WELCOME!

While we are waiting, please follow the steps below to sign in

Failure to do so will result in a "No Show". Starting 2 hours before the class you will be able to self-check-in.(Check-in will close 24 hours after the start time of class)

- Go to the calendar invite for the class that is on your Google calendar.
- Click the event link located in the calendar.
- You will see the class in Learn ++ where you previously enrolled.
- Click the "Check-In" box.
- Congrats you are now checked in and will be marked as Attended!

An Overview

Develop Intelligence

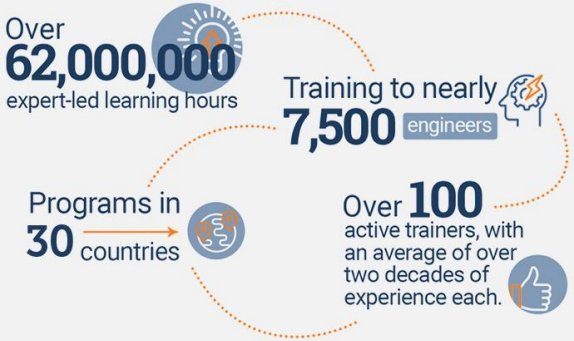
Our purpose…

We help organizations learn and adopt new technologies.

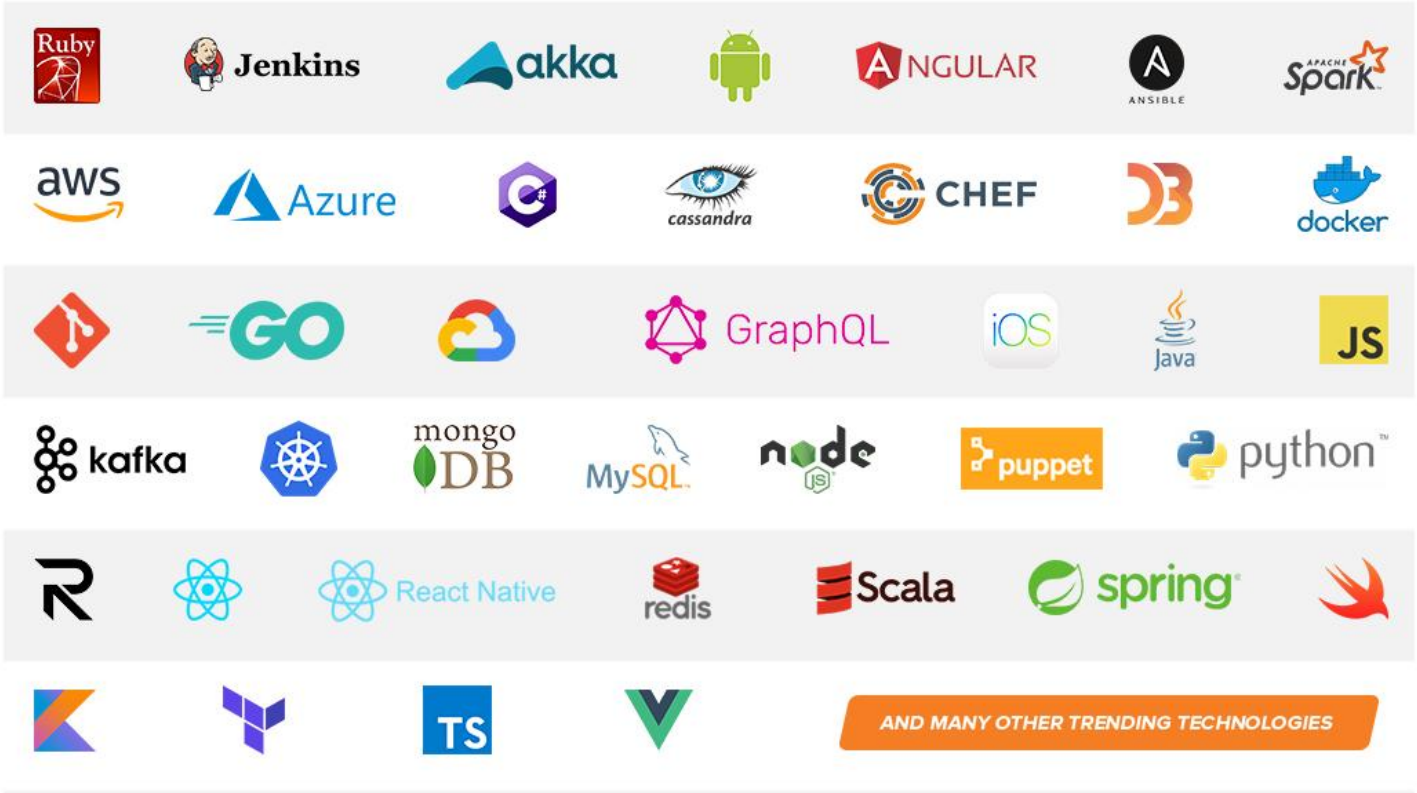
…Impacts you daily.

When you talk on the phone, watch a movie, connect with friends on social media, drive a car, fly on a plane, pay with a credit card, shop online, and order a latte with your mobile app, you are interacting with technology developed by one of our customers.

In 2018 alone...



Technologies we cover



Our Practitioners

Introduction to Terraform

Develop Intelligence

Schedule

Develop Intelligence

- Course runs 9:00 - 3:30 Pacific time
  - Lunch break from 12-1 Pacific time
  - 10-15 min. breaks as needed

Introductions: First You, Then Me

Develop Intelligence

- Name
- Job Role
- Experience with Terraform: 0-5
- Your best tip for surviving the lockdown

