Understanding terraform count, for\_each and for loop?

ami-01a4f99c4ac11b03c { Mumbai region amazon-linux ami }

When you are working with collection variables inside [Terrafrom](https://www.terraform.io/" \t "_blank) then you must understand the concept of loops with count, loops with for each and `for loop, otherwise, it will be really hard to iterate over collections such list, map and set.

This article is particularly focused on understanding how the loops work. We will see the example of each concept in more details -

Table of Content

1. [***Loops with count***](https://jhooq.com/terraform-for-and-for-each-loop/#1-_loops-with-count_)
2. [***Loops with for\_each***](https://jhooq.com/terraform-for-and-for-each-loop/#2-_loops-with-for_each_)
3. [***for loop***](https://jhooq.com/terraform-for-and-for-each-loop/#3-_for-loop_)

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**-**

terraform loop and for\_each loop

In the above collection, we have created a list of type string which 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

Here is the complete terraform file -

provider "aws" {

region = "eu-central-1"

access\_key = "YOUR ACCESS KEY"

secret\_key = "YOUR SECRET ACCESS KEY"

}

resource "aws\_instance" "ec2\_example" {

ami = "AMI-OF YOUR REGION"

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
2. Create three IAM users - user1, user2, user3

1. ***Loops with for\_each***

The for\_each meta-argument accepts a map or a set of strings, and creates an instance for each item in that map or set. Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

The for\_each is a little special in terraforming and you cannot 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 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

}

Here is the complete terraform file with implementation of for\_each

provider "aws" {

region = "eu-central-1"

access\_key = "YOUR ACCESS KEY"

secret\_key = "YOUR SECRET ACCESS KEY"

}

resource "aws\_instance" "ec2\_example" {

ami = "AMI OF YOUR REGION"

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 = "user3"

+ 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 = "user3"

+ 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>}