Chaitanya Sreeram (Batch-123)

**[vcube devops training institute]**  [ mobile: 912158928, email: sreeramchaitanya97@gmail.com]

cREATING Multiple instances with unique names using TERRAFORM -variable block

***EXPLAINATION:***

To create multiple instances writing terraform script can be achieved with meta data arguments **“count”** and **“for\_each”**

Using **“count”** meta data argument we can create multiple instances for example by adding count (count = 3) and web-server-${(count. index + 1)} in tags name of resource block code of aws\_instance resource. It creates multiple instances with names web-server-1, web-server-2 and web-server-3. But which in this case we have to specify unique names.

If we want to give our own specific names to our instances. It can be achieved using meta data argument **“for\_each”** to give specific names to our ec2 instances

***SOURCE CODE***:

*TERRAFORM BLOCK:*

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "5.54.1"

}

}

}

*PROVIDER BLOCK:*

provider “aws” {

region = “ap-south-1”

profile = “default”

}

*VARIABLE BLOCK CODE:*

Variable “instances\_names” {

type = list(string)

default = [“vcube”, “devops”, “aws”]

variable “instance\_names” {…}

In variable block we’ve declared a variable named “instance\_names”

type = list(string)

By using type we’re specifying the type of variable and given value as list(string) means that the variable should be a list and each element in the list is a string.

default = [“aws”, “devops”, “vcube”]

The default value here is a list of instance names that could represents our server instances

*RESOURCE BLOCK:*

resource "aws\_instance" "" {

for\_each = toset (var. instance\_names)

ami = "ami-0f58b397bc5c1f2e8"

instance\_type = "t2. micro"

tags = {

Name = each.value

}

}

for\_each = toset (local. instance\_names)

toset converts the list of instance names to a set, allowing Terraform to iterate over each unique value.

