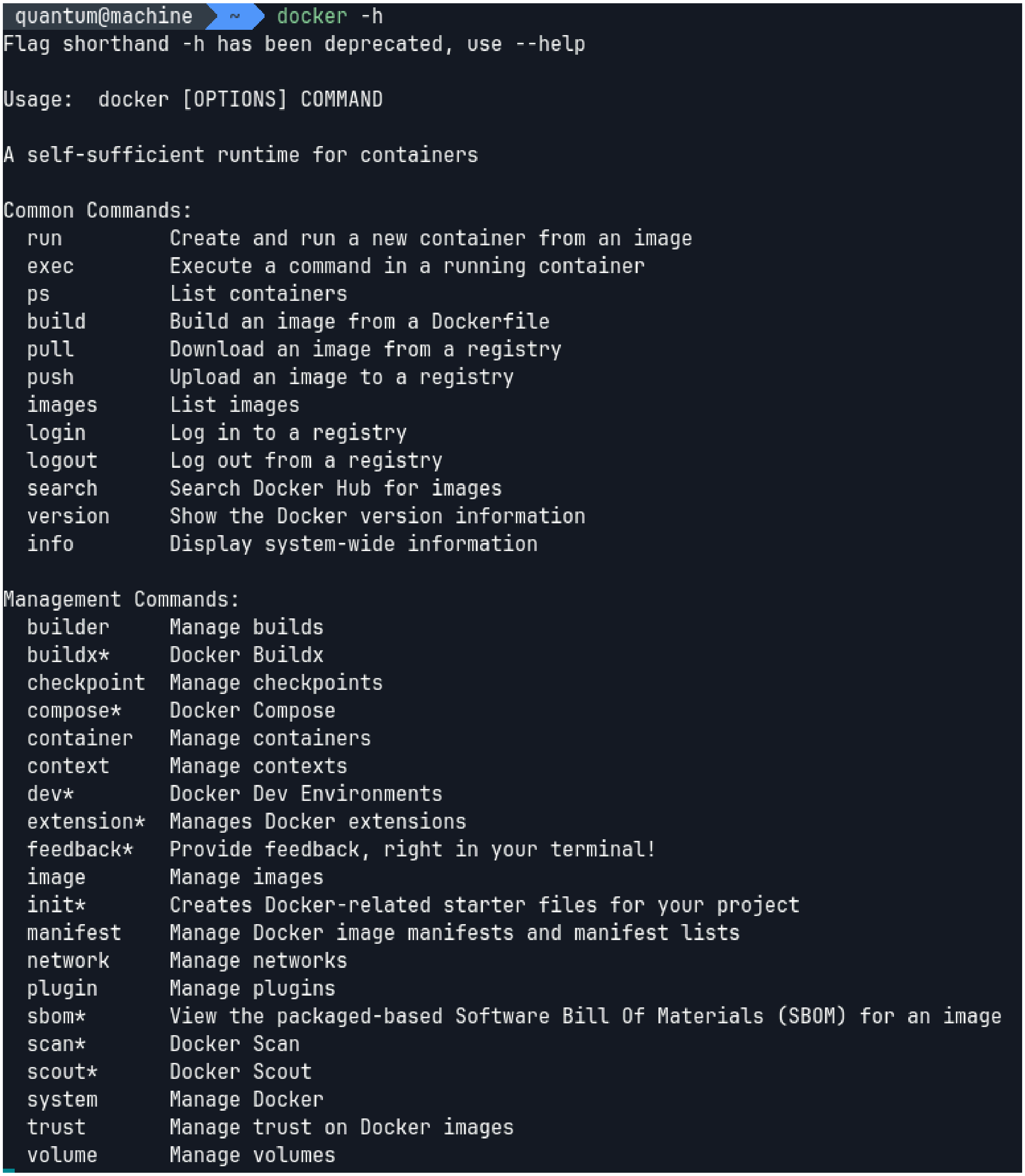
Name:Neeraj Rijhwani Roll no:64 Class:D15C

EXPERIMENT NO. 6 (A)

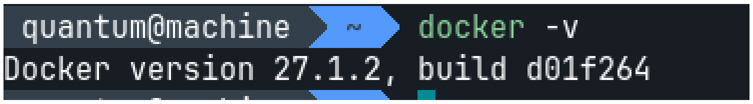
Prerequisite:
1) Download and Install Docker Desktop from https://www.docker.com/

Steps:

1. Check docker installation of looking at it’s help page

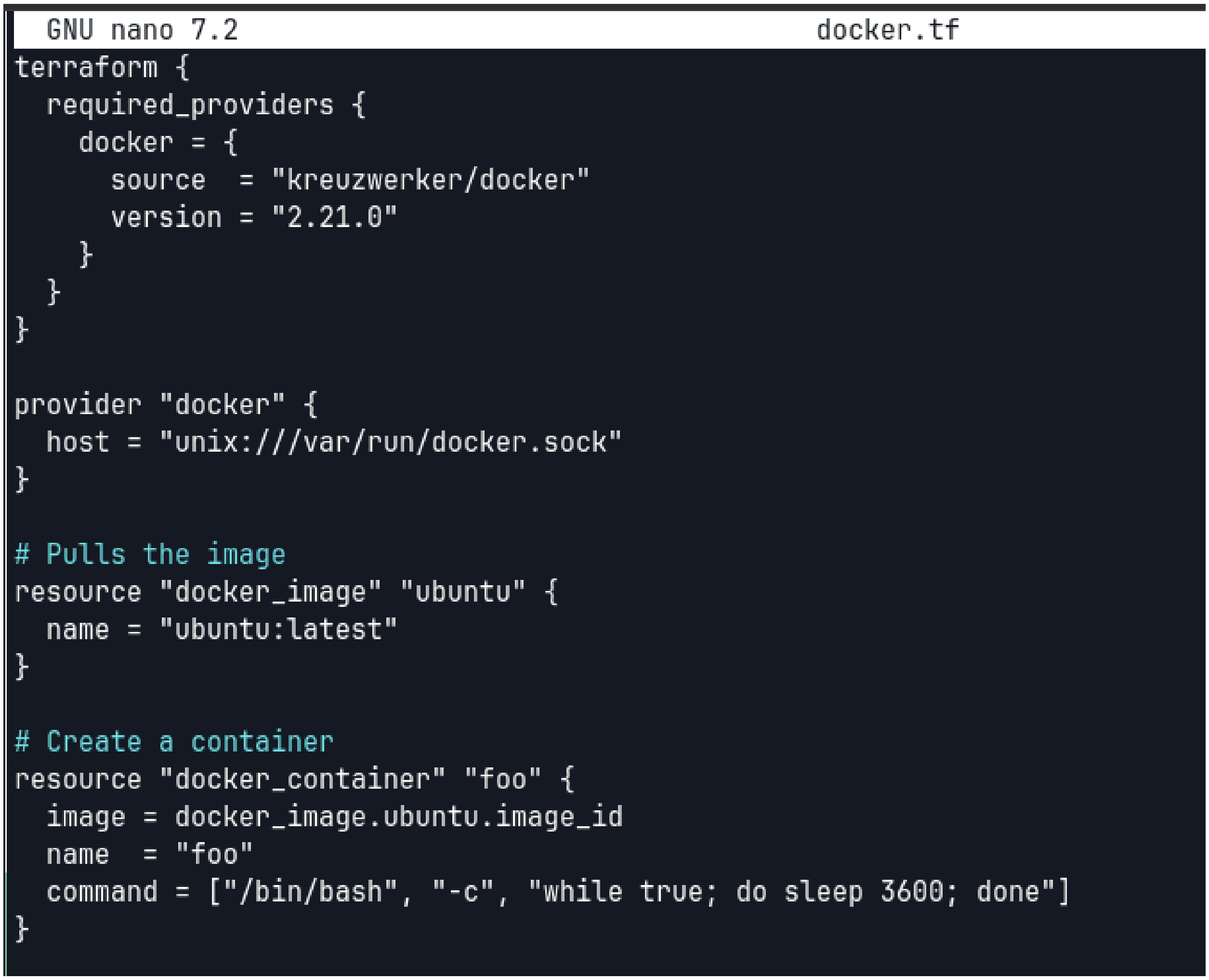


```
 quantum@machine  ~  docker -h
Flag shorthand -h has been deprecated, use --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Log in to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx
  checkpoint  Manage checkpoints
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  dev*        Docker Dev Environments
  extension*  Manages Docker extensions
  feedback*   Provide feedback, right in your terminal!
  image       Manage images
  init*       Creates Docker-related starter files for your project
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  sbom*       View the packaged-based Software Bill Of Materials (SBOM) for an image
  scan*       Docker Scan
  scout*      Docker Scout
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes
```



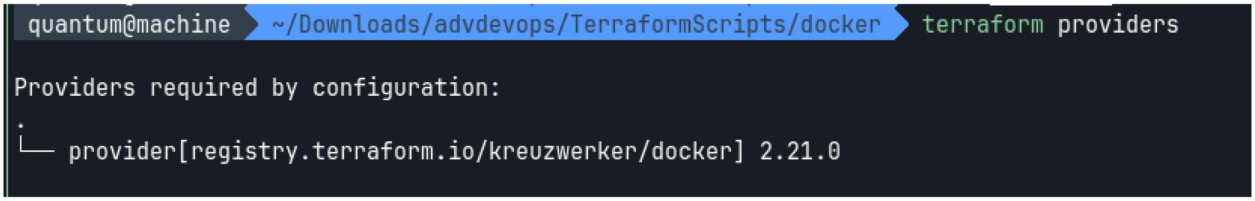
```
quantum@machine  ~  docker -v
Docker version 27.1.2, build d01f264
```

2. Create a new folder named **“Terraform Scripts”**, inside it create a new folder **docker** and create a new file named **docker.tf** with following contents inside it.



