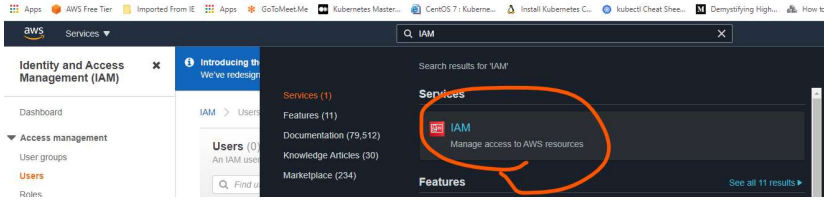
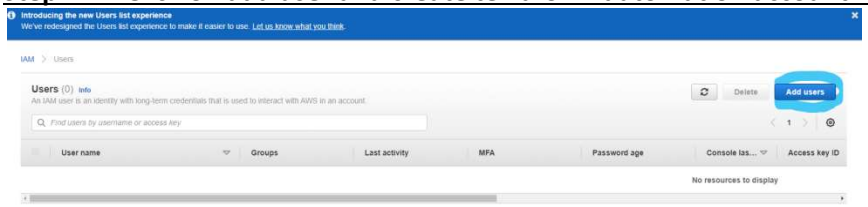
Providers

Note: you need to have an account created on AWS.

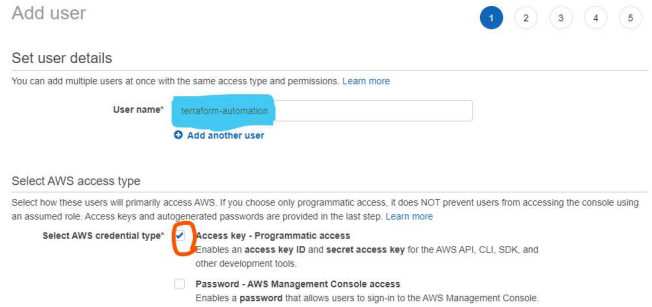
Step 1:- Open AWS console and search for IAM



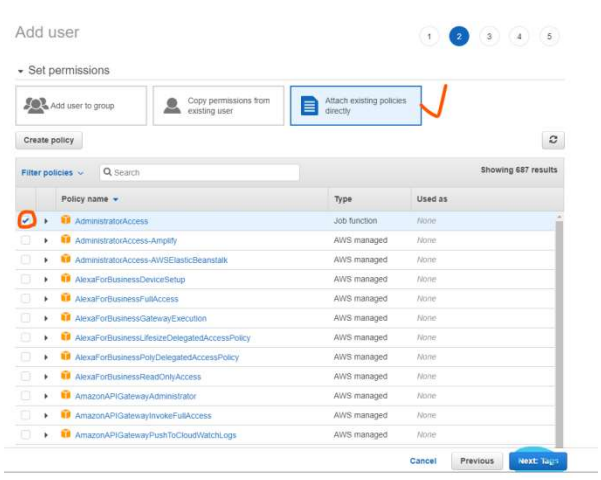
Step 1.1 :-Click on add user and create terraform-automation account for infra build

