

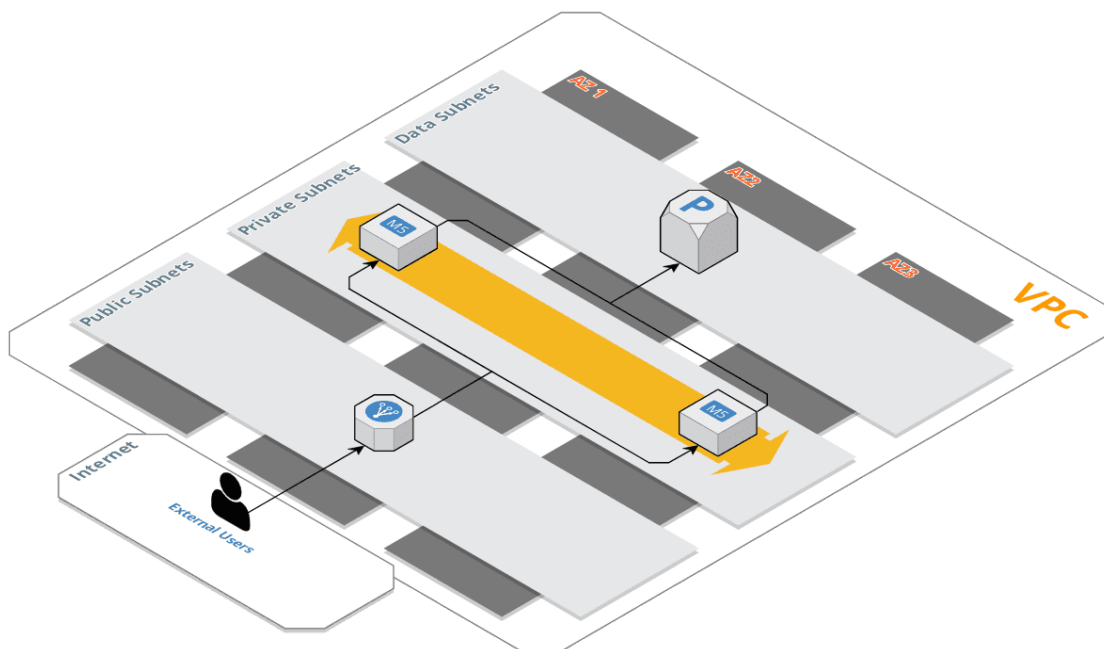


COSC2759 Tutorial/Lab Week 6

Goals of this lab

In this tutorial we will be implementing what was showed in the week 5 lecture to get a hands-on experience with AWS and IaC using Terraform, if you did not install Terraform in week 2, then please follow Tutorial Week 2, to install Terraform first.

We will be deploying an application in AWS and all the required supporting networking infrastructure that is required.



Set up

Log into your AWS Educate account and select your classroom

<https://www.awseducate.com/signin/SiteLogin>





Once you've signed in, go to your classroom

<https://www.awseducate.com/educator/s/educator-classrooms>

Find "Systems Deployment and Operations" from the list of Classrooms where I am a student, and select "Go to Classroom"

In the new window that opens, press the "Account Details" button to access your CLI credentials.

Press the show button on the window that opens up

```
Credentials
AWS Access
Session started at: 2020-04-02T01:29:50-0700
Session to end at: 2020-04-02T04:29:50-0700
Remaining session time: 2h58m14s

Term: 106 days 16:04:06

AWS CLI:
Copy and paste the following into ~/.aws/credentials

[default]
aws_access_key_id=
aws_secret_access_key=
aws_session_token=
```

Credentials set up option 1

Set up the default AWS credentials using the aws credentials file

Copy/paste the contents in the grey box into a file in your home directory "~/.aws/credentials"

Credentials setup option 2

An alternative way to do this, is to use environment variables in the active shell session instead of the default credentials.

