TASK 3: Infrastructure as Code (IaC) with Terraform

Objective: The goal of this task is to use Terraform to define and provision a Dockerized NGINX container locally on a Windows machine using Infrastructure as Code (IaC) principles.

What is this project about?

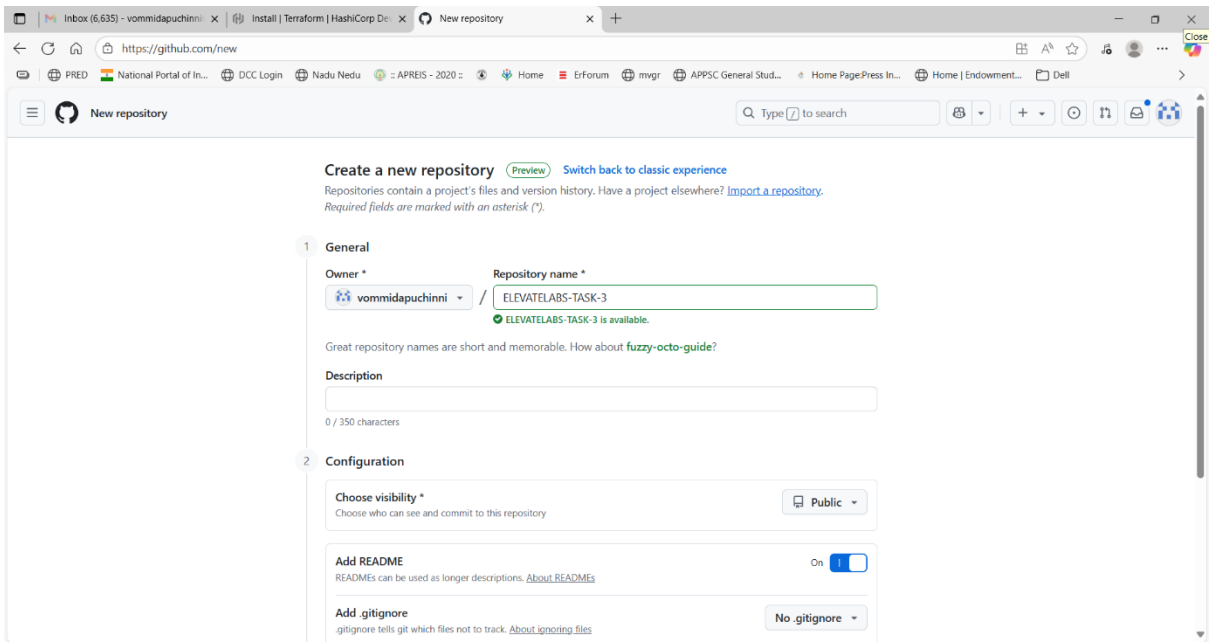
- Using Terraform to automate Docker container deployment.
- Pulling the latest NGINX image and running it on port 8080.
- Demonstrating the power of IaC by avoiding manual Docker CLI commands.

Tools & Technologies Used

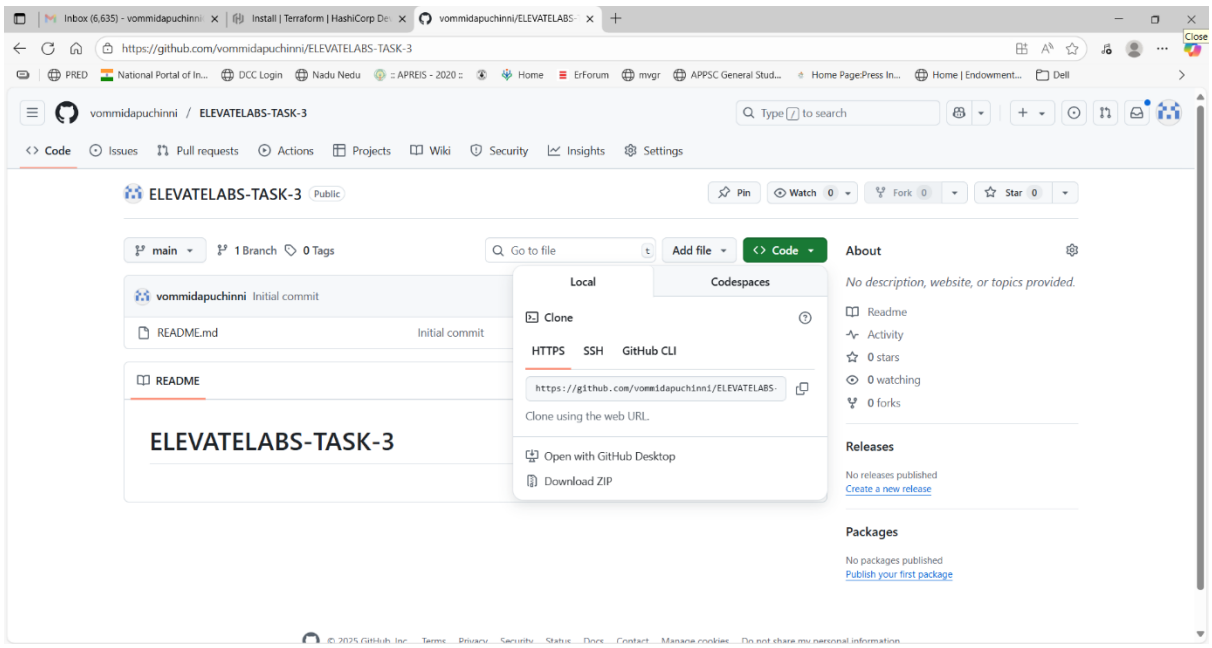
Tool	Description
Terraform	Open-source IaC tool for provisioning infrastructure
Docker	Container runtime used to run applications in isolated environmen
NGINX	Lightweight web server used for testing container setup

Create git repository:

open GitHub click on new repo, create new repo for this task

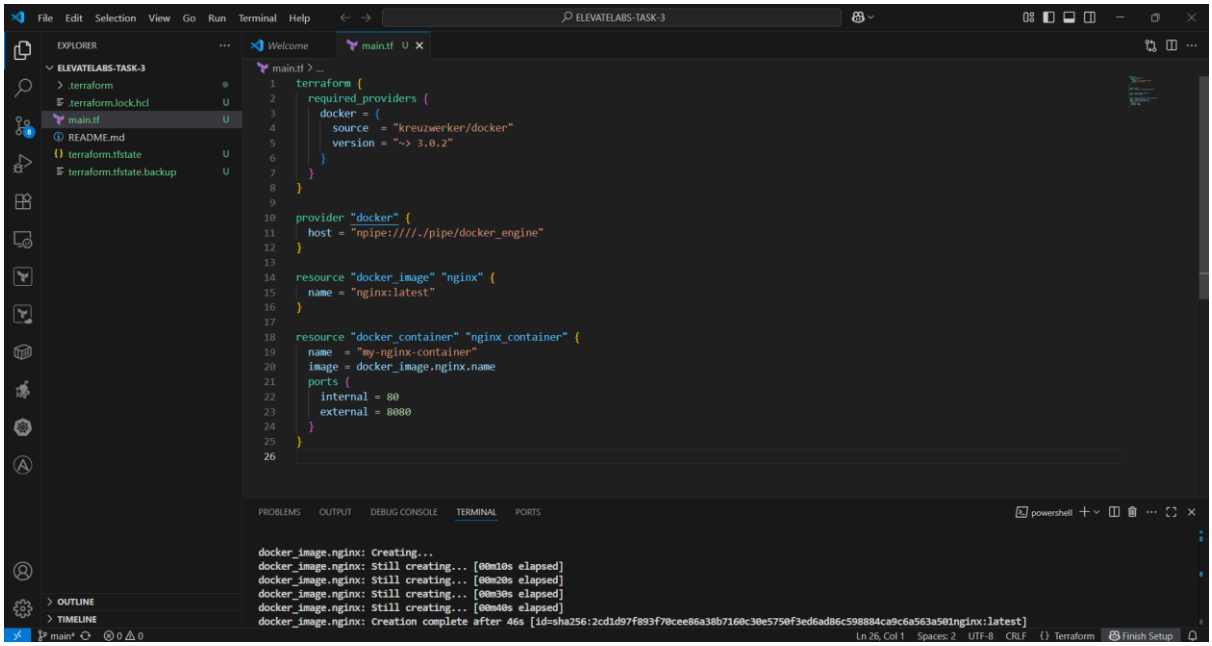


Clone the repo to vscode



Initially for doing this task we have to install docker desktop and terraform locally.

Terraform Configuration File (main.tf)




```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
```

```
      version = "~> 3.0.2"
    }
  }
}
provider "docker" {
  host = "npipe:////./pipe/docker_engine"
}
resource "docker_image" "nginx" {
  name = "nginx:latest"
}
resource "docker_container" "nginx_container" {
  name  = "my-nginx-container"
  image = docker_image.nginx.name
  ports {
    internal = 80
    external = 8080
  }
}
```

Line-by-Line Code Explanation

terraform block:

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.2"
    }
  }
```

```
}
```

- **Declares the required provider**: We need the docker provider.
- **Source**: "kreuzwerker/docker" is the official Docker provider for Terraform.
- **Version**: ~> 3.0.2 allows compatible versions like 3.0.x but not 3.1.

provider block:

```
provider "docker" {
  host = "npipe:////./pipe/docker_engine"
    }
```

- Tells Terraform to use **Windows Named Pipes** to connect to the Docker daemon.
- This is specific to **Docker Desktop on Windows**.

Docker Image Resource:

```
resource "docker_image" "nginx" {
  name = "nginx:latest"
}
```

- This tells Terraform to **pull the latest NGINX image** from Docker Hub.
- It is a prerequisite for running the container.

Docker Container Resource:

```
resource "docker_container" "nginx_container" {
  name  = "my-nginx-container"
  image = docker_image.nginx.name
  ports {
    internal = 80
    external = 8080
  }
```

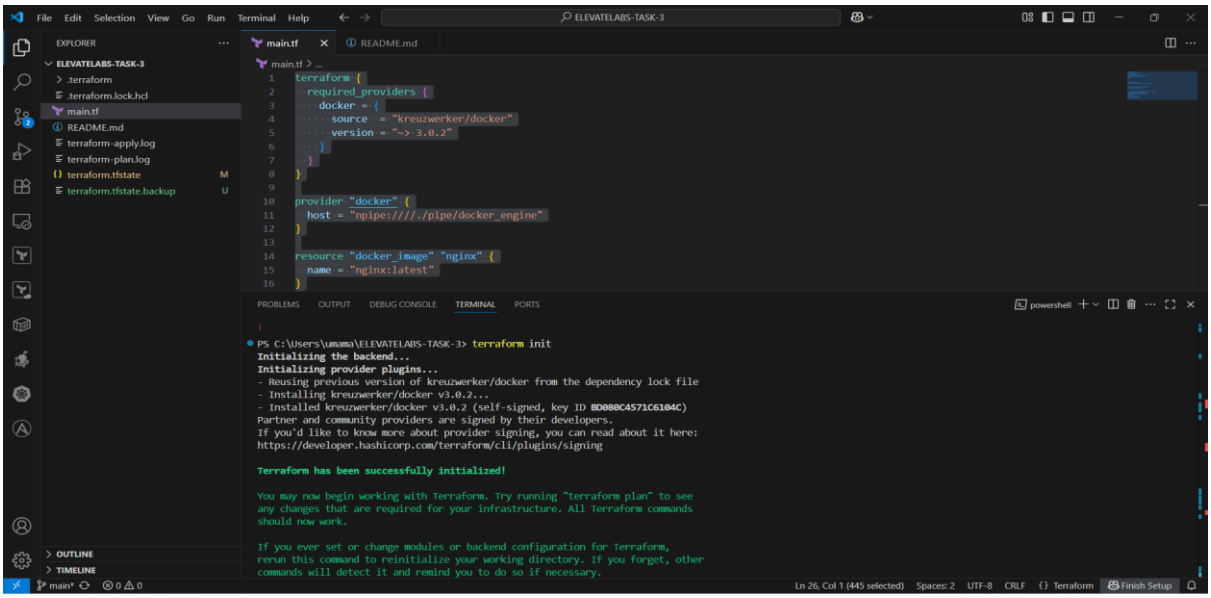
}

- **Container Name**: my-nginx-container
- **Image**: Refers to the image pulled in the previous step.
- **Ports**: Maps local port 8080 to container port 80 so you can access it at http://localhost:8080.

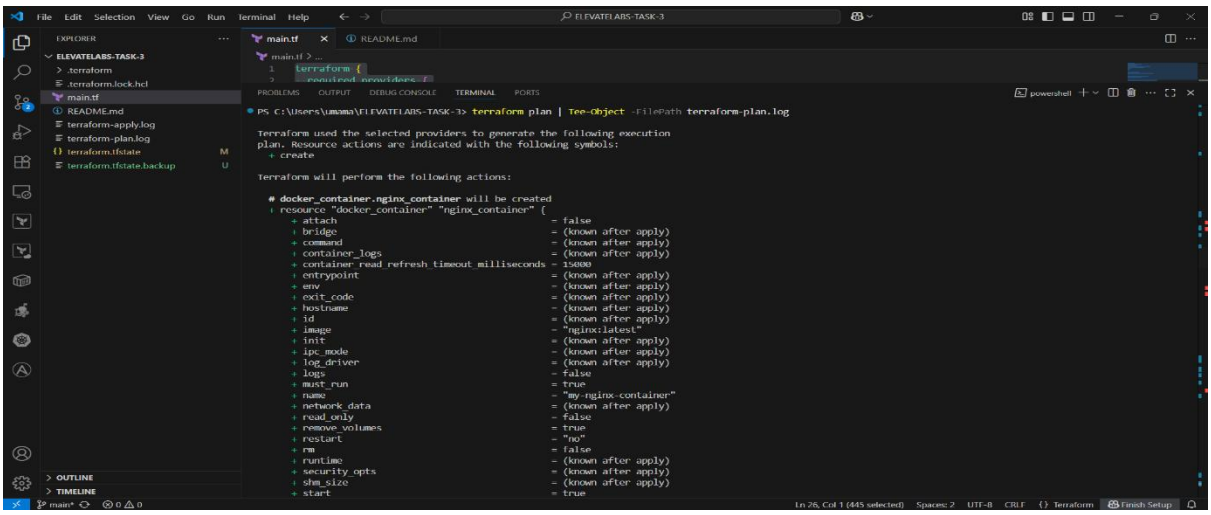
Execution Steps

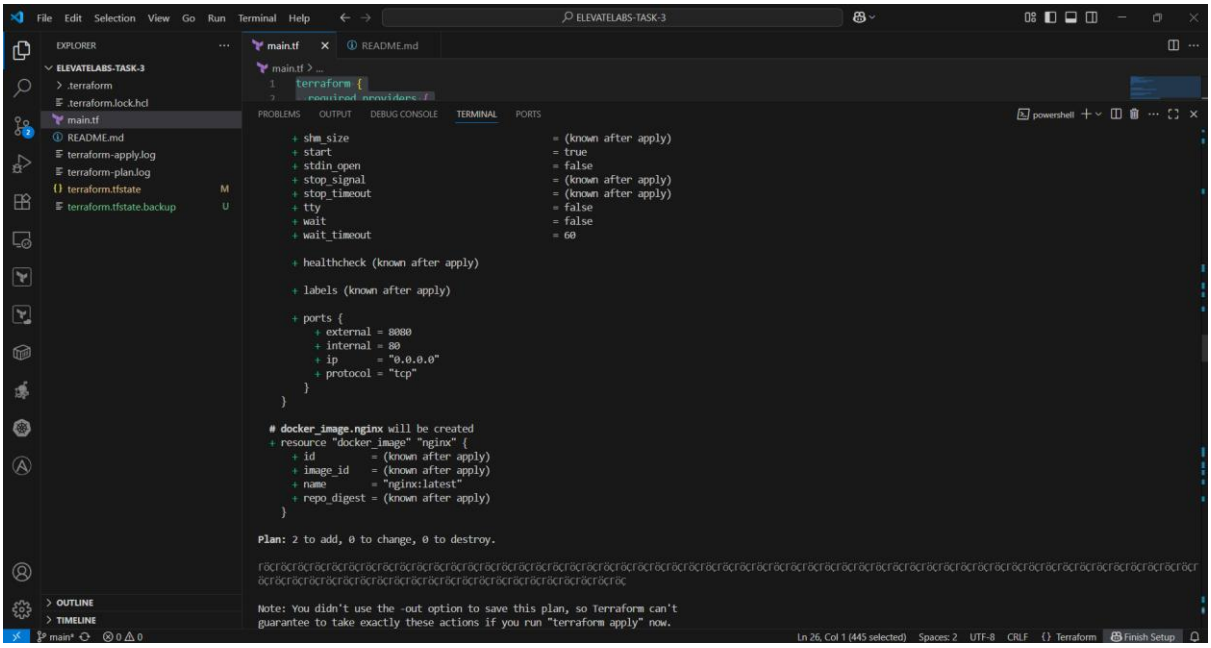
step-by-step Terraform Workflow:

Step 1 – Initialize
Run terraform init to download the Docker provider plugin.

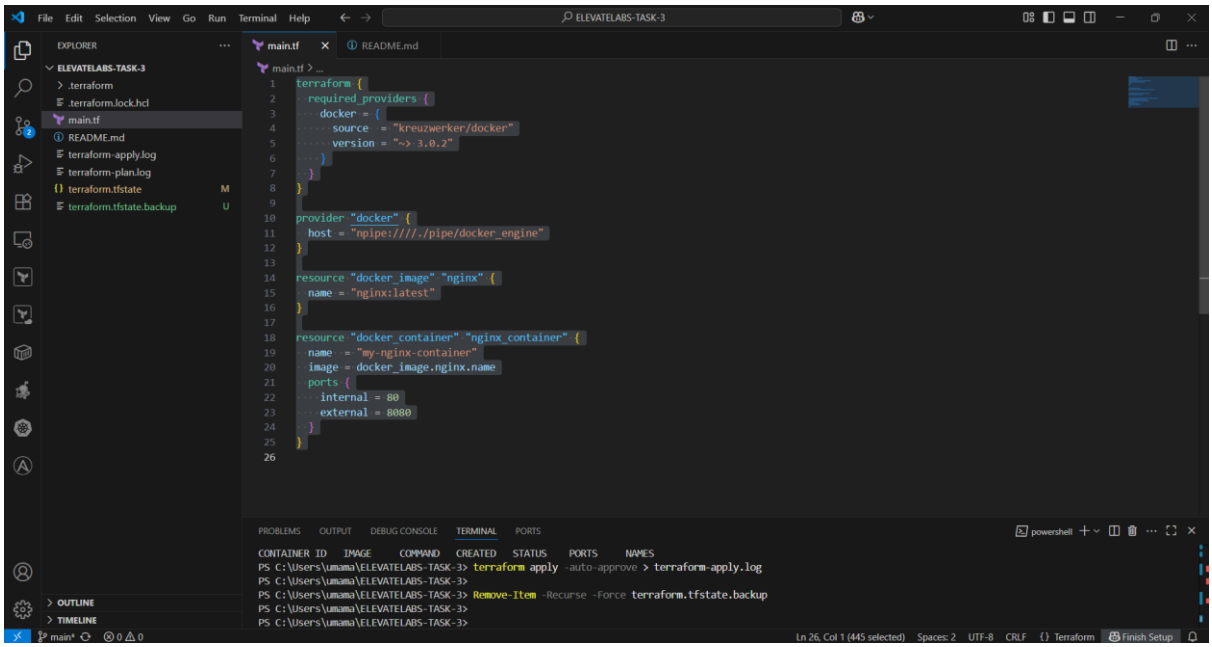


Step 2 – Preview Plan
Run terraform plan > plan.log to see what resources will be created.





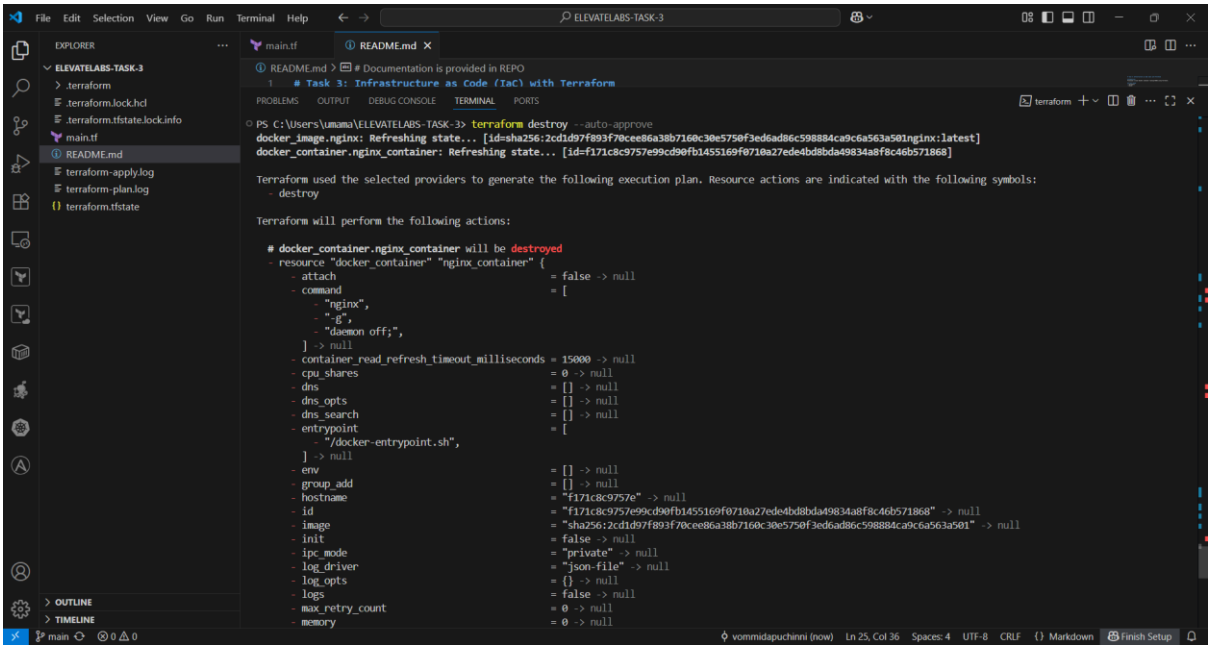
Step 3 – Apply Infrastructure
Run terraform apply -auto-approve to create the Docker image and container.

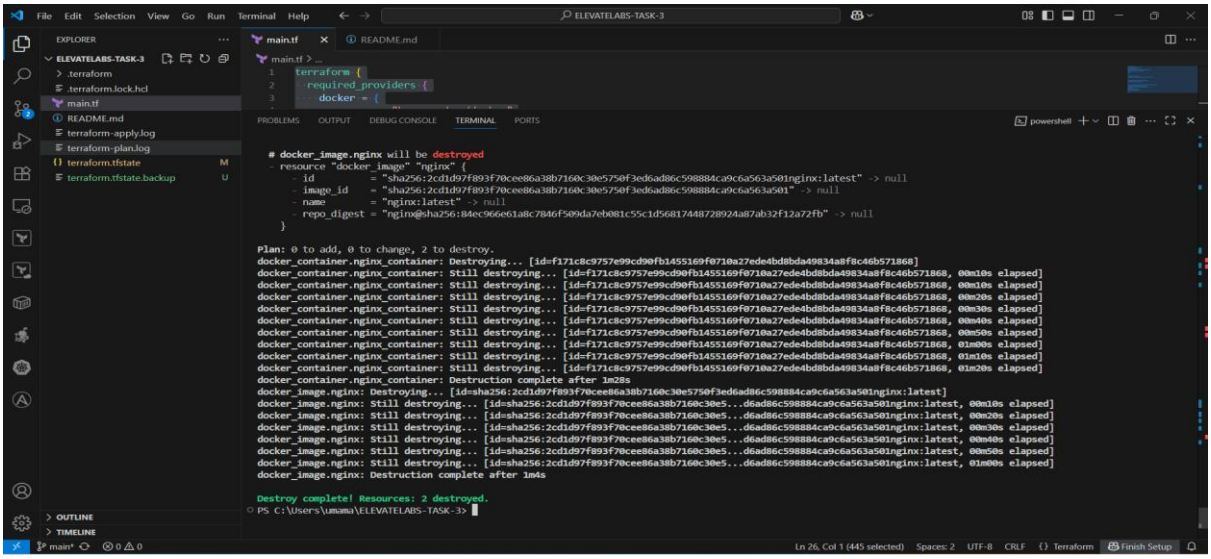


Step 4 – Verify Container
Run docker ps to check if the NGINX container is running.

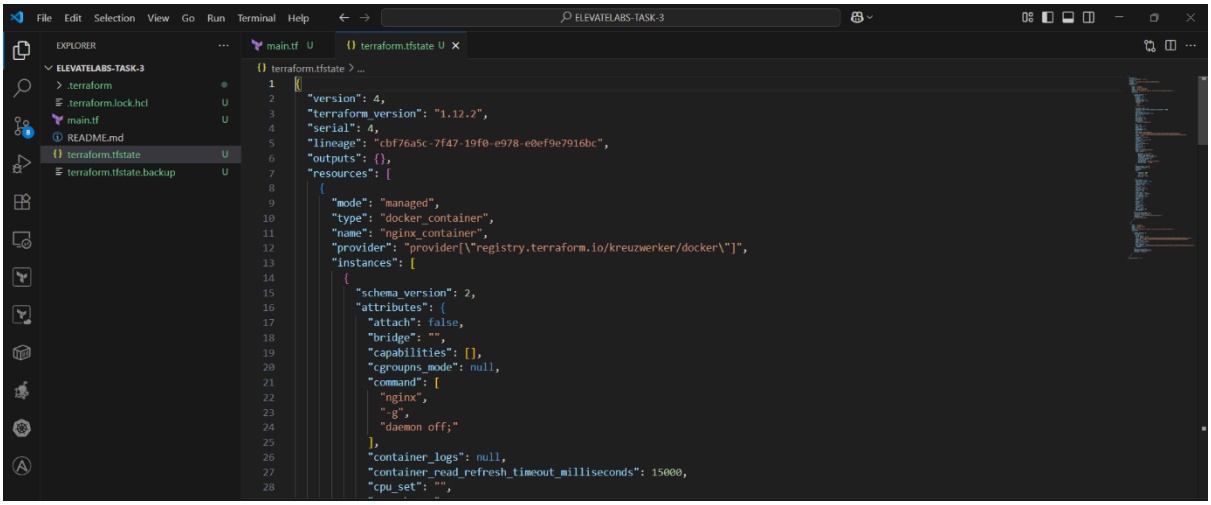
Step 5 – Access NGINX
Open your browser and go to http://localhost:8080 — you should see the NGINX welcome page.

Step 6 – Destroy Infrastructure
Run terraform destroy -auto-approve to remove the image and container.


