terraform

* Open source
* IAAC – ( infrastructure as a code ) like cloudformation
* Terraform support multiple cloud provider +200
* In aws cloudformation and Microsoft has a azure rm , openshift has a heat in general overall terraform use in all scenario , like docker Kubernetes all task we can managed using terraform
* Using terraform we can launch cloudprovider resources in terraform
* For example - suppose requirement is launch
* 10 vpc
* 50 ec2 instance
* 25 eip
* 20 s3
* In general way we launch them as separate but it is time taking process
* In terraform we can write a file this file we can re-use as many as times so, other requirement we only change numbers -cidr range-port no necessary changes we use this files for upcoming requirements for reusability
* So terraform we use for Reusability
* Terraform is written in ‘’ Go ‘’ language
* We written a file in terraform ‘’ HCL ‘’ hashicorp config language like json
* Example:
* Json HCL

Key:value blocks - provider { provider “aws”

Resource aws \_instance =

Output keypairs =

}

Variable

* Scenario to install terraform
* Launch a server ubuntu or amazon linux nad install terraform
* Check vrsion of terraform
* In terraform file extension is .tf is important like ex- main.tf , index.tf , variable.tf
* Terraform commands
* tereraform init
* terraform plan
* terraform apply
* terraform validate
* terraform fmt (for format check )
* installing terraform on ubuntu

sudo apt update

sudo apt install software-properties-common gnupg2 curl

curl https://apt.releases.hashicorp.com/gpg | gpg --dearmor > hashicorp.gpg

sudo install -o root -g root -m 644 hashicorp.gpg /etc/apt/trusted.gpg.d/

sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

sudo apt install terraform

terraform –version

* 1st task
* Create folder - mkdir terraform

Cd terraform /

* Create file - vi main.tf

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

provider "aws" {

region = "us-east-1"

access\_key = " AKIAZQ3DSHGTLEKD5KEF"

secret\_key = "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"

}

resource "aws\_instance" "myec2" {

ami= "ami-04b4f1a9cf54c11d0"

instance\_type= "t2.micro"

}

* Next step is run command - terraform init
* Example :

ubuntu@ip-172-31-89-187:~/terraform$ vi main.tf

ubuntu@ip-172-31-89-187:~/terraform$ terraform init

Initializing the backend...

Initializing provider plugins...

- Finding hashicorp/aws versions matching "~> 5.0"...

- Installing hashicorp/aws v5.90.1...

- Installed hashicorp/aws v5.90.1 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider

selections it made above. Include this file in your version control repository

so that Terraform can guarantee to make the same selections by default when

you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see

any changes that are required for your infrastructure. All Terraform commands

should now work.

* Next step is launch the server or execute command : - terraform plan

* Example :

ubuntu@ip-172-31-89-187:~/terraform$ terraform plan

Terraform used the selected providers to generate the

following execution plan. Resource actions are indicated

with the following symbols:

+ create

Terraform will perform the following actions:

# aws\_instance.my-ec2 will be created

+ resource "aws\_instance" "my-ec2" {

+ ami = "ami-04b4f1a9cf54c11d0"

+ arn = (known after apply)

+ associate\_public\_ip\_address = (known after apply)

+ availability\_zone = (known after apply)

+ cpu\_core\_count = (known after apply)

+ cpu\_threads\_per\_core = (known after apply)

+ disable\_api\_stop = (known after apply)

+ disable\_api\_termination = (known after apply)

+ ebs\_optimized = (known after apply)

+ enable\_primary\_ipv6 = (known after apply)

+ get\_password\_data = false

+ host\_id = (known after apply)

+ host\_resource\_group\_arn = (known after apply)

+ iam\_instance\_profile = (known after apply)

+ id = (known after apply)

+ instance\_initiated\_shutdown\_behavior = (known after apply)

+ instance\_lifecycle = (known after apply)

+ instance\_state = (known after apply)

+ instance\_type = "t2.micro"

+ ipv6\_address\_count = (known after apply)

+ ipv6\_addresses = (known after apply)

+ key\_name = (known after apply)

+ monitoring = (known after apply)

+ outpost\_arn = (known after apply)

+ password\_data = (known after apply)

+ placement\_group = (known after apply)

+ placement\_partition\_number = (known after apply)

+ primary\_network\_interface\_id = (known after apply)

