DevOps Project to automate infrastructure on AWS using Terraform and GitLab CICD

Before commencing the project, make sure you have a basic understanding of the following topics, as they will simplify the implementation process.

Basic Terraform Knowledge (resource)
Understanding of CICD (resource)
GitLab CI Knowledge (resource)

PREREQUISITES:

1) Aws account creation

Check out the official site to create aws account Here

2) GitLab account

- ✓ Login to https://gitlab.com
- ✓ You can sign in via GitHub/Gmail
- ✓ Verify email and phone
- ✓ fill up the questionaries
- ✓ provide group name & project name as per your choice

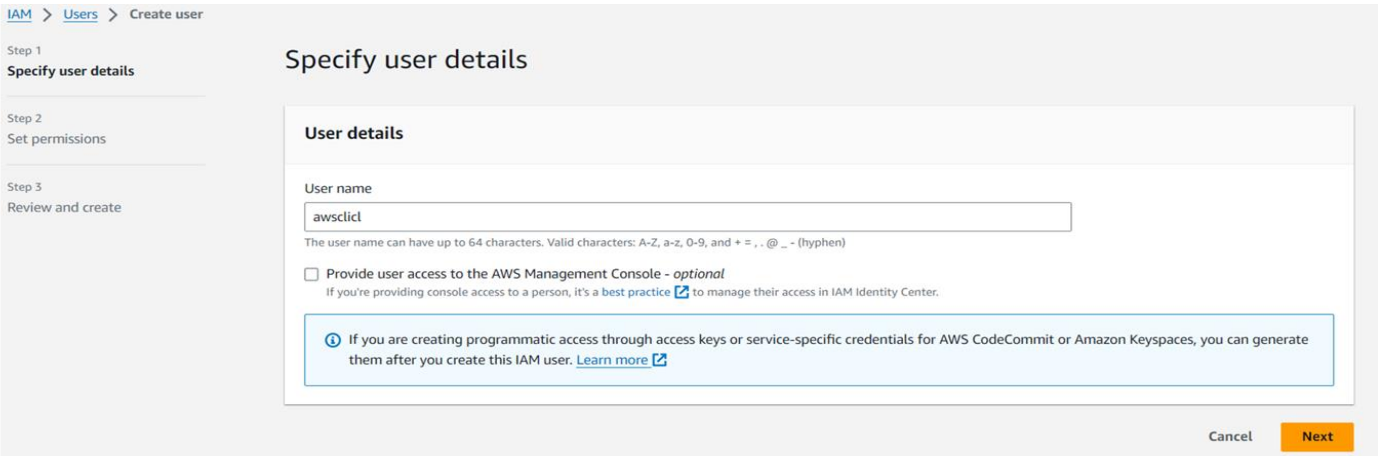
3) Terraform Installed

Check out the official website to install terraform Here

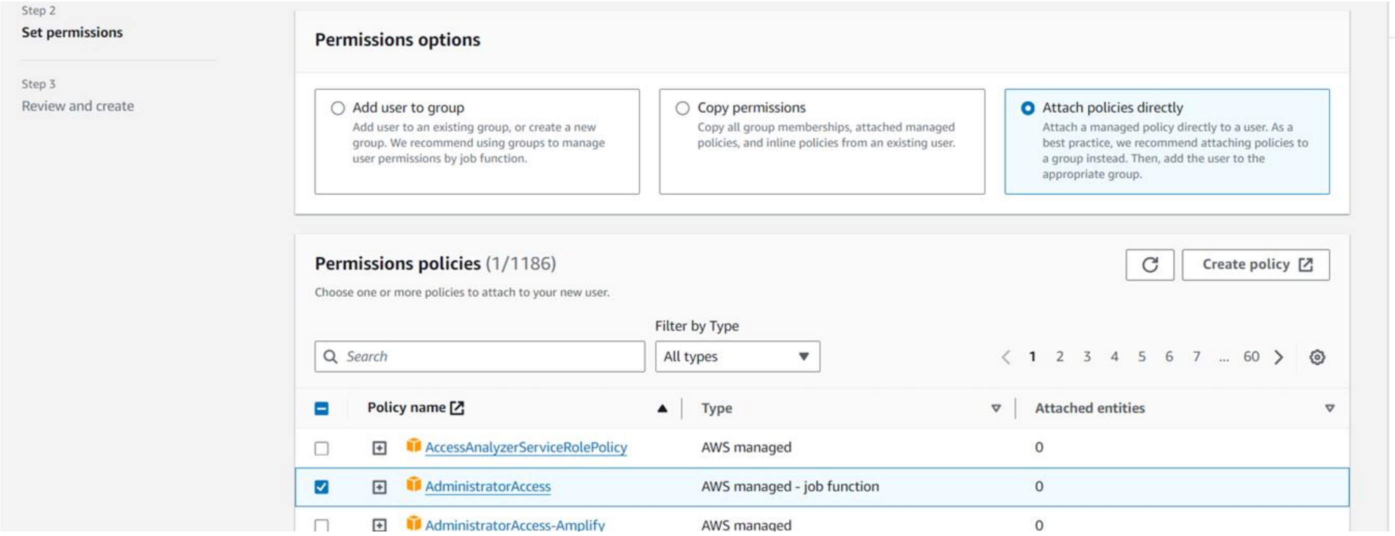
```
ubuntu@ip-172-31-25-121:~$ terraform --version
Terraform v1.9.1
on linux_amd64
ubuntu@ip-172-31-25-121:~$
```

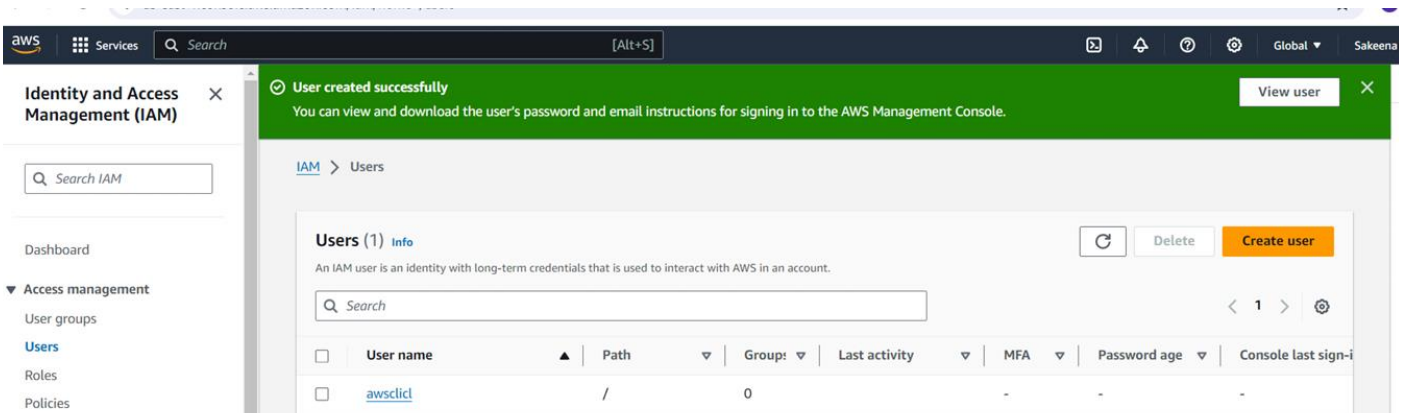
4) AWS CLI Installed

Navigate to the IAM dashboard on AWS, then select "Users." Enter the username and proceed to the next step

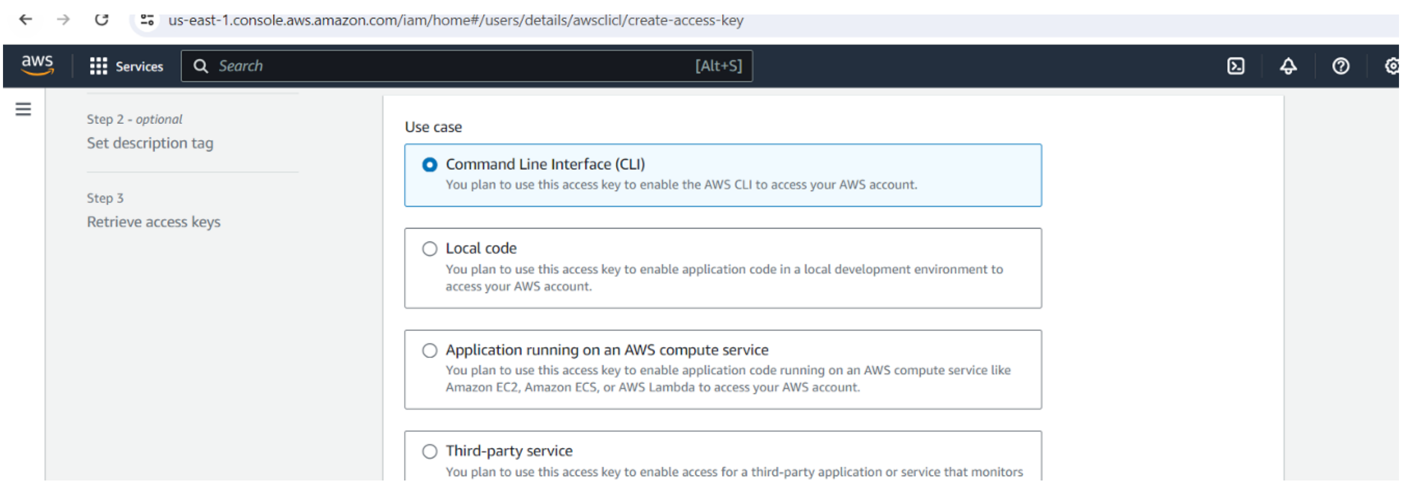


Assign permissions by attaching policies directly, opting for "Administrator access," and then create the user.