Note About Virtual Trainings

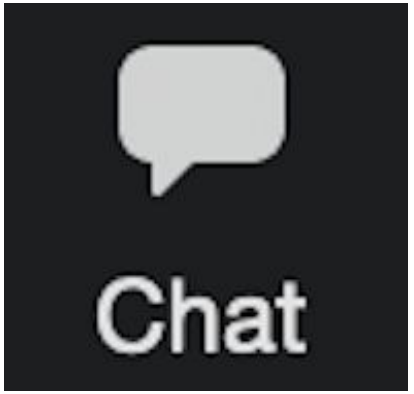
What we want

...what we've got

Virtual Training Expectations for You

Arrive on time / return on time

Mute unless speaking



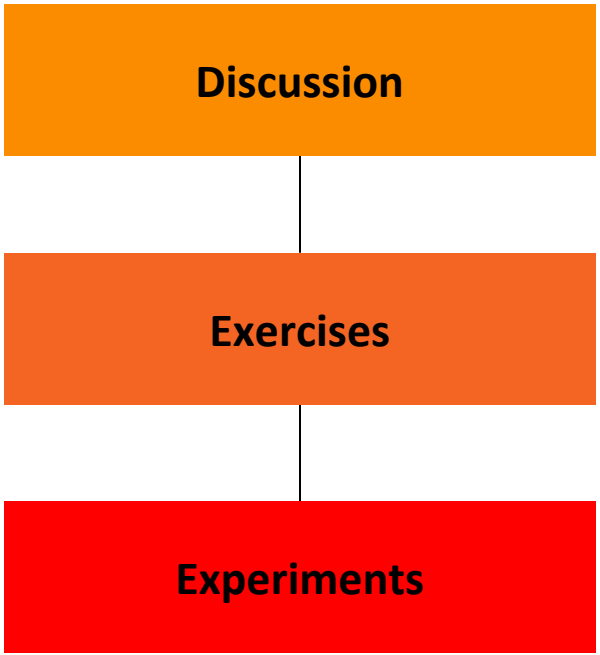
Use chat or ask questions verbally

Virtual Training Expectations for Me

I pledge to:

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so
- Use an on-screen timer for breaks so you know when to be back

Class Structure



Discussion

Develop Intelligence

- Lecture-style presentation
- Questions and thoughts are welcome at any time
- Tangents are welcome as well–this part of the course should be interactive and stimulating

Exercises

Develop Intelligence

- Groups of 2-3 recommended
  - i.e., you are encouraged to work through things together, but may work individually if you wish
- I'm here to help if you're stuck or have questions

Experiments

Develop Intelligence

- an extension of exercises
  - we can take a closer look together as to how this applies to real scenarios in your current work
- choose your own adventure or I can give some ideas to explore
- uncovering Terraform “gotchas” and limitations
- Anything spark ideas for you along the way that you’d like to see in action?
- How can I help you get what you need to out of this course?

An Important Note

- Terraform released version 0.12 in May 2019
  - some important foundational things changed with this release
- we'll be covering some syntax specific to 0.12 in this course
- in general, the syntax and approaches throughout the course will make use of 0.12 capabilities.

Read more about 0.12 from Hashicorp:

https://github.com/hashicorp/terraform/releases/tag/v0.12.0

https://www.hashicorp.com/blog/announcing-terraform-0-12

Other Good Sources for Learning

Develop Intelligence

- Terraform has been around a while now, and is pretty popular, so there are plenty of other sources for learning
  - Hashicorp's new learning portal: https://learn.hashicorp.com/terraform
  - Terraform repository for seeing the state of open issues: https://github.com/hashicorp/terraform/issues
  - Official Terraform docs: https://www.terraform.io/docs/index.html

What is Terraform?

Develop Intelligence

- “infrastructure as code”
- *declarative* domain-specific language
  - what is declarative?
- used to describe *idempotent* resource configurations, typically in cloud infrastructure
- according to Hashicorp:
  - *Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned*

What is Terraform? (cont'd)

- open source CLI tool for *infrastructure automation*
- utilizes plugin architecture
  - extensible to any environment, tool, or framework and works primarily by making API calls to those environments, tools, or frameworks
- detects implicit dependencies between resources and automatically creates a dependency graph
- builds in dependency order and automatically performs activities in parallel where possible
  - ...sequentially for dependent resources

Why Use Terraform?

Develop Intelligence

- readable
- repeatable
- certainty (i.e., no confusion about what will happen)
- standardized environments
- provision quickly
- disaster recovery

What Does Terraform (HCL) Look Like?

Develop Intelligence

```
resource "aws_instance" "web" {
  ami           = "ami-19827362728"
  instance_type = "t2.micro"

  tags = {
    Name = "my-first-instance"
  }
}
```

Exercise Prep: Let's Get Set Up

Develop Intelligence

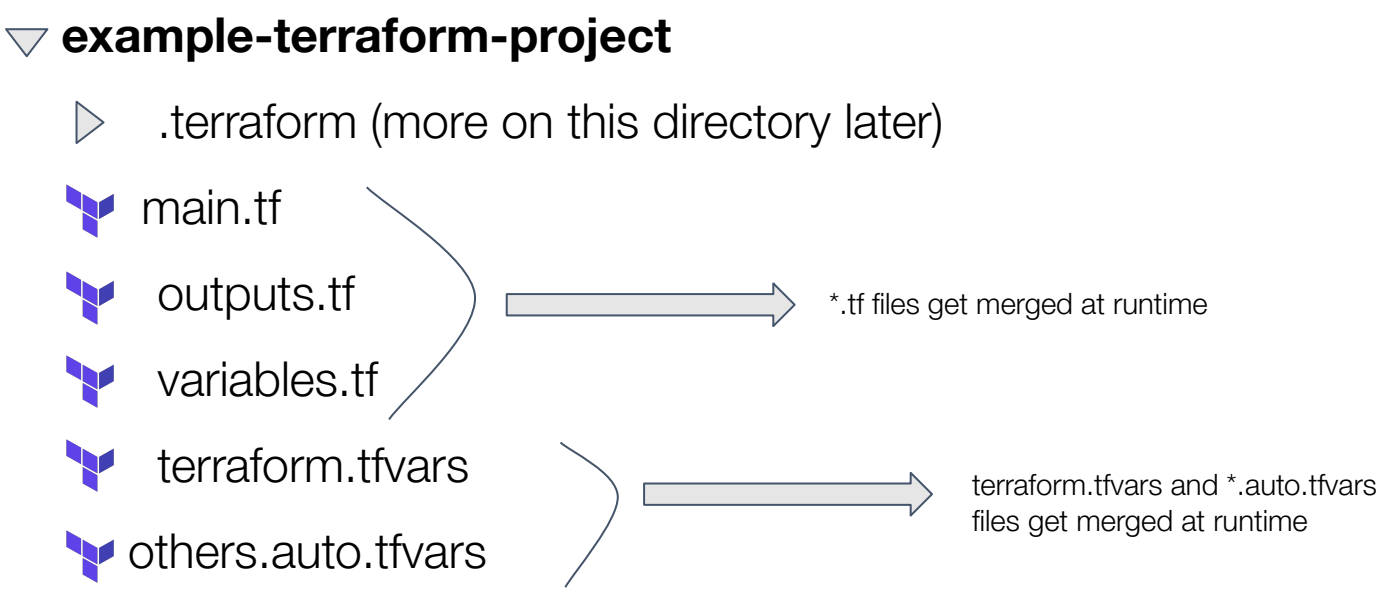
1. We will create an isolated (and consistent) development and execution environment where you can run Terraform

