Loop|Datasource

Loops

What is looping in Terraform?

Loops are commonly used in the Terraform community to make modules dynamic. For example, a Virtual Machine module can use loops to deploy multiple disks if needed or contain multiple IP addresses.

Three types of loop, {count, for_each, for}

- count parameter:- loop over resources.
- for_each expressions:- loop over resources and inline blocks within a resource.
- for expressions:- loop over lists and maps.

Step 1:- Example:- without count and with count

*First create a directory.

If you want to create a multiple instance or resource on infra, it become long entries on terraform file, as well chances for duplication, so to avoid that count will be used.

To avoid repeated resource blocks, count will be very useful.

#mkdir -p loop/{count,for_each}

#vim providers.tf

```
[root@porali count]# cat providers.tf
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>3.0"
    }
  }
}

#configure the AWS provider
provider "aws" {
  region  = "us-east-2"
  profile = "dev"
}
[root@porali count]#
```

#vim resource.tf

```
[root@porali demo1]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  key_name               = "drhiju86"
  vpc_security_group_ids = ["sg-b844c2cd"]
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = "robo"
  }
}


resource "aws_instance" "web1" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  key_name               = "drhiju86"
  vpc_security_group_ids = ["sg-b844c2cd"]
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = "robo"
  }
}
```

*Note:- you can see multiple resource blocks for EC2 instances, which doesn't make sense when it's in real time.

@Example:- with count

#vim resource.tf

```
[root@porali demo1]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  key_name               = "drhiju86"
  vpc_security_group_ids = ["sg-b844c2cd"]
  count = 2
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = "robo"
  }
}
[root@porali demo1]#
```

#terraform init

#terraform validate

#terraform plan

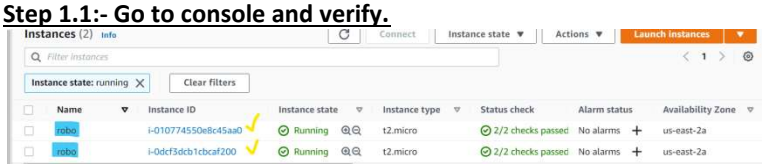
```
Terraform will perform the following actions:

  # aws_instance.web[0] will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-00f8e2c955f7ffa9b"
  # aws_instance.web[1] will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-00f8e2c955f7ffa9b"
```

#terraform apply

#terraform state list

```
[root@porali demo1]# terraform state list
aws_instance.web[0]
aws_instance.web[1]
[root@porali demo1]#
```

Step 1.1:- Go to console and verify.



*Note: - 1, “robo” is the reference for human, for AWS uniq instance ID is the reference.

2, aws_instance.web[0] or [1] is the index value, which is created in for loop method.

Step 1.2:- Example:- count with index value.

#vim providers.tf

#vim resource.tf

```
[root@porali count]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  vpc_security_group_ids = ["sg-b844c2cd"]
  key_name               = "drhiju86"
  count = 2
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = "robo.${count.index}"
  }
}
[root@porali count]#
```

#terraform validate

#terraform plan

```
+ tags                                   = {
    + "Name" = "robo.0"
  }
+ tags                                   = {
    + "Name" = "robo.1"
  }
```

#terraform apply

*Important Note:- these count method cannot be used for IAM user creation, since for instance creation ok to use.

Step 1.3:- Example with variable

#vim variable.tf

```
[root@porali demo2]# cat variable.tf
variable "name" {
  type = list
  default = ["robo", "porali", "I"]
}
[root@porali demo2]#
```

#vim resource.tf

```
[root@porali demo2]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  key_name               = "drhiju86"
  vpc_security_group_ids = ["sg-b844c2cd"]
  count = length(var.name)
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = var.name[count.index]
  }
}
[root@porali demo2]#
```

#terraform init

#terraform plan

```
+ tags                         = {
    + "Name" = "robo"
  }
+ tags                         = {
    + "Name" = "porali"
  }
+ tags                         = {
    + "Name" = "I"
  }
```

#terraform apply

*Important note :- By using function **length** and data type **list** with count on variable file, can able to create instance with mentioned names.

Step 1.4:- Example added name in variable list.

