ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu-20.04-<...>-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  owners = ["099720109477"] # Canonical
}

resource "aws_instance" "my_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = local.app_name
  }
}
```

Anatomy of Terraform

- **Resource**
  - Creates a “Thing”

```
locals {
  app_name = "hello-world"
}
data "aws_ami" "ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu-20.04-<...>-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  owners = ["099720109477"] # Canonical
}

resource "aws_instance" "my_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = local.app_name
  }
}
```

Anatomy of Terraform

- **Resource**
  - Creates a “Thing”
- **Data Source**
  - Loads a “Thing”

```
locals {
  app_name = "hello-world"
}
data "aws_ami" "ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu-20.04-<...>-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  owners = ["099720109477"] # Canonical
}

resource "aws_instance" "my_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = local.app_name
  }
}
```

Anatomy of Terraform

- **Resource**
  - Creates a “Thing”
- **Data Source**
  - Loads a “Thing”
- **Local**
  - A private variable

```
locals {
  app_name = "hello-world"
}
data "aws_ami" "ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu-20.04-<...>-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  owners = ["099720109477"] # Canonical
}

resource "aws_instance" "my_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = local.app_name
  }
}
```

Now what?

- <hand-wave AWS creds>
- `terraform plan`
- List of everything to add, change, or destroy

```
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:

  # aws_instance.my_server will be created
  + resource "aws_instance" "my_server" {
      + ami                          = "ami-042e8287309f5df03"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t3.micro"
      + ipv6_address_count           = (known after apply)
    ...
          + volume_type           = (known after apply)
        }
    }

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

THIS IS HUGE

And then...

- `terraform apply`
- Does a plan again
- “yes” to apply the changes

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

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.my_server: Creating...
aws_instance.my_server: Still creating... [10s elapsed]
aws_instance.my_server: Creation complete after 17s [id=i-0840e1e43425b2504]
```

And then...

- `terraform apply`
- Does a plan again
- “yes” to apply the changes

And then...

- `terraform apply`
- Does a plan again
- But nothing changes

```
$ terraform apply
aws_instance.my_server: Refreshing state... [id=i-0840e1e43425b2504]

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

And then...

- `terraform apply`
- Does a plan again
- “yes” to apply the changes

And then...

- `terraform apply`
- Does a plan again
- But nothing changes

And then...

- `terraform destroy`
- Does a plan again, deleting stuff
- “yes” to destroy everything

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.my_server: Destroying... [id=i-0840e1e43425b2504]
aws_instance.my_server: Still destroying... [id=i-0840e1e43425b2504, 10s elapsed]
```

Get to the good stuff!

Any number of Resources

- `count` - Create multiple of a thing
- Within that resource, `count.index`

```
resource "aws_instance" "my_server" {
  count = 5
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = "${local.app_name}-${count.index}"
  }
}
```

Array indexing

- Reference another array using index

```
locals {
  server_names = [
    "dog",
    "cat",
    "lizard",
  ]
}
resource "aws_instance" "my_server_named_better" {
  count = 3
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = "my-server-${local.server_names[count.index]}"
  }
}
```

Local Modules

- All files in one directory are read together
- Reference files in other directories with modules
- Repeatable

```
locals {
  server_config = {
    dog : { instance_type = "m3.large" }
    cat : { instance_type = "m3.medium" }
    lizard : { instance_type = "m3.small" }
  }
}

module "ami" {
  source = "./ami-lookup"
}

module "my-servers" {
  source   = "./my-server"
  for_each = local.server_config

  server_name          = "server-${each.key}"
  server_instance_type = each.value.instance_type
  ami_id               = module.ami.ubuntu
}
```

Local Modules - New Resources

- Variable
  - Define in your module
  - Assign during module creation
- Output
  - Reference with module name
  - Output from your module

```
variable "ami_id" { type = string }
variable "server_name" { type = string }
variable "server_instance_type" { type = string }
```

```
module "my-servers" {
  source    = "./my-server"
  for_each = local.server_config

  ami_id                = module.ami.ubuntu
  server_name           = "server-${each.key}"
  server_instance_type = each.value.instance_type
}
```

```
output "ubuntu" {
  value = data.aws_ami.ubuntu.id
}
```

Public Modules

- Terraform Registry
- Some pseudo-officially supported
- YMMV

- Good ones:
  - VPC: https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/
    - Fully featured VPC
    - Handles lots of pitfalls
  - USSBA: https://registry.terraform.io/search/modules?q=ussba
    - Managed by us!

Absolutely no way I have time for this,
but here it is just in case
or for reference or whatever

Statefile Management

- Statefiles contain the metadata for your resources
- S3 Backend
- Locking?
- Secrets (RDS!!!!!)
- Imports?
    - Create a resource outside terraform, then import into existing state

Workspaces

- Parallel sets of resources using the same terraform code
- Use cases:
  - Prod/test/dev
  - Multiple dev environments (per dev or app)
- Each has its own state file
- Base defaults in terraform variables file
- Workspace specific variable values in <workspace>.tfvars file
- ${terraform.workspace} is a handy way to propagate the env name into tags and resource names

Security

- Any secrets that find their way into the statefile are in the clear
- Tfvars may be a good place to list these, but make sure those don’t wind up in Github or a public S3 bucket
- Terraform apply can read AWS creds from environment variables, so you don’t have to put your IAM creds in tfvars, regardless of what tutorials say
- Provide non-repudiation by forcing users to use their own IAM creds and assume a “Terraformer” role

Multi-Account

Considerations with two or more AWS accounts:

- Scope of named resources
  - Globally Unique? (S3 Bucket)
  - Unique per Account? (Security Group Name)
  - Just a tag? (EC2 Instance “Name”)
- Accidentally running on the wrong account
  - “provider” has an allowed_account_ids field

Monolithic vs Multi-state

- Monolith
  - Everything in one Terraform state
  - Good
    - One stop shop
    - Only one place to apply
    - Dependencies easily available
  - Bad
    - Long-running apply
    - Spaghetti code
    - Conflicts resolution
    - Toe Stepping
- Multi-state
  - One state per repeated context
  - Good
    - Easier to read sections
    - Faster individual apply
  - Bad
    - Mutli-state x multi-workspace = confusing
    - “Where did I put that thing?”
    - Dependency Injection

Neat Commands

- terraform <...>
  - **fmt -recursive**
    - Format all your code
  - **validate**
    - Confirm code will (mostly) run properly
  - **show**
    - Eyeball the state
  - **taint <resource>**
    - Mark a resource for recreation
  - **console**
    - Bring up a CLI to play with resources; not perfect