Exercise 4b

Using Docker Compose with Terraform on AWS

Prior Knowledge
Unix command-line
Apt package manager
Amazon AWS Access key and SSH key

Learning Objectives
Understand how Terraform manages cloud infra

Software Requirements
- AWS
- Docker-Compose
- Terraform

Overview

Terraform is a very useful tool from Hashicorp that enables declaratively defining and managing infrastructure in various cloud environments, including AWS, Azure, Google, Kubernetes, Alibaba and many others.

https://terraform.io

We are going to use Terraform to instantiate a virtual machine in EC2 and run a Docker Compose workload in it.

Steps

1. Install terraform into the Ubuntu VM:

   ```
   wget -O - -q https://freo.me/install-tf | bash
   ```

2. Test it works:

   ```
   $ terraform -version

   Terraform v1.0.2
   on linux_amd64
   ```

3. Clone the sample repository:

   ```
   git clone https://github.com/pzfreo/clo-tf-docker.git
   cd clo-tf-docker
   ```


4. Take a look at the file main.tf. It is reasonably well commented and should make sense.

   ```
   code main.tf
   ```

   The main part is the bit that creates the AWS EC2 instance (everything else is creating subsidiary parts):

```
124  resource "aws_instance" "web" {
125    ami           = data.aws_ami.ubuntu.id
126    instance_type = lookup(var.awsprops, "instance_type")
127
128    key_name = var.student
129
130    root_block_device {
131      volume_size = 8
132    }
133
134    # Use the following User Data to install docker and docker compose, clone the repository
135    user_data = <<-EOF
136      #!/bin/bash
137      set -ex
138      sudo apt update
139      sudo apt install docker.io -y
140      sudo service docker start
141      sudo usermod -a -G docker ubuntu
142      sudo apt install python3-pip -y
143      sudo pip3 install docker-compose
144      cd /home/ubuntu
145      git clone ${lookup(var.awsprops, "dc-repository")} dc
146      cd dc
147      docker-compose up --build
148    EOF
149
150
151    vpc_security_group_ids = [
152      module.ec2_sg.security_group_id,
153      module.dev_ssh_sg.security_group_id
154
155    ]
156    iam_instance_profile = aws_iam_instance_profile.ec2_profile_clo_tfdc.name
157
158    tags = {
159      Name = format("%s-tf",var.student)
160    }
```

5. Notice how much control we have over subnets, VPCs, IAM roles, security groups, etc!

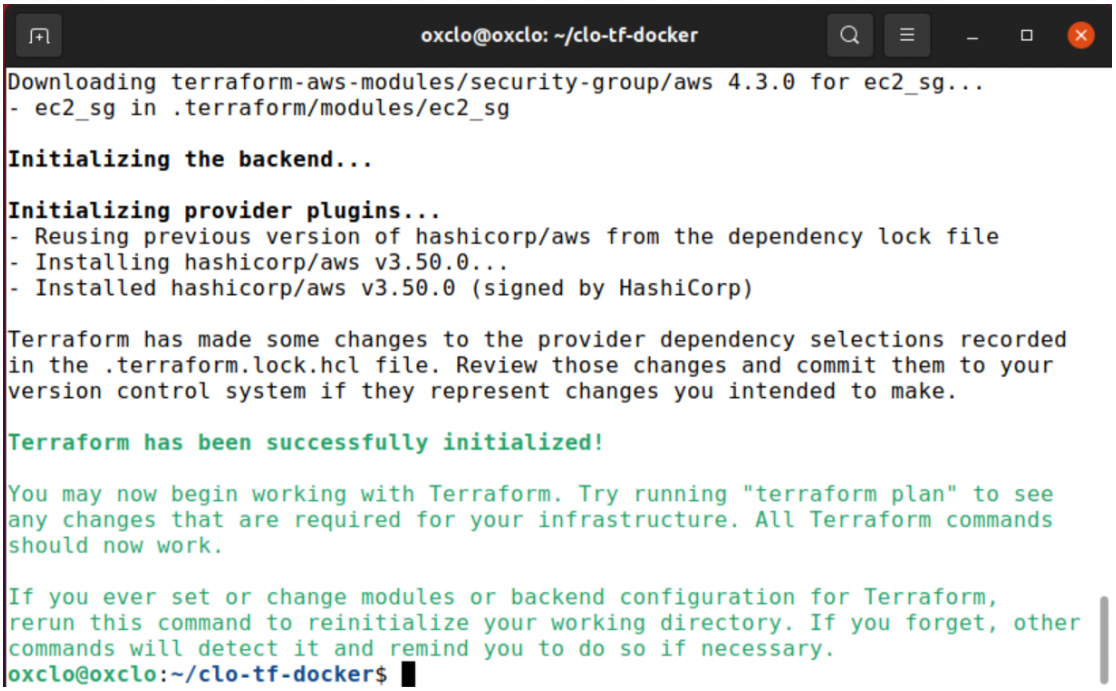
6. It is also really useful that you can use variables in this language (which is known as Hashicorp Configuration Language (HCL)).

