Terraform Best Practices — Using variables

Jack Roper
Apr 15 · 5 min read ★

When starting out with Terraform it's hard to know what is considered 'best practice' in a number of areas.

This post is the sixth in the series which focuses on point 6 in the list, 'Avoid hardcoding variables', and also shows best practice for general variable use in Terraform.

1. Use a consistent file structure across your projects
2. Use modules wherever possible
3. Use a consistent naming convention
4. Use a consistent format and style
5. Hold your state file remotely, not on your local machine
6. **Avoid hardcoding variables**
7. Less resources in a project are easier and faster to work with
8. Limit resources in the project to reduce the blast radius
9. Test your code

Variables — Best Practice

Variables are typically defined in the variables.tf file in your Terraform project. Using variables allow you to modify aspects of the module without modifying the code in the module itself.

As our goal is make our modules reusable, hardcoding variable values in those modules is a bad idea. Instead those values should be passed into the module as required.

Here's an example of a variables.tf file in the root module (root directory) of a project that is used to create a Windows desktop VM in Azure:

```
variable "subscription_id" {
 description = "Azure subscription ID"
}
variable "client_id" {
 description = "Azure subscription ID"
}
variable "client_secret" {
 description = "Azure client secret"
}
variable "tenant_id" {
 description = "Azure AD Tenant ID"
}
variable "global_settings" {
 description = "Setting read in from a global settings block"
}
variable "desktop_vm_image_publisher" {
 description = "VM Image publisher"
}
variable "desktop_vm_image_offer" {
 description = "VM Offer"
}
variable "desktop_vm_image_sku" {
 description = "VM SKU"
}
variable "desktop_vm_image_version" {
 description = "VM Image version"
}
variable "desktop_vm_size" {
 description = "VM Size"
}
```

Note that every variable has a description. You should always include one even if you think it is obvious what the variable is for.

Defaults and types can also be specified here. If they are, always put them in the same order. Description first, then type, then default. e.g.

```
variable "global_settings" {
 description = "Setting read in from a global settings block"
 type = map
 default = {}
}
```

You should always omit `type = map` if `default = {}` also exists as they are effective the same thing. The same applies to `type = list` and `default = []` .

Note that the variable name is plural 'global_settings'. Always use a plural name when defining a variable of a type map or list, as many values will potentially be read in.

A file called variables.auto.tfvars in the root module would then define the actual values, (the auto part of the filename means this file is read into the configuration automatically). e.g.

```
subscription_id = "6840913c-76e6-488d-xxxx-0a27872c70e6"
client_id       = "c0bcbf81-c51b-4ca2-xxxx-759c688e2d9f"
client_secret   = "zNGdvqm7Ft.xxxxxxx"
tenant_id       = "5759ecf2-97b4-4017-xxxx-4f0b25f016d2"

global_settings = {

  #Set of tags
  tags = {
    applicationName = "Windows VM Demo"
    businessUnit    = "Technical Solutions"
    costCenter      = "MPN Sponsorship"
    DR              = "NON-DR-ENABLED"
    deploymentType  = "Terraform"
    environment     = "Dev"
    owner           = "Jack Roper"
    version         = "0.1"
  }

}

# Desktop VM variables
desktop_vm_image_publisher   = "MicrosoftWindowsDesktop"
desktop_vm_image_offer       = "Windows-10"
desktop_vm_image_sku         = "20h1-pro"
desktop_vm_image_version     = "latest"
desktop_vm_size              = "Standard_B1s"
desktop_vm_static_ip_address = "10.0.1.5"
```

```
desktop_vm_name              = "demovmname"
desktop_vm_location          = "uksouth"
desktop_vm_short_location    = "uks"
desktop_vm_activity_tag      = "Windows Desktop"
```

A file in the root module would then call the child module to create the VM as the source, passing in the variables.

e.g. vm.tf

```
module windows_desktop_vm_using_local_module {
  source              = "./vm"
  resource_group_name = azurerm_resource_group.rg.name
  location            = var.desktop_vm_location
  sloc                = var.desktop_vm_short_location
  vm_subnet_id        = module.network.vnet_subnets[0]
  vm_name             = var.desktop_vm_name
  vm_size             = var.desktop_vm_size
  publisher           = var.desktop_vm_image_publisher
  offer               = var.desktop_vm_image_offer
  sku                 = var.desktop_vm_image_sku
  static_ip_address   = var.desktop_vm_static_ip_address
  activity_tag        = var.desktop_vm_activity_tag
  admin_password      = module.vmpassword.secretvalue
}
```

Note the variables here all reference var.*, and are not hardcoded values, e.g.

```
location            = "uksouth"
```

The exceptions here are the resource group which references the code block that creates the resource group to get the name output, the VM subnet id which is referenced from the output of another module, and the admin password which again is referenced from another module that pulls the password from keyvault.