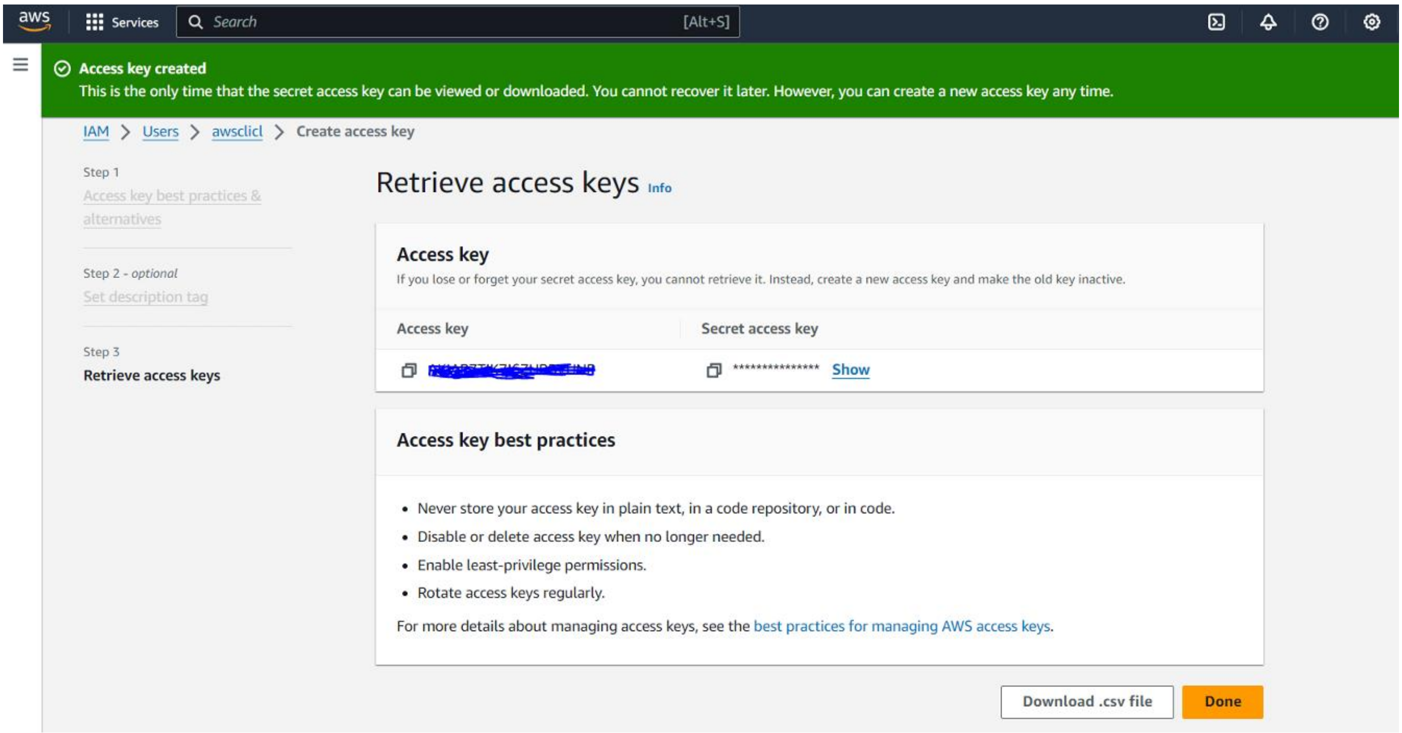




Within the user settings, locate "Create access key," and choose the command line interface (CLI) option to generate an access key.



Upon creation, you can view or download the access key and secret access key either from the console or via CSV download.



Now go to your terminal and follow below steps:

sudo apt install unzip
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws configure (input created accesskeyid and secret access key)
cat ~/.aws/config
cat ~/.aws/credentials
aws iam list-users (to list all IAM users in an AWS account)

5) Code editor (Vscode)

Download it from [Here]

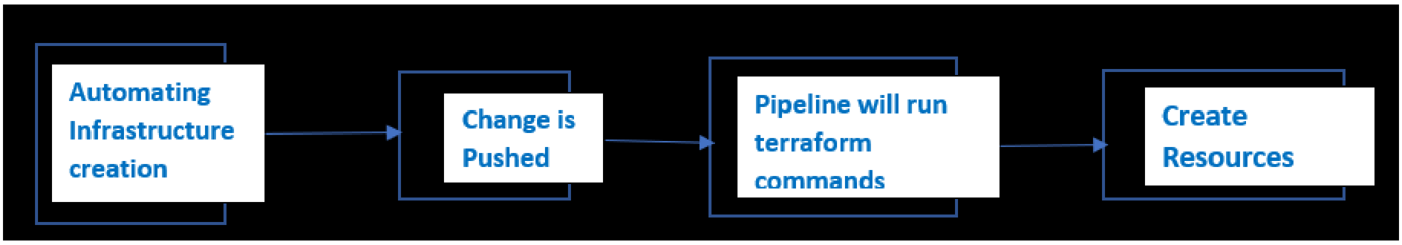
Let's begin with the project. This project is divided in to two parts.

Part1:

Here, we write terraform code, run terraform commands and create infrastructure manually to ensure everything works fine before automating

Part2:

Create CICD pipeline script on Gitlab to automate terraform resource creation



Step1: Create a new folder named "cicdtf" and open it in vscode to start writing the code.

```
ubuntu@ip-172-31-25-121:~$ mkdir cicdtf
ubuntu@ip-172-31-25-121:~$ ls
aws  awscliv2.zip  cicdtf
ubuntu@ip-172-31-25-121:~$ cd cicdtf/
ubuntu@ip-172-31-25-121:~/cicdtf$ code .
```

Step2: We will start writing our Terraform code in the "cicdtf" folder. The first step in writing Terraform code is to define a provider. To do this, we will create a file called provider.tf with the following content:

```
provider  "aws" {
    region = "us-east-1"
}

# This snippet configures Terraform to use the AWS provider
# and specifies that resources should be created in the us-east-1 region.
```

We will be deploying a VPC, a security group, a subnet and an EC2 instance as part of the initial phase.

The folder structure is as follows:

1. VPC Module (vpc folder):

Files:

main.tf: Defines resources like VPC, subnets, and security groups.
variables.tf: Declares input variables for customization.
outputs.tf: Specifies outputs like VPC ID, subnet IDs, etc.

2.EC2 Module (web folder):

Files:

main.tf: Configures EC2 instance details, including AMI, instance type, and security groups.
variables.tf: Defines variables needed for EC2 instance customization.
outputs.tf: Outputs instance details like public IP, instance ID, etc.

Snap of folder structure:

Let’s start with defining vpc,

The below Terraform script (main.tf) sets up an AWS Virtual Private Cloud (VPC) with a CIDR block of 10.0.0.0/16, enabling DNS support and hostnames. It creates a subnet (10.0.1.0/24) in us-east-1a with public IP mapping. Additionally, it establishes a security group allowing inbound SSH (port 22) traffic from any IP address and permitting all outbound traffic from the instances within the VPC.

To know more about modules and different parameters being used in this project, check out the official documentation of Terraform [Here]

Make use of below repositories to check out the code.

https://gitlab.com/N4si/cicdtf

https://gitlab.com/Sakeena19/cicdtf

```
# Define an AWS VPC resource named "myvpc"
resource "aws_vpc" "myvpc" {
    cidr_block           = "10.0.0.0/16"          # Define the CIDR block for the VPC
    enable_dns_hostnames = true                   # Enable DNS hostnames in the VPC
    enable_dns_support   = true                   # Enable DNS support in the VPC

    tags = {
        Name = "myvpc"                            # Set a tag for the VPC resource
    }
}

# Define an AWS subnet resource named "pb_sn"
resource "aws_subnet" "pb_sn" {
    vpc_id                  = aws_vpc.myvpc.id    # Reference the VPC ID from the "myvpc" resource
    cidr_block              = "10.0.1.0/24"       # Define the CIDR block for the subnet
    map_public_ip_on_launch = true                # Enable automatic assignment of public IPs to instances
    availability_zone       = "us-east-1a"        # Specify the availability zone for the subnet

    tags = {
        Name = "pb_sn1"                           # Set a tag for the subnet resource
    }
}

# Define an AWS security group resource named "sg"
resource "aws_security_group" "sg" {
    vpc_id      = aws_vpc.myvpc.id                # Reference the VPC ID from the "myvpc" resource
    name        = "my_sg"                         # Specify the name for the security group
    description = "Public Security Group"         # Provide a description for the security group

    # Define an ingress rule allowing inbound traffic on port 22 (SSH) from any IP address
    ingress {
        from_port   = 22                          # Specify the starting port for inbound traffic
        to_port     = 22                          # Specify the ending port for inbound traffic
        protocol    = "tcp"                       # Specify the protocol (TCP in this case)
        cidr_blocks = ["0.0.0.0/0"]               # Allow inbound traffic from any IP address
    }

    # Define an egress rule allowing all outbound traffic (any port, any protocol) to any IP address
    egress {
        from_port   = 0                           # Specify the starting port for outbound traffic
        to_port     = 0                           # Specify the ending port for outbound traffic
        protocol    = "-1"                        # Specify all protocols for outbound traffic
        cidr_blocks = ["0.0.0.0/0"]               # Allow outbound traffic to any IP address
    }
}
```

Step3:

we will create an EC2 instance in the web module and use the security group and subnet defined in the VPC module. This demonstrates how to share values between different modules in Terraform.