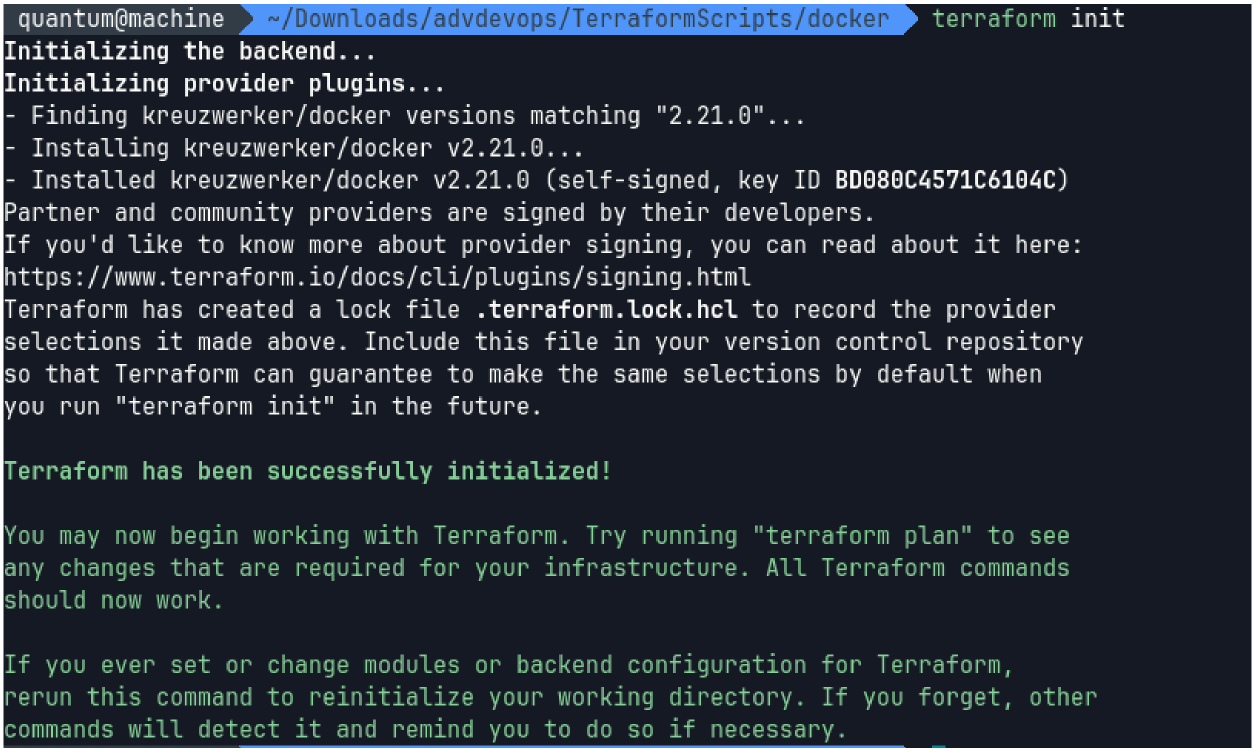
```
  GNU nano 7.2                                                docker.tf
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}

# Pulls the image
resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}

# Create a container
resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
  name  = "foo"
  command = ["/bin/bash", "-c", "while true; do sleep 3600; done"]
}
```

3. To see list of providers for current configuration file use command **terraform providers**



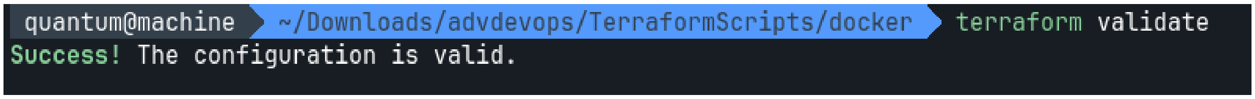
```
 quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  terraform providers

Providers required by configuration:
.
└── provider[registry.terraform.io/kreuzwerker/docker] 2.21.0
```

4. Execute command **terraform init** in the current directory



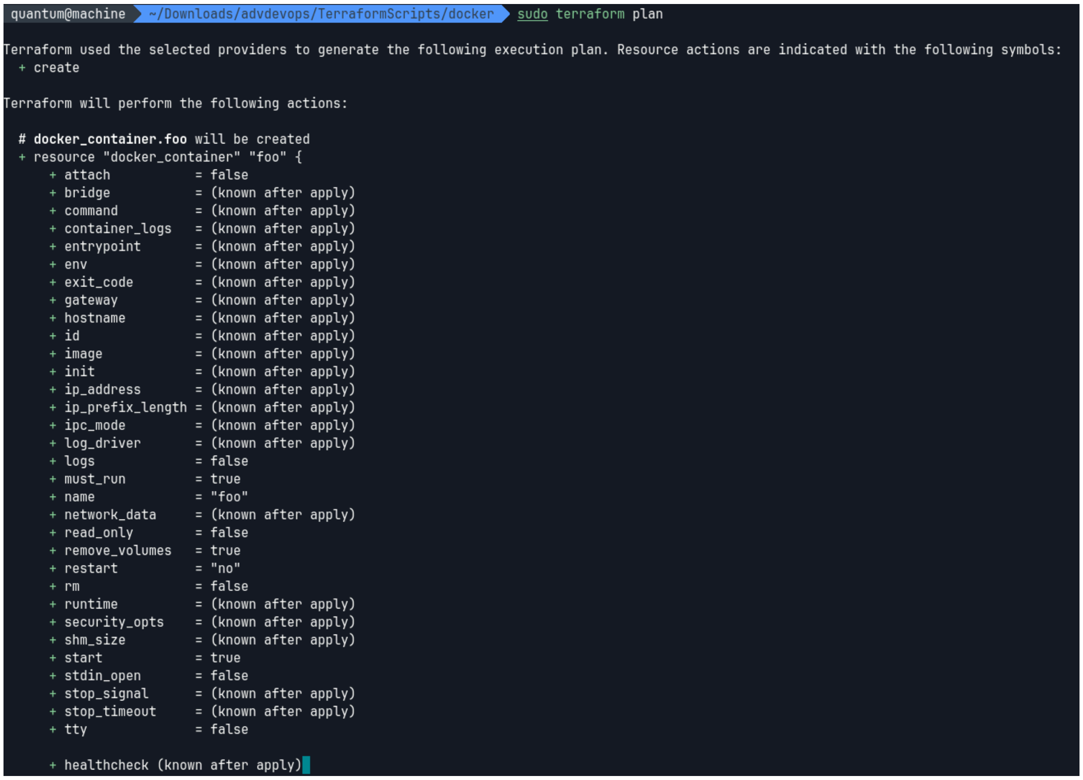
```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  terraform init
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "2.21.0"...
- Installing kreuzwerker/docker v2.21.0...
- Installed kreuzwerker/docker v2.21.0 (self-signed, key ID BD080C4571C6104C)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
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.

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.
```

