# 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**

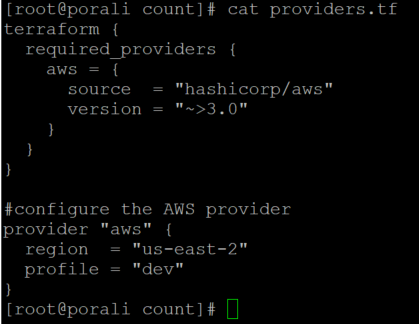
\*First create a directory.

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.

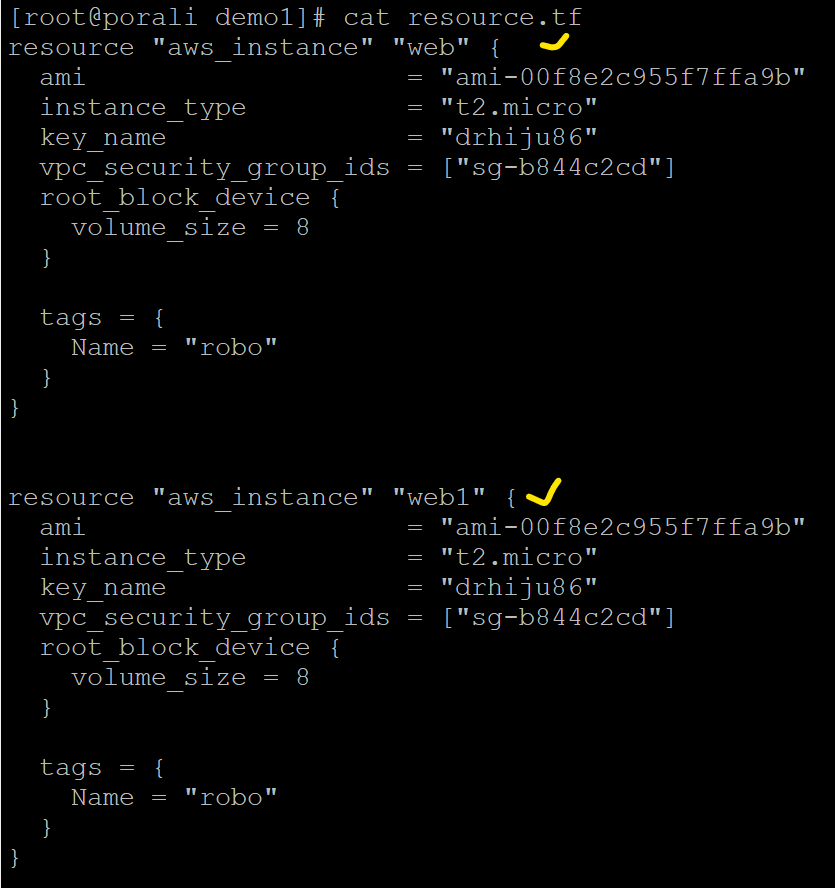
To avoid repeated resource blocks, count will be very useful.

