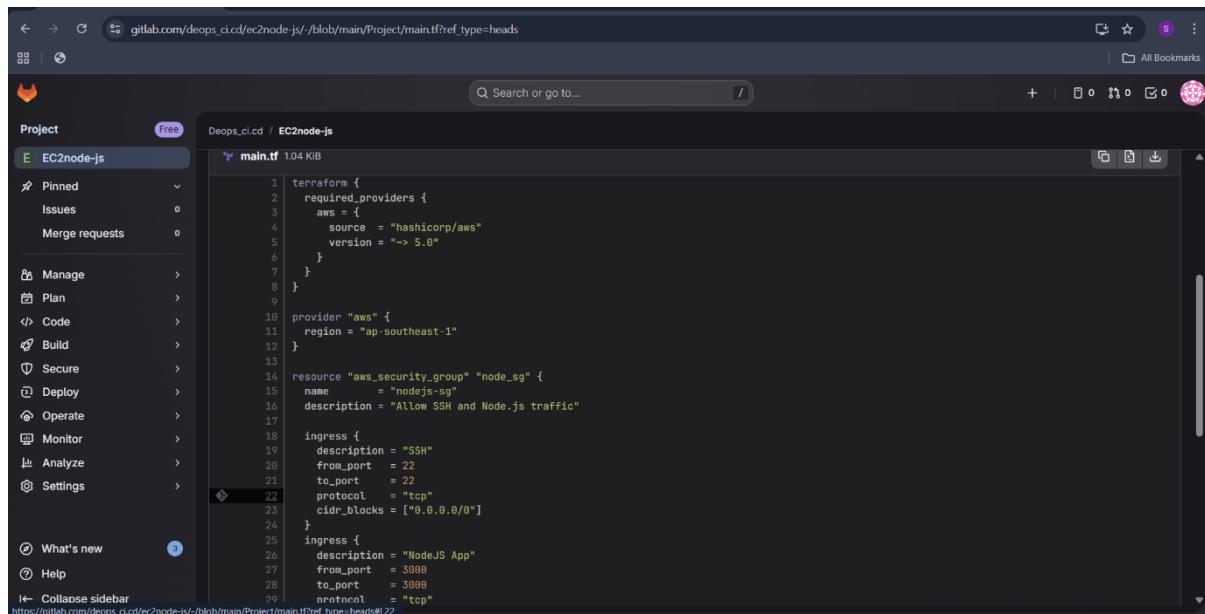


Weekly Test

Deploy the node js application in EC2 using terraform and gitlab ci/cd

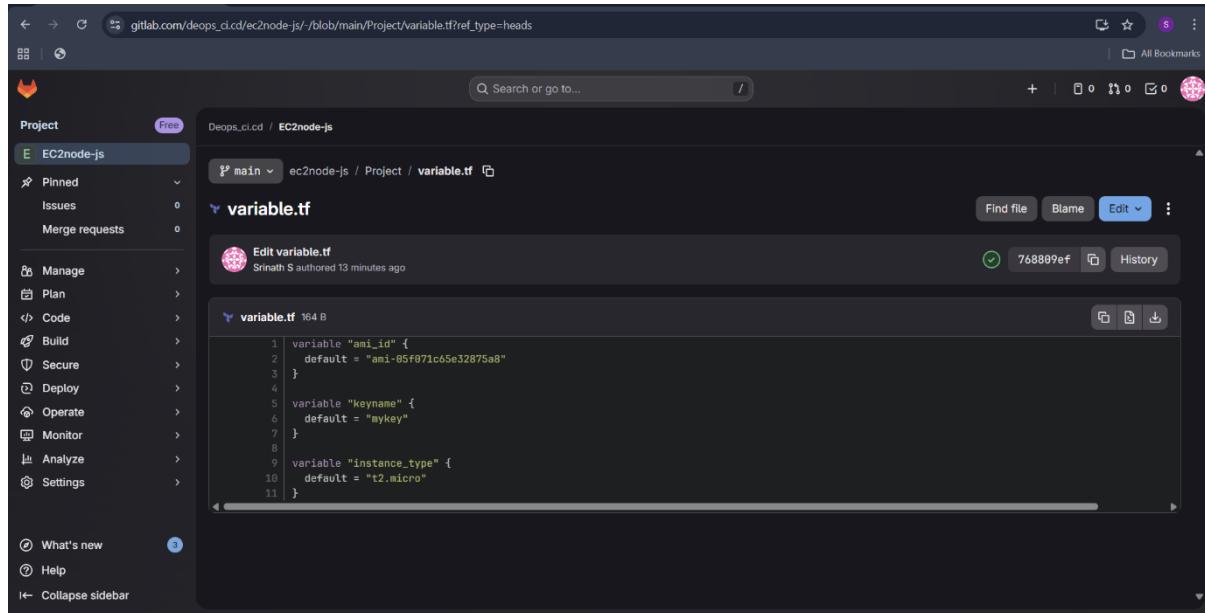
Create main.tf in Gitlab:



The screenshot shows a GitLab project named "Deops_ci_cd / EC2node-js". The "main.tf" file is displayed, containing Terraform configuration code. The code defines an AWS provider, creates an AWS Security Group (node_sg) with SSH and Node.js traffic ingress rules, and another security group for the NodeJS App with port 3000. The file is 1.04 KiB in size.

```
1 terraform {  
2   required_providers {  
3     aws = {  
4       source  = "hashicorp/aws"  
5       version = "~> 5.0"  
6     }  
7   }  
8 }  
9  
10 provider "aws" {  
11   region = "ap-southeast-1"  
12 }  
13  
14 resource "aws_security_group" "node_sg" {  
15   name            = "nodejs-sg"  
16   description     = "Allow SSH and Node.js traffic"  
17  
18   ingress {  
19     description = "SSH"  
20     from_port   = 22  
21     to_port     = 22  
22     protocol    = "tcp"  
23     cidr_blocks = ["0.0.0.0/0"]  
24   }  
25   ingress {  
26     description = "NodeJS App"  
27     from_port   = 3000  
28     to_port     = 3000  
29     protocol    = "tcp"  
30 }
```

Create Variable.tf file:



The screenshot shows a GitLab project named "Deops_ci_cd / EC2node-js". The "variable.tf" file is displayed, containing variable definitions for AMI ID, keyname, and instance type. The file is 164 B in size.

```
1 variable "ami_id" {  
2   default = "ami-05f071c65e32875a8"  
3 }  
4  
5 variable "keyname" {  
6   default = "mykey"  
7 }  
8  
9 variable "instance_type" {  
10  default = "t2.micro"  
11 }
```

Create output.tf file:

The screenshot shows the GitLab interface for a project named 'EC2node-js'. A modal window titled 'Using Terraform? Try the GitLab Managed Terraform State' is open, explaining the benefits of using GitLab's managed Terraform state backend. Below the modal, the 'output.tf' file is displayed in a code editor. The code contains a single line defining an output variable:

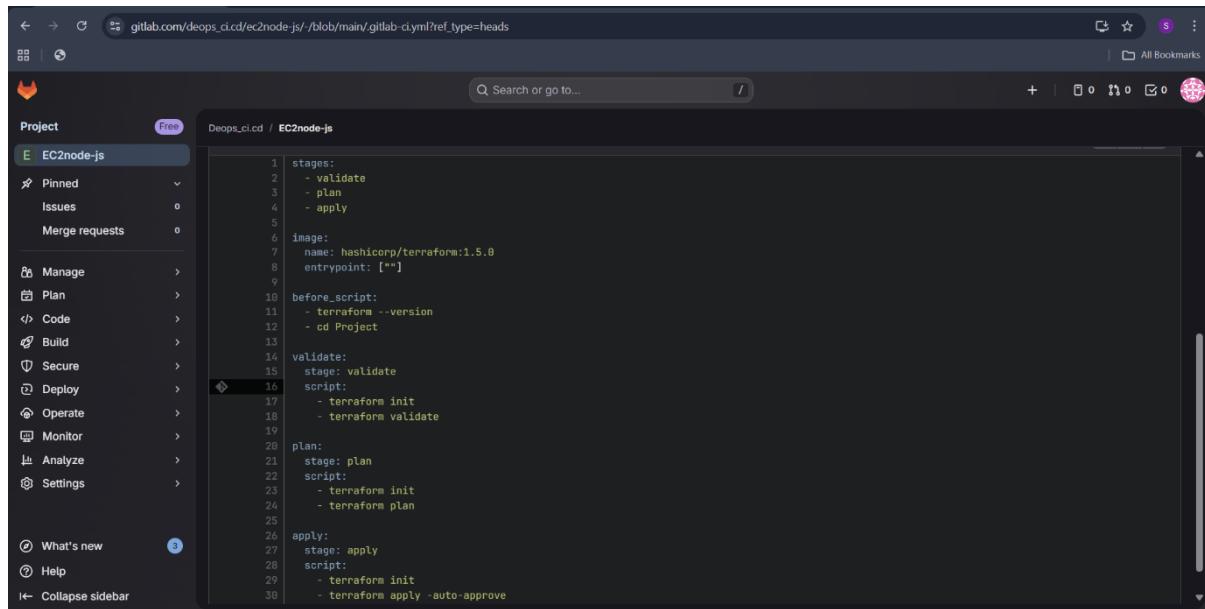
```
1 | output "ec2_public_ip" {  
2 |   value = aws_instance.node_app.public_ip  
3 | }  
4 |  
5 |
```