5. Use command **terraform validate** to check for validation and syntax errors of config file

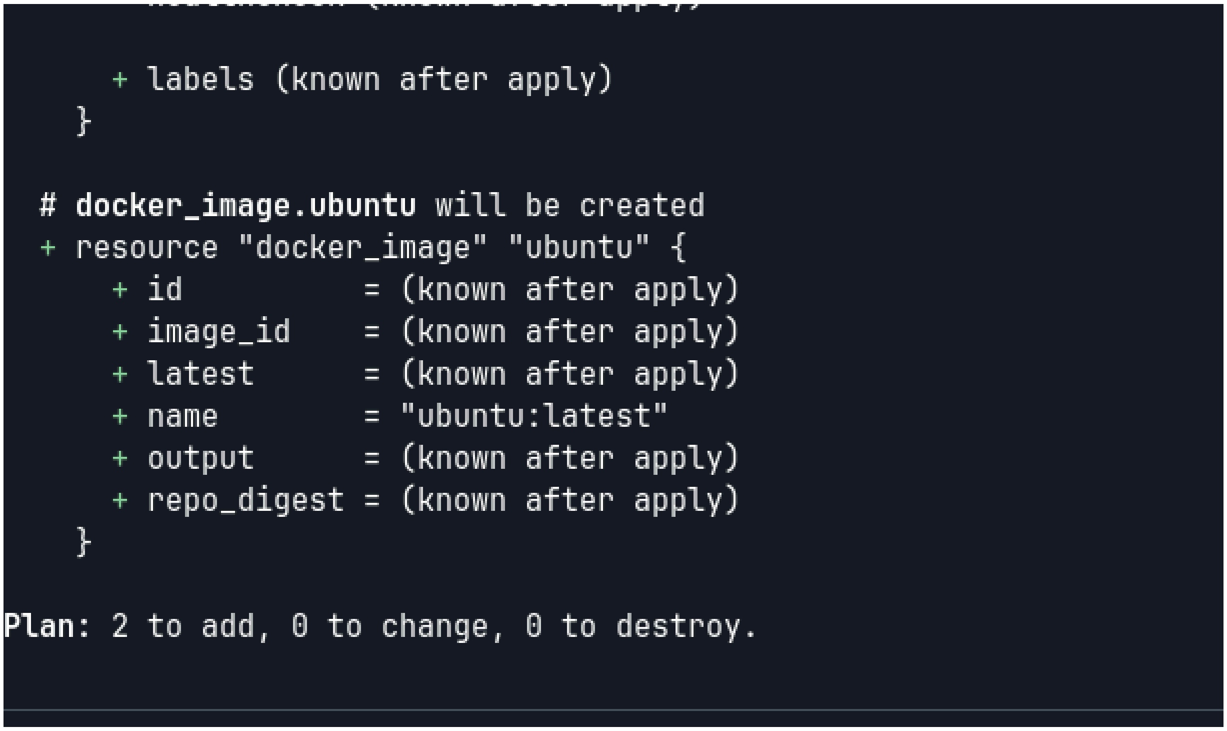


```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  terraform validate
Success! The configuration is valid.
```

5. Execute command **terraform plan** to see the changes that will be made

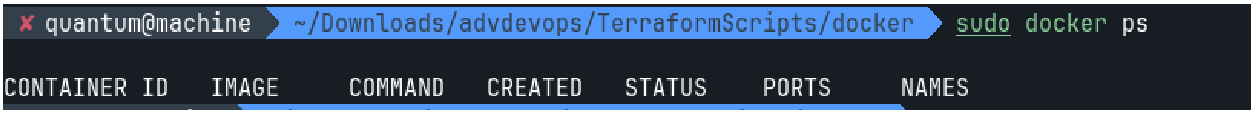


```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  sudo terraform plan

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:

  # docker_container.foo will be created
  + resource "docker_container" "foo" {
      + attach           = false
      + bridge           = (known after apply)
      + command          = (known after apply)
      + container_logs   = (known after apply)
      + entrypoint       = (known after apply)
      + env              = (known after apply)
      + exit_code        = (known after apply)
      + gateway          = (known after apply)
      + hostname         = (known after apply)
      + id               = (known after apply)
      + image            = (known after apply)
      + init             = (known after apply)
      + ip_address       = (known after apply)
      + ip_prefix_length = (known after apply)
      + ipc_mode         = (known after apply)
      + log_driver       = (known after apply)
      + logs             = false
      + must_run         = true
      + name             = "foo"
      + network_data     = (known after apply)
      + read_only        = false
      + remove_volumes   = true
      + restart          = "no"
      + rm               = false
      + runtime          = (known after apply)
      + security_opts    = (known after apply)
      + shm_size         = (known after apply)
      + start            = true
      + stdin_open       = false
      + stop_signal      = (known after apply)
      + stop_timeout     = (known after apply)
      + tty              = false

      + healthcheck (known after apply)
```



```
      + labels (known after apply)
    }

  # docker_image.ubuntu will be created
  + resource "docker_image" "ubuntu" {
      + id           = (known after apply)
      + image_id     = (known after apply)
      + latest       = (known after apply)
      + name         = "ubuntu:latest"
      + output       = (known after apply)
      + repo_digest  = (known after apply)
    }

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

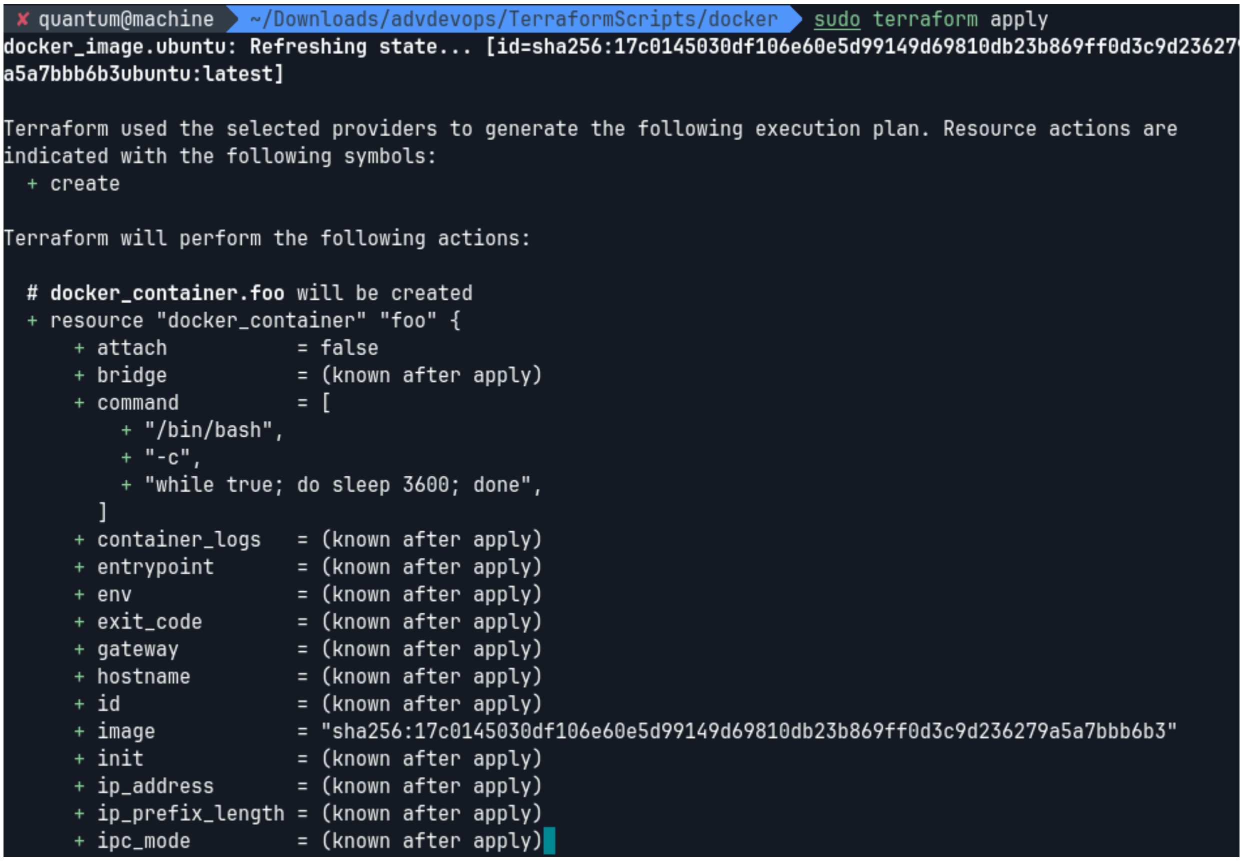
6. Check list of running docker containers using command **docker ps**



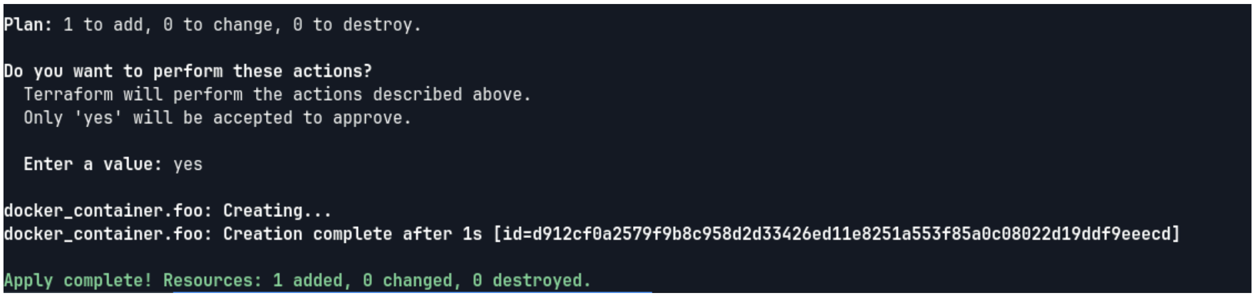
```
✘ quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

As we can see there are no active containers running

7. Execute **terraform apply** to apply configuration, which will automatically create and run the Ubuntu Linux container based on our configuration.