2. You have been emailed your student alias, access key, secret key, and console password (check your Junk folder if you don't have an email from me, and I can also send you the info via Zoom Chat)

3. Let's get these working in your development environment–this will allow you to create things and verify they exist in AWS

4. Access to your instructional repository: Course Setup

Format of a Terraform Project



Hashicorp Configuration Language (HCL)

Develop Intelligence

- The goal of HCL is to build a structured configuration language that is both human and machine friendly for use with command-line tools, but specifically targeted towards DevOps tools, servers, etc.
- Fully JSON compatible
- Made up of **stanzas** or **blocks**, which roughly equate to JSON objects. Each stanza/block maps to an object type as defined by **Terraform providers** (we'll talk more about providers later)
- https://github.com/hashicorp/hcl

Terraform Project Content Types

Develop Intelligence

`*.tf, *.tf.json`
- HCL or JSON
- these files define your declarative infrastructure and resources

`*.tfstate`
- JSON files that store state, reference to resources
- created and maintained by terraform

`terraform.tfvars, terraform.tfvars.json` and/or **`*.auto.tfvars, *.auto.tfvars.json`**
- HCL or JSON
- variable definitions in bulk
- (more to come on setting variable values at runtime)

Resources

- `*.tf` files contain your **HCL declarative** definitions

```
resource "aws_instance" "web" {
  ami           = "ami-19827362728"
  instance_type = "t2.micro"

  tags {
    Name = "my-first-instance"
  }
}
```

- most **blocks** in your HCL represent a **resource** to be created/maintained by Terraform

Resources

Develop Intelligence

- *resources* are key elements and captured as top-level objects (stanzas) in Terraform configuration files
- each resource stanza indicates the intent to *idempotently* create that resource
- body of resource contains configuration of attributes of that resource
- each provider (e.g., AWS, Azure, etc.) provides its own set of resources and defines the configuration attributes
- when a resource is created by Terraform, it's tracked in Terraform *state*
- resources can refer to attributes of other resources, creating implicit dependencies
    - dependencies trigger sequential creation

Terraform Commands and the CLI

Develop Intelligence

- The CLI is how you'll most often use terraform

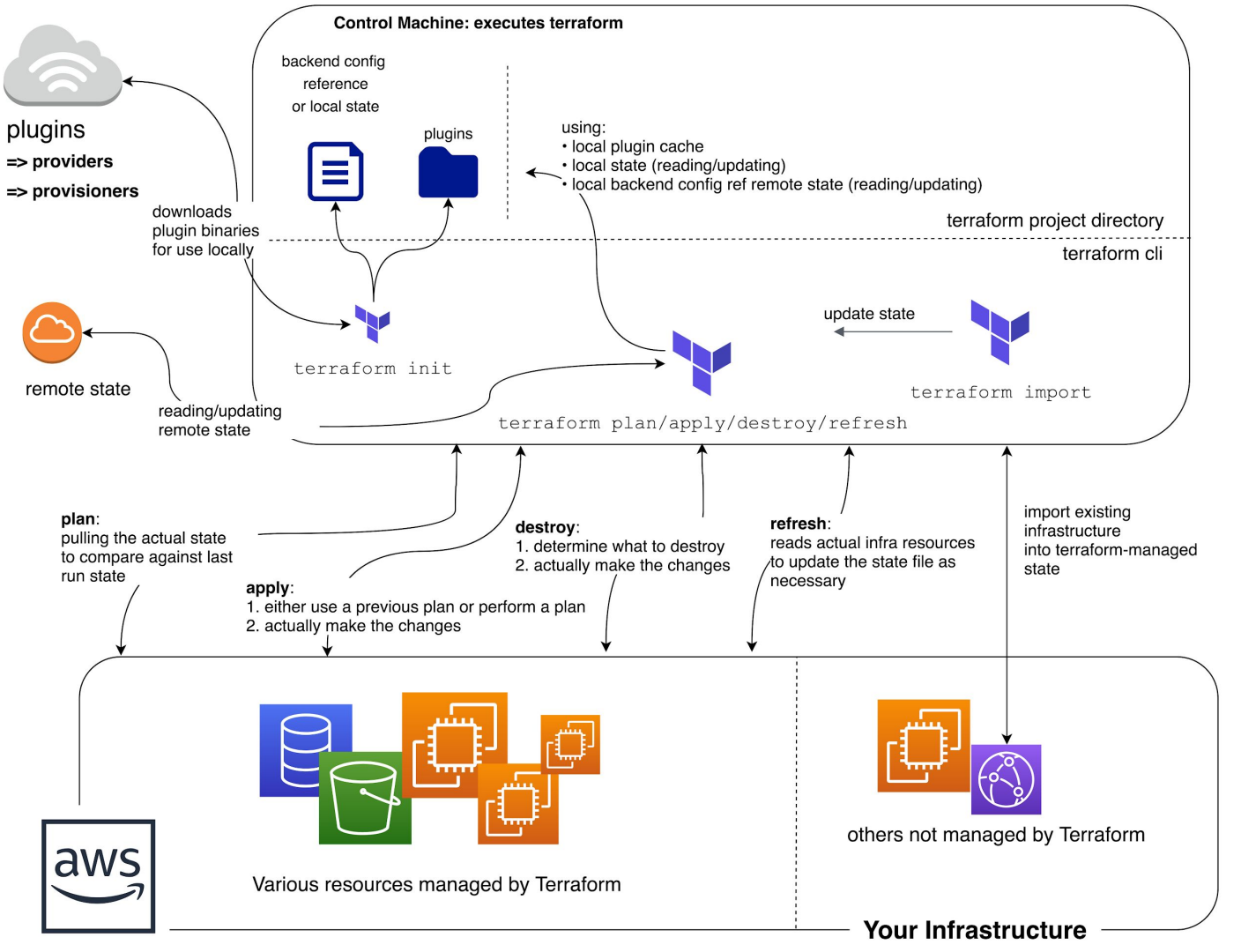
```
terraform init …
terraform plan …
terraform apply …
```

- And plenty more: **`terraform --help`** or https://www.terraform.io/docs/commands/index.html

- Third-party SDKs also available for running and interacting with Terraform (e.g., **`scalr`**)

Big picture look at

Terraform Command Flow



`terraform init`

Develop Intelligence

- a special command, run before other commands/operations
- what does it do?
  - downloads required provider packages
  - downloads modules referenced in the HCL (more on modules later)
  - initializes state
    - local state: ensuring local state file(s) exist
    - remote state: more complex initialization (more on remote state later)
  - basic syntax check

- idempotent

- remember the **`.terraform`** directory?
  - **`init`** downloads the provider packages and modules to this directory
  - also where state files live.

