

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

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

Basic Terraform Knowledge ([resource](#))

Understanding of CICD ([resource](#))

GitLab CI Knowledge ([resource](#))

PREREQUISITES:

1) Aws account creation

Check out the official site to create aws account [Here](#)

2) GitLab account

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

3) Terraform Installed

Check out the official website to install terraform [Here](#)

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

4) AWS CLI Installed

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

The screenshot shows the AWS IAM console 'Create user' page. The breadcrumb navigation is 'IAM > Users > Create user'. On the left, there is a sidebar with three steps: 'Step 1: Specify user details' (active), 'Step 2: Set permissions', and 'Step 3: Review and create'. The main content area is titled 'Specify user details' and contains a 'User details' section. In this section, the 'User name' field is filled with 'awscli1'. Below this field, a note states: 'The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , _ - (hyphen)'. There is an unchecked checkbox for 'Provide user access to the AWS Management Console - optional' with a note: 'If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.' At the bottom of the 'User details' section, there is a blue information box with a question mark icon and text: 'If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)'. At the bottom right of the page, there are 'Cancel' and 'Next' buttons.

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

Step 2
Set permissions

Permissions options

- ☐ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- ☐ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.
- ☒ Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1186)
Choose one or more policies to attach to your new user.

Filter by Type
Search: [] All types [v]

	Policy name	Type	Attached entities
<input type="checkbox"/>	AccessAnalyzerServiceRolePolicy	AWS managed	0
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job function	0
<input type="checkbox"/>	AdministratorAccess-Amplify	AWS managed	0

Identity and Access Management (IAM)

Search IAM []

Dashboard

Access management

- User groups
- Users**
- Roles
- Policies

User created successfully
You can view and download the user's password and email instructions for signing in to the AWS Management Console. [View user]

Users (1) Info [Refresh] [Delete] [Create user]

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

Search []

	User name	Path	Groups	Last activity	MFA	Password age	Console last sign-in
<input type="checkbox"/>	awscli	/	0		-	-	-

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

us-east-1.console.aws.amazon.com/iam/home#/users/details/awscli/create-access-key

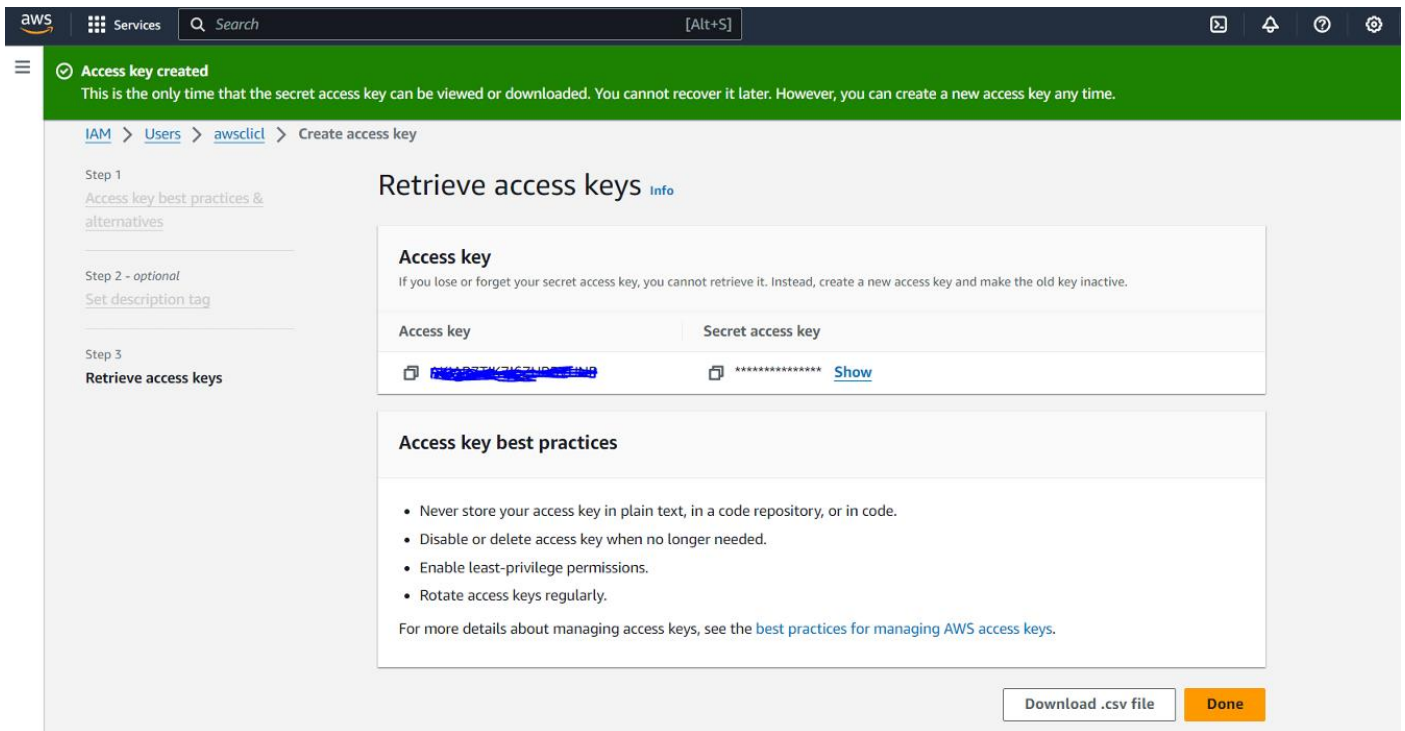
Step 2 - optional
Set description tag

Step 3
Retrieve access keys

Use case

- ☒ **Command Line Interface (CLI)**
You plan to use this access key to enable the AWS CLI to access your AWS account.
- ☐ Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.
- ☐ Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- ☐ Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors

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



Now go to your terminal and follow below steps:

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

5) Code editor (Vscode)

Download it from [Here](#)

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

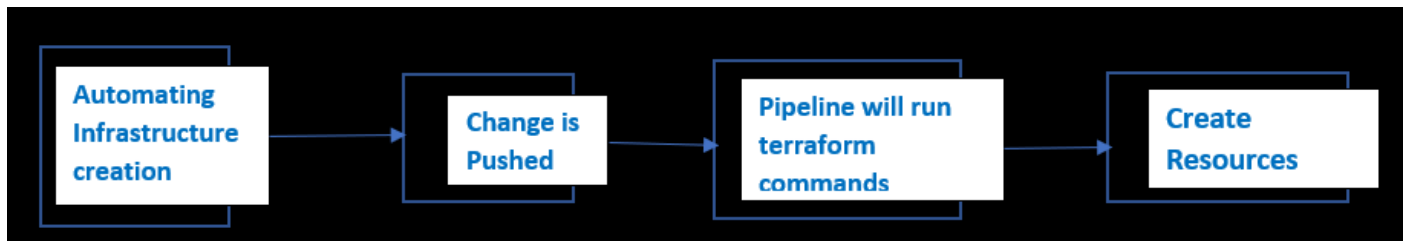
Part1:



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

Part2:

Create CICD pipeline script on Gitlab to automate terraform resource creation



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

```
ubuntu@ip-172-31-25-121:~$ mkdir cicdtf
ubuntu@ip-172-31-25-121:~$ ls
aws  awscli2.zip  cicdtf
ubuntu@ip-172-31-25-121:~$ cd cicdtf/
ubuntu@ip-172-31-25-121:~/cicdtf$ code .
```

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

```
1 provider "aws" {
2     region = "us-east-1"
3 }
4
5 # This snippet configures Terraform to use the AWS provider
6 # and specifies that resources should be created in the us-east-1 region.
```

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

The folder structure is as follows:

1. VPC Module (vpc folder):

Files:

main.tf: Defines resources like VPC, subnets, and security groups.

variables.tf: Declares input variables for customization.

outputs.tf: Specifies outputs like VPC ID, subnet IDs, etc.

2. EC2 Module (web folder):

Files:

main.tf: Configures EC2 instance details, including AMI, instance type, and security groups.

variables.tf: Defines variables needed for EC2 instance customization.

outputs.tf: Outputs instance details like public IP, instance ID, etc.

Snap of folder structure:



Let's start with defining vpc,

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

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

Make use of below repositories to check out the code.

<https://gitlab.com/N4si/cicdtf>

<https://gitlab.com/Sakeena19/cicdtf>

```

1 # Define an AWS VPC resource named "myvpc"
2 resource "aws_vpc" "myvpc" {
3     cidr_block      = "10.0.0.0/16"          # Define the CIDR block for the VPC
4     enable_dns_hostnames = true              # Enable DNS hostnames in the VPC
5     enable_dns_support = true               # Enable DNS support in the VPC
6
7     tags = {
8         Name = "myvpc"                      # Set a tag for the VPC resource
9     }
10 }
11
12 # Define an AWS subnet resource named "pb_sn"
13 resource "aws_subnet" "pb_sn" {
14     vpc_id            = aws_vpc.myvpc.id      # Reference the VPC ID from the "myvpc" resource
15     cidr_block        = "10.0.1.0/24"        # Define the CIDR block for the subnet
16     map_public_ip_on_launch = true            # Enable automatic assignment of public IPs to instances
17     availability_zone  = "us-east-1a"        # Specify the availability zone for the subnet
18
19     tags = {
20         Name = "pb_sn1"                     # Set a tag for the subnet resource
21     }
22 }
23
24 # Define an AWS security group resource named "sg"
25 resource "aws_security_group" "sg" {
26     vpc_id            = aws_vpc.myvpc.id      # Reference the VPC ID from the "myvpc" resource
27     name              = "my_sg"              # Specify the name for the security group
28     description       = "Public Security Group" # Provide a description for the security group
29
30     # Define an ingress rule allowing inbound traffic on port 22 (SSH) from any IP address
31     ingress {
32         from_port = 22                      # Specify the starting port for inbound traffic
33         to_port   = 22                      # Specify the ending port for inbound traffic
34         protocol  = "tcp"                   # Specify the protocol (TCP in this case)
35         cidr_blocks = ["0.0.0.0/0"]         # Allow inbound traffic from any IP address
36     }
37
38     # Define an egress rule allowing all outbound traffic (any port, any protocol) to any IP address
39     egress {
40         from_port = 0                      # Specify the starting port for outbound traffic
41         to_port   = 0                      # Specify the ending port for outbound traffic
42         protocol  = "-1"                   # Specify all protocols for outbound traffic
43         cidr_blocks = ["0.0.0.0/0"]         # Allow outbound traffic to any IP address
44     }
45 }
46

```

Step3:

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

Main Module (Root Module): The main.tf file acts as the parent module

Child Modules: The VPC and web modules are child modules.

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

Define Outputs: Specify the values (e.g., subnet ID, security group ID) as outputs in the VPC module.

Use Variables: Reference these outputs as variables in the web module.