tags = {

Name = each.value

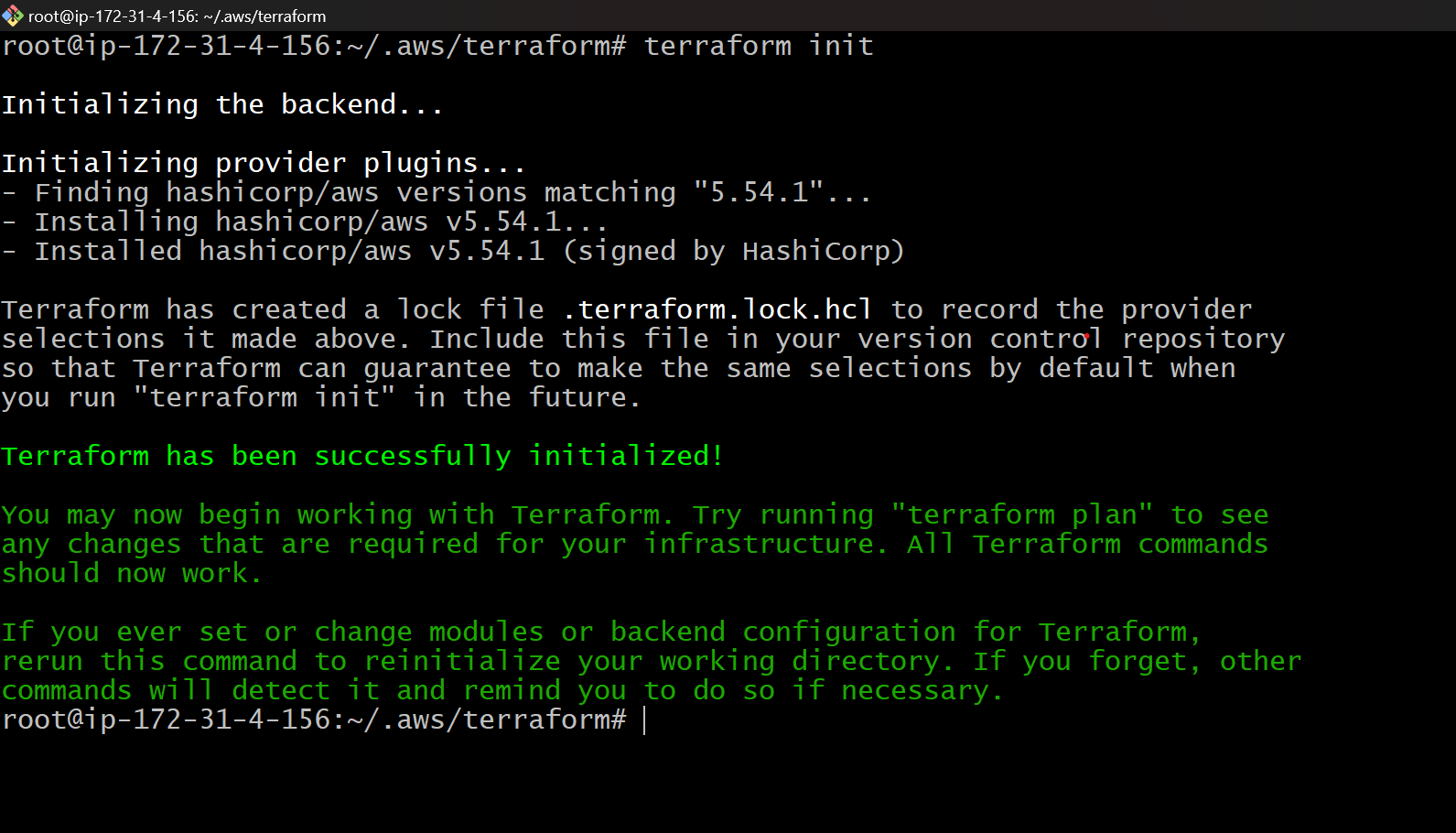
}

Here each. value sets each EC2 instance with a name corresponding to the current element in the var. instance\_names list.

Run command: terraform init

It initializes Terraform config directory and downloads and installs necessary provider plugins and sets up backend for state management. And fig-1 shows successful installation of terraform.

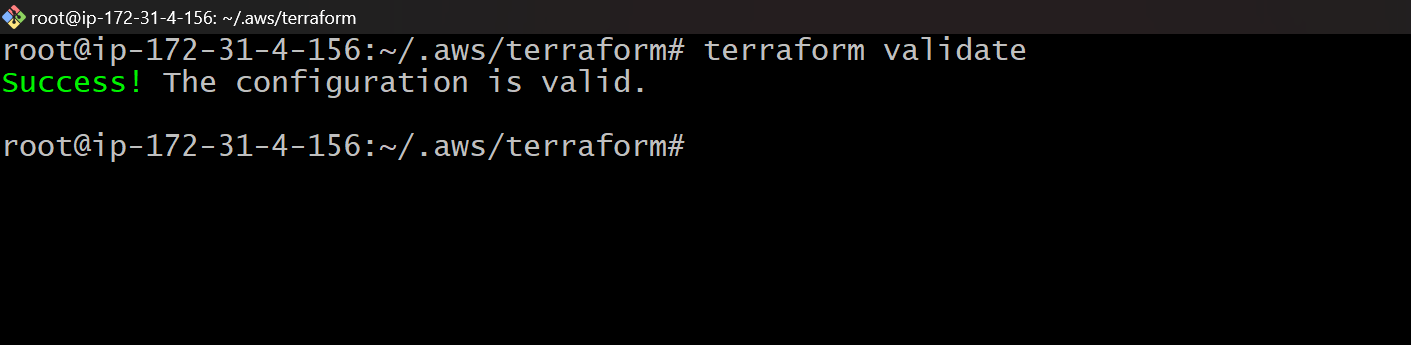
Figure-1:



Run command: terraform validate

Validates Terraform configuration files (terraform.tf, resource.tf, varible.tf and provider.tf) & ensures every block code is syntactically correct or not. Here fig -2 shows success that means the config is valid