#mkdir -p loop/{count,for\_each}

#vim providers.tf



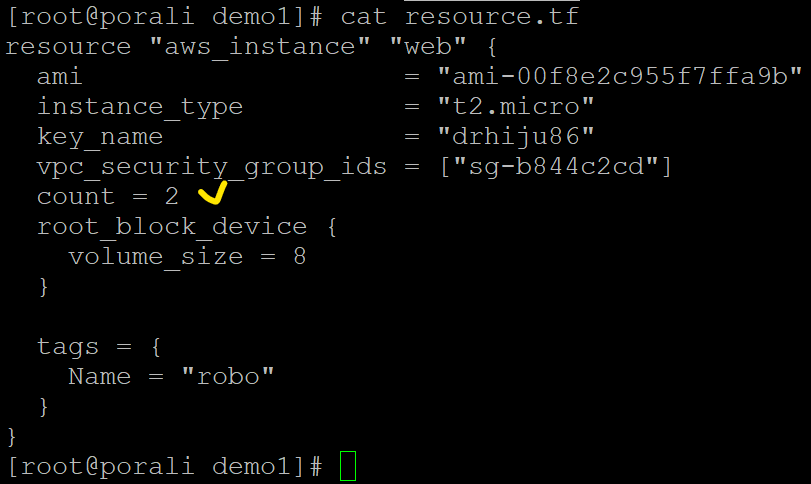
#vim resource.tf



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

@Example:- with count

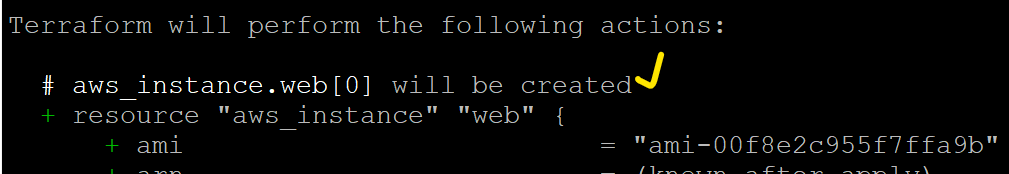
#vim resource.tf

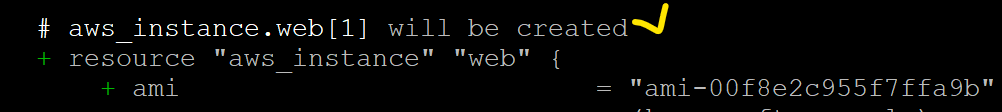


#terraform init

#terraform validate

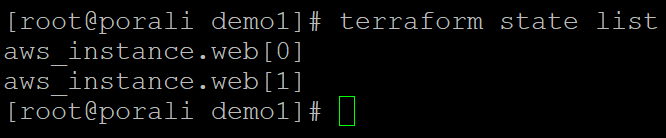
#terraform plan



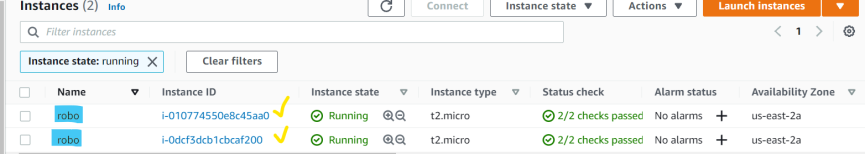


#terraform apply

#terraform state list



**Step 1.1:- Go to console and verify.**



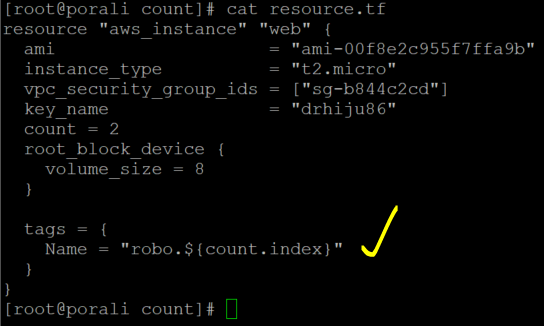
\*Note: - 1, “robo” is the reference for human, for AWS uniq instance ID is the reference.