Exercise 1: First Terraform Project

Develop Intelligence

Exercise 1

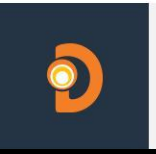
Extending Your Project

Develop Intelligence

- Input Variables
- Locals
- Data Sources
- Provisioners
  - **`remote-exec`**
  - **`local-exec`**
  - **`null_resource`**

Input Variables

- enable interchangeable values to be stored centrally and referenced single or multiple times
- similar to variables in other languages
- declared in **`variable`** stanzas
- parsed first
- cannot interpolate or reference other variables
- allow for default values
- optionally specify value type, e.g.,
  - **`List, Map, String`**



Example Variable Definition

Develop Intelligence

```
variable "instance_size" {
  default     = "t2.micro"
  type        = string # changed in 0.12
  description = "Size of EC2 instance"
}
```

Locals

- mutable values that allow for interpolation and inference
- **CAN** reference variables and other locals
- **CAN'T** be set via arguments from the command line
- use them when a value is used in many places in your code and that value is likely to change
- don't overuse them or your code can be difficult to read

Local Definitions

Develop Intelligence

```
locals {
  one         = "1"
  twelve      = "${local.one}2"
  onetwelve = "${local.one}${local.twelve}"
}
```

Data Sources

- logical references to data objects stored externally to the **`tfstate`** file
- allows you to reference resources not created by Terraform
- examples
  - current default region in AWS CLI
  - AMI ID search
  - AWS ARN lookup
  - AWS VPC CIDR range

Data Source Example: AWS AMI Lookup

Develop Intelligence

```
data "aws_ami" "latest-ubuntu" {
  most_recent = true
  owners = ["099720109477"]

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-*"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}
```

Provisioners

Develop Intelligence

- allow you to run commands during instance provisioning that are run on create, recreate, or taint correction (explained later), but not every time **`terraform apply`** is run
- ties custom logic to idempotent resources
- types
  - local
  - remote
  - **`chef`**
- connectors
  - SSH
  - WinRM

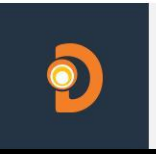
Provisioner Example: local-exec

Develop Intelligence

```
resource "aws_instance" "web" {
  ami           = "ami-19827362728"
  instance_type = "t2.micro"

  tags {
    Name = "my-first-instance"
  }

  provisioner "local-exec" {
    command = "echo 'created instance'"
  }
}
```

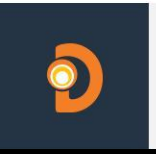


Provisioner Example: remote-exec

Develop Intelligence

```
resource “aws_instance” “web” {
  ...

  provisioner “remote-exec” {
    inline = [
"sudo sed -i
   's/^PasswordAuthentication.*/PasswordAuthentication yes/'
   /etc/ssh/sshd_config",
"sudo service sshd restart",
"wget https://repo.anaconda.com/Anaconda3-Linux-x86_64.sh",
"sh Anaconda3-Linux-x86_64.sh -b"
        ]
  }
}
```



Provisioner example: null_resource

Develop Intelligence

```
resource “null_resource” “first-tf-run” {
  provisioner “local-exec” {
    command = “echo ‘this will run on first tf
apply’”
  }
}
```

Exercise 2: Using Variables

Develop Intelligence

Exercise 2

How Terraform Works

- state and how to query it
- computing plans
- executing plans (**`terraform apply`**)

State

- stores information about resources that are created by Terraform
  - also includes values computed by the provider APIs
- local file
  - `.tfstate`
- or backends are also available…

Backends

Develop Intelligence

- determines how state is loaded and how operations like **`apply`** are executed

- enables non-local file state storage, remote execution, etc.

- why use a backend?
  - can store their state remotely and protect it to prevent corruption
    - some backends, e.g., *Terraform Cloud* automatically store all revisions
  - keep sensitive information off local disk
  - remote operations
    - apply can take a *LONG* time for large infrastructures

Backends (cont'd)

Develop Intelligence

- examples
  - S3
  - swift
  - http
  - Terraform Enterprise
  - etc.