File Edit Selection View Go Run Terminal Help
ELEVATELABS-TASK-3
EXPLORER
ELEVATELABS-TASK-3
.terraform
.terraform.lock.hcl
.terraform.tfstate.lock.info
main.tf
README.md
terraform-apply.log
terraform-plan.log
terraform.tfstate
main.tf
README.md
README.md > # Documentation is provided in REPO
Task 3: Infrastructure as Code (IaC) with Terraform
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
terraform
PS C:\Users\umama\ELEVATELABS-TASK-3> terraform destroy --auto-approve
docker_image.nginx: Refreshing state... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c598884ca9c6a563a501nginx:latest]
docker_container.nginx_container: Refreshing state... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868]
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:
# docker_container.nginx_container will be destroyed
- resource "docker_container" "nginx_container" {
- attach = false -> null
- command = [
"nginx",
"-g",
"daemon off;",
] -> null
- container_read_refresh_timeout_milliseconds = 15000 -> null
- cpu_shares = 0 -> null
- dns = [] -> null
- dns_opts = [] -> null
- dns_search = [] -> null
- entrypoint = [
"/docker-entrypoint.sh",
] -> null
- env = [] -> null
- group_add = [] -> null
- hostname = "f171c8c9757e" -> null
- id = "f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868" -> null
