Runbook for Terraform

A. Installing Terraform

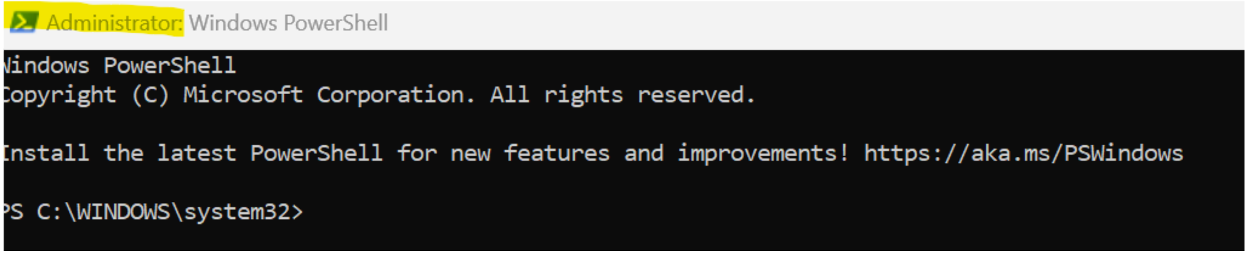
There exist two options to installing terraform. You can choose to install terraform using package managers such as **Chocolatey** for Windows, **Brew** for macOS or you can choose to install terraform using Binary files (manual process). Here's the official link with the installation guide to terraform [Install Terraform | Terraform | HashiCorp Developer]

1. Installation option with package Managers

❖ **Windows users**

Step1. Install chocolatey ([Chocolatey Software | Installing Chocolatey])

- Open Powershell as an administrator (Navigate to your search menu on windows and search for **Powershell** >> Right click on **Windows Powershell** >> select **Run as administrator** >> when propted to allowthis app make changes to your computer, select **YES >>** This will open powershell as an adminas seen in the screenshot below)



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32>
```

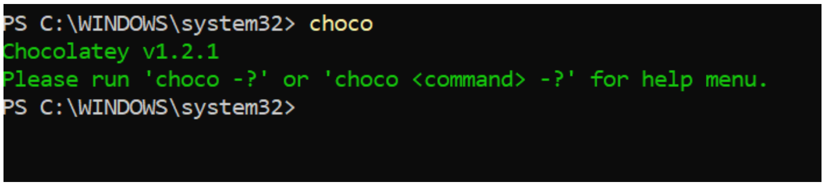
- Run the following commands (Note the third command should be run as one)

□ Get-ExecutionPolicy

□ Set-ExecutionPolicy AllSigned

□ Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

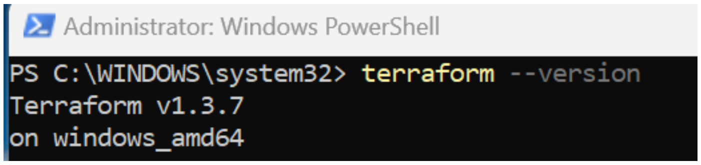
- Wait a few seconds for the command to complete
- Type **choco** and confirm it is installed as seen in the screenshot below



```
PS C:\WINDOWS\system32> choco
Chocolatey v1.2.1
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\WINDOWS\system32>
```

Step2. Install terraform by running the following commands:

- □ **Choco install terraform**
- □ Type **terraform - -version** to confirm it is installed



```
PS C:\WINDOWS\system32> terraform --version
Terraform v1.3.7
on windows_amd64
```

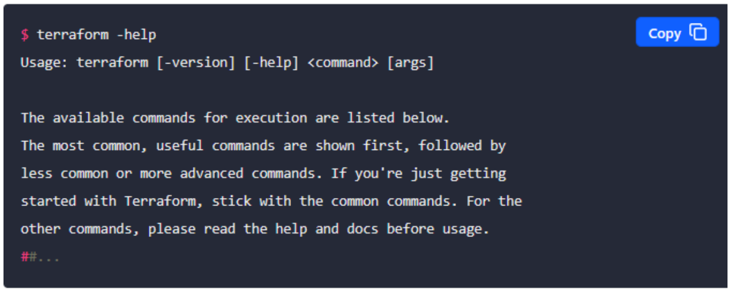
❖ Installation for macOS users

Step1. Install brew ([Homebrew — The Missing Package Manager for macOS (or Linux)](https://brew.sh)**)**

- Open Terminal and type the following commands:
  - □ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
  - □ Type your admin password for your mac laptop if prompted (note that you won't see your keystrokes in the Terminal window — it's a security measure)
  - □ (echo; echo 'eval "$(/opt/homebrew/bin/brew shellenv)"') >> /Users/gen/.zprofile
  - □ eval "$(/opt/homebrew/bin/brew shellenv)"
  - □ Check if brew is installed by typing **brew**

Step2. Install terraform

- Run the following commands on your terminal
  - □ brew tap hashicorp/tap
  - □ brew install hashicorp/tap/terraform
  - □ check if terraform is installed by typing **terraform -help** on your terminal



```
$ terraform -help
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.
##...
```

2. **Manual Installation option** (Please only use this option if you did not succeed with the package manager option above)

❖ **Windows users**

Step1. Here is the link to download the binary files for the different OS [Install | Terraform | HashiCorp Developer]

□ Follow the link and choose the **Windows** option

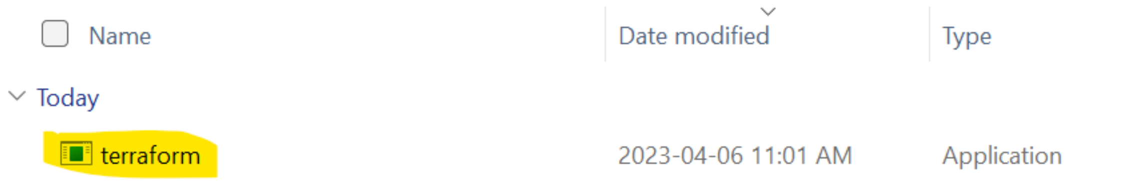
□ Under the **Binary download for Windows** click **Download** on the **386** option. This will download the binary file for terraform.

□ Navigate to **file explorer** in your **Downloads** folder on your computer and you will see the zipped file you just download

Today

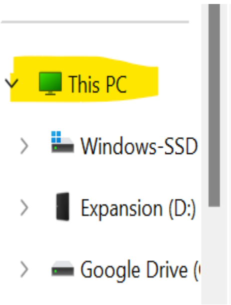
terraform_1.4.4_windows_386	2023-04-06 10:58 AM	WinZip File

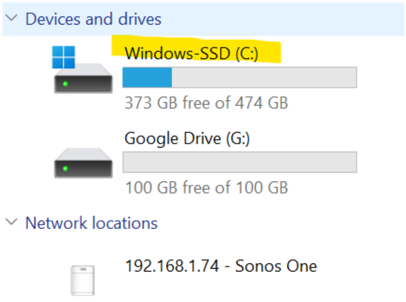
□ Right click on the **zipped file** and select **Extract all** and click on **extract,** copy the **terraform** file that displays after extract is complete

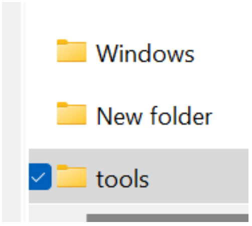


Name	Date modified	Type
Today		
terraform	2023-04-06 11:01 AM	Application

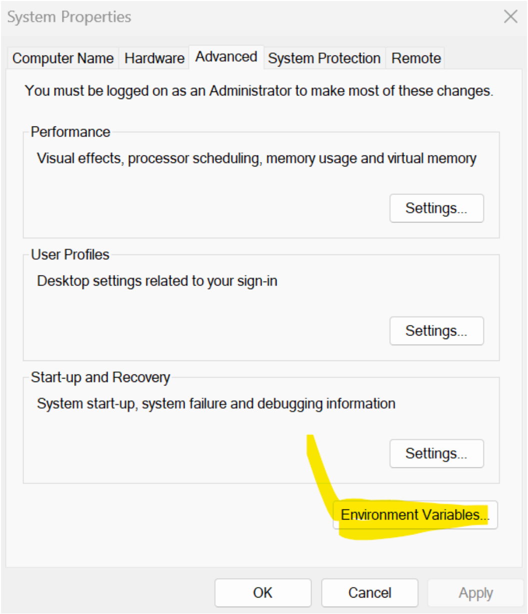
□ Click This PC on your computer >> double click on drive C (c:) >> Create a folder and name it **tools** >> paste the terraform file copied earlier into this tools folder







- □ On your windows search menu, search for **Edit the System Environment Variables** and click on it when it pops up on the search menu
- □ Click on Environment variables at the next prompt>> double click on **Path** >> select **New** >> Paste **C:\tools** in path section (as seen in the fourth screenshot below) and click **Ok , Ok and OK**
- □ Then type **terraform** on your terminal to confirm terraform has been installed successfully

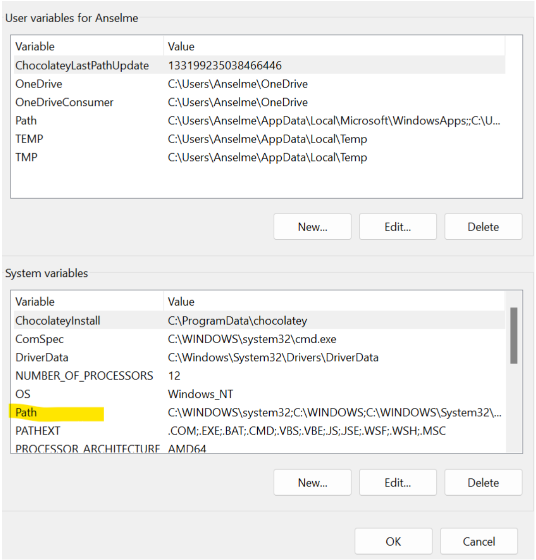


You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage and virtual memory

User Profiles
Desktop settings related to your sign-in

Start-up and Recovery
System start-up, system failure and debugging information

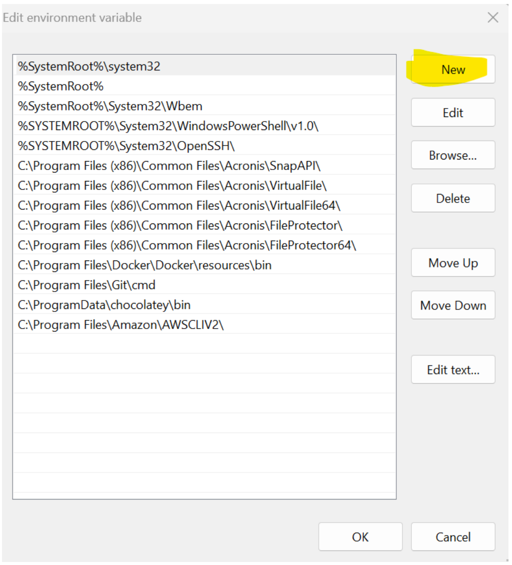


User variables for Anselme

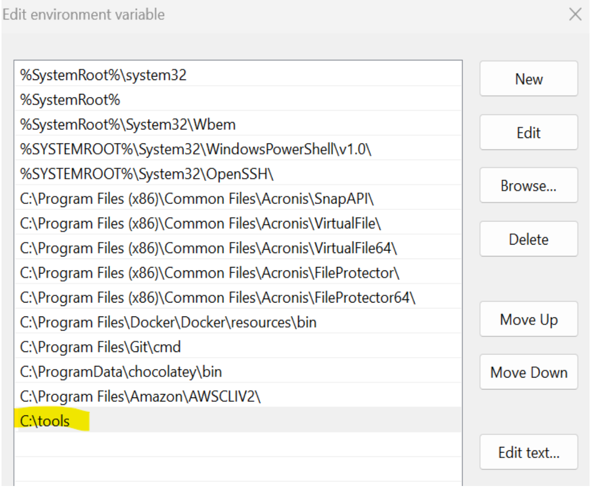
Variable	Value
ChocolateyLastPathUpdate	133199235038466446
OneDrive	C:\Users\Anselme\OneDrive
OneDriveConsumer	C:\Users\Anselme\OneDrive
Path	C:\Users\Anselme\AppData\Local\Microsoft\WindowsApps;;C:\U...
TEMP	C:\Users\Anselme\AppData\Local\Temp
TMP	C:\Users\Anselme\AppData\Local\Temp

System variables

Variable	Value
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64



```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files (x86)\Common Files\Acronis\SnapAPI\
C:\Program Files (x86)\Common Files\Acronis\VirtualFile\
C:\Program Files (x86)\Common Files\Acronis\VirtualFile64\
C:\Program Files (x86)\Common Files\Acronis\FileProtector\
C:\Program Files (x86)\Common Files\Acronis\FileProtector64\
C:\Program Files\Docker\Docker\resources\bin
C:\Program Files\Git\cmd
C:\ProgramData\chocolatey\bin
C:\Program Files\Amazon\AWSCLIV2\
```



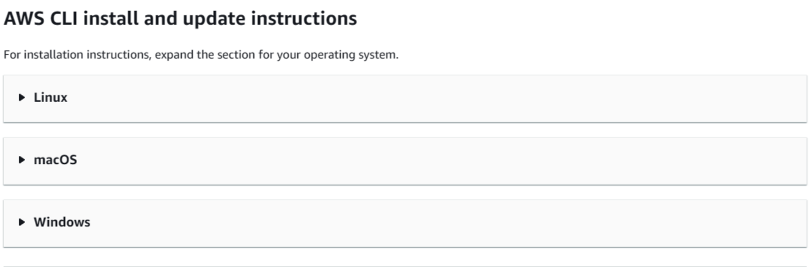
```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files (x86)\Common Files\Acronis\SnapAPI\
C:\Program Files (x86)\Common Files\Acronis\VirtualFile\
C:\Program Files (x86)\Common Files\Acronis\VirtualFile64\
C:\Program Files (x86)\Common Files\Acronis\FileProtector\
C:\Program Files (x86)\Common Files\Acronis\FileProtector64\
C:\Program Files\Docker\Docker\resources\bin
C:\Program Files\Git\cmd
C:\ProgramData\chocolatey\bin
C:\Program Files\Amazon\AWSCLIV2\
C:\tools
```

B. Install AWS CLI

Step1. Open link below for installation guide

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Step2. Select the OS of your personal computer

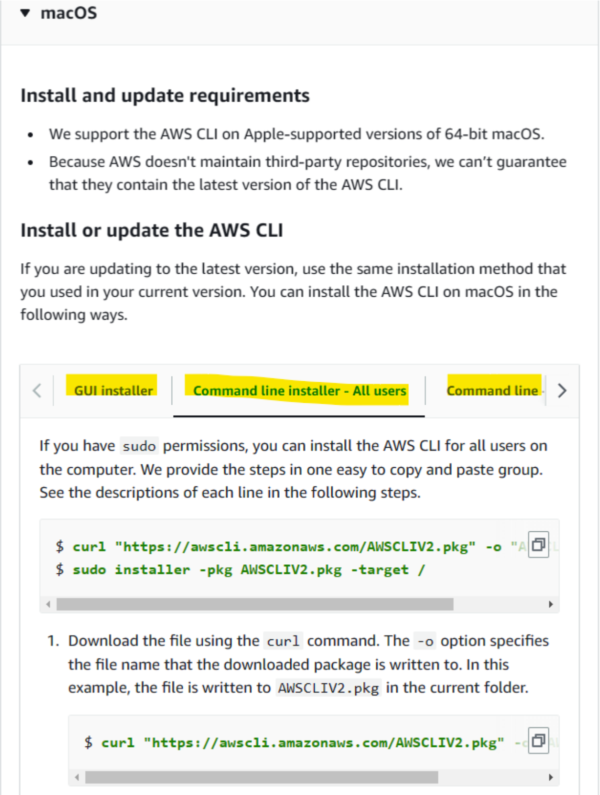


AWS CLI install and update instructions

For installation instructions, expand the section for your operating system.

1. For macOS users

Step3. Select the macOS installing option and select the installation method and follow the guide



Install and update requirements

- We support the AWS CLI on Apple-supported versions of 64-bit macOS.
- Because AWS doesn't maintain third-party repositories, we can't guarantee that they contain the latest version of the AWS CLI.

Install or update the AWS CLI

If you are updating to the latest version, use the same installation method that you used in your current version. You can install the AWS CLI on macOS in the following ways.

If you have `sudo` permissions, you can install the AWS CLI for all users on the computer. We provide the steps in one easy to copy and paste group. See the descriptions of each line in the following steps.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "A
$ sudo installer -pkg AWSCLIV2.pkg -target /
```

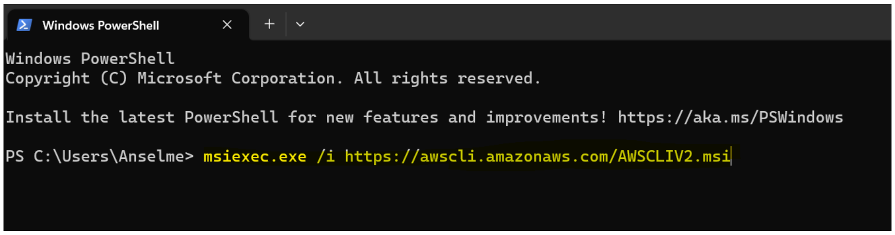
1. Download the file using the `curl` command. The `-o` option specifies the file name that the downloaded package is written to. In this example, the file is written to `AWSCLIV2.pkg` in the current folder.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -
```

2. For Windows Users

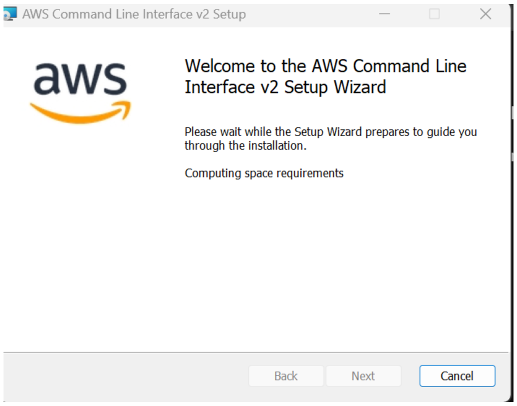
Step3. open the powershell terminal and run the following commands

- msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Anselme> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

- You will see the AWS CLi installation wizard display and just follow the prompts with next until you install and finish



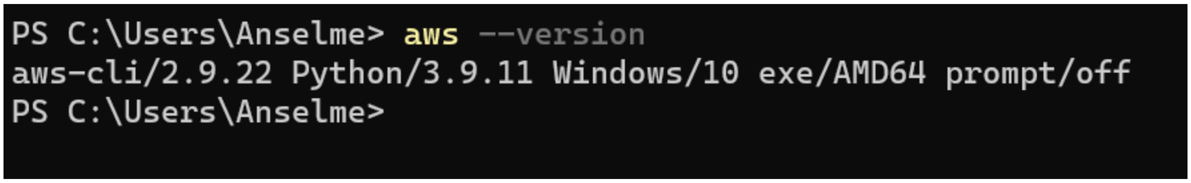
AWS Command Line Interface v2 Setup

Welcome to the AWS Command Line Interface v2 Setup Wizard

Please wait while the Setup Wizard prepares to guide you through the installation.

Computing space requirements

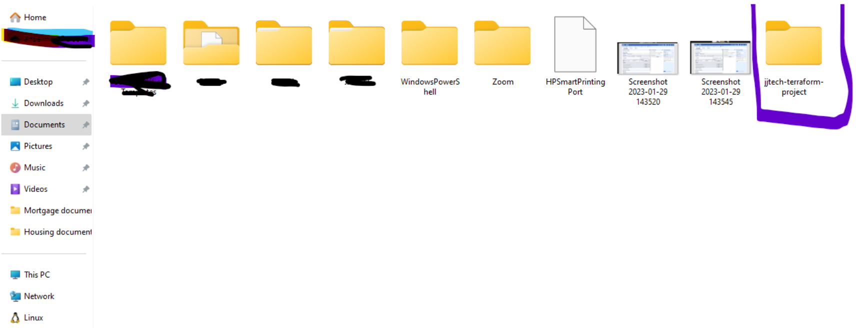
- To confirm that you have the aws cli installed, run **aws - -version** and you should see an output similar to the screenshot below.



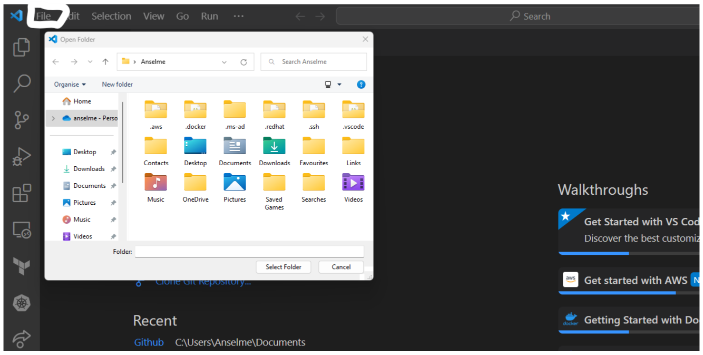
```
PS C:\Users\Anselme> aws --version
aws-cli/2.9.22 Python/3.9.11 Windows/10 exe/AMD64 prompt/off
PS C:\Users\Anselme>
```

C. Integrating VS CODE with Github

1. Navigate to windows explorer and in Documents directory, create a folder called **jjtech-terraform-projects**



2. Open VSCODE IDE, navigate to file section and click on open folder



3. Open jjtech-terraform-projects created above

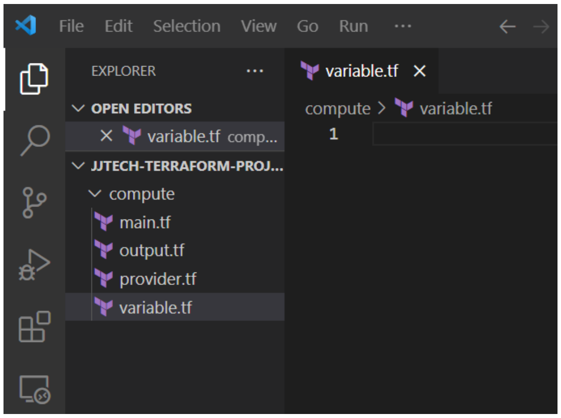
4. Create a Sub-folder called compute. Then create the following files:

   - provider.tf

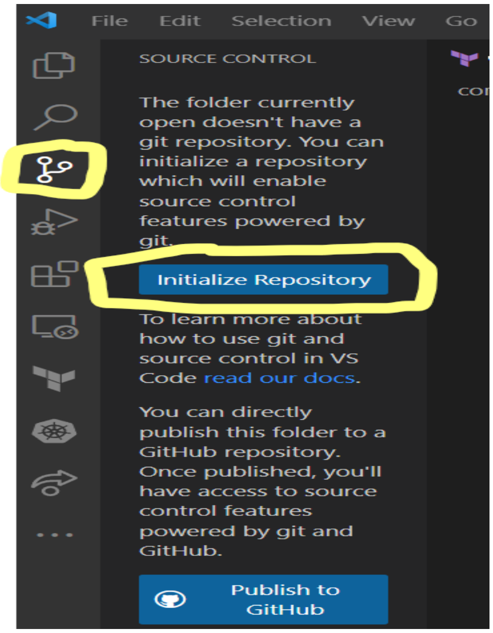
   - main.tf

   - variable.tf

   - output.tf



5. Click on the **source control** icon on the left section of VSCODE and click on **initialise Repository.** If prompted to login into Github**,** follow prompts for login and enter username and password.

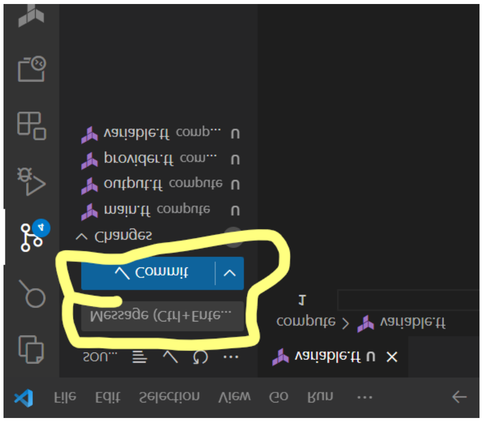


The folder currently open doesn't have a git repository. You can initialize a repository which will enable source control features powered by git.

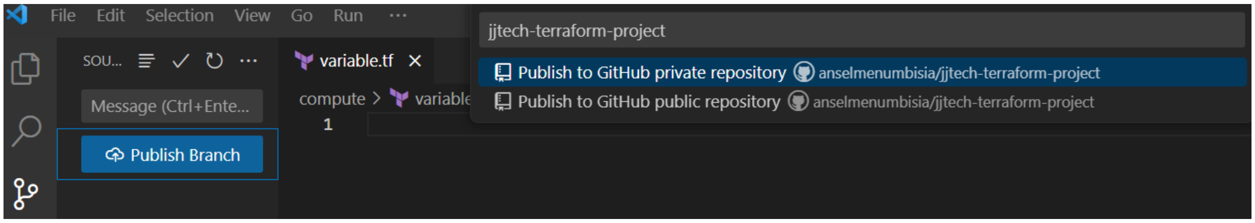
To learn more about how to use git and source control in VS Code read our docs.

You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by git and GitHub.

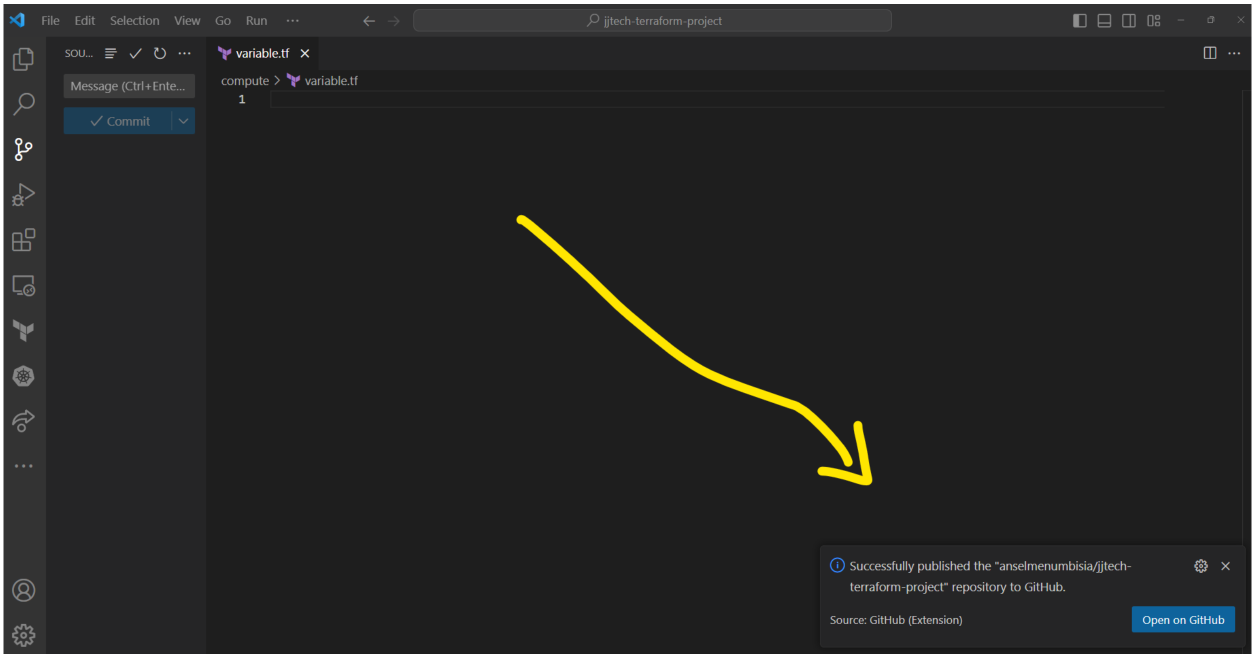
6. If repository was successfully initialised, you should see the following screen



7. Stage all changes by clicking on the + symbol next to **Changes**. (hover your cursor aroud **Changes** to view the + symbol). Once all changes are staged, fill in your first commit message and click on **Commit**.
8. Click **Publish Branch**. Pay attention to the repository name that will be created (You can modify the name). Click publish to github public repository and wait …


File
Edit
Selection
View
Go
Run
SOU...
Message (Ctrl+Ente...
Publish Branch
variable.tf
compute > variable
1
jjtech-terraform-project
Publish to GitHub private repository anselmenumbisia/jjtech-terraform-project
Publish to GitHub public repository anselmenumbisia/jjtech-terraform-project

9. You will notice a message at the bottom right corner of your screen to confirm that the repository has been successfully published.



Successfully published the "anselmenumbisia/jjtech-terraform-project" repository to GitHub.

Source: GitHub (Extension)

10. You can now open github to confirm that the repository was successfully created

D. Runbook Terraform provisioners

Local-Exec Provisioners:

In the example below, we create an EC2 instance in AWS. It makes use of a local-exec provisioner to save the private_ip address of the instance which is created in a text file called **private_ip.txt**, creates a folder called **Test** and moves the **private_ip.txt** file into the folder **Test**. This provisioner executes in the same working directory where **terraform apply** is run once the provisioning is successful.

1. Create a file and name it local-exec.tf
2. Copy and paste the terraform resource block for ec2 instance below
3. Run terraform apply and check to confirm that the task was accomplished.
4. Run terraform destroy and also confirm that the second provisioner is run and that the **destruction.txt** file is created with the message **“Destruction successful”**