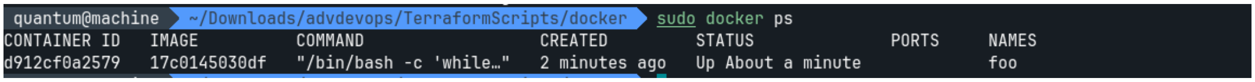


```
✘ quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  sudo terraform apply
docker_image.ubuntu: Refreshing state... [id=sha256:17c0145030df106e60e5d99149d69810db23b869ff0d3c9d23627
a5a7bbb6b3ubuntu:latest]

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:

  # docker_container.foo will be created
  + resource "docker_container" "foo" {
      + attach           = false
      + bridge           = (known after apply)
      + command          = [
          + "/bin/bash",
          + "-c",
          + "while true; do sleep 3600; done",
        ]
      + container_logs   = (known after apply)
      + entrypoint       = (known after apply)
      + env              = (known after apply)
      + exit_code        = (known after apply)
      + gateway          = (known after apply)
      + hostname         = (known after apply)
      + id               = (known after apply)
      + image            = "sha256:17c0145030df106e60e5d99149d69810db23b869ff0d3c9d236279a5a7bbb6b3"
      + init             = (known after apply)
      + ip_address       = (known after apply)
      + ip_prefix_length = (known after apply)
      + ipc_mode         = (known after apply)
```



```
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

docker_container.foo: Creating...
docker_container.foo: Creation complete after 1s [id=d912cf0a2579f9b8c958d2d33426ed11e8251a553f85a0c08022d19ddf9eeecd]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

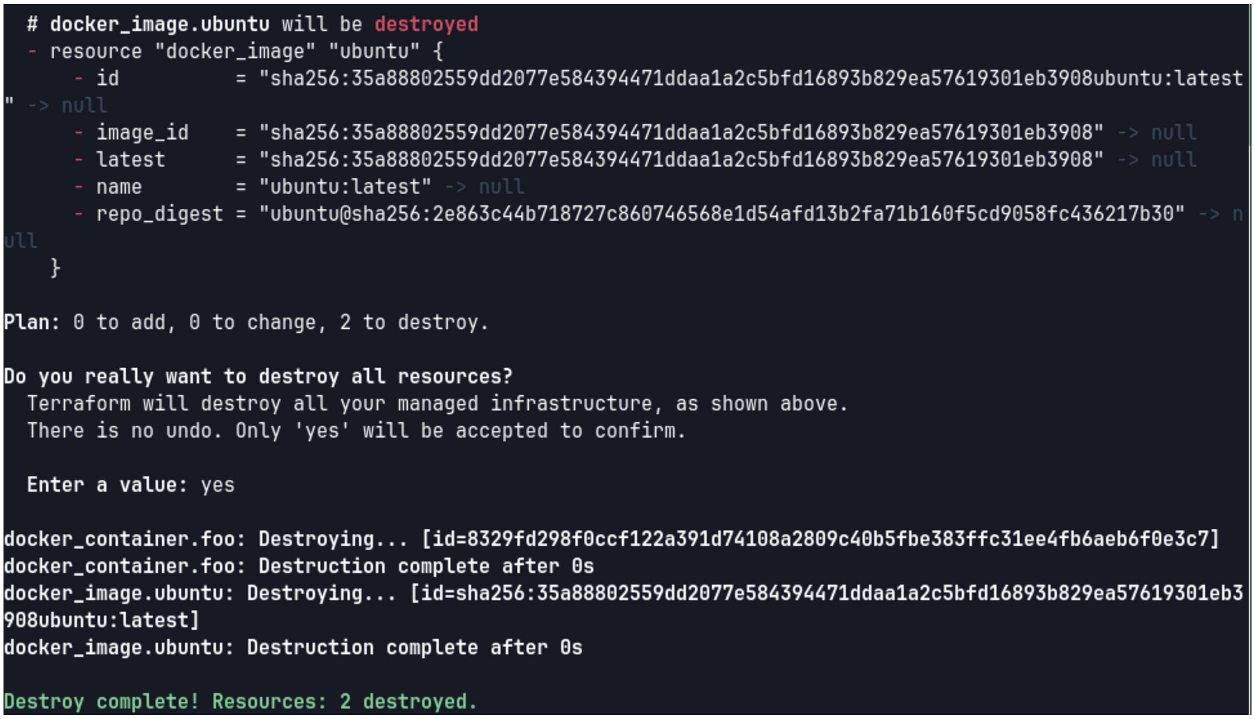
8. After executing **terraform apply**



```
 quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS               PORTS     NAMES
d912cf0a2579   17c0145030df   "/bin/bash -c 'while…"   2 minutes ago   Up About a minute              foo
```

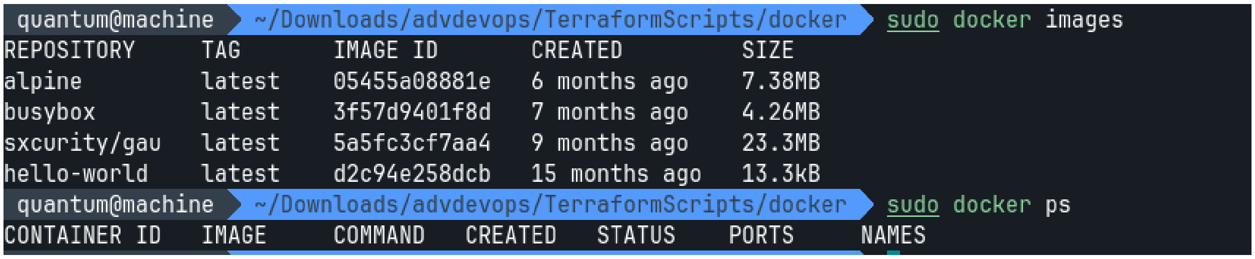
After execution, a new docker image and container will be created

9. Execute **terraform destroy** to delete the configuration, which will automatically delete the Container.



```
  # docker_image.ubuntu will be destroyed
  - resource "docker_image" "ubuntu" {
      - id          = "sha256:35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301eb3908ubuntu:latest
" -> null
      - image_id    = "sha256:35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301eb3908" -> null
      - latest      = "sha256:35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301eb3908" -> null
      - name        = "ubuntu:latest" -> null
      - repo_digest = "ubuntu@sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5cd9058fc436217b30" -> n
ull
    }

Plan: 0 to add, 0 to change, 2 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

docker_container.foo: Destroying... [id=8329fd298f0ccf122a391d74108a2809c40b5fbe383ffc31ee4fb6aeb6f0e3c7]
docker_container.foo: Destruction complete after 0s
docker_image.ubuntu: Destroying... [id=sha256:35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301eb3
908ubuntu:latest]
docker_image.ubuntu: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
```

10. Check whether docker image and container is removed or not



```
 quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  sudo docker images
REPOSITORY     TAG       IMAGE ID       CREATED         SIZE
alpine         latest    05455a08881e   6 months ago    7.38MB
busybox        latest    3f57d9401f8d   7 months ago    4.26MB
sxcurity/gau   latest    5a5fc3cf7aa4   9 months ago    23.3MB
hello-world    latest    d2c94e258dcb   15 months ago   13.3kB
 quantum@machine  ~/Downloads/advdevops/TerraformScripts/docker  sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

As we can see both containers and images are removed

EXPERIMENT NO. 6 (B)

Prerequisite:

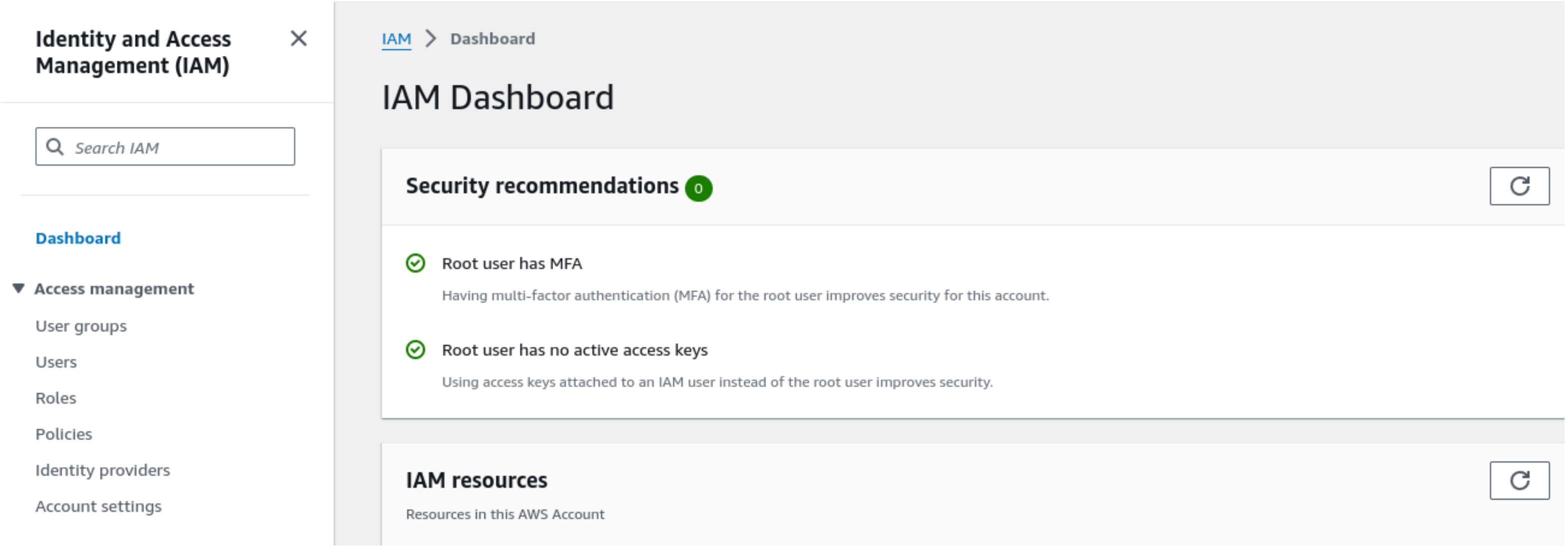
1) Any text editor to write and save scripts
2) Must have an AWS Access Key ID and Secret Access Key

Step 1: Write a Terraform Script in Atom for creating S3 Bucket on Amazon AWS

Ensure your bucket name is globally unique

Step 2: Get secret key and access key from AWS account

1. Go to IAM dashboard



IAM Dashboard

Security recommendations 0

Root user has MFA

Having multi-factor authentication (MFA) for the root user improves security for this account.

