Lab1. Creating a simple resource using Terraform

This lab will create a Docker Container on your Cloud9 EC2 AWS instance.

Lab Objectives

In this lab, you will:

- Review the Terraform registry and Docker documentation
- Build a simple main.tf configuration file based on example code
- Deploy a Docker image and container
- Test the container
- Modify the deployment
- Test the modified deployment
- Destroy the deployment

Solution

The solution to this lab can be found in awslabs/solutions/01. Try to use this only as a last resort if you are struggling to complete the step-by-step processes.

Start Lab

1. Ensure you have completed Lab 0 before attempting this lab.
2. In the IDE terminal window, enter the following command

   cd ~/environment/awslabs/01

3. This shifts your current working directory to awslabs/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 awslabs/01 folder

Task 1: Review the documentation and create a configuration file

1. Review docker/TF documentation: https://registry.terraform.io/providers/kreuzwerker/docker/3.0.1/docs

2. Note: At time of writing, version 3.0.1 was the latest version of this Provider. This may have changed. This lab was created and tested using version 3.0.1 so please ensure you use this version.

3. Click `**Use Provider**`

4. Copy the code block into the empty main.tf in the labs/01 folder. For convenience, the code is listed below:

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

provider "docker" {
  # Configuration options
}
```

5. From within the 'Example Usage' section of the documentation, copy the sections relating to "**Pulls the image**" and "**Create a container**" and add these lines below the current content of main.tf For convenience the code is listed below:

```
# Pulls the image
resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}

# Create a container
resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
```

```
  name  = "foo"
}
```

Task 2: Update the configuration file

1. Modify the docker_image resource block to deploy the latest httpd image rather than Ubuntu. Rename the resource to '**apache_web'** and reference the latest **httpd** image.

```
resource "docker_image" "apache_web" {
  name  = "httpd:latest"
}
```

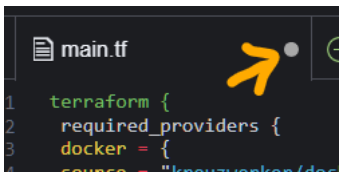
2. Modify the docker_container resource block to create a container called '**web_server'** using the docker_image resource '**apache_web'**

```
resource "docker_container" "web_server" {
  image = docker_image.apache_web.image_id
  name  = "web_server"
}
```

3. Add ports for internal as 80 and external as 8081. The docker_container resource block should now be...

```
resource "docker_container" "web_server" {
  image = docker_image.apache_web.image_id
  name  = "web_server"
  ports {
    internal = 80
    external = 8081
  }
}
```

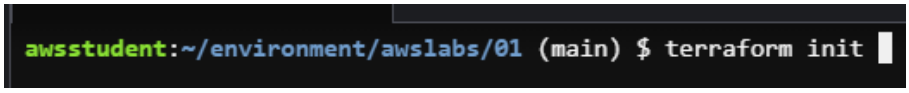
4. The Cloud9 IDE indicates file that have been changed but not saved by displaying a grey circle against the file…



5. Save your changes using Ctrl+s…

Task 3: Run Terraform & Test

1. Run **terraform init** ensuring you are in the correct working directory…

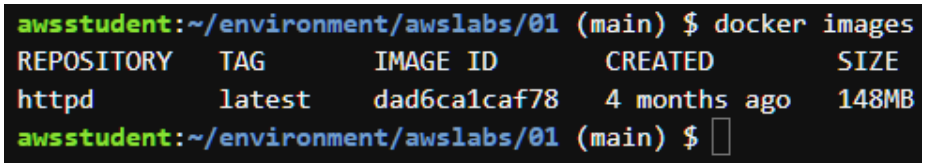


2. If there are any errors, correct them before continuing. (Use the solution guide if needed - but try first!)

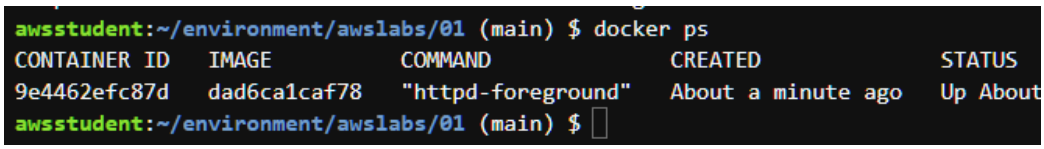
3. Run **terraform plan** -- review what will be created

4. Run **terraform apply** typing **yes** when prompted. Review output in the CLI

5. Run **docker images** in CLI. The httpd image has been downloaded..



6. Run **docker ps** to list your running containers…



We will now modify the deployment by changing the external port number from 8081 to 88. 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.

7. Note the alphanumeric value of the container ID. For example, something similar to: 9e4462efc87d

8. Type **curl http://127.0.0.1:8081** View the "it Works" response that is standard with an Apache web server (index.html)

9. Return to the main.tf and change the external port to 88.

10. Save the file

11. Run **terraform plan** and review the changes

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

13. Once the new deployment completes, type **curl http://127.0.0.1:8081**
This should fail as we no longer have a container running on port 8081

14. Type **curl http://127.0.0.1:88** We should return a success "it works"

15. Run **docker ps**

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.

Task 4: Destroy your 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