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.**

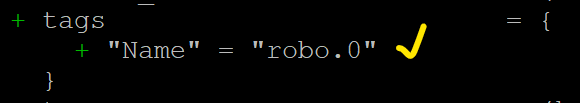
#vim providers.tf

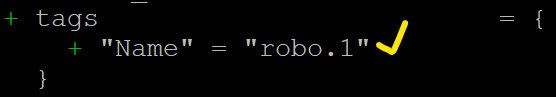
#vim resource.tf



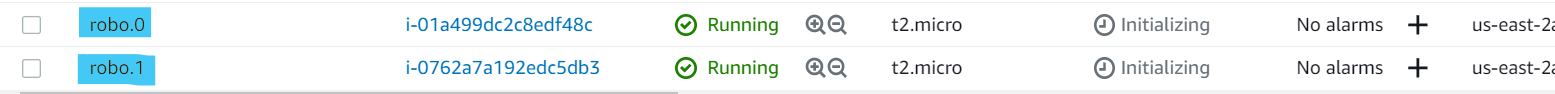
#terraform validate

#terraform plan





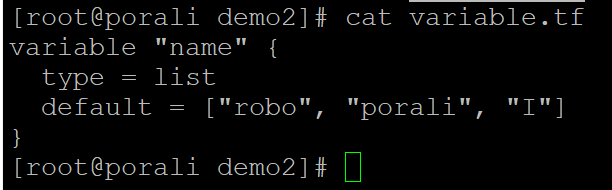
#terraform apply



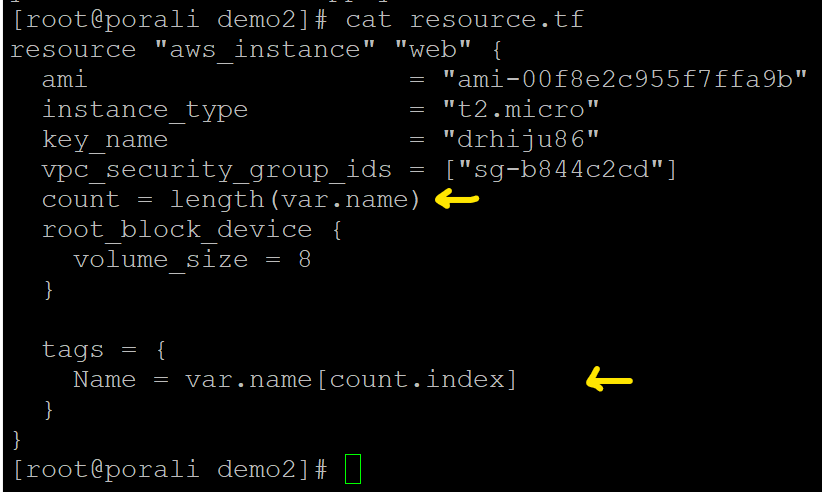
\*Important Note:- these count method cannot be used for IAM user creation, since for instance creation ok to use.

**Step 1.3:- Example with variable**

#vim variable.tf

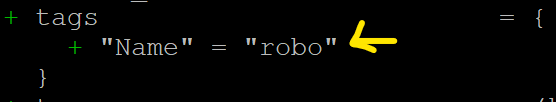


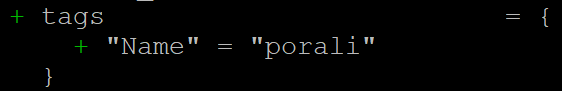
#vim resource.tf

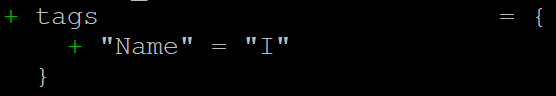


#terraform init

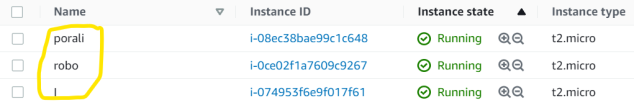
#terraform plan







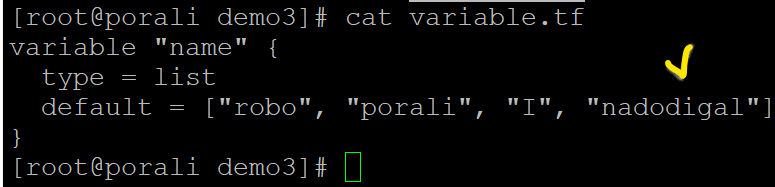
#terraform apply



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

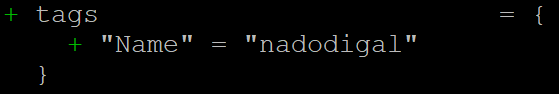
**Step 1.4:- Example added name in variable list.**

#vim variable.tf



#terraform init

#terraform plan



\*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.

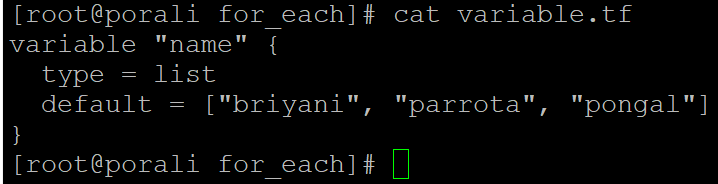
3, better to avoid using count, for this case we can use for\_each loop.