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

```
1 # Define an AWS EC2 instance resource named "server" # main.tf file in web module
2
3
4 resource "aws_instance" "server" {
5     ami           = "ami-04a81a99f5ec58529" # Specify the AMI ID (Amazon Machine Image) for the instance
6     instance_type = "t2.micro"               # Specify the instance type (e.g., t2.micro)
7     subnet_id     = var.sn                  # Use the subnet ID variable from the VPC module's output
8     security_groups = [var.sg]              # Use the security group ID variable from the VPC module's output
9
10    tags = {
11        Name = "my_server"                  # Set a tag for the EC2 instance for identification
12    }
13 }
```

Step4: define outputs.tf file in vpc module.

```
1 # outputs.tf file in vpc module
2
3 #Output the subnet ID
4 output "pb_sn" {
5     value = aws_subnet.pb_sn.id # This refers to the ID of the subnet created in the VPC module
6 }
7
8 # Output the security group ID
9 output "sg" {
10    value = aws_security_group.sg.id # This refers to the ID of the security group created in the VPC module
11 }
12
```

output "pb_sn": Defines an output variable named pb_sn.

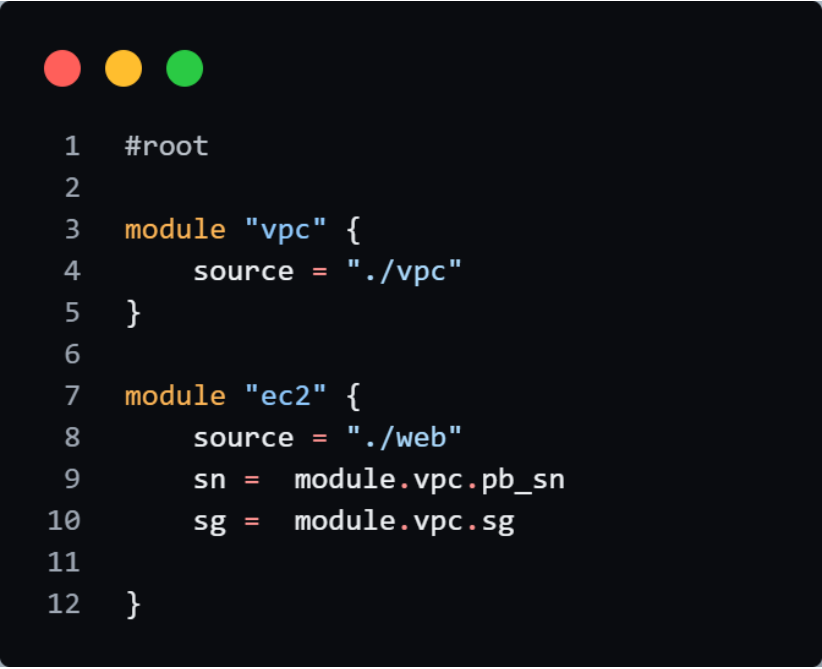
value = aws_subnet.pb_sn.id: This line assigns the ID of the subnet resource (aws_subnet.pb_sn) to the output variable. This allows other modules to access the subnet ID. Similar for security group as well.

Step5: Define variables.tf file in web module.

```
1 # variables.tf file in web module
2
3 # Define a variable to hold the security group ID
4 variable "sg" {
5     description = "The ID of the security group"
6 }
7
8 # Define a variable to hold the subnet ID
9 variable "sn" {
10     description = "The ID of the subnet"
11 }
```

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

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



```
1  #root
2
3  module "vpc" {
4      source = "./vpc"
5  }
6
7  module "ec2" {
8      source = "./web"
9      sn = module.vpc.pb_sn
10     sg = module.vpc.sg
11
12 }
```

source = "./vpc": Specifies the path to the VPC module directory. This imports the VPC module defined in the ./vpc folder.

source = "./web": Specifies the path to the web module directory. This imports the EC2 module defined in the ./web folder.

sn = module.vpc.pb_sn: Passes the subnet ID output (pb_sn) from the VPC module to the EC2 module, assigning it to the variable sn.

sg = module.vpc.sg: Passes the security group ID output (sg) from the VPC module to the EC2 module, assigning it to the variable sg.

Step7:

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

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

To check if our code is valid, use “**terraform validate**” command.

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

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


```
● ubuntu@ip-172-31-25-53:~/cicdtf$ terraform init
```

```
Initializing the backend...
```

```
Initializing modules...
```

```
- ec2 in web
```

```
- vpc in vpc
```

```
Initializing provider plugins...
```

```
- Finding latest version of hashicorp/aws...
```

```
- Installing hashicorp/aws v5.57.0...
```

```
- Installed hashicorp/aws v5.57.0 (signed by HashiCorp)
```

Terraform has created a lock file **.terraform.lock.hcl** to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

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

```
● ubuntu@ip-172-31-25-53:~/cicdtf$ terraform validate
```

```
Success! The configuration is valid.
```

```
● ubuntu@ip-172-31-25-53:~/cicdtf$ terraform plan
```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

```
# module.ec2.aws_instance.server will be created
```

```
+ resource "aws_instance" "server" {  
  + ami                        = "ami-04a81a99f5ec58529"  
  + arn                       = (known after apply)  
  + associate_public_ip_address = (known after apply)  
  + availability_zone         = (known after apply)  
  + cpu_core_count            = (known after apply)  
  + cpu_threads_per_core      = (known after apply)  
  + disable_api_stop          = (known after apply)  
  + disable_api_termination   = (known after apply)
```

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

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

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

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

```

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

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# module.ec2.aws_instance.server will be created
+ resource "aws_instance" "server" {
  + ami                  = "ami-04a81a99f5ec58529"
  + arn                  = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone     = (known after apply)
  + cpu_core_count       = (known after apply)
  + cpu_threads_per_core = (known after apply)
  + disable_api_stop      = (known after apply)
  + disable_api_termination = (known after apply)
  + ebs_optimized         = (known after apply)
  + get_password_data     = false
  + host_id               = (known after apply)
  + host_resource_group_arn = (known after apply)
  + iam_instance_profile   = (known after apply)
  + id                    = (known after apply)
  + owner_id              = (known after apply)
  + tags                  = {
    + "Name" = "myvpc"
  }
  + tags_all              = {
    + "Name" = "myvpc"
  }
}

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

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