```
resource "aws_instance" "web" {
 ami          =  var.ami
 instance_type = var.instance_type[2]
 key_name = "jenkinskp"

provisioner "local-exec" {
   when = create
   command = "echo 'This is my private IP ${self.private_ip}'>> private_ip.txt && mkdir Test && mv private_ip.txt Test"
}

provisioner "local-exec" {
   when = destroy
   command = "echo 'Destruction successful'>> destruction.txt"
}


}
```

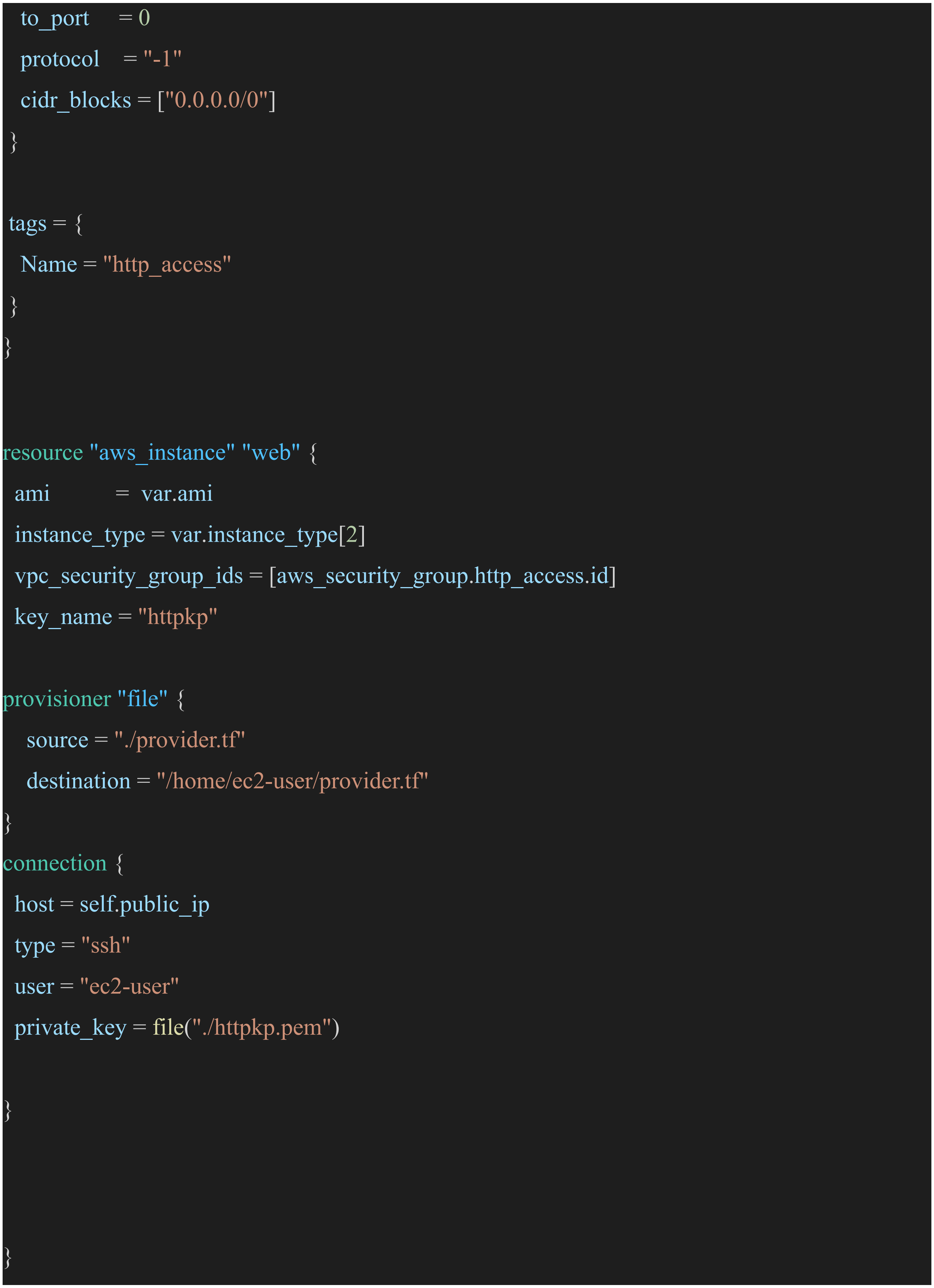
File Provisioners:

In this example, we want to copy the provider.tf file existing in our terraform directory to the home directory of ec2 when the instance is created. For this, we need to configure elements for the connection block such as security group and ssh key

1. Create a file in your terraform directory and name it file-provisioner.tf

2. Copy the code below and paste
3. Navigate to your aws console and create a keypair and name it **httpkp**
4. Copy the private key file (**httpkp.pem**) that was downloaded into your terraform directory
5. Apply your code
6. Ssh into ec2 instance to confirm that the provider.tf was successfully copy in the home directory of ec2 user.

```
resource "aws_security_group" "http_access" {
 name        = "http_access"
 description = "Allow HTTP inbound traffic"

 ingress {
  description = "HTTP Access"
  from_port   = 80
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
 }

 ingress {
  description = "SSH Access"
  from_port   = 22
  to_port     = 22
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
 }

 egress {
  from_port   = 0
```



```
   to_port     = 0
   protocol    = "-1"
   cidr_blocks = ["0.0.0.0/0"]
 }

 tags = {
   Name = "http_access"
 }
}


resource "aws_instance" "web" {
  ami           =  var.ami
  instance_type = var.instance_type[2]
  vpc_security_group_ids = [aws_security_group.http_access.id]
  key_name = "httpkp"

provisioner "file" {
    source = "./provider.tf"
    destination = "/home/ec2-user/provider.tf"
}
connection {
  host = self.public_ip
  type = "ssh"
  user = "ec2-user"
  private_key = file("./httpkp.pem")

}



}
```

Remote-exec provisioners;

The example below performs a simple task of installing and starting nginx on the EC2 instance that is created by Terraform. Once the EC2 instance creation is successful, Terraform's remote-exec provisioner logs in to the instance via SSH using the connection block and executes the commands specified in the inline attribute array.

1. Create a file and name it remote-exec-provisioner.tf
2. Copy the code below and paste it into the file
3. Create a a file called shell file in your terraform directory and name it nginx.sh and paste the shell script below:

   ```
   #!/bin/bash
   sudo yum update -y
   sudo amazon-linux-extras install nginx1 -y
   sudo systemctl enable nginx
   sudo systemctl start nginx
   ```
4. Navigate to your aws console and create a keypair and name it **httpkp**
5. Copy the private key file (**httpkp.pem**) that was downloaded into your terraform directory
6. Apply your terraform script
7. Navigate to your AWS management console and get the public IP address of your instance
8. Paste the IP address copied on the browser and confirm you have a welcome to Nginx messgae

```
resource "aws_security_group" "http_access" {
 name        = "http_access"
 description = "Allow HTTP inbound traffic"
```

```
  ingress {
    description = "HTTP Access"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "SSH Access"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "http_access"
  }
}


resource "aws_instance" "web" {
  ami           =  var.ami
  vpc_security_group_ids = [aws_security_group.http_access.id]
  instance_type = var.instance_type[2]
  key_name = "httpkp"

provisioner "file" {
    source = "./nginx.sh"
    destination = "/home/ec2-user/nginx.sh"
}

provisioner "remote-exec" {
    #script = "./nginx.sh"
    inline = [
      "chmod 777 ./nginx.sh",
      "./nginx.sh"
    ]

}
connection {
  host = self.public_ip
  type = "ssh"
  user = "ec2-user"
  private_key = file("./httpkp.pem")

}



}
```

C. Deploying Terraform script using Gitlab CI/CD

Overview:

In this tutorial, we will integrate **Terraform** with **GitLab CI/CD** and create various resources on **AWS.**

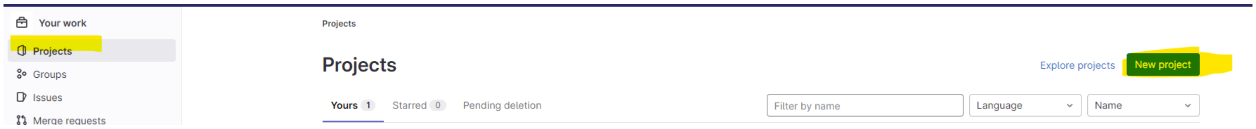
Prerequisite:

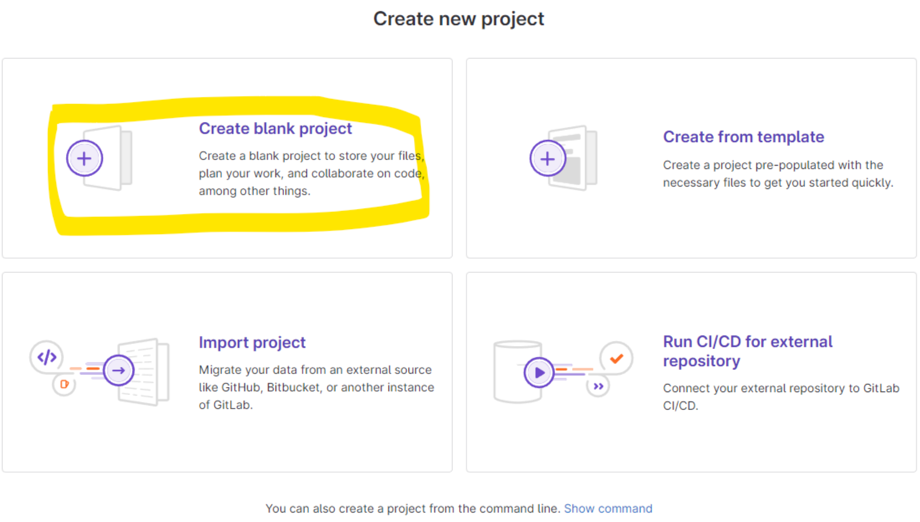
- **AWS & GitLab Account**
- Basic understanding of **AWS**, **Terraform** **&** **GitLab CI/CD**
- An **access key** & **secret key** created in the **AWS**

Lets, start with the configuration of the project:

Step1. Create a gitlab project

- Sign in to gitlab account. **Click on Create New Project, Create blank project** and fill out the information as required.





Create new project

Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

Create from template

Create a project pre-populated with the necessary files to get you started quickly.

Import project

Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.

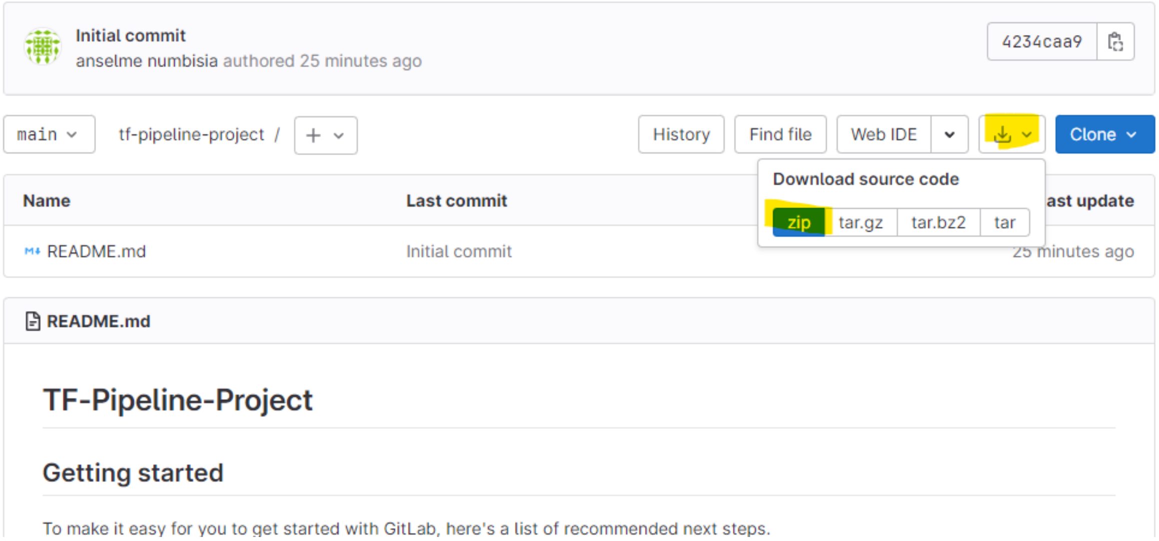
Run CI/CD for external repository

Connect your external repository to GitLab CI/CD.

You can also create a project from the command line. Show command

Step2. Create your terraform configuration file for resource creation. Get the sample code from [here]

- Download the source code from the repository



Initial commit

anselme numbisia authored 25 minutes ago

4234caa9

Name	Last commit	Last update
README.md	Initial commit	25 minutes ago

README.md

TF-Pipeline-Project

Getting started

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

- Next add a remote repository pointing to your own gitlab repository by running the command

- git init

- git remote add <remote name> <url> of your repository

- git status

- git add .

- git commit -m “initial pipeline commit”

- git push <remote name> master

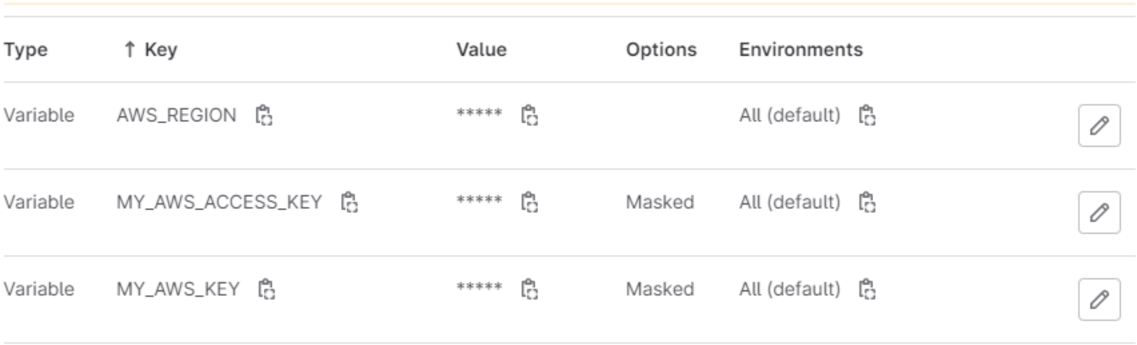
- When prompted for credentials, select password, then enter username and password

Step3. Create Environmental variables in Gitlab to store your AWS credentials

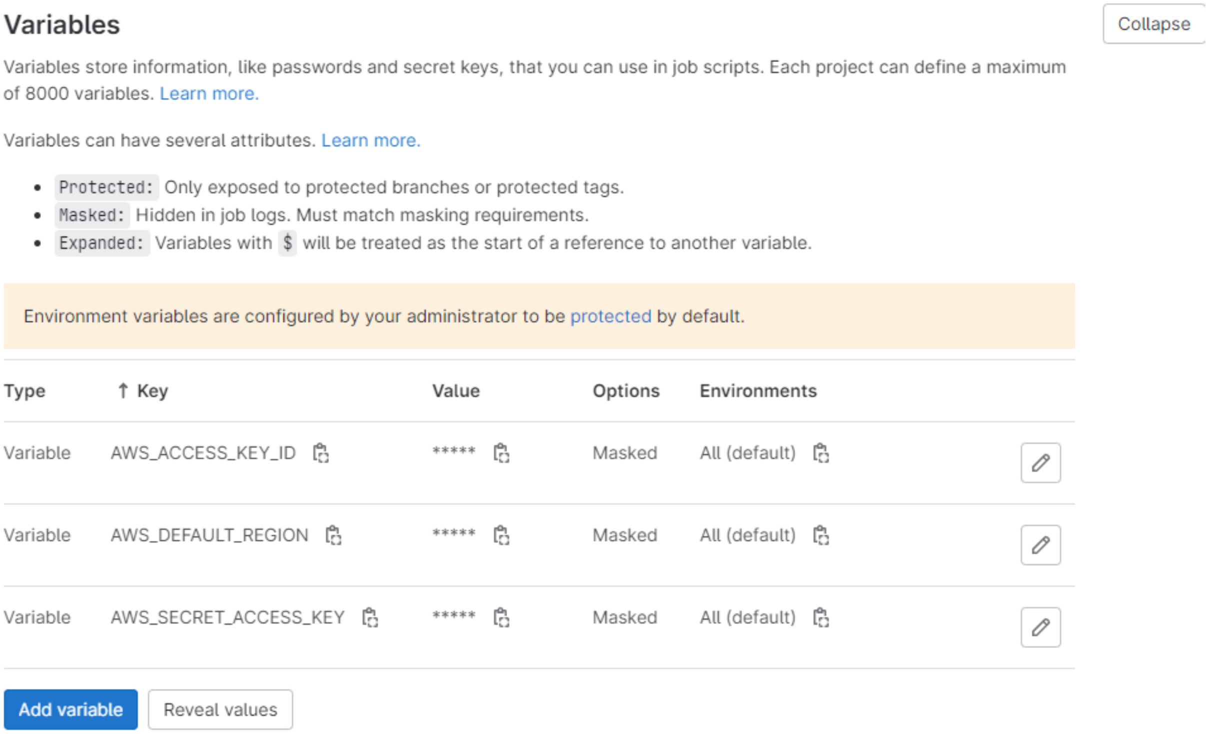
- In order to create the resources in the AWS account, we must need to have the **AWS Access Key** & **AWS Secret Key**

- Now, we need to store the **AWS Access Key** & **AWS Secret Key** in the secrets section of the repository

- Go to **settings** -> **CI/CD** -> **Variables** and click on **Expand** Under the variable section create the below variables and store your AWS_ACCESS_KEY_ID & AWS_SECRET_ACCESS_KEY**.** To easily get these values from your cli, run the command **notepad ~/.aws/credentials** from your cli.



Type	↑ Key	Value	Options	Environments
Variable	AWS_REGION	*****		All (default)
Variable	MY_AWS_ACCESS_KEY	*****	Masked	All (default)
Variable	MY_AWS_KEY	*****	Masked	All (default)



Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Each project can define a maximum of 8000 variables. Learn more.

Variables can have several attributes. Learn more.

- `Protected:` Only exposed to protected branches or protected tags.
- `Masked:` Hidden in job logs. Must match masking requirements.
- `Expanded:` Variables with `$` will be treated as the start of a reference to another variable.

Environment variables are configured by your administrator to be protected by default.

Type	↑ Key	Value	Options	Environments
Variable	AWS_ACCESS_KEY_ID	*****	Masked	All (default)
Variable	AWS_DEFAULT_REGION	*****	Masked	All (default)
Variable	AWS_SECRET_ACCESS_KEY	*****	Masked	All (default)

Step 4:- Create a **workflow** file

- Now in order to create the terraform resources automatically, we need to create a workflow file
- Create **.gitlab-ci.yml** file and add the below code to it
- The below job will run on every **push** and **pull request** that happens on the **main** branch. In the build section, I have specified the image name and commands in the script section.

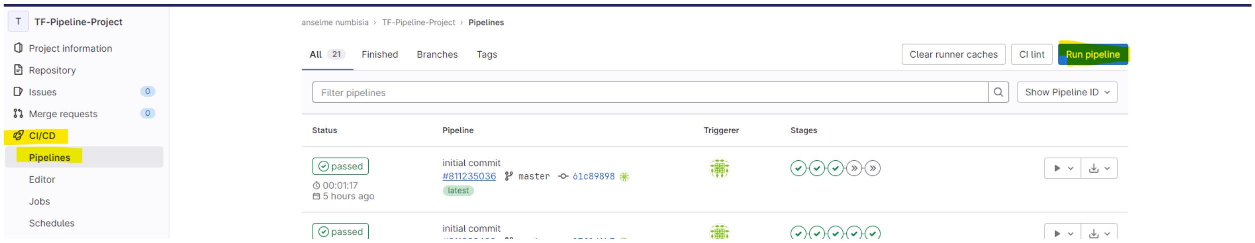
```
image:
  name: hashicorp/terraform
  entrypoint: [""]


variables:
  AWS_DEFAULT_REGION: ${AWS_REGION}
  AWS_ACCESS_KEY_ID: ${AWS_ACCESS_KEY_ID}
  AWS_SECRET_ACCESS_KEY : ${AWS_SECRET_ACCESS_KEY}

before_script:
  - rm -rf .terraform
  - terraform --version
  - terraform init -reconfigure

stages:
  - format
  - validate
  - plan
  - apply
  - destroy

format:
  stage: format
  script:
    - terraform fmt

validate:
  stage: validate
  script:
    - terraform validate
  dependencies:
    - format

plan:
  stage: plan
  script:
    - terraform plan -out "planfile"
  artifacts:
    paths:
      - planfile
  dependencies:
    - validate

apply:
  stage: apply
  allow_failure: true
  script:
    - terraform apply -auto-approve -input=false "planfile"
  dependencies:
    - plan
  when: manual

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

Step5. Push your source code to gitlab and navigate to CI/CD >> Pipeline and you should see a running pipeline job with the different stages.

