**Input Variables**

**Assigning variables**

There are multiple ways to assign variables. The order below is also the order in which variable values are chosen.

* **Command Line Flags:**

You can set variables directly on the command-line with the -var flag. Any command in Terraform that inspects the configuration accepts this flag, such as apply, plan, and refresh.

$ terraform apply -var 'region=us-east-1'

Once again, setting variables this way will not save them, and they'll have to be entered repeatedly as commands are executed.

* **From a file:**

To persist variable values, create a file and assign variables within this file. Create a file named terraform.tfvars with the following contents:

region = "us-east-1"

Terraform automatically loads all files in the current directory with the exact name of terraform.tfvars or any variation of \*.auto.tfvars. If the file is named something else, you can use the -var-file flag to specify a file name. Terraform doesn't recommend saving usernames and passwords to version control. You can create a local file with a name like secret.tfvars and use -var-file flag to load it. You can use multiple -var-file arguments in a single command, with some, checked in to version control and others not checked in.

$ terraform apply -var-file="secret.tfvars"

* **From environment variables:**

Terraform will read environment variables in the form of TF\_VAR\_name to find the value for a variable. For example, the TF\_VAR\_region variable can be set in the shell to set the region variable in Terraform.

* **UI input:**

If you execute terraform apply with any variable unspecified, Terraform will ask you to input the values interactively. These values are not saved, but this provides a convenient workflow when getting started with Terraform. UI input is not recommended for everyday use of Terraform.

* **Variable defaults:**

If no value is assigned to a variable via any of these methods and the variable has a default key in its declaration, that value will be used for the variable.

**Other Data Types**

Strings and numbers are the most commonly used variables, but lists (arrays) and maps (hashtables or dictionaries) can also be used.

* **Lists:**

Lists are defined either explicitly or implicitly.

# Declare implicitly by using brackets []

variable "cidrs" {

default = []

}

# Declare explicitly with 'list'

variable "cidrs" {

type = list

}

You can specify list values in a terraform.tfvars file.

cidrs = [ "10.0.0.0/16", "10.1.0.0/16" ]

* **Maps:**

A map is a key/value data structure that can contain other keys and values.

We've replaced our sensitive strings with variables, but we are still hard-coding AMIs. Unfortunately, AMIs are specific to the geographical region in use. One option is to ask the user to input the proper AMI for the region, but Terraform can do better than that with a map.

Maps are a way to create variables that are lookup tables. Let's extract our AMIs into a map and add support for the us-west-2 region.

variable "amis" {

type = map(string)

default = {

"us-east-1" = "ami-b374d5a5"

"us-west-2" = "ami-4b32be2b"

}

}

To use the amis map, edit aws\_instance to use var.amis keyed by var.region.

resource "aws\_instance" "example" {

ami = var.amis[var.region]

instance\_type = "t2.micro"

}

The square-bracket index notation used here is an example of how the map type expression is accessed as a variable, with [var.region] referencing the var.amis declaration for dynamic lookup.

For a static value lookup, the region could be hard-coded such as var.amis["us-west-2"].

output "ami" {

value = aws\_instance.example.ami

}

Read the selected AMI attribute from the aws\_instance resource.

Provision it by providing a region on the command line.

$ terraform apply -var region=us-west-2

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

ami = ami-def456

Formun Üstü