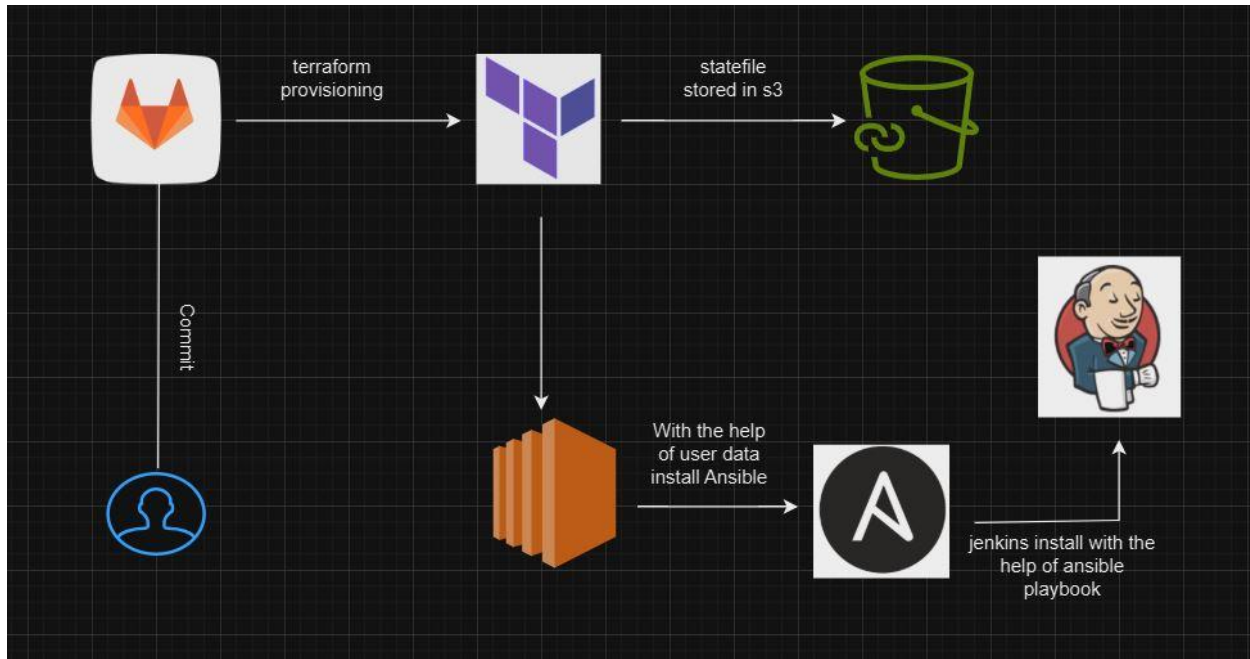


# Managing Infrastructure with GitLab CI/CD for Terraform: Plan, Validate, Apply, Destroy



## 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.

[IAM](#) > [Users](#) > [Terraform](#) > Create access key

## Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

- Step 1
- Access key best practices & alternatives
- Step 2 - optional
- Set description tag
- Step 3
- **Retrieve access keys**

## 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
 AKIA2HVQ5SBP6MFWV6W	 ***** <a href="#">Show</a>

### 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](#).

[Download .csv file](#)

[Done](#)

The screenshot shows the AWS IAM console interface. On the left is a navigation sidebar with sections for 'Identity and Access Management (IAM)' and 'Access reports'. The main content area is titled 'Terraform' and includes a 'Summary' section with fields for ARN, Console access, Created date, and Last console sign-in. Below this is a 'Permissions policies (1)' section with a search bar and a table listing the 'AdministratorAccess' policy, which is AWS managed and attached directly.

