**Shravani Anil Patil**

**D15A-38**

**EXPERIMENT NO. 6**

**Aim :To Build, change, and destroy AWS infrastructure Using Terraform (S3 bucket or Docker**) .

**Theory :**

**Terraform** is an open-source tool that enables developers and operations teams to define, provision, and manage cloud infrastructure through code. It uses a declarative language to specify the desired state of infrastructure, which can include servers, storage, networking components, and more. With Terraform, infrastructure changes can be automated, versioned, and tracked efficiently.

# Building Infrastructure

When you build infrastructure using Terraform, you define the desired state of your infrastructure in configuration files. For example, you may want to create an S3 bucket or deploy a Docker container on an EC2 instance. Terraform reads these configuration files and, using the specified cloud provider (such as AWS), it provisions the necessary resources to match the desired state.

* **Docker on AWS:** Terraform can deploy Docker containers on AWS infrastructure. This often involves setting up an EC2 instance and configuring it to run Docker containers, which encapsulate applications and their dependencies.

# Changing Infrastructure

As your needs evolve, you may need to modify the existing infrastructure. Terraform makes it easy to implement changes by updating the configuration files to reflect the new desired state. For instance, you might want to change the storage settings of an S3 bucket, add new security policies, or modify the Docker container's configuration.

Terraform's "plan" command helps you preview the changes that will be made to your infrastructure before applying them. This step ensures that you understand the impact of your changes and can avoid unintended consequences.

# Destroying Infrastructure

When certain resources are no longer needed, Terraform allows you to destroy them in a controlled manner. This might involve deleting an S3 bucket or terminating an EC2 instance running Docker containers. By running the "destroy" command, Terraform ensures that all associated resources are properly de-provisioned and removed.

Destroying infrastructure with Terraform is beneficial because it helps avoid unnecessary costs associated with unused resources and ensures that the environment remains clean and free of clutter.

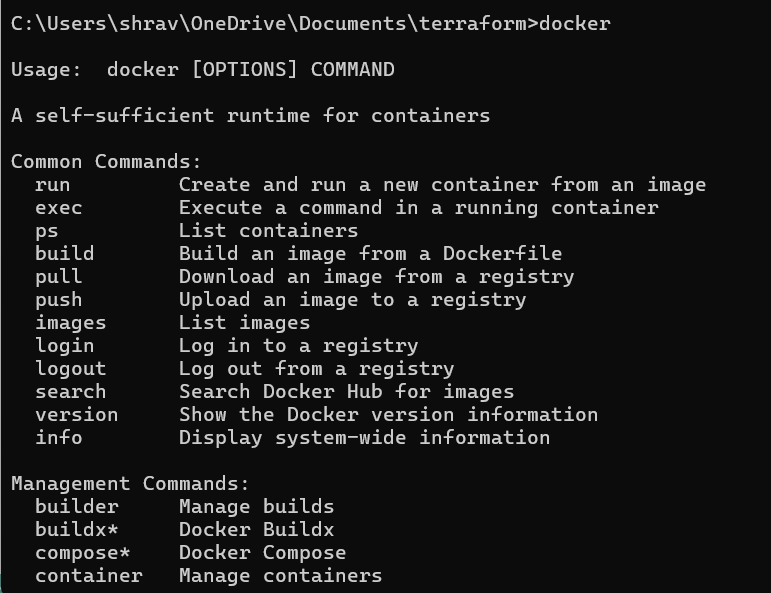
# Benefits of Using Terraform for AWS Infrastructure

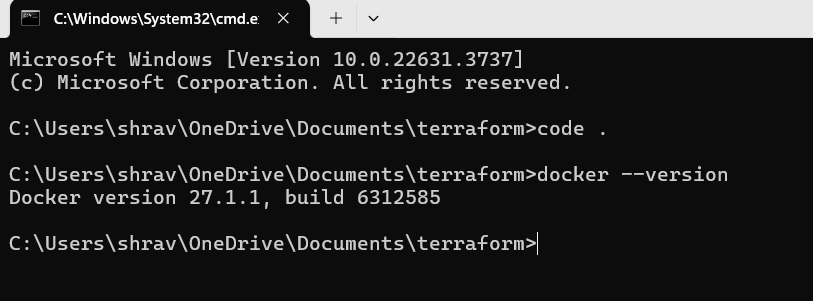
1. **Consistency:** Terraform ensures that infrastructure is consistent across environments by applying the same configuration files.
2. **Automation:** Manual processes are reduced, and infrastructure is provisioned, updated, and destroyed automatically based on code.
3. **Version Control:** Infrastructure configurations can be stored in version control systems (like Git), allowing teams to track changes, collaborate, and roll back if necessary.
4. **Scalability:** Terraform can manage complex infrastructures, scaling them up or down as needed, whether for small projects or large-scale applications.
5. **Modularity:** Terraform configurations can be broken down into reusable modules, making it easier to manage and scale infrastructure.