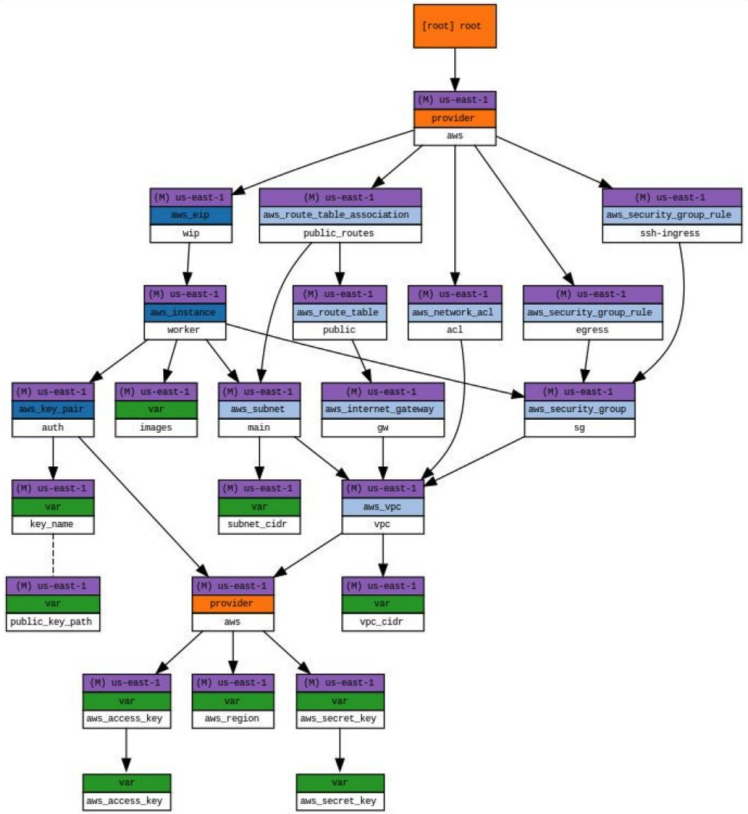
How to Query or See the Current State

Develop Intelligence

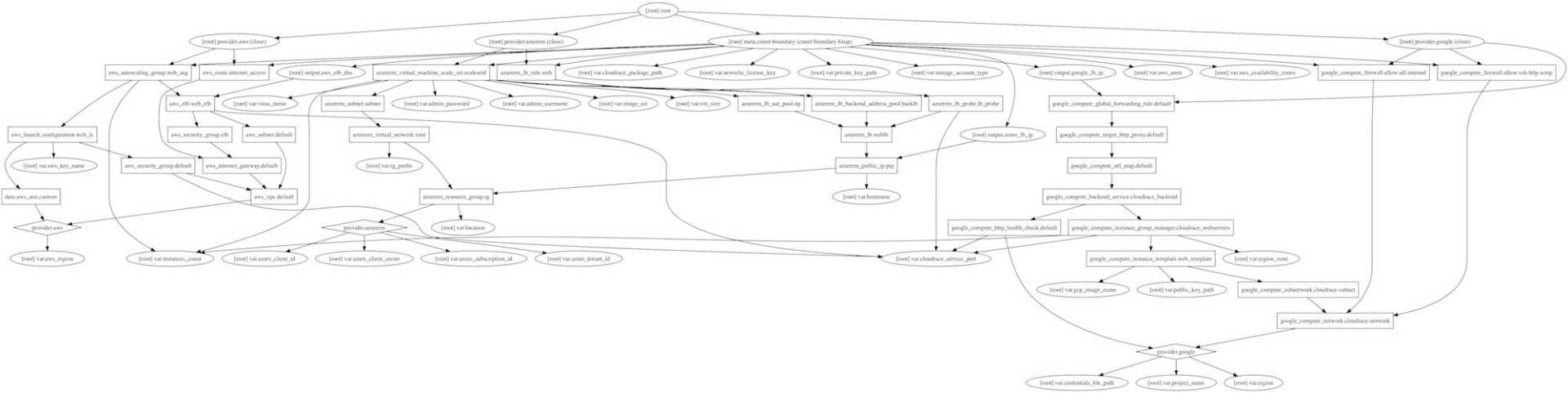
- CLI
  - **`terraform show [-json]`**
    https://www.terraform.io/docs/commands/show.html

- Remote State Data Type
  https://www.terraform.io/docs/providers/terraform/d/remote_state.html

Visualizing Graph Outputs



Visualizing Graph Outputs



Executing Plans: `terraform apply`

- What does **`terraform apply`** do?
  - syntax check
  - check for init
  - refresh state
  - execute plan
  - ask for input
  - execute changes

Exercise 3: Plans and Applies

Develop Intelligence

Exercise 3

Providers

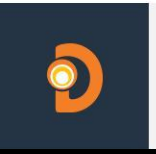
Develop Intelligence

- responsible for understanding API interactions and exposing resources
- Hashicorp helps companies create providers to be added to ecosystem
- declared in HCL config files as a **`provider`** stanza
- each Terraform project can have multiple providers, even of the same type
- describes resources, their inputs, outputs, and the logic to create and change them
- many options
  - AWS, GCP, Azure, and many many others
  - providers available for non-infra services as well such as gmail, MySQL, and Pagerduty

The AWS Provider

Develop Intelligence

- provider documentation
  - https://www.terraform.io/docs/providers/aws/index.html
- HUGE amount of resources
- something like 8 resources per service on average



Configuring the Provider

Develop Intelligence

```
provider "aws" {
  region     = "us-west-1"
  access_key = "[your access key]"
  secret_key = "[your secret access key]"
}
```

Reuse Patterns in Terraform

Develop Intelligence

- **Workspaces**: separate state files for the same HCL
- **Outputs**: automated use of terraform-managed resources
- **Modules**: packaged HCL for reuse

Workspaces

Develop Intelligence

- Workspaces allow you to use the same configuration (HCL/project) for multiple states
  - example: AWS Developer VPCs–each developer could have an identical environment as defined by the configuration, but each managed by a different state by way of separate workspaces

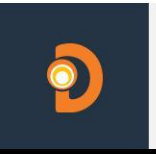
- nothing more than separately-named state files

- both local and remote state backends support workspaces

Outputs

Develop Intelligence

- *inputs* to a Terraform config are declared with variables stanzas
- *outputs* are declared with a special output stanza
- can be referenced through the modules interface or the CLI



Output Definition

Develop Intelligence

```
output "instance_public_ip" {
  value = aws_instance.web.public_ip
}
```

Modules

- *modules* are a critically important concept in Terraform
- basically, every Terraform working directory, as long as it has variable stanzas, is a module
- this allows developers to compose reusable blocks of configuration and reference them with module stanzas

Modules

- allow for modularized configuration (create separate modules for different parts of configuration), aka module composition
- every project has at least one module (the “root” module), but root can have a tree of children
- child modules have input variables passed in from parent module
- modules can be defined by configuration files in local filesystem or remote source

Modules

- can publish modules in Terraform registry to make them easy to find
  - **`source`** attribute identifies location of module, e.g.,

```
module "webserver" {
  source = "./webserver" # module in this dir
  instance_type = "t2.micro"
}
```

- most attributes of a module are input variables passed in from parent
- module's outputs can be accessed and used by parent (and passed to other child modules of the parent)

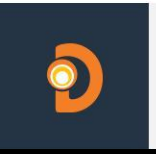
Module Sources

- Terraform allows the user to pull modules from various locations
  - local paths
  - Github
  - Terraform Registry
  - Bitbucket
  - HTTP
  - S3 Buckets

```
module "consul" {
  source = "github.com/hashicorp/example"
}
```

```
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0"
}
```

- More info
  - https://www.terraform.io/docs/modules/sources.html



Module Example

Develop Intelligence

```
variable “thing” {
  type = string
}

resource null_resource “null” {
  provisioner local-exec {
    command = "echo ${var.thing}"
  }
}
```

Using the Module

Develop Intelligence

```
module "my_module" "printer" {
  source = "./my_module"

  # this is a variable passed into module
  thing = "this should be printed"
}
```

Exercise 4: Querying State

Develop Intelligence

Exercise 4

Exercise 5

Exercise 6

Experimentation Time: Webserver

- Let's first do this by hand, then use Terraform
- Create an EC2 instance, type **t2.micro**, whatever distro you like
  - provision this instance to install **nginx**
  - make sure we can access **nginx** from the outside world
  - use online docs for help, use each other, and I'm here too if you need

Experimentation Time: Webserver (cont'd)

Develop Intelligence

- You can go in many different directions as you solve this, and there are plenty of things to play with along the way and after, some ideas:
  - imagine your Terraform code to spin up this instance needed to be reused, what are your options–make it reusable!
    - **hints**: workspaces, package as a module
  - pretend that some other automation needed the public IP from the created instance, how would you do this?
    - **hints**: outputs, and play with wrapping Terraform commands in a parent script (e.g., bash or Python)
  - ...and finally, ANYTHING else you'd like to start learning more about, **`terraform graph`** maybe?

