Terraform backend local to terraform cloud migration

A backend defines where Terraform stores its state data files.

Available Backends

By default, Terraform uses a backend called local, which stores state as a local file on disk.

To store your terraform backend in a remote location, you can use Terraform Cloud.

Steps:

1. Go to https://www.terraform.io/
2. Click on Try Terraform Cloud button.
3. Create an terraform cloud account

There are 3 ways to do it:

1. CLI-driven Run Workflow

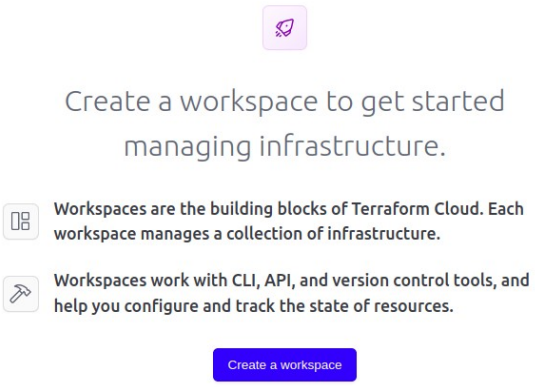
2. API-driven Run Workflow

3. UI and VCS-driven Run Workflow

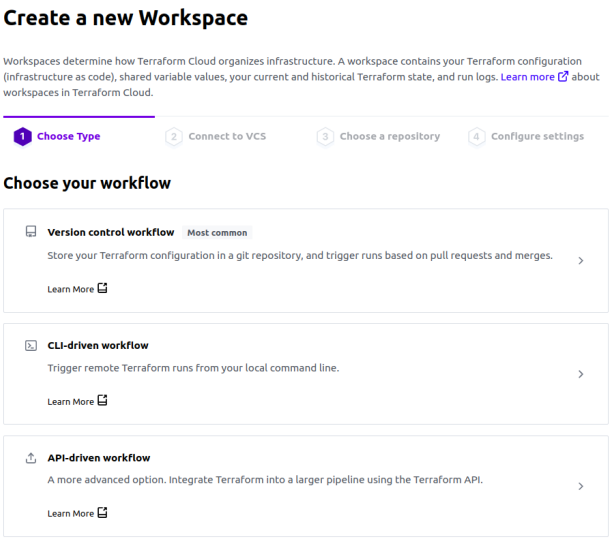
CLI-driven Run Workflow

STEPS:-

1. Create a workspace

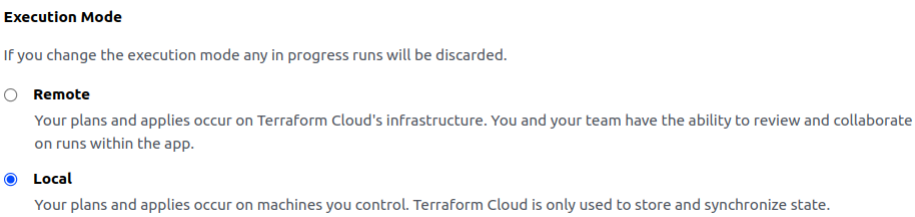


2. Choose workspace



3. Choose CLI-driven Workflow, give "Workspace Name" and "Description" and Create workspace.

4. Go to Workspace Settings -> General -> Execution Mode, select Local option so that terraform plan, apply and destroy can run from CLI only.



5. In your CLI, run “terraform login” command to login into your terraform cloud account, enter “yes”. It will redirect to Terraform cloud site in your default browser to create an API token.

6. Now enter the created token in your CLI, now you will be successfully logged-in into your terraform cloud account, and your token would be stored locally in “~/.terraform.d/credentials.tfrc.json” file

7. After creating the workspace, you will get a code of block to integrate your local terraform code with terraform cloud workspace, copy-paste this block in your main.tf file.