Create a file called rmit-creds.sh in your home folder and add this content to it





```
#!/bin/bash
```

```
export AWS_ACCESS_KEY_ID=<insert key id from grey box>
```

```
export AWS_SECRET_ACCESS_KEY=<insert access key from grey box>
```

```
export AWS_SESSION_TOKEN=<insert token from grey box>
```

```
export AWS_DEFAULT_REGION=us-east-1
```

Now on each the terminal session you are going to run terraform, you can type in

```
$> source ~/remit-creds.sh
```

And the environment variables will be set for you in that shell session

More details on authentication for terraform can be found here:

<https://www.terraform.io/docs/providers/aws/index.html#authentication>

Set up Repo

Create a new folder in your home directory (or where you would like your git repository to be) called week6tutorial.

Change into the directory and create a new .gitignore file

```
$> mkdir week6tutorial
```

```
$> cd week6tutorial
```

```
$> touch .gitignore
```

Open .gitignore in your ide (vs code) and add this content to prevent us from accidentally checking in the terraform cache directory and the different state files

```
.terraform
```

```
terraform.tfstate
```

```
.terraform.tfstate.lock.info
```

```
terraform.tfstate.backup
```



Create a new file called main.tf that will contain our terraform code

```
$>touch main.tf
```

Open main.tf in your IDE (vs code) and add the aws provider code block and save the file

```
provider "aws" {  
  version = "~> 2.23"  
  region  = "us-east-1"  
}
```

Initiate the git repository and commit the .gitignore and terraform file as the initial commit

```
$> git init
```

```
$> git add .
```

```
$> git commit -m "Initial commit"
```

Last thing we will do as part of the setup, is to initialise the terraform repository so it's ready to execute

```
$> terraform init
```

You should now be ready to start working with terraform and deploy resources using IaC.

But please remember, each student has \$50 of credit, so please remember to delete the resources you deploy, or you will run out.

```
$> terraform destroy
```





Virtual Private Cloud (VPC)

Like in the previous tutorial, we will start by creating a VPC in our AWS accounts. We will have to create the network that we will be deploying into.

So to do that we need to create the VPC resource in terraform

Doco: <https://www.terraform.io/docs/providers/aws/r/vpc.html>

1. Open main.tf and add the basic usage example to your file from the doco above.
2. We will also add a tag to the VPC while we are modifying this so we can easily find it in the console later.

Add the tag "Name" with the value "Tutorial Week 6"

The block should look like this when you are complete

```
resource "aws_vpc" "main" {  
  cidr_block = "10.0.0.0/16"  
  
  tags = {  
    Name = "Tutorial Week 6"  
  }  
}
```

3. Let's validate that the terraform code we wrote is correct and format it so indentations and styling is correct

```
$> terraform validate
```


```
$> terraform fmt
```

Fix any errors you see pop up

4. Next up we should run a terraform plan and see what will change in our environment before we run terraform apply

```
$> terraform plan
```





your output should look like this:

```
-----  
An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_vpc.main will be created  
+ resource "aws_vpc" "main" {  
  + arn                               = (known after apply)  
  + assign_generated_ipv6_cidr_block = false  
  + cidr_block                        = "10.0.0.0/16"  
  + default_network_acl_id           = (known after apply)  
  + default_route_table_id           = (known after apply)  
  + default_security_group_id        = (known after apply)  
  + dhcp_options_id                  = (known after apply)  
  + enable_classiclink                = (known after apply)  
  + enable_classiclink_dns_support   = (known after apply)  
  + enable_dns_hostnames              = (known after apply)  
  + enable_dns_support                = true  
  + id                               = (known after apply)  
  + instance_tenancy                 = "default"  
  + ipv6_association_id               = (known after apply)  
  + ipv6_cidr_block                   = (known after apply)  
  + main_route_table_id               = (known after apply)  
  + owner_id                         = (known after apply)  
  + tags                             = {  
    + "Name" = "Tutorial Week 6"  
  }  
}
```

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

5. Apply the changes and log into the console to verify that it was created. (review the tutorial sheet from tutorial week 5 for the steps to get into the console)


\$> terraform apply

Internet Gateway

Now that we have the VPC created, we need to create an internet gateway so we can connect to and from the services we deploy in our VPC

Doco: https://www.terraform.io/docs/providers/aws/r/internet_gateway.html

6. As before, copy paste the example from the doco and change the Tag to be "Tutorial Week 6" and it should look like this:



```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "Tutorial Week 6"
  }
}
```