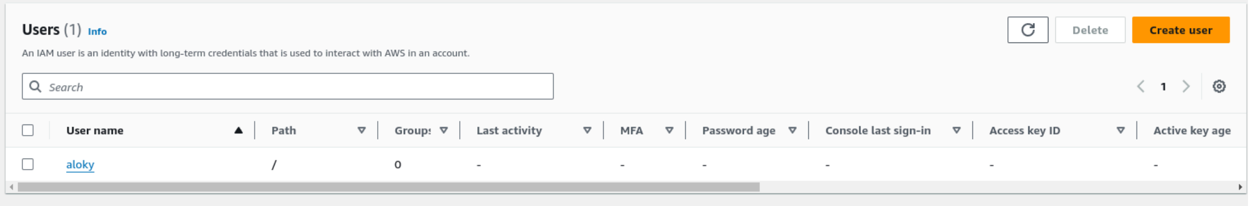
Root user has no active access keys

Using access keys attached to an IAM user instead of the root user improves security.

IAM resources

Resources in this AWS Account

oll Go to Users and select anNo:59 existingiv user if it exists or create a new one. I'm selecting existing one as i had already created before

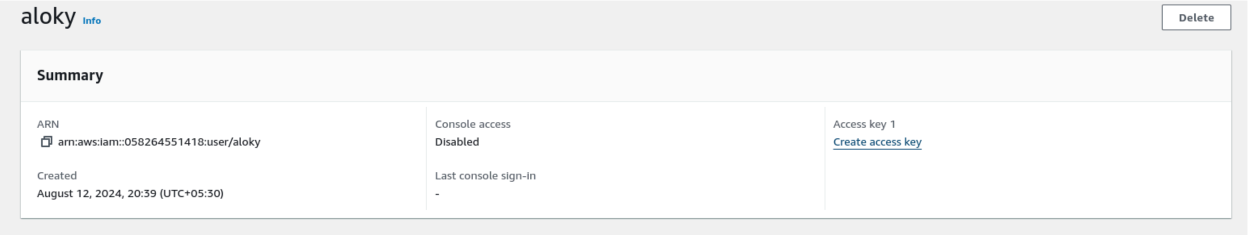


Users (1)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

□	User name ▲	Path	Group:	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age
□	aloky	/	0	-	-	-	-	-	-

3. On the user dashboard, click on **create access key**



aloky

Summary

ARN
arn:aws:iam::058264551418:user/aloky

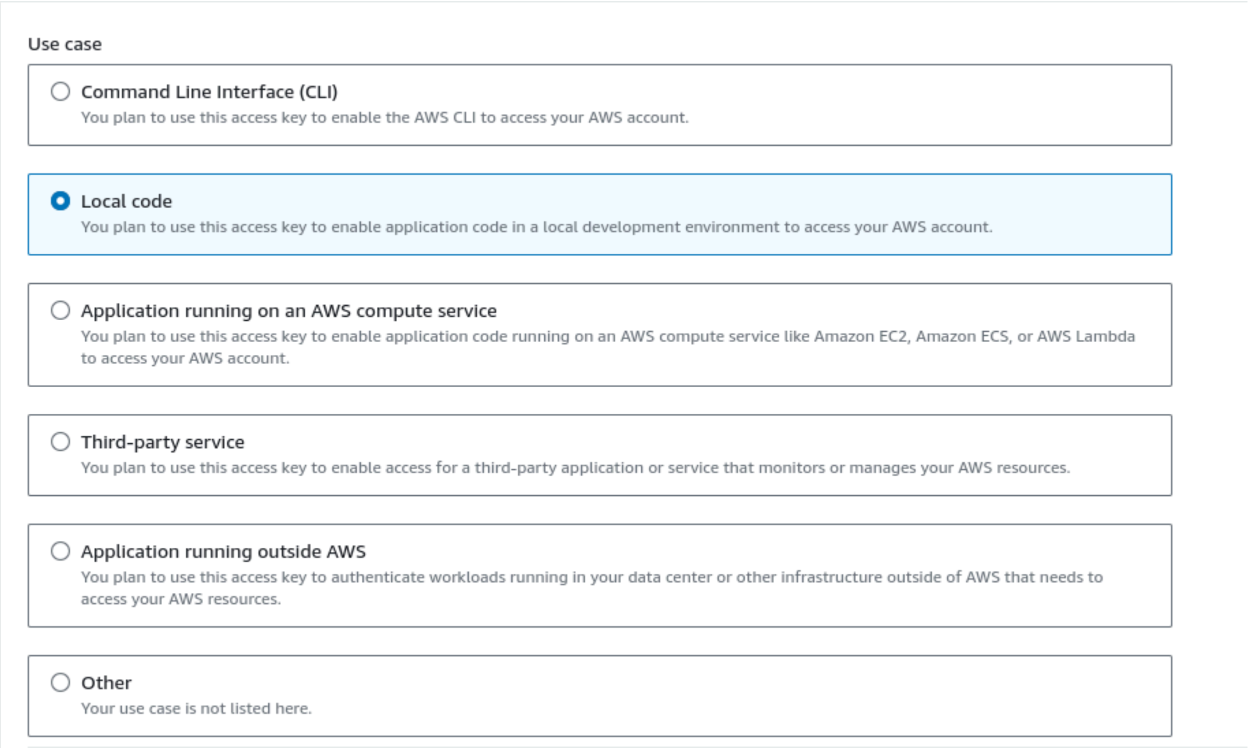
Created
August 12, 2024, 20:39 (UTC+05:30)

Console access
Disabled

Last console sign-in
-

Access key 1
Create access key

4. Select **Use case** as **Local Code** then Create access key



Use case

○ Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.

● Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.

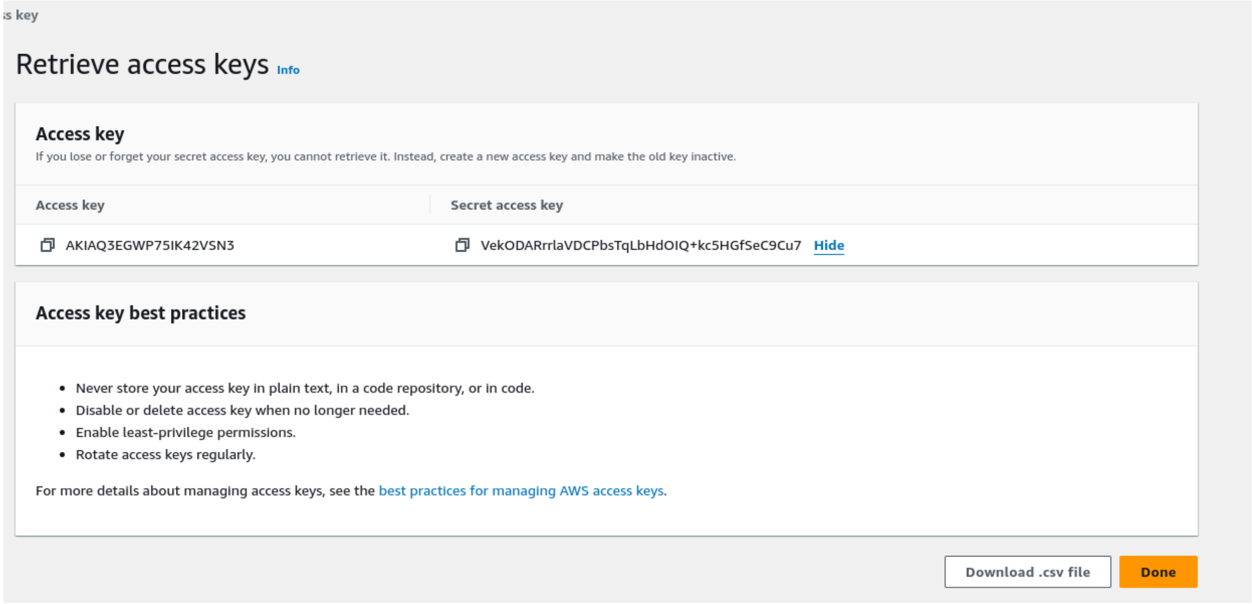
○ Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

○ Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

○ Application running outside AWS
You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.

○ Other
Your use case is not listed here.

