

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_tfcd.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)

- 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:

```
oxclo@oxclo: ~/clo-tf-docker
Downloading terraform-aws-modules/security-group/aws 4.3.0 for ec2_sg...
- ec2_sg in .terraform/modules/ec2_sg

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v3.50.0...
- Installed hashicorp/aws v3.50.0 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

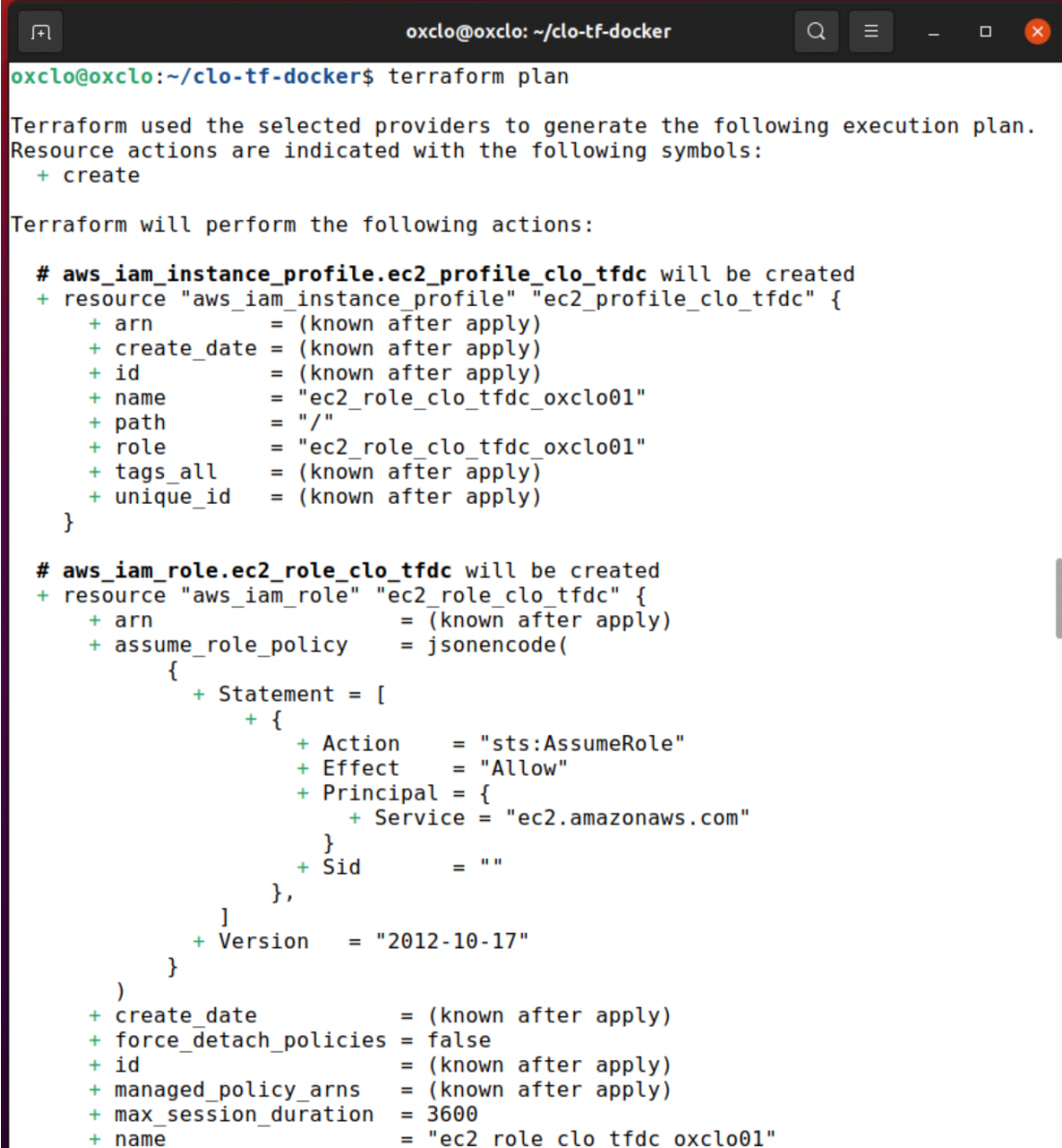
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
oxclo@oxclo:~/clo-tf-docker$
```