Step6. If prompted to validate account with credit card, proceed to clicking on **Validate Account**

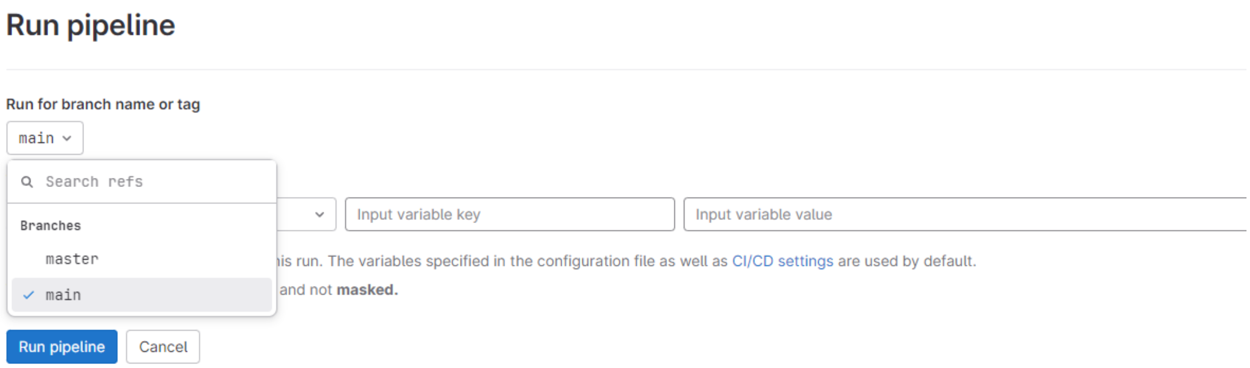
Step6. The default pipeline runs from the main branch. To modify this

- click on **Build** >> **Pipeline** >> and click **Run Pipeline** on far right end



Status	Pipeline	Triggerer	Stages
passed 00:01:17 5 hours ago	initial commit #811235036 master 61c89898 latest		✓✓✓»»
passed	initial commit		✓✓✓✓✓

- on the **Run Pipeline page**, change the branch from **main** to **master** and click **Run pipeline.** This should trigger a new pipeline job



Run pipeline

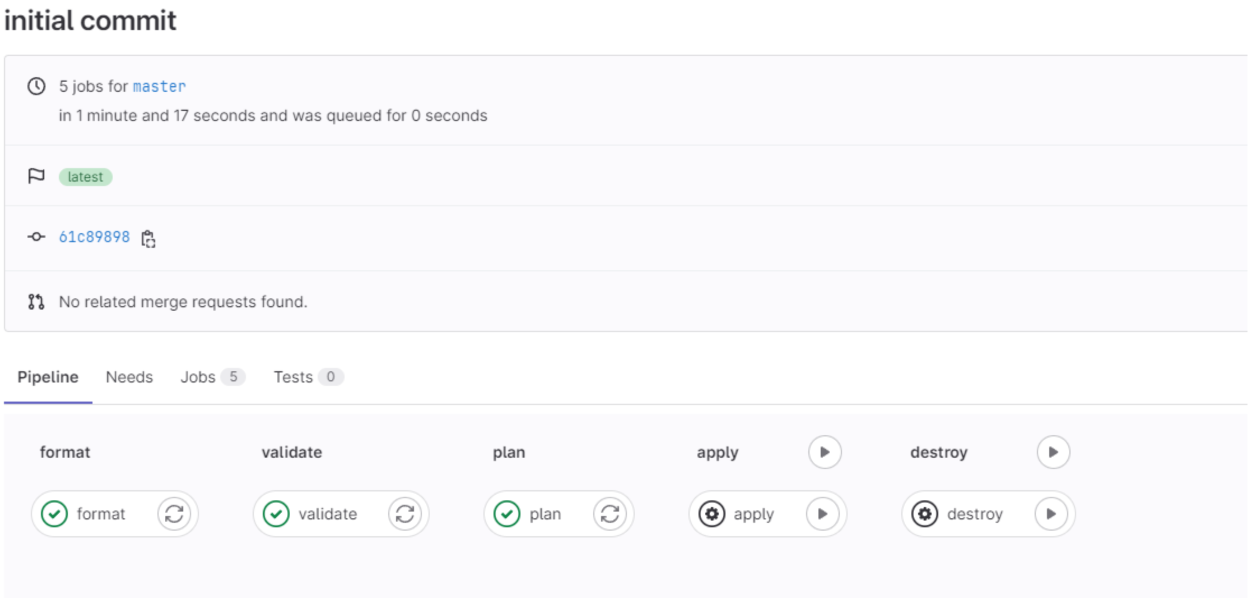
Run for branch name or tag

main

is run. The variables specified in the configuration file as well as CI/CD settings are used by default.

and not **masked.**

- You should now see the pipeline running



initial commit

5 jobs for master
in 1 minute and 17 seconds and was queued for 0 seconds

latest

61c89898

No related merge requests found.

- You can manually validate the apply stage and after resources create, you can then run the destroy stage as well manually.

D. Deploying AWS Resources using Terraform and Jenkins Pipeline

Overview:

Jenkins Pipeline

Jenkins is a self-contained, open source automation server used to automate tasks associated with building, testing, and delivering/deploying software. Jenkins Pipeline implements continuous deliver pipelines into Jenkins through use of plugins and a Jenkinsfile. The Jenkinsfile can be Declarative or Scripted and contains a list of steps for the pipeline to follow.

Prerequisites

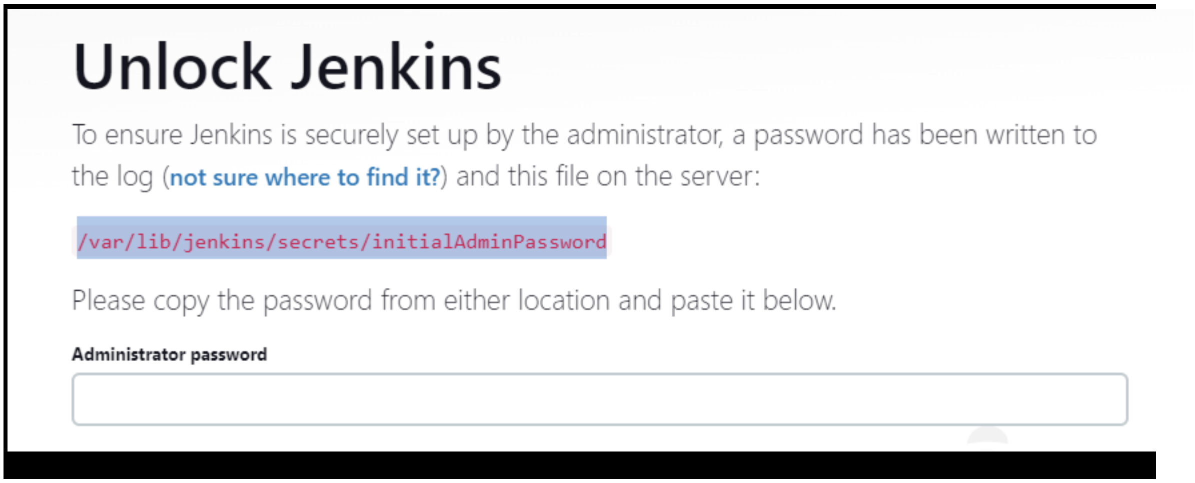
- Gitlab Account
- AWS CLI
- Install Terraform
- AWS Account
- AWS user with Admin permissions
- Preferred IDE (I used VSCode)

Getting started

1. Install Jenkins

- Create an **Amazon Linux 2 VM** instance and call it "Jenkins"

- Instance type: t2.micro
- Security Group (Open): 8080 and 22 to 0.0.0.0/0
- Key pair: Select or create a new keypair
- **Attach Jenkins server with IAM role having "AdministratorAccess"**
- User data (Copy the following user data): https://github.com/cvamsikrishna11/devops-fully-automated/blob/installations/jenkins-maven-ansible-setup.sh
- Launch Instance
- After launching this Jenkins server, attach a tag as **Key=Application, value=Jenkins**
- **Copy the public IP of your Jenkins server and run with on a browser and add :8080 example x.x.x.x:8080**
- **When prompted for the password, Ssh into your Jenkins server and run the command** **sudo cat** /var/lib/jenkins/secrets/initialAdminPassword Get the password and paste in required box



Unlock Jenkins

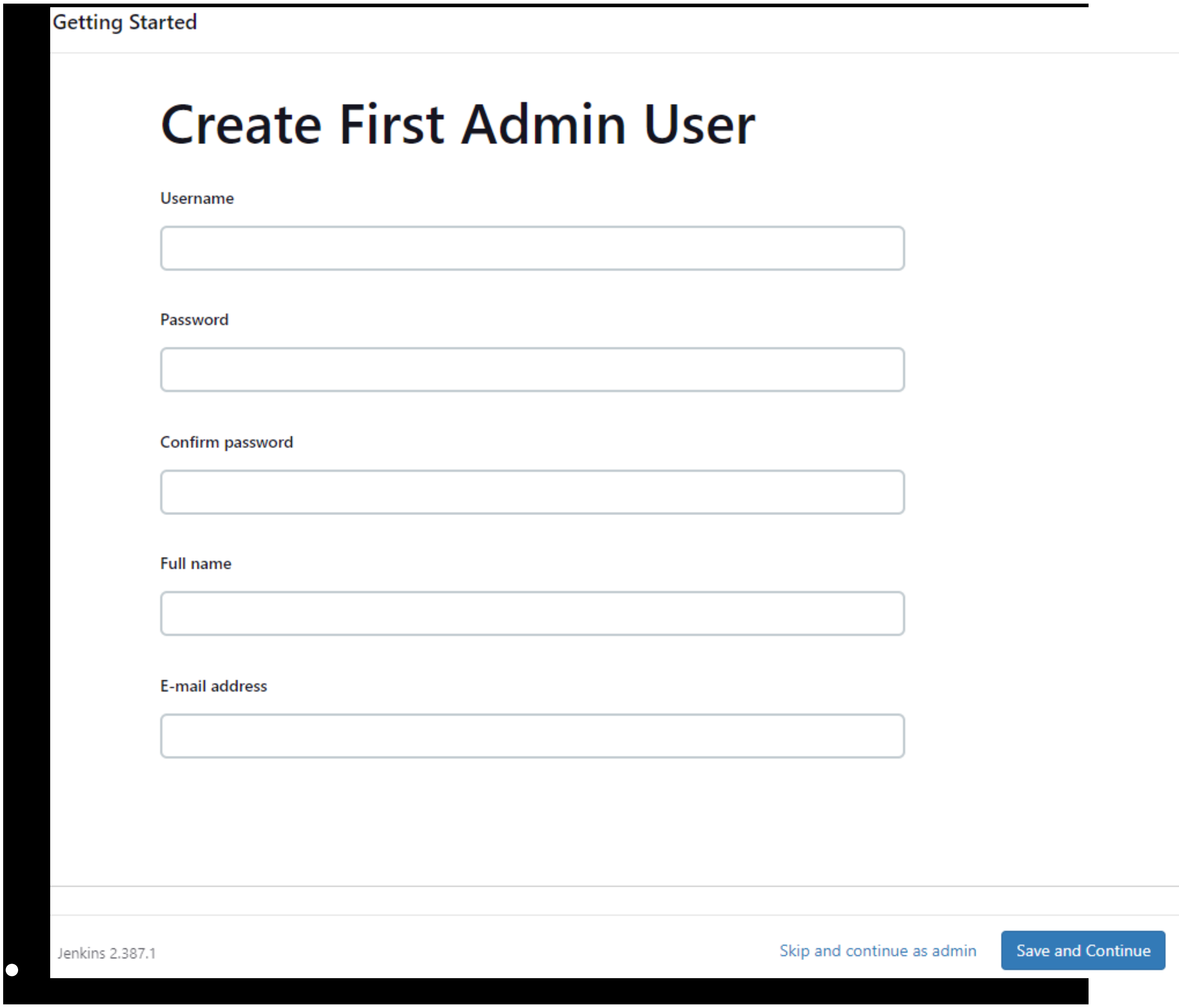
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

- **Click on install suggested plugins**
- **Fill out the form to create first time admin user and follow the prompts at the bottom right corner to access Jenkins.**



Getting Started

Create First Admin User

Username

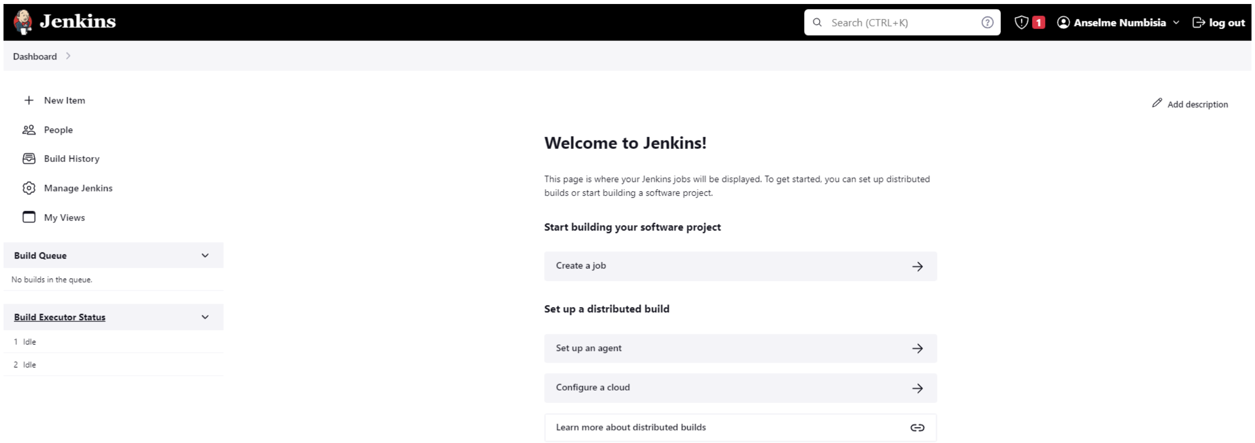
Password

Confirm password

Full name

E-mail address

Jenkins 2.387.1



Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Set up a distributed build

Build Queue

No builds in the queue.

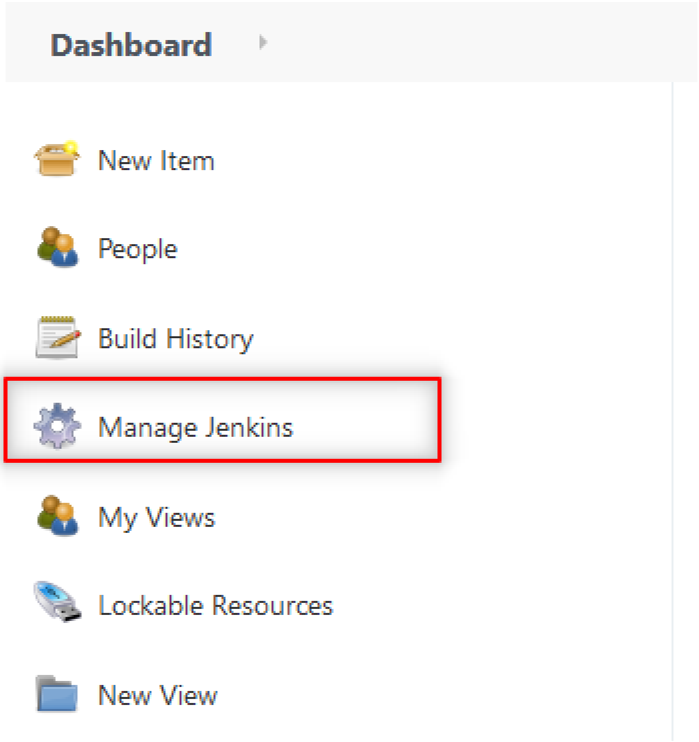
Build Executor Status

1 Idle

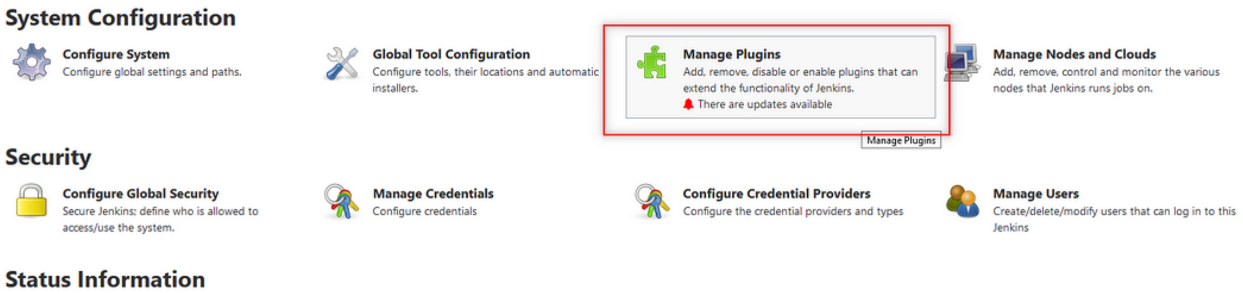
2 Idle

2. Install/configure terraform and gitlab plugins in Jenkins

   a. Click **Manage Jenkins** from left hand navigation.



b. Select **Manage Plugins** from **System Configuration** section.


System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
There are updates available

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

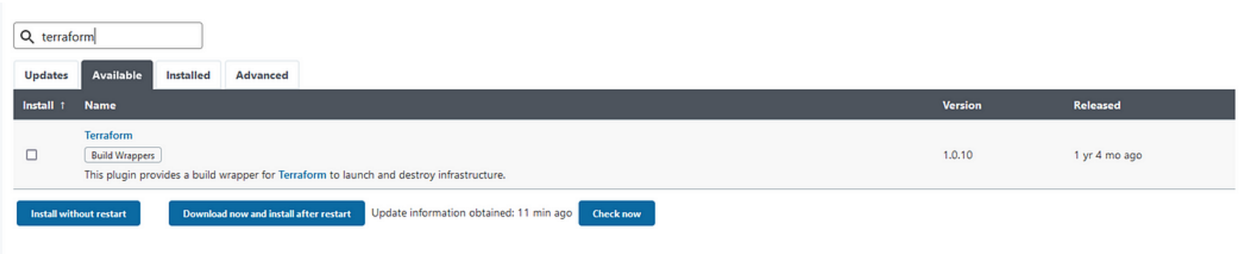
Manage Credentials
Configure credentials

Configure Credential Providers
Configure the credential providers and types

Manage Users
Create/delete/modify users that can log in to this Jenkins

Status Information

   c. Click the **Available** tab and search **Terraform** and then **gitlab**

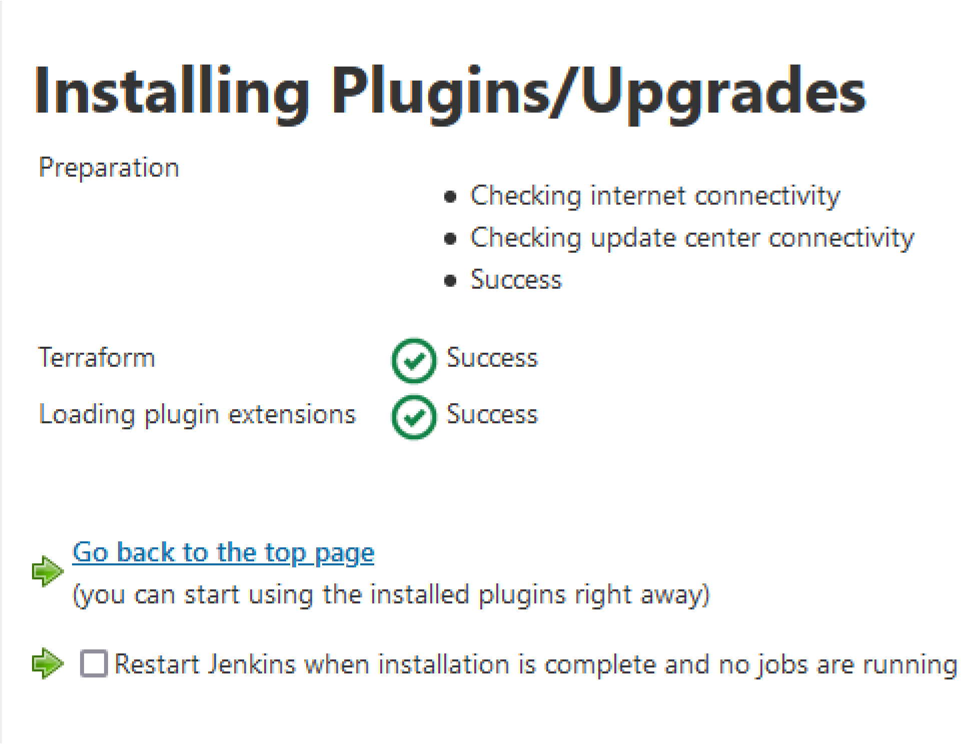


Install ↑	Name	Version	Released
☐	Terraform Build Wrappers This plugin provides a build wrapper for Terraform to launch and destroy infrastructure.	1.0.10	1 yr 4 mo ago

Update information obtained: 11 min ago

d. Select **Terraform** and then **Gitlab** and click **Install without restart**.

e. Restart Jenkins by running <Jenkinsurl:8080>, when prompted to restart, click **YES**



Installing Plugins/Upgrades

Preparation
- Checking internet connectivity
- Checking update center connectivity
- Success

Terraform ✓ Success

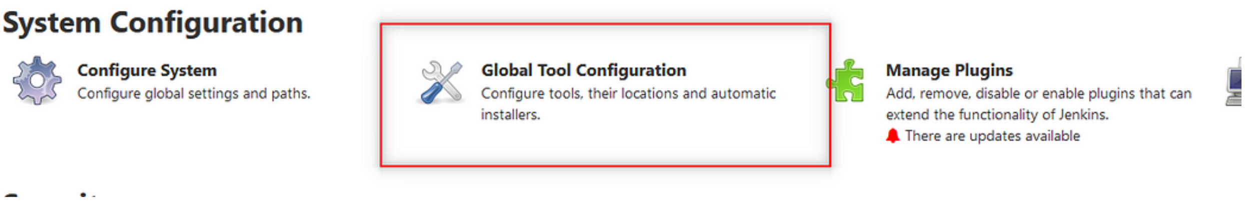
Loading plugin extensions ✓ Success

Go back to the top page
(you can start using the installed plugins right away)

☐ Restart Jenkins when installation is complete and no jobs are running

f. Click **Manage Jenkins** from left hand navigation.

g. Click **Tool** from **System configuration** section



System Configuration

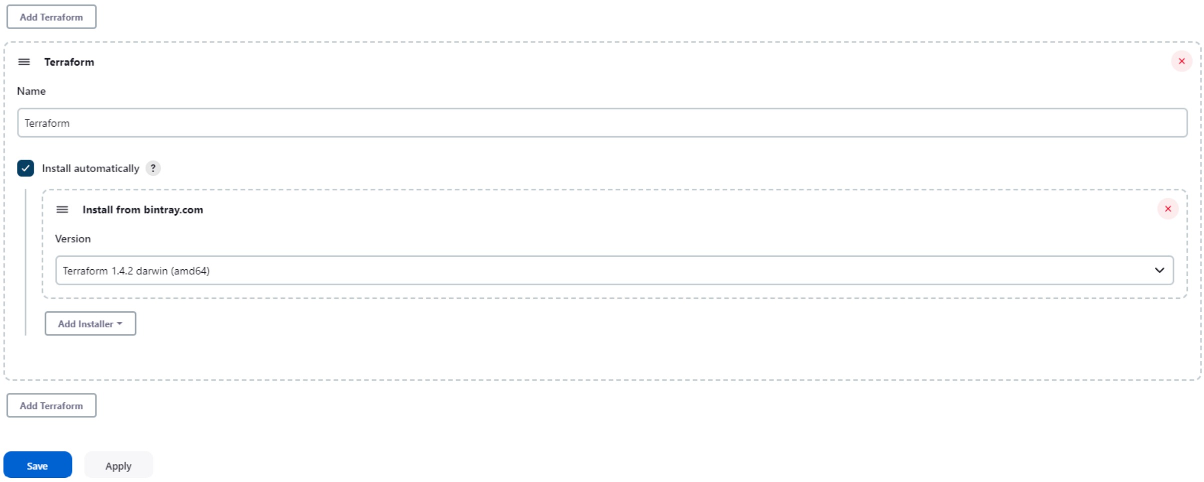
Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
There are updates available

h. Scroll down to the **Terraform** section and click **Add Terraform**.

i. Enter a **Name** of your choice. I'm going to use "Terraform" to make things simple. Ensure **install automatically** is checked. Save and apply.



Add Terraform

Terraform

Name

Terraform

☑ Install automatically ?

Install from bintray.com

Version

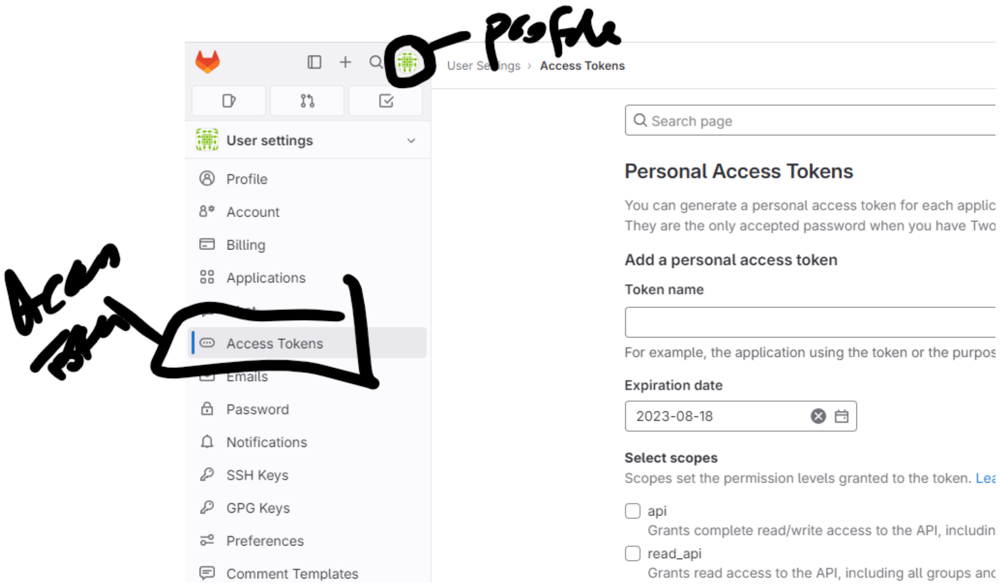
Terraform 1.4.2 darwin (amd64)

Add Installer ▾

Add Terraform

Save Apply

j. Create Gitlab token >> Navigate to Gitlab >> Click on profile logo >> the Access toke >>fill name, expiration>>select a role (guest)>> select al scopes>>create project access token>>Copy token and save in secure location.