5. After all above steps, access key is generated



s key

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

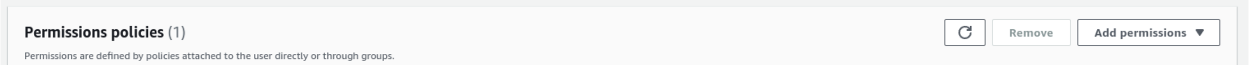
Access key	Secret access key
AKIAQ3EGWP75IK42VSN3	VekODARrrlaVDCPbsTqLbHdOIQ+kc5HGfSeC9Cu7 Hide

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the best practices for managing AWS access keys.

6. Ensure newly created user has necessary permissions to edit and create S3 buckets to do so, click on **Add permissions**

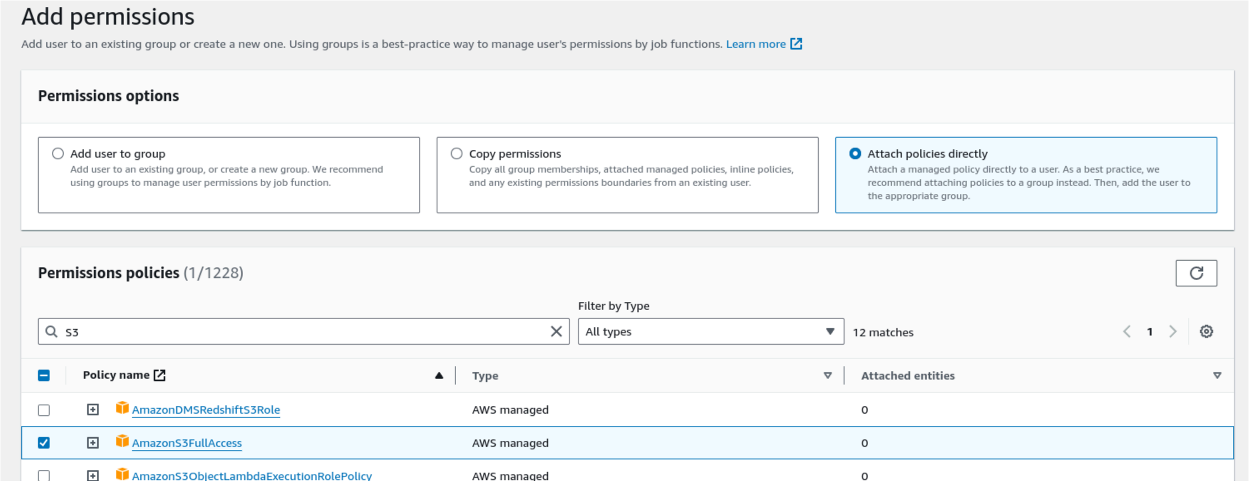


Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.

7. Select permission option as **Attach policies directly** and select **AmazonS3FullAccess** as permission policy from the list

\



Add permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, inline policies, and any existing permissions boundaries from an existing user.

● Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

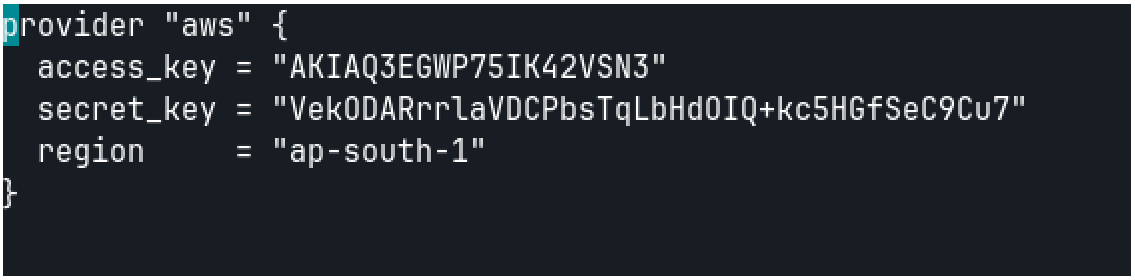
Permissions policies (1/1228)

Filter by Type

S3 | All types ▼ | 12 matches | 1

	Policy name	Type	Attached entities
☐	⊞ AmazonDMSRedshiftS3Role	AWS managed	0
☑	⊞ AmazonS3FullAccess	AWS managed	0
☐	⊞ AmazonS3ObjectLambdaExecutionRolePolicy	AWS managed	0

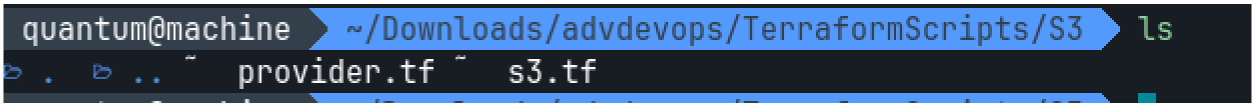
Step 3: Create a new provider.tf file and write the following contents into it.



```
provider "aws" {
  access_key = "AKIAQ3EGWP75IK42VSN3"
  secret_key = "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"
  region     = "ap-south-1"
}
```

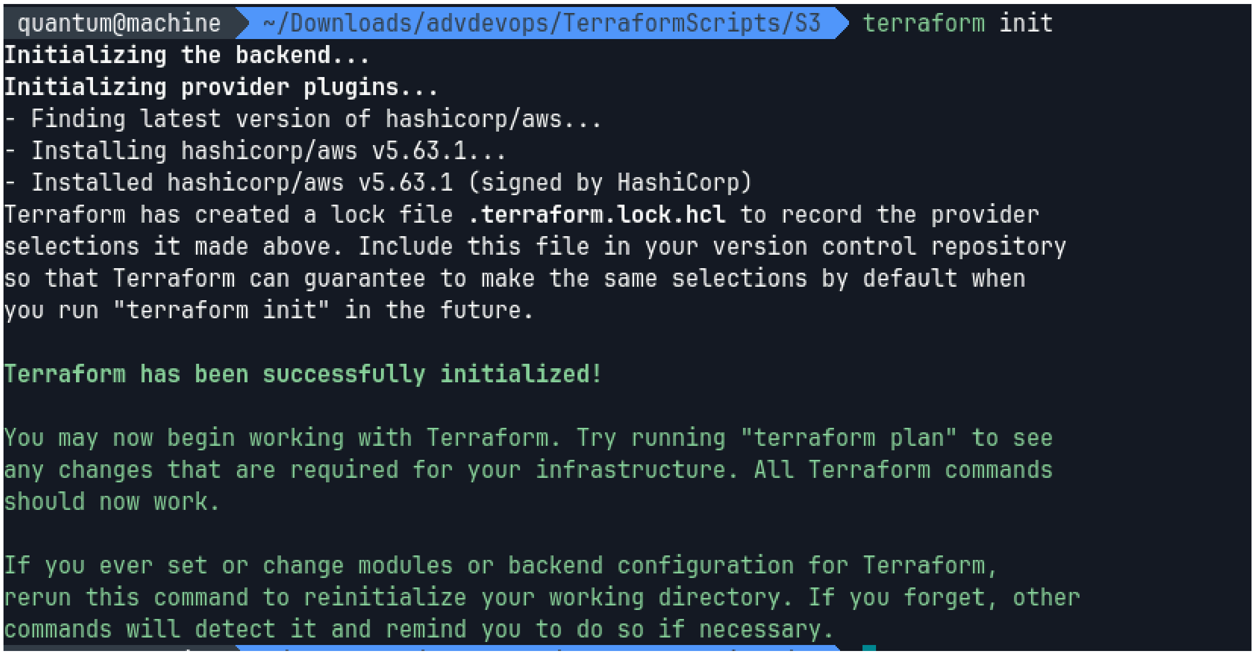
Save both the files in same directory TerraformScripts/S3

Step 4: Open terminal ang go to TerraformScripts/S3 directory where our .tf files are stored