**Implementation :**

# Terraform and Docker -

Step 1 : check docker installation and version





Step 2 : create docker.tf file and write following code for terraform and docker

# Code -

terraform { required\_providers {

docker = {

source = "kreuzwerker/docker" version = "~> 3.0.1"

}

} }

provider "docker" {

host = "npipe:////.//pipe//docker\_engine"

}

resource "docker\_image" "nginx" {

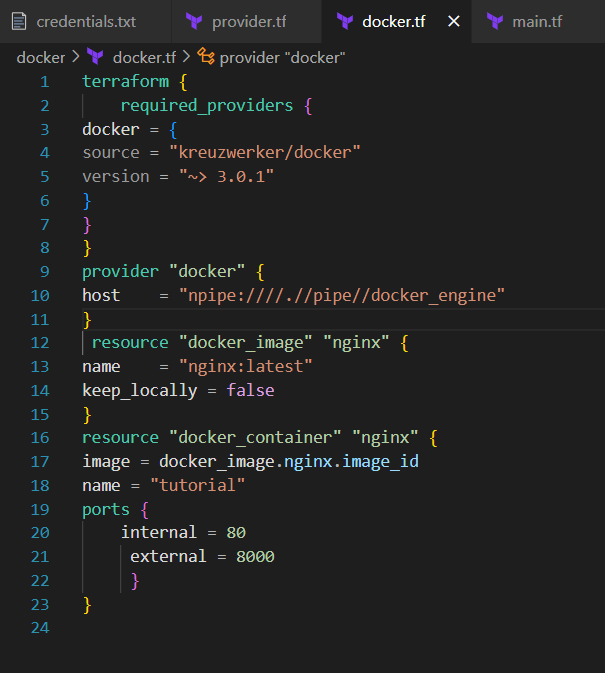
name = "nginx:latest" keep\_locally = false

} resource "docker\_container" "nginx" {

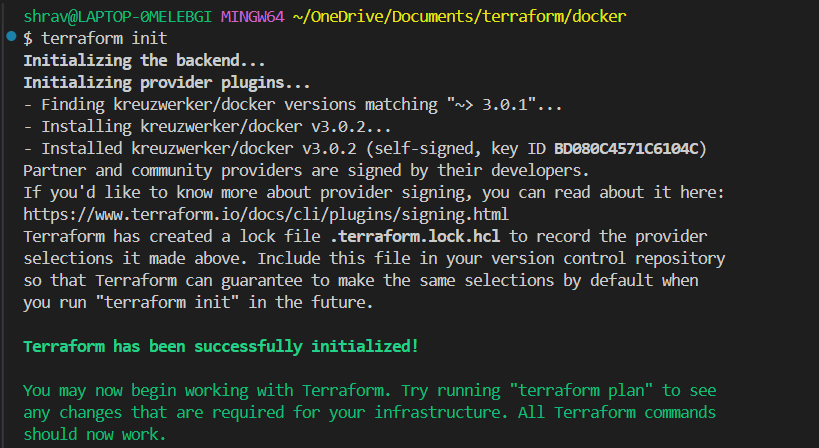
image = docker\_image.nginx.image\_id name = "tutorial" ports {