**Step 2:- For\_each loop**

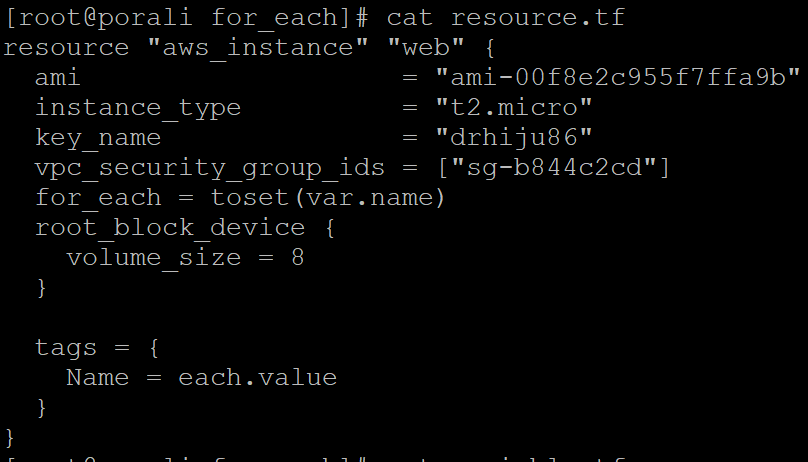
It will work in data type list, map, set only! list and set almost same, but set is fixed and list will work with index value.

\*toset:- changing list to toset, because list will not work in for\_each loop.