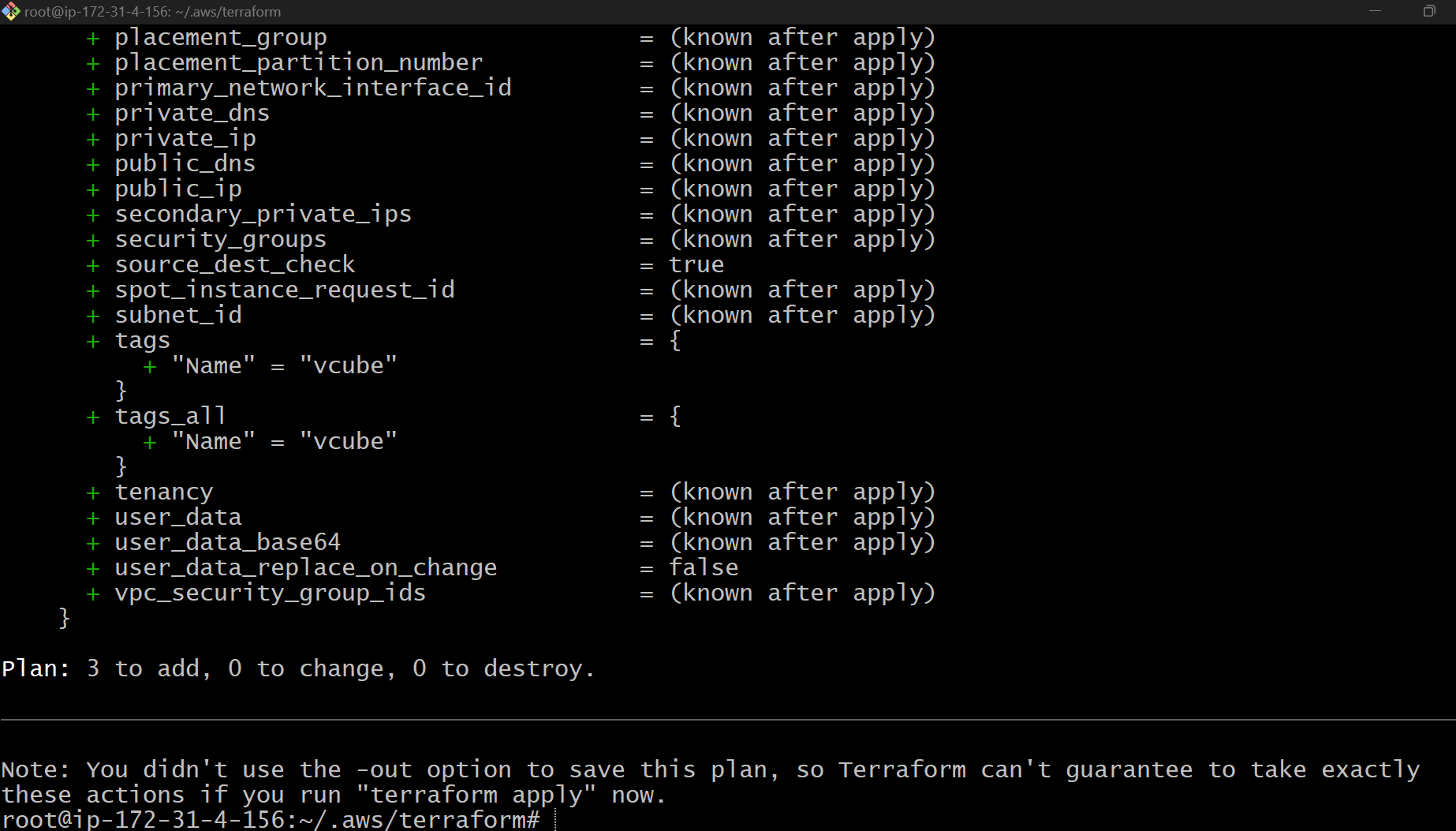
Figure-2:



Run Command: terraform plan

Creates an execution plan that shows what actions Terraform will take to achieve the desired state.

