Lab: Terraform Collections and Structure Types

Duration: 10 minutes

As you continue to work with Terraform, you're going to need a way to organize and structure data. This data could be input variables that you are giving to Terraform, or it could be the result of resource creation, like having Terraform create a fleet of web servers or other resources. Either way, you'll find that data needs to be organized yet accessible so it is referenceable throughout your configuration. The Terraform language uses the following types for values:

- **string:** a sequence of Unicode characters representing some text, like "hello".
- **number:** a numeric value. The number type can represent both whole numbers like 15 and fractional values like 6.283185.
- **bool:** a boolean value, either true or false. bool values can be used in conditional logic.
- **list (or tuple):** a sequence of values, like ["us-west-1a", "us-west-1c"]. Elements in a list or tuple are identified by consecutive whole numbers, starting with zero.
- **map (or object):** a group of values identified by named labels, like {name = "Mabel", age = 52}. Maps are used to store key/value pairs.

Strings, numbers, and bools are sometimes called primitive types. Lists/tuples and maps/objects are sometimes called complex types, structural types, or collection types. Up until this point, we've primarily worked with string, number, or bool, although there have been some instances where we've provided a collection by way of input variables. In this lab, we will learn how to use the different collections and structure types available to us.

- Task 1: Create a new list and reference its values using the index
- Task 2: Add a new map variable to replace static values in a resource
- Task 3: Iterate over a map to create multiple resources
- Task 4: Use a more complex map variable to group information to simplify readability

Task 1: Create a new list and reference its values

In Terraform, a *list* is a sequence of like values that are identified by an index number starting with zero. Let's create one in our configuration to learn more about it. Create a new variable that includes a list of different availability zones in AWS. In your `variables.tf` file, add the following variable:

```
variable "us-east-1-azs" {
    type = list(string)
    default = [
```

```
        "us-east-1a",
        "us-east-1b",
        "us-east-1c",
        "us-east-1d",
        "us-east-1e"
    ]
}
```

In your `main.tf` file, add the following code that will reference the new list that we just created:

```
resource "aws_subnet" "list_subnet" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.200.0/24"
  availability_zone = var.us-east-1-azs
}
```

Go and and run a `terraform plan`. You should receive an error, and that's because the new variable `us-east-1-azs` we just created is a list of strings, and the argument `availability_zones` is expecting a single string. Therefore, we need to use an identifier to select which element to use in the list of strings.

Let's fix it. Update the `list_subnet` configuration to specify a specific element referenced by its indexed value from the list we provided - remember that indexes start at `0`.

```
resource "aws_subnet" "list_subnet" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.200.0/24"
  availability_zone = var.us-east-1-azs[0]
}
```

Run a `terraform plan` again. Check out the output and notice that the new subnet will be created in `us-east-1a`, because that is the first string in our list of strings. If we used `var.us-east-1-azs[1]` in the configuration, Terraform would have built the subnet in `us-east-1b` since that's the second string in our list.

Go ahead and run `terraform apply` to apply the new configuration and build the subnet.

Task 2 - Add a new map variable to replace static values in a resource

Let's continue to improve our `list_subnet` so we're not using any static values. First, we'll work on getting rid of the static value used for the subnet CIDR block and use a map instead. Add the following code to `variables.tf`:

```
variable "ip" {
```

```
  type = map(string)
  default = {
    prod = "10.0.150.0/24"
    dev  = "10.0.250.0/24"
  }
}
```

Now, let's reference the new variable we just created. Modify the `list_subnet` in `main.tf`:

```
resource "aws_subnet" "list_subnet" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = var.ip["prod"]
  availability_zone = var.us-east-1-azs[0]
}
```

Run `terraform plan` to see the proposed changes. In this case, you should see that the subnet will be replaced. The new subnet will have an IP subnet of 10.0.150.0/24, because we are now referencing the value of the `prod` key in our map.

Go ahead and run `terraform apply -auto-approve` to apply the new changes.

Before we move on, let's fix one more thing here. We still have a static value that is "hardcoded" in our `list_subnet` that we should use a variable for instead. We already have a `var.environment` that dictates the environment we're working in, so we can simply use that in our `list_subnet`.

Modify the `list_subnet` in `main.tf` and update the cidr_block argument to the following:

```
resource "aws_subnet" "list_subnet" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = var.ip[var.environment]
  availability_zone = var.us-east-1-azs[0]
}
```

Run `terraform plan` to see the proposed changes. In this case, you should see that the subnet will again be replaced. The new subnet will have an IP subnet of `10.0.250.0/24`, because we are now using the value of `var.environment` to select the key in our map for the variable `var.ip` and the default is `dev`.

Go ahead and run `terraform apply -auto-approve` to apply the new changes.

Task 3: Iterate over a map to create multiple resources

While we're in much better shape for our `list_subnet`, we can still improve it. Oftentimes, you'll want to deploy multiple resources of the same type but each resource should be slightly different for

different use cases. In our example, if we wanted to deploy BOTH a `dev` and `prod` subnet, we would have to copy the resource block and create a second one so we could refer to the other subnet in our map. However, there's a fairly simple way that we can iterate over our map in a single resource block to create and manage both subnets.

To accomplish this, use a `for_each` to iterate over the map to create multiple subnets in the same AZ. Modify your `list_subnet` to the following:

```
resource "aws_subnet" "list_subnet" {
  for_each          = var.ip
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = each.value
  availability_zone = var.us-east-1-azs[0]
}
```

Run a `terraform plan` to see the proposed changes. Notice that our original subnet will be destroyed and Terraform will create two two subnets, one for `prod` with its respective CIDR block and one for `dev` with its respective CIDR block. That's because the `for_each` iterated over the map and will create a subnet for each key. The other major difference is the resource ID for each subnet. Notice how it's creating `aws_subnet.list_subnet["dev"]` and `aws_subnet.list_subnet["prod"]`, where the names are the keys listed in the map. This gives us a way to clearly understand what each subnet is. We could even use these values in a tag to name the subnet as well.

Go ahead and apply the new configuration using a `terraform apply -auto-approve`.

Using `terraform state list`, check out the new resources:

```
$ terraform state list
...
aws_subnet.list_subnet["dev"]
aws_subnet.list_subnet["prod"]
```

You can also use `terraform console` to view the resources and more detailed information about each one (use CTLR-C to get out when you're done):

```
$ terraform console
> aws_subnet.list_subnet
{
  "dev" = {
    "arn" = "arn:aws:ec2:us-east-1:1234567890:subnet/subnet-052
        d26040d4b91a51"
    "assign_ipv6_address_on_creation" = false
...
```

Task 4: Use a more complex map variable to group information to simplify readability

While the previous configuration works great, we're still limited to using only a single availability zone for both of our subnets. What if we wanted to use a single resource block but have unique settings for each subnet? Well, we can use a map of maps to group information together to make it easier to iterate over and, more importantly, make it easier to read for you and others using the code.

Create a "map of maps" to group information per environment. In `variables.tf`, add the following variable:

```
variable "env" {
  type = map(any)
  default = {
    prod = {
      ip = "10.0.150.0/24"
      az = "us-east-1a"
    }
    dev  = {
      ip = "10.0.250.0/24"
      az = "us-east-1e"
    }
  }
}
```

In `main.tf`, modify the `list_subnet` to the following:

```
resource "aws_subnet" "list_subnet" {
  for_each          = var.env
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = each.value.ip
  availability_zone = each.value.az
}
```

Run a `terraform plan` to view the proposed changes. Notice that only the `dev` subnet will be replaced since we're now placing it in a different availability zone, yet the `prod` subnet remains unchanged.

Go ahead and apply the configuration using `terraform apply -auto-approve`. Feel free to log into the AWS console to check out the new resources.

Once you're done, feel free to delete the variables and `list_subnet` that was created in this lab, although it's not required.