```

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

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

Instances (2) Info

Refresh

Connect

Instance state ▾

Actions ▾

Launch instances ▾

Find Instance by attribute or tag (case-sensitive)

All states ▾

<

1

>

⚙

<input type="checkbox"/>	Name <div>↗</div> ▾	Instance ID	Instance state ▾	Instance type ▾	Status check	Alarm status	Availability Zone ▾
<input type="checkbox"/>	terraform_project	i-075cf72513ff0d349	<div>✔ Running</div> <div>🔄</div> <div>🔍</div>	t2.micro	<div>✔ 2/2 checks passed</div>	<div>View alarms +</div>	us-east-1a
<input type="checkbox"/>	my_server	i-0314991e88264715d	<div>✔ Running</div> <div>🔄</div> <div>🔍</div>	t2.micro	<div>✔ 2/2 checks passed</div>	<div>View alarms +</div>	us-east-1a

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

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

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

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- destroy

Terraform will perform the following actions:

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

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

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

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

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

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

The code for creating s3 and DynamoDB is as follows:

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

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

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



```
1  provider "aws" {
2      region = "us-east-1" # Change to your preferred region
3  }
4
5  # Create an S3 bucket
6  resource "aws_s3_bucket" "mybucket" {
7      bucket = "s3statefile786" # Change to a unique bucket name
8
9      # Optional: Adding tags to the bucket
10     tags = {
11         Name          = "s3statefile786"
12         Environment    = "Dev"
13     }
14 }
15
16 # Manage versioning for the S3 bucket
17 resource "aws_s3_bucket_versioning" "mybucket_versioning" {
18     bucket = aws_s3_bucket.mybucket.id
19
20     versioning_configuration {
21         status = "Enabled"
22     }
23 }
24
25 # Manage server-side encryption for the S3 bucket
26 resource "aws_s3_bucket_server_side_encryption_configuration" "mybucket_encryption" {
27     bucket = aws_s3_bucket.mybucket.id
28
29     rule {
30         apply_server_side_encryption_by_default {
31             sse_algorithm = "AES256"
32         }
33     }
34 }
35
36 #create Dynamodb for state-locking
37
38 resource "aws_dynamodb_table" "state-lock" {
39     name = "state-lock"
40     billing_mode = "PAY_PER_REQUEST"
41     hash_key = "LOCKID"
42
43     attribute {
44         name = "LOCKID"
45         type = "S"
46     }
47 }
```

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

Amazon S3

► **Account snapshot** - updated every 24 hours All AWS Regions View Storage Lens dashboard

Storage lens provides visibility into storage usage and activity trends. [Learn more](#)

General purpose buckets | Directory buckets

General purpose buckets (1) All AWS Regions

Buckets are containers for data stored in S3.

Find buckets by name

Name	AWS Region	IAM Access Analyzer	Creation date
s3statefile786	US East (N. Virginia) us-east-1	View analyzer for us-east-1	July 11, 2024, 04:23:19 (UTC+05:30)

Share your feedback on Amazon DynamoDB

Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing. Share feedback

DynamoDB > Tables

Tables (1) All AWS Regions

Find tables by table name

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mo...	To
state-lock	Active	LOCKID (S)	-	0	Off	On-demand	On-demand	0 b

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

```

1 #backend configuration
2
3 terraform {
4     backend "s3" {
5         bucket = "s3statefile786"
6         key = "state-lock"
7         region = "us-east-1"
8         dynamodb_table = "state-lock"
9     }
10 }
11

```

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

Run "terraform init" to initialize the backend.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

  Terraform has been successfully initialized!

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


  If you ever set or change modules or backend configuration for Terraform,
  rerun this command to reinitialize your working directory. If you forget, other
  commands will detect it and remind you to do so if necessary.
○ ubuntu@ip-172-31-25-53:~/cicdtf$
```

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

Go to GitLab and create a new repository:

Click on new project -> create a blank project -> provide project name, visibility, enable readme -> create project.

Your work / Projects / New project / **Create blank project**



Create blank project

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


Project name


Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.



Project URL **Project slug**


 /

Project deployment target (optional)

Visibility Level 

☐  **Private**
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

☐  **Internal** 
The project can be accessed by any logged in user except external users.

☒  **Public**
The project can be accessed without any authentication.

Project Configuration

☒ **Initialize repository with a README**
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

☐ **Enable Static Application Security Testing (SAST)**
A tool that automatically finds and reports security vulnerabilities in your code.

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

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

To use only necessary files and ignore other files, create .gitignore file which can be found [Here](#)

To connect with your GitLab repo, use

git remote add origin <https://gitlab.com/Sakeena19/cicdtf.git>

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

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

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

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
<pre>● ubuntu@ip-172-31-25-53:~/cicdtf\$ git checkout -b dev Switched to a new branch 'dev' ● ubuntu@ip-172-31-25-53:~/cicdtf\$ git add . ● ubuntu@ip-172-31-25-53:~/cicdtf\$ git commit -m "initial commit" [dev (root-commit) 6de00d2] initial commit Committer: Ubuntu <ubuntu@ip-172-31-25-53.ec2.internal> Your name and email address were configured automatically based on your username and hostname. Please check that they are accurate. You can suppress this message by setting them explicitly. Run the following command and follow the instructions in your editor to edit your configuration file: git config --global --edit After doing this, you may fix the identity used for this commit with: git commit --amend --reset-author 14 files changed, 238 insertions(+)</pre>				


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ubuntu@ip-172-31-25-53:~/cicdtf$ git push -u origin dev
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (16/16), 4.15 KiB | 850.00 KiB/s, done.
Total 16 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for dev, visit:
remote: https://gitlab.com/Sakeena19/cicdtf/-/merge_requests/new?merge_request%5Bsource_branch%5D=dev
remote:
To https://gitlab.com/Sakeena19/cicdtf.git
* [new branch]      dev -> dev
branch 'dev' set up to track 'origin/dev'.
ubuntu@ip-172-31-25-53:~/cicdtf$
```