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

terraform plan

A terminal window titled 'oxclo@oxclo: ~/clo-tf-docker' showing the output of the 'terraform plan' command. The output indicates that two AWS IAM resources will be created: an instance profile and a role. The instance profile 'ec2_profile_clo_tfdc' will have a name of 'ec2_role_clo_tfdc_oxclo01'. The role 'ec2_role_clo_tfdc' will assume the instance profile and have a name of 'ec2_role_clo_tfdc_oxclo01'. The role's assume_role_policy is shown as a JSON object with a single statement allowing sts:AssumeRole from ec2.amazonaws.com.

```
oxclo@oxclo:~/clo-tf-docker$ terraform plan

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

# aws_iam_instance_profile.ec2_profile_clo_tfdc will be created
+ resource "aws_iam_instance_profile" "ec2_profile_clo_tfdc" {
  + arn          = (known after apply)
  + create_date  = (known after apply)
  + id           = (known after apply)
  + name         = "ec2_role_clo_tfdc_oxclo01"
  + path         = "/"
  + role         = "ec2_role_clo_tfdc_oxclo01"
  + tags_all     = (known after apply)
  + unique_id    = (known after apply)
}

# aws_iam_role.ec2_role_clo_tfdc will be created
+ resource "aws_iam_role" "ec2_role_clo_tfdc" {
  + arn              = (known after apply)
  + assume_role_policy = jsonencode(
    {
      + Statement = [
        + {
          + Action    = "sts:AssumeRole"
          + Effect    = "Allow"
          + Principal = {
            + Service = "ec2.amazonaws.com"
          }
          + Sid      = ""
        },
      ],
    }
  + Version = "2012-10-17"
)
+ create_date      = (known after apply)
+ force_detach_policies = false
+ id               = (known after apply)
+ managed_policy_arns = (known after apply)
+ max_session_duration = 3600
+ name             = "ec2_role_clo_tfdc_oxclo01"
```

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

This is the standard “wordpress” docker-compose:

```
1  version: "3.9"
2
3  services:
4    db:
5      image: mysql:5.7
6      volumes:
7        - db_data:/var/lib/mysql
8      restart: always
9      environment:
10       MYSQL_ROOT_PASSWORD: somewordpress
11       MYSQL_DATABASE: wordpress
12       MYSQL_USER: wordpress
13       MYSQL_PASSWORD: wordpress
14
15    wordpress:
16      depends_on:
17        - db
18      image: wordpress:latest
19      volumes:
20        - wordpress_data:/var/www/html
21      ports:
22        - "80:80"
23      restart: always
24      environment:
25       WORDPRESS_DB_HOST: db:3306
26       WORDPRESS_DB_USER: wordpress
27       WORDPRESS_DB_PASSWORD: wordpress
28       WORDPRESS_DB_NAME: wordpress
29  volumes:
30    db_data: {}
31    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:

Instance state: running

Clear filters

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check
<input type="checkbox"/>	oxclo01-tf	i-05b651ab70c7aaf50	Running	t3.micro	Initializing

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
```



```
oxclo@oxclo: ~/clo-tf-docker
module.ec2_sg.aws_security_group_rule.ingress_rules[0]: Refreshing state... [id=
sgrule-4173508200]
module.dev_ssh_sg.aws_security_group_rule.ingress_rules[0]: Refreshing state...
[id=sgrule-1327652232]
aws_instance.web: Refreshing state... [id=i-0583c52725c3085ad]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the
last "terraform apply":

# aws_iam_instance_profile.ec2_profile_clo_tfdc has been changed
~ resource "aws_iam_instance_profile" "ec2_profile_clo_tfdc" {
  id          = "ec2_role_clo_tfdc_oxclo01"
  name        = "ec2_role_clo_tfdc_oxclo01"
+ tags       = {}
  # (6 unchanged attributes hidden)
}
# aws_instance.web has been changed
~ resource "aws_instance" "web" {
  id          = "i-0583c52725c3085ad"
  tags       = {
    "Name" = "oxclo01-tf"
  }
  # (31 unchanged attributes hidden)

  ~ root_block_device {
    + tags = {}
    # (8 unchanged attributes hidden)
  }
  # (4 unchanged blocks hidden)
}

Unless you have made equivalent changes to your configuration, or ignored the
relevant attributes using ignore_changes, the following plan may include
actions to undo or respond to these changes.
```

```
No changes. Your infrastructure matches the configuration.

Your configuration already matches the changes detected above. If you'd like to
update the Terraform state to match, create and apply a refresh-only plan:
  terraform apply -refresh-only
oxclo@oxclo:~/clo-tf-docker$
```