#vim variable.tf



#vim resource.tf

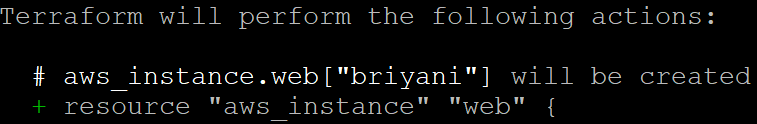


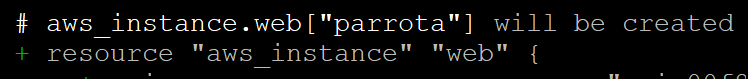
#terraform init

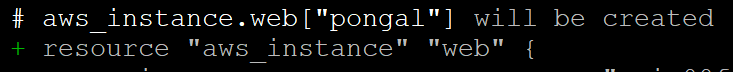
#terraform fmt

#terraform validate

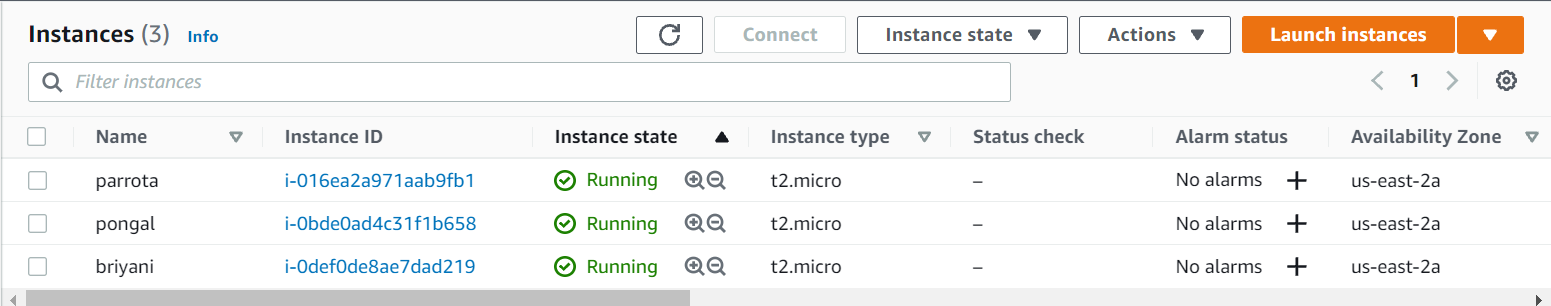
#terraform plan







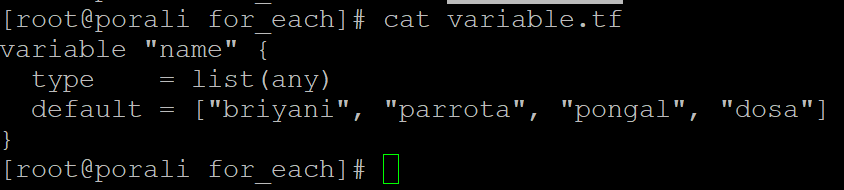
#terraform apply



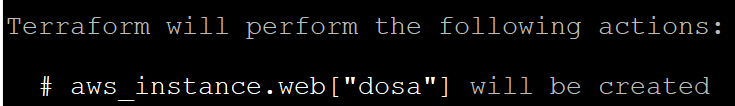
Note:- Now three instance have been created.

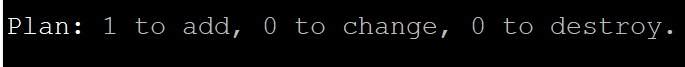
**Step 2.1 :- lets try to do testing by add one more value on variables**

#vim variable.tf



#terraform plan

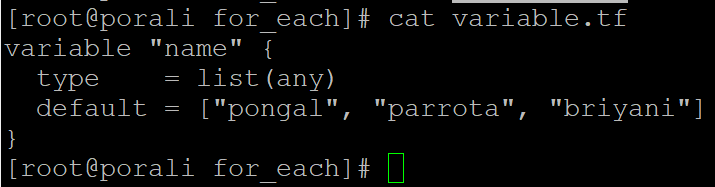




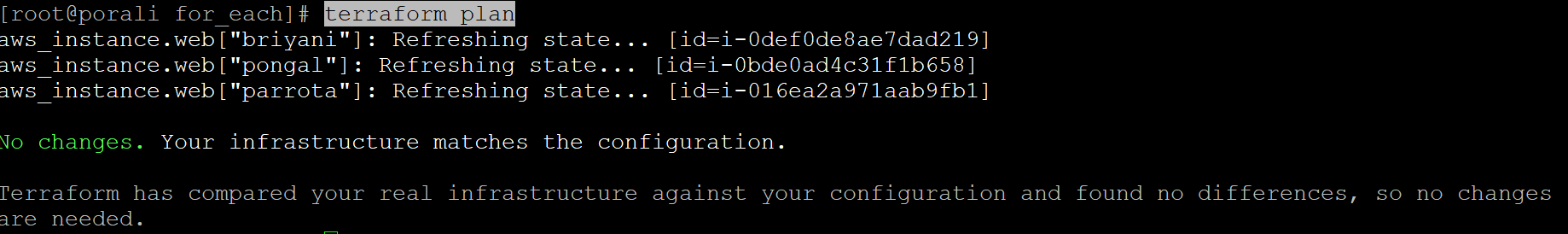
Note:- After execute “terraform plan”, it clearly says one resource going to be added.

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

#vim variable.tf



# terraform plan

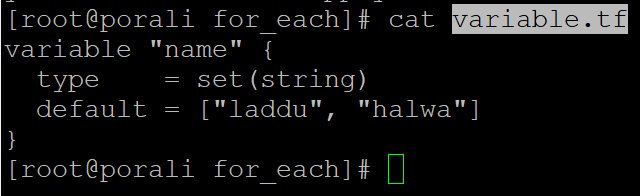


\*Important note:- There is no changes happened even the value interchanged into the list, because this is not creating with index position value, here its working with mapping value, this is benefit for for\_each loop, compare with count.