7. Change the default student name to your student name. (You can also override this when you use this plan)

8. Before we can use terraform, we need to initialise the system, which downloads any providers that are defined (in our case the AWS EC2 provider):

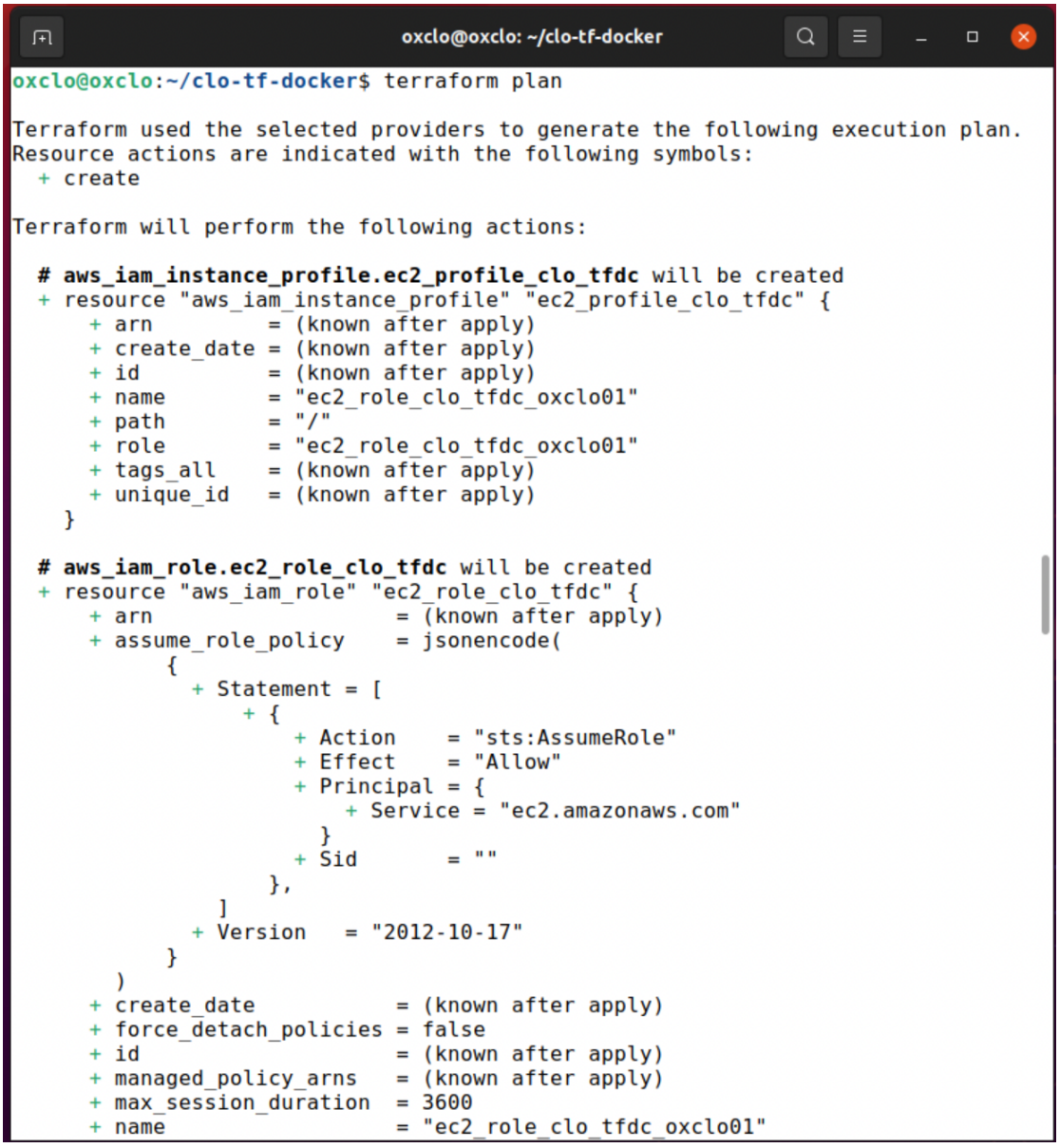
   terraform init

   You should see a bunch of stuff scroll by:



9. You can see what terraform plans to do by:

```
terraform plan
```



10. Take a look at the docker-compose.yaml:

    This is the standard “wordpress” docker-compose:

```
version: "3.9"

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - wordpress_data:/var/www/html
    ports:
      - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
  wordpress_data: {}
```

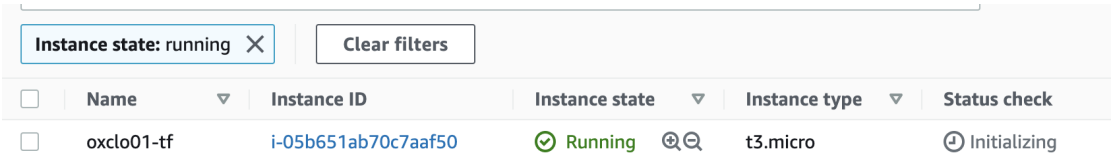

11. Now we can apply our terraform project:

```
terraform apply -auto-approve
```

12. Terraform will use your access key and secret to call EC2 to create everything needed:

```
          + device_index          = (known after apply)
          + network_interface_id  = (known after apply)
        }

      + root_block_device {
          + delete_on_termination = true
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = 8
          + volume_type           = (known after apply)
        }
    }

Plan: 2 to add, 0 to change, 1 to destroy.

Changes to Outputs:
  + instance_public_ip = {
      + public_ip = (known after apply)
    }
aws_iam_instance_profile.ec2_profile_clo_tfdc: Destroying... [id=ec2_role_clo_tf
dc_oxclo01]
aws_iam_instance_profile.ec2_profile_clo_tfdc: Destruction complete after 1s
aws_iam_instance_profile.ec2_profile_clo_tfdc: Creating...
aws_iam_instance_profile.ec2_profile_clo_tfdc: Creation complete after 1s [id=ec
2_role_clo_tfdc_oxclo01]
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Creation complete after 15s [id=i-0583c52725c3085ad]

Apply complete! Resources: 2 added, 0 changed, 1 destroyed.

Outputs:

instance_public_ip = {
  "public_ip" = "34.243.7.38"
}
```

13. You should see your instance starting in the EC2 console:



14. It will take a bit of time for the userdata / cloud-init script to run.
    If you want, you can SSH into the instance and

