QATIP Intermediate
Azure Lab01
A sample Terraform deployment

Contents

Lab Objectives

In this lab, you will:

- Review a simple sample terraform deployment configuration file
- Deploy a local Docker image and container
- Test the container
- Modify the deployment
- Test the modified deployment
- Destroy the deployment

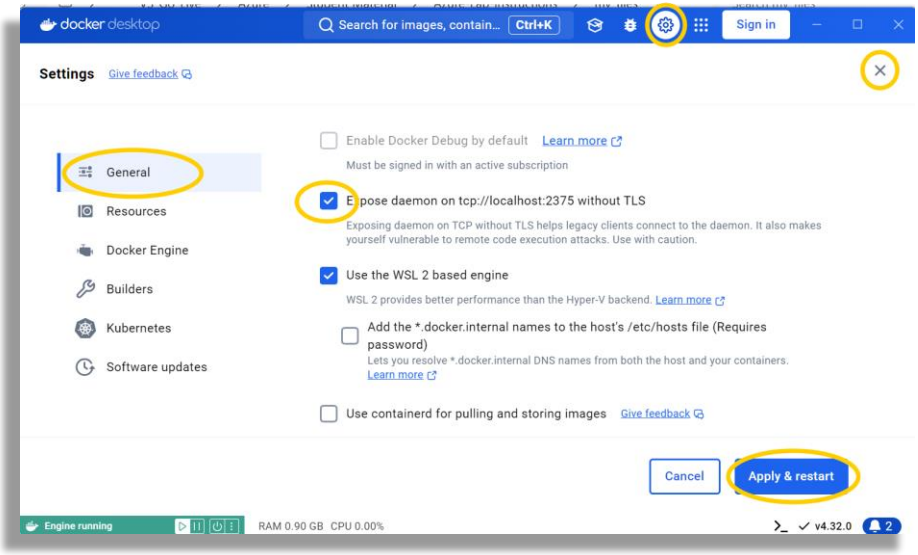
Teaching Points

This lab introduces you to Terraform and guides you through using it to interact with locally installed Docker. It introduces providers, used to indicate the infrastructure that will be managed, and also the structure and composition of terraform files. You will review an example terraform file before initializing the client, planning, and then applying the deployment. You will then modify the deployment before eventually destroying it.

Before you begin

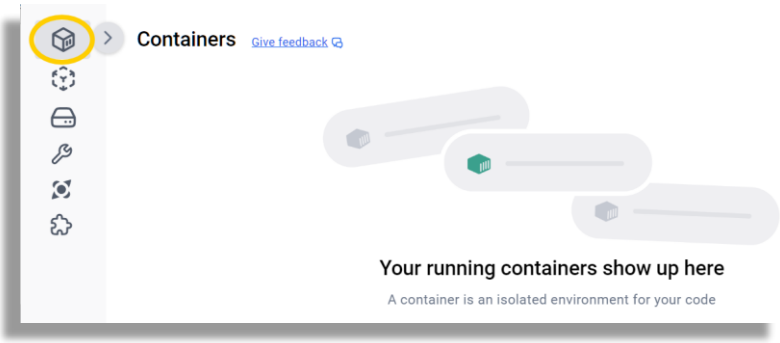
1. Before you begin this lab ensure you have completed Lab0.

2. On your remote virtual machine; start **Docker desktop** using the desktop shortcut and then use **settings** to ensure the docker service is listening on port 2375 without TLS..