Lastly the files in the vm module itself, firstly the variables.tf file:

```
variable "resource_group_name" {
}

variable "location" {
}
```

```
variable "sloc" {
}

variable "vm_size" {
}

variable "vm_subnet_id" {
}

variable "vm_name" {
}

variable "vm_os_disk_delete_flag" {
  default = true
}

variable "vm_data_disk_delete_flag" {
  default = true
}

variable "network_security_group_id" {
  default = ""
}

variable "static_ip_address" {
}

variable "publisher" {
}

variable "offer" {
}

variable "sku" {
}

variable "tags" {
  type        = map
  description = "All mandatory tags to use on all assets"

  default = {
    activityName       = "AzureVMWindowsDemo"
    automation         = "Terraform"
    costCenter1        = "A00000"
    dataClassification = "Demo"
    managedBy          = "jackwesleyroper"
    solutionOwner      = "jackwesleyroper"
  }
}

variable "activity_tag" {
}

variable "admin_password" {
}
```

Again note that defaults can be set for variables, these are taken if no value is passed in. In this case, I always want the following to apply as default unless a value is specified:

```
variable "vm_os_disk_delete_flag" {
  default = true
}

variable "vm_data_disk_delete_flag" {
  default = true
}
```

These variables are also missing their descriptions! Ideally, descriptions should match the Terraform doc description listed under the argument reference for the particular resource, in this case, azurerm_windows_virtual_machine. You'll see that the SKU description under source image reference is described as the following:

```
variable "sku" {
  description = "Specifies the SKU of the image used to create the
virtual machines."
}
```

These variables are then used by the module code in main.tf

```
resource "random_string" "nic_prefix" {
  length  = 4
  special = false
}

resource "azurerm_network_interface" "vm_nic" {
  name                = "${var.vm_name}-nic1"
  location            = var.location
  resource_group_name = var.resource_group_name

  ip_configuration {
    name                          =
"${var.vm_name}_nic_${random_string.nic_prefix.result}"
    subnet_id                     = var.vm_subnet_id
    private_ip_address_allocation = "Static"
    private_ip_address            = var.static_ip_address
  }
  tags = var.tags
}

resource "azurerm_network_interface_security_group_association"
"vm_nic_sg" {
  network_interface_id      = azurerm_network_interface.vm_nic.id
  network_security_group_id = var.network_security_group_id
  count                     = var.network_security_group_id == "" ?
0 : 1
}

resource "azurerm_virtual_machine" "windows_vm" {
  name                = var.vm_name
  vm_size             = var.vm_size
  location            = var.location
  resource_group_name = var.resource_group_name

  tags = merge(var.tags, { activityName = "${var.activity_tag} " })

  network_interface_ids = [
    "${azurerm_network_interface.vm_nic.id}",
  ]

  storage_image_reference {
    publisher = var.publisher
    offer     = var.offer
    sku       = var.sku
    version   = "latest"
  }

  identity {
    type = "SystemAssigned"
  }

  storage_os_disk {
    name              = "${var.vm_name}-os-disk"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }

  os_profile {
    admin_password = var.admin_password
    admin_username = "azureuser"
    computer_name  = var.vm_name
  }

  os_profile_windows_config {
    provision_vm_agent = true
  }

  delete_os_disk_on_termination    = var.vm_os_disk_delete_flag
  delete_data_disks_on_termination = var.vm_data_disk_delete_flag
}
```

You'll notice that this module could be further improved by adding the admin username, and storage options as variables. Hardcoding those options here is **bad**!

Finally, output.tf outputs the variables back to the root module:

```
output "vm_id" {
  value = "${azurerm_virtual_machine.windows_vm.id}"
}

output "vm_name" {
  value = "${azurerm_virtual_machine.windows_vm.name}"
}

output "vm_location" {
  value = "${azurerm_virtual_machine.windows_vm.location}"
}

output "vm_resource_group_name" {
  value =
"${azurerm_virtual_machine.windows_vm.resource_group_name}"
}
```

I clearly have some work to do to update my code above to follow my recommended best practices!

For more information check out the Terraform Docs!

Thanks for reading! 🍻

Input Variables - Configuration Language - Terraform by HashiCorp

Hands-on: Try the Customize Terraform Configuration with Variables tutorial on HashiCorp Learn. Input variables serve…

www.terraform.io

My colleague Jonnychipz also has a video on Variables:

[(113) Deploying Azure with Terraform — 3 — Using Variables — YouTube]

Terraform   Azure   DevOps   Infrastructure As Code   Terraform Modules