Chapter 3 - Terraform Tips

Interpolation in Terraform

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]}
```

Practice

Try ch03/300-variables-demo

Using outputs from a module

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

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

Practice

Try ch03/301-module-output-demo

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}
```

Practice

Try ch03/302-count-demo

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 ch03/303-attributes-path-demo

Built-in Functions

List of builin 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 ch03/304-builtin-functions-demo

Conditionals

Conditionals

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

The supported operators are:

- Equality: == and !=
- Numerical comparison: >, <, >=, <=
- Boolean logic: &&, ||, unary !

New Features in Terraform v0.12

- 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 = "${sha1(element(local_file.foo.*.filename, c
  count = "${length(local_file.foo.*.filename)}"
}

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

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)
  )}"
}
```

Demo

Try ch03/305-complex-computations

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_used_with_list" {
  value = "${var.env == "dev" ? data.template_file.dev.*.
}

output "failure_cannot_be_resolved" {
  value = "${var.env == "dev" ? data.template_file.dev.re
}
```

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_fi
}
```

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.re
}

output "success_for_list" {
  value = "${var.env == "dev" ? data.template_file.dev.*.
}
```

Demo

Try ch03/306-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

Before v0.12

```
lookup(var.value, "my_map")   // error
lookup(var.value, "my_list")  // error
```

Rich Value Types

Accessing attributes inside nested object after v0.12 (perhaps?)

```
${var.value.my_map}   // returns my_map
${var.value.my_list}  // returns my_list
```

Rich Value Types

Resources and Modules as Values after v0.12

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

Return the `example` module as an object.