## 2.Create S3 Bucket

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

## 3.Create Gitlab Account

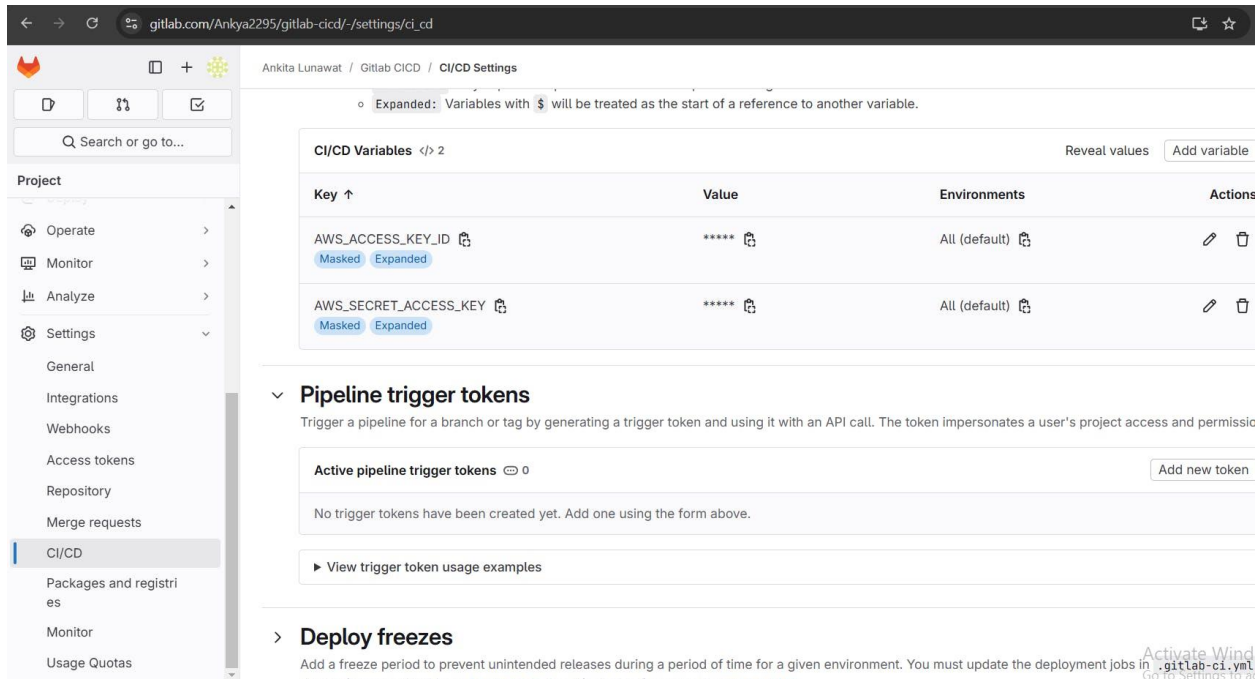
Go to [gitlab.com](https://gitlab.com)

Lets create New project/Repository

The screenshot shows the 'Create blank project' form in the GitLab web interface. The form includes fields for 'Project name' (filled with 'My awesome project'), 'Project URL' (filled with 'https://gitlab.com/' and a dropdown for 'Pick a group or namespace'), and 'Project slug' (filled with 'my-awesome-project'). There is also a section for 'Project deployment target (optional)' and a 'Visibility Level' section with radio buttons for 'Private', 'Internal', and 'Public' (which is selected).

## 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.



Expanded: Variables with \$ will be treated as the start of a reference to another variable.

Key ↑	Value	Environments	Actions
AWS_ACCESS_KEY_ID	*****	All (default)	<a href="#">Masked</a> <a href="#">Expanded</a> <a href="#">Edit</a> <a href="#">Delete</a>
AWS_SECRET_ACCESS_KEY	*****	All (default)	<a href="#">Masked</a> <a href="#">Expanded</a> <a href="#">Edit</a> <a href="#">Delete</a>

**Pipeline trigger tokens**  
Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions.

Active pipeline trigger tokens 0 [Add new token](#)

No trigger tokens have been created yet. Add one using the form above.

