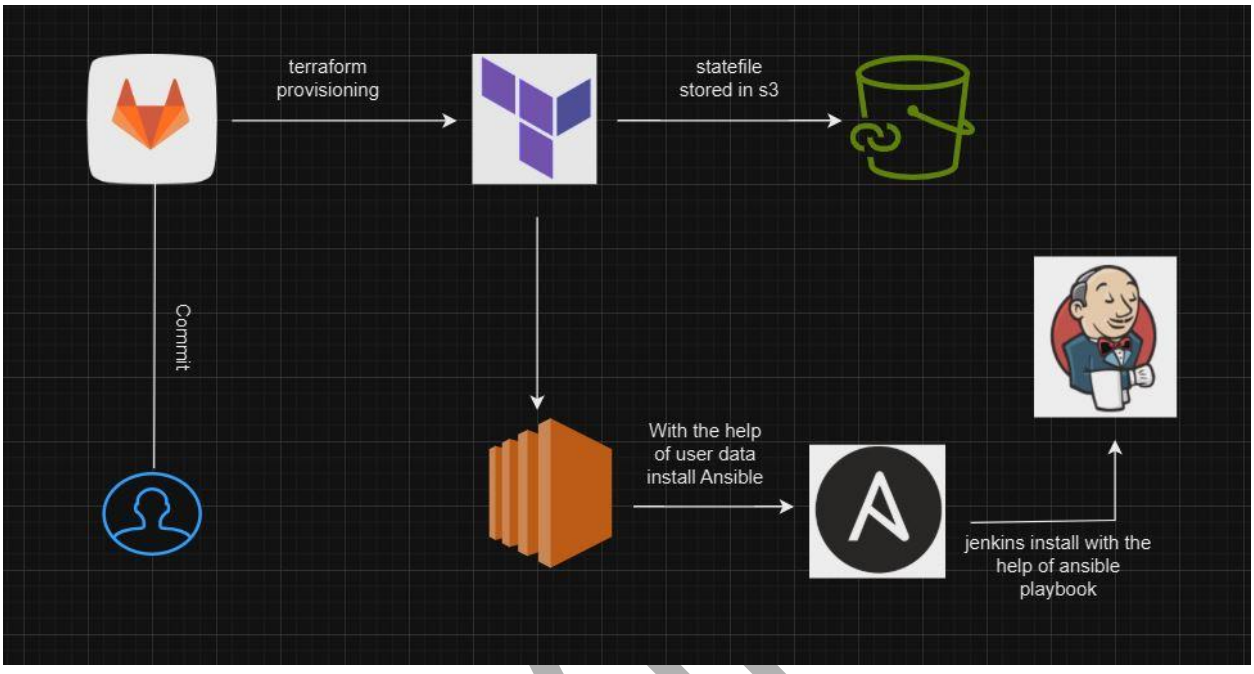
Managing Infrastructure with GitLab CI/CD for Terraform: Plan, Validate,Apply, Destroy



Prerequisites -

- Step1: create an IAM user
- Step2: Create S3 Bucket
- Step3: Create Gitlab Account
- Step4: Variables setup in Gitlab (Secrets)
- Step5: Terraform Files
- Step6: GitLab CI/CD configuration
- Step7: .gitlab-ci.yml
- Step8: Destroy

1. **GitLab:** GitLab is a platform where developers store and collaborate on their code, acting as a virtual workspace for coding projects.
2. **CI/CD:** CI means "Continuous Integration," and CD means "Continuous Deployment" or "Continuous Delivery."

- **Continuous Integration (CI):** Every time a developer makes changes to the code, CI tools automatically check if those changes work well with the existing code, acting as a "test run" to catch any mistakes early on.
- **Continuous Deployment/Delivery (CD):** Once the code is tested and ready, CD tools automatically release it to production, functioning like an automatic delivery system for software.

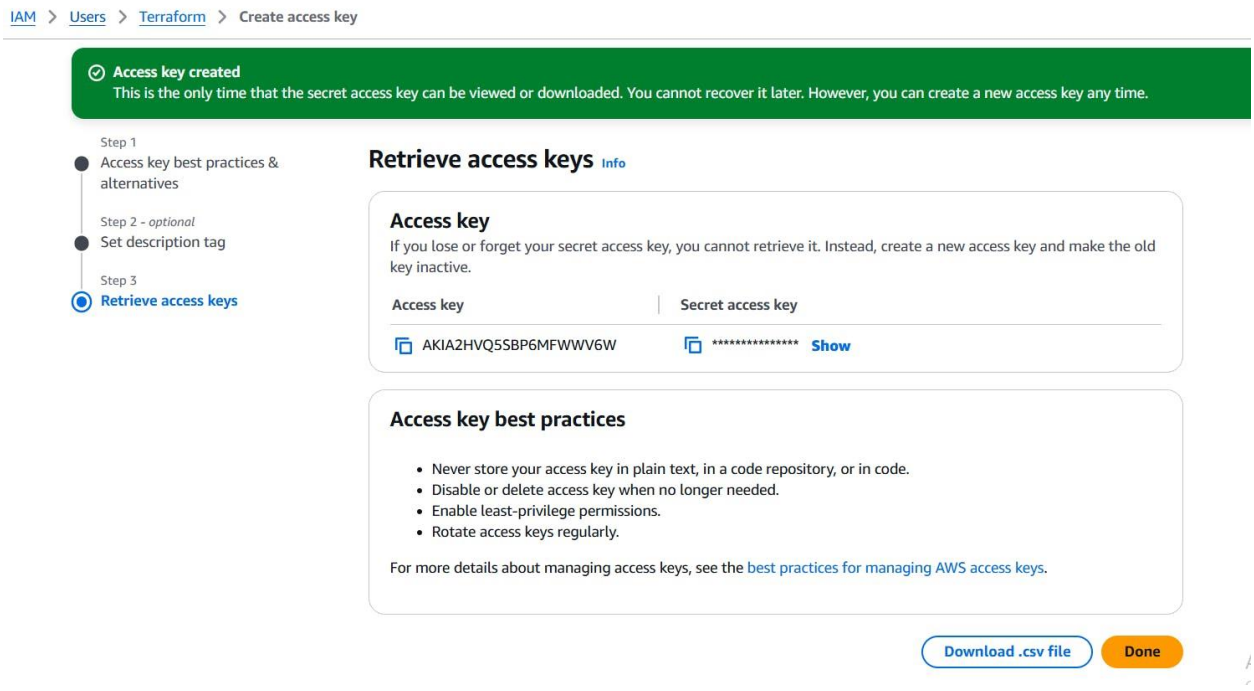
In Simple Words: GitLab CI/CD is a system that helps developers automatically check their code for errors and, if everything is fine, automatically release their software without manual effort. It's like having robots that test and deliver your code, saving time and reducing mistakes.