Create user data File:

The screenshot shows the GitLab interface for the same project 'EC2node-js'. A modal window titled 'Srinath S authored 7 minutes ago' is open, showing the commit details for a file named 'user_data.sh'. The file content is a shell script that installs Node.js and runs a Node.js application:

```
1 |#!/bin/bash  
2 |yum update -y  
3 |  
4 |# Install Node.js  
5 |curl -fsSL https://rpm.nodesource.com/setup_18.x | bash -  
6 |yum install -y nodejs git  
7 |  
8 |# Create app directory  
9 |mkdir -p /home/ec2-user/app  
10 |cd /home/ec2-user/app  
11 |  
12 |# Create Node.js app  
13 |cat <<EOF > index.js  
14 |const http = require('http');  
15 |  
16 |const server = http.createServer((req, res) => {  
17 |  res.end('Node.js File Running in EC2 using Gitlab and Terraform');  
18 |});  
19 |  
20 |server.listen(3000);  
21 |EOF  
22 |  
23 |# Run app in background  
24 |nohup node index.js > app.log 2>&1 &
```

Create .gitlab-ci.yml file:



The screenshot shows a GitLab project interface for a repository named 'Deops_ci_cd / EC2node-js'. The sidebar on the left contains various project management links like Pinned, Issues, Merge requests, Manage, Plan, Code, Build, Secure, Deploy, Operate, Monitor, Analyze, and Settings. The main content area displays the .gitlab-ci.yml file with the following code:

```
stages:
  - validate
  - plan
  - apply

image:
  name: hashicorp/terraform:1.5.0
  entrypoint: [""]

before_script:
  - terraform --version
  - cd Project

validate:
  stage: validate
  script:
    - terraform init
    - terraform validate

plan:
  stage: plan
  script:
    - terraform init
    - terraform plan

apply:
  stage: apply
  script:
    - terraform init
    - terraform apply --auto-approve
```

Code:

stages:

- validate
- plan
- apply

image:

```
name: hashicorp/terraform:1.5.0
entrypoint: [""]
```

before_script:

- terraform --version
- cd Project

validate:

stage: validate

script:

- terraform init

- terraform validate

plan:

stage: plan

script:

- terraform init

- terraform plan

apply:

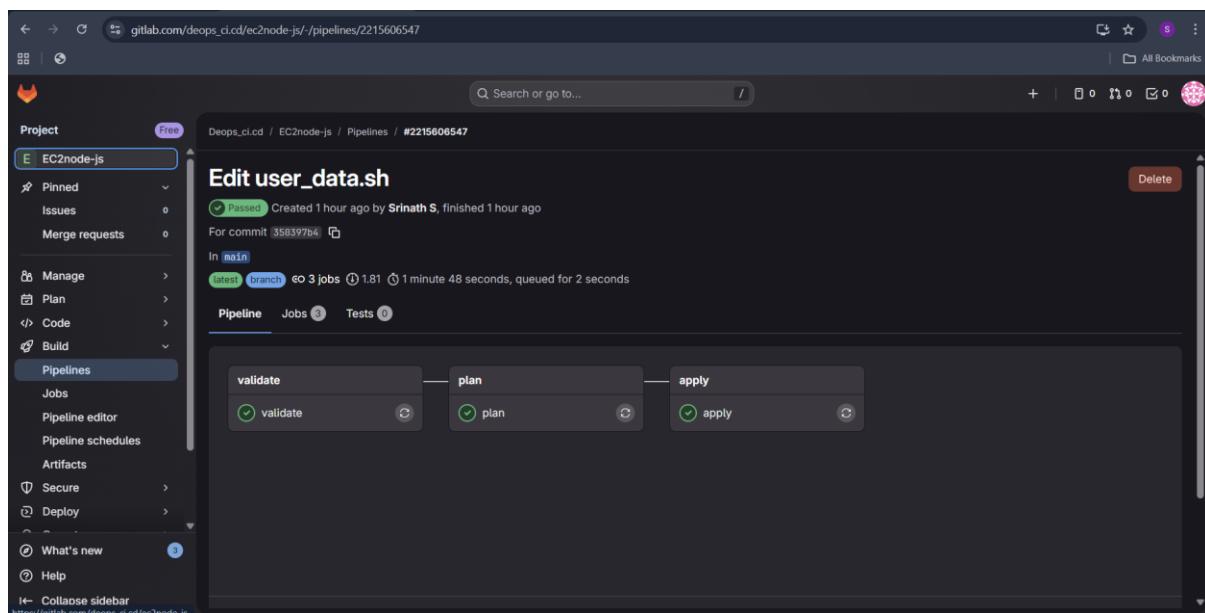
stage: apply

script:

- terraform init

- terraform apply -auto-approve

CI/CD pipe line:



Output:

