## Lab: Variables

We don't want to hardcode all of our values in the main.tf file. We can create a variable file for easier use. In the `variables block` lab, we created a few new variables, learned how to manually set their values, and even how to set the defaults. In this lab, we'll learn the other ways that we can set the values for our variables that are used across our Terraform configuration.

- Task 1: Set the value of a variable using environment variables
- Task 2: Declare the desired values using a tfvars file
- Task 3: Override the variable on the CLI

### Task 1: Set the value of a variable using environment variables

Often, the default values won't work and you will want to set a different value for certain variables. In Terraform OSS, there are 3 ways that we can set the value of a variable. The first way is setting an environment variable before running `terraform plan` or `terraform apply` command.

To set a value using an environment variable, we will use the `TF_VAR_` prefix, which is followed by the name of the variable. For example, to set the value of a variable named "variables_sub_cidr", we would need to set an environment variable called `TF_VAR_variables_sub_cidr` to the desired value.

On the CLI, use the following command to set an environment variable to set the value of our subnet CIDR block:

```
$ export TF_VAR_variables_sub_cidr="10.0.203.0/24"
```

#### Task 1.1

Run a `terraform plan` to see the results. You'll find that Terraform wants to replace the subnet since we updated the CIDR block of the subnet using an environment variable.

> Note the value set in an environment variable takes precedence over the default value set in the variable block.

```
Terraform used the selected providers to generate the following execution plan. Resource
-/+ destroy and then create replacement

Terraform will perform the following actions:
```



```
  # aws_subnet.terraform-subnet must be replaced
-/+ resource "aws_subnet" "terraform-subnet" {
      ~ arn                             = "arn:aws:ec2:us-east-1:603991114860:subnet/sub
      ~ availability_zone_id            = "use1-az6" -> (known after apply)
      ~ cidr_block                      = "10.0.202.0/24" -> "10.0.203.0/24" # forces re
      ~ id                              = "subnet-0b424eed2dc2822d0" -> (known after app
      + ipv6_cidr_block_association_id  = (known after apply)
      - map_customer_owned_ip_on_launch = false -> null
      ~ owner_id                        = "603991114860" -> (known after apply)
        tags                            = {
            "Name"      = "sub-variables-us-east-1a"
            "Terraform" = "true"
        }
        # (5 unchanged attributes hidden)
    }

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

**Task 1.2**

Let's go ahead and apply our new configuration, which will replace the subnet with one using the CIDR block of "10.0.203.0/24". Run a `terraform apply`. Don't forget to accept the changes by typing `yes`.

### Task 2: Declare the desired values using a tfvars file

Another way we can set the value of a variable is within a tfvars file. This is a special file that Terraform can use to retrieve specific values of variables without requiring the operator (you!) to modify the variables file or set environment variables. This is one of the most popular ways that Terraform users will set values in Terraform.

In the same Terraform directory, create a new file called `terraform.tfvars`. In that file, let's add the following code:

```
# Public Subnet Values
variables_sub_auto_ip = true
variables_sub_az      = "us-east-1d"
variables_sub_cidr    = "10.0.204.0/24"
```



### Task 2.1

Run a `terraform plan` to see the results. You'll find that Terraform wants to replace the subnet since we updated the CIDR block of the subnet using a tfvars file.

> Note the value set in a .tfvars file takes precedence over an environment variable and the default value set in the variable block.

```
Terraform used the selected providers to generate the following execution plan. Resource
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_subnet.terraform-subnet must be replaced
-/+ resource "aws_subnet" "terraform-subnet" {
      ~ arn                             = "arn:aws:ec2:us-east-1:603991114860:subnet/sub
      ~ availability_zone               = "us-east-1a" -> "us-east-1d" # forces replacem
      ~ availability_zone_id            = "use1-az6" -> (known after apply)
      ~ cidr_block                      = "10.0.203.0/24" -> "10.0.204.0/24" # forces re
      ~ id                              = "subnet-0d9ef3f20d902ff28" -> (known after app
      + ipv6_cidr_block_association_id  = (known after apply)
      - map_customer_owned_ip_on_launch = false -> null
      ~ owner_id                        = "603991114860" -> (known after apply)
      ~ tags                            = {
          ~ "Name"      = "sub-variables-us-east-1a" -> "sub-variables-us-east-1d"
            # (1 unchanged element hidden)
        }
      ~ tags_all                        = {
          ~ "Name"      = "sub-variables-us-east-1a" -> "sub-variables-us-east-1d"
            # (1 unchanged element hidden)
        }
        # (3 unchanged attributes hidden)
    }

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

### Task 2.2

Let's go ahead and apply our new configuration, which will replace the subnet with one using the CIDR block of "10.0.204.0/24". Run a `terraform apply`. Don't forget to accept the changes by typing `yes`.





### Task 3: Override the variable on the CLI

Finally, the last way that you can set the value for a Terraform variable is to simply set the value on the command line when running a `terraform plan` or `terraform apply` using a flag. You can set the value of a single variable using the `-var` flag, or you can set one or many variables using the `-var-file` flag and point to a file containing the variables and corresponding values.

On the CLI, run the following command:

```
$ terraform plan -var variables_sub_az="us-east-1e" -var variables_sub_cidr="10.0.205.0/
```

You'll see that we've now set the variable `variables_sub_az` equal to "us-east-1e" and the variable `variables_sub_cidr` to "10.0.205.0/24" which are different from our current infrastructure. As a result, Terraform wants to replace the existing subnet. Terraform uses the last value it finds, overriding any previous values.

> Any values set on the CLI will take precedence over ANY other value set in a different way (ENV, tfvars, default value)

```
Terraform used the selected providers to generate the following execution plan. Resource
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_subnet.terraform-subnet must be replaced
-/+ resource "aws_subnet" "terraform-subnet" {
      ~ arn                             = "arn:aws:ec2:us-east-1:603991114860:subnet/sub
      ~ availability_zone               = "us-east-1d" -> "us-east-1e" # forces replacem
      ~ availability_zone_id            = "use1-az4" -> (known after apply)
      ~ cidr_block                      = "10.0.204.0/24" -> "10.0.205.0/24" # forces re
      ~ id                              = "subnet-036f7e67555980f77" -> (known after app
      + ipv6_cidr_block_association_id  = (known after apply)
      - map_customer_owned_ip_on_launch = false -> null
      ~ owner_id                        = "603991114860" -> (known after apply)
      ~ tags                            = {
          ~ "Name"      = "sub-variables-us-east-1d" -> "sub-variables-us-east-1e"
            # (1 unchanged element hidden)
        }
      ~ tags_all                        = {
          ~ "Name"      = "sub-variables-us-east-1d" -> "sub-variables-us-east-1e"
            # (1 unchanged element hidden)
        }
        # (3 unchanged attributes hidden)
    }
```





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

**Task 3.1**

Let's go ahead and apply our new configuration, which will replace the subnet with one using the CIDR block of "10.0.204.0/24". Run a `terraform apply`. Don't forget to accept the changes by typing `yes`.