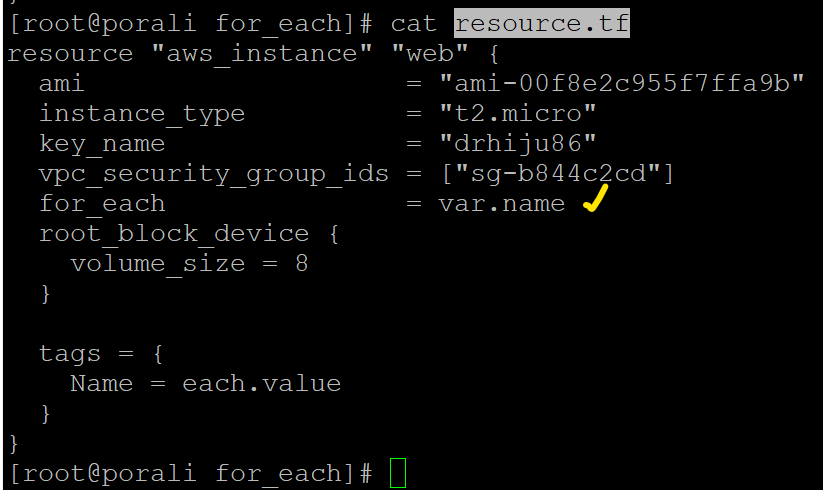
#terraform destroy

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

#vim variable.tf



#vim resource.tf



#terraform plan

# terraform apply



\*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



# terraform fmt

# terraform plan

# terraform apply



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

**Step 4:- Datasource**

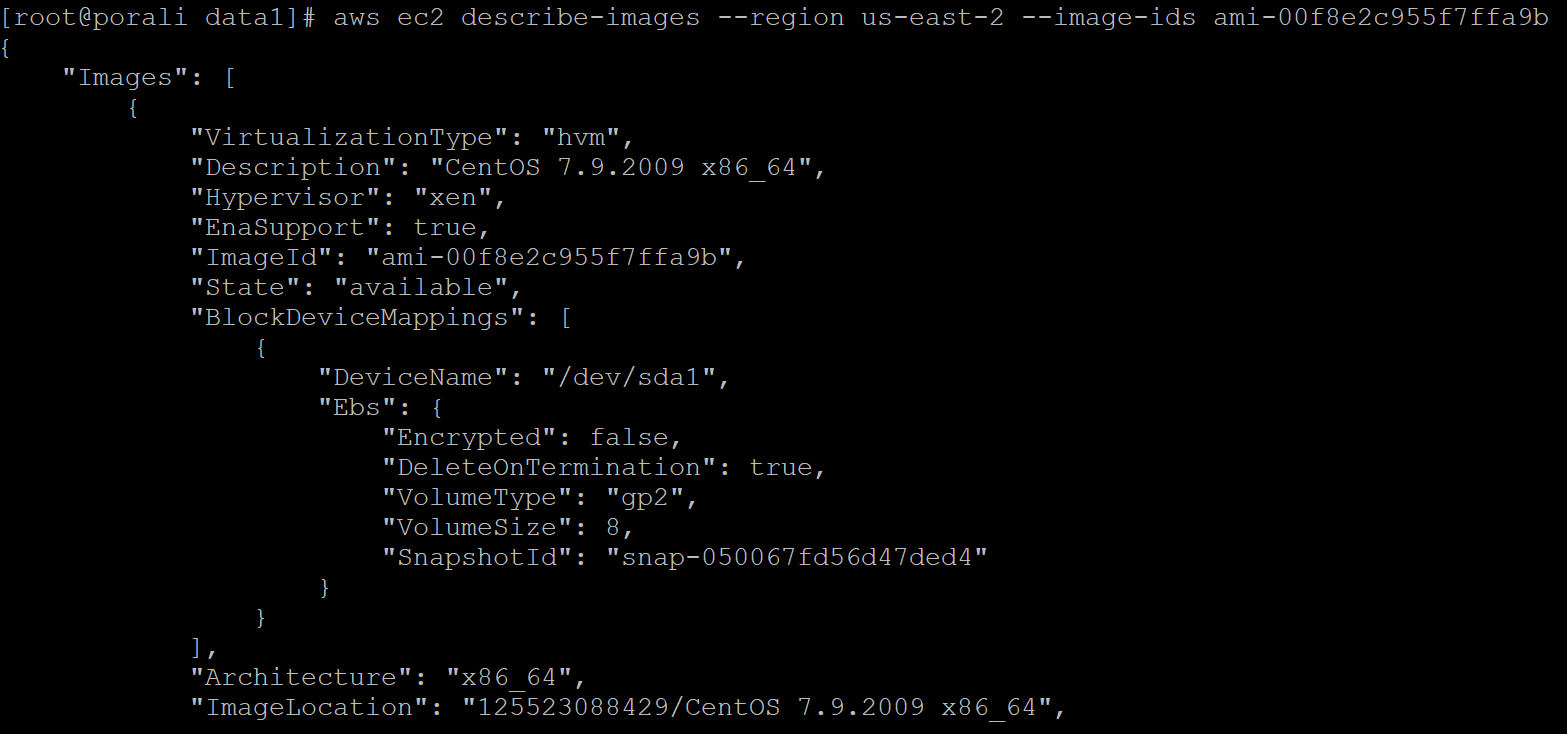
Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration. Use of data sources allows a Terraform configuration to build on information defined outside of Terraform, or defined by another separate Terraform configuration.