+ private\_dns = (known after apply)

+ private\_ip = (known after apply)

+ public\_dns = (known after apply)

+ public\_ip = (known after app

* Final command to execute or Luch an instance command: - terraform apply -auto-approve

* Example :

terraform apply -auto-a

pprove

Terraform used the selected providers to generate the

following execution plan. Resource actions are indicated

with the following symbols:

+ create

Terraform will perform the following actions:

# aws\_instance.my-ec2 will be created

+ resource "aws\_instance" "my-ec2" {

+ ami = "ami-04b4f1a9cf54c11d0"

+ arn = (known after apply)

+ associate\_public\_ip\_address = (known after apply)

+ availability\_zone = (known after apply)

#

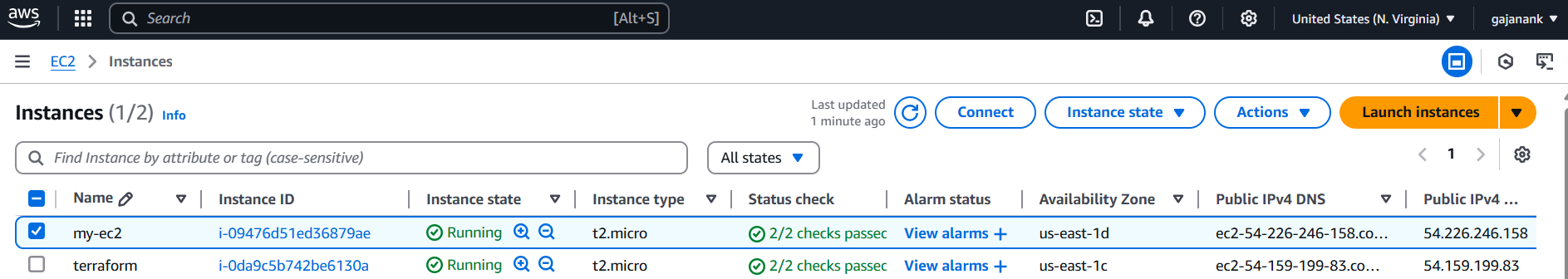
Plan: 1 to add, 0 to change, 0 to destroy.

aws\_instance.my-ec2: Creating...

aws\_instance.my-ec2: Still creating... [10s elapsed]

aws\_instance.my-ec2: Creation complete after 13s [id=i-09476d51ed36879ae]

* Instance lauch using terraform



* To delete or destroy server or activity command :- terraform destroy

* Example :

ubuntu@ip-172-31-89-187:~/terraform$ terraform destroy

aws\_instance.my-ec2: Refreshing state... [id=i-09476d51ed36879ae]

Terraform used the selected providers to generate the

following execution plan. Resource actions are indicated

with the following symbols:

- destroy

Terraform will perform the following actions:

# aws\_instance.my-ec2 will be destroyed

- resource "aws\_instance" "my-ec2" {

- ami = "ami-04b4f1a9cf54c11d0" -> null

- arn = "arn:aws:ec2:us-east-1:654654454182:instance/i-09476d51ed36879ae" -> null

- associate\_public\_ip\_address = true -> null

- availability\_zone = "us-east-1d" -> null

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

Do you really want to destroy all resources?

Terraform will destroy all your managed infrastructure, as shown above.

There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

aws\_instance.my-ec2: Destroying... [id=i-09476d51ed36879ae]

aws\_instance.my-ec2: Still destroying... [id=i-09476d51ed36879ae, 10s elapsed]

aws\_instance.my-ec2: Still destroying... [id=i-09476d51ed36879ae, 20s elapsed]

aws\_instance.my-ec2: Still destroying... [id=i-09476d51ed36879ae, 30s elapsed]