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

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

Sakeena Shaik / cicdtf / Repository

✔ You pushed to dev 5 minutes ago

Create merge request

dev cicdtf +

Compare History Find file Edit Code

initial commit

Ubuntu authored 7 minutes ago

6de00d2c

Name	Last commit	Last update
isolated-config	initial commit	7 minutes ago
vpc	initial commit	7 minutes ago
web	initial commit	7 minutes ago
.gitignore	initial commit	7 minutes ago
.terraform.lock.hcl	initial commit	7 minutes ago
backend.tf	initial commit	7 minutes ago

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

Sakeena Shaik / cicdtf / Repository

main cicdtf +

History Find file Edit Code

Merge branch 'dev' into 'main'

Sakeena Shaik authored just now

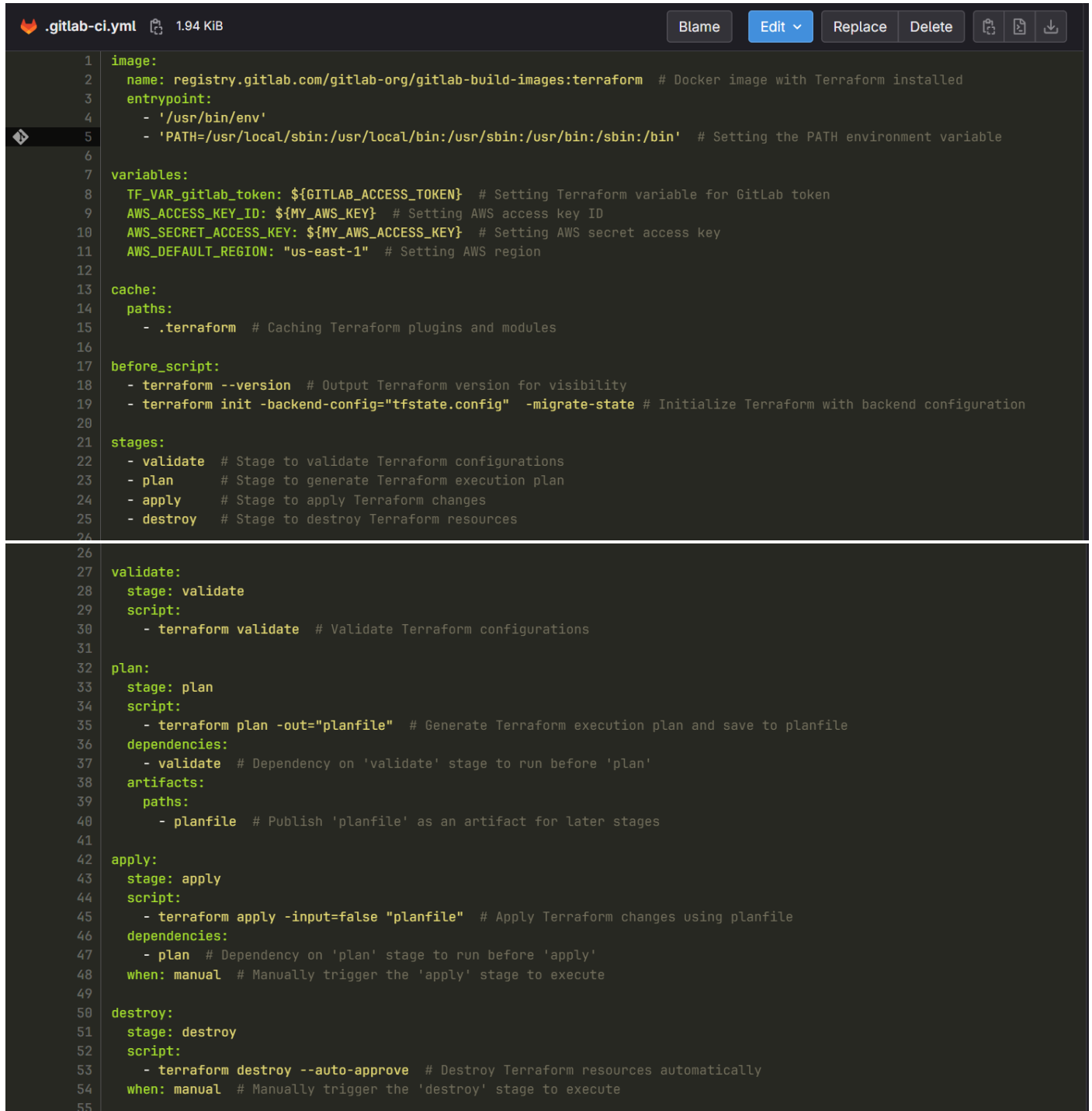
66268657

Name	Last commit	Last update
isolated-config	initial commit	9 minutes ago
vpc	initial commit	9 minutes ago
web	initial commit	9 minutes ago
.gitignore	initial commit	9 minutes ago
.terraform.lock.hcl	initial commit	9 minutes ago
README.md	initial commit	35 minutes ago
backend.tf	initial commit	9 minutes ago
main.tf	initial commit	9 minutes ago