User Settings › Access Tokens

Search page

User settings

Profile
Account
Billing
Applications
Access Tokens
Emails
Password
Notifications
SSH Keys
GPG Keys
Preferences
Comment Templates

Personal Access Tokens

You can generate a personal access token for each applic
They are the only accepted password when you have Two

Add a personal access token

Token name

For example, the application using the token or the purpos

Expiration date

2023-08-18

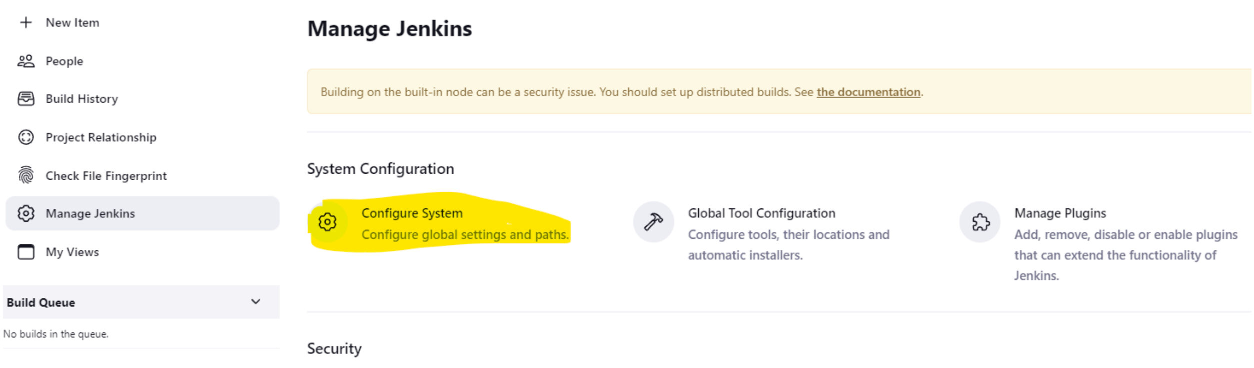
Select scopes

Scopes set the permission levels granted to the token. Lea

□ api
Grants complete read/write access to the API, includin

□ read_api
Grants read access to the API, including all groups anc

k. For gitlab configuration, Navigate to **Manage Jenkins** >> then **Systems** and lookout for **Gitlab >>** provide a name for the connection**,** for **Gitlab host URL** use https://gitlab.com>> for **Credentials**, click on **Add** and then **Jenkins**. For **kind** select **Gitlab API Token** >> provide the gitlabtoken >> for **ID** enter any name **e.g gitlab-creds** >> add the credential by clicking on the **add** section >> **Select credential from none to credential created** >> click on **test connection** and ensure to have a **success message** >> Once complete, **save** and then **apply**



No builds in the queue.

Manage Jenkins

Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.

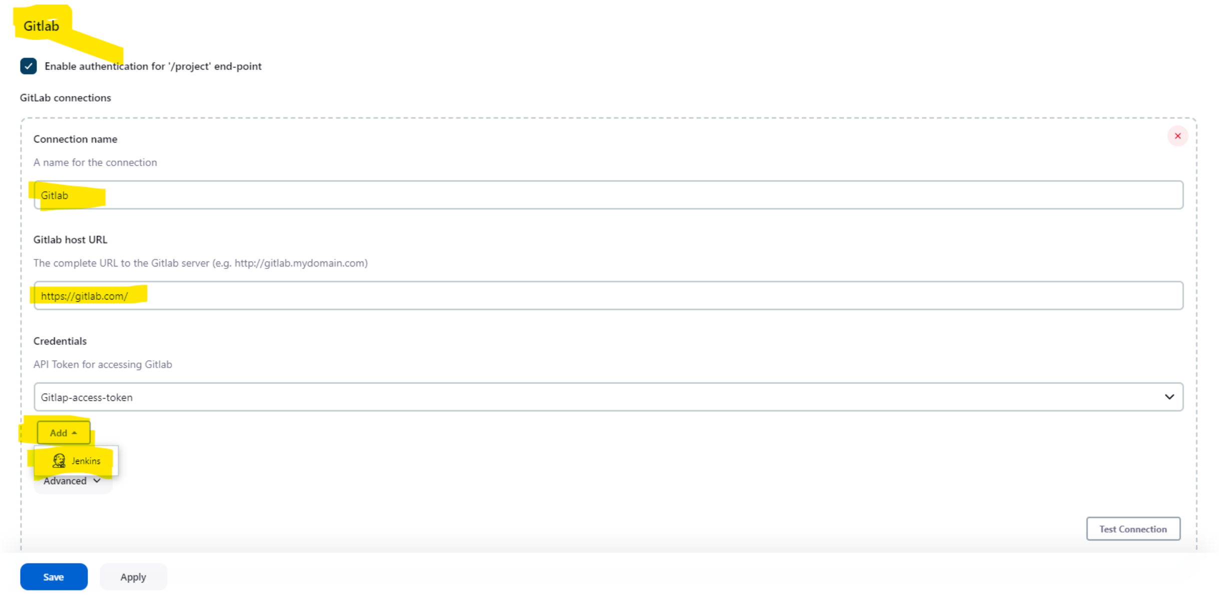
System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Security


Gitlab

☑ Enable authentication for '/project' end-point

GitLab connections

Connection name
A name for the connection
Gitlab

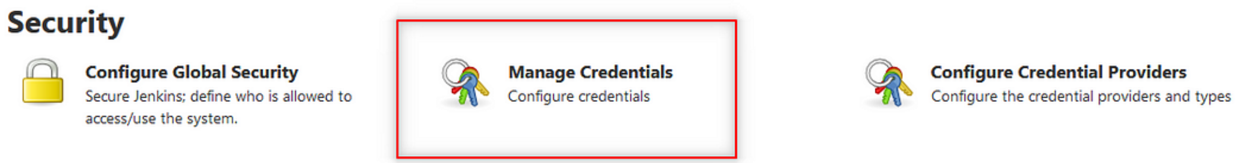
Gitlab host URL
The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)
https://gitlab.com/

Credentials
API Token for accessing Gitlab
Gitlap-access-token

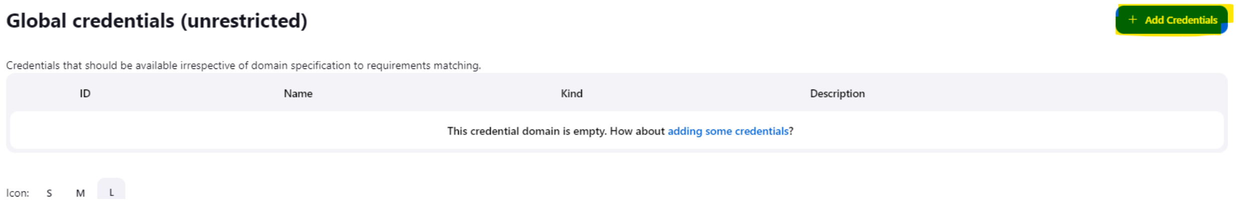
3. Manage Credentials on Jenkins (Only add these AWS creds if you have not attached an IAM role with admin access to the Jenkins server)

a. Click **Manage Jenkins**.

b. Click **Manage Credentials** in the **Security** section.


Security
Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.
Manage Credentials
Configure credentials
Configure Credential Providers
Configure the credential providers and types

c. Click on **systems** >> Global credentials (unrestricted) >> and add credentials on the top right corner

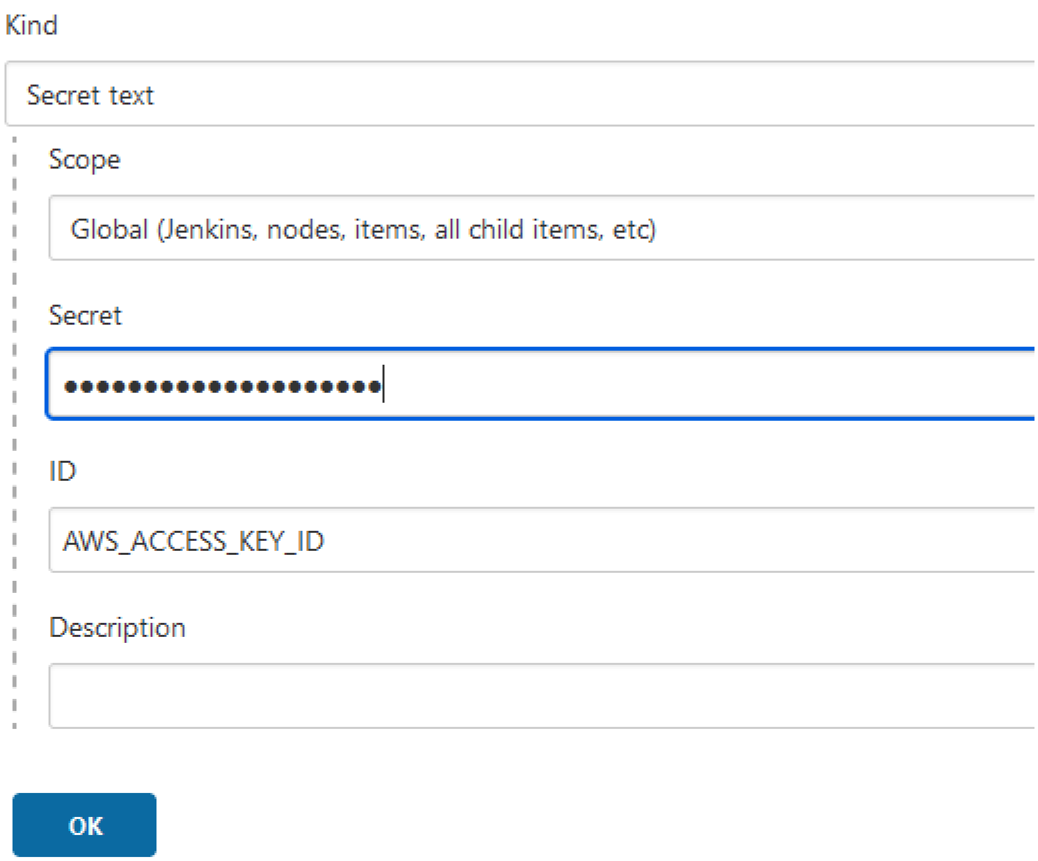


Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
This credential domain is empty. How about adding some credentials?			

d. For **Kind** select **Secret text**. For **ID** type "**AWS_ACCESS_KEY_ID**". For **Secret** paste your **Access Key** for your user. Then click **OK**



Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

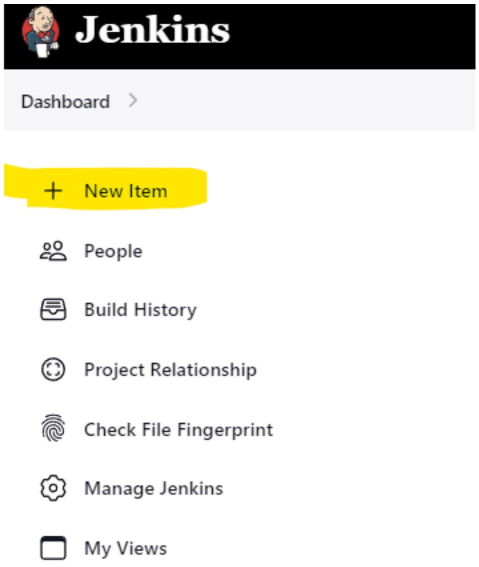
Secret: •••••••••••••••••••

ID: AWS_ACCESS_KEY_ID

Description:

e. Repeat previous step for your “**AWS_SECRET_ACCESS_KEY**”

4. Create pipeline job
    a. Navigate back to the Jenkins dashboard and click on **New Item**



Jenkins

Dashboard >

+ New Item

People

Build History

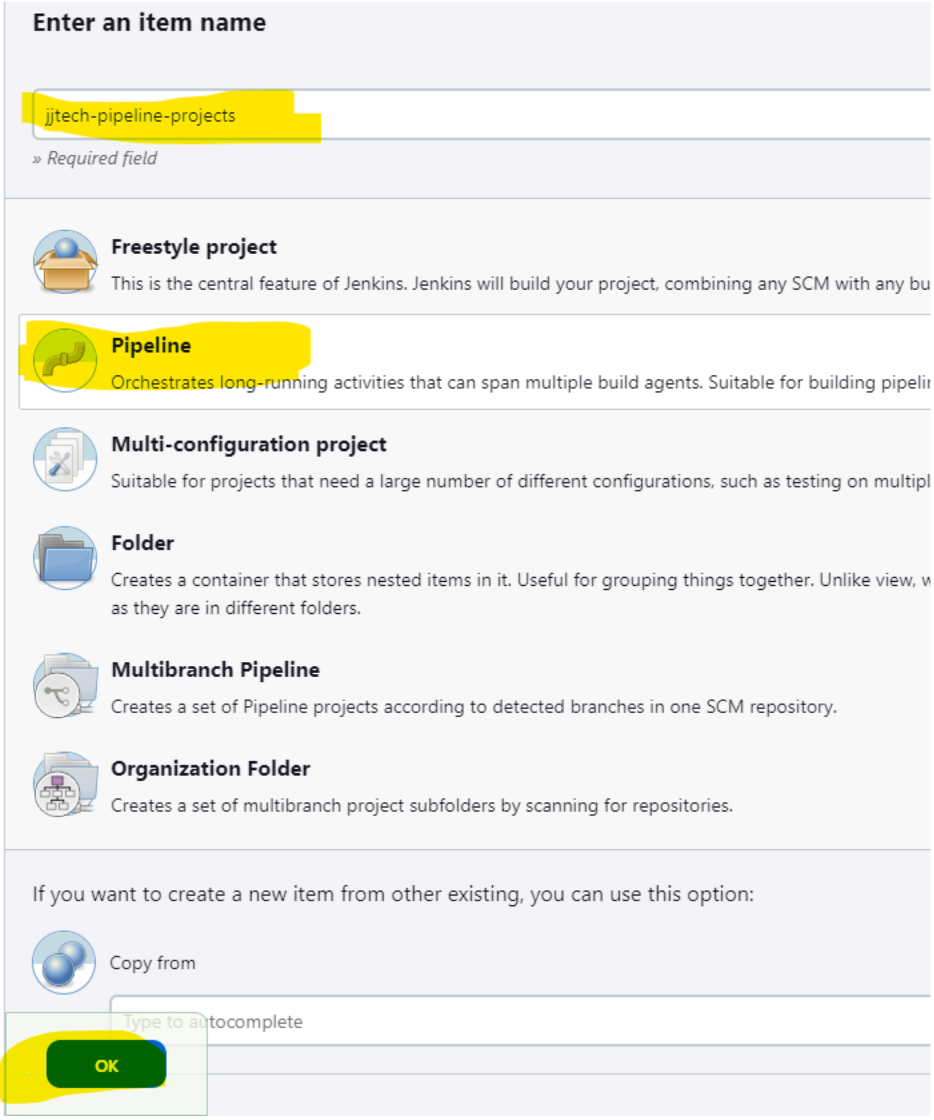
Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

b. Enter an item name (name of the pipeline project you want to create) >> choose **Pipeline** in the job fields and click **OK.**



Enter an item name

jjtech-pipeline-projects

» Required field

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any bu

Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipeli

Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multipl

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, w as they are in different folders.

Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder

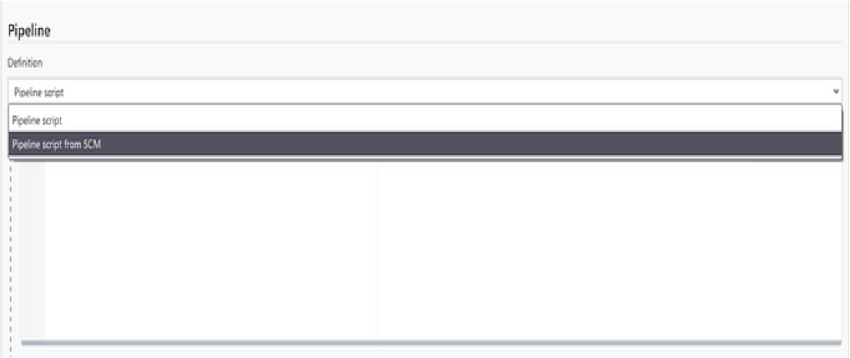
Creates a set of multibranch project subfolders by scanning for repositories.

If you want to create a new item from other existing, you can use this option:

Copy from

Type to autocomplete

c. Navigate down to **Pipeline and** click on **Pipeline script** and select **Pipeline script from SCM**



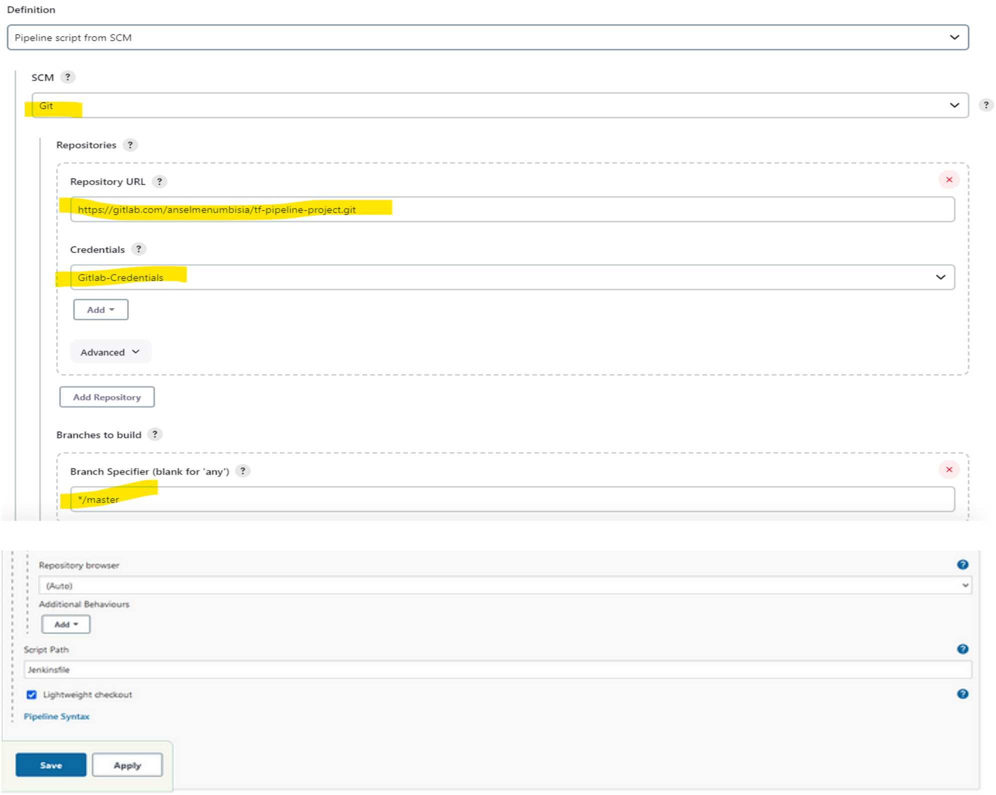
Pipeline

Definition

Pipeline script

d. Enter the following

- **SCM:** Git
- **Repository URL:** Your gitlab repo where you have the jenkinsfile (same as link from clone in gitlab)
- **Branch:** your primary branch e.g master
- **credentials:** select your gitlab credential created in the previous step. This is not required if your repo is public
- **Repository browser:** Auto
- **Script Path:** “jenkinsfile”
- click **save**



Definition

Pipeline script from SCM

SCM

Git

Repositories

Repository URL

https://gitlab.com/anselmenumbisia/tf-pipeline-project.git

Credentials

Gitlab-Credentials

Branches to build

Branch Specifier (blank for 'any')

*/master

Repository browser

(Auto)

Additional Behaviours

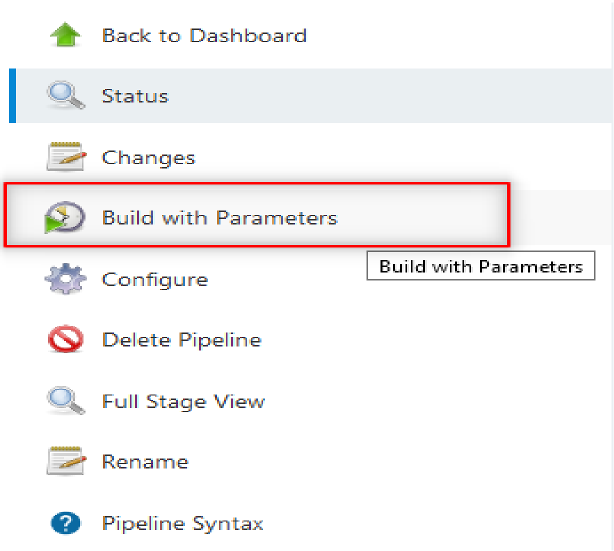
Script Path

Jenkinsfile

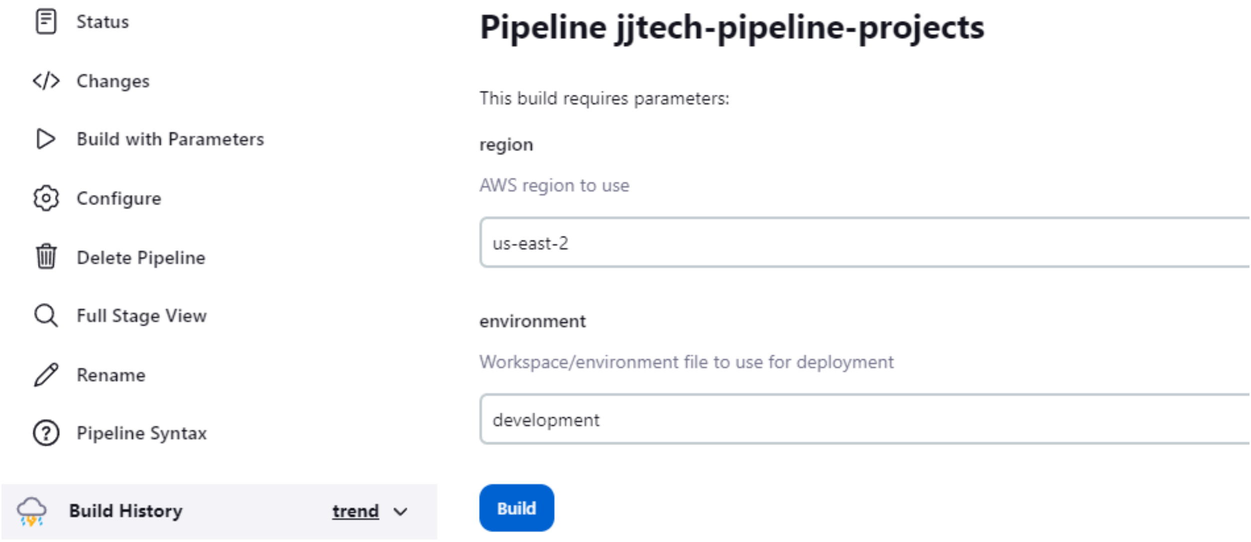
☑ Lightweight checkout

5. Run Jenkins Pipeline

a. Select **Build with Parameters** from the left navigation.



b. For the **environment** parameter type the name you want to use for your Workspace. The default is “**development**”. For the **region**, fill in the name of the region where you want to deploy the resources. Click on **Build**



Pipeline jjtech-pipeline-projects

This build requires parameters:

region

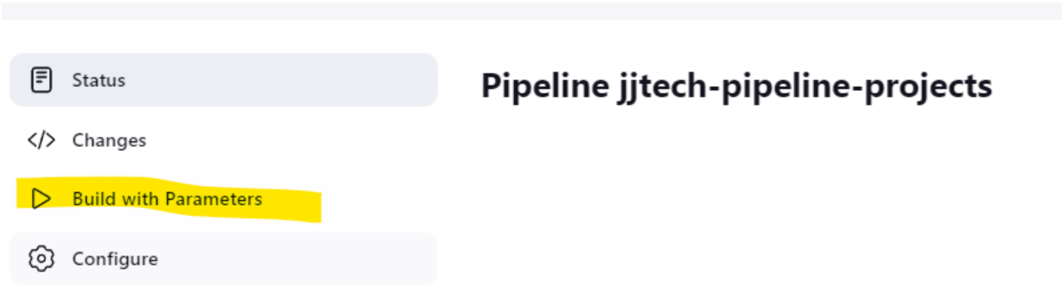
AWS region to use

us-east-2

environment

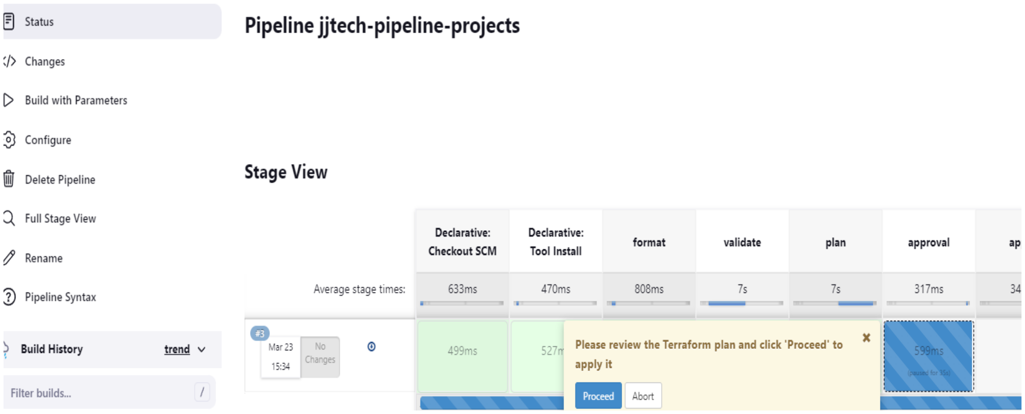
Workspace/environment file to use for deployment

development



Pipeline jjtech-pipeline-projects

c. Now you should see the steps of the pipeline begin and the time it takes to complete each stage. The pipeline will pause on after the **Plan** step and prompt for a manual approval to proceed. Click on proceed to continue the pipeline.



