

## **Chapter 2 - Terraform Tips**

# Objective

- Learn basic Terraform techniques
- Interpolations in Terraform
- What's new in Terraform v0.12

# Interpolations

## Using string variables

```
variable "example_string" {  
  type      = "string" // Optional, default to string  
  default   = "example" // Optional, default to ""  
}
```

```
${var.foo} // example
```

# Using map variables

```
variable "example_map" {  
  type = "map"  
  
  // Optional, default to {}  
  default = {  
    key1 = "value1"  
    key2 = "value2"  
  }  
}
```

```
// Get the whole map  
${var.example_map}  
  
// get the value of "key1" within the map  
${var.example_map["key1"]}
```

## Using list variables

```
variable "example_list" {  
  type      = "list"  
  default   = ["v1", "v2"] // Optional, default to []  
}
```

```
// Get the whole list  
${var.example_list}
```

```
// Return the ith element of the list  
${var.example_list[i]}
```

## Using outputs

```
output "foo" {  
  value = "foo"  
}
```

## Using outputs from a module

```
module "foo" {  
  source = "example-module"  
}
```

```
// Get the output from module  
${module.foo.example_output}
```



# Practice

Try [ch02/200-variables-module-output](#)

```
$ cd aws/ch02/practices/200-variables-module-output  
$ terraform init  
$ terraform apply
```

And see what happened.

## Count information

```
data "template_file" "foo" {  
  // interpolate the current index in a multi-count  
  // resource  
  template = "foo-${count.index}"  
  
  count = 3  
}
```

```
// ["foo-0", "foo-1", "foo-2"]  
${data.template_file.foo.*.rendered}
```

# Using attributes of resources

```
resource "example_resource" "foo" {  
  name = "foo"  
}
```

```
// Returns "foo"  
${example_resource.foo.name}
```

```
resource "example_resource" "foo" {  
  name = "foo-${count.index}"  
  
  count = 3  
}
```

```
// Returns ["foo-0", "foo-1", "foo-2"]  
${example_resource.foo.*.name}
```

# Using attributes of data resources

```
data "example_resource" "foo" {  
  name = "foo"  
}
```

```
// Returns "foo"  
${data.example_resource.foo.name}
```

```
data "example_resource" "foo" {  
  name = "foo-${count.index}"  
  
  count = 3  
}
```

```
// Returns ["foo-0", "foo-1", "foo-2"]  
${data.example_resource.foo.*.name}
```

## Path information

```
resource "local_file" "foo" {  
  content = "foo"  
  
  // The file foo.txt will be located at module's path  
  filename = "${path.module}/foo.txt"  
}
```

# Practice

Try [ch02/201-count-attributes-path](#)

```
$ cd aws/ch02/practices/201-count-attributes-path  
$ terraform init  
$ terraform apply
```

And see what happened.

# Built-in Functions

List of builtin functions

# Built-in Functions

## concat

```
// Combines two or more lists into a single list  
list3 = concat(list1, list2, ...)
```

## element

```
// Returns a single element from a list at the given  
// index.  
//  
// If the index is greater than the number of elements,  
// this function will wrap using a standard mod  
// algorithm.  
//  
// This function only works on flat lists  
item = element(list, index)
```



# Built-in Functions

## join

```
// Joins the list with the delimiter for a resultant  
// string.  
//  
// This function works only on flat lists.  
str = join(delim, list)
```

## file

```
// Reads the contents of a file into the string.  
// Variables in this file are not interpolated.  
ssh_key = file("/home/foo/.ssh/id_rsa.pub")
```

# Built-in Functions

## length

*// Returns the number of members in a given list.*

```
l1 = length(list)
```

*// Returns the number of members in a given map*

```
l2 = length(map)
```

*// Returns the number of characters in a given string.*

```
l3 = length(string)
```

# Built-in Functions

## lookup

```
// Performs a dynamic lookup into a map variable.  
//  
// Lookup "foo" from the map. Interpolation will failed  
// if "foo" doesn't exist.  
v = lookup(map, "foo")  
  
//  
// Lookup "foo" from the map. "bar" is returned if  
// "foo" doesn't exist.  
v = lookup(map, "foo", "bar")
```

# Practice

Try [ch02/202-builtin-functions-demo](#)

```
$ cd aws/ch02/practices/202-builtin-functions  
$ terraform init  
$ terraform apply
```

And see what happened.