- image = "sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c598884ca9c6a563a501" -> null
- init = false -> null
- ipc_mode = "private" -> null
- log_driver = "json-file" -> null
- log_opts = {} -> null
- logs = false -> null
- max_retry_count = 0 -> null
- memory = 0 -> null
OUTLINE
TIMELINE
main
vommidapuchinni (now) Ln 25, Col 36 Spaces: 4 UTF-8 CRLF {} Markdown Finish Setup

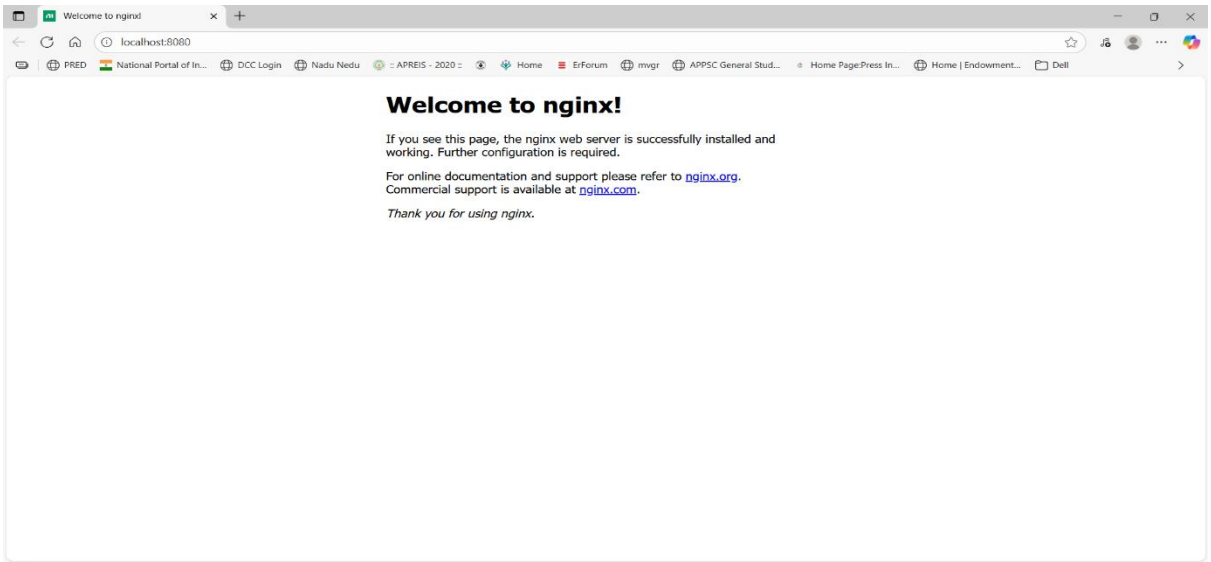

File Edit Selection View Go Run Terminal Help
ELEVATELABS-TASK-3
EXPLORER
ELEVATELABS-TASK-3
.terraform
.terraform.lock.hcl
main.tf
README.md
terraform-apply.log
terraform-plan.log
terraform.tfstate M
terraform.tfstate.backup U
main.tf
README.md
main.tf > ...
terraform {
required_providers {
docker = {
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
# docker_image.nginx will be destroyed
- resource "docker_image" "nginx" {
- id = "sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c598884ca9c6a563a501nginx:latest" -> null
- image_id = "sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c598884ca9c6a563a501" -> null
- name = "nginx:latest" -> null
- repo_digest = "nginx@sha256:84ec966e61a8c7846f509da7eb081c55c1d568174487289 24a87ab32f12a72fb" -> null
}
Plan: 0 to add, 0 to change, 2 to destroy.