internal = 80 external = 8000 }

}

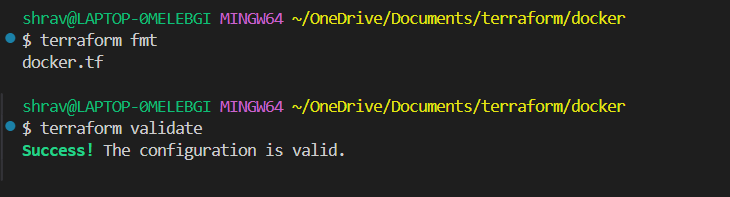


Step 3 : Type terraform init command to initialize terraform backend

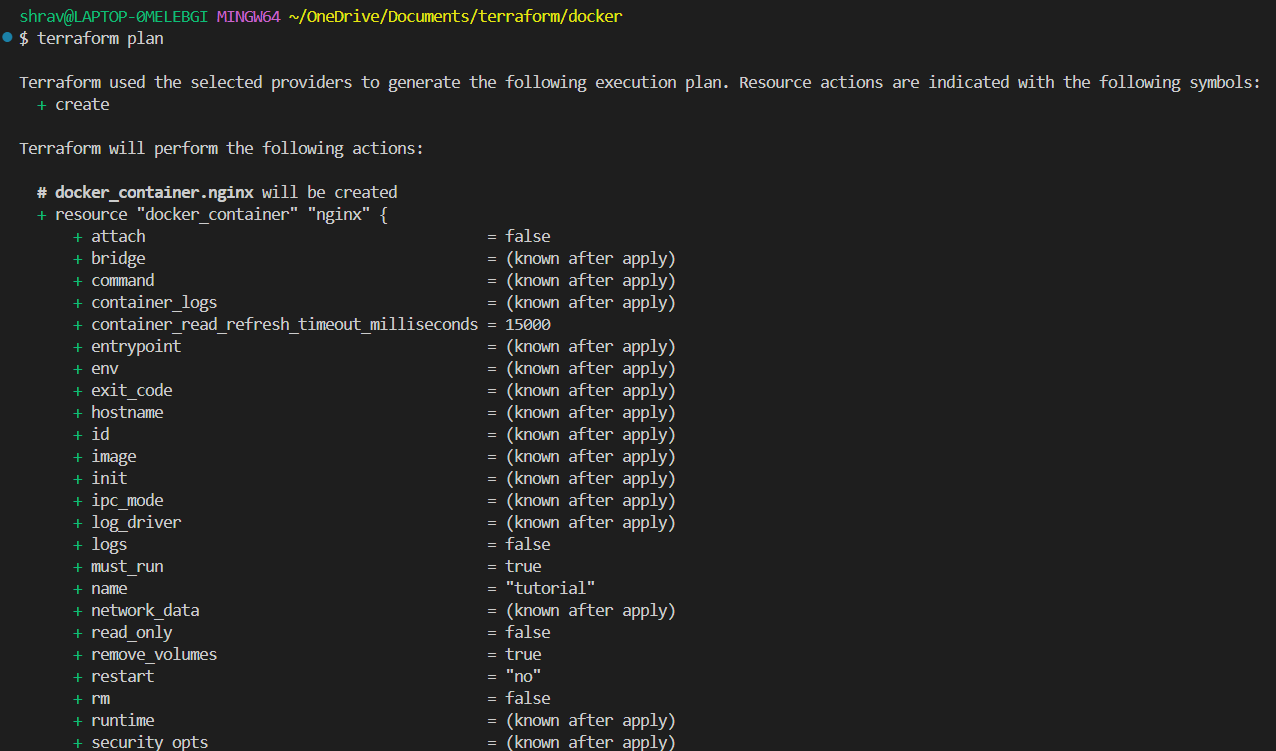


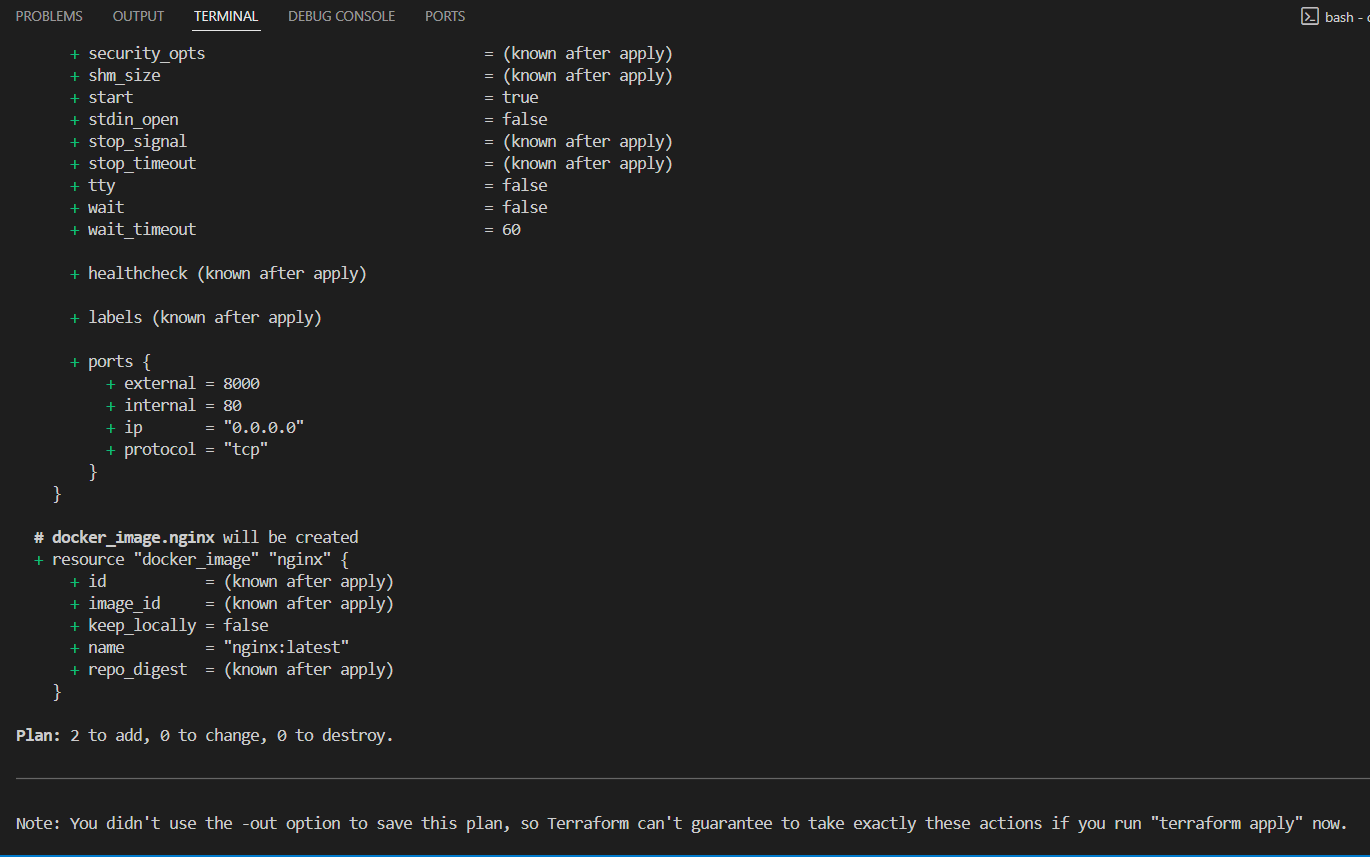
Step 4(EXTRA) : type terraform fmt and validate commands .