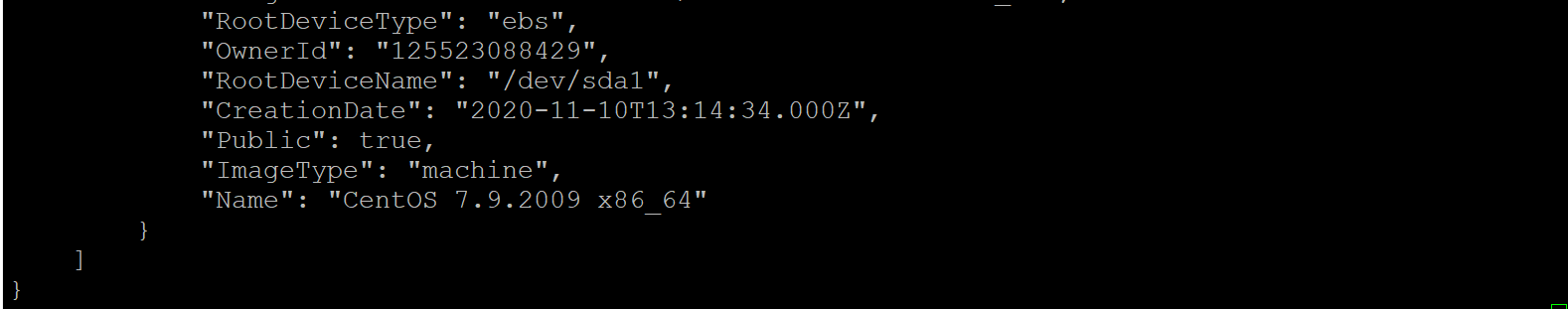
Each region will have different AMI-ID image, to use it dynamically in terraform, let see examples with datablock.

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

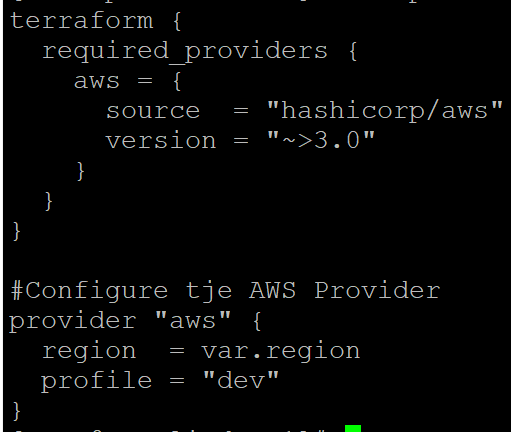
\*To get AMI-Image details, execute below command.