[View trigger token usage examples](#)

**Deploy freezes**  
Add a freeze period to prevent unintended releases during a period of time for a given environment. You must update the deployment jobs in .gitlab-ci.yml

## 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-GitlabCI/CD.git  
cd ANSIBLE  
ansible-playbook -i localhost Jenkins-playbook.yml
```

Ankita Lunawat

\*\*\*\*\*

[backend.tf](https://www.terraform.io/docs/backends/config.html)

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

Ankita Lunawat / Gitlab CICD

 **Gitlab CICD** 

main ▾


gitlab-cicd / 



+ ▾







Find file


Edit ▾

Code ▾

 **Update .gitlab-ci.yml file**  
Ankita Lunawat authored 49 minutes ago

 615b6c01  History

Name	Last commit	Last update
 .gitlab-ci.yml	Update .gitlab-ci.yml file	49 minutes ago
 README.md	Initial commit	2 hours ago
 backend.tf	Update backend.tf	1 hour ago
 install_jenkins.sh	Update jenkins.sh	2 hours ago
 main.tf	Update main.tf	1 hour ago
 provider.tf	Add new file	2 hours ago

 README.md

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

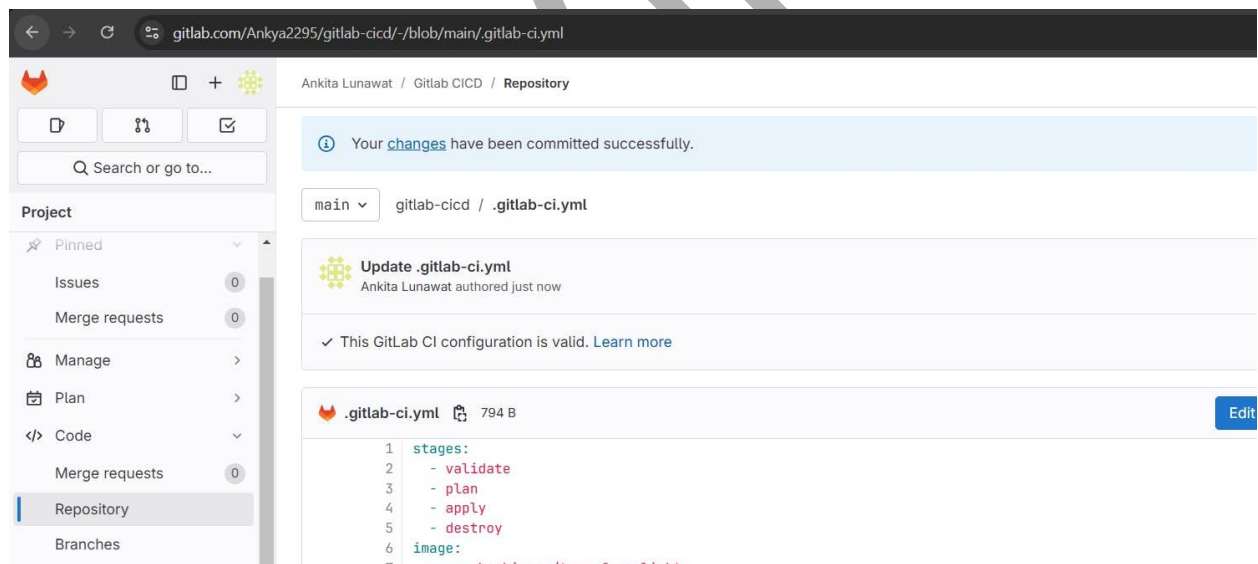
when: manual

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.

## Update backend.tf

✓ Passed Ankita Lunawat created pipeline for commit c27e5b8c 1 hour ago, finished 1 hour ago

For main

1 job 0.45 27 seconds, queued for 2 seconds

Pipeline Jobs 1 Tests 0

validate

✓ validate

Initialized and validated terraform code.

The screenshot displays the GitLab CI/CD interface. On the left, a sidebar shows the project navigation menu with 'Jobs' selected. The main area shows the pipeline details for commit c27e5b8c. A single job named 'validate' is shown as passed. Below this, a terminal window displays the output of the 'terraform init' and 'terraform validate' commands. The output indicates that Terraform has been successfully initialized and validated.