The two Terraform commands – terraform validate and terraform fmt – are used to maintain a clean, error-free, and well-structured Terraform codebase.

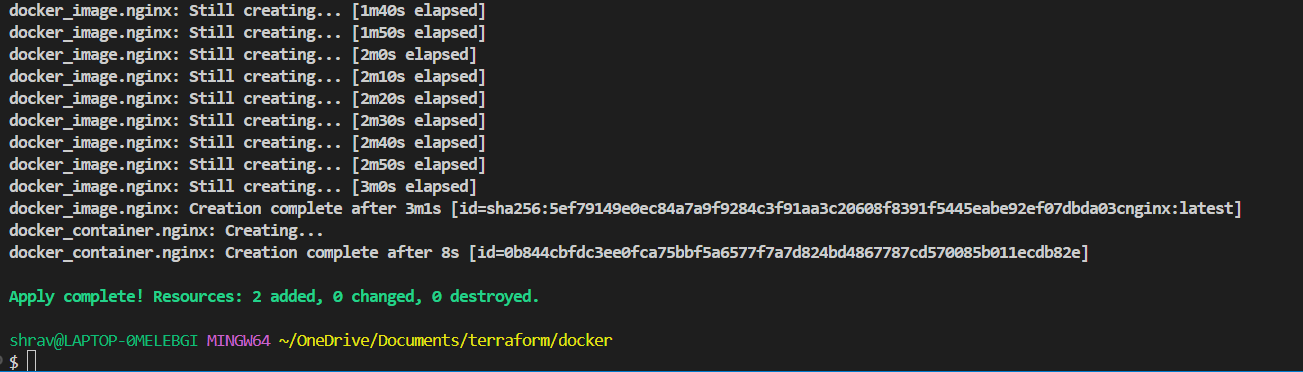


Step 5 : Type Terraform plan command to create execution plan .