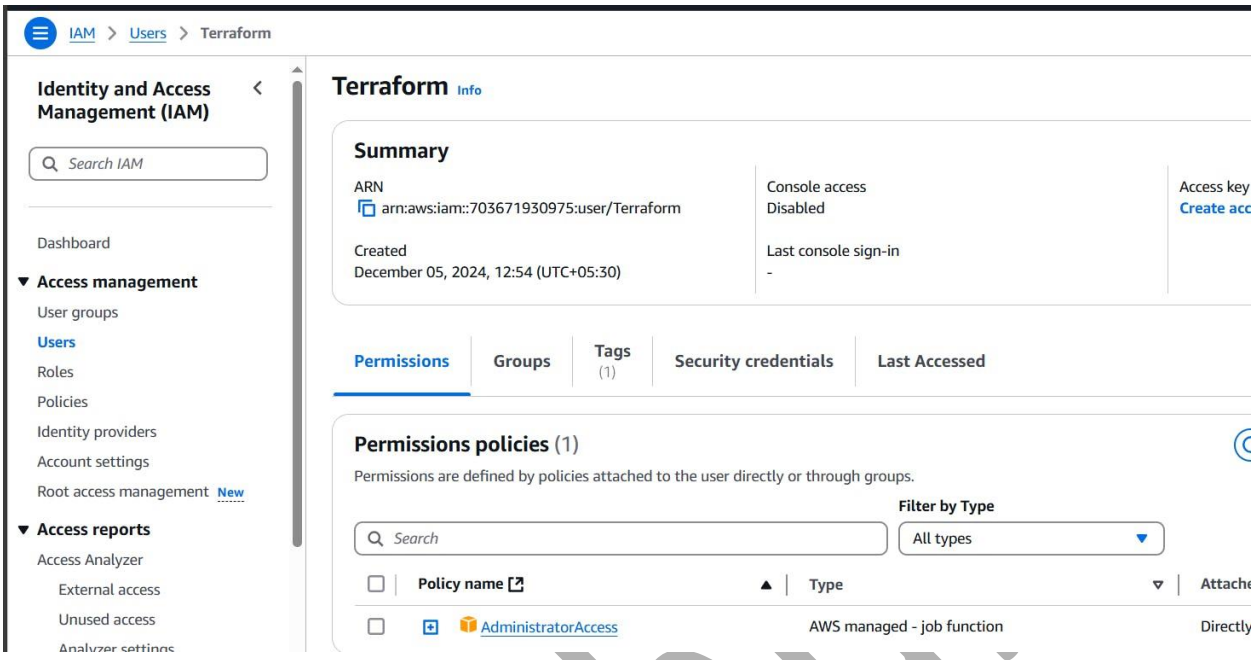
1.Create an IAM user

Navigate to the **AWS console**

Search for IAM → User →1. give a name 2. Click "Attach policies directly"3. Click this checkbox with Administrator access 4. create a user.

Click "Security credentials" →Click "Create access key" →Click this radio button with the CLI → Agree to terms →next → create access key →Download .csv file.