```
25 is 1.10.1. You can update by downloading from https://www.terraform.io/downloads.html
26 $ terraform init
27 Initializing the backend...
28 Successfully configured the backend "s3"! Terraform will automatically
29 use this backend unless the backend configuration changes.
30 Initializing provider plugins...
31 - Finding hashicorp/aws versions matching "~> 5.0"...
32 - Installing hashicorp/aws v5.80.0...
33 - Installed hashicorp/aws v5.80.0 (signed by HashiCorp)
34 Terraform has created a lock file .terraform.lock.hcl to record the provider
35 selections it made above. Include this file in your version control repository
36 so that Terraform can guarantee to make the same selections by default when
37 you run "terraform init" in the future.
38 Terraform has been successfully initialized!
39 You may now begin working with Terraform. Try running "terraform plan" to see
40 any changes that are required for your infrastructure. All Terraform commands
41 should now work.
42 If you ever set or change modules or backend configuration for Terraform,
43 rerun this command to reinitialize your working directory. If you forget, other
44 commands will detect it and remind you to do so if necessary.
45 $ terraform validate
46 Success! The configuration is valid.
47 Cleaning up project directory and file based variables
48 Job succeeded
```

Search visible log output

```
30 Initializing the backend...
31 Successfully configured the backend "s3"! Terraform will automatically
32 use this backend unless the backend configuration changes.
33 Initializing provider plugins...
34 - Finding hashicorp/aws versions matching "~> 5.0"...
35 - Installing hashicorp/aws v5.80.0...
36 - Installed hashicorp/aws v5.80.0 (signed by HashiCorp)
37 Terraform has created a lock file .terraform.lock.hcl to record the provider
38 selections it made above. Include this file in your version control repository
39 so that Terraform can guarantee to make the same selections by default when
40 you run "terraform init" in the future.
41 Terraform has been successfully initialized!
42 You may now begin working with Terraform. Try running "terraform plan" to see
43 any changes that are required for your infrastructure. All Terraform commands
44 should now work.
45 If you ever set or change modules or backend configuration for Terraform,
46 rerun this command to reinitialize your working directory. If you forget, other
47 commands will detect it and remind you to do so if necessary.
48 $ terraform apply -auto-approve tfplan
49 aws_instance.web: Creating...
50 aws_instance.web: Still creating... [10s elapsed]
51 aws_instance.web: Creation complete after 16s [id=i-09d8fb7f071540ff2]
52 Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
53 Cleaning up project directory and file based variables
54 Job succeeded
```

00:00

Click on Jobs to come back. See plan output.

Search visible log output

```
105     + encrypted      = (known after apply)
106     + iops           = (known after apply)
107     + kms_key_id     = (known after apply)
108     + tags_all       = (known after apply)
109     + throughput     = (known after apply)
110     + volume_id      = (known after apply)
111     + volume_size    = 8
112     + volume_type    = (known after apply)
113   }
114 }
115 Plan: 1 to add, 0 to change, 0 to destroy.
116
117 Saved the plan to: tfplan
118 To perform exactly these actions, run the following command to apply:
119   terraform apply "tfplan"
120 Uploading artifacts for successful job 00:02
121 Uploading artifacts...
122 tfplan: found 1 matching artifact files and directories
123 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/8554352181/artifacts?artifact_format=zip&artifact_type=archive new-url=https://gitlab.com
124 WARNING: Retrying... context=artifacts-uploader error=request redirected
125 Uploading artifacts as "archive" to coordinator... 201 Created id=8554352181 responseStatus=201 Created token=glcblt-66
126 Cleaning up project directory and file based variables 00:01
127 Job succeeded
```

Now come back and see apply output also.



Search visible log output