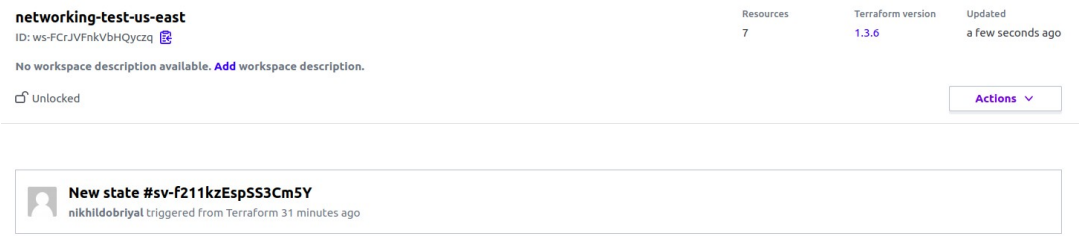
```
terraform {
  cloud {
    organization = "nikhil-org-001"

    workspaces {
      name = "networking-test-us-east"
    }
  }
}
```

8. Run “terraform init” command to initialize modules, initialize remote backend and initialize provider plugins.

```
admin1@system1:~/terraform-cloud/cli-driven/playing-with-terraform$ terraform init
Initializing modules...
- ec2 in module/ec2
- sg in module/sg
- vpc in module/vpc

Initializing Terraform Cloud...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v4.46.0...
- Installed hashicorp/aws v4.46.0 (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!
```

9. Now, run “terraform plan” to dry run your code, “terraform apply” command to apply your terraform code and enter “yes” to confirm.

After completion of your apply, you can go to your workspace in terraform cloud site and check your terraform backend there.



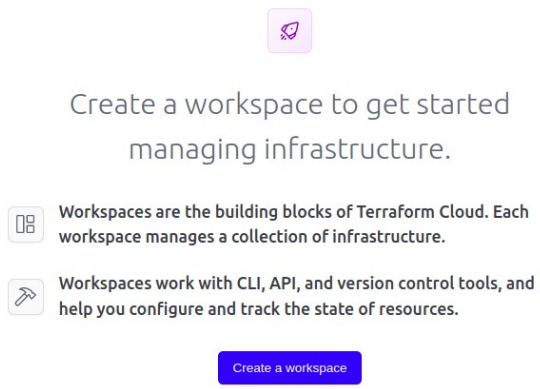
Now, you can see your state is stored there in your workspace, number of resources are 7 currently.

10. Just like you have applied your code from CLI, you can destroy your terraform infra by running “terraform destroy” command and enter “yes” for confirmation.

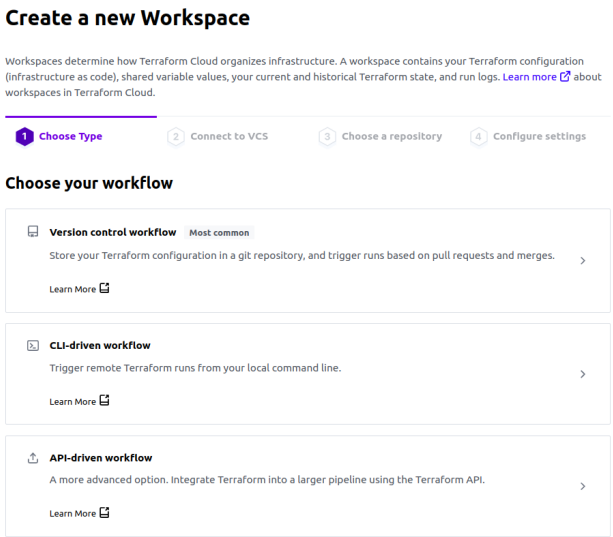
API-driven Run Workflow

STEPS:-

1. Create a workspace

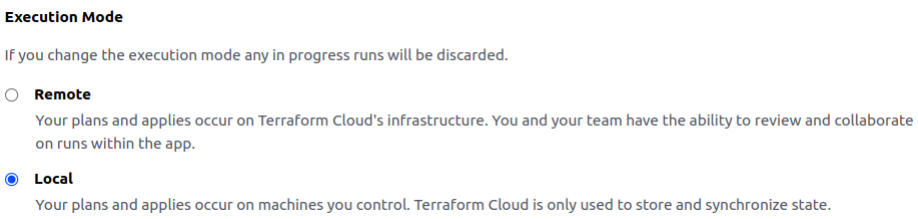


2. Choose your workflow.



3. Choose API-driven Workflow, give "Workspace Name" and "Description" and Create workspace.

4. Go to Workspace Settings -> General -> Execution Mode, select Local option so that terraform plan, apply and destroy can run from your CI/CD pipeline.



5. Write your CI/CD pipeline for terraform init, apply and destroy your AWS infra.

Here, we are using GitHub Actions for our CI/CD pipeline, create a directory .github/workflows in your Terraform code's root and create a <file-name>.yaml file where you will have to write CI/CD pipeline.