#vim variable.tf

```
[root@porali demo3]# cat variable.tf
variable "name" {
  type = list
  default = ["robo", "porali", "I", "nadodigal"]
}
[root@porali demo3]#
```

#terraform init

#terraform plan

```
+ tags                         = {
    + "Name" = "nadodigal"
  }
```

*Important note:- 1, New instance name has been added, so after apply it will create again 4 instances on AWS with same tag names, which is not advisable to use, but as said earlier, instance ID will be uniq.

2, incase if you interchange the order inside the list, then also it will create new sets of instances.

3, better to avoid using count, for this case we can use for_each loop.

Step 2:- For_each loop

It will work in data type list, map, set only! list and set almost same, but set is fixed and list will work with index value.

*toset:- changing list to toset, because list will not work in for_each loop.

#vim variable.tf

```
[root@porali for_each]# cat variable.tf
variable "name" {
  type = list
  default = ["briyani", "parrota", "pongal"]
}
[root@porali for_each]#
```

#vim resource.tf

```
[root@porali for_each]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  key_name               = "drhiju86"
  vpc_security_group_ids = ["sg-b844c2cd"]
  for_each = toset(var.name)
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = each.value
  }
}
```

#terraform init

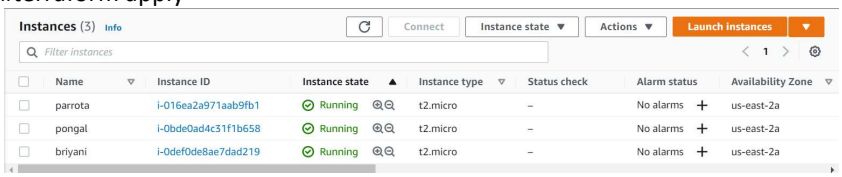
#terraform fmt

#terraform validate

#terraform plan

```
Terraform will perform the following actions:

  # aws_instance.web["briyani"] will be created
  + resource "aws_instance" "web" {
# aws_instance.web["parrota"] will be created
+ resource "aws_instance" "web" {
# aws_instance.web["pongal"] will be created
+ resource "aws_instance" "web" {
```

#terraform apply



Note:- Now three instance have been created.

Step 2.1 :- lets try to do testing by add one more value on variables

#vim variable.tf

```
[root@porali for_each]# cat variable.tf
variable "name" {
  type    = list(any)
  default = ["briyani", "parrota", "pongal", "dosa"]
}
[root@porali for_each]#
```

#terraform plan

```
Terraform will perform the following actions:

  # aws_instance.web["dosa"] will be created

Plan: 1 to add, 0 to change, 0 to destroy.
```

Note:- After execute “terraform plan”, it clearly says one resource going to be added.

Step2.2:- lets interchange the value in variables and check.

#vim variable.tf

```
[root@porali for_each]# cat variable.tf
variable "name" {
  type    = list(any)
  default = ["pongal", "parrota", "briyani"]
}
[root@porali for_each]#
```

terraform plan

```
[root@porali for_each]# terraform plan
aws_instance.web["briyani"]: Refreshing state... [id=i-0def0de8ae7dad219]
aws_instance.web["pongal"]: Refreshing state... [id=i-0bde0ad4c31f1b658]
aws_instance.web["parrota"]: Refreshing state... [id=i-016ea2a971aab9fb1]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes
are needed.
```

*Important note:- There is no changes happened even the value interchanged into the list, because this is not creating with index position value, here its working with mapping value, this is benefit for for_each loop, compare with count.

#terraform destroy

Step2.3:- let’s try to change type in variables

#vim variable.tf

```
[root@porali for_each]# cat variable.tf
variable "name" {
  type    = set(string)
  default = ["laddu", "halwa"]
}
[root@porali for_each]#
```

#vim resource.tf

```
[root@porali for_each]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = "ami-00f8e2c955f7ffa9b"
  instance_type          = "t2.micro"
  key_name               = "drhiju86"
  vpc_security_group_ids = ["sg-b844c2cd"]
  for_each               = var.name ✓
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = each.value
  }
}
[root@porali for_each]#
```

#terraform plan

terraform apply

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
laddu	i-02a6361cb62c18b21	Running	t2.micro	2/2 checks passed	No alarms	us-east-2a
halwa	i-0418bc1ca88baceea	Running	t2.micro	2/2 checks passed	No alarms	us-east-2a

*Important note:- here in variables, type “set” mentioned with element string, so the conclusion is type ‘set’ will never expect to work in order wise and no duplication will be there.