17. You can delete everything with:

terraform destroy -auto-approve

```
oxclo@oxclo: ~/clo-tf-docker

Plan: 0 to add, 0 to change, 10 to destroy.

Changes to Outputs:
  - instance_public_ip = {
    - public_ip = "34.243.7.38"
  } -> null
module.ec2_sg.aws_security_group_rule.ingress_rules[0]: Destroying... [id=sgrule-4173508200]
module.ec2_sg.aws_security_group_rule.egress_rules[0]: Destroying... [id=sgrule-3369823001]
module.ec2_sg.aws_security_group_rule.ingress_rules[2]: Destroying... [id=sgrule-1949751895]
module.dev_ssh_sg.aws_security_group_rule.ingress_rules[0]: Destroying... [id=sgrule-1327652232]
module.ec2_sg.aws_security_group_rule.ingress_rules[1]: Destroying... [id=sgrule-980022640]
aws_instance.web: Destroying... [id=i-0583c52725c3085ad]
module.dev_ssh_sg.aws_security_group_rule.ingress_rules[0]: Destruction complete after 1s
module.ec2_sg.aws_security_group_rule.ingress_rules[2]: Destruction complete after 1s
module.ec2_sg.aws_security_group_rule.ingress_rules[0]: Destruction complete after 1s
module.ec2_sg.aws_security_group_rule.egress_rules[0]: Destruction complete after 2s
module.ec2_sg.aws_security_group_rule.ingress_rules[1]: Destruction complete after 3s
aws_instance.web: Still destroying... [id=i-0583c52725c3085ad, 10s elapsed]
aws_instance.web: Still destroying... [id=i-0583c52725c3085ad, 20s elapsed]
aws_instance.web: Still destroying... [id=i-0583c52725c3085ad, 30s elapsed]
aws_instance.web: Still destroying... [id=i-0583c52725c3085ad, 40s elapsed]
aws_instance.web: Destruction complete after 42s
module.dev_ssh_sg.aws_security_group.this_name_prefix[0]: Destroying... [id=sg-013fc4b69067c3d19]
aws_iam_instance_profile.ec2_profile_clo_tf_dc: Destroying... [id=ec2_role_clo_tf_dc_oxclo01]
module.ec2_sg.aws_security_group.this_name_prefix[0]: Destroying... [id=sg-0f87079ff756e95a4]
module.dev_ssh_sg.aws_security_group.this_name_prefix[0]: Destruction complete after 1s
module.ec2_sg.aws_security_group.this_name_prefix[0]: Destruction complete after 2s
aws_iam_instance_profile.ec2_profile_clo_tf_dc: Destruction complete after 2s
aws_iam_role.ec2_role_clo_tf_dc: Destroying... [id=ec2_role_clo_tf_dc_oxclo01]
aws_iam_role.ec2_role_clo_tf_dc: Destruction complete after 1s

Destroy complete! Resources: 10 destroyed.
oxclo@oxclo:~/clo-tf-docker$
```

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.