Infrastructure as Code with Terraform
Agenda

- What is IaC?
  - Benefits
- What is Terraform?
- Why do we need Terraform?
- How do we use Terraform?
  - Providers
  - Resources
  - Variables (inputs)
  - Outputs
  - Data Structures
  - Modules
  - Conditionals
  - Iterations
  - Terraform State
- How do I get started?
What Is Infrastructure as Code?

The ability to describe/define your infrastructure and application in source code
Benefits of IaC

- Software methodologies, tools and practices
  - Code reviews
  - Automated testing
  - linting
Automation
Version Control
A young boy desperately needs an operation--

One that only Dr. Blake can perform!
Also....

Correlation

Visibility

Traceability
What is Terraform?

- Infrastructure as code management tool that uses a declarative language to build infrastructure
- Written in Go
- terraform.io
Imperative vs Declarative

Imperative (How)
- Buy chocolate cake mix
- Open cake mix box
- Pour cake mix in bowl
- Add ingredients
- Stir
- Pour in pan
- Preheat oven to 350
- Place pan in oven
- Bake at 350
- etc

Declarative (What)
I need a chocolate cake big enough to feed 20 people
Why do we need Terraform?
Infrastructure is hard!
Idempotent
Cloud agnostic
How do we use Terraform?

Installs as a single binary (https://www.terraform.io/downloads.html)

- MacOS
- Linux
- Windows
- FreeBSD
- Solaris
Usage

- Terraform init
  - initializes terraform directory
  - pulls in plugins for specified provider
  - Pulls in modules
- Terraform fmt
  - Rewrites terraform config files to canonical format and style
- Terraform validate
  - Runs checks that verify whether a configuration is syntactically valid and internally consistent
- terraform plan
  - A preview of what changes will be made
- Terraform apply
  - Applies changes
- Terraform destroy
  - Destroys all changes
- Terraform show
  - Shows resources from state file
Providers

Way to interact with service providers (which API to use)
# The default provider configuration

```hcl
provider "aws" {
  region = "us-east-1"
}
```
Resources

Bread and butter that represents the infrastructure components you want to manage

- Virtual machines
- Load balancers
- Firewall rules
- Virtual Networks
- Databases
- Message queues
- Data warehouses
- …etc
resource "aws_instance" "web" {
  ami = "${data.aws_ami.ubuntu.id}"
  instance_type = "t2.micro"

  tags = {
    Name = "HelloWorld"
  }
}

resource "aws_elb" "bar" {
  name = "foobar-terraform-elb"

  availability_zones = ["us-west-2a", "us-west-2b", "us-west-2c"]

  instances = ["${aws_instance.web.id}"

  tags = {
    Name = "foobar-terraform-elb"
  }
}
Variables

● Environment
  ○ Begins with TF_VAR_
    ■ export TF_VAR_somevariable=somevalue

● Inputs

● Outputs

● Data Structures
  ○ Strings
  ○ Arrays
  ○ Maps
Variable example code

```hcl
variable "image_id" {
  type = string
  default = "ami-abc123"
}

resource "aws_instance" "web" {
  instance_type = "t2.micro"
  ami          = var.image_id
}
```
Conditionals

If statements

If/else

Boolean operations
Conditional example

resource "aws_instance" "vpn" {
  count = "${var.something ? 1 : 0}"  
  CONDITION ? TRUEVAL : FALSEVAL
}

Iteration

resource "aws_iam_user" "example" {

  count = length(var.user_names)

  name  = var.user_names[count.index]
}

variable "user_names" {

  description = "names of users"

  type = "list"
Modules

Reusable code

Collection of resources

Conforms to D-R-Y (don’t repeat yourself) methodology
Module example

SQL module (tf_azurem_sql)

```hcl
resource "azurerm_sql_server" "test" {
  name = "${var.sql_server_name}"
  resource_group_name = "${var.resource_group_name}"
  location = "${var.resource_group_location}"
}

resource "azurerm_sql_database" "test" {
  name = "${var.sql_database_name}"
  resource_group_name = "${var.resource_group_name}"
  location = "${var.resource_group_location}"
  server_name = "${var.my_sql_server_name}"
}
```

SQL module instantiation

```hcl
module "sql_server_database" {
  source = "git://https://myrepo/sql/_git/tf_azurem_sql?ref=1.7"
  resource_group_name = "my_resource_group"
  resource_group_location = "useast1"
  sql_server_name = "my_sql_server_name"
  sql_database_name = "my-sql-database"
}
```
Functions

- String manipulation
- Numeric
- Collection
- Date and time
- …more

Ex.

> max(12, 54, 3) 54

> join("", ",", ["foo", "bar", "baz"]) foo, bar, baz

> timestamp() 2018-05-13T07:44:12Z

> cidrhost("10.12.127.0/20", 16) 10.12.112.16

> concat(["a", ""], ["b", "c"]) ["a", "","b","c",]
State

- Terraform stores state about your managed infrastructure and configuration.
- Used by Terraform to map
  - real world resources to your configuration
  - keep track of metadata
  - improve performance for large infrastructures.
- This state is stored by default in a local file named "terraform.tfstate"
- can also be stored remotely (works better in a team environment)
- uses local state to create plans and make changes to your infrastructure. Prior to any operation, Terraform does a refresh to update the state with the real infrastructure.
Current State
Desired State
How do I get started?

Understand the resources of the provider (very important)

Get a free tier account with a provider (GCP, AWS, Azure)

Download the binary

Read the docs

Use it
Recommendations

Use terraform plan output

Use remote state

Backup your statefile

Review plans (two sets of eyes)

Use secret management - don’t store secrets directly in tf config files or env variables

Plan structure
Resources

- Terraform.io
- The Terraform Book - James Turnbull
- Terraform Up and Running - Yevginy Brikman
- Me
  - greg.greenlee@insight.com
  - @BIT_greggreenle