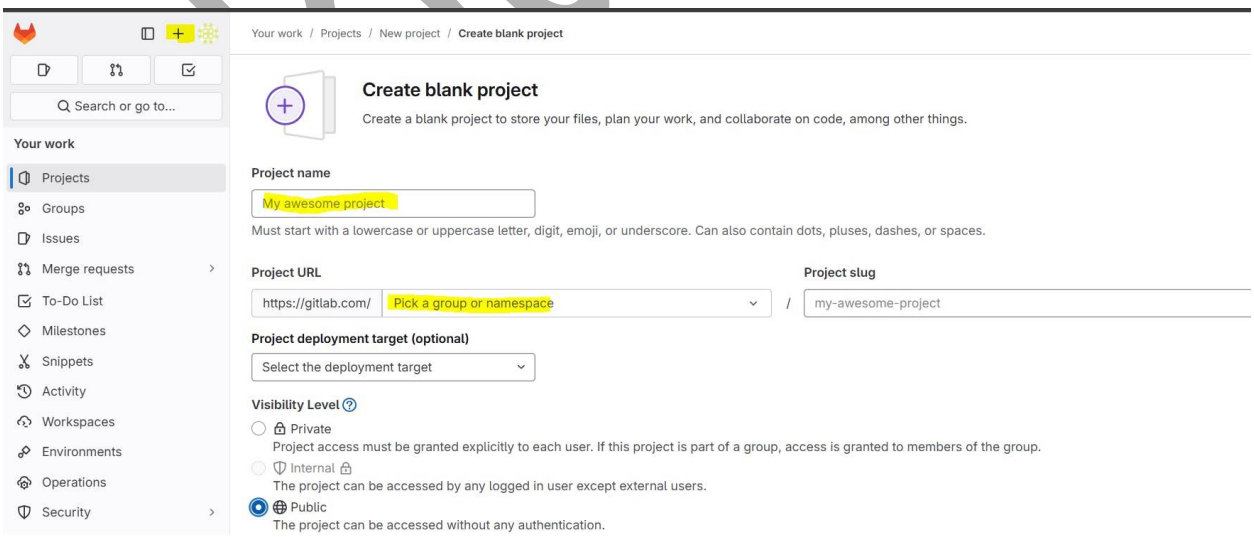
2.Create S3 Bucket

Navigate to AWS Console and search for s3 and create a s3 Bucket.

3.Create Gitlab Account

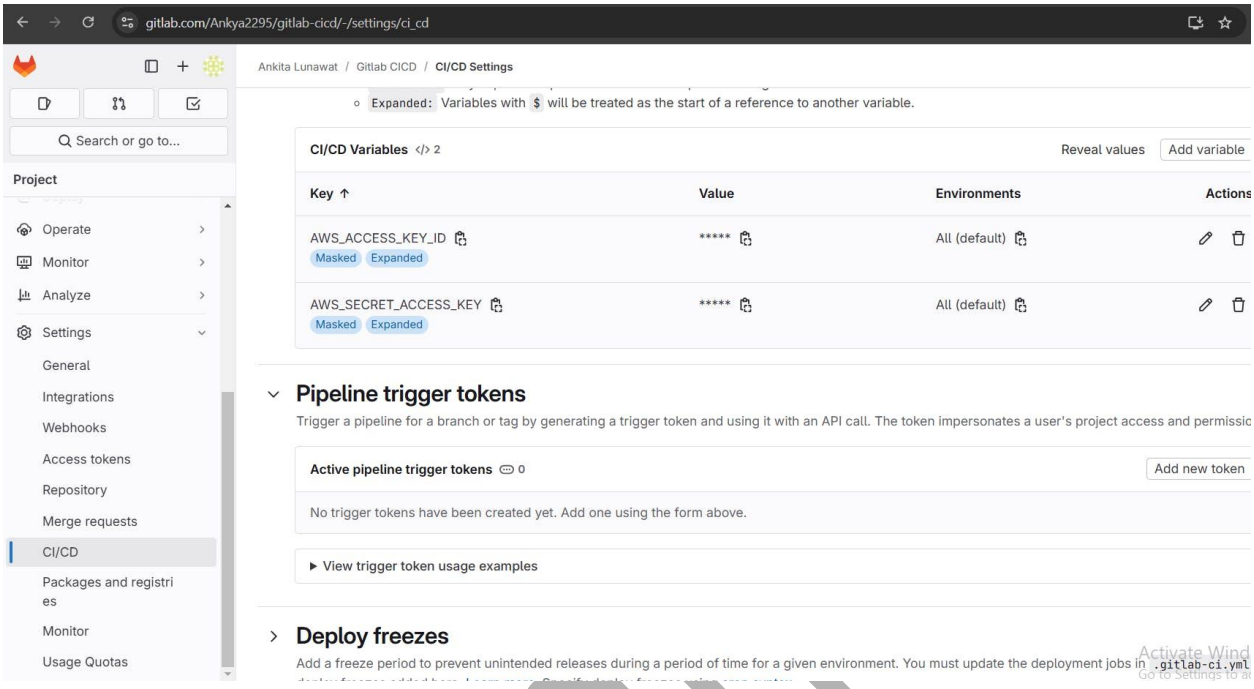
Go to gitlab.com

Lets create New project/Repository



4.Variables setup in Gitlab (Secrets)

Inside your repository → Click on Settings → ci/cd →Click on Expand at variables →Click on Add variable like below added.



5.Terraform Files

Create a blank repository in Gitlab and add these files.

[main.tf]

ami-053b12d3152c0cc71 - AMI ID of instance and Key name we have to add in this file.

```
resource "aws_security_group" "Jenkins-sg" {
  name        = "Jenkins-Security Group"
  description = "Open 22,443,80,8080"
 # Define a single ingress rule to allow traffic on all specified ports
  ingress = [
    for port in [22, 80, 443, 8080] : {
      description      = "TLS from VPC"
      from_port        = port
      to_port          = port
      protocol         = "tcp"
      cidr_blocks      = ["0.0.0.0/0"]
```

```
    ipv6_cidr_blocks = []
    prefix_list_ids  = []
    security_groups  = []
    self             = false
  }
 ]
 egress {
  from_port   = 0
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
 }
 tags = {
  Name = "Jenkins-sg"
 }
}
resource "aws_instance" "web" {
 ami                    = "ami-053b12d3152c0cc71"  #change Ami if you different region
 instance_type          = "t2.medium"
 key_name               = "a"   #change key name
 vpc_security_group_ids = [aws_security_group.Jenkins-sg.id]
 user_data              = templatefile("./install_jenkins.sh", {})
 tags = {
  Name = "Jenkins-sonar"
 }
 root_block_device {
  volume_size = 8
 }
}
```

****

provider.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}
# Configure the AWS Provider
provider "aws" {
  region = "ap-south-1"     #change to desired region.
}
```

****

install_jenkins.sh

```
#!/bin/bash
exec > >(tee -i /var/log/user-data.log)
exec 2>&1
sudo apt update -y
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible -y
sudo apt install git -y
mkdir Ansible && cd Ansible
pwd
git clone https://github.com/Ankita2295/Terraform-GitlabCICD.git
cd ANSIBLE
ansible-playbook -i localhost Jenkins-playbook.yml
```

****

backend.tf

s3 bucket name we have to mention in below code.