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

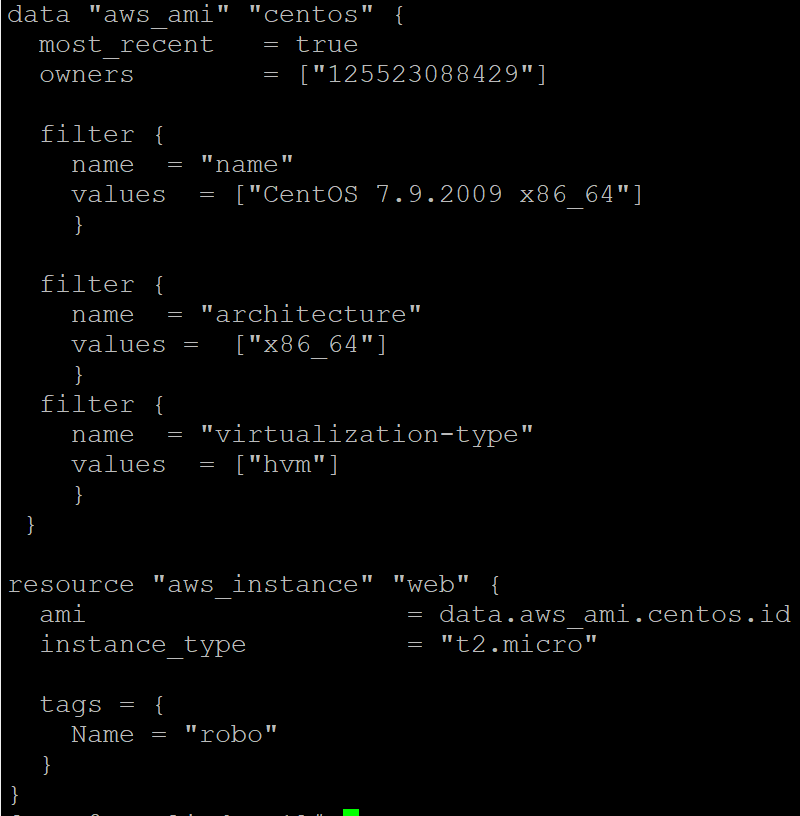




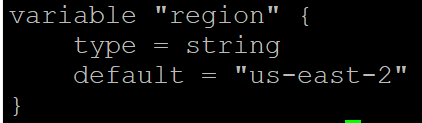
#cat providers.tf



#cat resource.tf



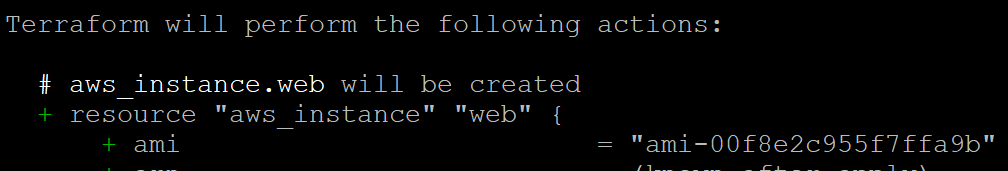
# cat variable.tf



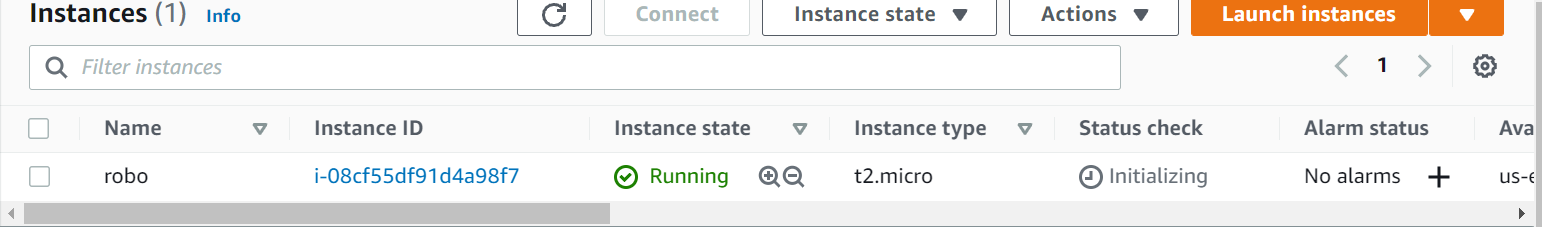
# terraform init

# terraform validate

# terraform plan



# terraform apply



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