Main Module (Root Module): The main.tf file acts as the parent module
Child Modules: The VPC and web modules are child modules.

To share values from one child module to another, we follow these steps:

Define Outputs: Specify the values (e.g., subnet ID, security group ID) as outputs in the VPC module.
Use Variables: Reference these outputs as variables in the web module.

The script in main.tf file in web module is as follows:

```
# Define an AWS EC2 instance resource named "server" # main.tf file in web module


resource "aws_instance" "server" {
    ami             = "ami-04a81a99f5ec58529"   # Specify the AMI ID (Amazon Machine Image) for the instance
    instance_type   = "t2.micro"                # Specify the instance type (e.g., t2.micro)
    subnet_id       = var.sn                    # Use the subnet ID variable from the VPC module's output
    security_groups = [var.sg]                  # Use the security group ID variable from the VPC module's output

    tags = {
        Name = "my_server"                     # Set a tag for the EC2 instance for identification
    }
}
```

Step4: define outputs.tf file in vpc module.

```
# outputs.tf file in vpc module

#Output the subnet ID
output "pb_sn" {
    value = aws_subnet.pb_sn.id  # This refers to the ID of the subnet created in the VPC module
}

# Output the security group ID
output "sg" {
    value = aws_security_group.sg.id  # This refers to the ID of the security group created in the VPC module
}

```

output "pb_sn": Defines an output variable named pb_sn.
value = aws_subnet.pb_sn.id: This line assigns the ID of the subnet resource (aws_subnet.pb_sn) to the output variable. This allows other modules to access the subnet ID. Similar for security group as well.

Step5: Define variables.tf file in web module.

```
# variables.tf file in web module

# Define a variable to hold the security group ID
variable "sg" {
    description = "The ID of the security group"
}

# Define a variable to hold the subnet ID
variable "sn" {
    description = "The ID of the subnet"
}
```

These variables are used to pass the security group ID and subnet ID from the VPC module to the web module.

Step6: Now to start using these modules, we have to define both vpc and web in the root module(main.tf) as shown below.

```
#root

module "vpc" {
    source = "./vpc"
}

module "ec2" {
    source = "./web"
    sn =  module.vpc.pb_sn
    sg =  module.vpc.sg

}
```

source = "./vpc": Specifies the path to the VPC module directory. This imports the VPC module defined in the ./vpc folder.
source = "./web": Specifies the path to the web module directory. This imports the EC2 module defined in the ./web folder.
sn = module.vpc.pb_sn: Passes the subnet ID output (pb_sn) from the VPC module to the EC2 module, assigning it to the variable sn.
sg = module.vpc.sg: Passes the security group ID output (sg) from the VPC module to the EC2 module, assigning it to the variable sg.

Step7:

Now to check whether the code is working fine, lets run terraform commands. Make sure to connect aws with terraform (using aws configure) before running and save all the files if not done already.

To initialize terraform, use "terraform init" command which setups everything necessary for terraform to manage your infrastructure such as modules, plugins, backend config etc., as defined in your configuration files.

To check if our code is valid, use "terraform validate" command.

Run "terraform plan" command is used to create an execution plan to see what changes terraform will make to your infrastructure without actually applying those changes.

In below snap shown, it's going to create 4 components: vpc, ec2 instance , subnet and security group will be created.

```
ubuntu@ip-172-31-25-53:~/cicdtf$ terraform init
Initializing the backend...
Initializing modules...
- ec2 in web
- vpc in vpc
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.57.0...
- Installed hashicorp/aws v5.57.0 (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.
ubuntu@ip-172-31-25-53:~/cicdtf$
```

```
ubuntu@ip-172-31-25-53:~/cicdtf$ terraform validate
Success! The configuration is valid.

ubuntu@ip-172-31-25-53:~/cicdtf$ 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:

  # module.ec2.aws_instance.server will be created
  + resource "aws_instance" "server" {
      + ami                                  = "ami-04a81a99f5ec58529"
      + 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)
```

```
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = true
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "myvpc"
        }
      + tags_all                             = {
          + "Name" = "myvpc"
        }
    }

Plan: 4 to add, 0 to change, 0 to destroy.

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

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"
```

You can also run by checking "terraform apply -auto-approve" command which executes the terraform plan without requiring interactive communication and proceeds with deployment.

```
ubuntu@ip-172-31-25-53:~/cicdtf$ terraform apply -auto-approve

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:

  # module.ec2.aws_instance.server will be created
  + resource "aws_instance" "server" {
      + ami                                  = "ami-04a81a99f5ec58529"
      + 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)
      + 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)
```

```
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "myvpc"
        }
      + tags_all                             = {
          + "Name" = "myvpc"
        }
    }

Plan: 4 to add, 0 to change, 0 to destroy.
module.vpc.aws_vpc.myvpc: Creating...
module.vpc.aws_vpc.myvpc: Still creating... [10s elapsed]
module.vpc.aws_vpc.myvpc: Creation complete after 12s [id=vpc-0d6cd7cbe2e2ee71b]
module.vpc.aws_subnet.pb_sn: Creating...
module.vpc.aws_security_group.sg: Creating...
module.vpc.aws_security_group.sg: Creation complete after 3s [id=sg-0d133baae578874f6]
module.vpc.aws_subnet.pb_sn: Still creating... [10s elapsed]
module.vpc.aws_subnet.pb_sn: Creation complete after 11s [id=subnet-0fd237c24b4c931aa]
module.ec2.aws_instance.server: Creating...
module.ec2.aws_instance.server: Still creating... [10s elapsed]
module.ec2.aws_instance.server: Still creating... [20s elapsed]
module.ec2.aws_instance.server: Still creating... [30s elapsed]
module.ec2.aws_instance.server: Creation complete after 32s [id=i-0314991e88264715d]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
```

When we run apply, terraform. tfstate file will be created which is not a good practise to have it in local machine, we will setup backend in later steps to store on S3 using DynamoDB.

Also it will create vpc, subnet and ec2 instance as well which can be verified in your aws console.

Instances (2) Info — Connect | Instance state | Actions | Launch instances

Find Instance by attribute or tag (case-sensitive) — All states

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
terraform_project	i-075cf72513ff0d349	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a
my_server	i-0314991e88264715d	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a

Now that the code is working fine locally, we'll configure a backend on S3, push the code to GitLab, and proceed with the second part of the project: setting up a CI/CD pipeline to automate the infrastructure deployment tasks we previously performed manually.

before this, delete everything using “terraform destroy -auto-approve” to proceed with automation.

```
ubuntu@ip-172-31-25-53:~/cicdtf$ terraform destroy -auto-approve
module.vpc.aws_vpc.myvpc: Refreshing state... [id=vpc-0d6cd7cbe2e2ee71b]
module.vpc.aws_subnet.pb_sn: Refreshing state... [id=subnet-0fd237c24b4c931aa]
module.vpc.aws_security_group.sg: Refreshing state... [id=sg-0d133baae578874f6]
module.ec2.aws_instance.server: Refreshing state... [id=i-0314991e88264715d]

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:

  # module.ec2.aws_instance.server will be destroyed
  - resource "aws_instance" "server" {
      - ami                          = "ami-04a81a99f5ec58529" -> null
```

Step8: Set up a backend using S3 and Dynamo DB.

Follow below video or documentation mentioned which has a complete process on how to setup S3 bucket and DynamoDB in detail.

https://developer.hashicorp.com/terraform/language/settings/backends/s3

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket

https://youtu.be/o04xfWEouKM?si=OGNj1c9R2iqe9TOM

The code for creating s3 and DynamoDB is as follows:

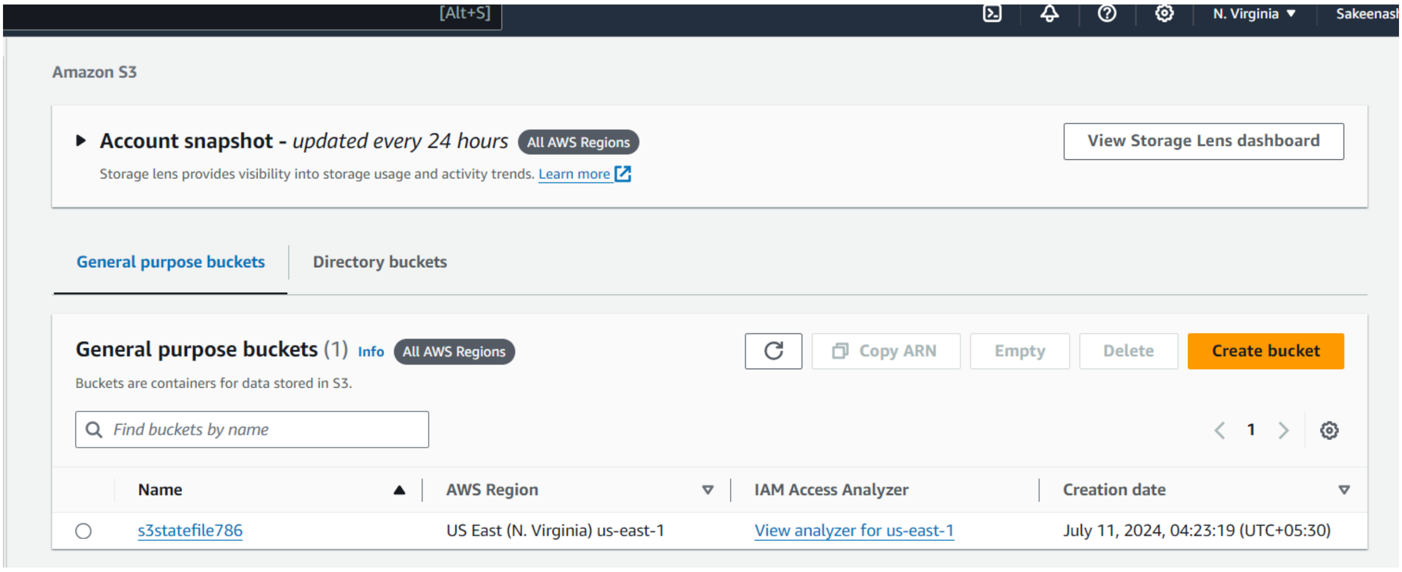
Once the code has been written in a file, run below terraform commands to create s3 and DynamoDB table.