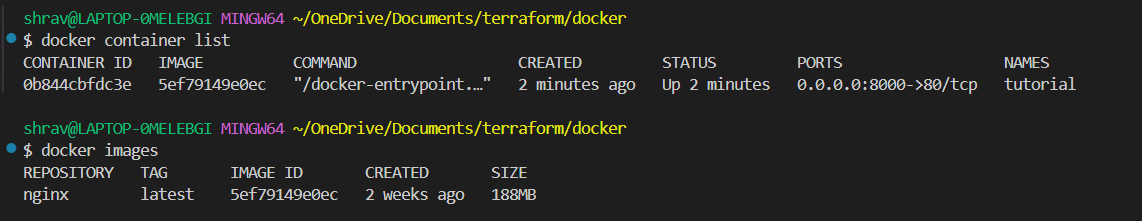


Step 6 : Type terraform apply to apply changes .

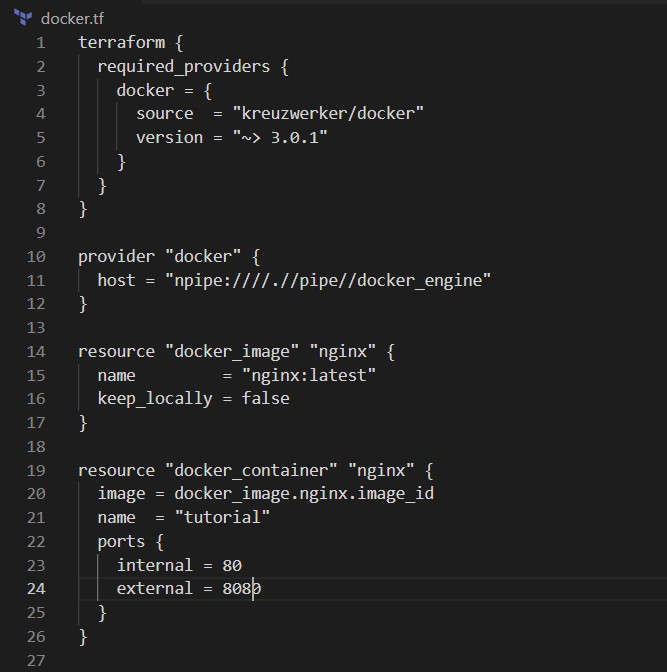


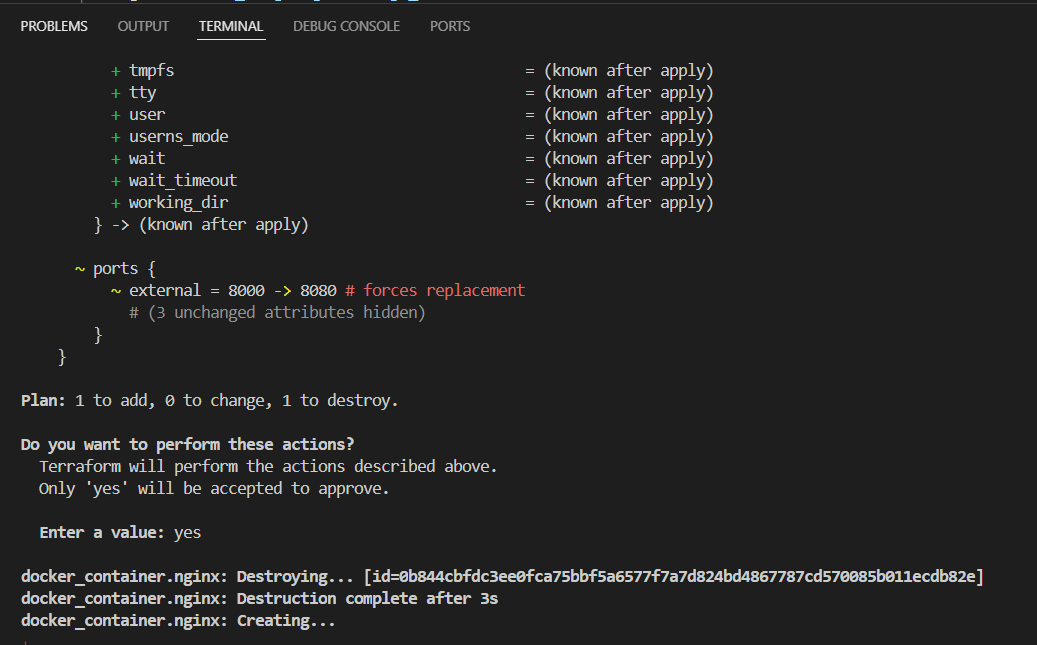
Step 7 : Docker container after step 6 execution BEFORE -