First, give a name for workflow

```
name: Create AWS Infra
```

Now, give the event when this pipeline will trigger.

```
on:
  push:
    branches: [master]
```

Now, define the pipeline's job name, runner(where the pipeline will runs on) and environment variables for your pipeline(which user has to define in your GitHub repo's secrets).

```
jobs:
  job1:
    runs-on: ubuntu-latest
    env:
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      TF_API_TOKEN: ${{secrets.TF_API_TOKEN}}
```

Now, define steps of your job in pipeline

```
    steps:
      - name: Checkout Codebase
        uses: actions/checkout@v2

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          # terraform_version: 0.13.0
          cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}
```

The first step is for checking out the codebase(which is terraform code of our repo) and the second step would sets up Terraform CLI in your GitHub Actions workflow

```
      - name: Terraform Initialize
        run: terraform init

      - name: Terraform apply
        run: terraform apply -auto-approve
```

Third step is for running “terraform init” command in runner(which will initialize modules, initialize remote backend and initialize provider plugins) and the fourth step is for terraform apply command(which will deploy infra through CI/CD).

.github > workflows > ! pipeline.yaml

```yaml
name: Create AWS Infra
on:
  push:
    branches: [master]
jobs:
  job1:
    runs-on: ubuntu-latest
    env:
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      TF_API_TOKEN: ${{secrets.TF_API_TOKEN}}
    steps:
      - name: Checkout Codebase
        uses: actions/checkout@v2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          # terraform_version: 0.13.0
          cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}
      - name: Terraform Initialize
        run: terraform init
      - name: Terraform apply
        run: terraform apply -auto-approve
```

6. After creating the workspace, you will get a code of block to integrate your local terraform code with terraform cloud workspace, copy-paste this block in your main.tf file.

```
terraform {
  cloud {
    organization = "nikhil-org-001"

    workspaces {
      name = "networking-test-us-east"
    }
  }
}
```

7. Now, go to Workspace Settings -> Version Control

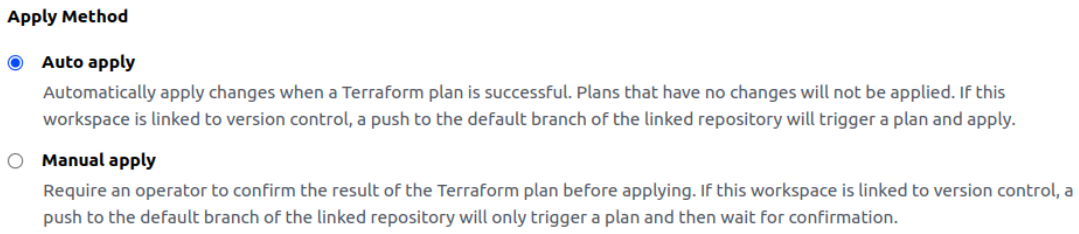
Version Control

Not Connected

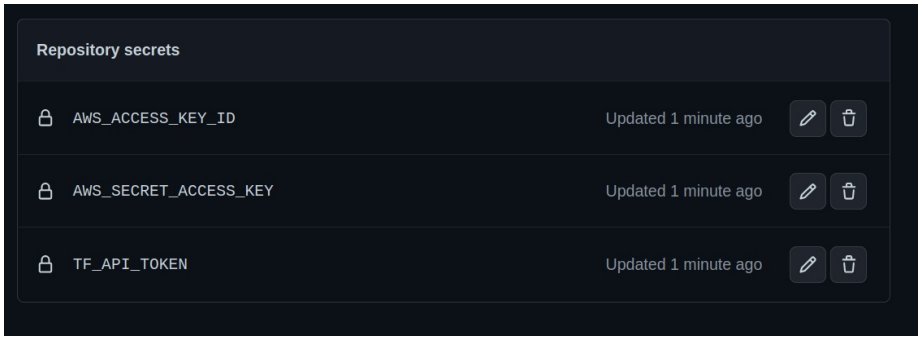
Connecting the workspace to Version Control will enable automatic runs on git commits, automatic plan-only runs on pull requests and status updates. Read more about connecting VCS providers to Terraform Cloud.