```

28 is 1.10.1. You can update by downloading from https://www.terraform.io/downloads.html
29 $ terraform init
30 Initializing the backend...
31 Successfully configured the backend "s3"! Terraform will automatically
32 use this backend unless the backend configuration changes.
33 Initializing provider plugins...
34 - Finding hashicorp/aws versions matching "~> 5.0"...
35 - Installing hashicorp/aws v5.80.0...
36 - Installed hashicorp/aws v5.80.0 (signed by HashiCorp)
37 Terraform has created a lock file .terraform.lock.hcl to record the provider
38 selections it made above. Include this file in your version control repository
39 so that Terraform can guarantee to make the same selections by default when
40 you run "terraform init" in the future.
41 Terraform has been successfully initialized!
42 You may now begin working with Terraform. Try running "terraform plan" to see
43 any changes that are required for your infrastructure. All Terraform commands
44 should now work.
45 If you ever set or change modules or backend configuration for Terraform,
46 rerun this command to reinitialize your working directory. If you forget, other
47 commands will detect it and remind you to do so if necessary.
48 $ terraform apply -auto-approve tfplan
49 aws_instance.web: Creating...
50 aws_instance.web: Still creating... [10s elapsed]
51 aws_instance.web: Creation complete after 16s [id=i-09d8fb7f071540ff2]
52 Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
53 Cleaning up project directory and file based variables
54 Job succeeded

```

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 < 1 >

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4
<input type="checkbox"/> i-02ee93f78ee2a252d	<input checked="" type="checkbox"/> Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1a	ec2-52-66-80-208.ap-s...	52.	
<input type="checkbox"/> Jenkins-sonar	<input type="checkbox"/> Terminated	t2.medium	-	View alarms +	ap-south-1b	-	-	
<input checked="" type="checkbox"/> Jenkins	<input checked="" type="checkbox"/> 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	Public IPv4 address 43.204.219.224   open address	Private IPv4 addresses 172.31.12.26
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-43-204-219-224.ap-south-1.compute.amazonaws.com   open address



All12FinishedBranchesTags

View analyticsClear runner cachesNew pipeline

Filter pipelines

Show Pipeline ID

Status	Pipeline	Created by	Stages
<div>✓ Passed</div> <div>00:01:46</div> <div>3 minutes ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#1574735307main91bcd22c</div> <div>latest</div>		<div></div> <div></div> <div></div>
<div>✓ Passed</div> <div>00:03:48</div> <div>5 minutes ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#1574725814mainaee97ffc</div>		<div></div> <div></div> <div></div> <div></div>

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>

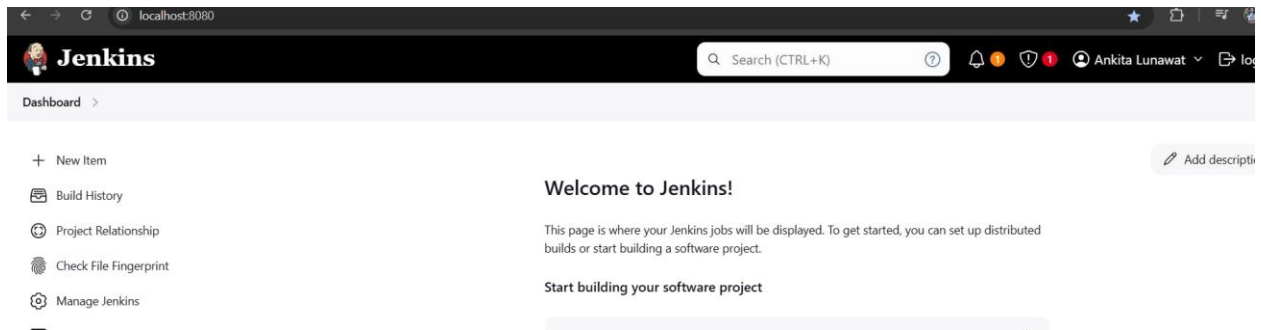
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the password and sign in.

Getting Started

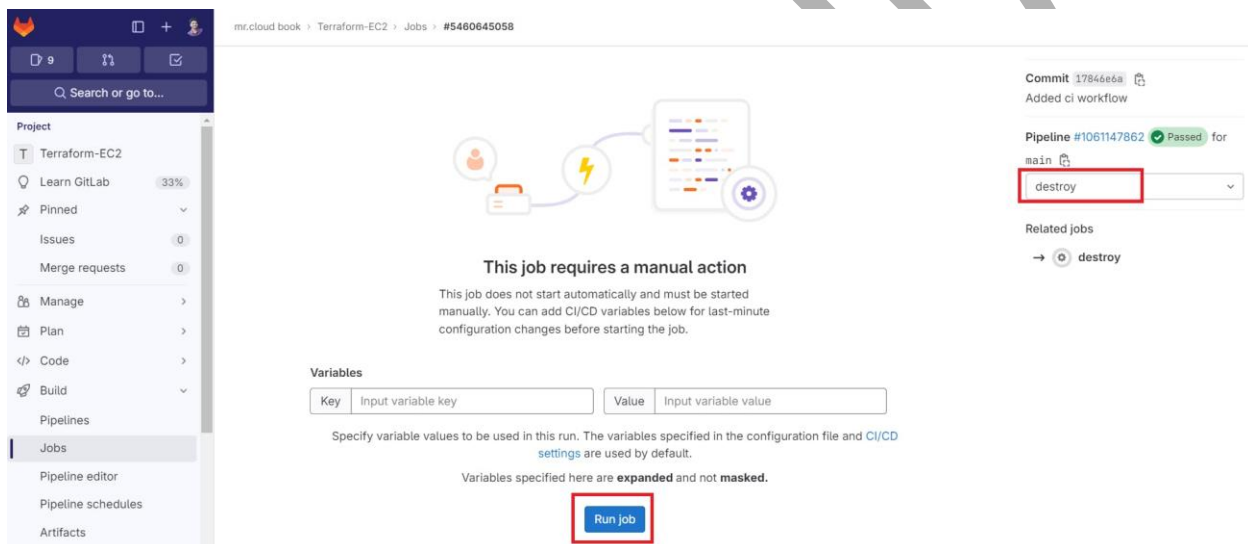
## Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding	** token Macro
✓ Timestampers	Workspace Cleanup	Ant	Gradle	Build Timeout
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline: Stage View	** Credentials
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication	** Plain Credentials
LDAP	Email Extension	Mailer		** Trilead API
				** SSH Credentials
				Credentials Binding
				** SCM API
				** Pipeline: API
				** commons-lang3 v3.x Jenkins API
				Timestampers
				** Caffeine API
				** Script Security
				** JAXB
				** SnakeYAML API
				** Jackson 2 API
				** commons-text API
				** Pipeline: Supporting APIs
				** Plugin Utilities API
				** Font Awesome API
				** Bootstrap 5 API
				** JQuery3 API
				** ECharts API
				** Display URL API
				** - required dependency



## 8.Destroy

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



Search visible log output