docker_container.nginx_container: Destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 00m10s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 00m20s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 00m30s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 00m40s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 00m50s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 01m00s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 01m10s elapsed]
docker_container.nginx_container: Still destroying... [id=f171c8c9757e99cd90fb1455169f0710a27ede4bd8bda49834a8f8c46b571868, 01m20s elapsed]
docker_container.nginx_container: Destruction complete after 1m28s
docker_image.nginx: Destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c598884ca9c6a563a501nginx:latest]
docker_image.nginx: Still destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5...d6ad86c598884ca9c6a563a501nginx:latest, 00m10s elapsed]
docker_image.nginx: Still destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5...d6ad86c598884ca9c6a563a501nginx:latest, 00m20s elapsed]
docker_image.nginx: Still destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5...d6ad86c598884ca9c6a563a501nginx:latest, 00m30s elapsed]
docker_image.nginx: Still destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5...d6ad86c598884ca9c6a563a501nginx:latest, 00m40s elapsed]
docker_image.nginx: Still destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5...d6ad86c598884ca9c6a563a501nginx:latest, 00m50s elapsed]
docker_image.nginx: Still destroying... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5...d6ad86c598884ca9c6a563a501nginx:latest, 01m00s elapsed]
docker_image.nginx: Destruction complete after 1m4s
Destroy complete! Resources: 2 destroyed.
PS C:\Users\umama\ELEVATELABS-TASK-3>
OUTLINE
TIMELINE
main*
Ln 26, Col 1 (445 selected) Spaces: 2 UTF-8 CRLF {} Terraform Finish Setup

State file

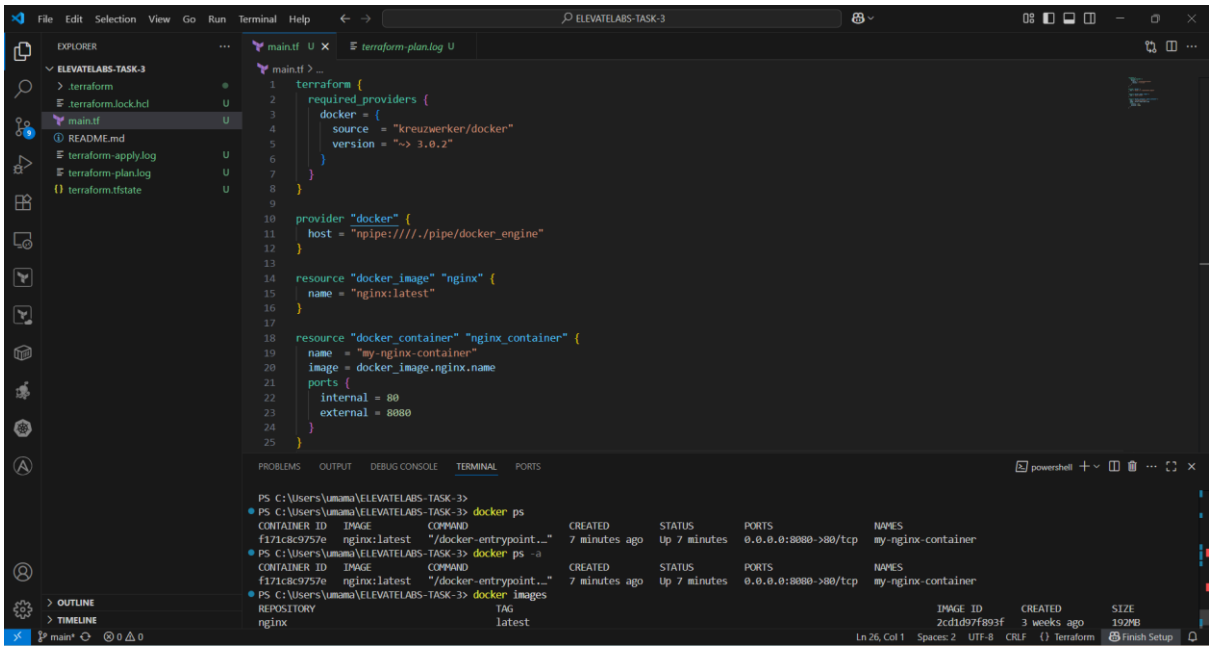

File Edit Selection View Go Run Terminal Help
ELEVATELABS-TASK-3
EXPLORER
ELEVATELABS-TASK-3
.terraform
.terraform.lock.hcl U
main.tf U
README.md
terraform.tfstate U
terraform.tfstate.backup U
main.tf U
terraform.tfstate U
terraform.tfstate > ...
{
"version": 4,
"terraform_version": "1.12.2",
"serial": 4,
"lineage": "cbf76a5c-7f47-19f0-e978-e0ef9e7916bc",
"outputs": {},
"resources": [
{
"mode": "managed",
"type": "docker_container",
"name": "nginx_container",
"provider": "provider[\"registry.terraform.io/kreuzwerker/docker\"]",
"instances": [
{
"schema_version": 2,
"attributes": {
"attach": false,
"bridge": "",
"capabilities": [],
"cgroupns_mode": null,
"command": [
"nginx",
"-g",
"daemon off;"
],
"container_logs": null,
"container_read_refresh_timeout_milliseconds": 15000,
"cpu_set": "",