Connect to version control

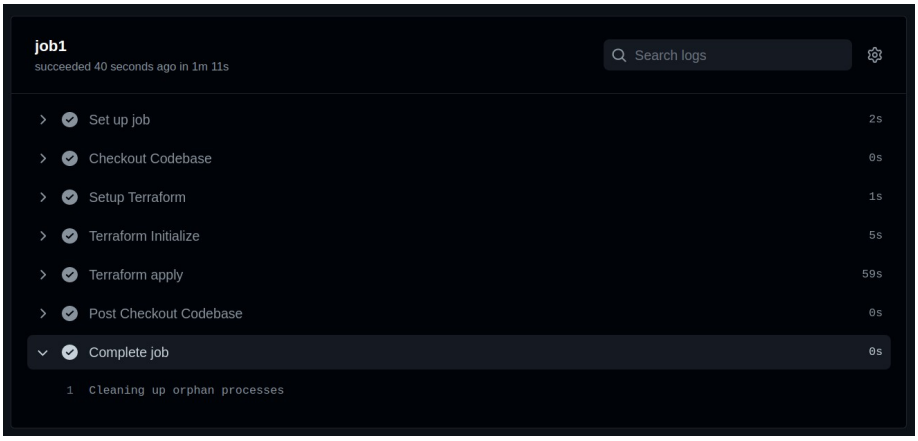
And click on Connect to version control, choose Version Control -> click on GitHub, select your repository and then select “Auto apply” in apply method and save the settings.



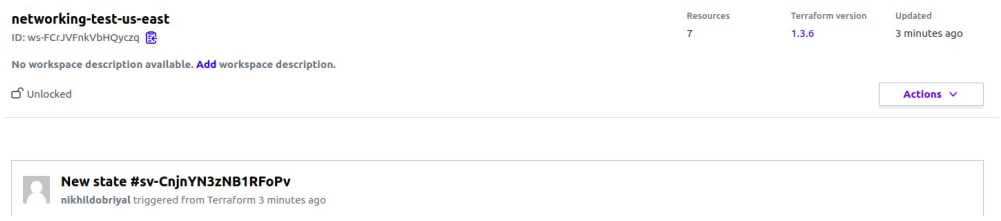
8. Define your AWS account credentials and terraform account token in your repo’s secrets.



9. Push your code into your master branch, which will trigger your pipeline.



As soon as you will push your code, your pipeline will trigger and jobs will be scheduled. Thus, your infra will be deployed and state will be stored in terraform cloud workspace.



Now, you can see your state is stored there in your workspace, number of resources are 7 currently.

9. Just like you have applied your code from CI/CD, you can destroy your terraform infra by creating another pipeline for destroying the infra which will trigger on manually dispatch the workflow.

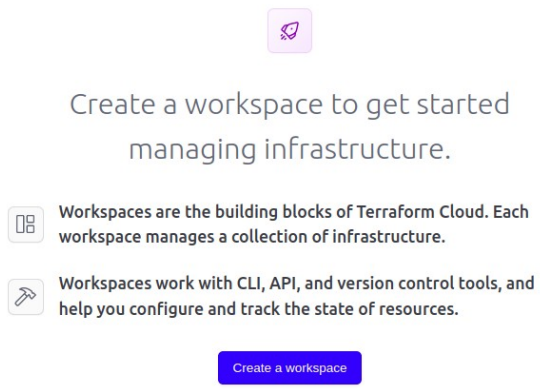
.github > workflows > ! destroy.yaml

```yaml
name: Destroy AWS Infra
on:
  workflow_dispatch:
jobs:
  job1:
    runs-on: ubuntu-latest
    env:
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      TF_API_TOKEN: ${{secrets.TF_API_TOKEN}}
    steps:
      - name: Checkout Codebase
        uses: actions/checkout@v2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          # terraform_version: 0.13.0
          cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}
      - name: Terraform Initialize
        run: terraform init
      - name: Terraform destroy
        run: terraform destroy -auto-approve
```