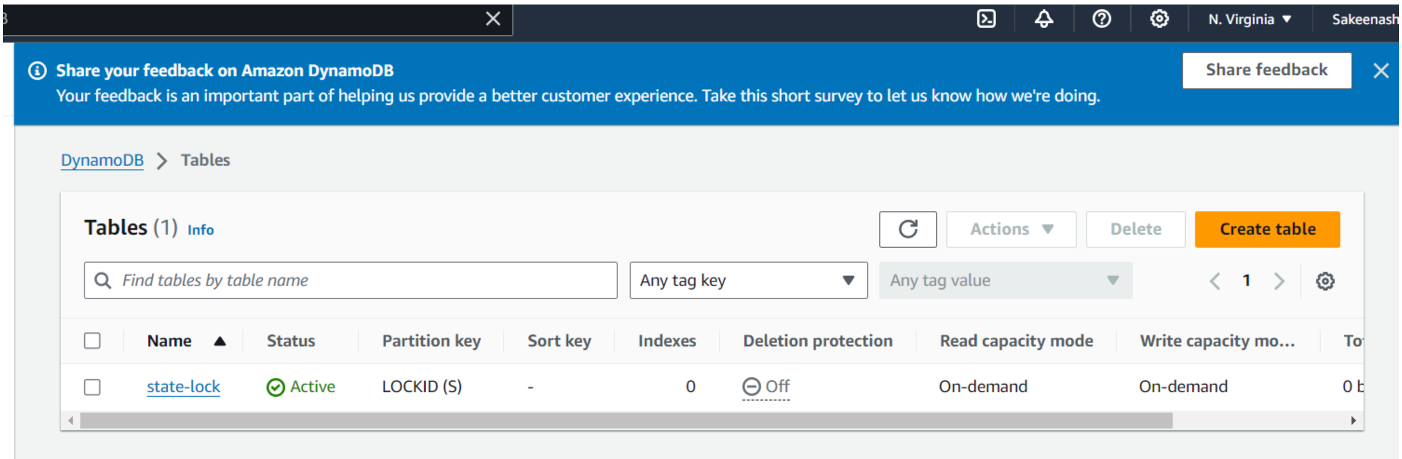
- terraform init (initialize your working directory)
- terraform plan (plan the changes)
- terraform apply (apply the changes)

This configuration will create an S3 bucket with versioning and server-side encryption enabled, as well as a DynamoDB table named state-lock with on-demand billing and a string primary key LOCKID.

```
provider "aws" {
  region = "us-east-1"  # Change to your preferred region
}

# Create an S3 bucket
resource "aws_s3_bucket" "mybucket" {
    bucket = "s3statefile786"  # Change to a unique bucket name

    # Optional: Adding tags to the bucket
    tags = {
        Name        = "s3statefile786"
        Environment = "Dev"
    }
}

# Manage versioning for the S3 bucket
resource "aws_s3_bucket_versioning" "mybucket_versioning" {
    bucket = aws_s3_bucket.mybucket.id

    versioning_configuration {
        status = "Enabled"
    }
}

# Manage server-side encryption for the S3 bucket
resource "aws_s3_bucket_server_side_encryption_configuration" "mybucket_encryption" {
    bucket = aws_s3_bucket.mybucket.id

    rule {
        apply_server_side_encryption_by_default {
            sse_algorithm = "AES256"
        }
    }
}

#create Dynamodb for state-locking

resource "aws_dynamodb_table" "state-lock" {
    name = "state-lock"
    billing_mode = "PAY_PER_REQUEST"
    hash_key = "LOCKID"

    attribute {
        name = "LOCKID"
        type = "S"
    }
}
```

After applying the changes, you can verify whether the s3 bucket and DynamoDB table created in your aws console.





Now create an backend.tf file which will have your bucket details and dynamo dB table.

```
#backend configuration

terraform {
    backend "s3" {
        bucket = "s3statefile786"
        key = "state-lock"
        region = "us-east-1"
        dynamodb_table = "state-lock"
    }
}
```

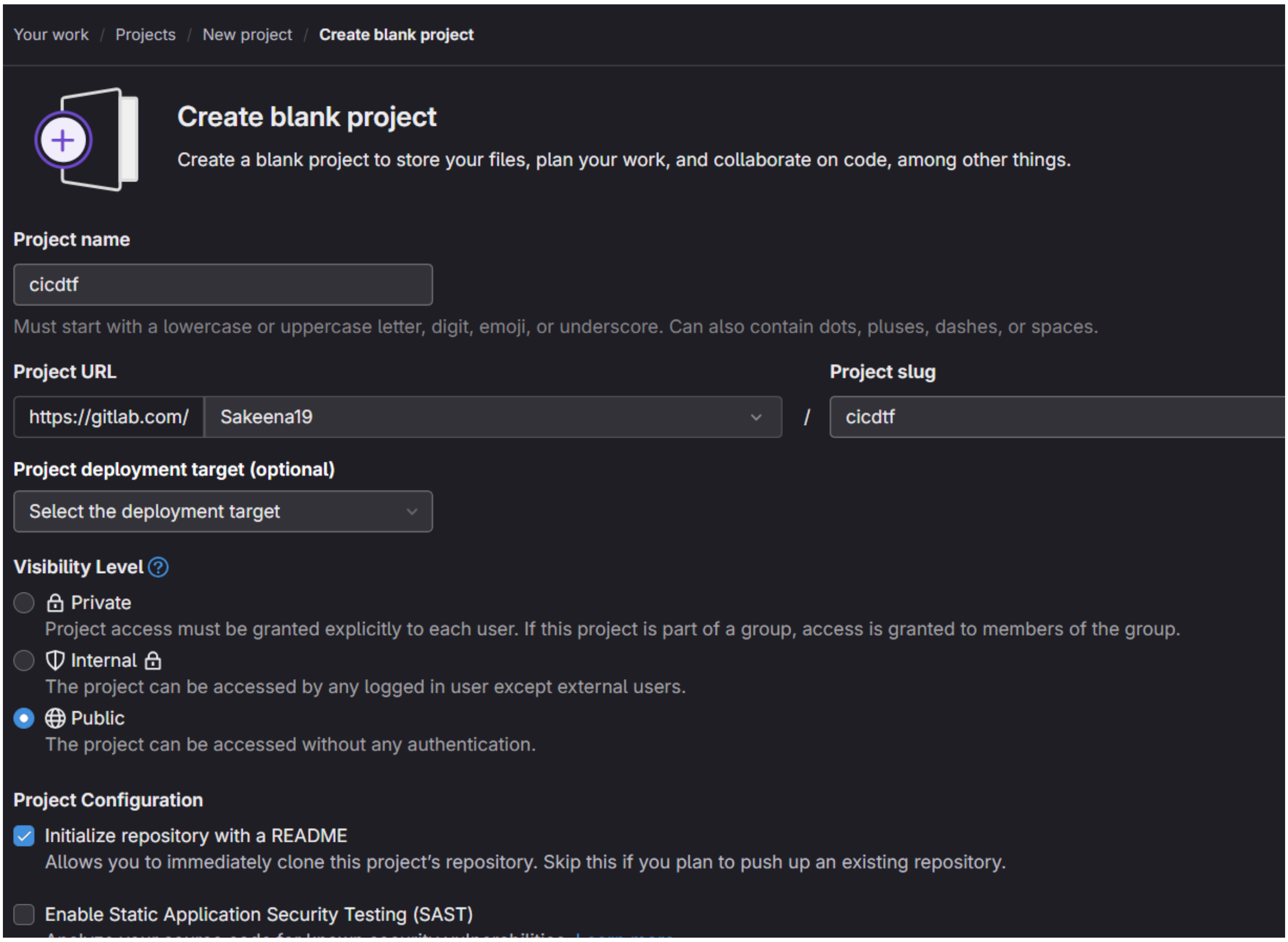
- **backend "s3"**: Specifies that the Terraform state will be stored in an S3 bucket.
- **bucket = "s3statefile786"**: Sets the name of the S3 bucket where the state file will be stored.
- **key = "state-lock"**: Defines the path within the S3 bucket where the state file will be stored. This can be thought of as the "file name" for the state file within the bucket.
- **region = "us-east-1"**: Indicates the AWS region where the S3 bucket is located.
- **dynamodb_table = "state-lock"**: Specifies the DynamoDB table used for state locking to prevent concurrent modifications to the state file. This helps ensure that only one Terraform process can modify the state at a time, preventing conflicts.

Run “terraform init” to initialize the backend.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

ubuntu@ip-172-31-25-53:~/cicdtf$ terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.57.0

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.
ubuntu@ip-172-31-25-53:~/cicdtf$
```

To automate all the above actions, let’s move to part2, i.e., create GitLab repo, push code to repo and setup cicd pipeline.

Go to GitLab and create a new repository:
Click on new project -> create a blank project -> provide project name, visibility, enable readme -> create project.



To push the code, first step is to initialize the repository.

