Terraform Identifiers

Argument names, block type names, and the names of most Terraform-specific constructs like resources, input variables, etc. are all identifiers.

Identifiers can contain **letters, digits**, **underscores (\_),** and **hyphens (-).** The first character of an identifier must not be a digit, to avoid ambiguity with literal numbers.

Comments

The Terraform language supports three different syntaxes for comments:

* [#](https://developer.hashicorp.com/terraform/language/syntax/configuration) begins a single-line comment, ending at the end of the line.
* // also begins a single-line comment, as an alternative to #.
* /\* and \*/ are start and end delimiters for a comment that might span over multiple lines.

**Variables:**

The Terraform language includes a few kinds of blocks for requesting or publishing named values.

* **Local Values** are a convenience feature for assigning a short name to an expression.
* **Input** **Variables** serve as parameters for a Terraform module, so users can customize behavior without editing the source.
* **Output** **Values** are like return values for a Terraform module.

Local Variable

* A local value assigns a name to an expression, so you can use the name multiple times within a module instead of repeating the expression.
* Local values are like a function's temporary local variables.
* Once a local value is declared, you can reference it in expressions as **local.<NAME>.**
* A set of related local values can be declared together in a single locals block:
* The expressions in local values are not limited to literal constants; they can also reference other values in the module in order to transform or combine them, including variables, resource attributes, or other local values

**locals {**

  service\_name   = "forum"

  owner         = "Community Team"

}

**locals {**

  # Common tags to be assigned to all resources

  common\_tags = {

    Service   = local.service\_name

    Owner     = local.owner

  }

}

Input Variables

* This functionality allows you to share modules across different Terraform configurations, making your module composable and reusable.
* When you declare variables in the root module of your configuration, you can set their values using CLI options and environment variables. When you declare them in **child modules**, the calling module should pass values in the **module** block.
* The name of a variable can be any valid **identifier** **except** the following: **source, version, providers, count, for\_each, lifecycle, depends\_on, locals.**

**Examples of Variable Declarations**

**variable** "resource\_group\_name" {

type = string

nullable = false

default = "Terraform-rg"

}

**variable** "resource\_group\_location" {

type = string

default = "eastus"

}

**#Creating a Resource Group**

resource "azurerm\_resource\_group" "rg" {

name = **var.resource\_group\_name**

location = **var.resource\_group\_location**

}

**Using Input Variable Values:**

Within the module that declared a variable, its value can be accessed from within expressions as **var.<NAME>.**

**Terraform Commands**

* terraform validate
* terraform plan -out=tfplan

Assigning Values to Variables

When variables are declared in the root module of your configuration, they can be set in several ways:

**Option1:** Individually, with the -var command line option.

terraform apply **-var**='resource\_group\_location=westus' -var="resource\_group\_name=Demo\_rg"

**Option2: In variable definition files.**

In variable definitions (.tfvars) files, either specified on the command line or automatically loaded.

**dev.tfvars file**

resource\_group\_location=westus

resource\_group\_name=Demo\_rg

**Command**

terraform apply **-var-file**="dev.tfvars"

Note: Terraform also automatically loads a number of variable definitions files if they are present:

* Files named exactly **terraform.tfvars**
* Any files with names ending in **.auto.tfvars**

**Option3: As environment variables.**

As a fallback for the other ways of defining variables, Terraform searches the environment of its own process for environment variables named TF\_VAR\_ followed by the name of a declared variable.

**Windows**

$Env:TF\_VAR\_resource\_group\_name= "Demo\_rg"

$Env:TF\_VAR\_resource\_group\_location = 'westus'

**Linux**

export TF\_VAR\_ resource\_group\_name ="Demo-rg"

export TF\_VAR\_resource\_group\_location = 'westus'

**Precedence**, with later sources taking precedence over earlier ones:

1. Environment variables
2. The terraform.tfvars file, if present.
3. The terraform.tfvars.json file, if present.
4. Any \*.auto.tfvars or \*.auto.tfvars.json files, processed in lexical order of their filenames.
5. Any -var and -var-file options on the command line, in the order they are provided.

Output Variables

Output values make information about your infrastructure available on the command line, and can expose information for other Terraform configurations to use.

Output values are similar to return values in programming languages.

Output values have several uses:

* A root module can use outputs to print certain values in the CLI output after running terraform apply.
* A child module can use outputs to expose a subset of its resource attributes to a parent module.

**Declaring an Output Value**

output azureVM\_public\_ip {

value = azurerm\_public\_ip.myVMPublicIP.ip\_address

depends\_on = [

azurerm\_linux\_virtual\_machine.my\_terraform\_vm

]

}

**Variable All in One Example**

variable "N1" {

  default     = 100

  type        = number

  description = "First Number"

}

variable "N2" {

  default     = 100

  type        = number

  description = "Second Number"

}

locals {

  Sum = var.N1 + var.N2

}

output "Sum" {

  value = local.Sum

}