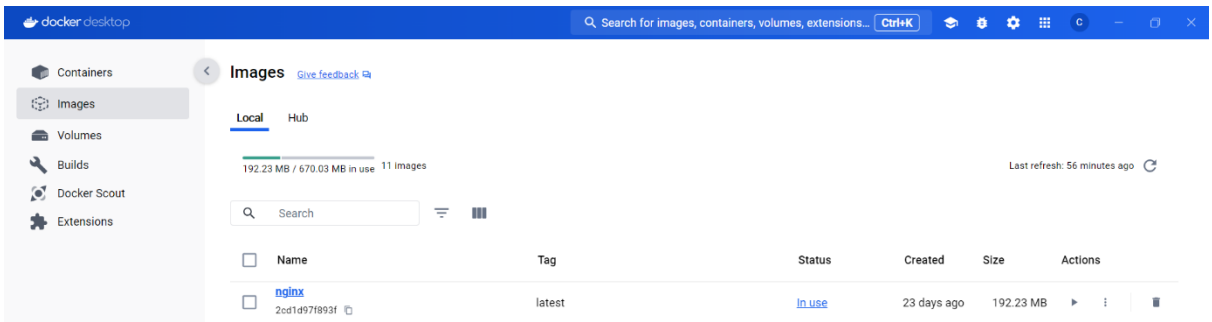
Final Output:

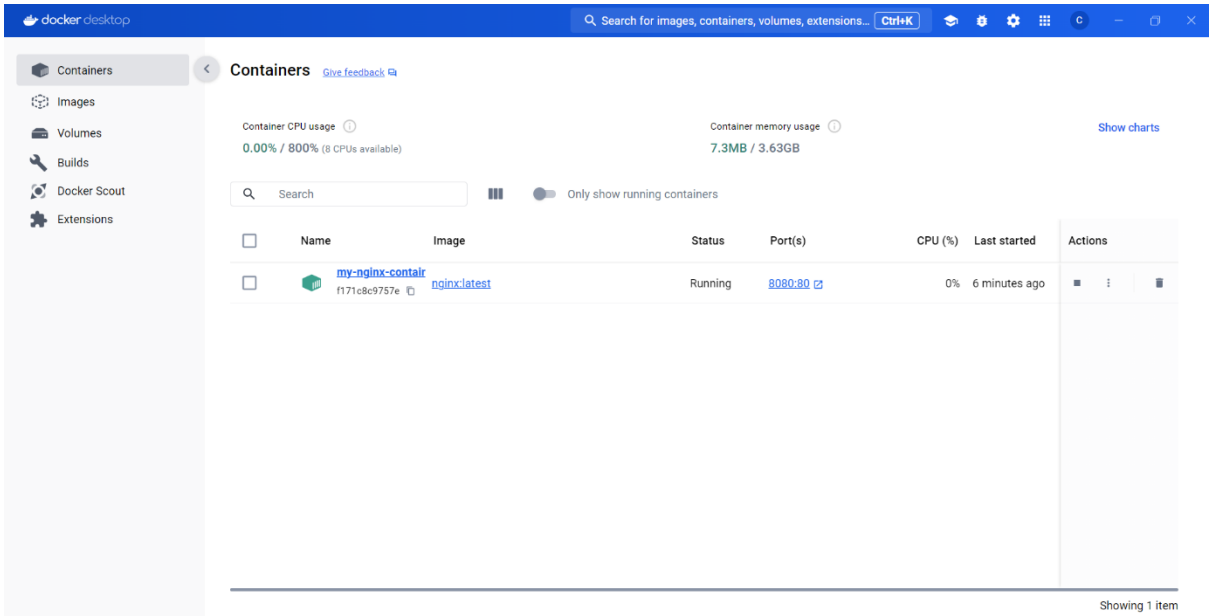
- NGINX is now accessible on your browser via: http://localhost:8080



- We see that container is running or not and image is there or not by using docker ps and docker images

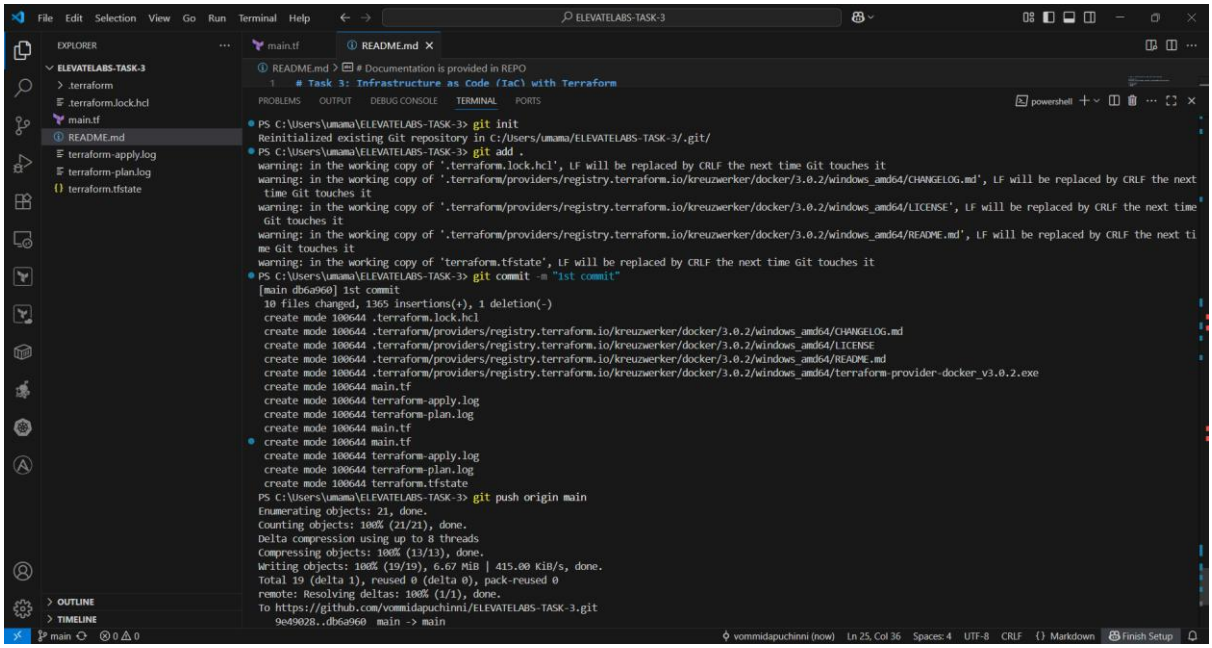


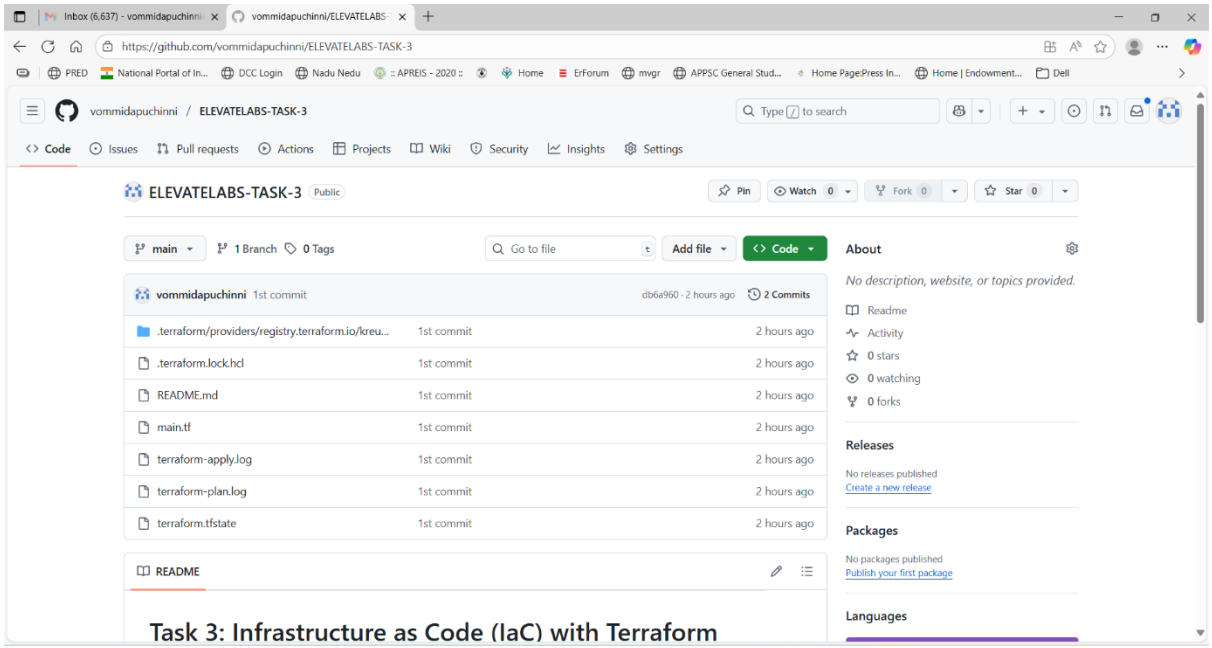




Git Workflow: Below are the steps followed to push the Terraform project to GitHub

- **Initialize Git →** Initialize Git
- **Add Files →** git add .
- **Commit Changes →** git commit -m "1st commit"
- **Push to GitHub →** git push origin main





Key Concepts Illustrated:

- Infrastructure as Code (IaC) – Manage Docker setup using Terraform files instead of manual commands.
- Docker Provider – Lets Terraform create and manage Docker containers and images.
- Resource Dependency – Terraform makes sure image is pulled before creating the container.
- Windows Docker Host – Uses npipe path to connect Terraform with Docker on Windows.

Conclusion

With just a few lines of code, we used **Terraform** to automate the process of:

- Pulling an image
- Creating a container
- Mapping ports
- Running a web server

This demonstrates the real-world utility of **IaC (Infrastructure as Code)** — making deployments consistent, repeatable, and version-controlled.