Introduction to Terraform

Develop Intelligence

Welcome to Day 2!

Day 1 Recap and Review

- Let’s discuss some takeaways from yesterday
- What questions do you have after sleeping on it?
- Corrections, omissions
- OK, let’s get into it

Error Handling and Debugging

Develop Intelligence

- Most errors fall into one of four types
  - Process Errors
  - Syntax Errors
  - Validation Errors
  - Passthrough Errors

Process Errors

- errors due to process not being followed

- e.g.,
  - running **`apply`** before **`init`**
  - variables not fully populated

Syntax Errors

Develop Intelligence

- caused by an error in syntax, e.g.,
  - HCL codebase syntax or parameter errors
  - incorrect usage of built in functions
  - type errors

Validation Errors

Develop Intelligence

- preliminary validation built into provider occuring before plan
- usually more detailed

Provider Errors / Passthrough

Develop Intelligence

- errors received from provider or third party API while in process of refresh, plan, apply, etc.
  - usually most difficult to troubleshoot
  - requires knowledge of provider's tech (e.g., AWS)

Troubleshooting Commands

Develop Intelligence

- **`terraform validate`**
  - performs a syntax check on all terraform files in the directory
  - displays an error if any of the files doesn't validate
  - does *NOT* check formatting
  - what does it check…?

terraform validate

Develop Intelligence

- invalid HCL syntax (e.g., missing quote or equal sign)
- invalid HCL references (e.g., variable name or attribute which doesn't exist)
- same provider declared multiple times
- same module declared multiple times
- same resource declared multiple times
- invalid module name
- interpolation used in places where it's unsupported (e.g., variable, **`depends_on`**, **`module.source`**, **`provider`**)
- missing value for a variable (none of **`-var foo=...`** flag, **`-var-file=foo.vars`** flag, **`TF_VAR_foo`** environment variable, terraform.tfvars, or default value in the configuration)

Troubleshooting: `terraform fmt`

Develop Intelligence

- rewrites Terraform files in a canonical format/style
- by default, scans the current directory for configuration files
  - if the dir argument is provided then it will scan that given directory instead
  - if dir is a single dash (-) then **`fmt`** will read from standard input

Troubleshooting: `terraform graph`

Develop Intelligence

- generates a visual representation of either a configuration or execution plan
  - output is in DOT format, which can be used by GraphViz to generate charts: https://www.terraform.io/docs/internals/graph.html

- e.g.,

```
terraform graph | dot -Tsvg > graph.svg
```

Troubleshooting: `terraform console`

Develop Intelligence

- creates an interactive console for testing interpolations
  - similar to running the Python interpreter in interactive mode
- great for testing complex conditionals

Exercise 7

Interpolation

- embedded within strings in Terraform, whether you're using the HCL or JSON, you can interpolate other values.
  - These interpolations are wrapped in **`${...}`**, such as **`${var.foo}`**
- allows you to reference variables, attributes of resources, call functions, etc.
- simple math like
  - **`${count.index + 1}`**
- allows for conditional statements
- https://www.terraform.io/docs/configuration-0-11/interpolation.html

Built-in Functions

Develop Intelligence

- built-in functions:
  - Terraform ships with built-in functions
  - called with the syntax **`name(arg, arg2, ...)`**
  - e.g., to read a file:

    `${file("path.txt")}`
  - https://www.terraform.io/docs/configuration-0-11/interpolation.html#built-in-functions

Conditionals

- interpolations may contain conditionals to branch on the final value

- syntax

```
CONDITION ? TRUEVAL : FALSEVAL
```

Primitive Data Types (new for v0.12!)

Develop Intelligence

- **`string`**
  - use the var prefix followed by the variable name
  - e.g., **`${var.foo}`** is how you would use the variable in HCL for interpolation or reference
- **`number`**
  - can be referenced as a number, so in arithmetic for example

    `${var.foo + 1}`
- **`bool`**
  - can be referenced as a boolean in logic, so something like

    `${var.foo == true ? “foo is true” : “foo is false”}`

Extended Data Types (new for v0.12!)

Develop Intelligence

- **`list(<type>)`**
  - ordered list of things, i.e., array
    - e.g., **`${var.subnets}`** would get the value of the subnets *list*
    - you can also return list elements by index: **`${var.subnets[0]}`**
- **`set(<type>)`**
  - similar to a list, but: requires a type, unique values, no ordering
- **`map(<type>)`**
  - a collection of values where each is identified by a string
  - e.g., **`${var.amis["us-east-1"]}`** would get the value of the **`us-east-1`** key within the **`amis`** map variable

Extended Data Types (cont'd)

Develop Intelligence

- `object({ <attr name> = <type>, ... })`
  - like many other language object types, with properties containing other values
- `tuple([<type>, ...])`
  - very similar to a list, mixed strictly defined typed list of things