7. Do a terraform validate, format and then a plan

\$> terraform validate

\$> terraform fmt

\$> terraform plan

Verify that the plan outputs only creating an internet gateway and that the tag is correct

8. Apply the changes using terraform. We will use the auto-approve flag here as well, so you don't have to type yes every time

\$> terraform apply --auto-approve

9. Verify in the console that it was created correctly

Default Route table


We need to update the default route table in the VPC so AWS knows to send internet bound traffic to the internet gateway. We do that by creating a default route table resource

Doco: https://www.terraform.io/docs/providers/aws/r/default_route_table.html

10. Add the example to your main.tf file and update it so the reference to the VPC is correct and add the route that we need

cidr_block should be "0.0.0.0/0" which indicates internet bound traffic

gateway_id should be a reference to the internet gateway that deployed earlier



```
resource "aws_default_route_table" "main" {
  default_route_table_id = aws_vpc.main.default_route_table_id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "Tutorial Week 6 default table"
  }
}
```

11. Run validate, fmt, and plan to validate your terraform and make sure the expected changes will occur when we run apply

```
$> terraform validate
```

```
$> terraform fmt
```

```
$> terraform plan
```

12. Fix any issues you see and validate that the default route table resource will be created

13. Apply your changes and validate that the deployment worked in the AWS console

```
$> terraform apply --auto-approve
```

Subnets

At this point we have everything ready for deployment except for somewhere to place the VMs, to do that we need to deploy subnets. We will deploy 9 of them in total, 3 in each availability zone so we have redundancy in case there is an outage in AWS.

We will deploy a standard three layer structure, public, private, data


Doco: <https://www.terraform.io/docs/providers/aws/r/subnet.html>

14. From the doco, make a subnet block in main.tf, and give these settings:

```
local name: public_az1
```

```
cidr: 10.0.0.0/22
```





az: us-east-1a
public ip: true
tag: name = Public AZ1


Should look similar to this when complete

```
resource "aws_subnet" "public_az1" {  
  vpc_id           = aws_vpc.main.id  
  cidr_block       = "10.0.0.0/22"  
  availability_zone = "us-east-1a"  
  map_public_ip_on_launch = true  
  
  tags = {  
    Name = "Public AZ1"  
  }  
}
```

15. Validate, format and run a plan to make sure what the expected outcome will be when you apply

\$> terraform validate
\$> terraform fmt
\$> terraform plan

Output of plan should look like this, with a different vpc_id



```
-----
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_subnet.public_az1 will be created
+ resource "aws_subnet" "public_az1" {
  + arn                               = (known after apply)
  + assign_ipv6_address_on_creation = false
  + availability_zone                = "us-east-1a"
  + availability_zone_id             = (known after apply)
  + cidr_block                       = "10.0.0.0/22"
  + id                              = (known after apply)
  + ipv6_cidr_block                  = (known after apply)
  + ipv6_cidr_block_association_id = (known after apply)
  + map_public_ip_on_launch         = true
  + owner_id                        = (known after apply)
  + tags                            = {
    + "Name" = "Public AZ1"
  }
  + vpc_id                          = "vpc-0358ca07e0be58583"
}
```

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

16. Apply the changes to your environment, and verify that the changes were applied successfully in the AWS Console

```
$> terraform apply --auto-approve
```

17. Next up please add another 8 subnets with these details

Local name	CIDR	AZ	Tag
public_az2	10.0.4.0/22	us-east-1b	Public AZ2
Public_az3	10.0.8.0/22	Us-east-1c	Public AZ3
Private_az1	10.0.16.0/22	Us-east-1a	Private AZ1
Private_az2	10.0.20.0/22	Us-east-1b	Private AZ2
Private_az3	10.0.24.0/22	Us-east-1c	Private AZ3
Data_az1	10.0.32.0/22	Us-east-1a	Data AZ1
Data_az2	10.0.36.0/22	Us-east-1b	Data AZ2
Data_az3	10.0.40.0/22	Us-east-1c	Data AZ3



18. Run validate, format, and plan on your changes to make sure it does what it is supposed to

```
$> terraform validate  
$> terraform fmt  
$> terraform plan
```

19. Fix any issues and apply them to your environment. Validate in the AWS Console after applying to make sure everything is in order

```
$> terraform apply --auto-approve
```

Application Deployment

We now have an environment to deploy our application into, so the next part we will look at is deploying the application. As we did in the earlier tutorial, we will deploy the TechTestApp provided in an AMI in an auto-scaling group with a load balancer in front, backed by an RDS database.