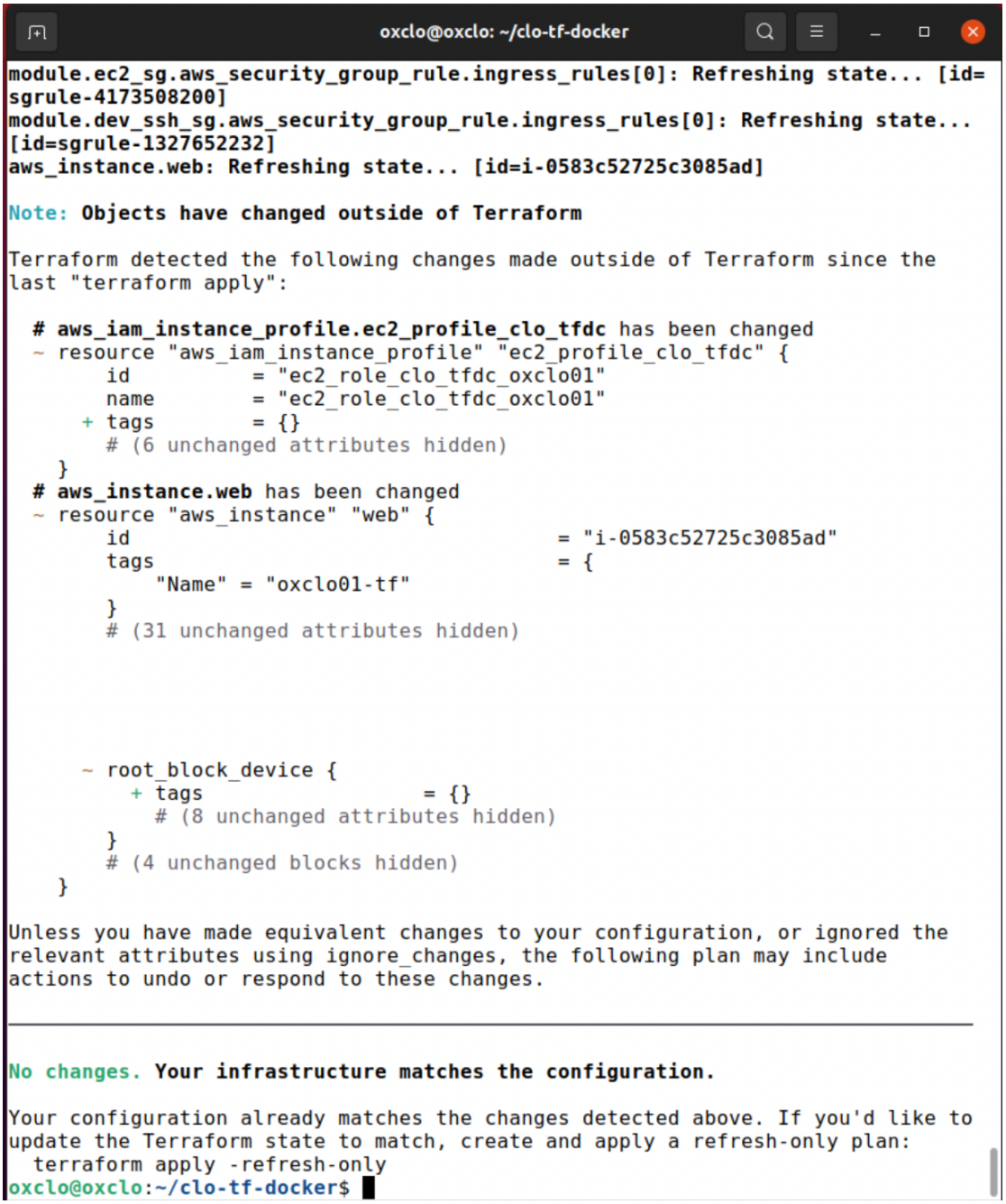
    ```
    tail -f /var/log/cloud-init-output.log
    ```

    to see what is happening

15. Once it does, you should be able to access your service on port 80 on the server.
    The IP address was printed out as part of the startup.