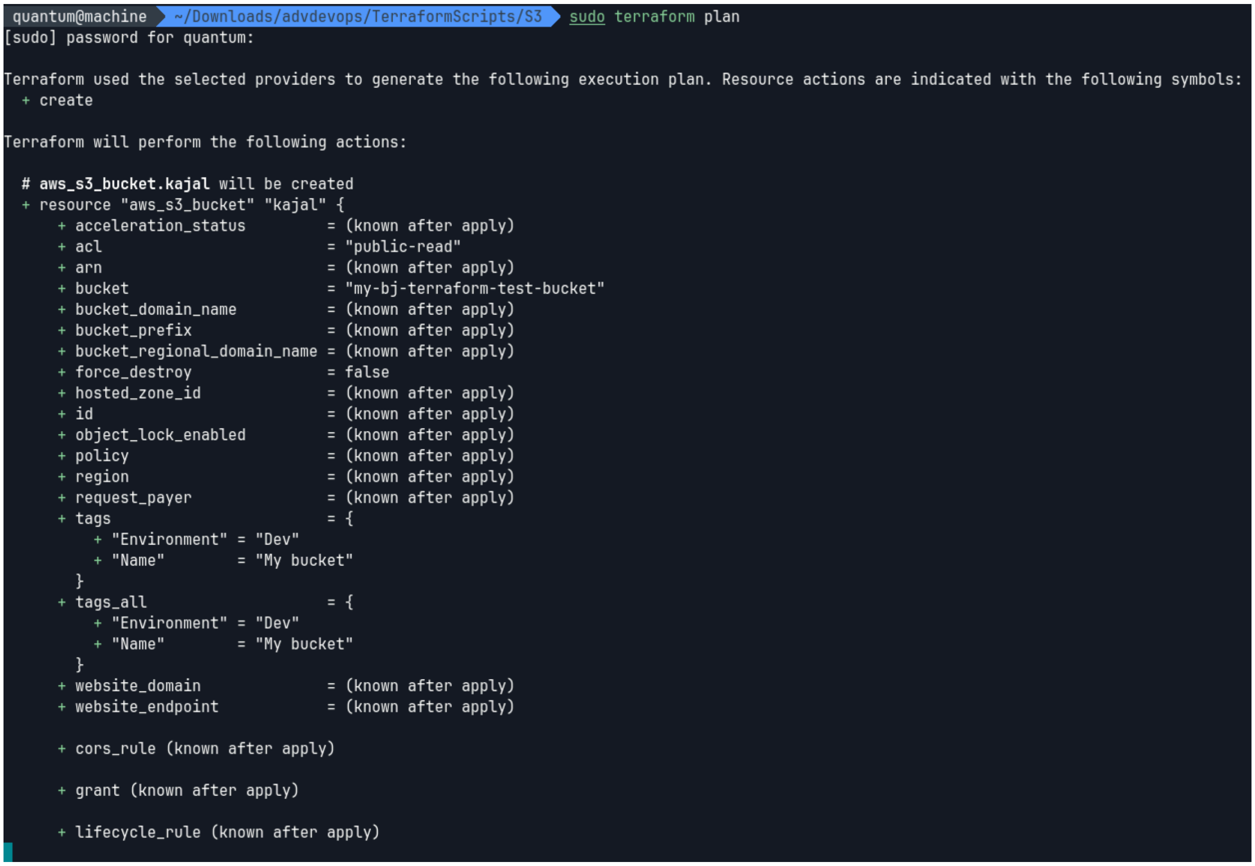
```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/S3  ls
🗁 .  🗁 ..    provider.tf    s3.tf
```

Step 5: Execute **terraform init** to initialize the resources

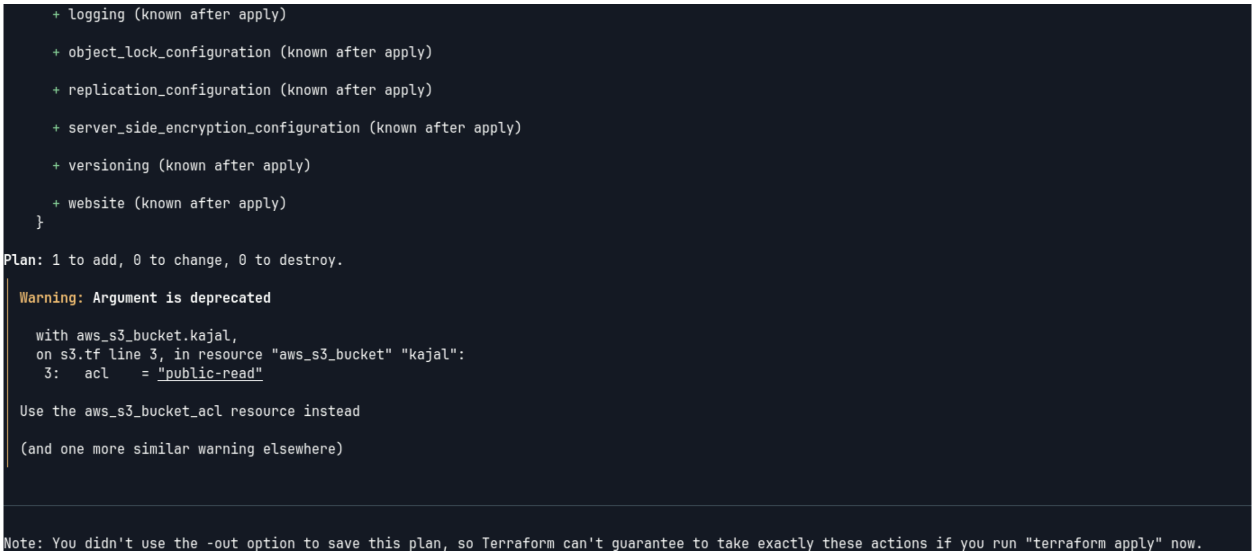


```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/S3  terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.63.1...
- Installed hashicorp/aws v5.63.1 (signed by HashiCorp)
Terraform has created 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.

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.
```

Step 6: Execute Terraform plan to see the available resources

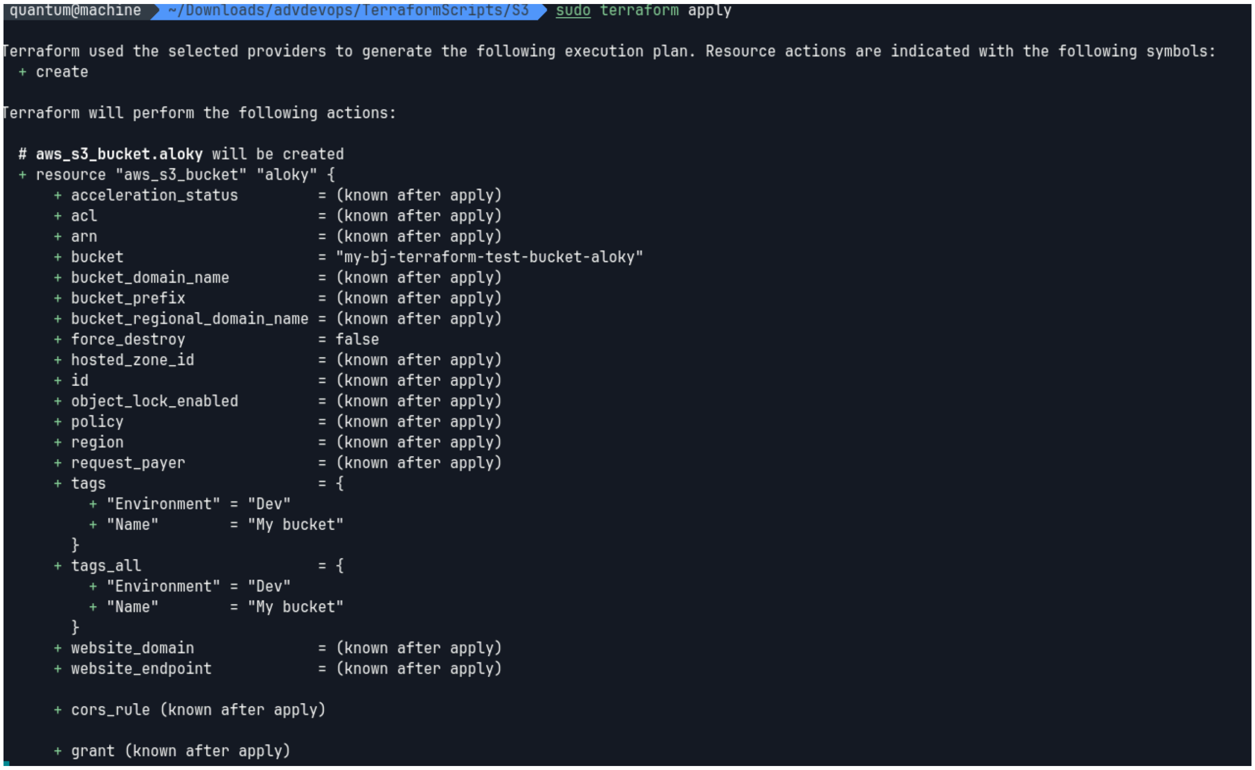


```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/S3  sudo terraform plan
[sudo] password for quantum:

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_s3_bucket.kajal will be created
  + resource "aws_s3_bucket" "kajal" {
      + acceleration_status         = (known after apply)
      + acl                         = "public-read"
      + arn                         = (known after apply)
      + bucket                      = "my-bj-terraform-test-bucket"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Environment" = "Dev"
          + "Name"        = "My bucket"
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "My bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + cors_rule (known after apply)

      + grant (known after apply)

      + lifecycle_rule (known after apply)
```

