Lab: Introduction to the Terraform Variables Block

As you being to write Terraform templates with a focus on reusability and DRY development (don't repeat yourself), you'll quickly being to realize that variables are going to simplify and increase usability for your Terraform configuration. Input variables allow aspects of a module or configuration to be customized without altering the module's own source code. This allows modules to be shared between different configurations.

Input variables (commonly referenced as just 'variables') are often declared in a separate file called `variables.tf`, although this is not required. Most people will consolidate variable declaration in this file for organization and simplification of management. Each variable used in a Terraform configuration must be declared before it can be used. Variables are declared in a variable block - one block for each variable. The variable block contains the variable name, most importantly, and then often includes additional information such as the type, a description, a default value, and other options.

The variable block follows the following pattern:

Template

```
variable “<VARIABLE_NAME>” {
  # Block body
  type = <VARIABLE_TYPE>
  description = <DESCRIPTION>
  default = <EXPRESSION>
  sensitive = <BOOLEAN>
  validation = <RULES>
}
```

Example

```
variable "aws_region" {
  type        = string
  description = "region used to deploy workloads"
  default     = "us-east-1"
  validation {
    condition     = can(regex("^us-", var.aws_region))
    error_message = "The aws_region value must be a valid region in the USA, starting wi
  }
}
```


The value of a Terraform variable can be set multiple ways, including setting a default value, interactively passing a value when executing a terraform plan and apply, using an environment variable, or setting the value in a `.tfvars` file. Each of these different options follows a strict order of precedence that Terraform uses to set the value of a variable.

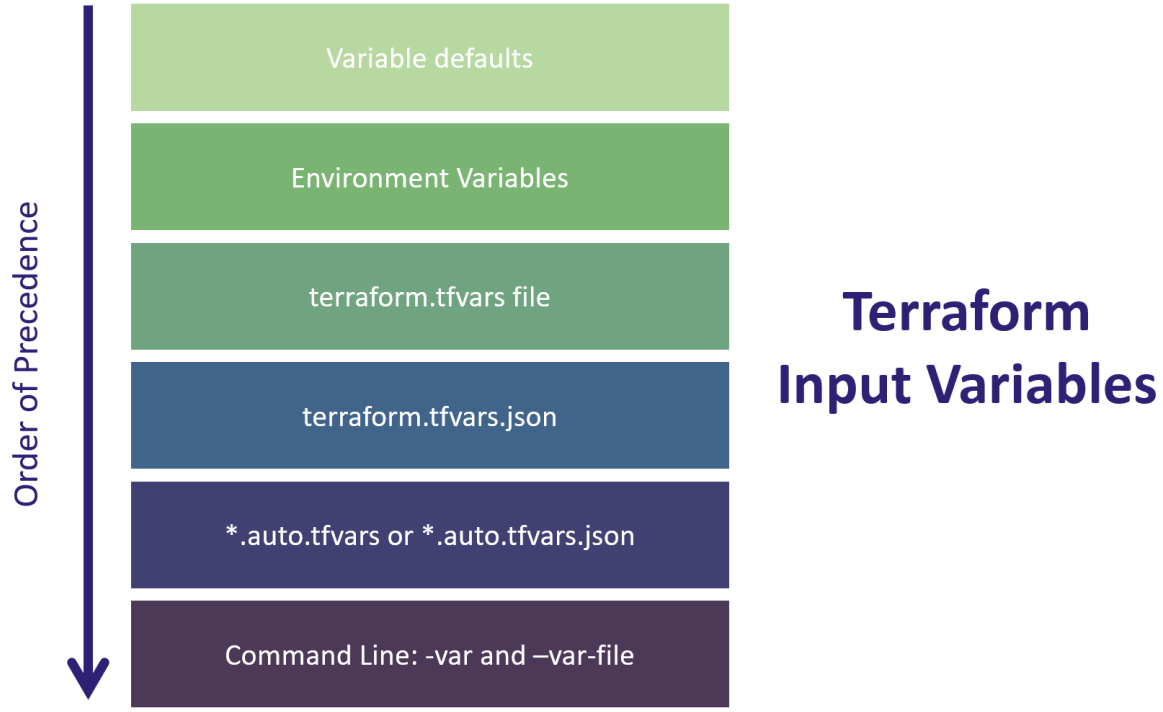

Figure 1: Terraform Input Variables - Order of Precedence

- Task 1: Add a new VPC resource block with static values
- Task 2: Define new variable blocks to declare new variables
- Task 3: Modify the value of the variable by adding defaults

Task 1: Add a new VPC resource block with static values

Before we can add any meaningful new variables, we should add a new resource to use as a demo. Let's add a new VPC resource that we'll use in this lab. In the `main.tf` file, add the following code at the bottom. ***Do not remove the existing VPC resource***

```
resource "aws_subnet" "variables-subnet" {
  vpc_id                  = aws_vpc.vpc.id
```

```
  cidr_block              = "10.0.250.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true

  tags = {
    Name      = "sub-variables-us-east-1a"
    Terraform = "true"
  }
}
```