```
224 - name = "Jenkins-Security Group" -> null
225 - owner_id = "703671930975" -> null
226 - revoke_rules_on_delete = false -> null
227 - tags = {
228   - "Name" = "Jenkins-sg"
229 } -> null
230 - tags_all = {
231   - "Name" = "Jenkins-sg"
232 } -> null
233 - vpc_id = "vpc-0fb8b7c8be4f0d84a" -> null
234 }
235 Plan: 0 to add, 0 to change, 2 to destroy.
236 aws_instance.web: Destroying... [id=i-0b6a8aa9d5838c12f]
237 aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 10s elapsed]
238 aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 20s elapsed]
239 aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 30s elapsed]
240 aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 40s elapsed]
241 aws_instance.web: Still destroying... [id=i-0b6a8aa9d5838c12f, 50s elapsed]
242 aws_instance.web: Destruction complete after 53s
243 aws_security_group.Jenkins-sg: Destroying... [id=sg-00e944efd1ccd5b0e]
244 aws_security_group.Jenkins-sg: Destruction complete after 1s
245 Destroy complete! Resources: 2 destroyed.
✓ 246 Cleaning up project directory and file based variables
247 Job succeeded
```

Destroy is completed.

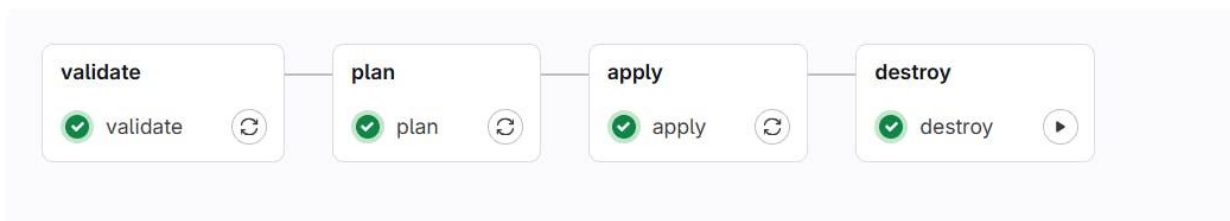
## Update .gitlab-ci.yml file

✓ 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



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.

Cloudhub