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

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











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



```
1 image:
2   name: registry.gitlab.com/gitlab-org/gitlab-build-images:terraform # Docker image with Terraform installed
3   entrypoint:
4     - '/usr/bin/env'
5     - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin' # Setting the PATH environment variable
6
7 variables:
8   TF_VAR_gitlab_token: ${GITLAB_ACCESS_TOKEN} # Setting Terraform variable for GitLab token
9   AWS_ACCESS_KEY_ID: ${MY_AWS_KEY} # Setting AWS access key ID
10  AWS_SECRET_ACCESS_KEY: ${MY_AWS_ACCESS_KEY} # Setting AWS secret access key
11  AWS_DEFAULT_REGION: "us-east-1" # Setting AWS region
12
13 cache:
14   paths:
15     - .terraform # Caching Terraform plugins and modules
16
17 before_script:
18   - terraform --version # Output Terraform version for visibility
19   - terraform init -backend-config="tfstate.config" -migrate-state # Initialize Terraform with backend configuration
20
21 stages:
22   - validate # Stage to validate Terraform configurations
23   - plan # Stage to generate Terraform execution plan
24   - apply # Stage to apply Terraform changes
25   - destroy # Stage to destroy Terraform resources
26
27 validate:
28   stage: validate
29   script:
30     - terraform validate # Validate Terraform configurations
31
32 plan:
33   stage: plan
34   script:
35     - terraform plan -out="planfile" # Generate Terraform execution plan and save to planfile
36   dependencies:
37     - validate # Dependency on 'validate' stage to run before 'plan'
38   artifacts:
39     paths:
40       - planfile # Publish 'planfile' as an artifact for later stages
41
42 apply:
43   stage: apply
44   script:
45     - terraform apply -input=false "planfile" # Apply Terraform changes using planfile
46   dependencies:
47     - plan # Dependency on 'plan' stage to run before 'apply'
48   when: manual # Manually trigger the 'apply' stage to execute
49
50 destroy:
51   stage: destroy
52   script:
53     - terraform destroy --auto-approve # Destroy Terraform resources automatically
54   when: manual # Manually trigger the 'destroy' stage to execute
55
```

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

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

CI/CD Variables </> 2		Reveal values	Add variable
Key ↑	Value	Environments	Actions
MY_AWS_ACCESS_KEY  Protected Expanded	***** 	All (default) 	 
MY_SECRET_KEY  Expanded	***** 	All (default) 	 

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

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

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

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

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

Sakeena Shaik / cicdtf / Pipelines / #1368918339

Update .gitlab-ci.yml file

Passed

Sakeena Shaik created pipeline for commit 11631325 1 minute ago, finished just now

For main

latest 40 4 jobs 1.39 1 minute 23 seconds, queued for 1 seconds

Pipeline

Needs

Jobs 4

Tests 0

validate

validate

plan

plan

apply

apply


destroy

destroy

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

1) Logs from validate stage:

validate

✓ Passed Started 13 minutes ago by  Sakeena Shaik

Search job log



```
1 Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
2   on green-2.saas-linux-small-amd64.runners-manager.gitlab.com/default ns46NMmJ, system ID: s_85d7af184313
3 ✓ Preparing the "docker+machine" executor 00:06
4   Using Docker executor with image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
5   Authenticating with credentials from job payload (GitLab Registry)
6   Pulling docker image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
7   Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcd92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/g
   itlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d656
   3a3e101052ba0f8f1f1f601d476611d7df1b ...
8 ✓ Preparing environment 00:01
9   Running on runner-ns46nmmj-project-59788171-concurrent-0 via runner-ns46nmmj-s-l-s-amd64-1720665055-7dc177a9...
10 ✓ Getting source from Git repository 00:01
11   Fetching changes with git depth set to 20...
12   Initialized empty Git repository in /builds/Sakeena19/cicdtf/.git/
13   Created fresh repository.
14   Checking out 11631325 as detached HEAD (ref is main)...
15   Skipping Git submodules setup
16   $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 ✓ Restoring cache 00:09
18   Checking cache for default-protected...
19   Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
20   Successfully extracted cache
21 ✓ Executing "step_script" stage of the job script 00:08
22   Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcd92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/g
   itlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d656
   3a3e101052ba0f8f1f1f601d476611d7df1b ...
```

Search job log



```
23 $ terraform --version
24 Terraform v1.4.2
25 on linux_amd64
26 + provider registry.terraform.io/hashicorp/aws v5.57.0
27 Your version of Terraform is out of date! The latest version
28 is 1.9.2. You can update by downloading from https://www.terraform.io/downloads.html
29 $ terraform init -backend-config="tfstate.config" -migrate-state
30 Initializing the backend...
31 Backend configuration changed!
32 Terraform has detected that the configuration specified for the backend
33 has changed. Terraform will now check for existing state in the backends.
34 Successfully configured the backend "s3"! Terraform will automatically
35 use this backend unless the backend configuration changes.
36 Initializing modules...
37 Initializing provider plugins...
38 - Reusing previous version of hashicorp/aws from the dependency lock file
39 - Using previously-installed hashicorp/aws v5.57.0
40 Terraform has been successfully initialized!
41 You may now begin working with Terraform. Try running "terraform plan" to see
42 any changes that are required for your infrastructure. All Terraform commands
43 should now work.
44 If you ever set or change modules or backend configuration for Terraform,
45 rerun this command to reinitialize your working directory. If you forget, other
46 commands will detect it and remind you to do so if necessary.
47 $ terraform validate
48 Success! The configuration is valid.
49 ✓ Saving cache for successful job 00:11
50   Creating cache default-protected...
51   .terraform: found 12 matching artifact files and directories
52   Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
53   Created cache
54 ✓ Cleaning up project directory and file based variables 00:01
55 Job succeeded
```

2) Logs from plan stage:

Sakeena Shaik / cicdtf / Jobs / #7313817355

plan

✓ Passed Started 15 minutes ago by Sakeena Shaik

Search job log

```
1 Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
2   on green-1.saas-linux-small-amd64.runners-manager.gitlab.com/default JlgUopmM, system ID: s_deaa2ca09de7
3 ✓ Preparing the "docker+machine" executor 00:06
4   Using Docker executor with image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
5   Authenticating with credentials from job payload (GitLab Registry)
6   Pulling docker image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
7   Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/g
   itlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d656
   3a3e101052ba0f8f1f1f601d476611d7df1b ...
8 ✓ Preparing environment 00:01
9   Running on runner-jlguopmm-project-59788171-concurrent-0 via runner-jlguopmm-s-l-s-amd64-1720665060-076fa23f...
10 ✓ Getting source from Git repository 00:01
11   Fetching changes with git depth set to 20...
12   Initialized empty Git repository in /builds/Sakeena19/cicdtf/.git/
13   Created fresh repository.
14   Checking out 11631325 as detached HEAD (ref is main)...
15   Skipping Git submodules setup
16   $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 ✓ Restoring cache 00:11
18   Checking cache for default-protected...
19   Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
20   Successfully extracted cache
21 ✓ Executing "step_script" stage of the job script 00:10
22   Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcdb92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/g
   itlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d656
   3a3e101052ba0f8f1f1f601d476611d7df1b ...
23 $ terraform --version
24 Terraform v1.4.2
25 on linux_amd64
26 + provider registry.terraform.io/hashicorp/aws v5.57.0
27 Your version of Terraform is out of date! The latest version
28 is 1.9.2. You can update by downloading from https://www.terraform.io/downloads.html
29 $ terraform init -backend-config="tfstate.config" -migrate-state
30 Initializing the backend...
31 Initializing modules...
32 Initializing provider plugins...
33 - Reusing previous version of hashicorp/aws from the dependency lock file
34 - Using previously-installed hashicorp/aws v5.57.0
35 Terraform has been successfully initialized!
36 You may now begin working with Terraform. Try running "terraform plan" to see
37 any changes that are required for your infrastructure. All Terraform commands
38 should now work.
39 If you ever set or change modules or backend configuration for Terraform,
40 rerun this command to reinitialize your working directory. If you forget, other
41 commands will detect it and remind you to do so if necessary.
42 $ terraform plan -out="planfile"
43 Terraform used the selected providers to generate the following execution
44 plan. Resource actions are indicated with the following symbols:
45   + create
46 Terraform will perform the following actions:
47   # module.ec2.aws_instance.server will be created
48   + resource "aws_instance" "server" {
49     + ami                               = "ami-04a81a99f5ec58529"
50     + arn                               = (known after apply)
51     + associate_public_ip_address      = (known after apply)
52     + availability_zone                = (known after apply)
53     + cpu_core_count                   = (known after apply)
54     + cpu_threads_per_core             = (known after apply)
55     + disable_api_stop                  = (known after apply)
```

```

179     + main_route_table_id           = (known after apply)
180     + owner_id                     = (known after apply)
181     + tags                         = {
182       + "Name" = "myvpc"
183     }
184     + tags_all                     = {
185       + "Name" = "myvpc"
186     }
187   }
188 Plan: 4 to add, 0 to change, 0 to destroy.
189
190 Saved the plan to: planfile
191 To perform exactly these actions, run the following command to apply:
192   terraform apply "planfile"
193 Saving cache for successful job
194 Creating cache default-protected...
195 .terraform: found 12 matching artifact files and directories
196 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
197 Created cache
198 Uploading artifacts for successful job
199 Uploading artifacts...
200 planfile: found 1 matching artifact files and directories
201 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/7313817355/artifacts?artifact_format=zip&artifact_type=archive new-url=https://gitlab.com
202 WARNING: Retrying... context=artifacts-uploader error=request redirected
203 Uploading artifacts as "archive" to coordinator... 201 Created id=7313817355 responseStatus=201 Created token=glcvt-66
204 Cleaning up project directory and file based variables
205 Job succeeded

```

3) Logs from apply:

Sakeena Shaik / cicdtf / Jobs / #7313817356

apply

✓ Passed
Started just now by Sakeena Shaik

1 Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
2 on green-2.saas-linux-small-amd64.runners-manager.gitlab.com/default ns46NMmJ, system ID: s_85d7af184313
3 Preparing the "docker+machine" executor
4 Using Docker executor with image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
5 Authenticating with credentials from job payload (GitLab Registry)
6 Pulling docker image registry.gitlab.com/gitlab-org/gitlab-build-images:terraform ...
7 Using docker image sha256:850ebc144b5d518ad6cd2f0e09ec653a6bdcd92f434ad19b571542d6ac3a61a for registry.gitlab.com/gitlab-org/g
itlab-build-images:terraform with digest registry.gitlab.com/gitlab-org/gitlab-build-images@sha256:a114d505c9b0422648307333d656
3a3e101052ba0f8f1f1f601d476611d7df1b ...
8 Preparing environment
9 Running on runner-ns46nmj-project-59788171-concurrent-0 via runner-ns46nmj-s-l-s-amd64-1720666239-82aa6311...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/Sakeena19/cicdtf/.git/
13 Created fresh repository.
14 Checking out 11631325 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 \$ git remote set-url origin "\${CI_REPOSITORY_URL}"
17 Restoring cache
18 Checking cache for default-protected...
19 Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
20 Successfully extracted cache
21 Downloading artifacts
22 Downloading artifacts for plan (7313817355)...
23 Downloading artifacts from coordinator... ok host=storage.googleapis.com id=7313817355 responseStatus=200 OK token=glcvt


```

35 Initializing provider plugins...
36 - Reusing previous version of hashicorp/aws from the dependency lock file
37 - Using previously-installed hashicorp/aws v5.57.0
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 apply -input=false "planfile"
46 module.vpc.aws_vpc.myvpc: Creating...
47 module.vpc.aws_vpc.myvpc: Still creating... [10s elapsed]
48 module.vpc.aws_vpc.myvpc: Creation complete after 12s [id=vpc-04505ae4c1fd43057]
49 module.vpc.aws_subnet.pb_sn: Creating...
50 module.vpc.aws_security_group.sg: Creating...
51 module.vpc.aws_security_group.sg: Creation complete after 2s [id=sg-0d7b8162c87df1b3e]
52 module.vpc.aws_subnet.pb_sn: Still creating... [10s elapsed]
53 module.vpc.aws_subnet.pb_sn: Creation complete after 11s [id=subnet-07c1e73bd369841d2]
54 module.ec2.aws_instance.server: Creating...
55 module.ec2.aws_instance.server: Still creating... [10s elapsed]
56 module.ec2.aws_instance.server: Still creating... [20s elapsed]
57 module.ec2.aws_instance.server: Still creating... [30s elapsed]
58 module.ec2.aws_instance.server: Creation complete after 32s [id=i-023a7418b3239ade7]
59 Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
60 Saving cache for successful job
61 Creating cache default-protected...
62 .terraform: found 12 matching artifact files and directories
63 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
64 Created cache
65 Cleaning up project directory and file based variables
66 Job succeeded

