ENFORCING TAGGING STANDARDS

It's typically a good idea to come up with a tagging standard for your team and create Terraform modules that enforce this standard as code. One way to do this is to manually ensure that every resource in every module sets the proper tags, but with many resources, this is tedious and error prone. If there are tags that you want to apply to *all* of your AWS resources, a more reliable approach is to add the `default_tags` block to the `aws` provider in every one of your modules:

```
provider "aws" {
  region = "us-east-2"

  # Tags to apply to all AWS resources by default
  default_tags {
    tags = {
      Owner     = "team-foo"
      ManagedBy = "Terraform"
    }
  }
}
```

The preceding code will ensure that every single AWS resource you create in this module will include the `Owner` and `ManagedBy` tags (the only exceptions are resources that don't support tags and the `aws_autoscaling_group` resource, which does support tags but doesn't work with `default_tags`, which is why you had to do all that work in the previous section to set tags in the `webserver-cluster` module). `default_tags` gives you a way to ensure all resources have a common baseline of tags while still allowing you to override those tags on a resource-by-resource basis. In Chapter 9, you'll see how to define and enforce policies as code such as "all resources must have a `ManagedBy` tag" using tools such as OPA.

Loops with for Expressions

You've now seen how to use loops to create multiple copies of entire resources and inline blocks, but what if you need a loop to set a single variable or parameter?

Imagine that you wrote some Terraform code that took in a list of names:

```
variable "names" {
  description = "A list of names"
  type        = list(string)
  default     = ["neo", "trinity", "morpheus"]
}
```

How could you convert all of these names to uppercase? In a general-purpose programming language such as Python, you could write the following for-loop:

```
names = ["neo", "trinity", "morpheus"]

upper_case_names = []
for name in names:
    upper_case_names.append(name.upper())

print upper_case_names

# Prints out: ['NEO', 'TRINITY', 'MORPHEUS']
```

Python offers another way to write the exact same code in one line using a syntax known as a *list comprehension*:

```
names = ["neo", "trinity", "morpheus"]
upper_case_names = [name.upper() for name in names]
print upper_case_names

# Prints out: ['NEO', 'TRINITY', 'MORPHEUS']
```

Python also allows you to filter the resulting list by specifying a condition:

```
names = ["neo", "trinity", "morpheus"]
short_upper_case_names = [name.upper() for name in names if
len(name) < 5]
print short_upper_case_names
```

```
# Prints out: ['NEO']
```

Terraform offers similar functionality in the form of a *for* expression (not to be confused with the `for_each` expression you saw in the previous section). The basic syntax of a `for` expression is as follows:

```
[for <ITEM> in <LIST> : <OUTPUT>]
```

where `LIST` is a list to loop over, `ITEM` is the local variable name to assign to each item in `LIST`, and `OUTPUT` is an expression that transforms `ITEM` in some way. For example, here is the Terraform code to convert the list of names in `var.names` to uppercase:

```
output "upper_names" {
  value = [for name in var.names : upper(name)]
}
```

If you run `terraform apply` on this code, you get the following output:

```
$ terraform apply

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

Outputs:

upper_names = [
  "NEO",
  "TRINITY",
  "MORPHEUS",
]
```

Just as with Python's list comprehensions, you can filter the resulting list by specifying a condition:

```
output "short_upper_names" {
  value = [for name in var.names : upper(name) if length(name) <
5]
}
```

Running `terraform apply` on this code gives you this:

```
short_upper_names = [
  "NEO",
]
```

Terraform's `for` expression also allows you to loop over a map using the following syntax:

```
[for <KEY>, <VALUE> in <MAP> : <OUTPUT>]
```

Here, `MAP` is a map to loop over, `KEY` and `VALUE` are the local variable names to assign to each key-value pair in `MAP`, and `OUTPUT` is an expression that transforms `KEY` and `VALUE` in some way. Here's an example:

```
variable "hero_thousand_faces" {
  description = "map"
  type        = map(string)
  default     = {
    neo      = "hero"
    trinity  = "love interest"
    morpheus = "mentor"
  }
}

output "bios" {
  value = [for name, role in var.hero_thousand_faces : "${name}
is the ${role}"]
}
```

When you run `terraform apply` on this code, you get the following:

```
bios = [
  "morpheus is the mentor",
  "neo is the hero",
  "trinity is the love interest",
]
```