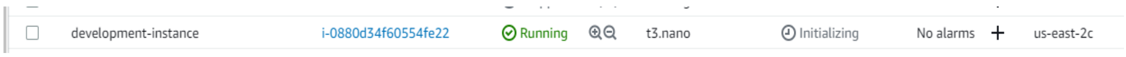
Pipeline jjtech-pipeline-projects

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	format	validate	plan	approval	ap
Average stage times:	633ms	470ms	808ms	7s	7s	317ms	34
#3 Mar 23 15:34 No Changes	499ms	527m				599ms	

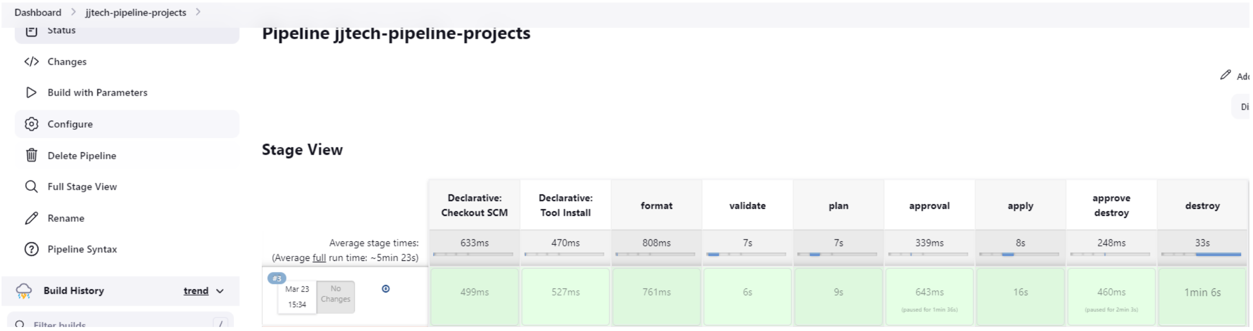
Please review the Terraform plan and click 'Proceed' to apply it

d. Pipeline apply phase is now complete . Navigate to aws console to confirm resource creation



□	development-instance	i-0880d34f60554fe22	✓ Running		t3.nano	Initializing	No alarms +	us-east-2c

e. Proceed to validate the pipeline to destroy the resources provisioned



Pipeline jjtech-pipeline-projects

Stage View

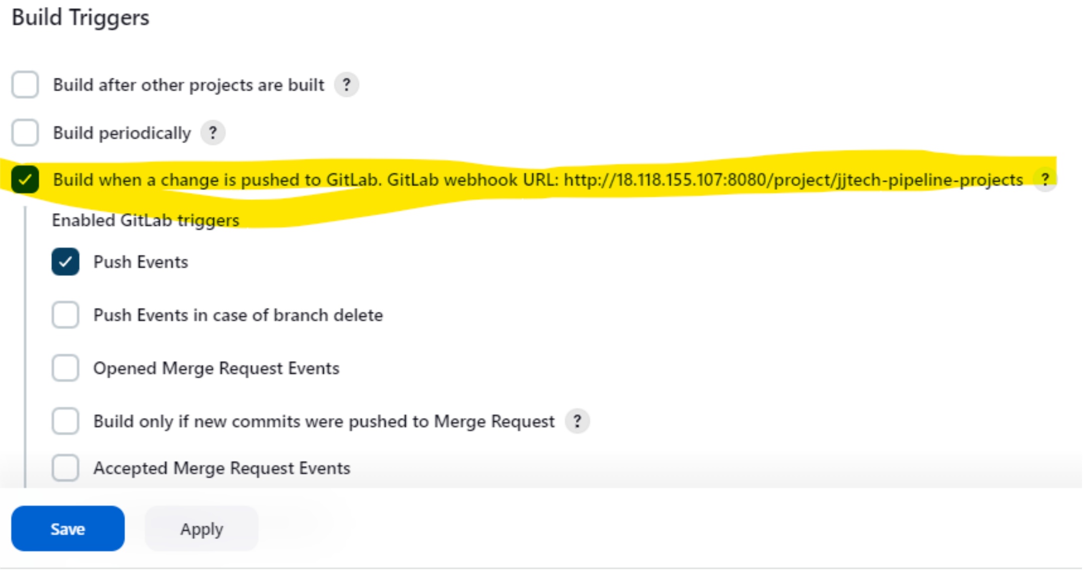
	Declarative: Checkout SCM	Declarative: Tool Install	format	validate	plan	approval	apply	approve destroy	destroy
Average stage times: (Average full run time: ~5min 23s)	633ms	470ms	808ms	7s	7s	339ms	8s	248ms	33s
#3 Mar 23 15:34 No Changes	499ms	527ms	761ms	6s	9s	643ms (paused for 1min 36s)	16s	460ms (paused for 2min 3s)	1min 6s

f. Verify that the resources have been destroyed.

6. Trigger build based on push event on the gitlab repo. Integration tutorial here

a. In Jenkins

- **Step 1:** Go the the “Dashboard >> **Click on the Jenkins project >> click on configure**” of your Jenkins project.
- **Step 2:** Go to the “***Build Triggers***” section.
- **Step 3:** Under the “***Build when a change is pushed to Gitlab***” checkbox, click the “***advanced***” button.



Build Triggers

□ Build after other projects are built ?

□ Build periodically ?

☑ Build when a change is pushed to GitLab. GitLab webhook URL: http://18.118.155.107:8080/project/jjtech-pipeline-projects ?

Enabled GitLab triggers

☑ Push Events

□ Push Events in case of branch delete

□ Opened Merge Request Events

□ Build only if new commits were pushed to Merge Request ?

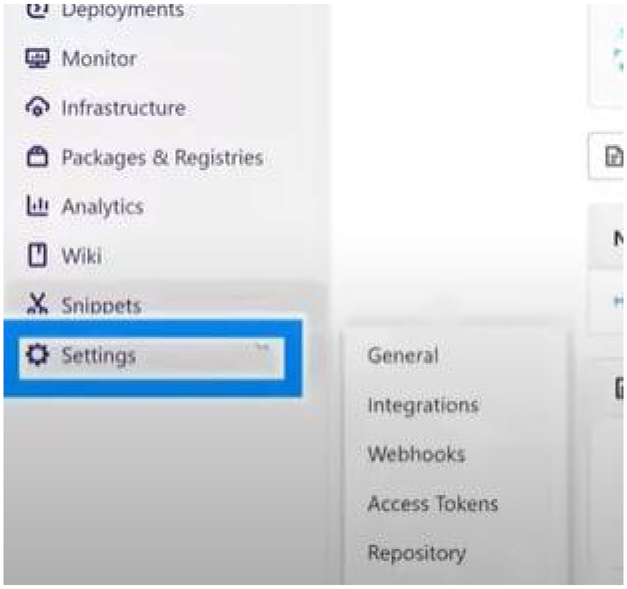
□ Accepted Merge Request Events

- **Step 4:** Scroll down and Click the “***Generate***” button under the “***Secret Token***” field.

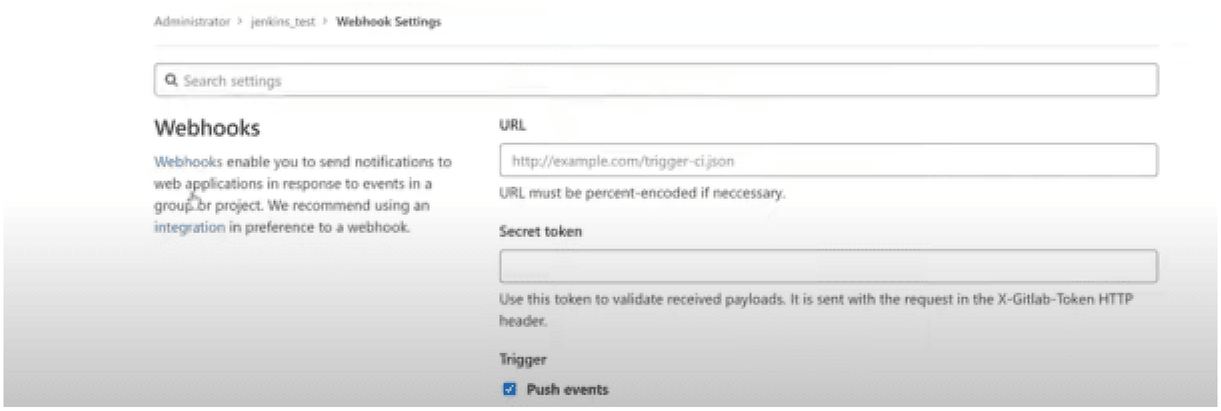
- **Step 5:** Copy the resulting token, and **save** the job configuration.

b. Webhook Integration

- **Step 1: Navigate to gitlab.** In the left navigation pane, select the "***Settings***" option. Then click on the "***Webhooks***" option.



- **Step 2:** Now, in the Integration settings window, under the "***Integrations***" section, select the "***Webhook***" hyperlink.
- **Step 3:** In the "***Webhook Settings***" window, under the "***Webhooks***" section, paste the webhook URL (such as ***https://JENKINS_URL/project/YOUR_JOB***) which you have copied in Jenkins server.



Administrator > jenkins_test > Webhook Settings

Search settings

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

http://example.com/trigger-ci.json

URL must be percent-encoded if neccessary.

Secret token

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☑ Push events

- **Step 4:** Paste the secret token which you have generated in the Jenkins server and check the **Push events** and click **Add webhook** at the bottom.
- **Step 5:** Scroll down to the webhook created and click on **Test** connection. Select **push events** and you should see **Hook executed successfully: HTTP 200**

c. Make changes to your source code, then push to the gitlab repo.

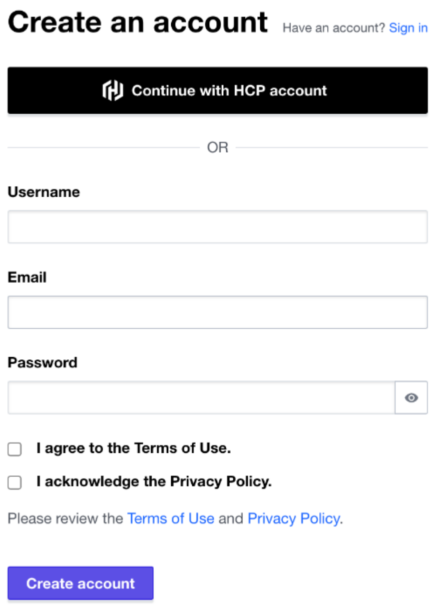
d. Navigate back to Jenkins and notice that a pipeline job is triggered.

E. Terraform Cloud

CLI-Driven Workflow

1. Create Terraform Cloud account

- Visit https://app.terraform.io/signup/account and follow the prompts to create a free Terraform Cloud account.



Create an account Have an account? Sign in

Continue with HCP account

OR

Username

Email

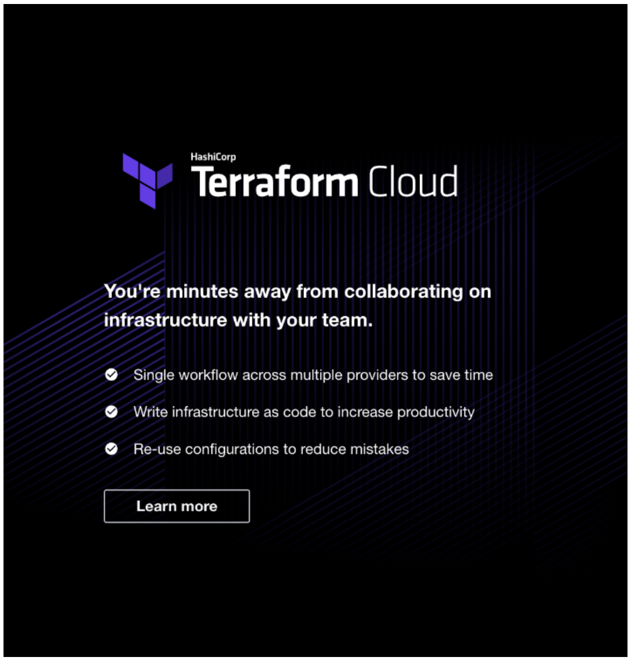
Password

□ I agree to the Terms of Use.

□ I acknowledge the Privacy Policy.

Please review the Terms of Use and Privacy Policy.

Create account



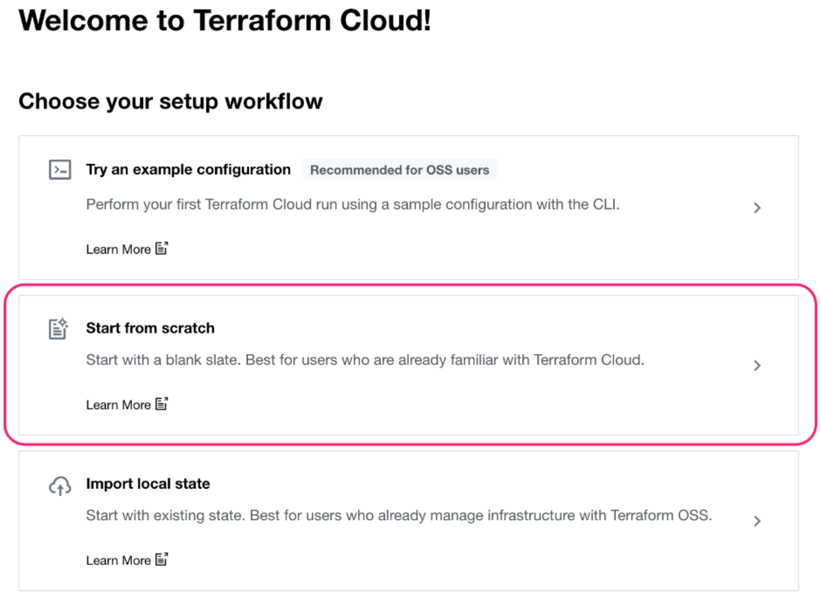
HashiCorp Terraform Cloud

You're minutes away from collaborating on infrastructure with your team.

- Single workflow across multiple providers to save time
- Write infrastructure as code to increase productivity
- Re-use configurations to reduce mistakes

Learn more

- **step2**. When you sign up, you will receive an email asking you to confirm your email address. Confirm your email address before moving on. When you click the link to confirm your email address, the Terraform Cloud UI will ask which setup workflow you would like use. Select **Start from scratch**.



Welcome to Terraform Cloud!

Choose your setup workflow

Try an example configuration — Recommended for OSS users

Perform your first Terraform Cloud run using a sample configuration with the CLI.

Learn More

Start from scratch

Start with a blank slate. Best for users who are already familiar with Terraform Cloud.

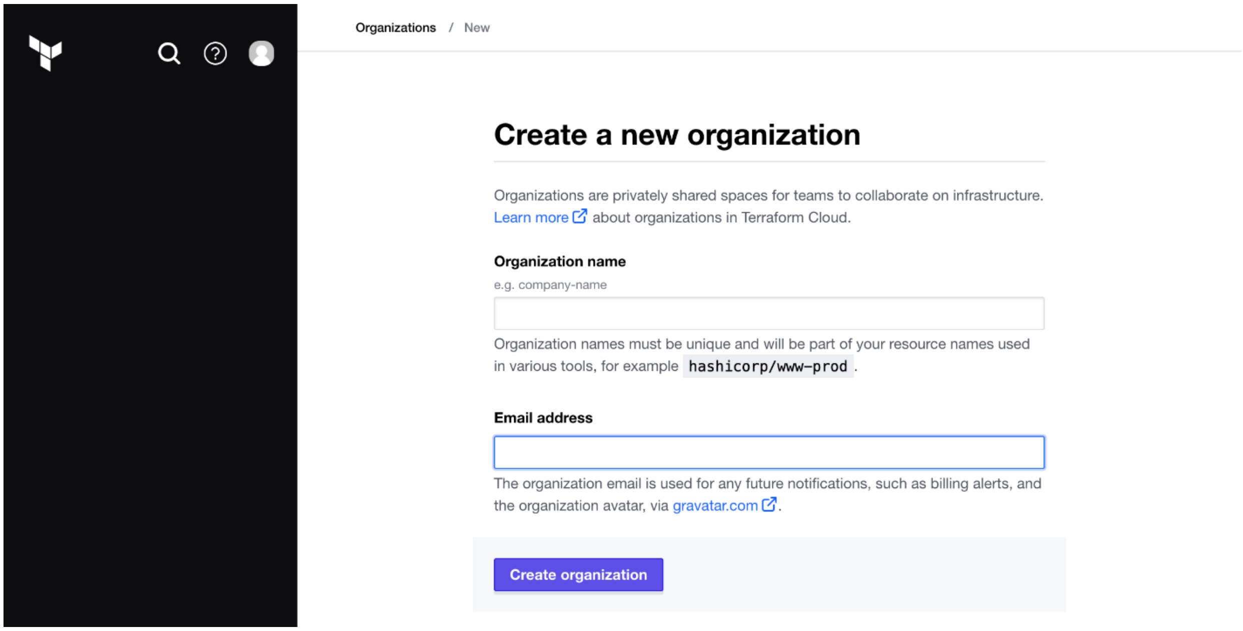
Learn More

Import local state

Start with existing state. Best for users who already manage infrastructure with Terraform OSS.

Learn More

Step3. create a new organization. Creating organizations of up to 5 users is free, and the members you add to the organization will be able to collaborate on your workspaces and share private modules and providers.



Organizations / New

Create a new organization

Organizations are privately shared spaces for teams to collaborate on infrastructure. Learn more about organizations in Terraform Cloud.

Organization name

e.g. company-name

Organization names must be unique and will be part of your resource names used in various tools, for example `hashicorp/www-prod`.

Email address

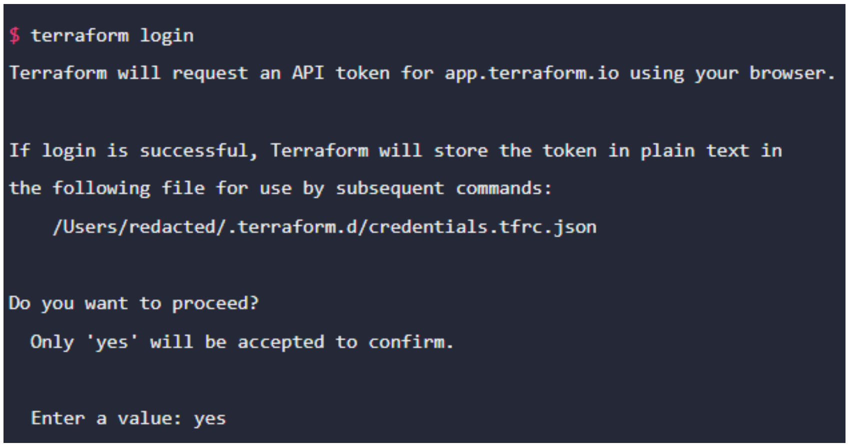
The organization email is used for any future notifications, such as billing alerts, and the organization avatar, via gravatar.com.

2. Log in to Terraform Cloud from the CLI

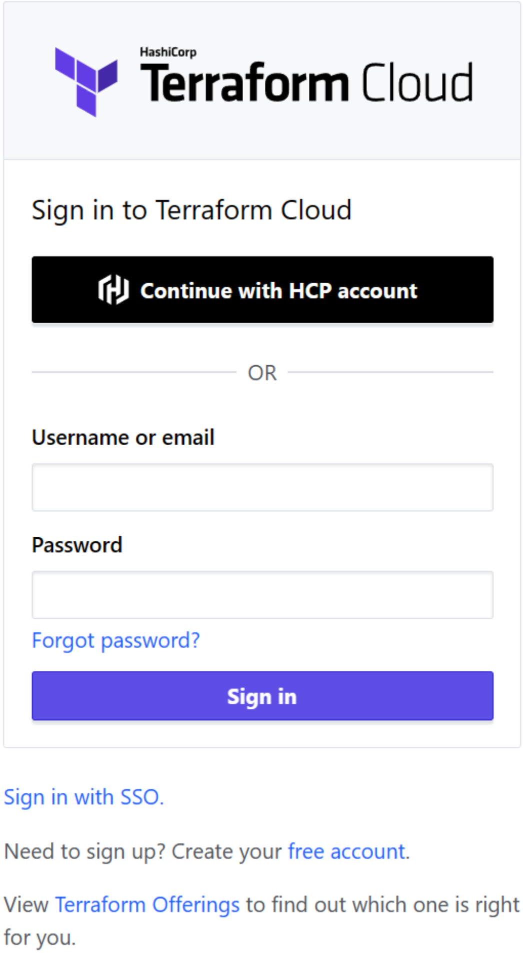
Terraform Cloud runs Terraform operations and stores state remotely, so you can use Terraform without worrying about the stability of your local machine, or the security of your state file.

To use Terraform Cloud from the command line, you must log in. Logging in allows you to trigger remote plans and runs, migrate state to the cloud, and perform other remote operations on configurations with Terraform Cloud

Step1. run the **terraform login** subcommand. Respond **yes** to the prompt to confirm that you want to authenticate. A browser window will automatically open to the Terraform Cloud login screen. Enter a token name in the web UI, or leave the default name, terraform login.



```
$ terraform login
Terraform will request an API token for app.terraform.io using your browser.

If login is successful, Terraform will store the token in plain text in
the following file for use by subsequent commands:
    /Users/redacted/.terraform.d/credentials.tfrc.json

Do you want to proceed?
  Only 'yes' will be accepted to confirm.

  Enter a value: yes
```



HashiCorp Terraform Cloud

Sign in to Terraform Cloud

Username or email

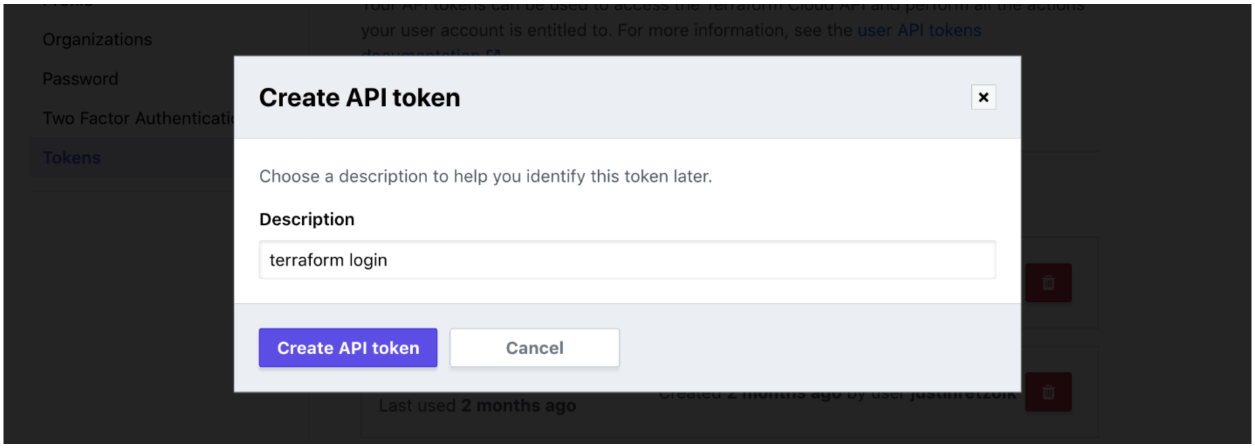
Password

Sign in with SSO.

Need to sign up? Create your free account.

View Terraform Offerings to find out which one is right for you.