```

4) Logs from destroy:

Sakeena Shaik / cicdtf / Jobs / #7313817357

Search job log

```

220 - tags = {
221   - "Name" = "myvpc"
222 } -> null
223 - tags_all = {
224   - "Name" = "myvpc"
225 } -> null
226 }
227 Plan: 0 to add, 0 to change, 4 to destroy.
228 module.ec2.aws_instance.server: Destroying... [id=i-023a7418b3239ade7]
229 module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 10s elapsed]
230 module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 20s elapsed]
231 module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 30s elapsed]
232 module.ec2.aws_instance.server: Still destroying... [id=i-023a7418b3239ade7, 40s elapsed]
233 module.ec2.aws_instance.server: Destruction complete after 41s
234 module.vpc.aws_subnet.pb_sn: Destroying... [id=subnet-07c1e73bd369841d2]
235 module.vpc.aws_security_group.sg: Destroying... [id=sg-0d7b8162c87df1b3e]
236 module.vpc.aws_subnet.pb_sn: Destruction complete after 0s
237 module.vpc.aws_security_group.sg: Destruction complete after 0s
238 module.vpc.aws_vpc.myvpc: Destroying... [id=vpc-04505ae4c1fd43057]
239 module.vpc.aws_vpc.myvpc: Destruction complete after 1s
240 Destroy complete! Resources: 4 destroyed.
241 Saving cache for successful job
242 Creating cache default-protected...
243 .terraform: found 12 matching artifact files and directories
244 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59788171/default-protected
245 Created cache
246 Cleaning up project directory and file based variables
247 Job succeeded

```

All 4 states have been executed.

Sakeena Shaik / cidtf / Pipelines

All5FinishedBranchesTags

Clear runner cachesCI lintRun pipeline

Filter pipelines

Show Pipeline IID

Status	Pipeline	Created by	Stages
<div>Passed</div> <div>00:04:20</div> <div>3 hours ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#5main11631325latest</div>		<div></div>

Instances (2) Info

Find Instance by attribute or tag (case-sensitive)

Running

1

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input type="checkbox"/>	terraform_project	i-075cf72513ff0d349	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a
<input type="checkbox"/>	my_server	i-0618b03b5c63acd62	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a

Instance summary for i-0618b03b5c63acd62 (my_server) Info

Updated 1 minute ago

ConnectInstance stateActions

Instance ID

i-0618b03b5c63acd62 (my_server)

IPv6 address

-

Hostname type

IP name: ip-10-0-1-34.ec2.internal

Answer private resource DNS name

-

Auto-assigned IP address

54.235.24.32 [Public IP]

IAM Role

-

IMDSv2

Required

Public IPv4 address

54.235.24.32 | open address

Instance state

Running

Private IP DNS name (IPv4 only)

ip-10-0-1-34.ec2.internal

Instance type

t2.micro

VPC ID

vpc-0c49eb1d501192d52 (myvpc)

Subnet ID

subnet-001ede8f7e4c8fed3 (pb_sn1)

Instance ARN

arn:aws:ec2:us-east-1:339712799118:instance/i-0618b03b5c63acd62

Private IPv4 addresses

10.0.1.34

Public IPv4 DNS

ec2-54-235-24-32.compute-1.amazonaws.com | open address

Elastic IP addresses

-

AWS Compute Optimizer finding

Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name

-

DetailsStatus and alarmsMonitoringSecurityNetworkingStorageTags

▼ Security details

IAM Role

-

Security groups

sg-00269f3e89f269862 (my_sg)

Owner ID

339712799118

Launch time

Thu Jul 11 2024 09:02:22 GMT+0530 (India Standard Time)

Amazon S3 > Buckets > s3statefile786 > terraform/ > staging/

staging/

Copy S3 URI

ObjectsProperties

Objects (1) Info

Copy S3 URICopy URLDownloadOpenDeleteActionsCreate folderUpload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefixShow versions1

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	terraform.tfstate	tfstate	July 11, 2024, 09:02:54 (UTC+05:30)	10.0 KB	Standard

DynamoDB > Tables

Tables (1) Info

Find tables by table name

Any tag key

Any tag value

< 1 >

⚙

<input type="checkbox"/>	Name ▲	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mo...	To
<input type="checkbox"/>	state-lock	Active	LOCKID (S)	-	0	Off	On-demand	On-demand	0 b

Verify full logs in below text file:

Logs.txt

The pipeline performs the following steps:

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