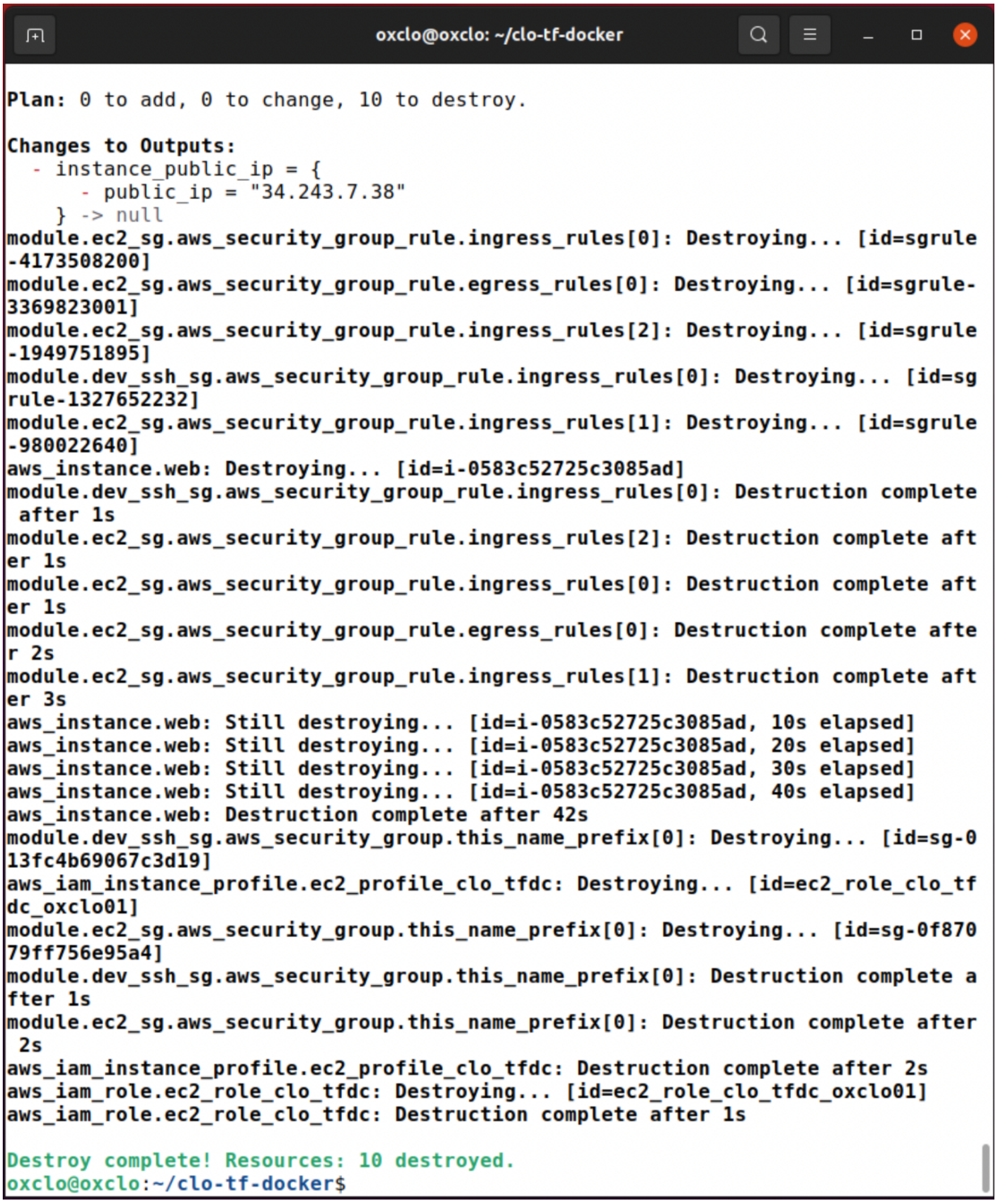
16. You can also check if the current infrastructure matches the plan:

    ```
    terraform plan
    ```

17. You can delete everything with:

    terraform destroy -auto-approve



18. That's all!

Extension

Fork the clo-tf-docker repository in Github and modify it to use the docker-compose file from Ex 4.

Hints:

1. You will need to change which repository is pulled from Git in main.tf
2. You will need to use the docker-compose file from the node-docker repository
3. Notice that because I have previously done “docker-compose push”, it is possible to “docker-compose pull / docker-compose up” without the source
4. If we were being more rigorous, we could version the docker images and create a git tag / release to specify the exact version of the docker-compose file to use.