```
terraform {
  backend "s3" {
    bucket = "<s3-bucket>" # Replace with your actual S3 bucket name
    key    = "Gitlab/terraform.tfstate"
    region = "ap-south-1"
  }
}
```

****

6.GitLab CI/CD configuration

****

```
stages:
  - validate
  - plan
  - apply
  - destroy
```

****

stages:: This section defines the stages in the CI/CD pipeline. In your configuration, you have four stages: validate, plan, apply, and destroy.

****

```
image:
  name: hashicorp/terraform:light
  entrypoint:
    - '/usr/bin/env'
    - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
```

****

image: specifies the Docker image to use for the GitLab Runner. In this case, you're using the "hashicorp/terraform:light" image for running Terraform commands. The entrypoint lines set the environment to include commonly used paths.

****

before_script:

- export AWS_ACCESS_KEY=${AWS_ACCESS_KEY_ID}
- export AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
- rm -rf .terraform
- terraform --version
- terraform init

---

before_script: This section defines commands to run before each job in the pipeline.

- The first two lines export the AWS access key and secret access key as environment variables, which are used for AWS authentication in your Terraform configuration.
- rm -rf .terraform: This command removes any existing Terraform configuration and state files to ensure a clean environment. * terraform --version: This command displays the Terraform version for debugging and version confirmation. * terraform init: This command initializes Terraform in the working directory, setting up the environment for Terraform operations.

****

validate:

stage: validate

script:

- terraform validate

****

validate: defines a job named "validate" in the "validate" stage, which checks the Terraform configuration for errors.

- script: specifies the commands to run as part of this job, which in this case is terraform validate, used to check the syntax and structure of your Terraform files.

****

plan:

stage: plan

```
  script:
    - terraform plan -out=tfplan
  artifacts:
    paths:
      - tfplan
```

****

plan:: This job, in the "plan" stage, creates a Terraform plan by running terraform plan -out=tfplan, and it saves the plan as an artifact named tfplan.

- script:: Runs terraform plan -out=tfplan, which generates a plan and saves it as "tfplan" in the working directory. * artifacts:: Specifies the artifacts (output files) of this job, indicating that the "tfplan" file should be preserved as an artifact.

****

```
apply:
  stage: apply
  script:
    - terraform apply -auto-approve tfplan
  dependencies:
    - plan
```

****

apply:: This job, in the "apply" stage, applies the Terraform plan generated in the previous stage.

- script:: Runs terraform apply -auto-approve tfplan, which applies the changes specified in the "tfplan" file. * dependencies:: Specifies that this job depends on the successful completion of the "plan" job.

****