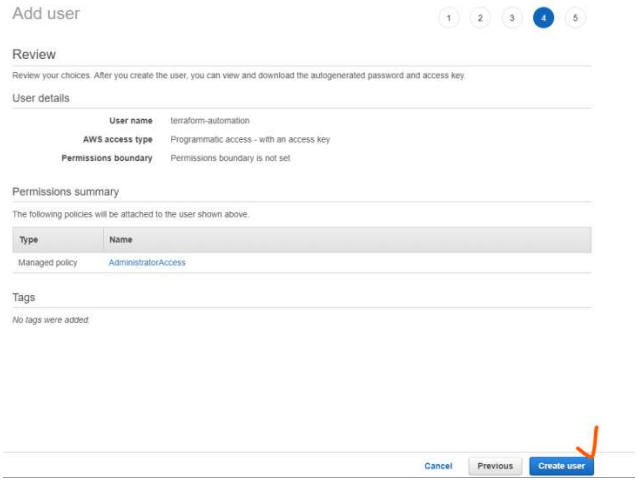


Step 1.2 Enter username and select Access key – programmatic access

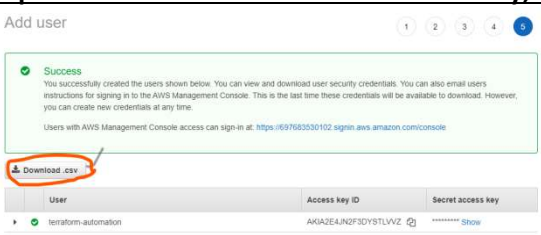


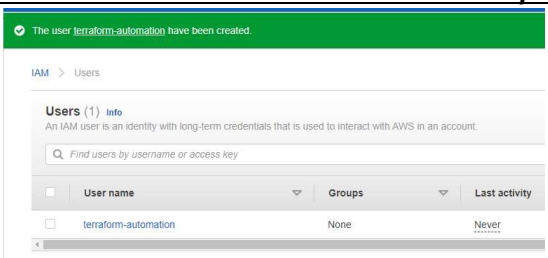
Step 1.3 Attach existing policy to user by selecting administrator access





Step 1.4:- User has been created successfully, now download both the access and secret keys.





Step 2:- Go back to GCP cloud and Install awscli module on centos machine.

yum install awscli –y

#aws --version

```
[root@porali ~]# aws --version
aws-cli/1.14.28 Python/2.7.5 Linux/3.10.0-1160.42.2.el7.x86_64 botocore/1.8.35
[root@porali ~]#
```

Step 3:-To build infra for authentication, secret keys will be used. There are three types,

1, environmental variable (ex:- EXPORT TF_ACCESS_ID), if we have different env like test,stage,dev& prod then for every build we need to add the env user variables.

2, user profile, configure the user profile on server and terraform will easily communicate with any env at anytime, this is advisable.

3, direct method, we will be storing the keys on terraform code, which will be placed on version control and that have being used or misused since everyone will be using the same keys.

User profile method:- create profile for dev env.

aws configure --profile dev

```
[root@porali ~]# aws configure --profile dev
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]:
Default output format [None]:
[root@porali ~]#
```

Step 3.1 :-Go to .aws folder and you view the details.

```
[root@porali .aws]# pwd
/root/.aws
[root@porali .aws]# cat config
[profile dev]
[root@porali .aws]# cat credentials
[dev]
aws_access_key_id = [redacted]
aws_secret_access_key = [redacted]
[root@porali .aws]#
```

Step 4:- Whats is providers? and Lets create a provider.tf

*Terraform relies on plugins called "providers" to interact with remote systems. Terraform configurations must declare which providers they require, so that Terraform can install and use them. Additionally, some providers require configuration (like endpoint URLs or cloud regions) before they can be used.

*Each provider adds a set of resource types and/or data sources that Terraform can manage. Every resource type is implemented by a provider; without providers, Terraform can't manage any kind of infrastructure. Most providers configure a specific infrastructure platform (either cloud or self-hosted).

Provider.tf file will have the information about terraform version, source of cloud detail, region and profile etc...

#vim providers.tf

```
[root@porali demo1]# 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 demo1]#
```

Step 4.1:- Run terraform init.

The terraform init command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

#terraform init

```
[root@porali ec2]# terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v3.0.0

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.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
[root@porali ec2]#
```

@now you can able to see the hidden file and folder, which is not supposed to be modified manually.

```
[root@porali ec2]# ls -lrta
total 12
drwxr-xr-x 3 root root   52 Sep 27 00:04 ..
-rw-r--r-- 1 root root  201 Sep 27 00:47 providers.tf
drwxr-xr-x 3 root root   23 Sep 27 01:07 .terraform
-rw-r--r-- 1 root root 1029 Sep 27 01:07 .terraform.lock.hcl
```

Important Note:- * If you specify the version = "3.0" then it will initialize the stable plugins and it advisable to use stable version for prod.