Step 3:- Lets add output.tf variable and try to get the output of instance ip.

vim output.tf

```
[root@porali for_each]# cat output.tf
output "public_ip" {
  value = {
    for instance in aws_instance.web :
    instance.tags.Name => instance.public_ip
  }
}
```

terraform fmt
terraform plan
terraform apply

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

public_ip = {
  "halwa" = "3.12.34.141"
  "laddu" = "13.58.56.59"
}
[root@porali for_each]#
```

*Important note:- so with for_each loop we can get the ouput with instance name and ip.

Step 4:- Datasource

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration. Use of data sources allows a Terraform configuration to build on information defined outside of Terraform, or defined by another separate Terraform configuration.

Each region will have different AMI-ID image, to use it dynamically in terraform, let see examples with datablock.

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami

*To get AMI-Image details, execute below command.

#aws ec2 describe-images --region us-east-2 --image-ids ami-00f8e2c955f7ffa9b

```
[root@porali data1]# aws ec2 describe-images --region us-east-2 --image-ids ami-00f8e2c955f7ffa9b
{
    "Images": [
        {
            "VirtualizationType": "hvm",
            "Description": "CentOS 7.9.2009 x86_64",
            "Hypervisor": "xen",
            "EnaSupport": true,
            "ImageId": "ami-00f8e2c955f7ffa9b",
            "State": "available",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/sda1",
                    "Ebs": {
                        "Encrypted": false,
                        "DeleteOnTermination": true,
                        "VolumeType": "gp2",
                        "VolumeSize": 8,
                        "SnapshotId": "snap-050067fd56d47ded4"
                    }
                }
            ],
            "Architecture": "x86_64",
            "ImageLocation": "125523088429/CentOS 7.9.2009 x86_64",
            "RootDeviceType": "ebs",
            "OwnerId": "125523088429",
            "RootDeviceName": "/dev/sda1",
            "CreationDate": "2020-11-10T13:14:34.000Z",
            "Public": true,
            "ImageType": "machine",
            "Name": "CentOS 7.9.2009 x86_64"
        }
    ]
}
```

#cat providers.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>3.0"
    }
  }
}

#Configure tje AWS Provider
provider "aws" {
  region  = var.region
  profile = "dev"
}
```

#cat resource.tf

```
data "aws_ami" "centos" {
  most_recent   = true
  owners        = ["125523088429"]

  filter {
    name   = "name"
    values  = ["CentOS 7.9.2009 x86_64"]
    }

  filter {
    name   = "architecture"
    values =  ["x86_64"]
    }
  filter {
    name   = "virtualization-type"
    values  = ["hvm"]
    }
 }

resource "aws_instance" "web" {
  ami                    = data.aws_ami.centos.id
  instance_type          = "t2.micro"

  tags = {
    Name = "robo"
  }
}
```

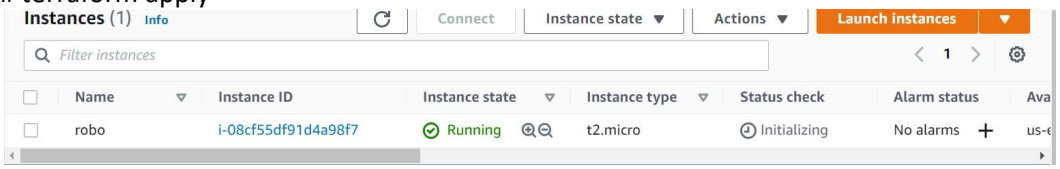
cat variable.tf

```
variable "region" {
    type = string
    default = "us-east-2"
}
```

terraform init
terraform validate
terraform plan

```
Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-00f8e2c955f7ffa9b"
```

terraform apply



Important note:- Now EC2 has been created by using datasource filter options.