```
destroy:
  stage: destroy
  script:
    - terraform init
    - terraform destroy -auto-approve
  when: manual
```

dependencies:

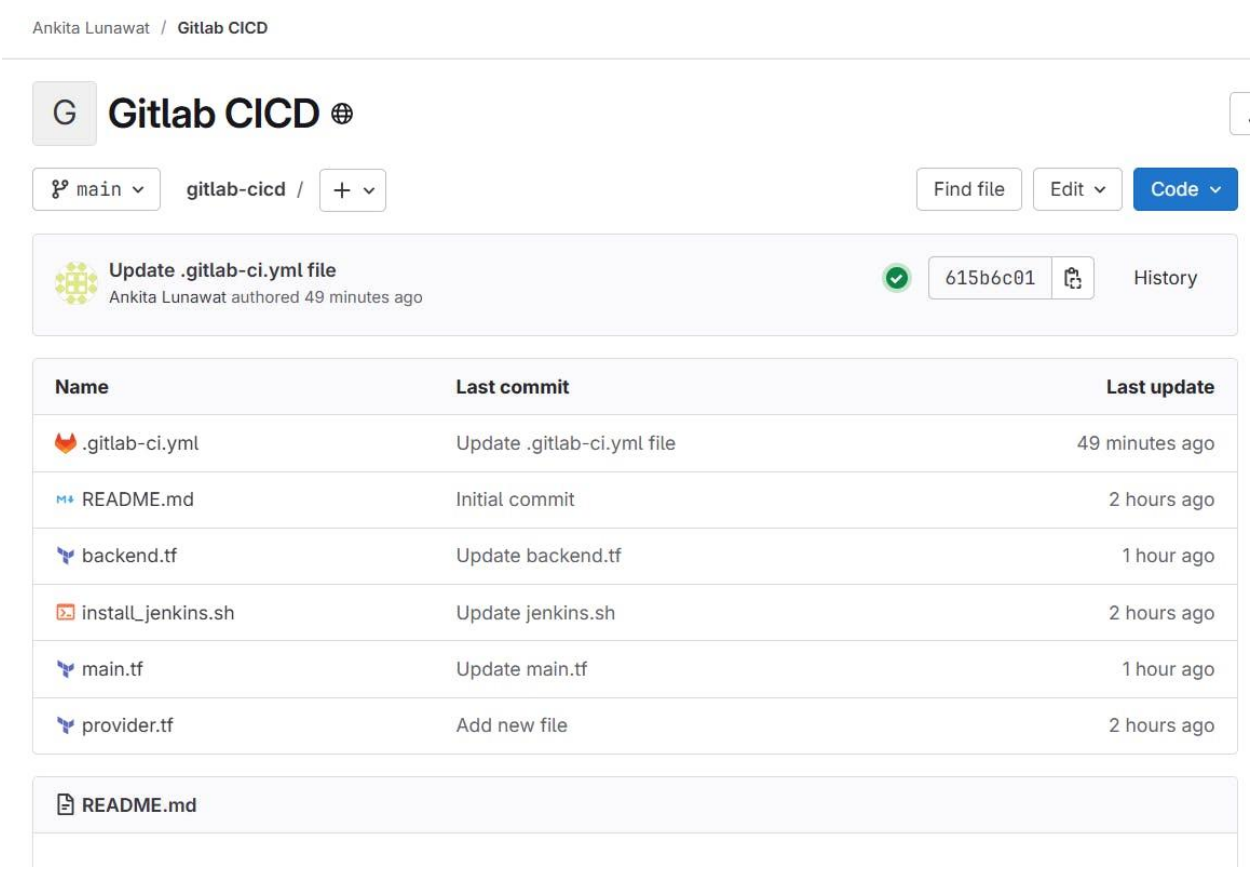
- apply

****

destroy: This job, in the "destroy" stage, is meant for removing the resources managed by Terraform.

- script:: Runs terraform init to set up the Terraform environment and then executes terraform destroy -auto-approve to remove the resources, with the -auto-approve flag allowing for non-interactive execution. * when: manual: Indicates that this job must be manually triggered by a user. * dependencies:: Ensures this job relies on the successful completion of the "apply" job, meaning resources can only be destroyed if they have been applied by a previous "apply" job.

7. .gitlab-ci.yml



Full Gitlab CI/CD configuration file and add it to the repository

Click on + →Click on New file.The name of the file is .gitlab-ci.yml

Copy this content and add it

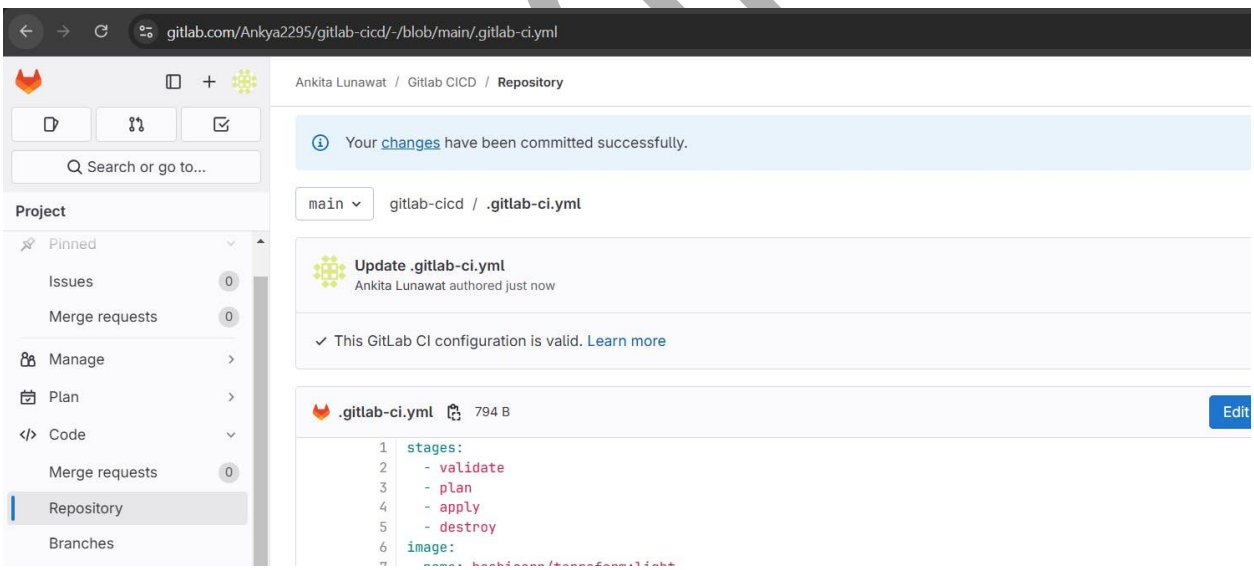
****

```
stages:
  - validate
  - plan
  - apply
  - destroy
image:
  name: hashicorp/terraform:light
  entrypoint:
    - '/usr/bin/env'
    - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
before_script:
  - export AWS_ACCESS_KEY=${AWS_ACCESS_KEY_ID}
  - export AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
  - rm -rf .terraform
  - terraform --version
  - terraform init
validate:
  stage: validate
  script:
    - terraform validate
plan:
  stage: plan
  script:
    - terraform plan -out=tfplan
  artifacts:
    paths:
      - tfplan
apply:
  stage: apply
```