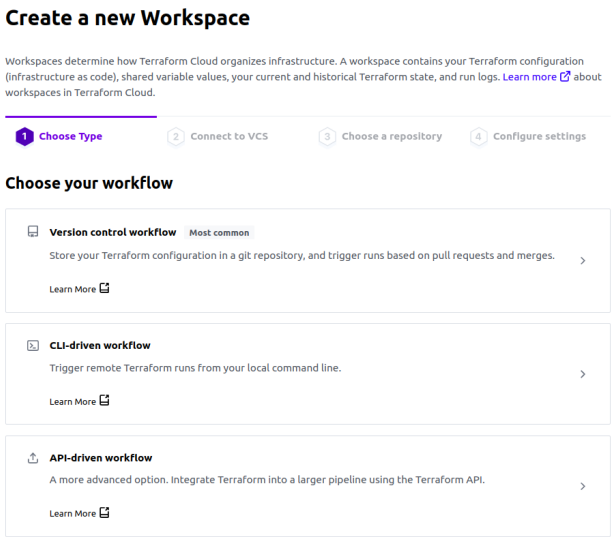
UI and VCS-driven Run Workflow

STEPS:-

1. Create a workspace

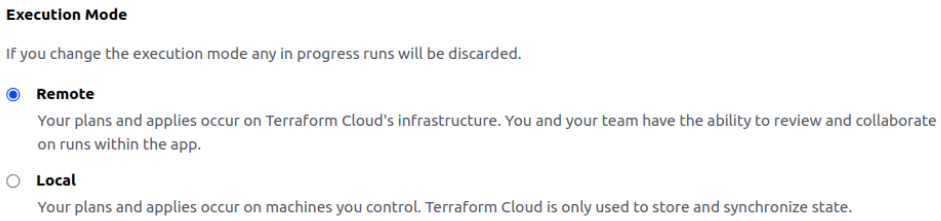


2. Choose your workflow.



3. Choose Version control workflow, give "Workspace Name" and "Description" and Create workspace.

4. Go to Workspace Settings -> General -> Execution Mode, select Remote option so that terraform plan, apply and destroy can run from your terraform cloud account only.

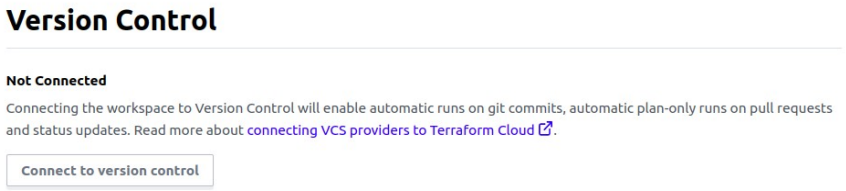


5. After creating the workspace, you will get a code of block to integrate your local terraform code with terraform cloud workspace, copy-paste this block in your main.tf file.

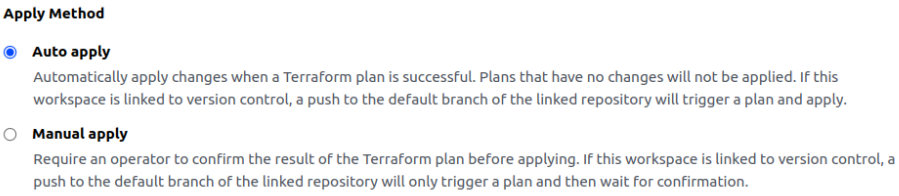
```
terraform {
  cloud {
    organization = "nikhil-org-001"

    workspaces {
      name = "networking-test-us-east"
    }
  }
}
```

6. Now, push your code to a Github repository.

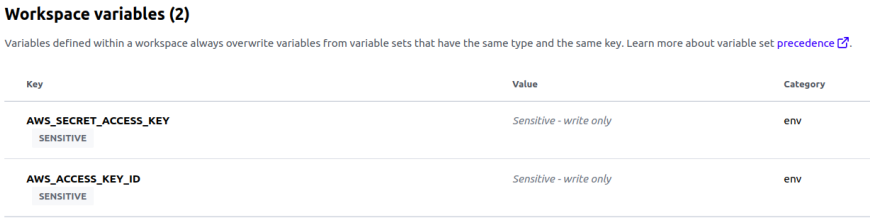
7. Configure your terraform workspace, go to Workspace Settings -> Version Control



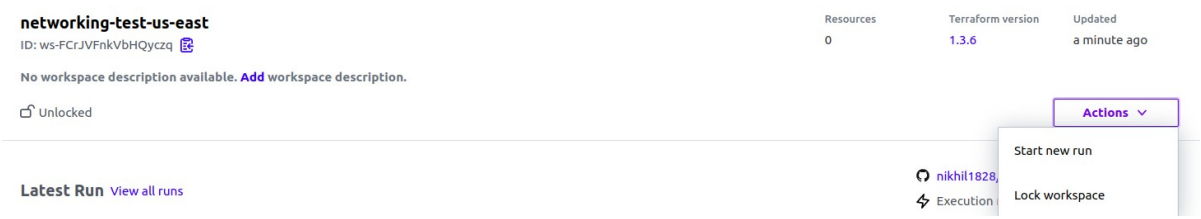
And click on Connect to version control, choose Version Control -> click on GitHub, select your repository and then select "Auto apply" in apply method and save the settings.