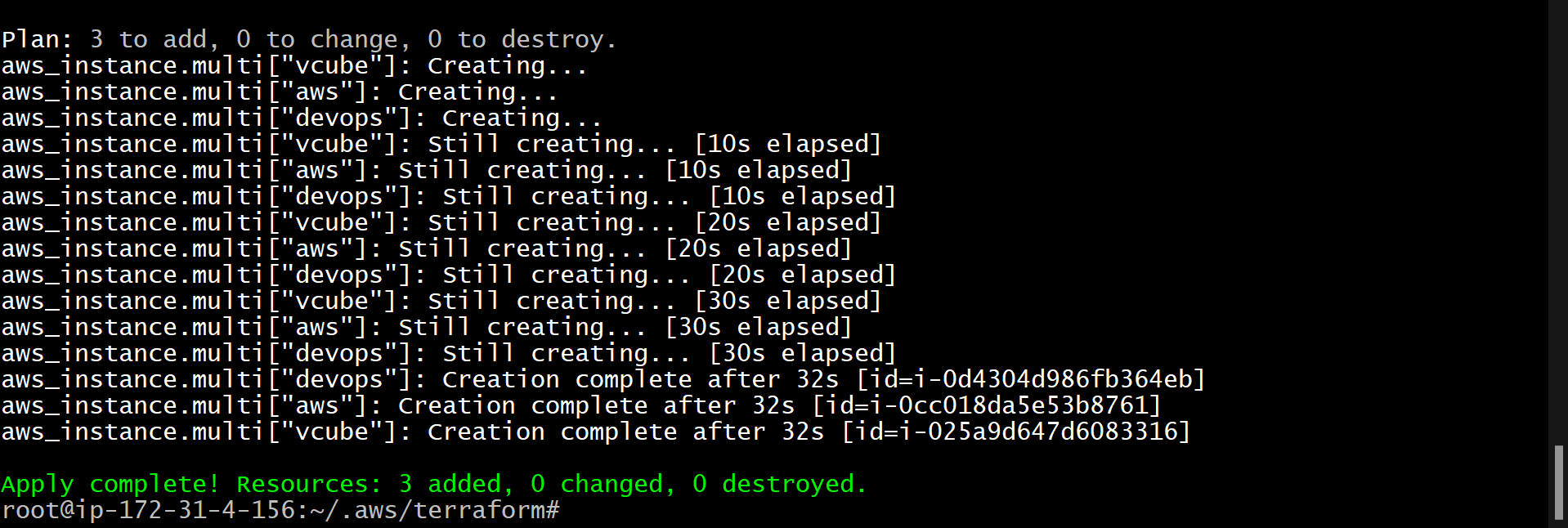
Figure-3:



Run Command: terraform apply – auto-approve

To execute our planned actions here in this case we’re creating 3 instances are above to create as shown in fig 4.

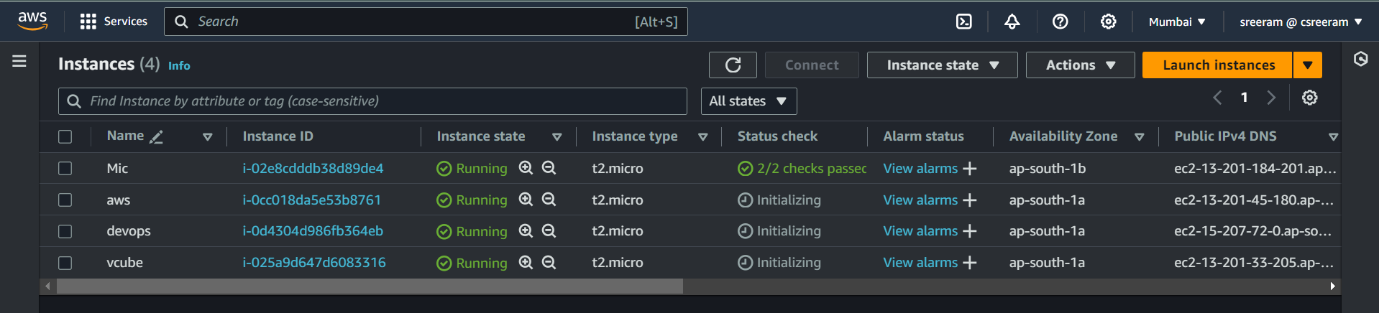
Figure-4:



Here we’ve set our code to create 3 instances and all the above commands have run successfully so it should create 3 instances.

And to confirm whether it’s created or not go to aws console and check for if three newly added running instances naming aws, devops and vcube must be present in your EC2 Dashboard as shown in fig-5.

Figure-5:



Three instances naming aws, vcube and devops are successfully created.

Run command: terraform destroy

It helps us to destroy the infrastructure we created using terraform which is shown in fig-6 and confirmation that destroy command destroyed managed infrastructure’s as shown in fig-7 respectively.

Figure-6:

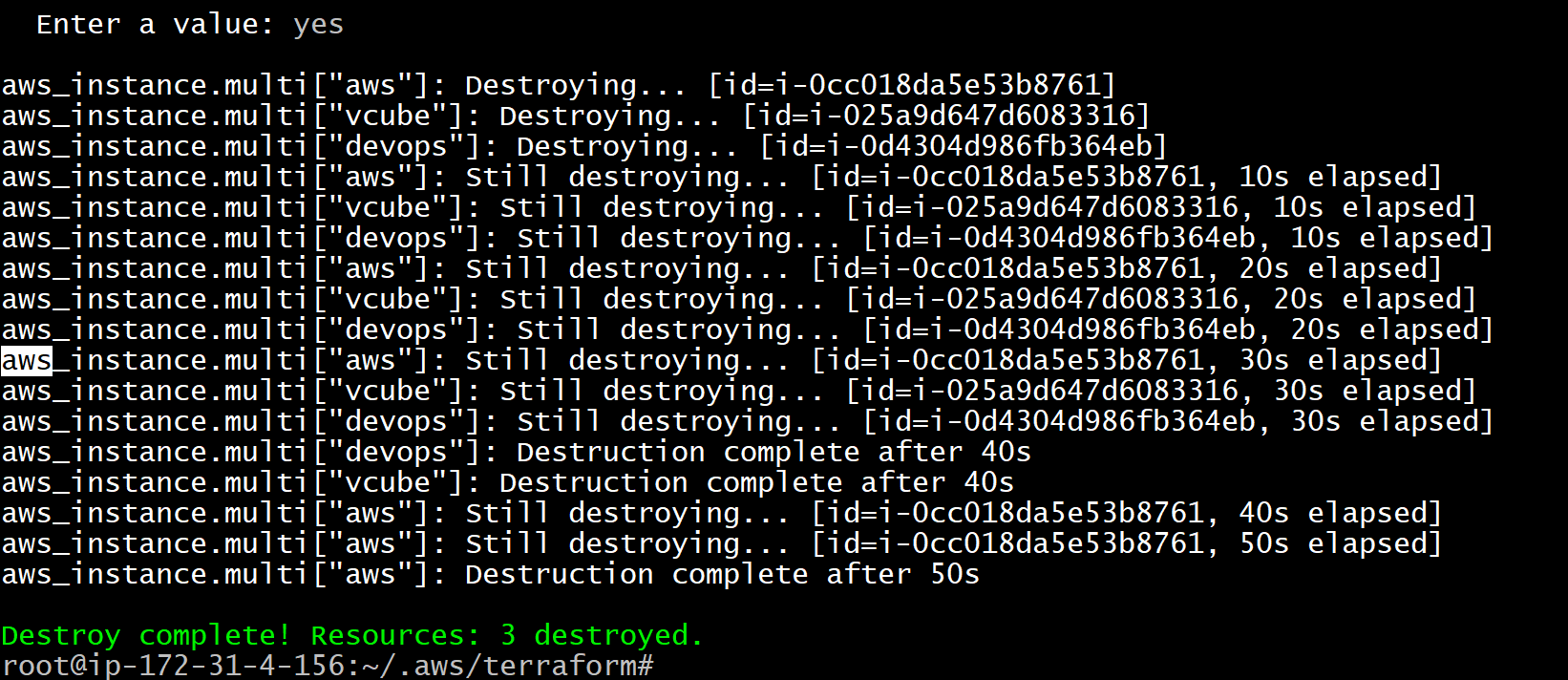
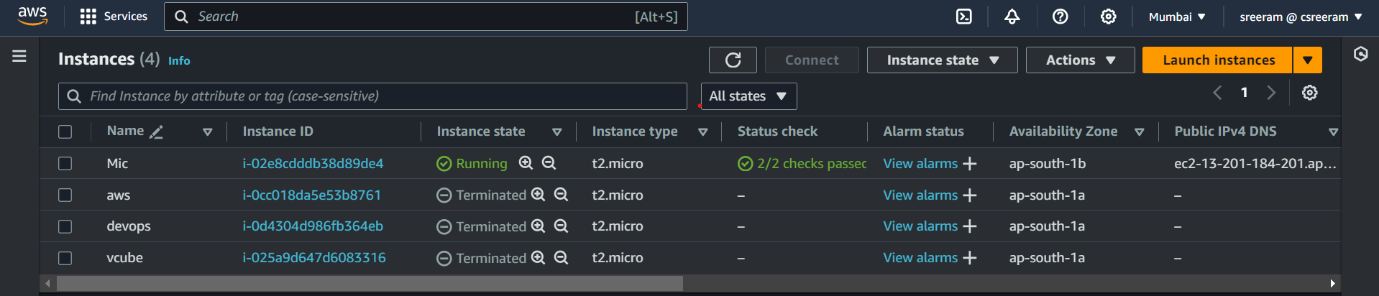


Figure-7:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_THE END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_