```
  script:
    - terraform apply -auto-approve tfplan
  dependencies:
    - plan
destroy:
  stage: destroy
  script:
    - terraform init
    - terraform destroy -auto-approve
  when: manual
  dependencies:
    - apply
```

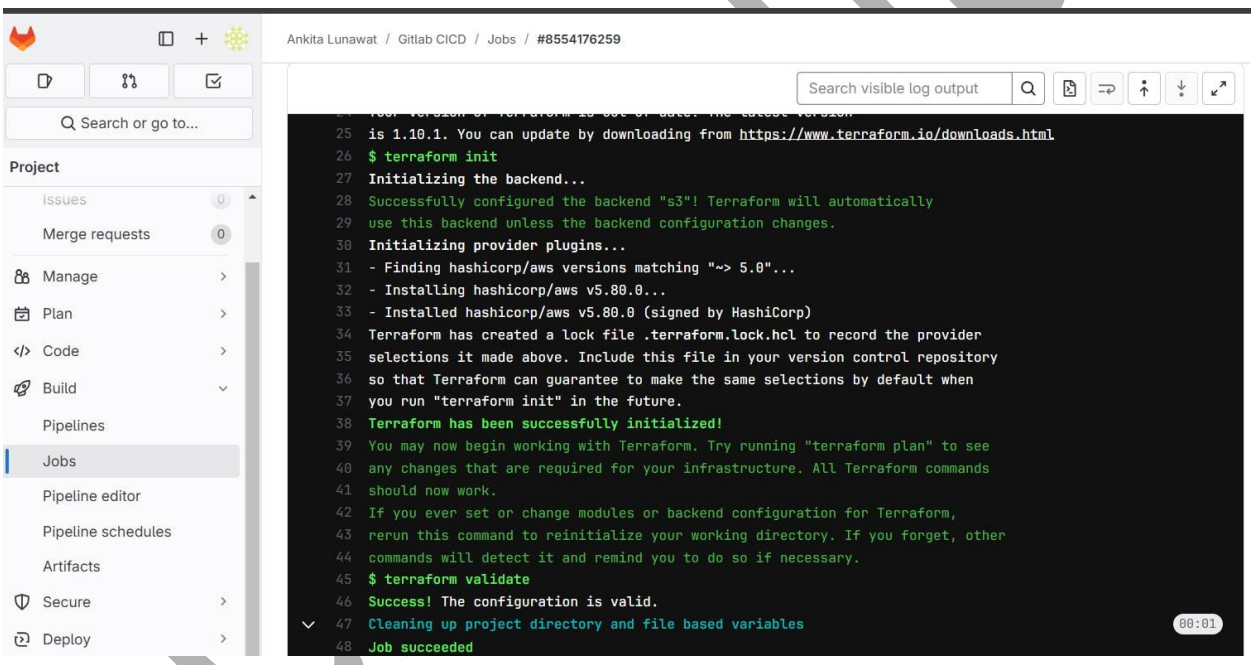
****

Click commit. It will automatically start the build. Now click on Build → Pipelines



It will open like this. Click on validate to see the build output.

Initialized and validated terraform code.



Ankita Lunawat / Gitlab CICD / Jobs / #8554396973

Search visible log output

```
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.80.0...
- Installed hashicorp/aws v5.80.0 (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.
$ terraform apply -auto-approve tfplan
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Creation complete after 16s [id=i-09d8fb7f071540ff2]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
Cleaning up project directory and file based variables
Job succeeded
```

00:00

Click on Jobs to come back. See plan output.

Ankita Lunawat / Gitlab CICD / Jobs / #8554352181

Search visible log output

```
          + encrypted              = (known after apply)
          + iops                   = (known after apply)
          + kms_key_id             = (known after apply)
          + tags_all               = (known after apply)
          + throughput             = (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.
──────────────────────────────────────────────────────────────────────────────
Saved the plan to: tfplan
To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
Uploading artifacts for successful job                                    00:02
Uploading artifacts...
tfplan: found 1 matching artifact files and directories
WARNING: Upload request redirected                  location=https://gitlab.com/api/v4/jobs/8554352181/artifacts?artifact_format=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=8554352181 responseStatus=201 Created token=glcbt-66
Cleaning up project directory and file based variables                    00:01
Job succeeded
```

Now come back and see apply output also.

Ankita Lunawat / Gitlab CICD / Jobs / #8554396973

Search visible log output