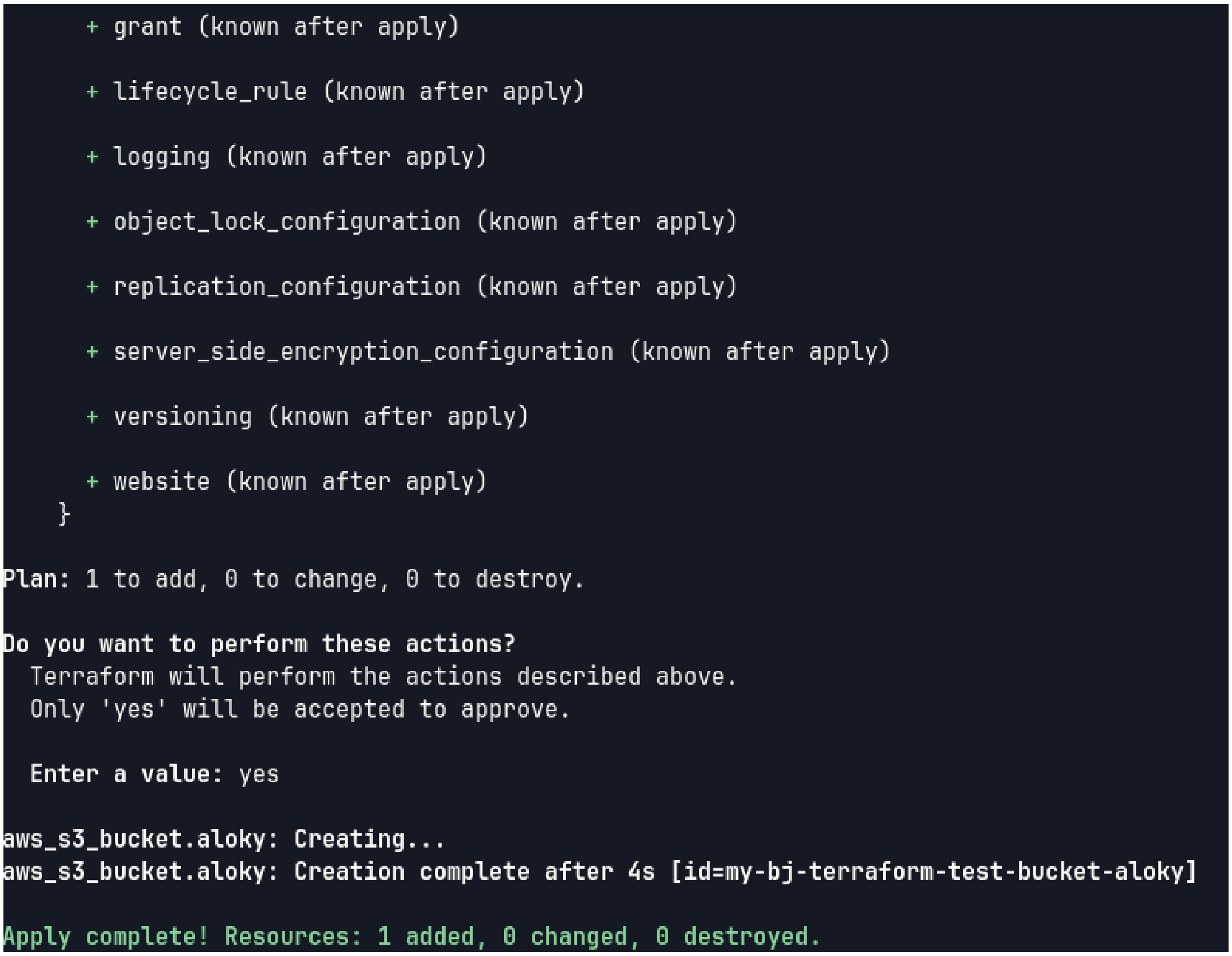


```
      + logging (known after apply)

      + object_lock_configuration (known after apply)

      + replication_configuration (known after apply)

      + server_side_encryption_configuration (known after apply)

      + versioning (known after apply)

      + website (known after apply)
    }

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

│ Warning: Argument is deprecated
│
│   with aws_s3_bucket.kajal,
│   on s3.tf line 3, in resource "aws_s3_bucket" "kajal":
│    3:   acl    = "public-read"
│
│ Use the aws_s3_bucket_acl resource instead
│
│ (and one more similar warning elsewhere)

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

Step 7: Execute **terraform apply** to apply the configuration, which will automatically create an S3 bucket based on our configuration.

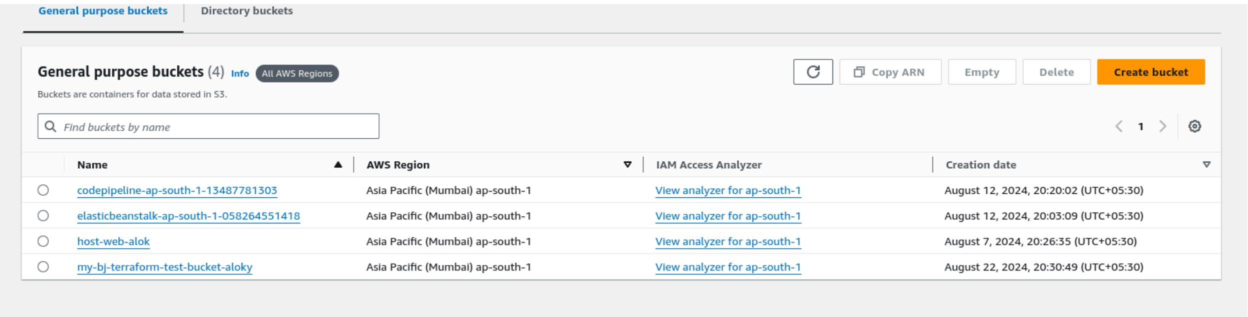


```
quantum@machine  ~/Downloads/advdevops/TerraformScripts/S3  sudo terraform apply

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_s3_bucket.aloky will be created
  + resource "aws_s3_bucket" "aloky" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = "my-bj-terraform-test-bucket-aloky"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Environment" = "Dev"
          + "Name"        = "My bucket"
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "My bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + cors_rule (known after apply)

      + grant (known after apply)
```



```
      + grant (known after apply)

      + lifecycle_rule (known after apply)

      + logging (known after apply)

      + object_lock_configuration (known after apply)

      + replication_configuration (known after apply)

      + server_side_encryption_configuration (known after apply)

      + versioning (known after apply)

      + website (known after apply)
    }

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_s3_bucket.aloky: Creating...
aws_s3_bucket.aloky: Creation complete after 4s [id=my-bj-terraform-test-bucket-aloky]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

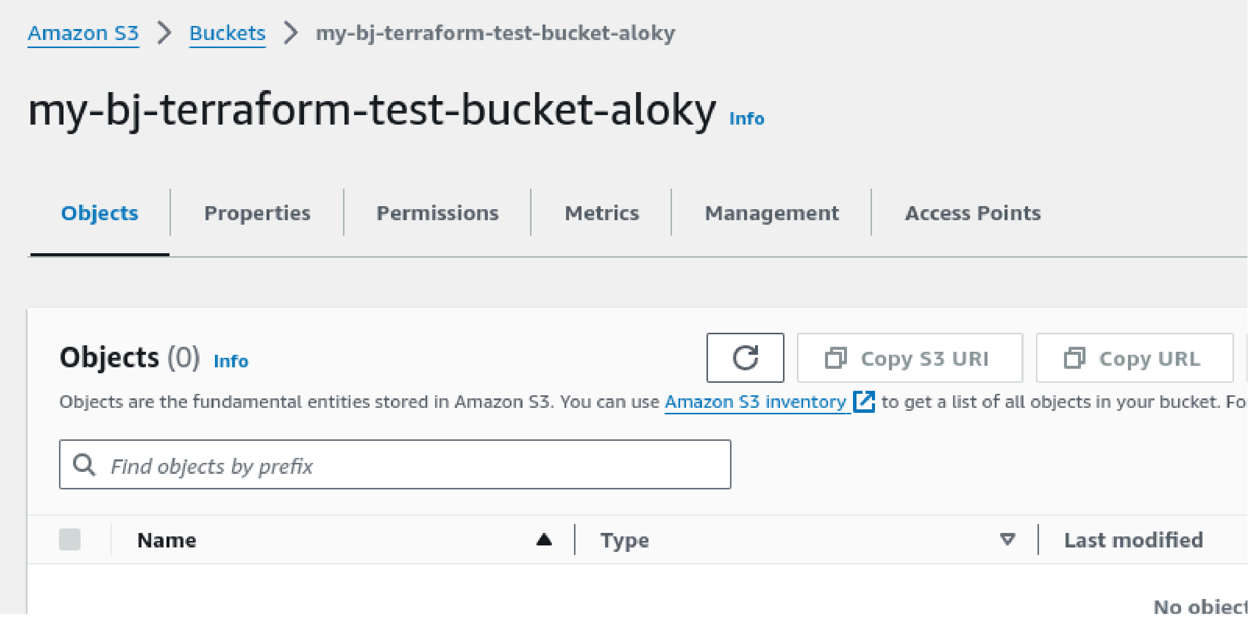
Step 8: Go to **S3 Buckets** dashboard to see newly created S3 bucket using terraform



General purpose buckets (4)

Buckets are containers for data stored in S3.

	Name	AWS Region	IAM Access Analyzer	Creation date
○	codepipeline-ap-south-1-13487781303	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	August 12, 2024, 20:20:02 (UTC+05:30)
○	elasticbeanstalk-ap-south-1-058264551418	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	August 12, 2024, 20:03:09 (UTC+05:30)
○	host-web-alok	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	August 7, 2024, 20:26:35 (UTC+05:30)
○	my-bj-terraform-test-bucket-aloky	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	August 22, 2024, 20:30:49 (UTC+05:30)



my-bj-terraform-test-bucket-aloky

Objects (0) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. Fo

No object