Step2. Click **Create API token** to generate the authentication token. Save a copy of the token in a secure location. It provides access to your Terraform Cloud organization. Terraform will also store your token locally at the file path specified in the command output



your user account is entitled to. For more information, see the user API tokens

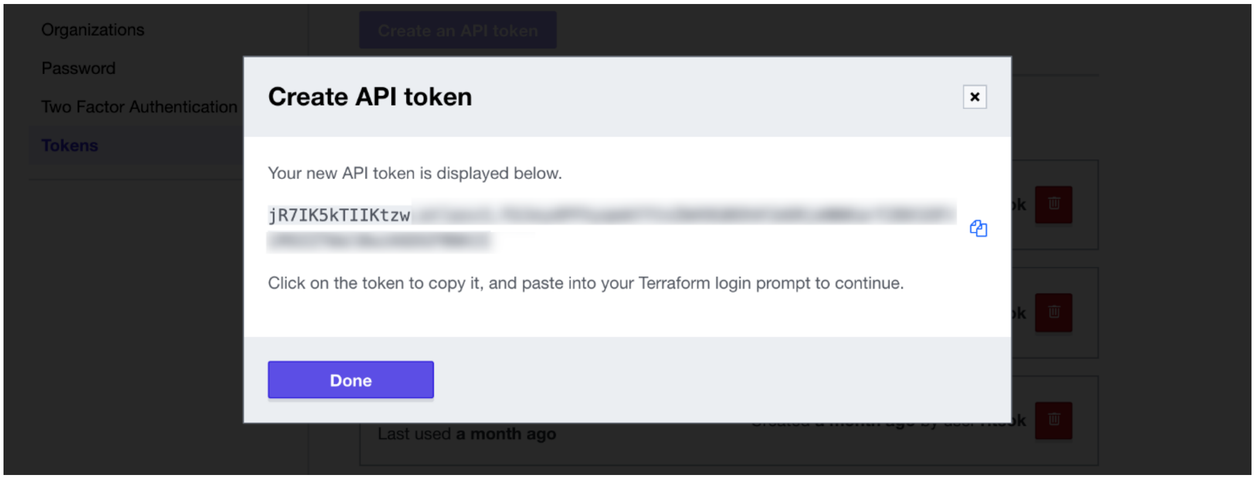
Create API token

Choose a description to help you identify this token later.

Description

terraform login

Last used **2 months ago**



Create API token

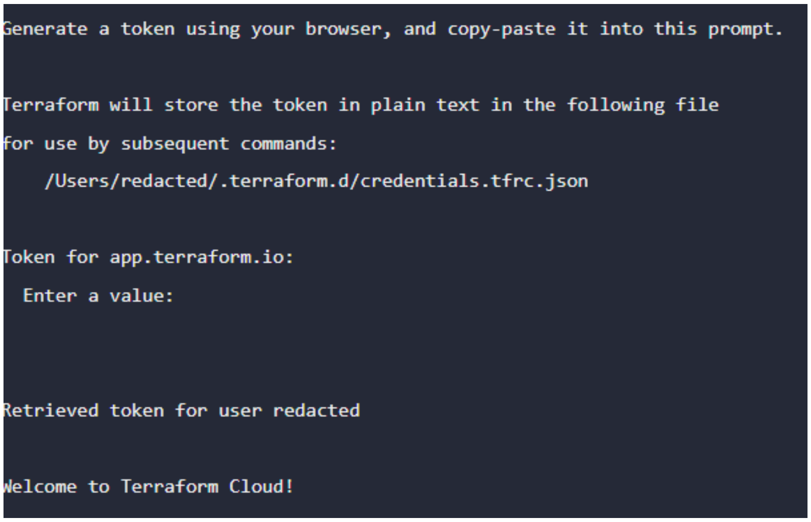
Your new API token is displayed below.

jR7IK5kTIIKtzw[redacted]

Click on the token to copy it, and paste into your Terraform login prompt to continue.

Last used **a month ago**

Step3. When the Terraform CLI prompts you, paste the user token exactly once into your terminal. Terraform will hide the token for security when you paste it into your terminal. Press **Enter** to complete the authentication process.

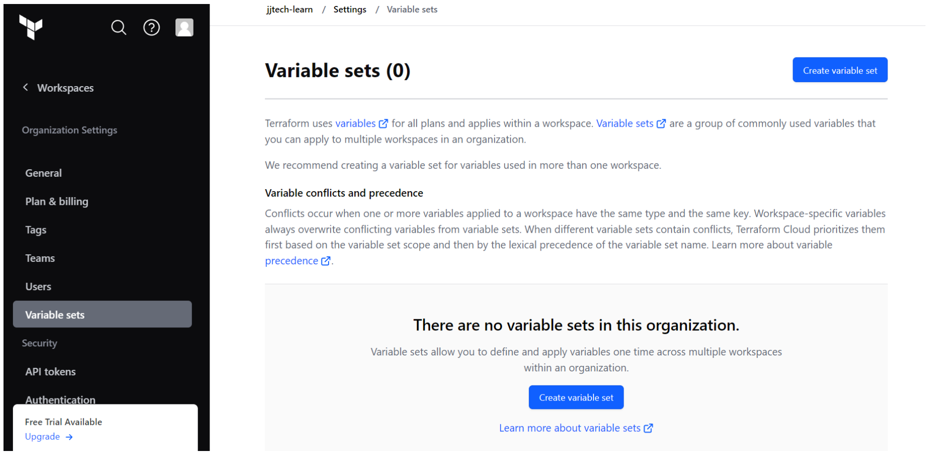


```
Generate a token using your browser, and copy-paste it into this prompt.

Terraform will store the token in plain text in the following file
for use by subsequent commands:
    /Users/redacted/.terraform.d/credentials.tfrc.json

Token for app.terraform.io:
  Enter a value:


Retrieved token for user redacted

Welcome to Terraform Cloud!
```

3. Creating variable sets

Variable sets allow you to avoid redefining the same variables across workspaces, so you can standardize common configurations throughout your organization. One common use case for variable sets is for provider credentials. By defining a variable set for your credentials, you can easily reuse the same variables across multiple workspaces and efficiently and securely rotate your credentials. We will create a variable set for our AWS credentials.

Step1. Navigate to terraform cloud >> click on settings >> Variable sets



Variable sets (0)

Terraform uses variables for all plans and applies within a workspace. Variable sets are a group of commonly used variables that you can apply to multiple workspaces in an organization.

We recommend creating a variable set for variables used in more than one workspace.

Variable conflicts and precedence

Conflicts occur when one or more variables applied to a workspace have the same type and the same key. Workspace-specific variables always overwrite conflicting variables from variable sets. When different variable sets contain conflicts, Terraform Cloud prioritizes them first based on the variable set scope and then by the lexical precedence of the variable set name. Learn more about variable precedence.

There are no variable sets in this organization.

Variable sets allow you to define and apply variables one time across multiple workspaces within an organization.

Step2. Click **Create Variable sets**

- Name: (provide name)

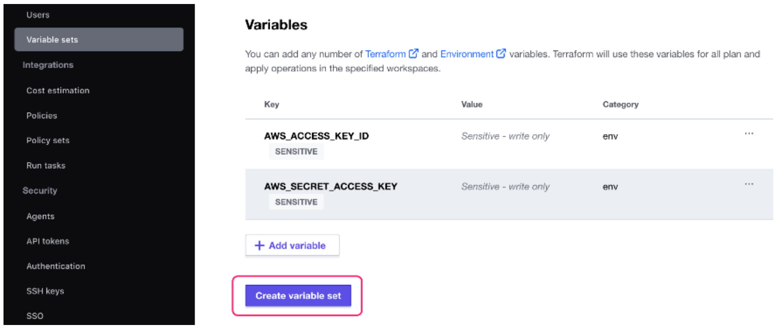
- Description:

- Workspaces: Apply to all workspaces in this organisation

Step3. Click +**Add Variable**. Select the **Environment variable** option. Set the key to AWS_ACCESS_KEY_ID and the value to your AWS **Access Key ID**. Mark it as **Sensitive** and click **Add variable**.

Step4. Click + **Add Variable** again. Define another environment variable named AWS_SECRET_ACCESS_KEY and set it to your AWS **Secret access key**. Mark it as **Sensitive** and click **Add variable**.

Step4. Click **create Variable sets**



Variables

You can add any number of Terraform and Environment variables. Terraform will use these variables for all plan and apply operations in the specified workspaces.

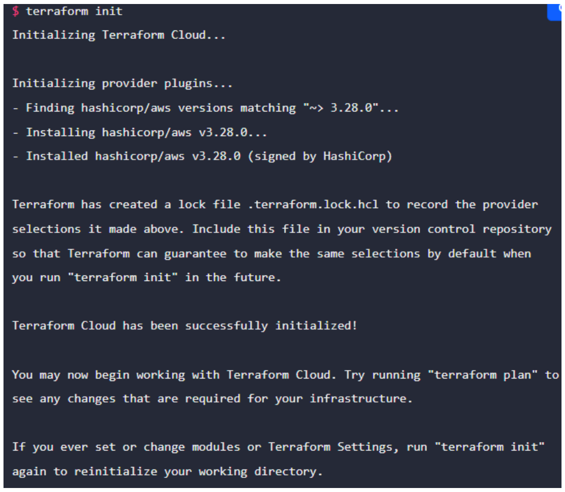
Key	Value	Category
AWS_ACCESS_KEY_ID SENSITIVE	*Sensitive - write only*	env
AWS_SECRET_ACCESS_KEY SENSITIVE	*Sensitive - write only*	env

4. Create Workspace

Step1. Navigate to vscode where you have your configuration file and configure **provider.tf** file to add configuration for **terraform cloud.** This cloud block specifies which Terraform Cloud organization and workspace to use for the operations in this working directory. When using the CLI-driven Terraform Cloud workflow, running terraform init on configuration with a cloud block creates the Terraform Cloud workspace specified in the block, if it does not already exist.

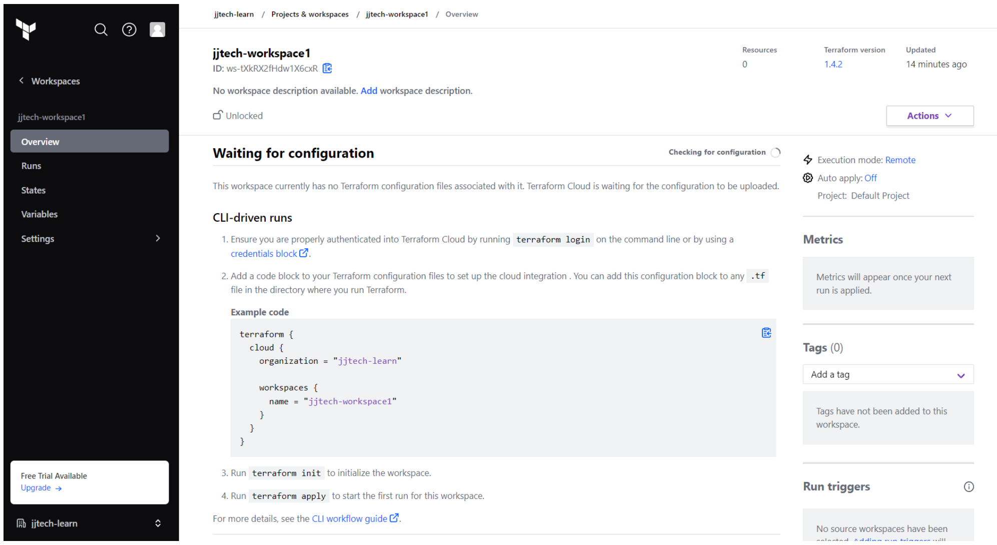
```
terraform {
 cloud {
   organization = "Provide name of TFC Organisation"
   workspaces {
     name = "pass name of a workspace to plan to create"
   }
 }
 required_version = ">= 1.1.0"
 required_providers {
   aws = {
     source  = "hashicorp/aws"
     version = "4.55.0"
   }

 }
}
```

Step2. Run **terraform init**



```
$ terraform init
Initializing Terraform Cloud...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.28.0"...
- Installing hashicorp/aws v3.28.0...
- Installed hashicorp/aws v3.28.0 (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 Cloud has been successfully initialized!

You may now begin working with Terraform Cloud. Try running "terraform plan" to
see any changes that are required for your infrastructure.

If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.
```

As part of the initialization process, Terraform created the new **jjtech-workspace1** workspace in our Terraform Cloud organization, configured for CLI-driven runs.



jjtech-workspace1

ID: ws-tXkRX2fHdw1X6cxR

Resources	Terraform version	Updated
0	1.4.2	14 minutes ago

No workspace description available. Add workspace description.

Unlocked

Waiting for configuration

Checking for configuration

This workspace currently has no Terraform configuration files associated with it. Terraform Cloud is waiting for the configuration to be uploaded.

CLI-driven runs

1. Ensure you are properly authenticated into Terraform Cloud by running `terraform login` on the command line or by using a credentials block.
2. Add a code block to your Terraform configuration files to set up the cloud integration . You can add this configuration block to any `.tf` file in the directory where you run Terraform.

Example code

```
terraform {
  cloud {
    organization = "jjtech-learn"

    workspaces {
      name = "jjtech-workspace1"
    }
  }
}
```

3. Run `terraform init` to initialize the workspace.
4. Run `terraform apply` to start the first run for this workspace.

For more details, see the CLI workflow guide.

Execution mode: Remote

Auto apply: Off

Project: Default Project

Metrics

Metrics will appear once your next run is applied.

Tags (0)

Tags have not been added to this workspace.

Run triggers

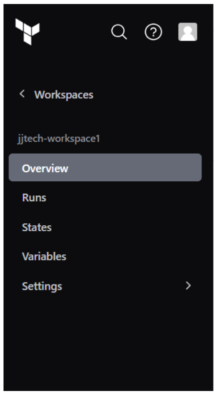
No source workspaces have been

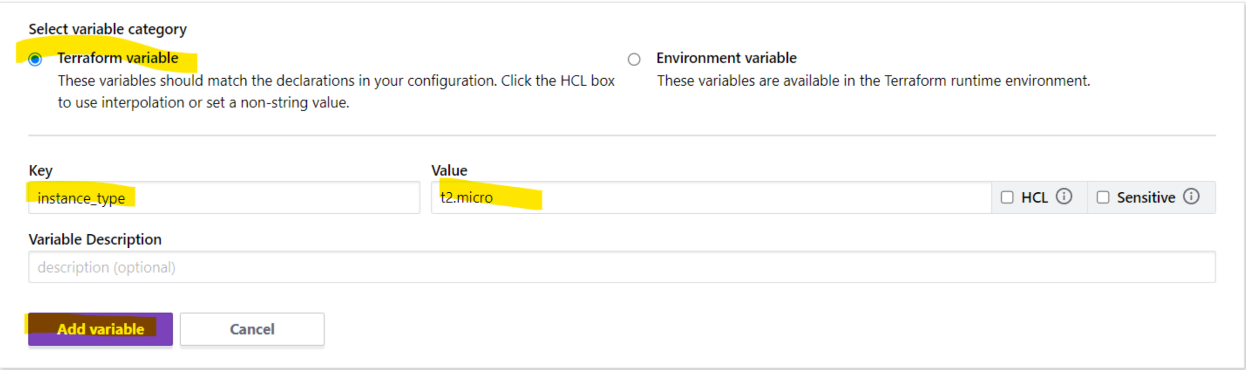
5. Create Infrastructure

You now have a Terraform Cloud workspace configured to use AWS credentials defined in a Terraform Cloud variable set. You can further configure your workspace using workspace-specific variables.

Step1. Navigate to your TFC and create a variable for your instance type and other variables as passed in your cli configuration file.

- click variables >> Under **Worskspace Variables**, click **Add variable** >> select **Terraform variable** and add **Key and Value** for the variable and **Add variable**.




Select variable category

● Terraform variable
These variables should match the declarations in your configuration. Click the HCL box to use interpolation or set a non-string value.

○ Environment variable
These variables are available in the Terraform runtime environment.

Key: instance_type

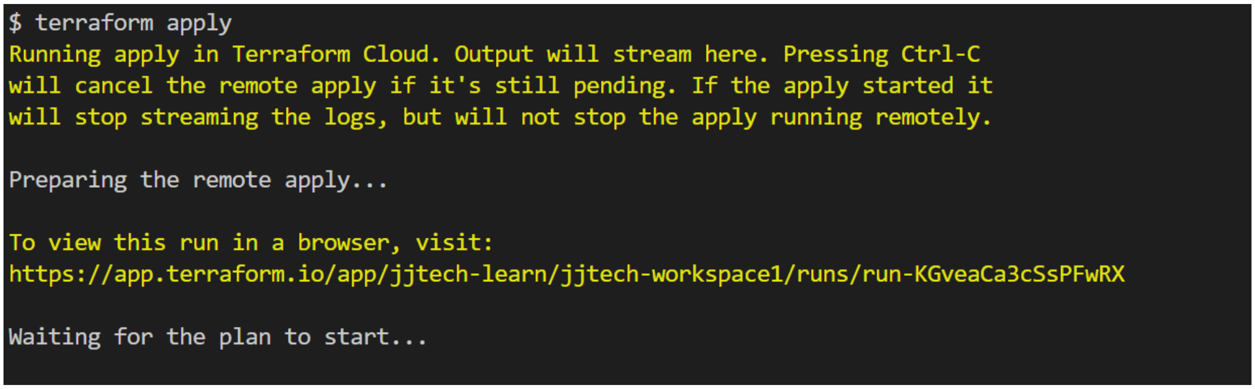
Value: t2.micro

□ HCL

□ Sensitive

Variable Description: description (optional)

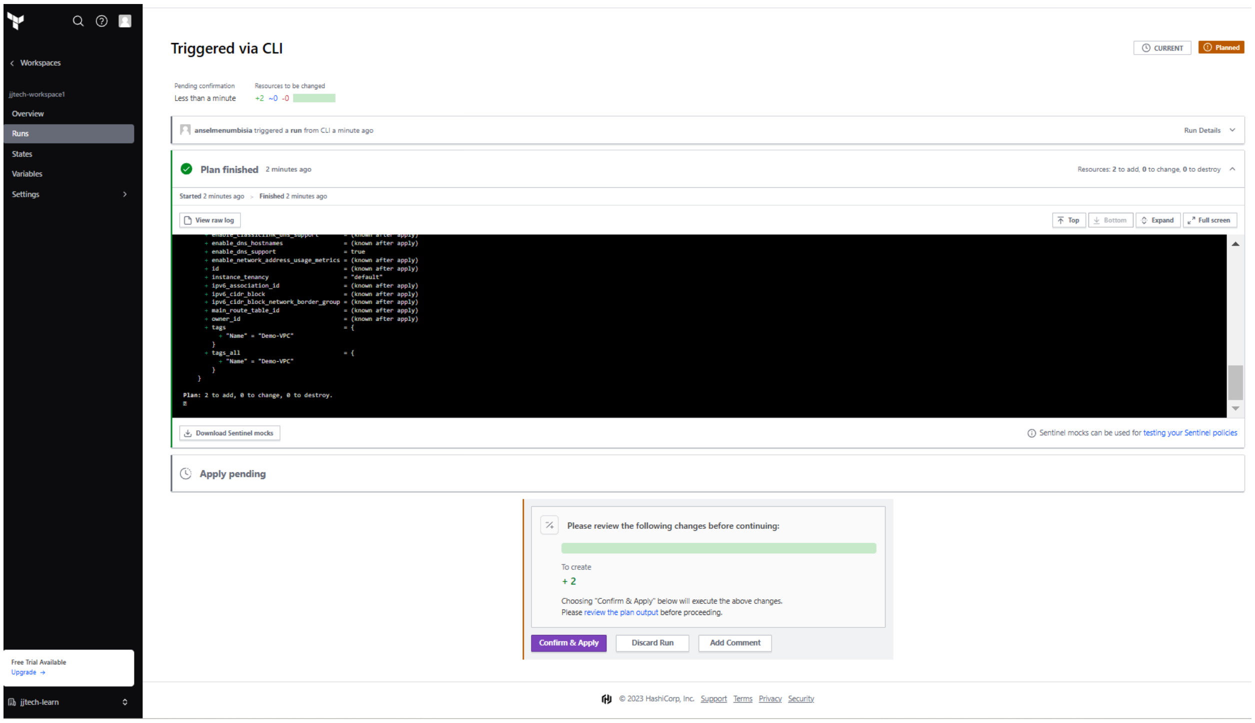
Step2. Navigate back to visual and run terraform apply to provision the infrastructure. Terraform will apply the run on TFC.



```
$ terraform apply
Running apply in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will cancel the remote apply if it's still pending. If the apply started it
will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/jjtech-learn/jjtech-workspace1/runs/run-KGveaCa3cSsPFwRX

Waiting for the plan to start...
```

Step3. Terraform will trigger your run in Terraform Cloud and stream the output to your terminal. Alternatively, you can follow and manage the run in the Terraform Cloud UI.

Navigate to the run URL that Terraform displays in your command output.



Triggered via CLI

Pending confirmation: Less than a minute

Resources to be changed: +2 ~0 -0

anselmenumbisia triggered a **run** from CLI a minute ago

Plan finished 2 minutes ago

Resources: **2** to add, **0** to change, **0** to destroy

Started 2 minutes ago > **Finished** 2 minutes ago

```
      + enable_classiclink_dns_support       = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + 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" = "Demo-VPC"
        }
      + tags_all                             = {
          + "Name" = "Demo-VPC"
        }
    }

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

Sentinel mocks can be used for testing your Sentinel policies

Apply pending

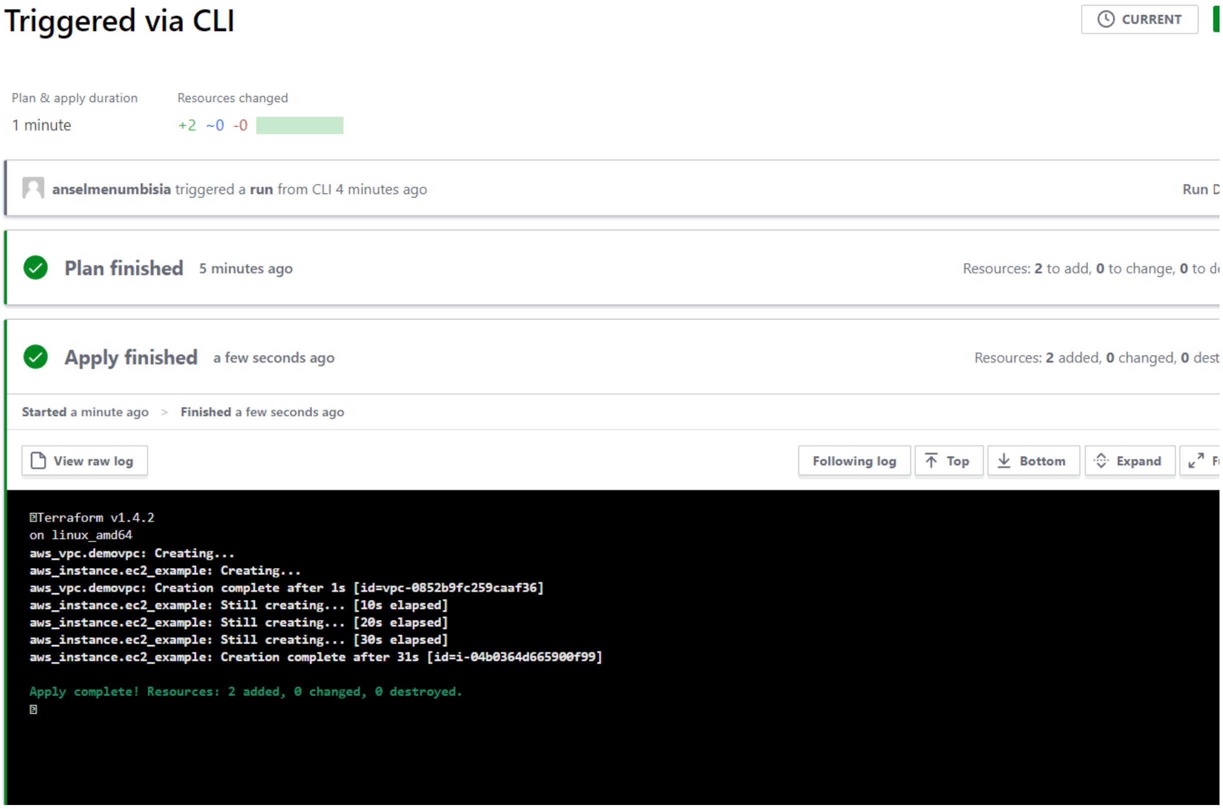
Please review the following changes before continuing:

To create

+ 2

Choosing "Confirm & Apply" below will execute the above changes.
Please review the plan output before proceeding.

Step4. Confirm and **apply** to trigger the apply from TFC.



Triggered via CLI

Plan & apply duration
1 minute

Resources changed
+2 ~0 -0

anselmenumbisia triggered a run from CLI 4 minutes ago

Plan finished 5 minutes ago

Resources: 2 to add, 0 to change, 0 to d

Apply finished a few seconds ago

Resources: 2 added, 0 changed, 0 dest

Started a minute ago > Finished a few seconds ago

```
Terraform v1.4.2
on linux_amd64
aws_vpc.demovpc: Creating...
aws_instance.ec2_example: Creating...
aws_vpc.demovpc: Creation complete after 1s [id=vpc-0852b9fc259caaf36]
aws_instance.ec2_example: Still creating... [10s elapsed]
aws_instance.ec2_example: Still creating... [20s elapsed]
aws_instance.ec2_example: Still creating... [30s elapsed]
aws_instance.ec2_example: Creation complete after 31s [id=i-04b0364d665900f99]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Step5. Navigate back to your cli and run **terraform destroy** and follow **from step 2** to destroy the resources created.

VCS driven Workflow

In addition to the CLI-driven workflow, Terraform Cloud offers a Version Control (VCS)-driven workflow that automatically triggers runs based on changes to your VCS repositories. The CLI-driven workflow allows you to quickly iterate on your configuration and work locally, while the VCS-driven workflow enables collaboration within teams by establishing your shared repositories as the source of truth for infrastructure configuration.

