terraform count, for_each, and for loop

Table of Content

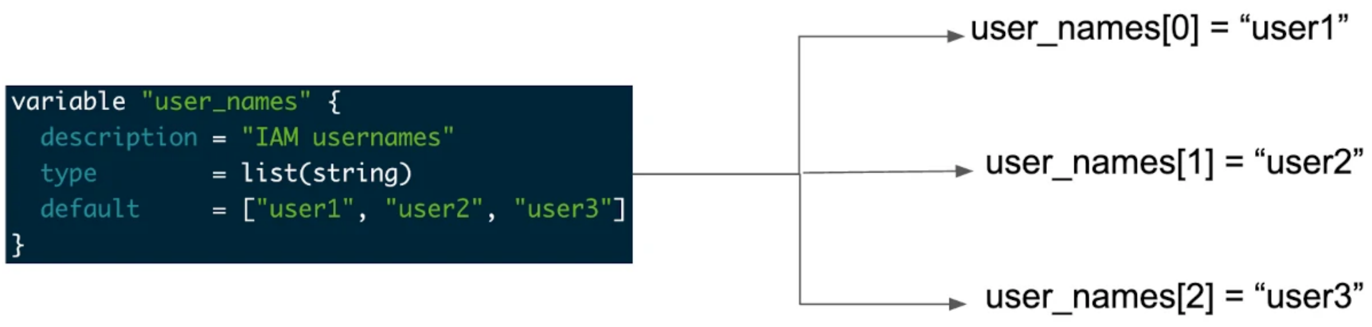
1. ***Loops with count***

As the name suggests we need to use `count` but to use the `count` first we need to declare collections inside our terraform file.

Let's create a collection variable of type `list(string)` -

```
variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

Here is the pictorial representation of the above `list variable` -



```
variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

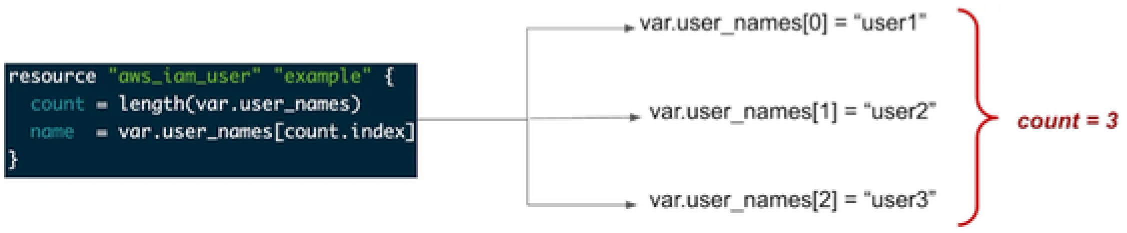
Terraform loop and for_each loop

In the above collection, we have created a `list` of type `string` that contains usernames and these usernames we are going to use for creating `aws_iam_user`.

The code snippet shows how we are going to iterate over the `list(string)` -

```
resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}
```

terraform loop and for_each loop


```
resource "aws_iam_user" "example" {
  count  = length(var.user_names)
  name   = var.user_names[count.index]
}
```
var.user_names[0] = "user1"
var.user_names[1] = "user2"
var.user_names[2] = "user3"
count = 3

Here is the complete `terraform` file -

```
provider "aws" {
   region     = "eu-central-1"
   access_key = "AKIATQ37NXB2OBQHAALW"
   secret_key = "ilKygurap8zSErv7jySTDi2796WGqMkEtN6txxxx"
}
resource "aws_instance" "ec2_example" {

   ami           = "ami-0767046d1677be5a0"
   instance_type =  "t2.micro"
   count = 1

   tags = {
           Name = "Terraform EC2"
   }

}

resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}

variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

Once you apply this terraform configuration using the `terraform apply` command, it will do the following on `AWS`-

1. Create one `ec2` instance

   Create three `IAM` users - `user1`, `user2`, `user3`

2. *Loops with for_each*

The `for_each` is a little special in terraforming and you can not use it on any collection variable.

*Note: - **It can only be used on*** `set(string)` or `map(string)`.

The reason why `for_each` does not work on a `list(string)` is because a list can contain duplicate values but if you are using `set(string)` or `map(string)` then it does not support duplicate values.

Let's first create a `set(string)` variable -

```
variable "user_names" {
  description = "IAM usernames"
  type        = set(string)
  default     = ["user1", "user2", "user3s"]
}
```

Now let's iterate over the variable `user_names`.

```
resource "aws_iam_user" "example" {
  for_each = var.user_names
  name  = each.value
}
```

BASH

Here is the complete terraform file with implementation of `for_each`

```
provider "aws" {
   region     = "eu-central-1"
   access_key = "AKIATQ37NXB2NN3D4ARS"
   secret_key = "3v9mlwZQvmccL3ou1dxiDeEf1bWaG3kccpVlXXXX"
}
resource "aws_instance" "ec2_example" {

   ami           = "ami-0767046d1677be5a0"
   instance_type =  "t2.micro"
   count = 1

   tags = {
           Name = "Terraform EC2"
   }

}

resource "aws_iam_user" "example" {
  for_each = var.user_names
  name  = each.value
}

variable "user_names" {
  description = "IAM usernames"
  type        = set(string)
  default     = ["user1", "user2", "user3"]
}
```

You can apply the above terraform configuration by running the command `terraform apply`.

Here is the output which you will notice after running the command -

```
# aws_iam_user.example["user1"] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user1"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }

  # aws_iam_user.example["user2"] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user2"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }

  # aws_iam_user.example["user3s"] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user3s"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }
```

3. ***for loop***

The `for` loop is pretty simple and if you have used any programming language before then I guess you will be pretty much familiar with the `for` loop.

Only the difference you will notice over here is the syntax in Terraform.

I am going to take the same example by declaring a `list(string)` and adding three users to it - `user1`, `user2`, `user3`

```
variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

You can use the above-declared variable inside your terraform file in a very simple way -

```
output "print_the_names" {
  value = [for name in var.user_names : name]
}
```

You apply the above terraform configuration by running the command `terraform apply`. And if you see the logs where it will show how many users it is going to create -

```
# aws_iam_user.example["user1"] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user1"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }

  # aws_iam_user.example["user2"] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user2"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }

  # aws_iam_user.example["user3s"] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "user3s"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }
```

How to iterate over MAP?

We can use a similar approach to iterate over the `map` also. But always keep in mind you need to specify the type of the `map` like `string` or `number`.

Here is the same example which I have taken but modified a bit for `map` -

```
variable "iam_users" {
  description = "map"
  type        = map(string)
  default     = {
    user1      = "normal user"
    user2  = "admin user"
    user3 = "root user"
  }
}
```

Now let’s iterate over the `map`

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

Here is the difference between *list* and *map* syntax

For list -

```
{for <ITEM> in <LIST> : <OUTPUT_KEY> => <OUTPUT_VALUE>}
```

For Map -

```
{for <KEY>, <VALUE> in <MAP> : <OUTPUT_KEY> => <OUTPUT_VALUE>}
```