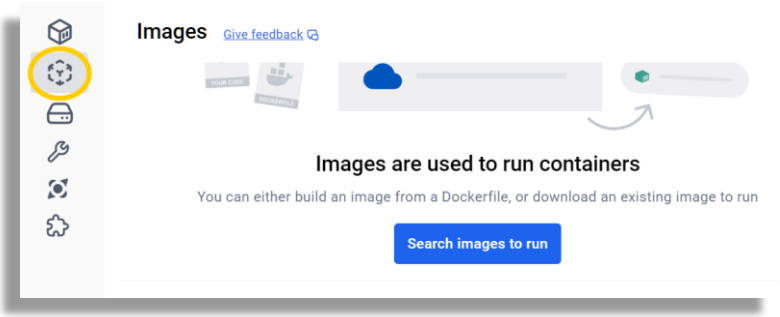


3. Select “Apply & Restart” and then exit settings using the lower X at top-right

4. Verify no containers exist (your version of Docker Desktop may differ)..



5. Verify no local images exist..



6. Minimize Docker but keep it running in the background.

Task1 Review the sample terraform configuration file

1. In the IDE, open C:\azure-tf-int\lab\01\main.tf

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

provider "docker" {
  host = "tcp://localhost:2375"
}

# Pull the image
resource "docker_image" "httpd" {
  name = "httpd:latest"
}

# Create a container
resource "docker_container" "webserver" {
  image = docker_image.httpd.image_id
  name  = "webserver"
  ports {
    internal = 80
    external = 88
  }
}
```

Breaking It Down

Defining the Terraform Block

```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
      version = "3.0.1"
    }
  }
}
```

This is the terraform code block. It specifies the provider(s) required for the configuration, here it is Docker, indicating that we want to use Terraform to manage Docker resources. The source section, line 4, identifies the Docker provider component to be downloaded during initialization, and line 5, indicates which version is required.

Initializing the Provider

```
10  provider "docker" {
11    host = "tcp://localhost:2375"
12  }
```

The provider block details parameters to be used when initializing the Docker provider. The configuration needed here is dependent upon the platform on which Docker is running. We are using Docker Desktop for Windows where the docker service is accessible using tcp port 2375.

Pulling the Docker Image

```
14  # Pull the image
15  resource "docker_image" "httpd" {
16    name = "httpd:latest"
17  }
```

This is one of two blocks of code that indicate the resources we want Terraform to instruct Docker to create. Here we are defining a docker image that needs to be pulled from Docker hub. Line 16 identifies the image we want, the latest version of “httpd”, an Apache webserver.

Creating the Docker Container

```
19  # Create a container
20  resource "docker_container" "webserver" {
21    image = docker_image.httpd.image_id
22    name  = "webserver"
23    ports {
24      internal = 80
25      external = 88
26    }
27  }
```

This is the second block of code that indicates the resources we want Terraform to instruct Docker to create. Here we are defining that we want docker to create a container called “web-server” (line22) and we want it to be accessible using ports 80 and 88 (lines 24-25). A docker container is created from a docker image. Line 21 indicates that the image to be used to create this container is the one we ask Docker to pull from Docker hub in lines 15-17. This is an example of one resource code block referencing another and it also indicates to Terraform that it must wait for Docker to download the image ***before*** it can ask it to create the container using it. This is known as implicit dependency; the container depends upon the image existing.

Task2 Run Terraform & Test

1. In your IDE terminal, ensure you are in the correct working directory (C:\azure-tf-int\labs\01) and then run the terraform initialization command **terraform init**
2. Run **terraform plan** and review what will be created

```
  # docker_image.httpd will be created
  + resource "docker_image" "httpd" {
      + id          = (known after apply)
      + image_id    = (known after apply)
      + name        = "httpd:latest"
      + repo_digest = (known after apply)
    }

Plan: 2 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save
"terraform apply" now.
PS C:\azurelabs\lab1>
```

3. Run **terraform apply** typing **yes** when prompted.

```
Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

4. If prompted, Windows Defender should be configured to allow access
5. Switch to Docker Desktop and verify the image download and container creation.
6. Switch back to the IDE terminal and run **docker images** ..

```
PS C:\azurelabs\lab1> docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
httpd        latest    f81fca8b7f74   6 months ago   148MB
PS C:\azurelabs\lab1>
```

   This is the httpd image that Terraform instructed Docker to download

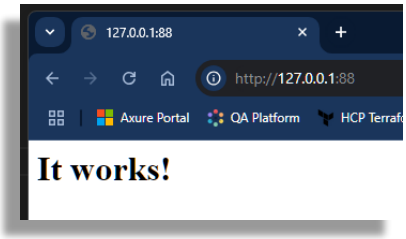
7. Run **docker ps** to list running containers.

```
PS C:\azurelabs\lab1> docker ps
CONTAINER ID   IMAGE          COMMAND              CREATED         STATUS         PORTS                  NAMES
b8d1c2d79433   f81fca8b7f74   "httpd-foreground"   2 seconds ago   Up 2 seconds   0.0.0.0:88->80/tcp   webserver
PS C:\azurelabs\lab1>
```

This is the container named ‘webserver’, that Terraform instructed Docker to start, using the downloaded image. Note the last few digits of the unique container ID for your running container (it will differ from the one shown here)

7. Use a new web browser tab within your virtual desktop to access the container using http://127.0.0.1:88

8. View the “**It Works**" response from the Apache web server (index.html)



9. We will now modify the deployment by changing the external port number from 88 to 8088. Some resource changes can be applied to the existing instance of a resource, whilst others require the destruction of the original resource and the creation of a replacement. Let us see whether a port change modifies or recreates this resource.