* incase if you mention the version = "~> 3.0" then it will initialize the latest version for terraform plugins or modules. We can use this for test or development.

* incase if you change the version on providers, then you need to delete the terraform hidden lock file and folder.

```
root@porali ec2]# ls -lrt .terraform/providers/registry.terraform.io/hashicorp/aws/3.0.0/linux_amd64/terraform-provid
r-aws_v3.0.0_x5
-rwxr-xr-x 1 root root 151076864 Sep 27 01:07 .terraform/providers/registry.terraform.io/hashicorp/aws/3.0.0/linux_amd
64/terraform-provider-aws_v3.0.0_x5
[root@porali ec2]# cat .terraform.lock.hcl
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version     = "3.0.0"
  constraints = "3.0.0"
  hashes = [
    "h1:ULKfwySvQ4pDhy027ryRhLxDhg640wsojYc+7NHMFBU=",
    "zh:25294510ae9c250502f2e37ac32b01017439735f098f82a1728772427626a2fd",
    "zh:3b723e7772d47bd8cc11bea6e5d3e0b5e1df8398c0e7aaf510e3a8a54e0f1874",
    "zh:4b7b73b86f4a0705d5d2a7f1d3ad3279706bdb3957a48f4a389c36918fba838e",
    "zh:9e26cdc3be97e3001c253c0ca28c5c8ff2d5476373calbeb849f3f3957ce7f1a",
    "zh:9e73cf1304bf57968d3048d70c0b766d41497430a2a9a7a718a196f3a385106a",
    "zh:a30b5b66facfbb2b02814e4cd33ca9899f9ade5bbf478f78c41d2fe789f0582a",
    "zh:b06fb5da094db41cb5e430c95c988b73f32695e9f90f25499e926842dbd21b21",
    "zh:c5a4ff607e9e9edee3fcd6d6666241fb532adf88ea1fe24f2aa1eb36845b3ca3",
    "zh:df568a69087831c1780fac4395630a2cfb3cdf67b7dffbfe16bd78c64770bb75",
    "zh:fce1b69dd673aace19508640b0b9b7eb1ef7e746d76cb846b49e7d52e0f5fb7e",
  ]
}
[root@porali ec2]#
```

Step 5:- Let create instances on AWS cloud by using resource file.

*resource.tf or main.tf, will contain the main set of configuration for your module. You can also create other configuration files and organize them however makes sense for your project.

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance

#vim resource.tf

```
[root@porali ec2]# 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"
  }
}
```

Note:- ami will be differ from region wise.

Step 5.1:- terraform fmt, terraform validate, terraform plan and terraform apply

*The **terraform fmt** command is used to rewrite Terraform configuration files to a canonical format and style.

* The **terraform validate** command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc

* The **terraform plan** command evaluates a Terraform configuration to determine the desired state of all the resources it declares, then compares that desired state to the real infrastructure objects being managed with the current working directory and workspace.

*The **terraform apply** command executes the actions proposed in a Terraform plan.The most straightforward way to use terraform apply is to run it without any arguments at all, in which case it will automatically create a new execution plan (as if you had run terraform plan) and then prompt you to approve that plan, before taking the indicated actions.Another way to use terraform apply is to pass it the filename of a saved plan file you created earlier with terraform plan -out=..., in which case Terraform will apply the changes in the plan without any confirmation prompt. This two-step workflow is primarily intended for when running Terraform in automation.