Count Parameter

- resources can be duplicated or conditionally created via the count parameter

```
resource "aws_instance" "web" {
  count         = 2
  ami           = "${var.ami}"
  instance_type = "${var.instance_type}"

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

Data Reference

Develop Intelligence

- attributes of current resource
  - syntax is **`self.ATTRIBUTE`**
  - e.g., **`${self.private_ip}`** interpolates resource's private IP address

Data Reference (cont'd)

Develop Intelligence

- attributes of other resources
  - syntax is **`TYPE.NAME.ATTRIBUTE`**
  - **`${aws_instance.web.id}`**
    - interpolate ID attribute from the **`aws_instance`** resource **`web`**
    - if resource has a count attribute set, you can access individual attributes with a zero-based index, such as **`${aws_instance.web.0.id}`**
    - or use the splat syntax to get a list of all the attributes: **`${aws_instance.web.*.id}`**
  - **UPDATE**: 0.12 syntax for these is now:
    `${aws_instance.web[0].id}`
    `${aws_instance.web[*].id}`

Data Sources and Reference

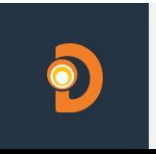
Develop Intelligence

- attributes of a data source
  - **`data.TYPE.NAME.ATTRIBUTE`**
  - **`${data.aws_ami.ubuntu.id}`**
    - interpolate **`id`** attribute from the **`aws_ami`** data source **`ubuntu`**
    - if data source has a **`count`** attribute set, access individual attributes with a zero-based index, e.g.,
      `${data.aws_subnet.example.0.cidr_block}`
      `${data.aws_subnet.example[0].cidr_block}` (0.12)
    - or use the splat syntax to get a list of all the attributes:
      `${data.aws_subnet.example.*.cidr_block}`
      `${data.aws_subnet.example[*].cidr_block}` (0.12)

Data Sources and Reference (cont'd)

Develop Intelligence

- Referencing values output from another module
  - **`module.MODULE_NAME.MODULE_OUTPUT_NAME`**



Resource Example w/Conditional

Develop Intelligence

```
resource "aws_instance" "web" {
  ami           = "${var.ami}"
  instance_type = "${var.instance_type}"

  tags {
    Name = "${var.env == "production" ?
"production-web" : "staging-web" }"
  }
}
```

Exercise 8

Keeping Terraform in Sync with Infra

Develop Intelligence

- configuration drift
  - things change!
  - Terraform can bring those things back in line naturally
- **`plan`**
  - when executing a plan, Terraform can output machine readable syntax (exit codes) that can be used to monitor for manual infra changes
  - if the infra changes, plans will suddenly detect drift and inform alarms
- **`apply`**
  - thanks to Terraform's idempotency, corrections are natural and easy

Keeping Terraform in Sync with Infra

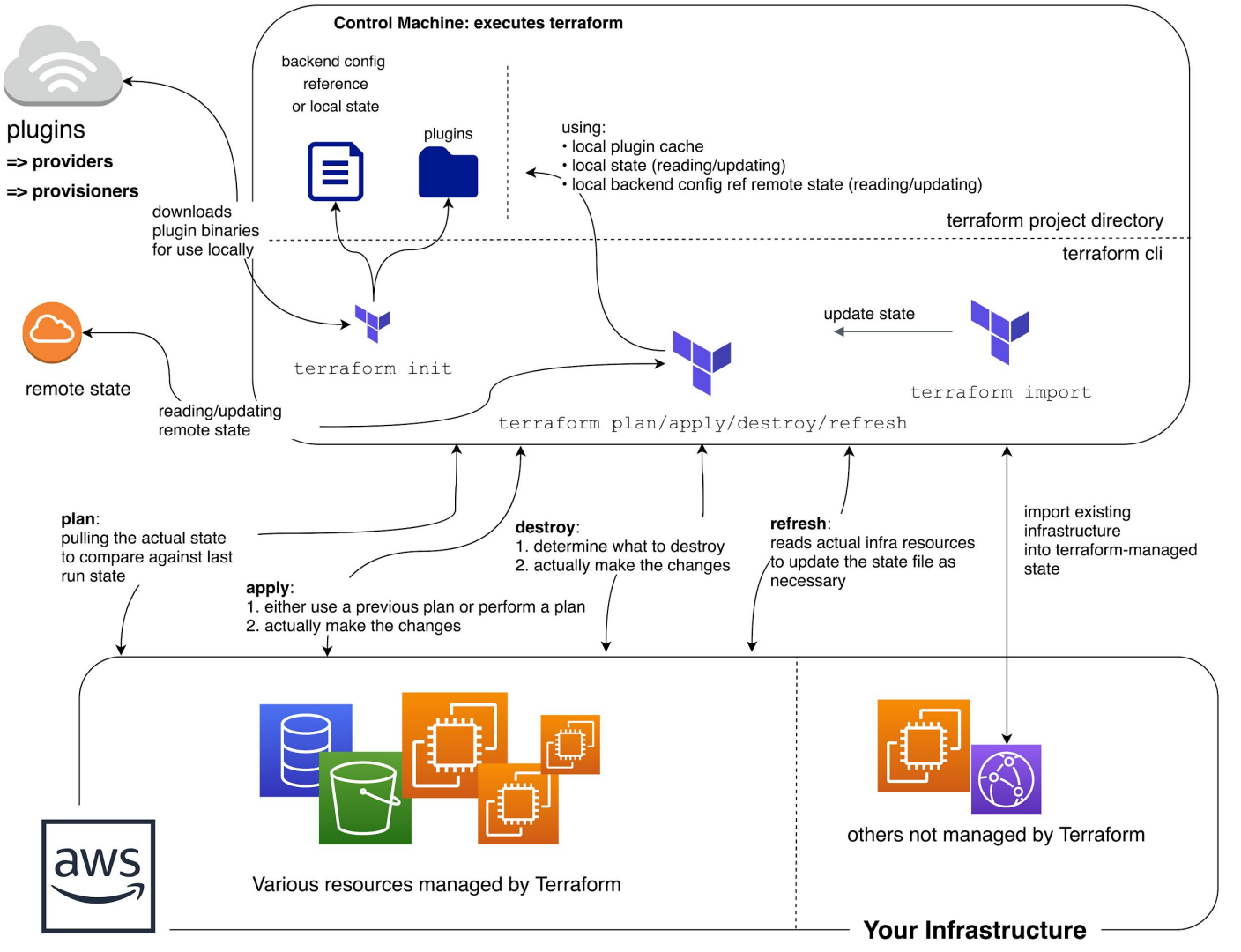
Develop Intelligence

- what if we want to keep the changes?
- you can import them
  - use **`terraform import`** to pull in the changes to the state
  - must also change the Terraform config to match any changes
  - if you have a clean plan with no planned changes, you were successful
- e.g.,