# Conditionals

# Conditionals

```
resource "example_resource" "foo" {  
  name = "${var.env == "prod" ? "foo-prod" : "foo-dev"}"  
}
```

# New Features in Terraform v0.12

v0.12 is the next major release

New features:

- For Expressions
- Conditional Improvements
- Rich Value Types
- ...

# For Expressions in Terraform v0.12

Before v0.12

```
resource "local_file" "foo" {
  filename = "${path.module}/foo-${count.index}.txt"
  content = "foo-${count.index}"
  count = 3
}

data "template_file" "do_hash" {
  template = "${sha1(
    element(local_file.foo.*.filename, count.index)
  )}"
  count = "${length(local_file.foo.*.filename)}"
}

resource "local_file" "foo_hash" {
  filename = "${path.module}/foo-hash.txt"
  content = "${join(",",
    data.template_file.do_hash.*.rendered
  )}"
}
```



# For Expressions in Terraform v0.12

After v0.12

```
resource "local_file" "foo" {
  filename = "${path.module}/foo-${count.index}.txt"
  content  = "foo-${count.index}"
  count    = 3
}

resource "local_file" "foo_hash" {
  filename = "${path.module}/foo-hash.txt"
  content  = "${join(", ",
    for f in local_file.foo.*.filename:
      sha1(f)
  )}"
}
```

## Practice

Try [ch02/203-complex-computations](#)

```
$ cd aws/ch02/practices/203-complex-computations  
$ terraform init  
$ terraform apply
```

Try to refactor it using `for` after Terraform v0.12 released.

# Conditional Improvements in Terraform v0.12

Before v0.12

```
data "template_file" "dev" {  
  template = "dev"  
  count = "${var.env == "dev" ? 1 : 0}"  
}  
  
data "template_file" "prod" {  
  template = "prod"  
  count = "${var.env == "dev" ? 0 : 1}"  
}  
  
output "failure_cannot_be_resolved" {  
  value = "${var.env == "dev" ?  
    data.template_file.dev.rendered :  
    data.template_file.prod.rendered  
  }"  
}
```

Both variables are resolved no matter what `var.env` is.

# Conditional Improvements in Terraform v0.12

Before v0.12

We can hack it by using `join`.

```
data "template_file" "dev" {
  template = "dev"
  count = "${var.env == "dev" ? 1 : 0}"
}

data "template_file" "prod" {
  template = "prod"
  count = "${var.env == "dev" ? 0 : 1}"
}

output "success" {
  value = "${var.env == "dev" ?
    join("", data.template_file.dev.*.rendered) :
    join("", data.template_file.prod.*.rendered)
  }"
}
```

# Conditional Improvements in Terraform v0.12

After Terraform v0.12

```
data "template_file" "dev" {
  template = "dev"
  count = "${var.env == "dev" ? 1 : 0}"
}

data "template_file" "prod" {
  template = "prod"
  count = "${var.env == "dev" ? 0 : 1}"
}

output "success" {
  value = "${var.env == "dev" ?
    data.template_file.dev.rendered :
    data.template_file.prod.rendered
  }"
}
```

# Practice

Try [ch02/204-err-conditional](#)

## **Rich Value Types**

# Rich Value Types

Before v0.12

```
module "example" {  
  source = "../example-module"  
  value = {  
    key = "value"  
  
    my_map = {  
      key = "value"  
    }  
  
    my_list = ["v1", "v2"]  
  }  
}
```



# Rich Value Types

After v0.12

```
module "subnets" {  
  source = "./subnets"  
  
  parent_vpc_id = "vpc-abcd1234"  
  networks = {  
    production_a = {  
      network_number      = 1  
      availability_zone    = "us-east-1a"  
    }  
    production_b = {  
      network_number      = 2  
      availability_zone    = "us-east-1b"  
    }  
    staging_a = {  
      network_number      = 1  
      availability_zone    = "us-east-1a"  
    }  
  }  
}
```

# Rich Value Types

Resources and Modules as Values after v0.12

```
output "example_module" {  
  value = ${module.example}  
}
```

Return the `example` module as an object.

# Key Takeaways

- Use `${...}` to access variables
- Built-in functions are useful
- Shortages of current Terraform
  - Weak type system
  - Not intuitive for some cases (e.g., count)
- Big change in Terraform v0.12