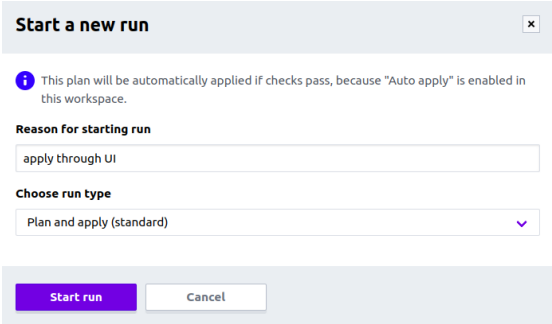
8. Define your AWS account credentials in your workspace's Variables section.

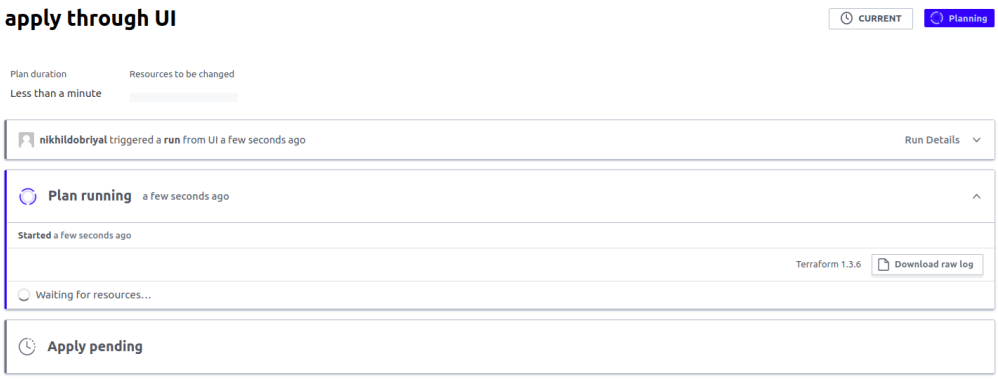


9. Go to home page of your workspace, click on "Actions" button and click on "Start new run"

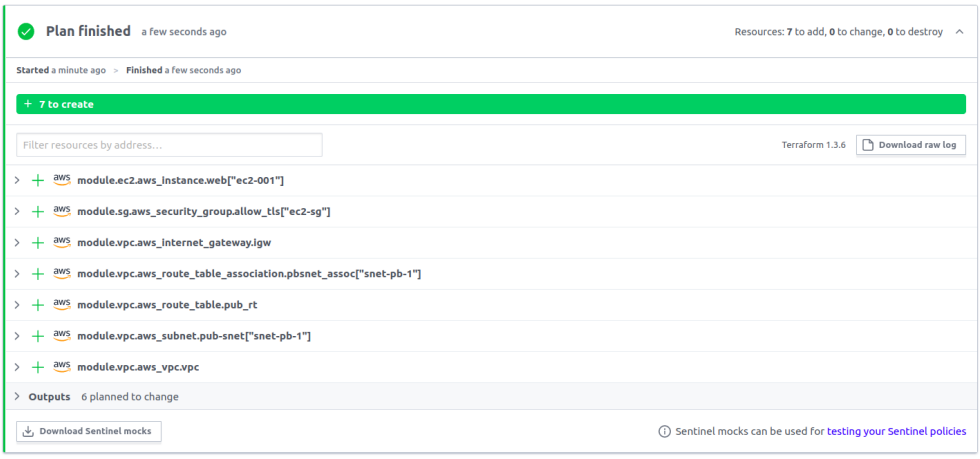


10. Give a reason to start your run and choose run type “Plan and apply (standard)” and click on “Start run” button.

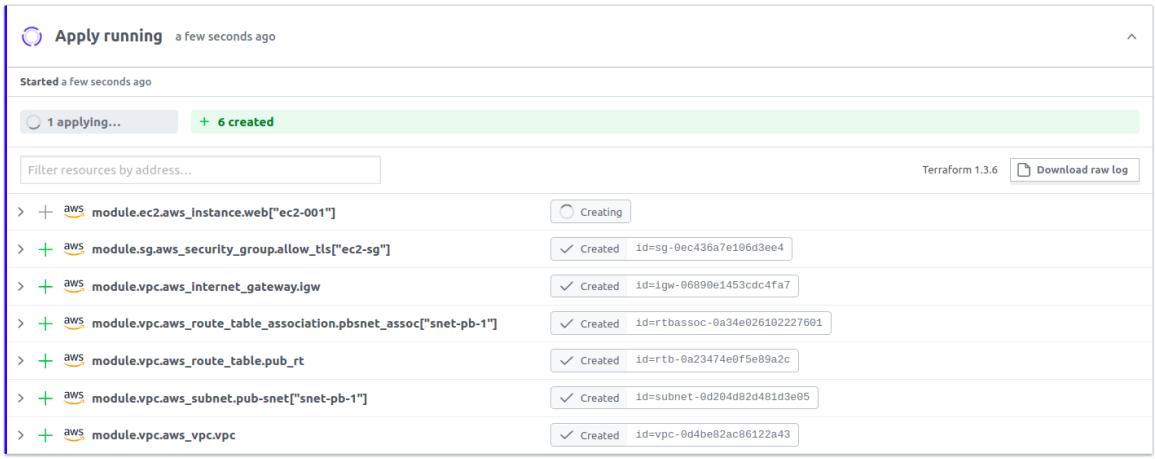




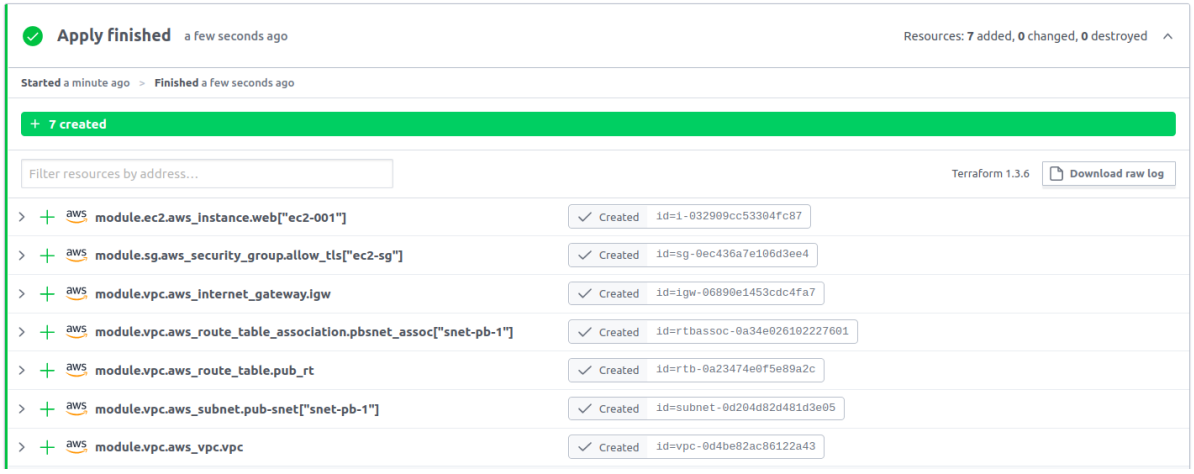
Plan is running



Plan is finished

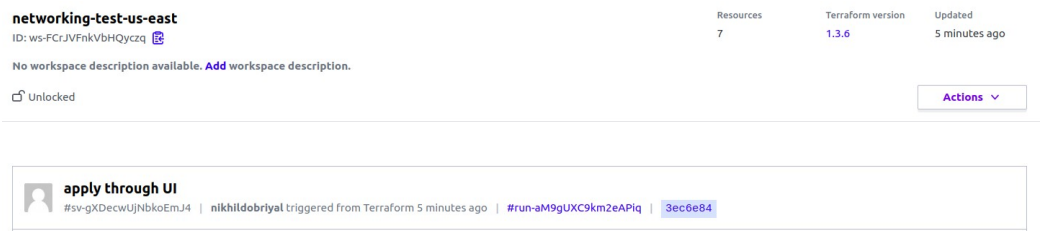


Apply is running

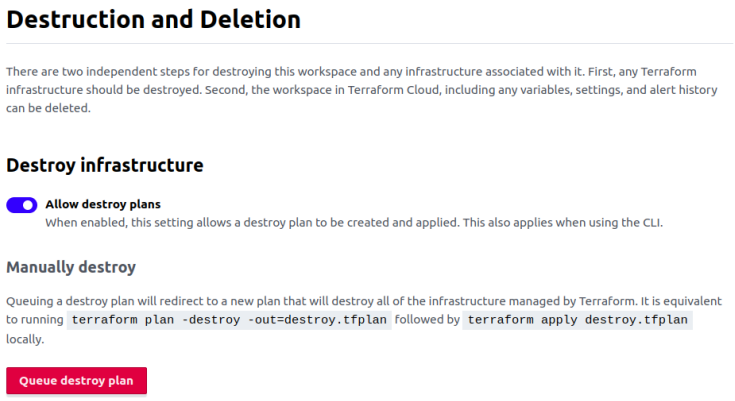


Apply is finished

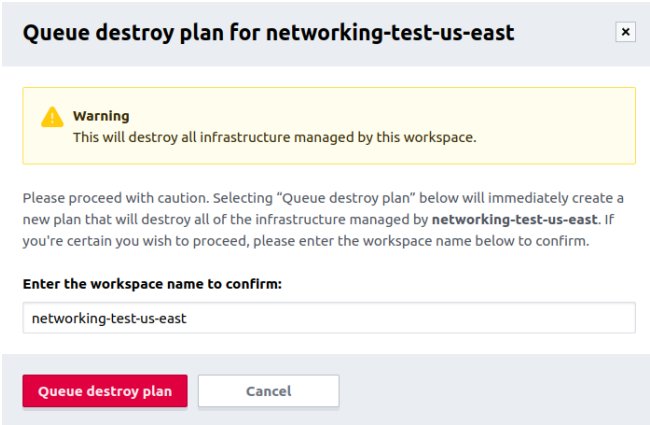