```
ubuntu@ip-172-31-25-53:~/cicdtf$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ubuntu/cicdtf/.git/
ubuntu@ip-172-31-25-53:~/cicdtf$
```

To use only necessary files and ignore other files, create .gitignore file which can be found Here

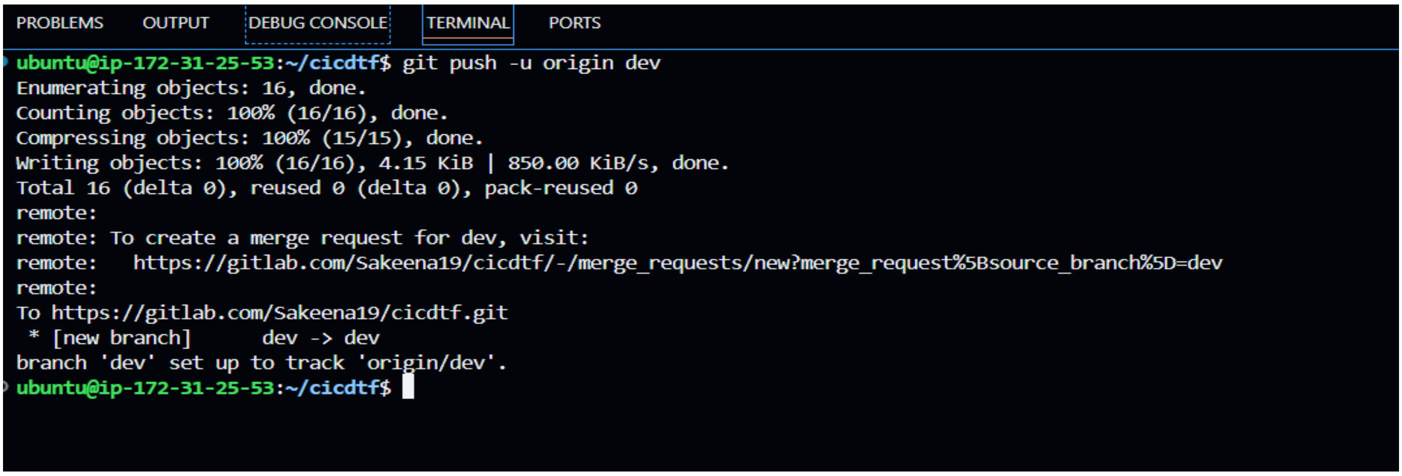
To connect with your GitLab repo, use
git remote add origin https://gitlab.com/Sakeena19/cicdtf.git

```
ubuntu@ip-172-31-25-53:~/cicdtf$ git remote add origin https://gitlab.com/Sakeena19/cicdtf.git
ubuntu@ip-172-31-25-53:~/cicdtf$ git remote -v
origin  https://gitlab.com/Sakeena19/cicdtf.git (fetch)
origin  https://gitlab.com/Sakeena19/cicdtf.git (push)
ubuntu@ip-172-31-25-53:~/cicdtf$
```

The next step is to create a branch called "dev" because we cannot directly push our code to the main branch. This allows us to make changes and test them safely before merging into the main branch which is the best practice.

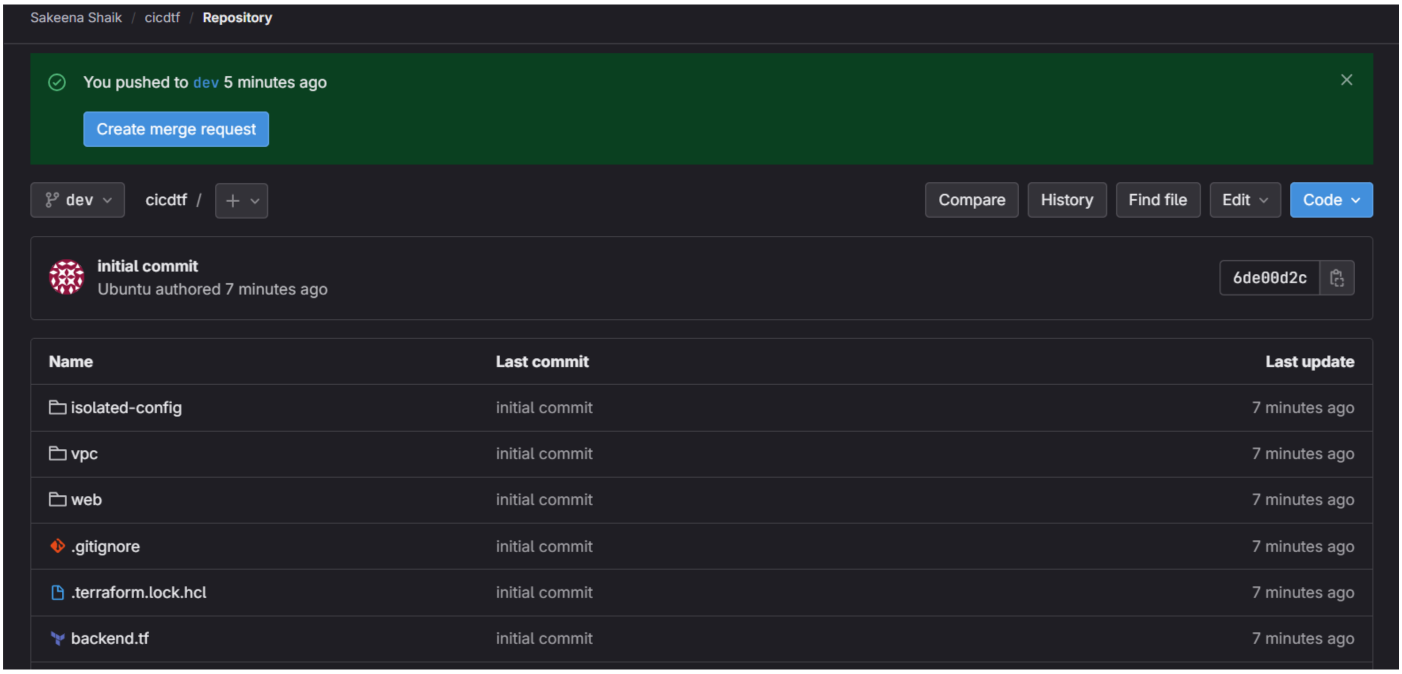
To create a branch, use "git checkout -b dev" which will create a branch and switch at a time.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

ubuntu@ip-172-31-25-53:~/cicdtf$ git checkout -b dev
Switched to a new branch 'dev'
ubuntu@ip-172-31-25-53:~/cicdtf$ git add .
ubuntu@ip-172-31-25-53:~/cicdtf$ git commit -m "initial commit"
[dev (root-commit) 6de00d2] initial commit
 Committer: Ubuntu <ubuntu@ip-172-31-25-53.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 14 files changed, 238 insertions(+)
```

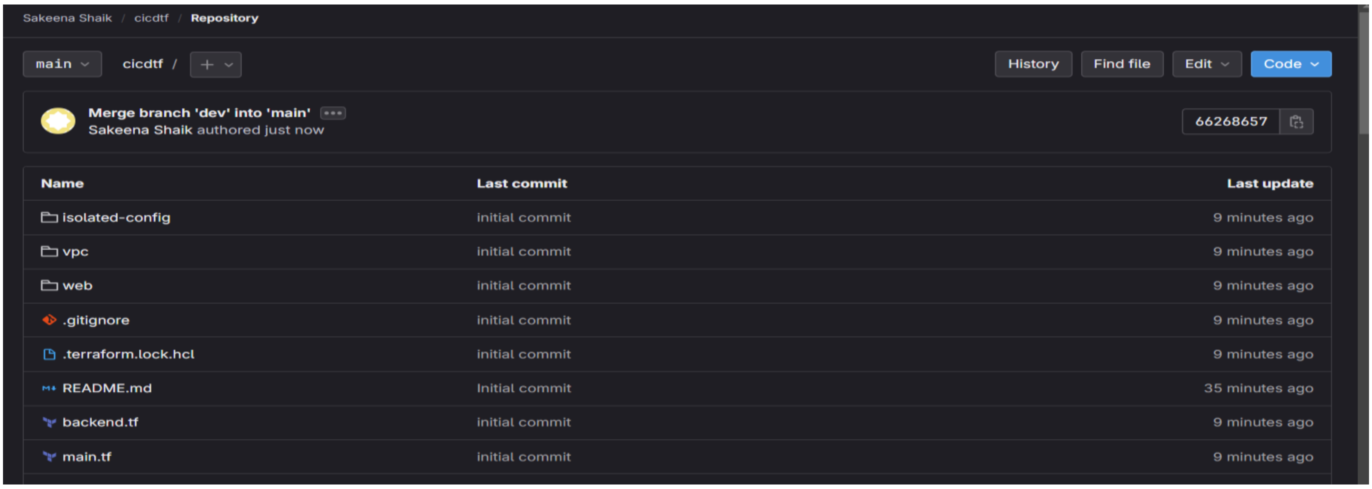


🔷git add . (Adds all changes from current working Dir to staging area)
🔷git commit -m “initial commit” (commits the staged changes to local repo)
🔷git push -u origin dev (pushes code from local to remote repo i.e., GitLab in the branch named dev)

The dev branch should be created in your GitLab repo from which you can create merge request to merge from dev to main.



After merging you can view your code available in main branch.



Now as the code is ready, lets write a CICD pipeline script in Gitlab.

The pipeline configuration file must be named "gitlab-ci.yml" for GitLab to recognize it as the CI/CD configuration file. This naming convention ensures that GitLab understands and processes the configuration defined within.

The main purpose of defining this file is to automate the terraform commands so that whenever a person makes any change in infrastructure the pipeline will trigger automatically.

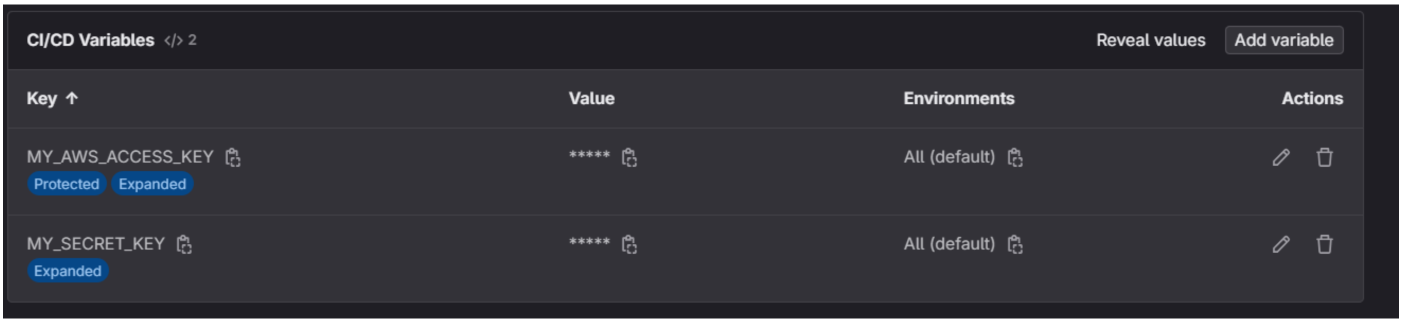
.gitlab-ci.yml 1.94 KiB

Blame Edit Replace Delete

```yaml
image:
  name: registry.gitlab.com/gitlab-org/gitlab-build-images:terraform  # Docker image with Terraform installed
  entrypoint:
    - '/usr/bin/env'
    - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'  # Setting the PATH environment variable

variables:
  TF_VAR_gitlab_token: ${GITLAB_ACCESS_TOKEN}  # Setting Terraform variable for GitLab token
  AWS_ACCESS_KEY_ID: ${MY_AWS_KEY}  # Setting AWS access key ID
  AWS_SECRET_ACCESS_KEY: ${MY_AWS_ACCESS_KEY}  # Setting AWS secret access key
  AWS_DEFAULT_REGION: "us-east-1"  # Setting AWS region

cache:
  paths:
    - .terraform  # Caching Terraform plugins and modules

before_script:
  - terraform --version  # Output Terraform version for visibility
  - terraform init -backend-config="tfstate.config"  -migrate-state # Initialize Terraform with backend configuration

stages:
  - validate  # Stage to validate Terraform configurations
  - plan      # Stage to generate Terraform execution plan
  - apply     # Stage to apply Terraform changes
  - destroy   # Stage to destroy Terraform resources

validate:
  stage: validate
  script:
    - terraform validate  # Validate Terraform configurations

plan:
  stage: plan
  script:
    - terraform plan -out="planfile"  # Generate Terraform execution plan and save to planfile
  dependencies:
    - validate  # Dependency on 'validate' stage to run before 'plan'
  artifacts:
    paths:
      - planfile  # Publish 'planfile' as an artifact for later stages

apply:
  stage: apply
  script:
    - terraform apply -input=false "planfile"  # Apply Terraform changes using planfile
  dependencies:
    - plan  # Dependency on 'plan' stage to run before 'apply'
  when: manual  # Manually trigger the 'apply' stage to execute

destroy:
  stage: destroy
  script:
    - terraform destroy --auto-approve  # Destroy Terraform resources automatically
  when: manual  # Manually trigger the 'destroy' stage to execute
```

As it’s not a best practise to hardcode the aws secret and access key in code, variables can be created for storing access keys and secret access keys in your GitLab repository.

Navigate to your project repo -> settings -> CICD ->Variables -> Add variables -> Add variables for your access key and secret access key.



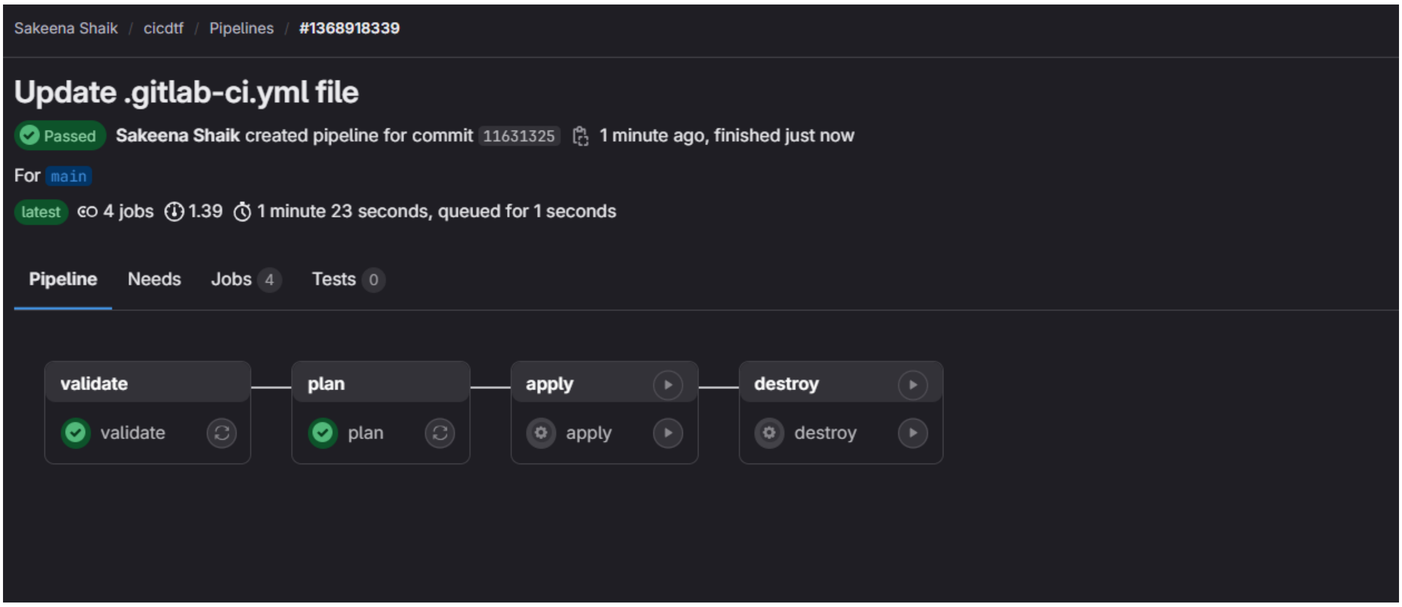
Once the above changes done, the pipeline will start triggering automatically executing all the steps we scripted in .gitlab-ci.yml file.

cicd pipeline script explanation (.gitlab-ci.yml file):

This GitLab CI/CD pipeline script is designed to automate the deployment and management of infrastructure using Terraform. The script uses a Docker image that has Terraform installed and sets environment variables for AWS credentials and a GitLab token. It also caches Terraform plugins and modules to improve efficiency. The pipeline is divided into four stages: validate, plan, apply, and destroy.

In the validate stage, the script checks if the Terraform configuration files are correct. The plan stage then generates a Terraform execution plan and saves it as an artifact called planfile. The apply stage uses this plan to create or update the infrastructure, but this stage must be triggered manually to execute. Similarly, the destroy stage, which is also manually triggered, destroys the Terraform-managed resources automatically.

Before running these stages, the script outputs the Terraform version and initializes Terraform with a backend configuration specified in a tfstate.config file. By organizing the pipeline in this way, the script ensures that infrastructure changes are validated, planned, and applied in a controlled and orderly manner, with the option to manually control the application and destruction of infrastructure changes.



Whenever the pipeline executes, the validate and plan stages run automatically, while the apply and destroy stages require manual execution, as defined in the script. This approach follows industry best practices, allowing verification of changes and manual approval before they are applied or destroyed.

1) Logs from validate stage:

Sakeena Shaik / cicdtf / Jobs / #7313817352

validate

Passed Started 13 minutes ago by **Sakeena Shaik**

Search job log

```
Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
  on green-2.saas-linux-small-amd64.runners-manager.gitlab.com/default ns46NMmJ, system ID: s_85d7af184313
Preparing the "docker+machine" executor                                                     00:06
Using Docker executor with image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
Authenticating with credentials from job payload (GitLab Registry)
Pulling docker image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/gitlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b04226483073330d6563a3e101052ba0f8f1f1f601d476611d7df1b ...
Preparing environment                                                                       00:01
Running on runner-ns46nmmj-project-59788171-concurrent-0 via runner-ns46nmmj-s-l-s-amd64-1720665055-7dc177a9...
Getting source from Git repository                                                          00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/Sakeena19/cicdtf/.git/
Created fresh repository.
Checking out 11631325 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Restoring cache                                                                             00:09
Checking cache for default-protected...
Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Successfully extracted cache
Executing "step_script" stage of the job script                                             00:08
Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/gitlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b04226483073330d6563a3e101052ba0f8f1f1f601d476611d7df1b ...
```

Search job log

```
$ terraform --version
Terraform v1.4.2
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.57.0
Your version of Terraform is out of date! The latest version
is 1.9.2. You can update by downloading from https://www.terraform.io/downloads.html
$ terraform init -backend-config="tfstate.config"  -migrate-state
Initializing the backend...
Backend configuration changed!
Terraform has detected that the configuration specified for the backend
has changed. Terraform will now check for existing state in the backends.
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.57.0
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.
$ terraform validate
Success! The configuration is valid.
Saving cache for successful job                                                             00:11
Creating cache default-protected...
.terraform: found 12 matching artifact files and directories
Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Created cache
Cleaning up project directory and file based variables                                      00:01
Job succeeded
```

2) Logs from plan stage:

Sakeena Shaik / cicdtf / Jobs / **#7313817355**

plan

Passed Started 15 minutes ago by **Sakeena Shaik**

Search job log

```
Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
  on green-1.saas-linux-small-amd64.runners-manager.gitlab.com/default JLgUopmM, system ID: s_deaa2ca09de7
Preparing the "docker+machine" executor                                   00:06
Using Docker executor with image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
Authenticating with credentials from job payload (GitLab Registry)
Pulling docker image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/gitlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d6563a3e101052ba0f8f1f1f601d476611d7df1b ...
Preparing environment                                                     00:01
Running on runner-jlguopmm-project-59788171-concurrent-0 via runner-jlguopmm-s-l-s-amd64-1720665060-076fa23f...
Getting source from Git repository                                        00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/Sakeena19/cicdtf/.git/
Created fresh repository.
Checking out 11631325 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Restoring cache                                                           00:11
Checking cache for default-protected...
Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Successfully extracted cache
Executing "step_script" stage of the job script                           00:10
Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/gitlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d6563a3e101052ba0f8f1f1f601d476611d7df1b ...
$ terraform --version
Terraform v1.4.2
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.57.0
Your version of Terraform is out of date! The latest version
is 1.9.2. You can update by downloading from https://www.terraform.io/downloads.html
$ terraform init -backend-config="tfstate.config"  -migrate-state
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.57.0
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.
$ terraform plan -out="planfile"
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:
  # module.ec2.aws_instance.server will be created
  + resource "aws_instance" "server" {
      + ami                                  = "ami-04a81a99f5ec58529"
      + 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)
```

```
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "myvpc"
        }
      + tags_all                             = {
          + "Name" = "myvpc"
        }
    }
Plan: 4 to add, 0 to change, 0 to destroy.

Saved the plan to: planfile
To perform exactly these actions, run the following command to apply:
    terraform apply "planfile"
Saving cache for successful job                                                    00:11
Creating cache default-protected...
.terraform: found 12 matching artifact files and directories
Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Created cache
Uploading artifacts for successful job                                             00:02
Uploading artifacts...
planfile: found 1 matching artifact files and directories
WARNING: Upload request redirected                    location=https://gitlab.com/api/v4/jobs/7313817355/artifacts?artifact_forma
t=zip&artifact_type=archive new-url=https://gitlab.com
WARNING: Retrying...                                  context=artifacts-uploader error=request redirected
Uploading artifacts as "archive" to coordinator... 201 Created  id=7313817355 responseStatus=201 Created token=glcbt-66
Cleaning up project directory and file based variables                             00:01
Job succeeded
```

3) Logs from apply:

Sakeena Shaik / cicdtf / Jobs / #7313817356

apply

Passed Started just now by Sakeena Shaik

Search job log

```
Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
  on green-2.saas-linux-small-amd64.runners-manager.gitlab.com/default ns46NMmJ, system ID: s_85d7af184313
Preparing the "docker+machine" executor                                            00:05
Using Docker executor with image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
Authenticating with credentials from job payload (GitLab Registry)
Pulling docker image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/g
itlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d656
3a3e101052ba0f8f1f1f601d476611d7df1b ...
Preparing environment                                                              00:02
Running on runner-ns46nmmj-project-59788171-concurrent-0 via runner-ns46nmmj-s-l-s-amd64-1720666239-82aa6311...
Getting source from Git repository                                                 00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/Sakeena19/cicdtf/.git/
Created fresh repository.
Checking out 11631325 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Restoring cache                                                                    00:09
Checking cache for default-protected...
Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Successfully extracted cache
Downloading artifacts                                                              00:01
Downloading artifacts for plan (7313817355)...
Downloading artifacts from coordinator... ok        host=storage.googleapis.com id=7313817355 responseStatus=200 OK token=glcbt
```

```
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.57.0
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.
$ terraform apply -input=false "planfile"
module.vpc.aws_vpc.myvpc: Creating...
module.vpc.aws_vpc.myvpc: Still creating... [10s elapsed]
module.vpc.aws_vpc.myvpc: Creation complete after 12s [id=vpc-04505ae4c1fd43057]
module.vpc.aws_subnet.pb_sn: Creating...
module.vpc.aws_security_group.sg: Creating...
module.vpc.aws_security_group.sg: Creation complete after 2s [id=sg-0d7b8162c87df1b3e]
module.vpc.aws_subnet.pb_sn: Still creating... [10s elapsed]
module.vpc.aws_subnet.pb_sn: Creation complete after 11s [id=subnet-07c1e73bd369841d2]
module.ec2.aws_instance.server: Creating...
module.ec2.aws_instance.server: Still creating... [10s elapsed]
module.ec2.aws_instance.server: Still creating... [20s elapsed]
module.ec2.aws_instance.server: Still creating... [30s elapsed]
module.ec2.aws_instance.server: Creation complete after 32s [id=i-023a7418b3239ade7]
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
Saving cache for successful job                                              00:11
Creating cache default-protected...
.terraform: found 12 matching artifact files and directories
Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Created cache
Cleaning up project directory and file based variables                       00:01
Job succeeded
```

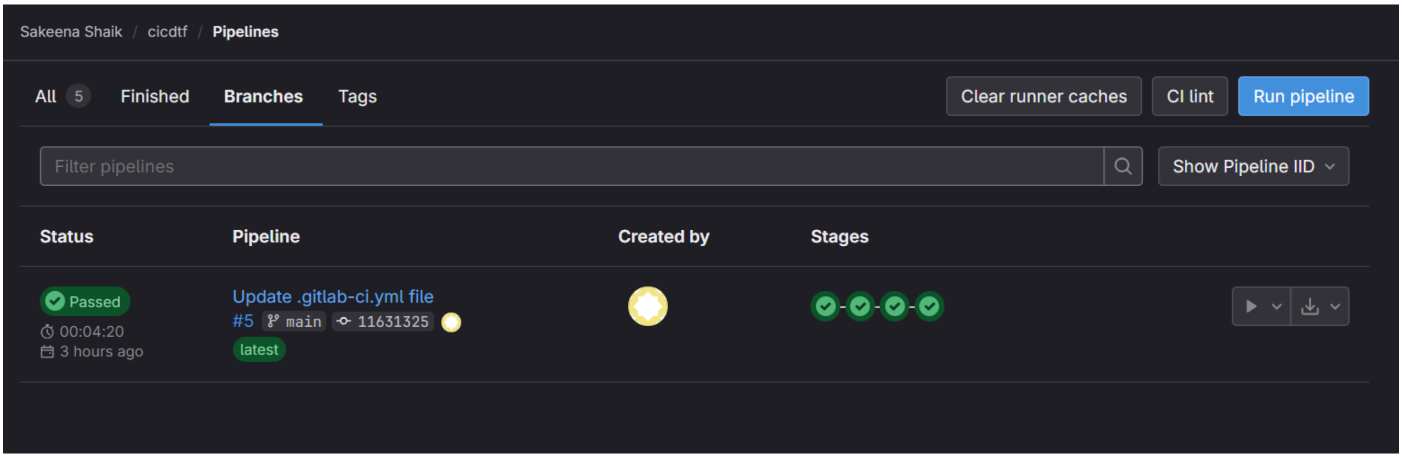
4) Logs from destroy:

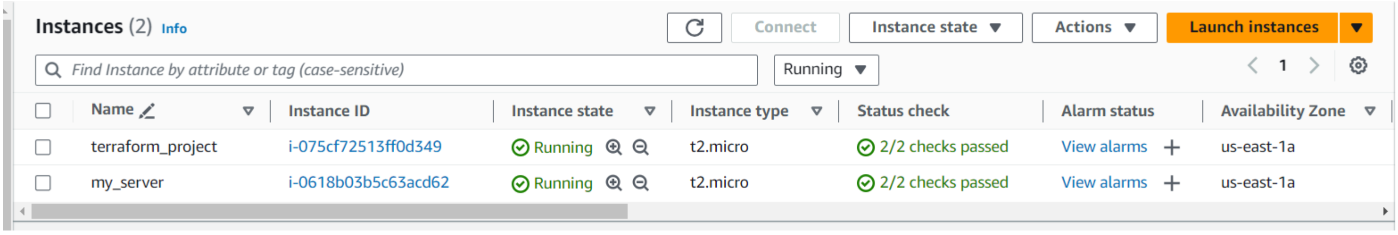
Sakeena Shaik / cicdtf / Jobs / #7313817357