Key Pair

First thing we need to set up, is a key pair to make sure you can log into the instance. You should already have an SSH key set up for you to access GitHub in your ubuntu VM, so we will reuse this key.

Doco: https://www.terraform.io/docs/providers/aws/r/key_pair.html

We're assuming you used the default name (id_rsa) for your key files, if that is not the case, please replace with the name you used.

20. Create a new file in your folder called "terraform.tfvars" to help us pass in variables


```
$> touch terraform.tfvars
```

21. Copy the content of ~/.ssh/id_rsa.pub and place it in the terraform.tfvars file

add a line saying public_key = "key"

```
public_key = "<paste in key here>"
```

make sure there is no new-line in your key and save



That variable will be passed in every time you run terraform

22. Next we define the variable input that we want. We will do that in a new file to keep the terraform clean. Create a new file called “variables.tf” and register a variable called public_key of type string

\$> touch variables.tf

```
variable "public_key" {  
  type = string  
}
```

23. Next up we will create a new file to keep our app infrastructure in as well. Create a new file called app.tf

\$> touch app.tf

24. Copy the example into your new file and replace the public_key string with the variable reference

```
resource "aws_key_pair" "deployer" {  
  key_name    = "deployer-key"  
  public_key = var.public_key  
}
```

25. Run terraform validate, format, and plan to verify the changes

\$> terraform validate

\$> terraform fmt

\$> terraform plan

fix any issues you find

26. Apply your changes and validate that the key was created properly in the AWS console

\$> terraform apply --auto-approve

Security Group

First thing we will do is set up the security group for our application, we need to configure this so we can add it to the instances as they are created as part of the auto scaling group.



Doco: https://www.terraform.io/docs/providers/aws/r/security_group.html

We will need to provide access for to SSH and to HTTP on these instances, so we can get to the web site, and so we can log in and run the database update script.

27. Copy the example from the doco as a starting point, and add two ingress rules, one for port 80 and one for port 22, allowing full access from the internet.

Name it "allow_http_ssh" and give it a good description

```
resource "aws_security_group" "allow_http_ssh" {
  description = "Allow SSH and http inbound traffic"
  vpc_id      = aws_vpc.main.id

  ingress {
    description = "ssh from internet"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "http from internet"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }


  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "allow_http_ssh"
  }
}
```

28. Run terraform validate, format and plan to check that everything is correct and the expected changes will get applied

\$> terraform validate





```
$> terraform fmt
$> terraform plan
```

Fix an issues

29. Apply the changes and validate that they got deployed as expected in the AWS Console

```
$> terraform apply --auto-approve
```

Launch Configuration

Next up we will create the configuration for launching new EC2 instances in the Auto Scaling group.

Doco: https://www.terraform.io/docs/providers/aws/r/launch_configuration.html

30. A launch configuration needs to know which AMI to deploy, so we will add that into the terraform.tfvars file and create a new variable in the variables.tf file

Add a new entry in terraform.tfvars with ami_id as the key and "ami-055d18a0205f9ddb1" as the value

```
ami_id = "ami-055d18a0205f9ddb1"
```

31. Open the variables.tf file and add a new variable called "ami_id" of type string

```
variable "ami_id" {
  type = string
}
```

32. Next, open up app.tf and add the launch configuration based on the resource section in the example in the doco.

make sure the image_id point to the variable reference to ami_id

Also add in the security group we created and the key_pair



```
resource "aws_launch_configuration" "todo_app" {
  name           = "web_config"
  image_id       = var.ami_id
  instance_type  = "t2.micro"
  security_groups = [aws_security_group.allow_http_ssh.id]

  key_name = aws_key_pair.deployer.key_name
}
```

33. Run terraform validate, format and plan to check that everything is correct and the expected changes will get applied

```
$> terraform validate
```

```
$> terraform fmt
```

```
$> terraform plan
```

Fix an issues

34. Apply the changes and validate that they got deployed as expected in the AWS Console

```
$> terraform apply --auto-approve
```

Target Group


Next we will create the target group for the load balancer, as we need to register that with the auto scaling group so we will create all the dependencies first.