Now, you can see your state is stored there in your workspace, number of resources are **7** currently.

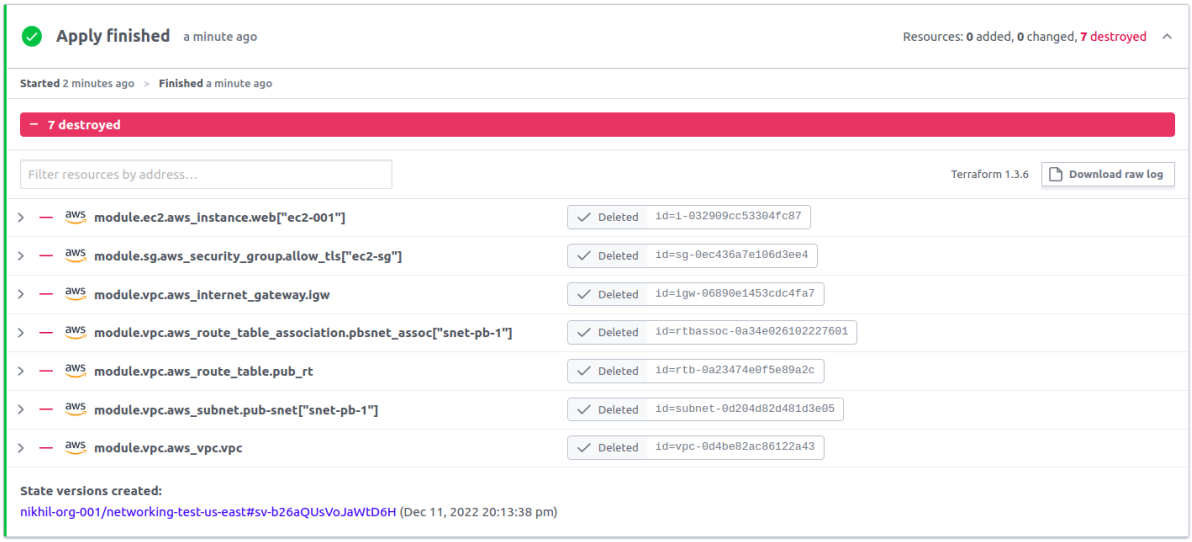


11. Through UI, you can also delete your infra, Go to workspace settings -> Destruction and Deletion, click on “Queue destroy plan”.



12. Give workspace name for confirmation and destroy the infra.





Destroy done