**📘 Task 4– Infrastructure as Code (IaC) using Terraform with Docker**

**🎯 Objective:**

To provision and manage a local **Docker container** using **Terraform**, showcasing Infrastructure as Code principles.

**🛠 Tools Used:**

* **Terraform**
* **Docker Desktop for Windows**
* **Windows 10/11 CMD or PowerShell**

**📂 Project Files:**

* main.tf – Terraform configuration file
* Execution logs (to be captured manually during use)

**✅ Final Working Terraform Code (main.tf)**

hcl

CopyEdit

terraform {

required\_providers {

docker = {

source = "kreuzwerker/docker"

version = "~> 3.0.2"

}

}

}

provider "docker" {}

resource "docker\_image" "nginx\_image" {

name = "nginx:latest"

keep\_locally = false

}

resource "docker\_container" "nginx\_container" {

name = "nginx\_server"

image = docker\_image.nginx\_image.name

ports {

internal = 80

external = 8081 # Changed from 8080 to 8081 because 8080 was already in use

}

}

**🔄 Step-by-Step Execution and Explanation**

**✅ Step 1: Install Terraform and Docker**

* Download Terraform from: terraform.io
* Download Docker Desktop: docker.com
* Add Terraform to the Windows **System Path**.

**✅ Step 2: Create the Project Folder**

bash

CopyEdit

mkdir terraform-docker

cd terraform-docker

**✅ Step 3: Create the main.tf File**

* Paste the code provided above.

**✅ Step 4: Initialize Terraform**

bash

CopyEdit

terraform init

🔍 *Why?* Downloads the Docker provider and prepares the Terraform environment.

**✅ Step 5: Plan the Execution**

bash

CopyEdit

terraform plan

🔍 *Why?* Shows what Terraform will create. Useful for validating configuration.

**✅ Step 6: Apply the Changes**

bash

CopyEdit

terraform apply

🔍 *Why?* This command provisions the Docker image and container on your system.

* Open browser: http://localhost:8081 to view NGINX.

**✅ Step 7: List the Terraform State**

bash

CopyEdit

terraform state list

🔍 *Why?* Confirms that Terraform is tracking the image and container.

**✅ Step 8: Destroy the Resources**

bash

CopyEdit

terraform destroy

🔍 *Why?* Cleans up the Docker container and image provisioned by Terraform.

**⚠️ Mistakes We Made and What We Learned**

| **Mistake** | **What Happened** | **What We Learned** |
| --- | --- | --- |
| ❌ Used docker\_image.nginx\_image.latest | Gave error: Unsupported attribute | ✅ We must use docker\_image.nginx\_image.name instead. |
| ❌ Docker Daemon Not Running | Terraform couldn't connect to Docker | ✅ Always ensure Docker Desktop is **running** before using Terraform with Docker. |
| ❌ Port 8080 was already in use | Terraform failed to start container | ✅ Use a different port like 8081 or check for apps using that port. |

**📌 Purpose of Each Section in the Code**

| **Block** | **Purpose** |
| --- | --- |
| terraform | Defines required providers (here, Docker plugin) |
| provider "docker" | Connects Terraform with the local Docker engine |
| resource "docker\_image" | Downloads and manages the Docker image (nginx:latest) |
| resource "docker\_container" | Creates a running container using the above image |
| ports | Maps internal container port (80) to local machine port (8081) |

**📎 Notes for Submission**

* Include your main.tf
* Attach screenshots or copy-pasted logs from:
  + terraform init
  + terraform plan
  + terraform apply
  + terraform state list
  + terraform destroy

**🧠 Conclusion**

This task helped understand how to:

* Use Terraform as Infrastructure as Code
* Connect with Docker
* Handle configuration, planning, provisioning, and destruction of infrastructure using just code

**1. What is IaC (Infrastructure as Code)?**

**IaC** is the practice of managing and provisioning infrastructure (servers, networks, databases, etc.) using code instead of manual setup.  
It allows version control, automation, repeatability, and reduces human errors.

🧠 *Example:* You can write code to create a Docker container or an AWS EC2 instance rather than clicking through the GUI.

**2. How does Terraform work?**

Terraform works in **three major steps**:

1. **Write**: You define infrastructure using .tf files (HCL – HashiCorp Configuration Language).
2. **Plan**: Terraform shows what it will do (add/change/delete).
3. **Apply**: It creates or modifies real infrastructure to match the code.

Terraform uses **providers** (like AWS, Azure, Docker) to interact with real services.

**3. What is a Terraform state file?**

The **Terraform state file** (terraform.tfstate) keeps track of the actual infrastructure that Terraform manages.

📦 It stores:

* Resource names
* IDs
* Current settings

Terraform uses it to understand what already exists and what changes are needed.

**4. Difference between terraform plan and terraform apply**

| **Command** | **Purpose** |
| --- | --- |
| terraform plan | Shows what will happen **without** changing anything |
| terraform apply | Actually creates/changes infrastructure |

🧠 *Tip:* Use plan before apply to preview changes safely.

**5. What are Terraform providers?**

**Providers** are plugins in Terraform that let it talk to other platforms (like Docker, AWS, GCP, etc.).

🧱 Example:

* Docker provider → Manages containers
* AWS provider → Manages EC2, S3, RDS

You declare them like this:

hcl

CopyEdit

provider "docker" {}

**6. What is resource dependency in Terraform?**

A **resource dependency** occurs when one resource needs another to exist first.

Terraform automatically builds a **dependency graph** and decides the order.

📌 *Example:* If a Docker container uses a Docker image, the image is created first.

You can also force it manually using:

hcl

CopyEdit

depends\_on = [resource.name]

**7. How do you handle secret variables in Terraform?**

Secrets (like passwords or keys) should **not be hardcoded**.

✅ Best practices:

* Use **Terraform variables** and pass them through terraform.tfvars
* Use .gitignore to avoid exposing files
* Use **environment variables**:

bash

CopyEdit

export TF\_VAR\_db\_password="secret123"

* Use **tools** like HashiCorp Vault or AWS Secrets Manager

**8. Explain the benefits of Terraform.**

| **Benefit** | **Description** |
| --- | --- |
| ✅ IaC Automation | Easy to create, update, delete infrastructure via code |
| ✅ Platform-agnostic | Works with AWS, Azure, GCP, Docker, etc. |
| ✅ Version control | Infrastructure can be tracked using Git |
| ✅ Safe changes | Preview changes before applying (plan) |
| ✅ Reusability | Use modules and templates for repeatable setups |
| ✅ Community support | Lots of documentation and provider support |