```
is 1.10.1. You can update by downloading from https://www.terraform.io/downloads.html
$ terraform init
Initializing the backend...
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.80.0...
- Installed hashicorp/aws v5.80.0 (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.
$ terraform apply -auto-approve tfplan
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Creation complete after 16s [id=i-09d8fb7f071540ff2]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
Cleaning up project directory and file based variables
Job succeeded
```

00:00

Go to the AWS console to check if the EC2 instance is provisioned.

Instances (1/3) Info

Last updated 1 minute ago | 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	Public IPv4 DNS	Pul
	i-02ee93f78ee2a252d	Running	t2.medium	2/2 checks passec	View alarms +	ap-south-1a	ec2-52-66-80-208.ap-s...	52.
Jenkins-sonar	i-09d8fb7f071540ff2	Terminated	t2.medium	–	View alarms +	ap-south-1b	–	–
Jenkins	i-0b6a8aa9d5838c12f	Running	t2.medium	Initializing	View alarms +	ap-south-1b	ec2-43-204-219-224.ap...	43.

i-0b6a8aa9d5838c12f (Jenkins)

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary Info

Instance ID
i-0b6a8aa9d5838c12f

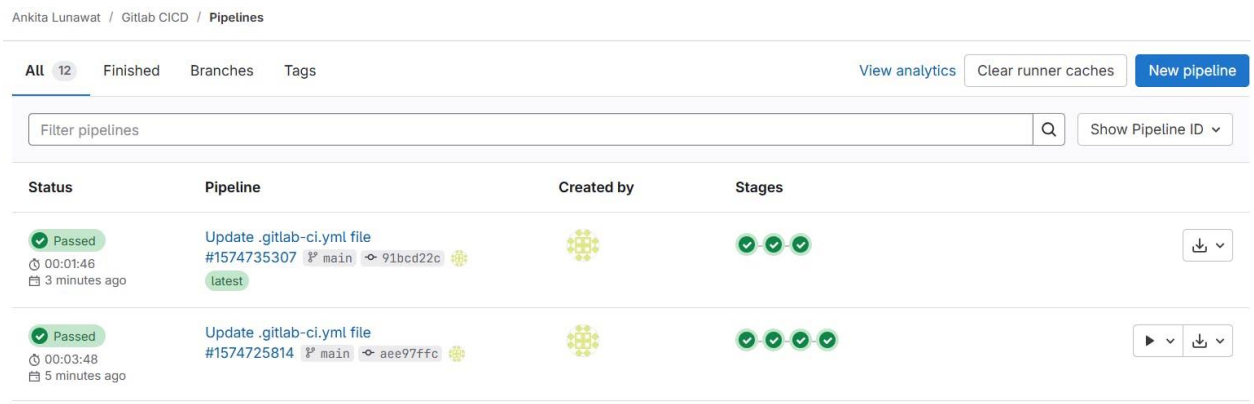
IPv6 address
–

Public IPv4 address
43.204.219.224 | open address

Instance state
Running

Private IPv4 addresses
172.31.12.26

Public IPv4 DNS
ec2-43-204-219-224.ap-south-1.compute.amazonaws.com | open address



Connect to the instance using Putty or MobaXterm with the following commands.

****

cd /

cd Ansible #mkdir used in shell script

cd ANSIBLE #cloned repo

ls #to see ansible playbook

Now come back to

****

cd /home/ubuntu

cd /var/log/

ls

cat user-data.log

****

The Ansible playbook has finished running to install Jenkins.

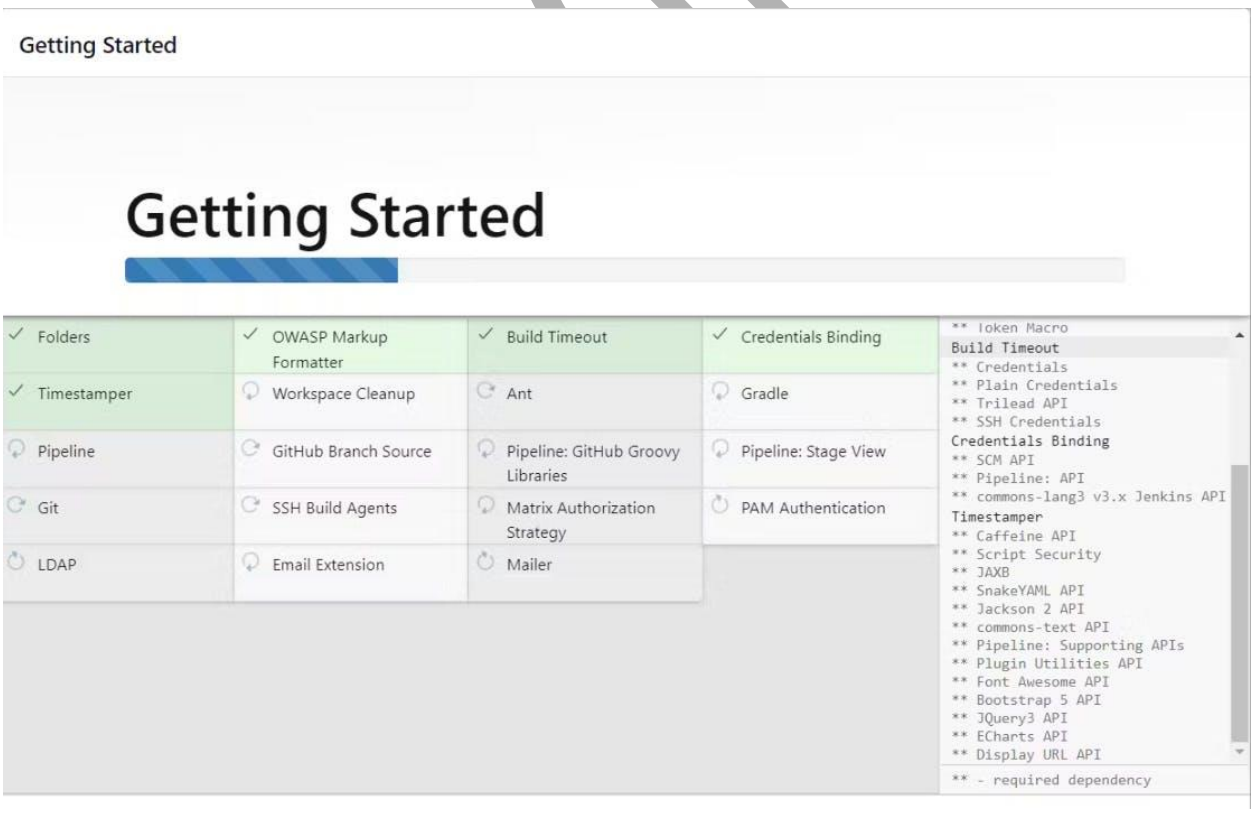
```
No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
/Ansible
Cloning into 'ANSIBLE'...
[WARNING]: Unable to parse /Ansible/ANSIBLE/localhost as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that
the implicit localhost does not match 'all'

PLAY [Install jenkins] *********************************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Update all packages to their latest version] *****************************
changed: [localhost]

TASK [download jenkins key] ****************************************************
changed: [localhost]

TASK [Add Jenkins repo] ********************************************************
changed: [localhost]

TASK [Update all packages to their latest version] *****************************
ok: [localhost]