Task 1.1

Run `terraform plan` to validate the changes to your configuration. You should see that Terraform is going to add a new VPC resource that we just added.

```
Terraform used the selected providers to generate the following execution plan. Resource
  + create

Terraform will perform the following actions:

  # aws_subnet.variables-subnet will be created
  + resource "aws_subnet" "variables-subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.250.0/24"
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + map_public_ip_on_launch                        = true
      + owner_id                                       = (known after apply)
      + tags                                           = {
          + "Name"      = "sub-variables-us-east-1a"
          + "Terraform" = "true"
        }
      + tags_all                                       = {
          + "Name"      = "sub-variables-us-east-1a"
          + "Terraform" = "true"
        }
      + vpc_id                                         = "vpc-069bc66bf87daccd6"
    }

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

Once the plan looks good, let's apply our configuration by issuing a `terraform apply`

```
$ terraform apply

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

You will be prompted to confirm the changes before they're applied. Respond with `yes`.

Task 2: Define new variable blocks to declare new variables

Now that we have a new resources to use, let's change some of those static values to use variables instead. In your `variables.tf` file, add the following variable blocks anywhere in the file:

```
variable "variables_sub_cidr" {
  description = "CIDR Block for the Variables Subnet"
  type        = string
}

variable "variables_sub_az" {
  description = "Availability Zone used Variables Subnet"
  type        = string
}

variable "variables_sub_auto_ip" {
  description = "Set Automatic IP Assigment for Variables Subnet"
  type        = bool
}
```

Task 2.1

In your `main.tf` file, let's remove our static values and replace them with our new variables to make our configuration a bit more flexible. Update the configuration file so the new VPC resource reflects the changes shown below:

```
resource "aws_subnet" "variables-subnet" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.variables_sub_cidr
  availability_zone       = var.variables_sub_az
  map_public_ip_on_launch = var.variables_sub_auto_ip

  tags = {
    Name      = "sub-variables-${var.variables_sub_az}"
```

```
    Terraform = "true"
  }
}
```

Task 2.2

We now have new variables declared in our `variables.tf` file along with a modified resource (our new subnet) to use those variables. Let's run a `terraform plan` to see what happens with our new resource and variables. Did you get something similar to what is shown below?

```
$ terraform plan
var.variables_sub_auto_ip
  Set Automatic IP Assigment for variables Subnet

  Enter a value:
```

At this point, we've declared the variables and have referenced them in our subnet resource, but Terraform has no idea what value you want to use for the new variables. Let's provide it with our values.

- for `var.variables_sub_auto_ip`, type in "true" and press enter
- for `var.variables_sub_az`, type in "us-east-1a" and press enter
- for `var.variables_sub_cidr`, type in "10.0.250.0/24" and press enter

Now that Terraform knows the values we want to use, it can proceed with our configuration. When the `terraform plan` is completed, it should have found that there are no changes needed to our infrastructure because we've defined the same values for our subnet, but using variables instead.

```
No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no
```

Task 3: Modify the value of the variable by adding defaults

Another common method of providing values for variables is by setting default values in the variable block. If no other values are provided during a `terraform plan` or `terraform apply`, then Terraform will use the default values. This is especially handy when you have variables where the value doesn't often change.

In the `variables.tf` file, modify the new variables we added earlier to now include a default argument and value.

```
variable "variables_sub_cidr" {
  description = "CIDR Block for the Variables Subnet"
  type        = string
  default     = "10.0.202.0/24"
}

variable "variables_sub_az" {
  description = "Availability Zone used for Variables Subnet"
  type        = string
  default     = "us-east-1a"
}

variable "variables_sub_auto_ip" {
  description = "Set Automatic IP Assigment for Variables Subnet"
  type        = bool
  default     = true
}
```

Task 3.1

Let's test our new defaults by running another `terraform plan`. You should see that Terraform no longer asks us for a value for the variables since it will just use the default value if you don't specify one using another method. And since we provided different values, Terraform wants to update our infrastructure.

```
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.variables-subnet must be replaced
-/+ resource "aws_subnet" "variables-subnet" {
      ~ arn                             = "arn:aws:ec2:us-east-1:603991114860:subnet/sub
      ~ availability_zone_id            = "use1-az6" -> (known after apply)
      ~ cidr_block                      = "10.0.250.0/24" -> "10.0.202.0/24" # 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 3.1

Let's go ahead and apply our new configuration, which will replace the subnet with one using the CIDR block of "10.0.202.0/24". Run a `terraform apply`. Don't forget to accept the changes by typing `yes`.