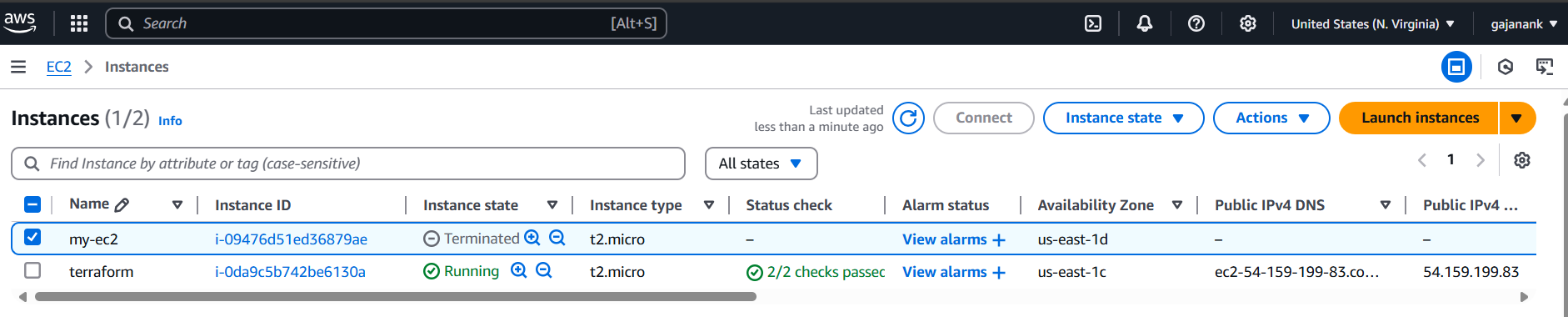
aws\_instance.my-ec2: Still destroying... [id=i-09476d51ed36879ae, 40s elapsed]

aws\_instance.my-ec2: Still destroying... [id=i-09476d51ed36879ae, 50s elapsed]

aws\_instance.my-ec2: Destruction complete after 50s

Destroy complete! Resources: 1 destroyed.

* Example : instance deleted through using terraform



\*\* new task

* Create folder - mkdir terraform
* Create file in that folder - cd terraform $ vi main.tf

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "my-ec2" {

ami= "ami-04b4f1a9cf54c11d0"

instance\_type= "t2.micro"

count = 2

tags ={

Name = "terraform-int-${count.index+1}"

}

* Without access key and secret access key mention in vi file we can add access key & secret access key later through commands and execute file
* Run commands :-

export AWS\_ACCESS\_KEY\_ID=AKIAZQ3DSHGTNL75Y3GW

export AWS\_SECRET\_ACCESS\_KEY=NjR855Yjy3hu8/VZUR9BzMwkqwTnVVLVj6GfCKoo

* Run command - terraform plan

* Example :-

ubuntu@ip-172-31-89-187:~/terraform$ export AWS\_ACCESS\_KEY\_ID=AKIAZQ3DSHGTNL75Y3GW

ubuntu@ip-172-31-89-187:~/terraform$ export AWS\_SECRET\_ACCESS\_KEY=NjR855Yjy3hu8/VZUR9BzMwkqwTnVVLVj6GfCKoo

ubuntu@ip-172-31-89-187:~/terraform$ terraform plan

Terraform used the selected providers to generate the

following execution plan. Resource actions are indicated

with the following symbols:

+ create

Terraform will perform the following actions:

# aws\_instance.my-ec2[0] will be created

+ resource "aws\_instance" "my-ec2" {

+ ami = "ami-04b4f1a9cf54c11d0"

+ arn = (known after apply)

+ associate\_public\_ip\_address = (known after apply)

+ network\_interface (known after apply)

+ private\_dns\_name\_options (known after apply)

+ root\_block\_device (known after apply)

}

Plan: 2 to add, 0 to change, 0 to destroy.

* Next command - terraform apply -auto-approve

* Example :-

ubuntu@ip-172-31-89-187:~/terraform$ terraform apply -auto-a

pprove

Terraform used the selected providers to generate the

following execution plan. Resource actions are indicated

with the following symbols:

+ create

Terraform will perform the following actions:

# aws\_instance.my-ec2[0] will be created

+ resource "aws\_instance" "my-ec2" {

+ ami = "ami-04b4f1a9cf54c11d0"

+ arn = (known after apply)

+ associate\_public\_ip\_address = (known after apply)

+ availability\_zone = (known after

#

Plan: 2 to add, 0 to change, 0 to destroy.

aws\_instance.my-ec2[0]: Creating...

aws\_instance.my-ec2[1]: Creating...

aws\_instance.my-ec2[0]: Still creating... [10s elapsed]