You will configure a VCS integration for your organization, connect your workspace to a VCS repository, and trigger a speculative plan based on a pull request. Then, you will merge the pull request to automatically apply changes to your infrastructure using Terraform Cloud.

1. **Create a new Gitlab Project**

- □ Navigate to Gitlab and create a new project >> copy the URL of the project
- □ Navigate to your vscode and switch to the directory where you have your terraform source code
- □ Run **git init** to initialise the directory
- □ Run **git remote add origin YOUR_REMOTE URL** copied from above
- □ Ensure you do not have a **terraform cloud** block in your terraform file (provider.tf), if yes, comment it out. When using the VCS-driven workflow for Terraform Cloud, you do not need to define the cloud block in your configuration.

```
terraform {
 # cloud {
 #   organization = "jjtech-learn"
 #   workspaces {
 #     name = "jjtech-workspace1"
 #   }
 # }
 # required_version = ">= 1.1.0"
 required_providers {
   aws = {
     source  = "hashicorp/aws"
     version = "4.55.0"
   }

 }
}
```

- □ Now add your changes with **git add .**

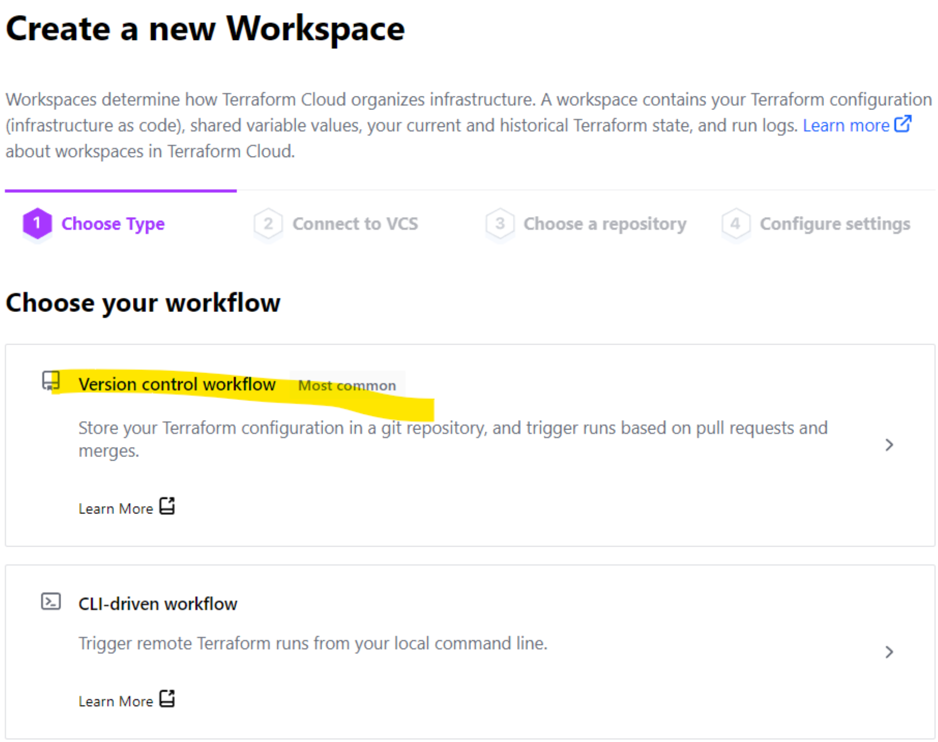
□ Commit changes with **git commit -m “message here”**

□ Push configuration to new project repo **git push origin master**

2. Enable VCS integration

□ Navigate to your terraform cloud account

□ Create a new workspace and select **Version Control Workflow**



Create a new Workspace

Workspaces determine how Terraform Cloud organizes infrastructure. A workspace contains your Terraform configuration (infrastructure as code), shared variable values, your current and historical Terraform state, and run logs. Learn more about workspaces in Terraform Cloud.

Choose your workflow

Version control workflow Most common

Store your Terraform configuration in a git repository, and trigger runs based on pull requests and merges.

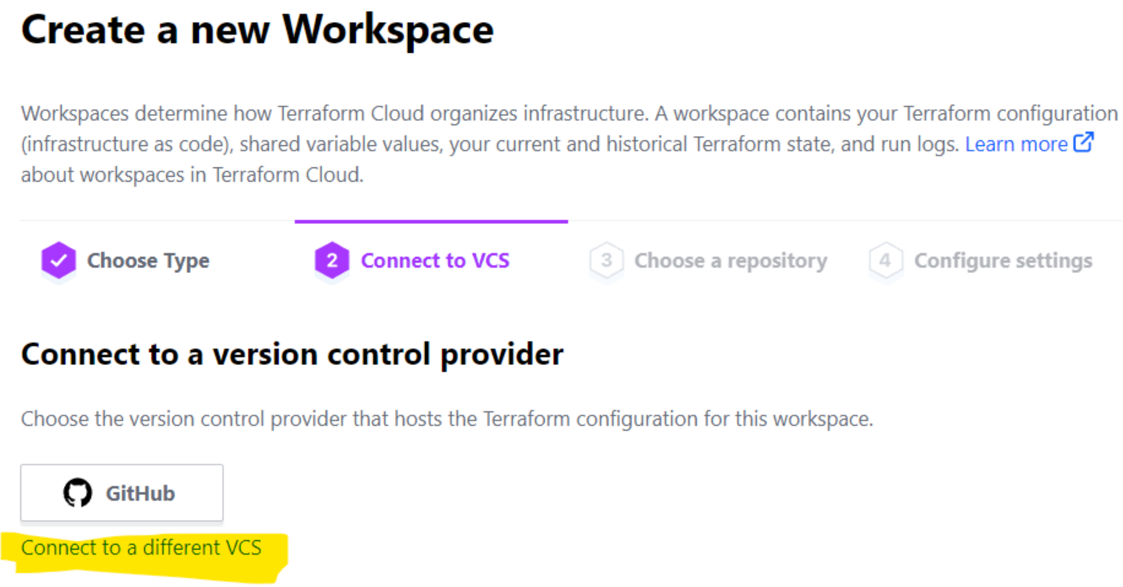
Learn More

CLI-driven workflow

Trigger remote Terraform runs from your local command line.

Learn More

□ Under connect to a version control provider, click on **connect to a different VCS** as the only option displayed is **github**



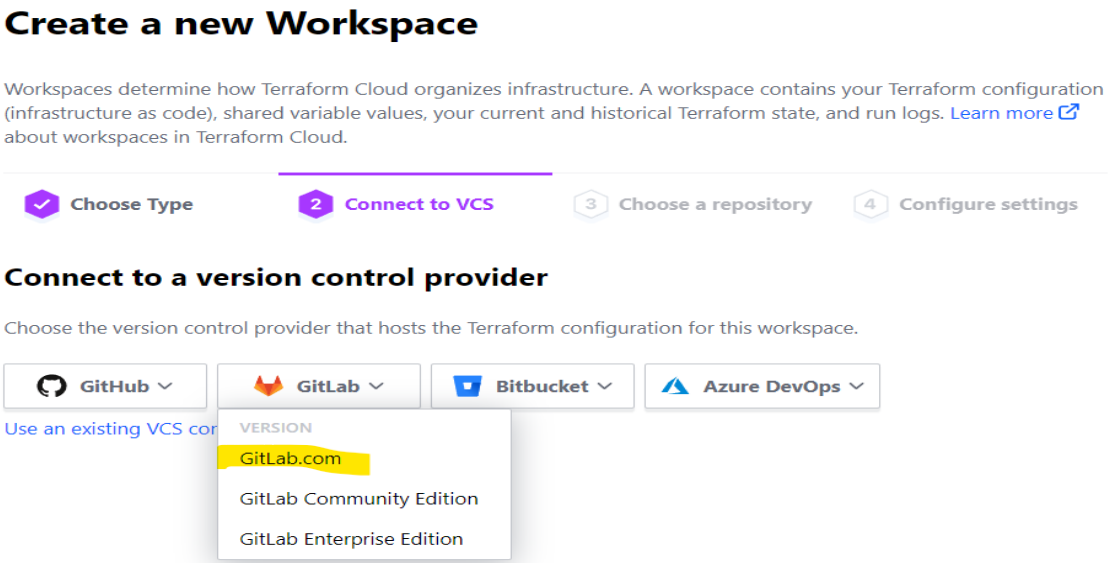
Create a new Workspace

Workspaces determine how Terraform Cloud organizes infrastructure. A workspace contains your Terraform configuration (infrastructure as code), shared variable values, your current and historical Terraform state, and run logs. Learn more about workspaces in Terraform Cloud.

Connect to a version control provider

Choose the version control provider that hosts the Terraform configuration for this workspace.

□ On the dropdown for gitlab, select gitlab.com



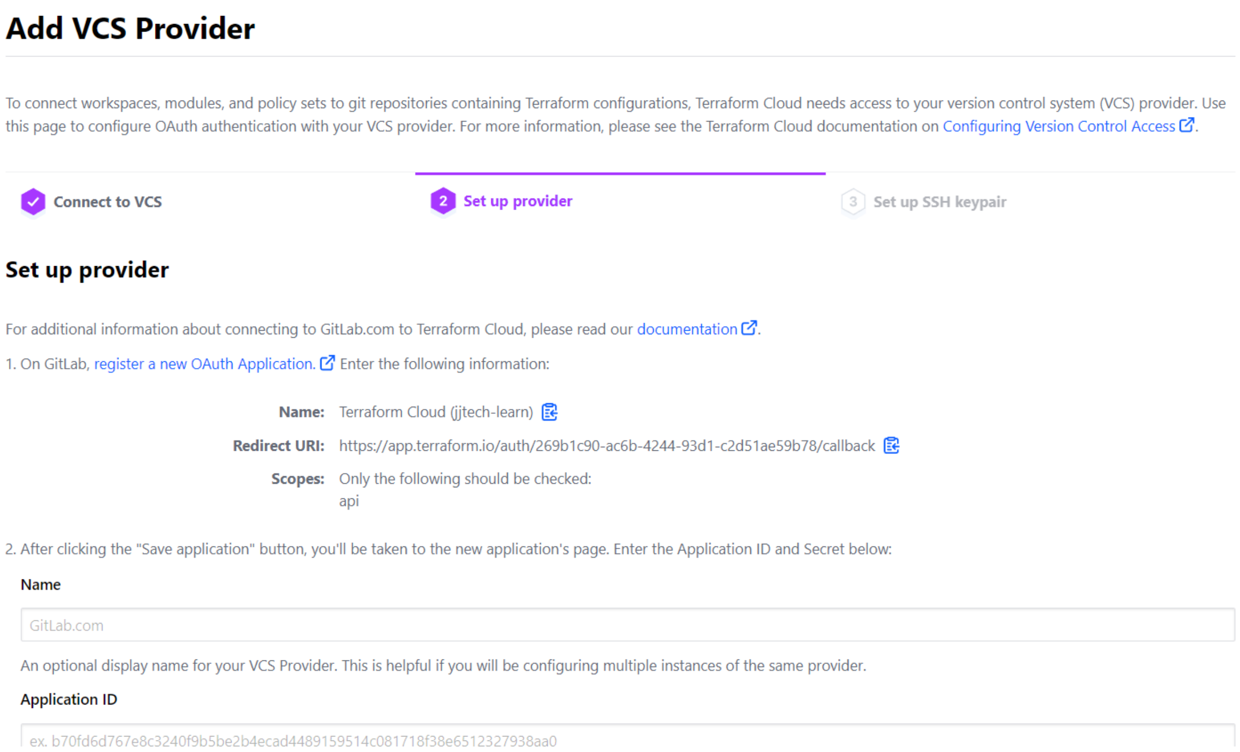
Create a new Workspace

Workspaces determine how Terraform Cloud organizes infrastructure. A workspace contains your Terraform configuration (infrastructure as code), shared variable values, your current and historical Terraform state, and run logs. Learn more about workspaces in Terraform Cloud.

Connect to a version control provider

Choose the version control provider that hosts the Terraform configuration for this workspace.

□ Follow the instructions in the next prompt from 1 nad 2.



Add VCS Provider

To connect workspaces, modules, and policy sets to git repositories containing Terraform configurations, Terraform Cloud needs access to your version control system (VCS) provider. Use this page to configure OAuth authentication with your VCS provider. For more information, please see the Terraform Cloud documentation on Configuring Version Control Access.

Set up provider

For additional information about connecting to GitLab.com to Terraform Cloud, please read our documentation.

1. On GitLab, register a new OAuth Application. Enter the following information:

Name: Terraform Cloud (jjtech-learn)

Redirect URI: https://app.terraform.io/auth/269b1c90-ac6b-4244-93d1-c2d51ae59b78/callback

Scopes: Only the following should be checked:
api

2. After clicking the "Save application" button, you'll be taken to the new application's page. Enter the Application ID and Secret below:

Name

GitLab.com

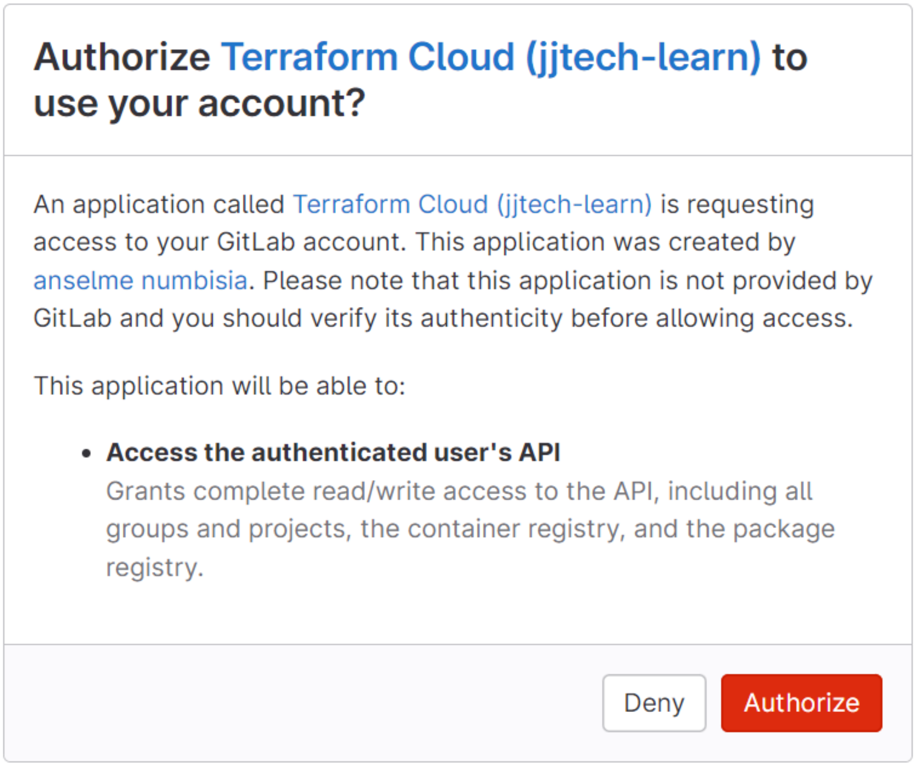
An optional display name for your VCS Provider. This is helpful if you will be configuring multiple instances of the same provider.

Application ID

ex. b70fd6d767e8c3240f9b5be2b4ecad4489159514c081718f38e6512327938aa0

□ After filling out the required information, Click on **connect and continue**

□ When prompted to authorise Terraform cloud access to gitlab, select **Authorize**



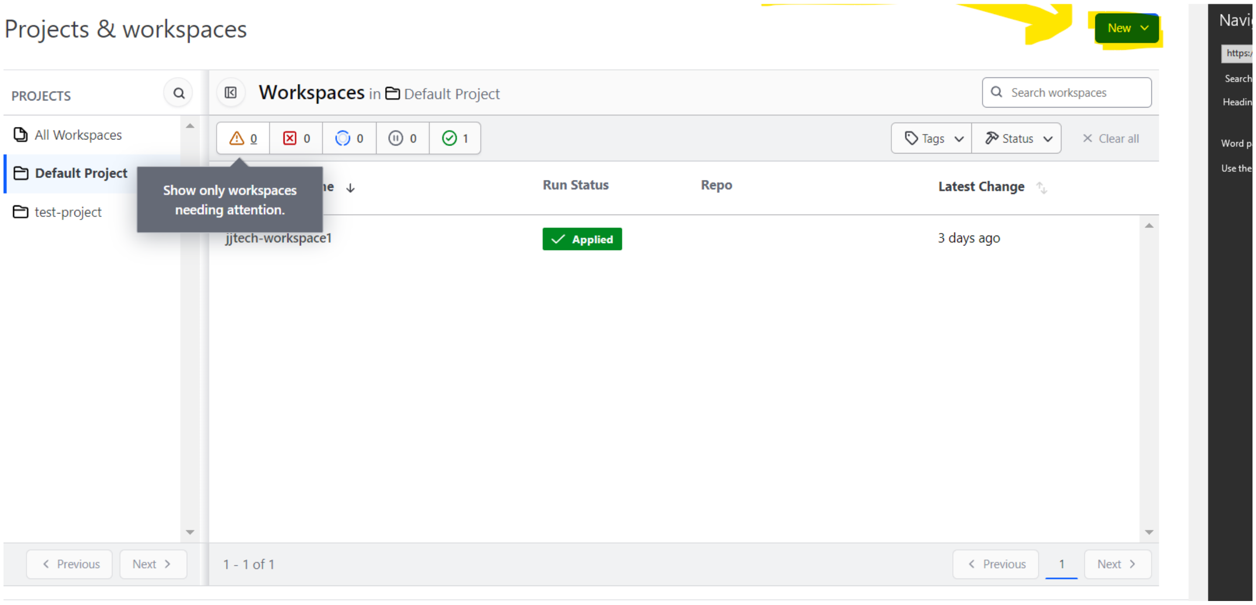
Authorize Terraform Cloud (jjtech-learn) to use your account?

An application called Terraform Cloud (jjtech-learn) is requesting access to your GitLab account. This application was created by anselme numbisia. Please note that this application is not provided by GitLab and you should verify its authenticity before allowing access.

This application will be able to:

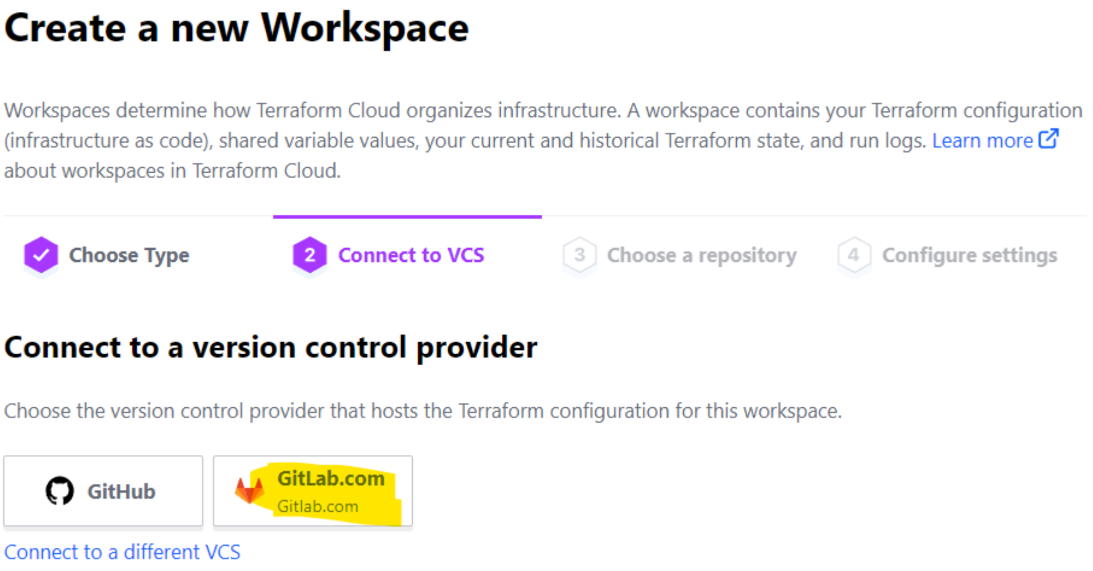
- **Access the authenticated user's API**
  Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

- □ On the next page for SSH, scroll down and click on **skip and finish** as we do not need to configure ssh access. You can follow instrictions from this page to set up ssh keys
- □ After configuring your provider, navigate back to workspaces in terraform cloud and create a new workspace by clicking on **New**, then **Workkspace**



...e ↓	Run Status	Repo	Latest Change
jjtech-workspace1	✓ Applied		3 days ago

□ In the next step select **Version control workflow** >> under **connect to a version control provider,** select **gitlab**



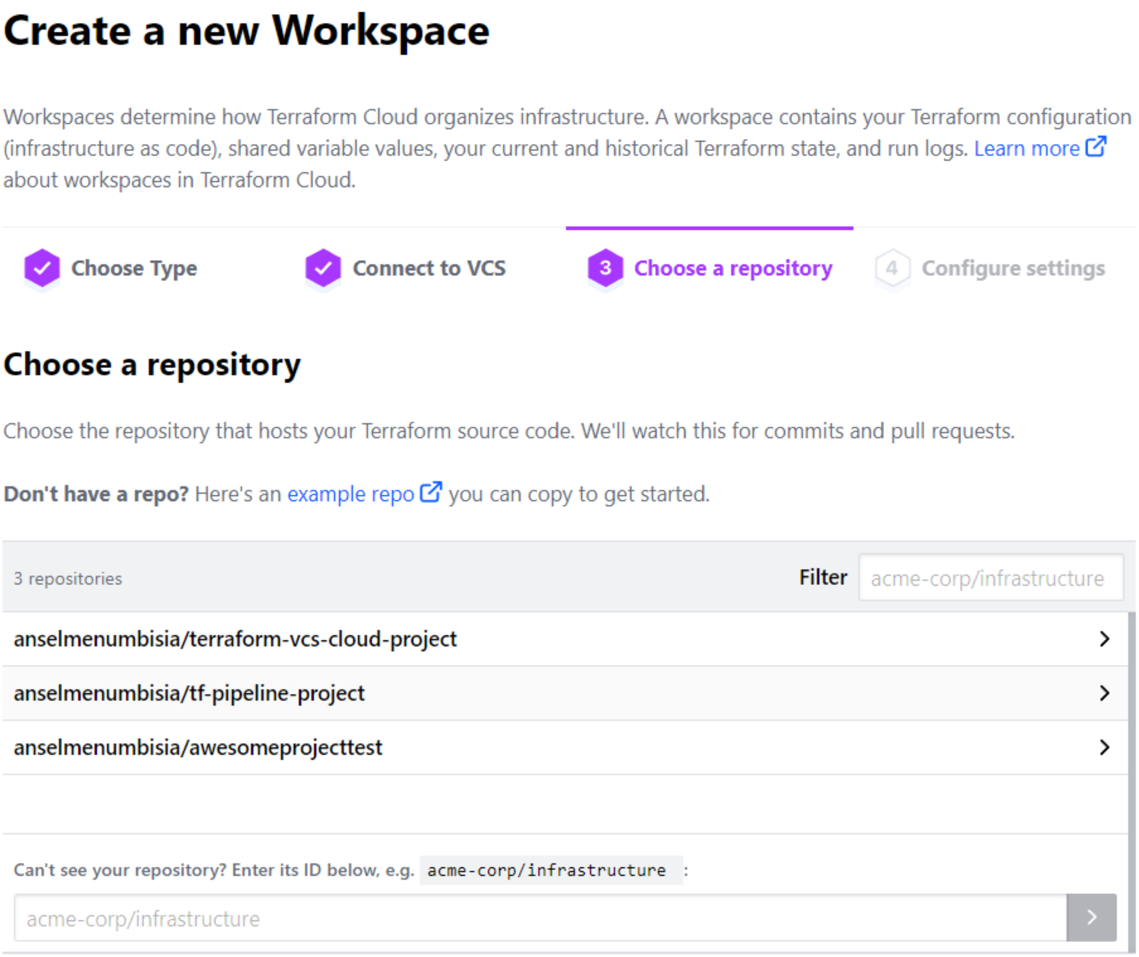
Create a new Workspace

Workspaces determine how Terraform Cloud organizes infrastructure. A workspace contains your Terraform configuration (infrastructure as code), shared variable values, your current and historical Terraform state, and run logs. Learn more about workspaces in Terraform Cloud.

Connect to a version control provider

Choose the version control provider that hosts the Terraform configuration for this workspace.

□ Choose the repository where your terraform project is found



Create a new Workspace

Workspaces determine how Terraform Cloud organizes infrastructure. A workspace contains your Terraform configuration (infrastructure as code), shared variable values, your current and historical Terraform state, and run logs. Learn more about workspaces in Terraform Cloud.

Choose a repository

Choose the repository that hosts your Terraform source code. We'll watch this for commits and pull requests.

Don't have a repo? Here's an example repo you can copy to get started.