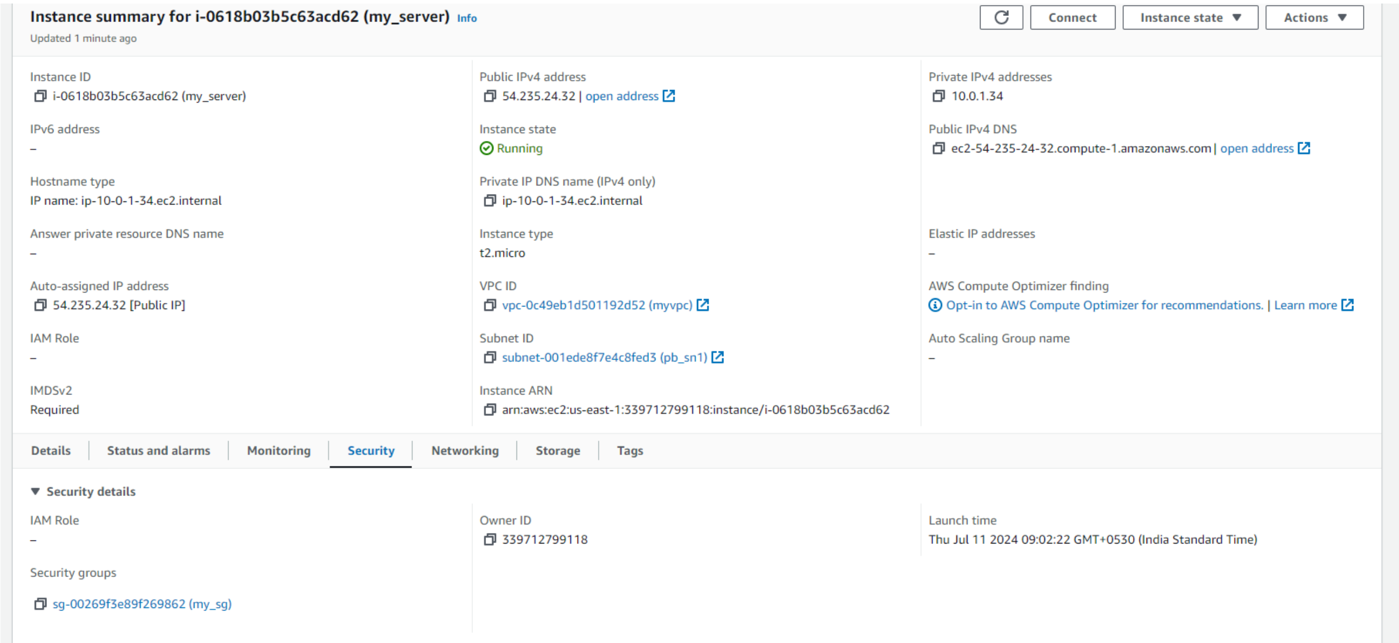
Search job log

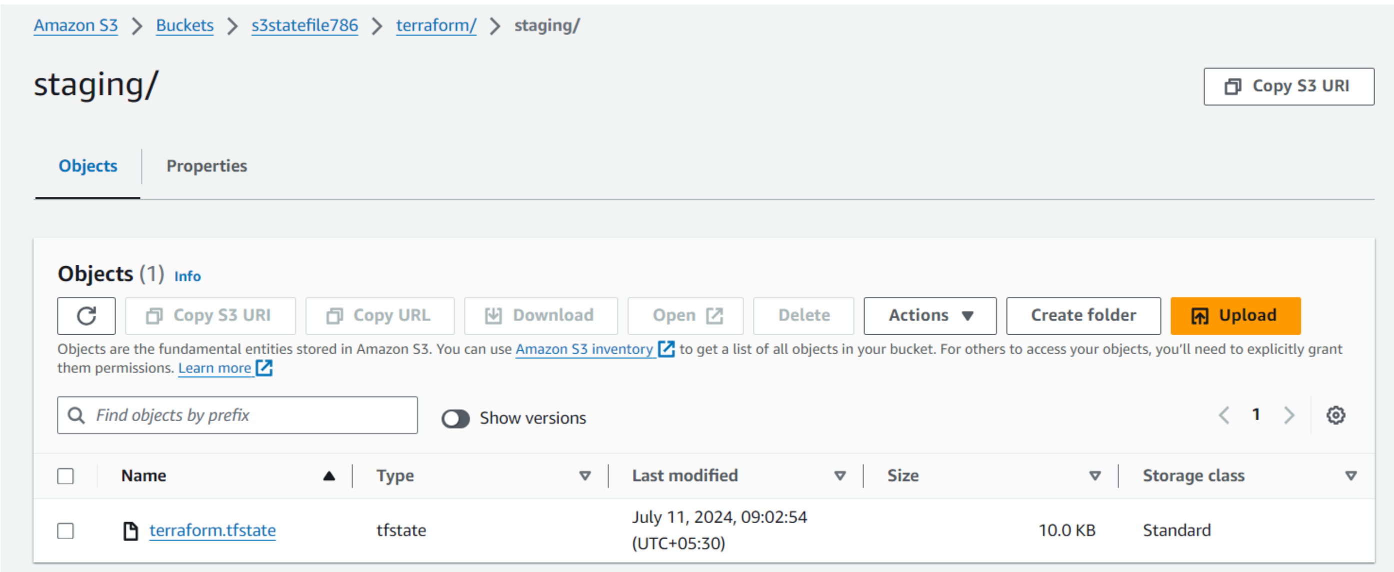
```
        - tags                          = {
            - "Name" = "myvpc"
          } -> null
        - tags_all                      = {
            - "Name" = "myvpc"
          } -> null
      }
Plan: 0 to add, 0 to change, 4 to destroy.
module.ec2.aws_instance.server: Destroying... [id=i-023a7418b3239ade7]
module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 10s elapsed]
module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 20s elapsed]
module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 30s elapsed]
module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 40s elapsed]
module.ec2.aws_instance.server: Destruction complete after 41s
module.vpc.aws_subnet.pb_sn: Destroying... [id=subnet-07c1e73bd369841d2]
module.vpc.aws_security_group.sg: Destroying... [id=sg-0d7b8162c87df1b3e]
module.vpc.aws_subnet.pb_sn: Destruction complete after 0s
module.vpc.aws_security_group.sg: Destruction complete after 0s
module.vpc.aws_vpc.myvpc: Destroying... [id=vpc-04505ae4c1fd43057]
module.vpc.aws_vpc.myvpc: Destruction complete after 1s
Destroy complete! Resources: 4 destroyed.
Saving cache for successful job                                              00:11
Creating cache default-protected...
.terraform: found 12 matching artifact files and directories
Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
Created cache
Cleaning up project directory and file based variables                       00:01
Job succeeded
```

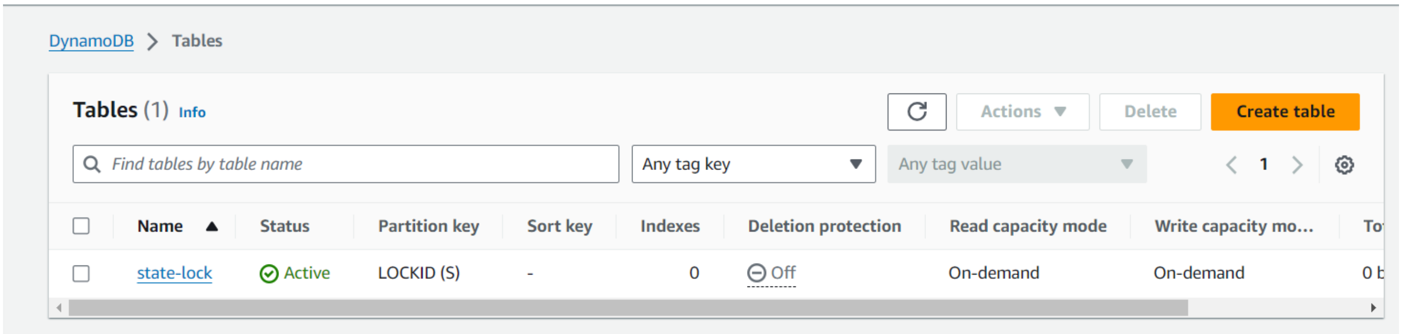
All 4 states have been executed.


Sakeena Shaik / cicdtf / Pipelines
All 5
Finished
Branches
Tags
Clear runner caches
CI lint
Run pipeline
Filter pipelines
Show Pipeline IID
Status
Pipeline
Created by
Stages
Passed
00:04:20
3 hours ago
Update .gitlab-ci.yml file
#5 main 11631325
latest


Instances (2) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
Running
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
terraform_project
i-075cf72513ff0d349
Running
t2.micro
2/2 checks passed
View alarms
us-east-1a
my_server
i-0618b03b5c63acd62
Running
t2.micro
2/2 checks passed
View alarms
us-east-1a


Instance summary for i-0618b03b5c63acd62 (my_server) Info
Updated 1 minute ago
Connect
Instance state
Actions
Instance ID
i-0618b03b5c63acd62 (my_server)
Public IPv4 address
54.235.24.32 | open address
Private IPv4 addresses
10.0.1.34
IPv6 address
–
Instance state
Running
Public IPv4 DNS
ec2-54-235-24-32.compute-1.amazonaws.com | open address
Hostname type
IP name: ip-10-0-1-34.ec2.internal
Private IP DNS name (IPv4 only)
ip-10-0-1-34.ec2.internal
Answer private resource DNS name
–
Instance type
t2.micro
Elastic IP addresses
–
Auto-assigned IP address
54.235.24.32 [Public IP]
VPC ID
vpc-0c49eb1d501192d52 (myvpc)
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role
–
Subnet ID
subnet-001ede8f7e4c8fed3 (pb_sn1)
Auto Scaling Group name
–
IMDSv2
Required
Instance ARN
arn:aws:ec2:us-east-1:339712799118:instance/i-0618b03b5c63acd62
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Security details
IAM Role
–
Owner ID
339712799118
Launch time
Thu Jul 11 2024 09:02:22 GMT+0530 (India Standard Time)
Security groups
sg-00269f3e89f269862 (my_sg)


Amazon S3 > Buckets > s3statefile786 > terraform/ > staging/
staging/
Copy S3 URI
Objects
Properties
Objects (1) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Show versions
1
Name
Type
Last modified
Size
Storage class
terraform.tfstate
tfstate
July 11, 2024, 09:02:54 (UTC+05:30)
10.0 KB
Standard



Verify full logs in below text file:

Logs.txt

The pipeline performs the following steps:

- Initializes Terraform with the specified backend configuration.
- Applies the Terraform plan to create infrastructure resources (VPC, Subnet, Security Group, and EC2 instance).
- Saves .terraform directory to cache for future use.
- Cleans up the environment after the job is completed.