AFTER -

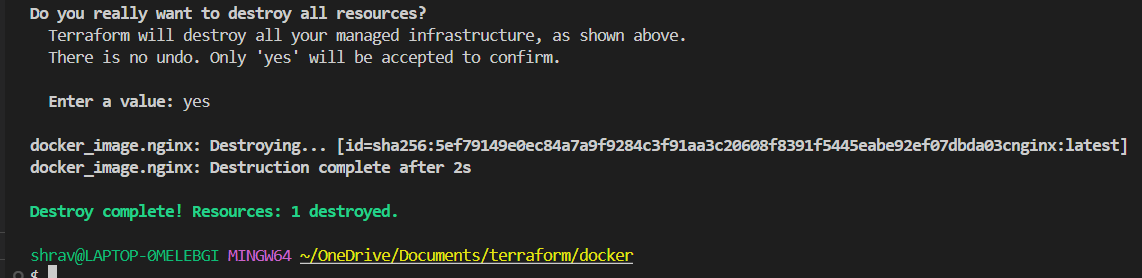
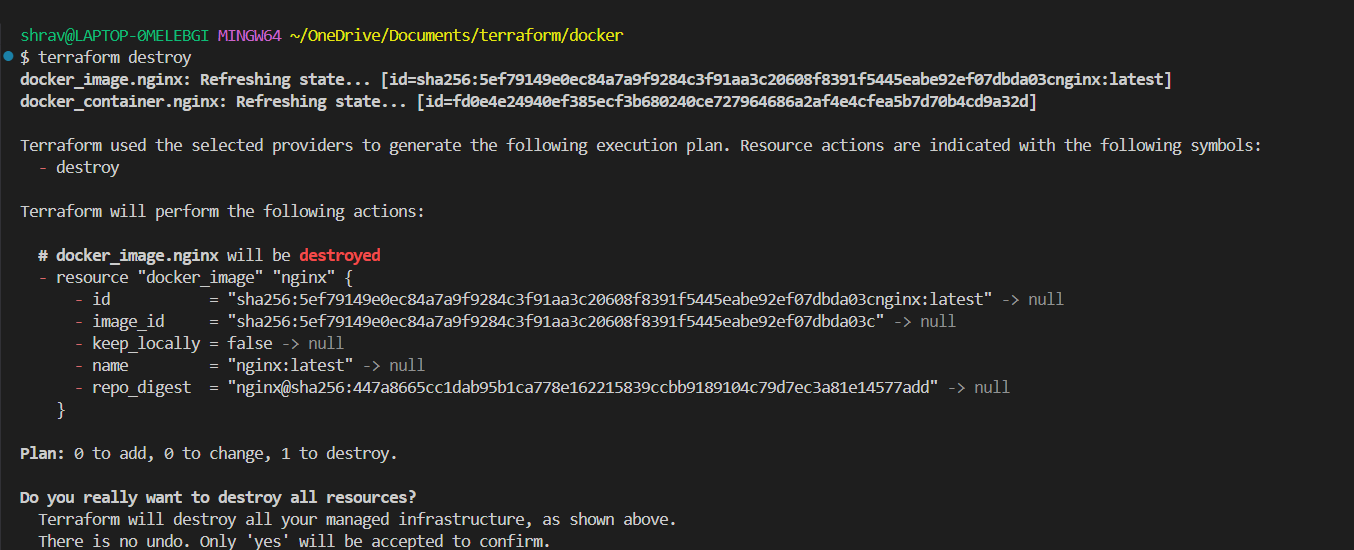


Step 8 (EXTRA ) : Execution of change .





Step 9 : terraform destroy to destroy infrastructure.



Step 10 : Docker after destroy command.