```
terraform import aws_instance.my_instance i-abcd1234
```

Big picture look at

Terraform Command Flow



Exercise 9: Resource Counts and Conditionals

Develop Intelligence

Exercise 9

Terraform Expressions

Develop Intelligence

- can set attributes, outputs and locals to expressions
- expressions can refer to
  - literal values or complex literal values: **`true, 13,“us-west1”,[1, 2],{a:1, b:2}`**
  - resource or data source attributes: **`<RESOURCE TYPE>.<NAME>, data.<DATA TYPE>.<NAME>`**
  - type indices: **`local.list[3], local.object.attrname, local.map["keyname"]`**
  - variables: **`var.<NAME>`**
  - locals: **`local.<NAME>`**

  …

Terraform Expressions (cont'd)

Develop Intelligence

- module outputs: **`module.<MODULE NAME>.<OUTPUT NAME>`**
- path variables: **`path.module, path.root, path.cwd`**
- workspace setting: **`terraform.workspace`**
- built-in functions using any of the above as arguments
  - **`max(5, 12, var.my_value)`**
- arithmetic, logical, or comparison operators combining the above
- conditional expressions: **`var.a != "" ? var.a : "default-a"`**
- string template interpolation: **`"Hello, ${var.name}!"`**
- string template directives (new in v0.12):

```
"Hello, %{ if var.name != "" }${var.name}%{ else }unnamed%{ endif }!"
```

Terraform Expressions (cont'd)

Develop Intelligence

- Multi-line string templates ("heredoc" syntax, new in v0.12)
- Looping string directive (**`for/endfor`**, new in v0.12)

```
<<EOT
%{ for ip in aws_instance.example.*.private_ip }
server ${ip}
%{ endfor }
EOT
```

Terraform Expressions

- **Splat syntax (v0.12 at top, v0.11 below)**

```
var.list[*].id
```

```
var.list[*].interfaces[0].name
```

```
${aws_instance.web.*.id}
```

Terraform Expressions

- **`for`** expressions to convert lists/maps/tuples/objects to other lists/maps/tubles/objects (new in v0.12)

```
[for s in var.list : upper(s)]
```

```
{for s in var.list : s => upper(s)}
```

```
[for s in var.list : upper(s) if s != ""]
```

```
[for k, v in var.map : length(k) + length(v)]
```

```
{for s in var.list : substr(s, 0, 1) => s... if s != ""}
```

Terraform Expressions

- **`dynamic`** blocks (new in v0.12)

```
resource "aws_security_group" "example" {
  name = "example" # can use expressions here

  dynamic "ingress" {
    for_each = var.service_ports
    content {
      from_port = ingress.value
      to_port   = ingress.value
      protocol  = "tcp"
    }
  }
}
```

Terraform Meta-arguments

Develop Intelligence

- resources, data sources, modules, and outputs can have meta-arguments (available across all types of all providers)
- modules have: `source`, `version`, `providers`
- outputs have: `depends_on`
- resources have: `depends_on`, `count`, `for_each` (new in v0.12), `provider`, `lifecycle`, `provisioner` (`provisioner` can have `connection` inside)
- data sources have same as resources except for `lifecycle`
- `depends_on` forces a dependency on another object even if no implicit dependency by referring to an attribute of another object
- `lifecycle` controls how resources are modified when configuration changes
- `for_each` is like count except it iterates over a set (unordered list) or map, has `each.key`, `each.value` instead of `count.index` to refer to each index

Terraform Meta-arguments

- `providers` and `provider` are used when dealing with multiple providers in the same configuration

```
provider "aws" {
  alias  = "usw1"
  region = "us-west-1"
}

provider "aws" {
  alias  = "usw2"
  region = "us-west-2"
}

module "tunnel" {
  source    = "./tunnel"
  providers = {
    aws.src = "aws.usw1"
    aws.dst = "aws.usw2"
  }
}
```

```
# default configuration
provider "google" {
  region = "us-central1"
}

# alternative, aliased configuration
provider "google" {
  alias  = "europe"
  region = "europe-west1"
}

resource "google_compute_instance" "example" {
  # This "provider" meta-argument selects the google provider
  # configuration whose alias is "europe", rather than the
  # default configuration.
  provider = google.europe

  # ...
}
```

Upgrading from v0.11 to v0.12

Develop Intelligence

- Full details at: https://www.terraform.io/upgrade-guides/0-12.html
- Upgrade to v0.11.14 first and run `terraform init`, `terraform apply`, and `terraform 0.12checklist` (this command makes suggestions for things to change before running automated upgrade command)
- If the output of `terraform 0.12checklist` suggests some updates, complete them and rerun it and verify no other tasks are suggested
- Then run the automated `terraform 0.12upgrade` command, then check automatically updated files for `TF-UPGRADE-TODO` markers/comments and make required changes
- Rerun `terraform plan` and verify no changes are required

Backends

Develop Intelligence

- Backends are the concept that terraform uses to store state
- Defaults to local tfstate file.
- Others available:
  - S3
  - HTTP
  - Consul
  - Artifactory
  - Etcd
  - Terraform Enterprise
  - etc...

Backends: S3

Develop Intelligence

Because this course is about Terraform with AWS specifically, let's talk about the S3 state backend:

- Uses a bucket and path to an object (the state file) in the bucket for a central place to store state
- Supports locking using an AWS Dynamo DB table
- See https://www.terraform.io/docs/backends/types/s3.html for more info

Exercise 10

Develop Intelligence

Exercise 11

Experimentation Time

Develop Intelligence

- Let’s discuss some idea for experiments
  - You should be just about armed with all the knowledge you need to explore any part of Terraform you like, groups encouraged!
  - What would be most useful to your actual work?

Experimentation Time–Some Ideas

- How could you use Terraform from a server that has no internet access?
  - **`terraform import, terraform graph`**
  - **`tfenv`** for managing versions of the terraform binary
  - new capabilities in 0.12, especially https://www.terraform.io/docs/configuration/expressions.html
  - Look at the **`student-environments`** terraform code in the repo. This is how I set up all of your student accounts/aliases in AWS in prep for this course. Can you identify ways that could be used to improve this terraform, especially in light of 0.12 capabilities?
  - Terraform gotchas: https://blog.gruntwork.io/terraform-tips-tricks-loops-if-statements-and-gotchas-f739bbae55f9 and https://heap.io/blog/engineering/terraform-gotchas

YOU'RE NOW READY FOR THE WILD WORLD OF TERRAFORM

THANKS FOR BEING HERE!

Develop Intelligence

What should I do now?

Develop Intelligence

- Get in touch at any time: dave@developintelligence.com
- Take what you've learned, experiment, and then experiment some more!
- Get involved in the Terraform community
- Follow releases, Github issues, etc.
  - It's the best way to stay up on the current state of affairs!

Surveys: Check your email for a survey link

(Complete the survey now - takes 2 minutes)