Terraform Resources
List & Objects

Resource List

- By default, resource & module creation results in a single object
  - When creating an EC2 resource, only one resource is being created
- There are many cases where multiple resources of the same type are needed
  - For example, creating a fleet of EC2 instances for batch work
  - You are tasked to deploy 10 EC2 instances, with same arguments & values. How would you do it?
    - Manually create 10 separate ec2 resources with duplicate arguments and values. Or…
    - Create one ec2 resource, which is a list of length 10

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"

  tags = {
    Name = "Server ${count.index}"
  }
}
```

HashiCorp Terraform

Resource List

- Terraform provides an easy way to create a list of resources, sharing same argument & values.
- Using count keyword, supply a number of how many of that resource to create
  - count is also used for conditional resource creation, remember? count = 0 means, do not create
- When using count, Terraform stores the resource in the state as an array/list
  - Without count, resource stored as: aws_instance.server
  - With count, resource stored as: aws_instance.server[0], aws_instance.server[1], ...
- You can access each resource by the index

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"

  tags = {
    Name = "Server ${count.index}"
  }
}
```

HashiCorp Terraform

Resource List

- When creating resource with count, in background, Terraform goes through a loop
  - Where length of the loop is the value of count. In example, length = 4
- In each cycle of the loop, you have access to the given index at that time
- Terraform exposes the index in a variable called count.index
- count.index is useful for providing arguments with index specific values
  - In example, the name of the server is labeled based on the index, at the given cycle
    - "Server 0", "Server 1", "Server 2"...

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"

  tags = {
    Name = "Server ${count.index}"
  }
}
```

HashiCorp Terraform

Resource Object

- Terraform provides powerful way to create a mapping of resources
  - Similar to count, creating logical grouping of resources
  - Useful to create a set of resources that have the same arguments, but differ in value
- Using the for_each keyword, provide an object with key/value pairs
  - Terraform will create instances of the resource by iterating each key/value pair

```
resource "azurerm_resource_group" "rg" {
  for_each = {
    a_group = "eastus"
    another_group = "westus2"
  }
  name     = each.key
  location = each.value
}
```

HashiCorp Terraform

Resource Object

- Terraform exposes the each variable, which has two important properties:
  - each.key -> the value for the key in the key/value pair
  - each.value -> the value for the value in the key/value pair
- The each variable exposes the current key/value pairs for the iteration
- In the example, we have two items or two key/value pairs, which means Terraform creates two resources
- Check the state file to see how resources are being stored when using for_each

```
resource "azurerm_resource_group" "rg" {
  for_each = {
    a_group = "eastus"
    another_group = "westus2"
  }
  name     = each.key
  location = each.value
}
```

HashiCorp Terraform

Resource Object

- for_each is powerful and has many different usecases, can get complex quickly
- for_each can be ran on a set of a list, does not have to be an object
  - A set is similar to an array/list, but it only retains unique values of that array/list
  - When for_each runs on a set, only each.key property is exposed
    - There is no key/value pair
  - In this case, Terraform stores the resource with the key as the index
    - aws_iam_user.the-accounts["Todd"], aws_iam_user.the-accounts["James"], ...

```
resource "aws_iam_user" "the-accounts" {
  for_each = toset( ["Todd", "James", "Alice", "Dottie"] )
  name     = each.key
}
```

HashiCorp Terraform