10. Return to the main.tf file and change the external port on line 25 to **8088**.

```
23    ports {
24      internal = 80
25      external = 8080
26    }
```

11. Save the file changes using Ctrl+s

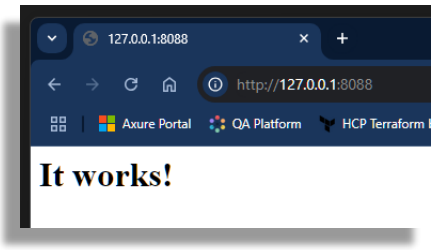
12. Run **terraform plan** again and review the changes. The change of port will force the current container to be deleted and a new one created…

```
      ~ ports {
          ~ external = 88 -> 8088 # forces replacement
            # (3 unchanged attributes hidden)
        }
    }

Plan: 1 to add, 0 to change, 1 to destroy.
```

13. Run **terraform apply**, typing **yes** when prompted

14. Browse to **http://127.0.0.1:8088** This should return a success "**It works**" showing that the new container is listening on port 8088



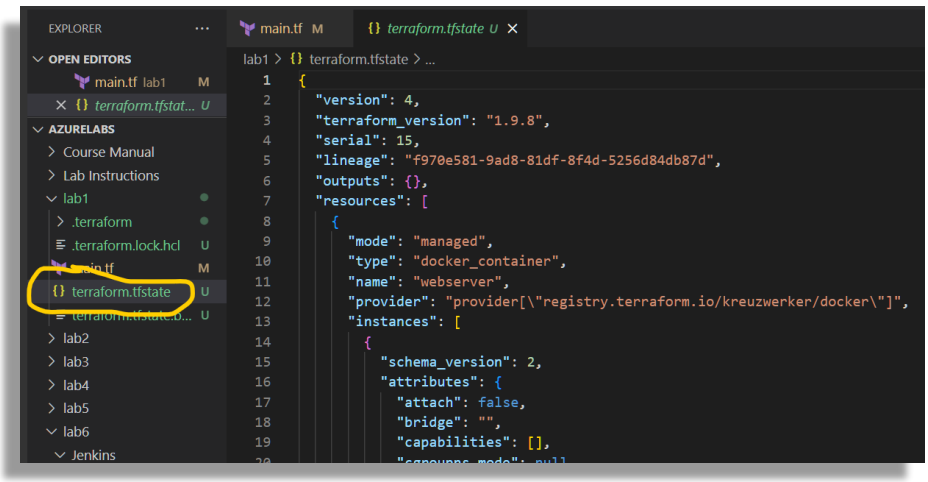
15. Run **docker ps** again

```
PS C:\azurelabs\lab1> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                    NAMES
83efc18dafa6   f81fca8b7f74   "httpd-foreground"   9 seconds ago   Up 8 seconds   0.0.0.0:8088->80/tcp   webserver
PS C:\azurelabs\lab1>
```

Note that your CONTAINER ID will have changed – Docker containers are immutable so Terraform destroyed the original container and deployed a replacement using the same image. This is an important point to be aware of as we progress through the course. Whether we 'update' or 'replace' a resource when making changes depends upon the resource itself and the changes we make.

Task3 Examine the state file

1. When 'terraform apply' is run for the first time, a persistent record of resources Terraform deploys and manages is created in the form of a state file, terraform.tfstate. This file is updated whenever you ask Terraform to create, change, or destroy resources thereafter. By default this file will be in your working directory. Click on it now to examine its contents..



Task4 Destroy the deployment

1. Run **terraform destroy**, review the output and type **yes**
2. Run **docker ps** to confirm your container has been deleted
3. Run **docker images** and confirm your image has been deleted
4. Revisit the state file and view its now reduced content
5. Close the Docker Desktop window

Congratulations, you have completed this lab