#terraform fmt

#terraform validate

```
[root@porali ec2]# terraform fmt
[root@porali ec2]# terraform validate
Success! The configuration is valid.

[root@porali ec2]#
```

#terraform plan

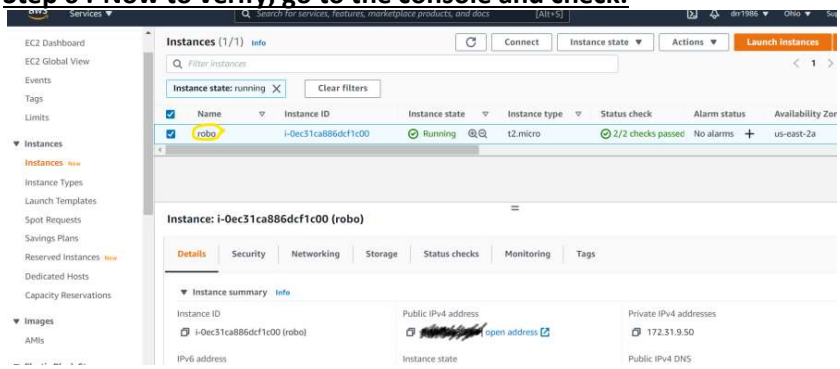
```
[root@porali ec2]# terraform plan

Terraform used the selected providers to generate the following
the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-00f8e2c955f7ffa9b"
      + 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)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (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                     = "drhiju86"
      + outpost_arn                  = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (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 apply)
      + secondary_private_ips        = (known after apply)
      + security_groups              = (known after apply)
      + source_dest_check            = true
      + subnet_id                    = (known after apply)
      + tags                         = {
          + "Name" = "robo"
        }
      + tenancy                      = (known after apply)
      + volume_tags                  = (known after apply)
      + vpc_security_group_ids       = [
          + "sg-b844c2cd",
        ]

      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
        }

      + root_block_device {
          + delete_on_termination = true
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = 8
          + volume_type           = (known after apply)
        }
    }

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

Note: You didn't use the -out option to save this plan,
you run "terraform apply" now.
```

Note:- before apply, make sure to verify with terraform plan. And see the plan output (add, change, destroy) incase if the value shown 0 means nothing going to happen, 1 means modification going to happened.

#terraform apply → will create the infr or instance according to resource file config.

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

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Creation complete after 23s [id=i-0ec31ca886dcf1c00]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[root@porali ec2]#
```

Step 6 :-Now to verify, go to the console and check.



*Terraform must store state about your managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.This state is stored by default in a local file named "**terraform.tfstate**", but it can also be stored remotely, which works better in a team environment.

```
[root@porali ec2]# ls -lrt
total 12
-rw-r--r-- 1 root root  203 Sep 30 01:04 providers.tf
-rw-r--r-- 1 root root  284 Sep 30 01:05 resource.tf
-rw-r--r-- 1 root root 3293 Sep 30 01:19 terraform.tfstate
[root@porali ec2]#
```

* The **terraform show** command is used to provide human-readable output from a state or plan file. This can be used to inspect a plan to ensure that the planned operations are expected, or to inspect the current state as Terraform sees it. Machine-readable output is generated by adding the -json command-line flag.
#terraform show

```
[root@porali ec2]# terraform show
# aws_instance.web:
resource "aws_instance" "web" {
    ami                          = "ami-00f8e2c955f7ffa9b"
    arn                          = "arn:aws:ec2:us-east-2:697683530102:instance/i-0ec31ca886dcf1c00"
    associate_public_ip_address  = true
    availability_zone            = "us-east-2a"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    hibernation                  = false
    id                           = "i-0ec31ca886dcf1c00"
    instance_state               = "running"
    instance_type                = "t2.micro"
    ipv6_address_count           = 0
    ipv6_addresses               = []
    key_name                     = "drhiju86"
    monitoring                   = false
    primary_network_interface_id = "eni-0fdf4a29e389bf7fc"
    private_dns                  = "ip-172-31-9-50.us-east-2.compute.internal"
    private_ip                   = "172.31.9.50"
    public_dns                   = "[redacted]2.compute.amazonaws.com"
    public_ip                    = "[redacted]"
    secondary_private_ips        = []
    security_groups              = [
        "default",
    ]
```

Note:- Make sure to destroy, incase instance is not in use.

*The **terraform destroy** command terminates resources managed by your Terraform project. This command is the inverse of terraform apply in that it terminates all the resources specified in your Terraform state. It does not destroy resources running elsewhere that are not managed by the current Terraform project.
#terraform destroy