3 repositories   **Filter** acme-corp/infrastructure

- anselmenumbisia/terraform-vcs-cloud-project
- anselmenumbisia/tf-pipeline-project
- anselmenumbisia/awesomeprojecttest

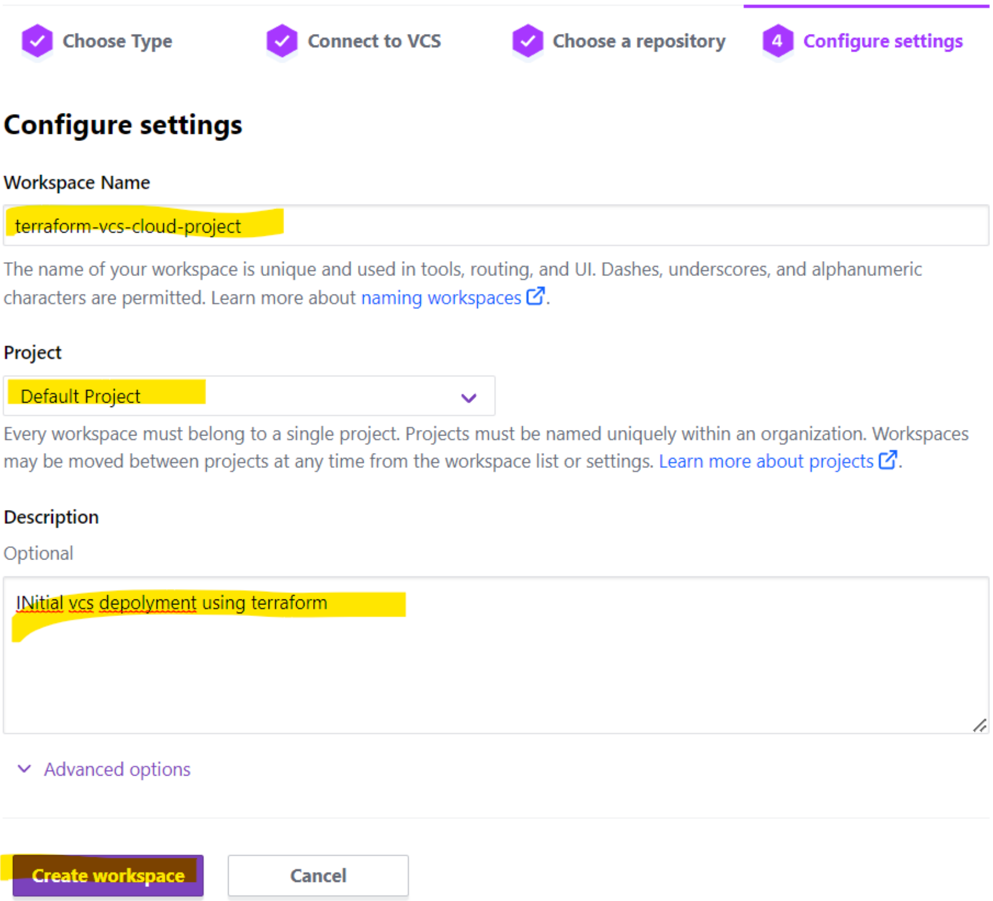
Can't see your repository? Enter its ID below, e.g. `acme-corp/infrastructure` :

acme-corp/infrastructure

□ Provide a Workspace name

□ Select **default Project** in **Project section**

□ Provide a description



Configure settings

Workspace Name

terraform-vcs-cloud-project

The name of your workspace is unique and used in tools, routing, and UI. Dashes, underscores, and alphanumeric characters are permitted. Learn more about naming workspaces.

Project

Default Project

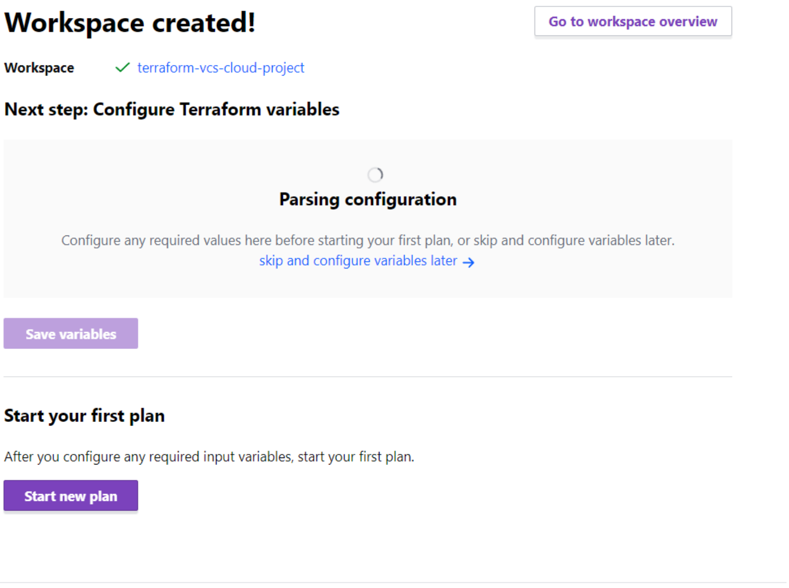
Every workspace must belong to a single project. Projects must be named uniquely within an organization. Workspaces may be moved between projects at any time from the workspace list or settings. Learn more about projects.

Description

Optional

INitial vcs depolyment using terraform

- □ Then create workspace.



Workspace created!

Workspace ✓ terraform-vcs-cloud-project

Next step: Configure Terraform variables

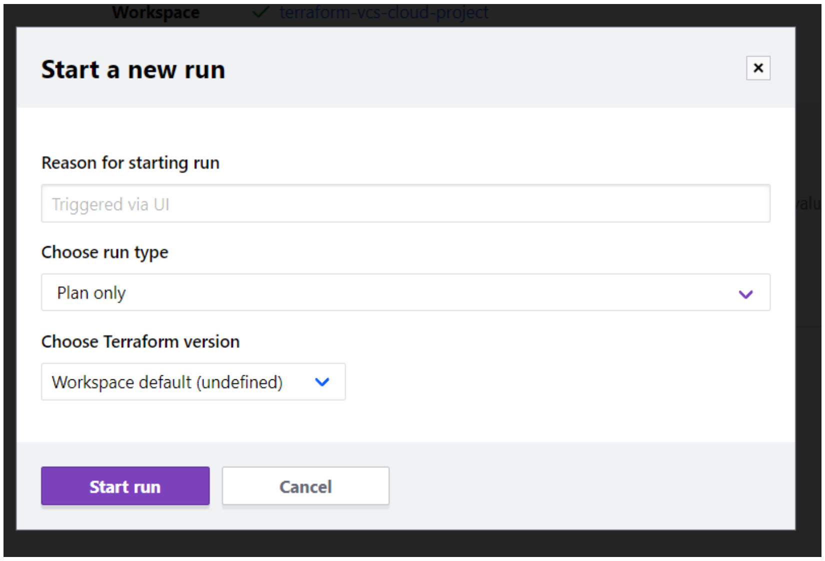
Parsing configuration

Configure any required values here before starting your first plan, or skip and configure variables later.
skip and configure variables later →

Start your first plan

After you configure any required input variables, start your first plan.

- □ Click on **start a new plan** >> provide reason for starting run >> under **Choose type run**, select **Plan only >> start run**



Workspace ✓ terraform-vcs-cloud-project

Start a new run

Reason for starting run

Triggered via UI

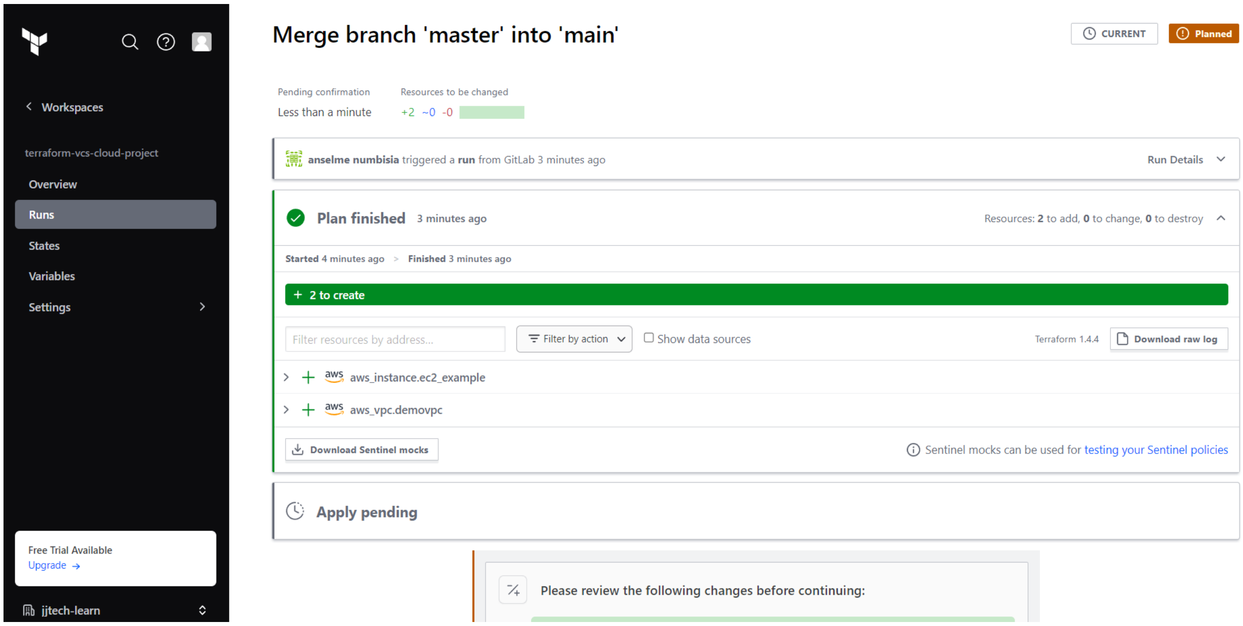
Choose run type

Plan only

Choose Terraform version

Workspace default (undefined)

- □ Observe how plan is triggered. If failed, Navigate back to gitlab and crate a merge request from master to main branch and once merge request is approved, terraform cloud triggers plan



Merge branch 'master' into 'main'

Pending confirmation: Less than a minute

Resources to be changed: +2 ~0 -0

anselme numbisia triggered a **run** from GitLab 3 minutes ago

Plan finished 3 minutes ago

Resources: **2** to add, **0** to change, **0** to destroy

Started 4 minutes ago > **Finished** 3 minutes ago

+ 2 to create

Terraform 1.4.4

aws_instance.ec2_example

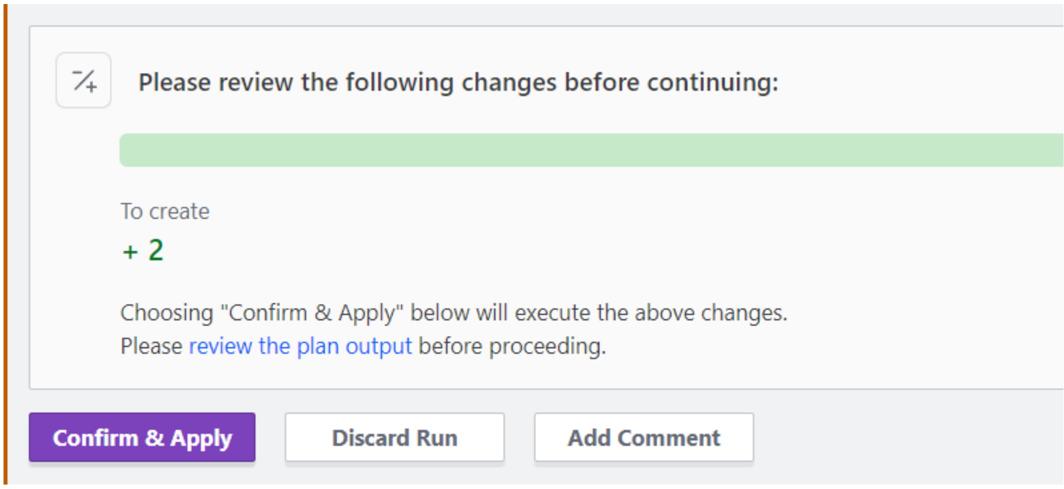
aws_vpc.demovpc

Sentinel mocks can be used for testing your Sentinel policies

Apply pending

Please review the following changes before continuing:

- □ If plan is Ok, click on **confirm and apply**

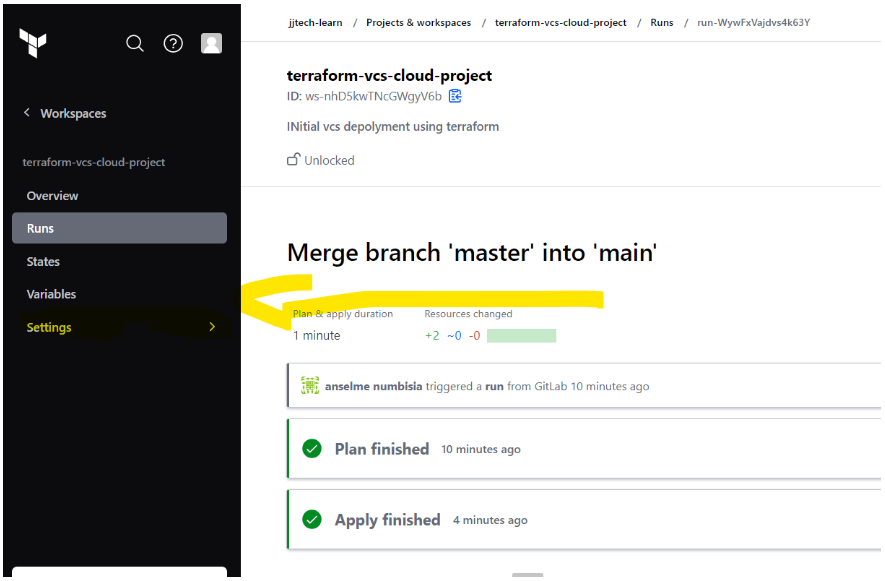


Please review the following changes before continuing:

To create
+ 2

Choosing "Confirm & Apply" below will execute the above changes.
Please review the plan output before proceeding.

□ To destroy the infrastructure, naviaget to the setttings tab of your terraform cloud workspace



jjtech-learn / Projects & workspaces / terraform-vcs-cloud-project / Runs / run-WywFxVajdvs4k63Y

terraform-vcs-cloud-project
ID: ws-nhD5kwTNcGWgyV6b

INitial vcs depolyment using terraform

Unlocked

Merge branch 'master' into 'main'

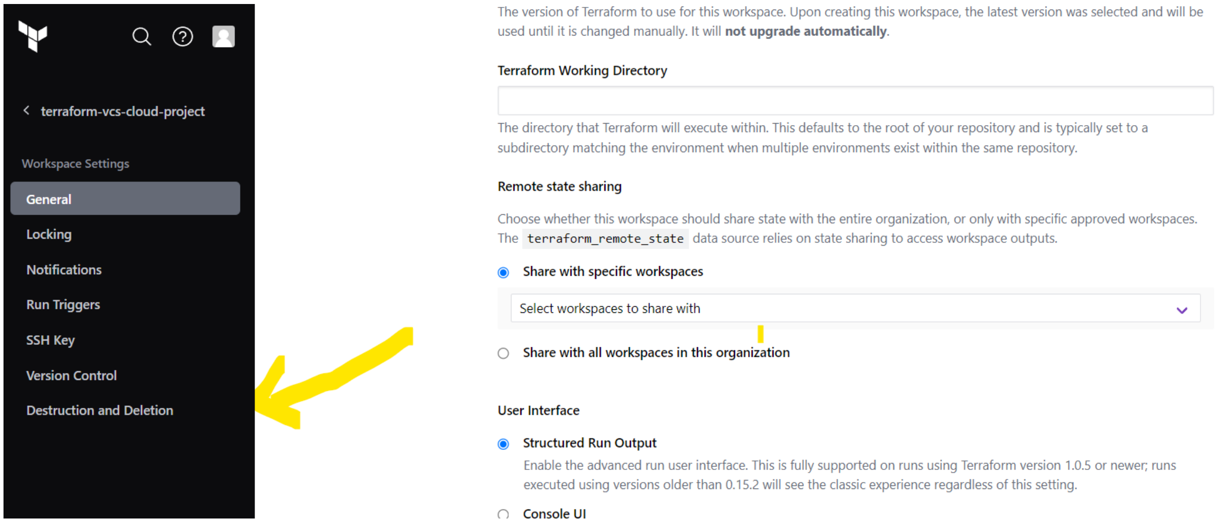
Plan & apply duration	Resources changed
1 minute	+2 ~0 -0

anselme numbisia triggered a run from GitLab 10 minutes ago

Plan finished 10 minutes ago

Apply finished 4 minutes ago

□ In the next page, click on Destruction and Deletion in the next tab



The version of Terraform to use for this workspace. Upon creating this workspace, the latest version was selected and will be used until it is changed manually. It will **not upgrade automatically**.

Terraform Working Directory

The directory that Terraform will execute within. This defaults to the root of your repository and is typically set to a subdirectory matching the environment when multiple environments exist within the same repository.

Remote state sharing

Choose whether this workspace should share state with the entire organization, or only with specific approved workspaces. The `terraform_remote_state` data source relies on state sharing to access workspace outputs.

● Share with specific workspaces

Select workspaces to share with

○ Share with all workspaces in this organization

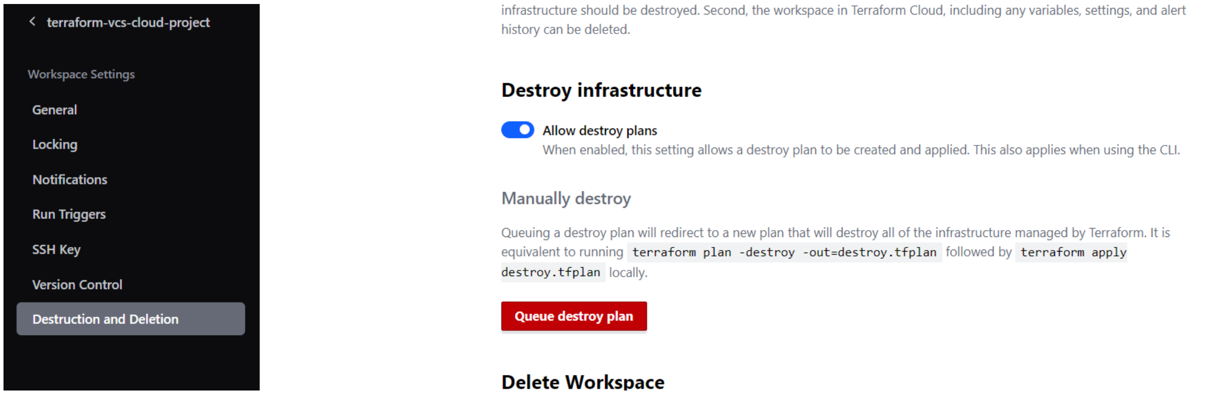
User Interface

● Structured Run Output

Enable the advanced run user interface. This is fully supported on runs using Terraform version 1.0.5 or newer; runs executed using versions older than 0.15.2 will see the classic experience regardless of this setting.

○ Console UI

□ Select Queue destroy plan



infrastructure should be destroyed. Second, the workspace in Terraform Cloud, including any variables, settings, and alert history can be deleted.

Destroy infrastructure

Allow destroy plans

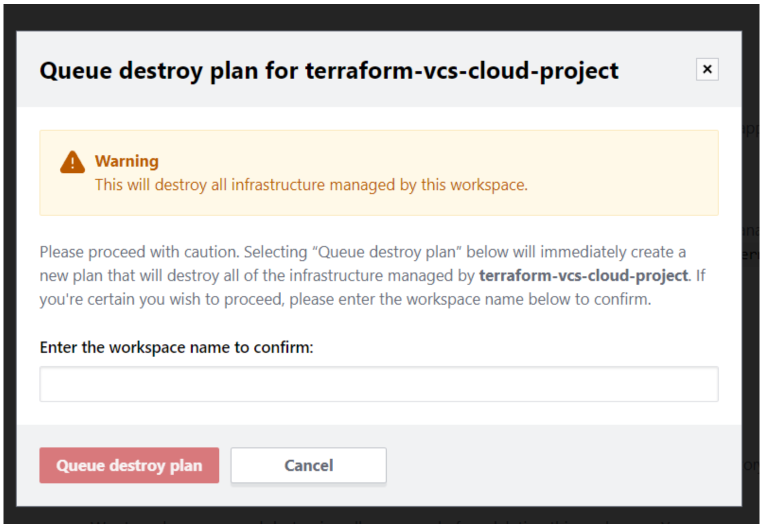
When enabled, this setting allows a destroy plan to be created and applied. This also applies when using the CLI.

Manually destroy

Queuing a destroy plan will redirect to a new plan that will destroy all of the infrastructure managed by Terraform. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally.

Delete Workspace

□ On the next prompt , enter the workspace name and click **queue destroy plan**



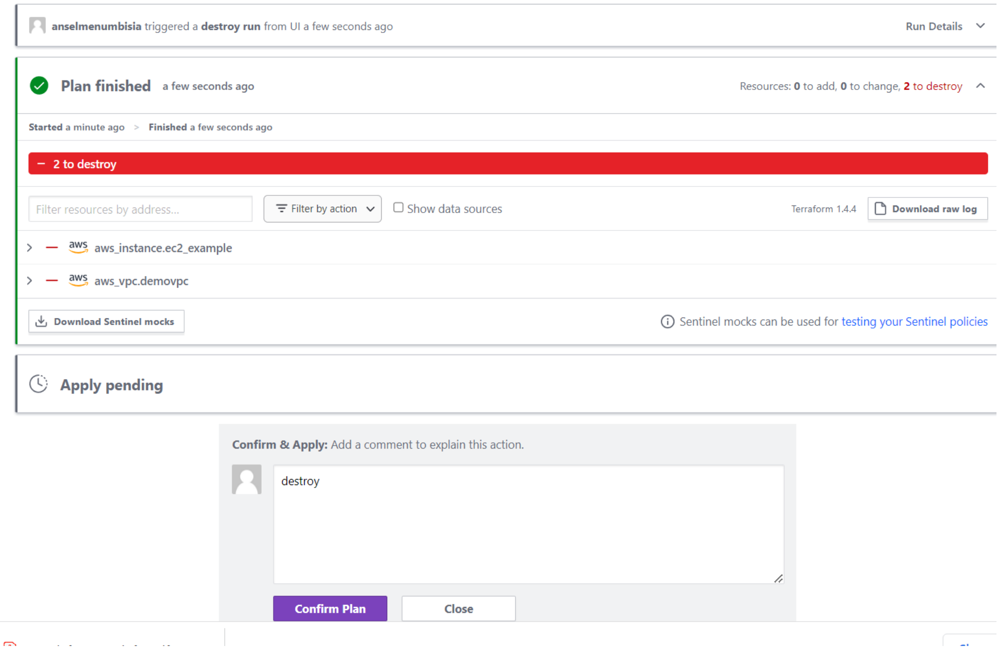
Queue destroy plan for terraform-vcs-cloud-project

Warning
This will destroy all infrastructure managed by this workspace.

Please proceed with caution. Selecting "Queue destroy plan" below will immediately create a new plan that will destroy all of the infrastructure managed by **terraform-vcs-cloud-project**. If you're certain you wish to proceed, please enter the workspace name below to confirm.

Enter the workspace name to confirm:

□ Approve the destroy and click on **Confirm plan** to destroy



anselmenumbisia triggered a **destroy run** from UI a few seconds ago

Plan finished a few seconds ago — Resources: **0** to add, **0** to change, **2** to destroy

Started a minute ago > **Finished** a few seconds ago

− 2 to destroy

Terraform 1.4.4

− aws_instance.ec2_example

− aws_vpc.demovpc

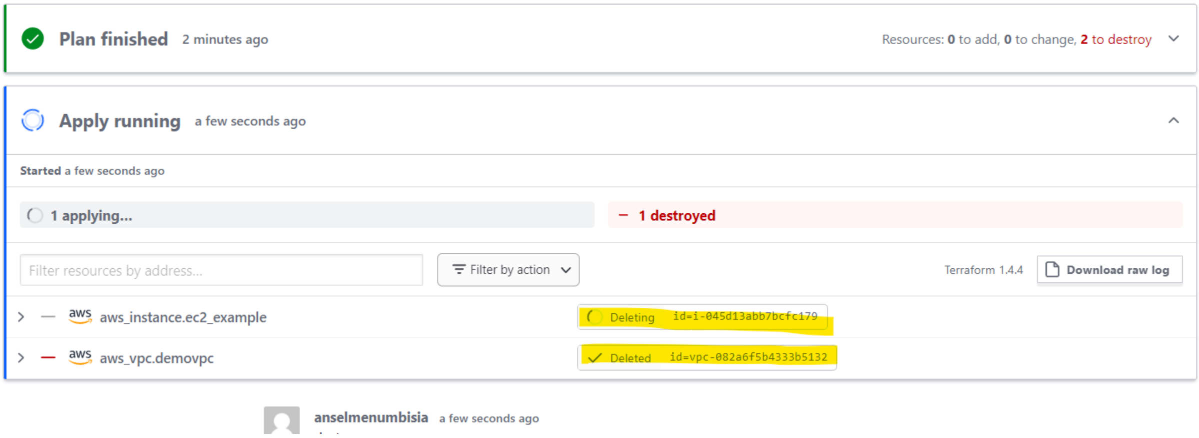
Sentinel mocks can be used for testing your Sentinel policies

Apply pending

Confirm & Apply: Add a comment to explain this action.

destroy

□ Resources are getting deleted



✓ Plan finished 2 minutes ago

Resources: 0 to add, 0 to change, 2 to destroy

Apply running a few seconds ago

Started a few seconds ago

1 applying...

— 1 destroyed

Filter resources by address...

Filter by action

Terraform 1.4.4

Download raw log

aws_instance.ec2_example	Deleting id=i-045d13abb7bcfc179
aws_vpc.demovpc	✓ Deleted id=vpc-082a6f5b4333b5132

anselmenumbisia a few seconds ago