

# QATIP Intermediate

## AWS Lab01

### A sample Terraform deployment

Lab Objectives .....	1
Teaching Points.....	1
Before you begin .....	1
Task1 Review the sample terraform configuration file .....	2
Task2 Run Terraform & Test .....	4
Task3 Examine the state file .....	5
Task4 Lab Clean-up .....	5

#### 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. Ensure you have completed Lab0 before attempting this lab.
2. In the IDE terminal window, enter the following command

`cd ~/environment/aws-tf-int/labs/01`

3. This shifts your current working directory to aws-tf-int/labs/01. **Ensure all commands are executed in this directory**
4. Close any existing files and use the Explorer pane to open **main.tf** in the labs/01 folder

## Task1 Review the sample terraform configuration file

1. In the IDE, open main.tf

```
1 terraform {
2   required_providers {
3     docker = {
4       source = "kreuzwerker/docker"
5       version = "3.0.1"
6     }
7   }
8 }
9
10 provider "docker" {
11   host = "unix:///var/run/docker.sock"
12 }
13
14 # Pulls the image
15 resource "docker_image" "apache_web" {
16   name = "httpd:latest"
17 }
18
19 # Create a container
20 resource "docker_container" "web_server" {
21   image = docker_image.apache_web.image_id
22   name = "web_server"
23   ports {
24     internal = 80
25     external = 88
26   }
27 }
```

## Breaking It Down

### Defining the Terraform Block

```
1 terraform {
2   required_providers {
3     docker = {
4       source = "kreuzwerker/docker"
5       version = "3.0.1"
6     }
7   }
8 }
```

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 = "unix:///var/run/docker.sock"  
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 Cloud9 here which runs on a Unix derivative operating system.

### 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. Run **terraform init**
2. Run **terraform plan** -- review what will be created
3. Run **terraform apply** typing **yes** when prompted. Review output in the CLI
4. Run **docker images** in CLI. This should show that httpd image has been downloaded (your IMAGE ID will differ)..

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
httpd	latest	dad6ca1caf78	4 months ago	148MB

5. Run **docker ps** to list your running containers (your CONTAINER ID will differ)...

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ecdc455452a6	f81fca8b7f74	"httpd-foreground"	7 seconds ago	Up 6 seconds	0.0.0.0:88->80/tcp	web_server

6. Note the last few characters of your container ID, yours will differ from the one shown above.
7. Type **curl http://127.0.0.1:88** View the "**It Works**" response that is standard with an Apache web server (index.html)
8. 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 the 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.
9. Return to the main.tf and change the external port to 8088.
10. Save the file
11. Run **terraform plan** and review the changes
12. Run **terraform apply**, typing **yes** when prompted
13. Type **curl <http://127.0.0.1:8088>** This should return a success "it works"

14. Type `curl http://127.0.0.1:88` An error message is returned.

```
<html><body><h1>It works!</h1></body></html>
```

```
curl: (7) Failed to connect to 127.0.0.1 port 88 after 0 ms: Connection refused
```

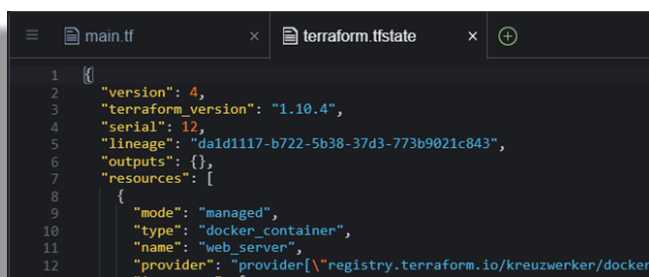
15. Run `docker ps`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
cd148da1feb6	f81fca8b7f74	"httpd-foreground"	6 seconds ago	Up 5 seconds	0.0.0.0:8088->80/tcp	web_server

16. Note that the name of the container is the same, but the ID has changed - Terraform destroyed the original container and deployed a replacement. This is a crucial point to be aware of as we progress through the course. Whether we 'update' or 'replace' a resource depends upon the resource itself and the changes we make.

### Task3 Examine the state file

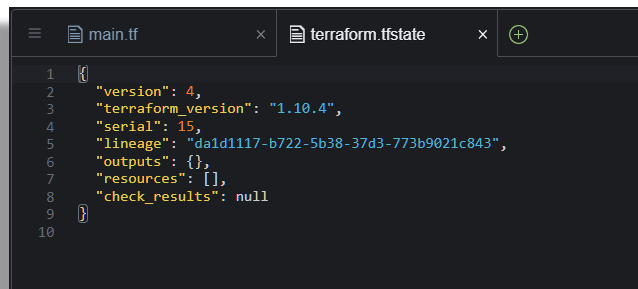
1. When 'terraform apply' is run for the first time, a persistent record of resources Terraform deploys and manages is created in the form of a state file, `terraform.tfstate`. This file is updated whenever you ask Terraform to create, change, or destroy resources thereafter. By default this file will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



```
1 {
2   "version": 4,
3   "terraform_version": "1.10.4",
4   "serial": 12,
5   "lineage": "da1d1117-b722-5b38-37d3-773b9021c843",
6   "outputs": {},
7   "resources": [
8     {
9       "mode": "managed",
10      "type": "docker_container",
11      "name": "web_server",
12      "provider": "provider[\"registry.terraform.io/kreuzwerker/docker\"]",
13      "instances": [
```

### Task4 Lab Clean-up

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



The image shows a code editor window with two tabs: 'main.tf' and 'terraform.tfstate'. The 'terraform.tfstate' tab is active, displaying a JSON configuration for a Terraform state file. The code is as follows:

```
1  {
2    "version": 4,
3    "terraform_version": "1.10.4",
4    "serial": 15,
5    "lineage": "da1d1117-b722-5b38-37d3-773b9021c843",
6    "outputs": {},
7    "resources": [],
8    "check_results": null
9  }
```

**### Congratulations, you have completed this lab ###**