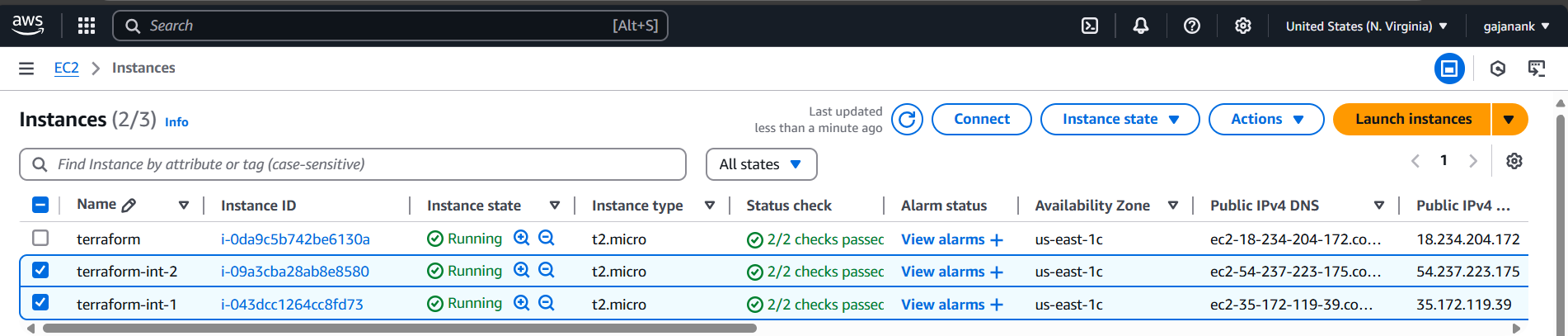
aws\_instance.my-ec2[1]: Still creating... [10s elapsed]

aws\_instance.my-ec2[1]: Creation complete after 13s [id=i-09a3cba28ab8e8580]

aws\_instance.my-ec2[0]: Creation complete after 13s [id=i-043dcc1264cc8fd73]

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

# instance wii be create through terraform with using access and secret key through export commands



* Without using console we add security group through CLI using terraform

Vi main.tf

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "my-ec2" {

ami= "ami-04b4f1a9cf54c11d0"

instance\_type= "t2.micro" vpc\_security\_group\_ids=[aws\_securiy\_group.server-sg.id]

tags ={

Name = "server"

}

}

resource "aws\_security\_group" "server-sg" {

ingress {

from\_port = 22

to\_port = 22

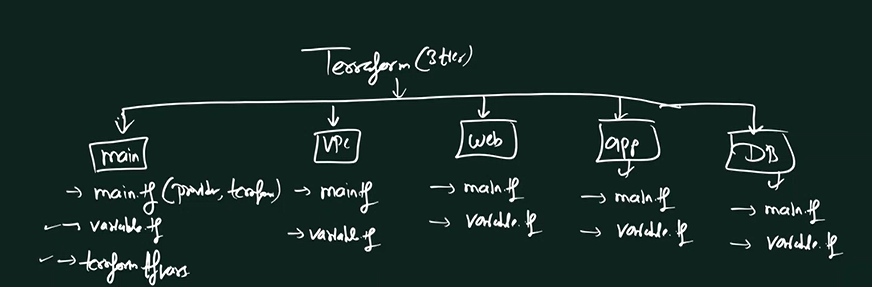
protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

* Variables in terraform
* Input variable
* Output variable
* Local variable
* Remote execute in terraform
* Local execute in terraform
* Terraform command s
* ( All other commands in terraform & there uses ) :- terraform -help
* terraform workspace (to delete any workspace or list,show.new,select in terraform)
* terraform workspace list (to check how many workspace and deafault workspace)
* Terraform cloud
* Create instance using terraform cloud
* \* Steps
* Terraform cloud - sign up -email
* Create organisation
* Create workspace
* New project
* Git (LINK PROJECT)
* Repo – main.tf & variable.tf
* Success
* Secret & access key
* Terraform plan
* Terraform apply
* Create three tier architecture using terraform
* Vpc
* 3 subnet
* Igw
* Route table – public and private
* Web server -nginx
* App server -php
* Db server – rds
* Using modules (sub folder) in terraform or different file for each



\* New task

* Any resource has already in console we handle or managed in using( terraform import ) command that we use
* Import s3 bucket in terraform

&

* In console Import any services or other services in terraform

Ex- s3 , ec2 , rds , etc

* Any resource has already in console we handle or managed in using ( terraform import ) command we use
* Basically import consoles services or resources in terraform
* Use ( terraform import ) command