TASK [Install fontconfig] ******************************************************
```

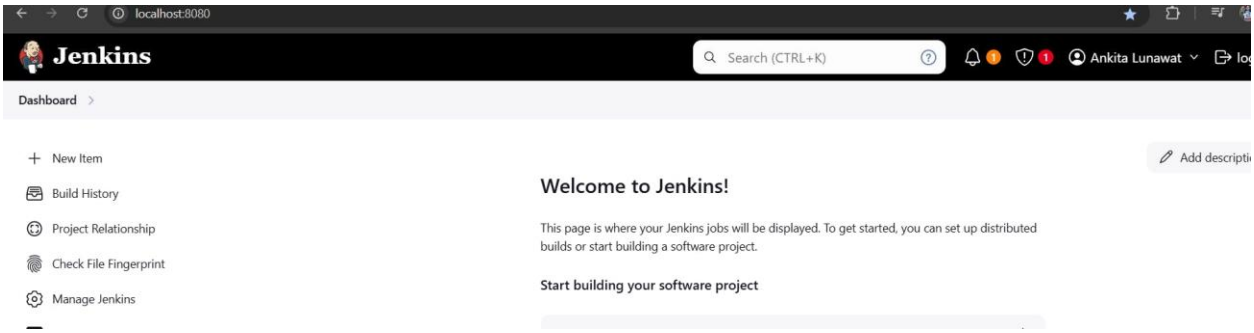
Copy the public IP of the EC2 instance.

<Ec2-public-ip:8080>

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

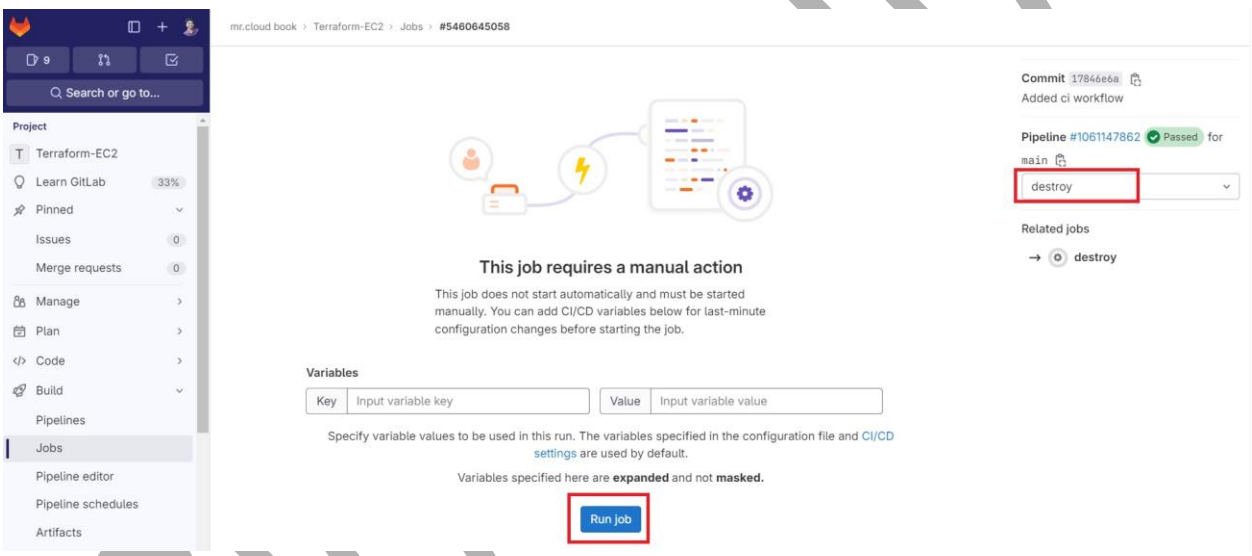
Copy the password and sign in.





8.Destroy

Return to GitLab and manually select destroy to delete resources by clicking on >> in stages.Now select destroy and the Run job



Ankita Lunawat / Gitlab CICD / Jobs / #8554798628

Search visible log output

```
      - name                   = "Jenkins-Security Group" -> null
      - owner_id               = "703671930975" -> null
      - revoke_rules_on_delete = false -> null
      - tags                   = {
          - "Name" = "Jenkins-sg"
        } -> null
      - tags_all               = {
          - "Name" = "Jenkins-sg"
        } -> null
      - vpc_id                 = "vpc-0fb8b7c8be4f0d84a" -> null
    }
Plan: 0 to add, 0 to change, 2 to destroy.
aws_instance.web: Destroying... [id=i-0b6a8aa9d5838c12f]
aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 10s elapsed]
aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 20s elapsed]
aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 30s elapsed]
aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 40s elapsed]
aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 50s elapsed]
aws_instance.web: Destruction complete after 53s
aws_security_group.Jenkins-sg: Destroying... [id=sg-00e944efd1ccd5b0e]
aws_security_group.Jenkins-sg: Destruction complete after 1s
Destroy complete! Resources: 2 destroyed.
Cleaning up project directory and file based variables
Job succeeded
```

00:01

Destroy is completed.

Ankita Lunawat / Gitlab CICD / Pipelines / #1574774542

Update .gitlab-ci.yml file

Passed **Ankita Lunawat** created pipeline for commit `615b6c01` 7 minutes ago, finished 2 minutes ago

For `main`

latest 4 jobs 3.23 3 minutes 14 seconds, queued for 0 seconds

Pipeline Jobs 4 Tests 0

validate	plan	apply	destroy
validate	plan	apply	destroy

CICD looks like this.

In conclusion, GitLab CI/CD simplifies and speeds up the software development process, allowing developers to concentrate on creating innovative and valuable software while the CI/CD pipeline manages the rest. As you start using GitLab CI/CD, remember that it's not just about automation; it's about delivering better software faster, a goal every development team can support.

Embrace GitLab CI/CD to enhance your software projects with automation, collaboration, and quality assurance, allowing your code to improve rapidly and reliably.