Doco: https://www.terraform.io/docs/providers/aws/r/lb_target_group.html

35. We will be using an instance target group, so use the instance target group example in the doco as your base and create a new target group in app.tf

call it "todo_app" and name it "todo-app-target-group"

app will be using http on port 80, so add that too



```
resource "aws_lb_target_group" "todo_app" {
  name      = "todo-app-target-group"
  port      = 80
  protocol  = "HTTP"
  vpc_id    = aws_vpc.main.id
}
```

36. Run terraform validate, format and plan to check that everything is correct and the expected changes will get applied

```
$> terraform validate
```

```
$> terraform fmt
```

```
$> terraform plan
```

Fix an issues

37. Apply the changes and validate that they got deployed as expected in the AWS Console

```
$> terraform apply --auto-approve
```

Auto Scaling Group

Now that we have created all the pre-requisites for the auto scaling group, we will create that and run some instances

Doco: https://www.terraform.io/docs/providers/aws/r/autoscaling_group.html

38. The example for the auto scaling group has a lot of extra variables that won't need, so please use it as a rough guide as you create the terraform for the auto scaling group. It should look like this at the end:

```
resource "aws_autoscaling_group" "todo_app" {
  name                  = "terraform-asg-example"
  launch_configuration = aws_launch_configuration.todo_app.name
  min_size              = 1
  max_size              = 1
  vpc_zone_identifier   = [aws_subnet.private_az1.id, aws_subnet.private_az2.id, aws_subnet.private_az3.id]
  target_group_arns    = [aws_lb_target_group.todo_app.arn]
}
```

It references both the target groups and launch configuration. We are also setting which subnets the instances will be created in.



The rest will be left to default values

39. Run terraform validate, format and plan to check that everything is correct and the expected changes will get applied

```
$> terraform validate
```

```
$> terraform fmt
```

```
$> terraform plan
```

Fix an issues

40. Apply the changes and validate that they got deployed as expected in the AWS Console

```
$> terraform apply --auto-approve
```

Load Balancer

Next we will deploy in this tutorial is the load balancer, so we can connect to the servers as they are scaled up and down without knowing the exact ip of the server.

Doco: <https://www.terraform.io/docs/providers/aws/r/lb.html>

41. We will create an application load balancer, review the example in the doco and set up a load balancer in your app.tf file

make sure to make it external, and assign the security group we created earlier for. Also assign it to be deployed in the three public subnets


```
resource "aws_lb" "todo_app" {  
  name           = "todo-app-lb"  
  internal       = false  
  load_balancer_type = "application"  
  security_groups = [aws_security_group.allow_http_ssh.id]  
  subnets       = [aws_subnet.public_az1.id, aws_subnet.public_az2.id, aws_subnet.public_az3.id]  
  
  tags = {  
    Environment = "production"  
  }  
}
```

42. Run terraform validate, format and plan to check that everything is correct and the expected changes will get applied

```
$> terraform validate
```

```
$> terraform fmt
```





```
$> terraform plan
```

Fix an issues

43. Apply the changes and validate that they got deployed as expected in the AWS Console

```
$> terraform apply --auto-approve
```

Load Balancer Listener

The last resource we will be deploying in this tutorial, is the listener for the load balancer. It is used to define the routing, and ties the port and protocol to the instances in the target group

Doco: https://www.terraform.io/docs/providers/aws/r/lb_listener.html

44. Review the example in the doco and add a listener for port 80 in the app.tf file

set the default action to be forward, and add the target group arn reference


```
resource "aws_lb_listener" "front_end" {  
  load_balancer_arn = aws_lb.todo_app.arn  
  port              = "80"  
  protocol          = "HTTP"  
  
  default_action {  
    type              = "forward"  
    target_group_arn = aws_lb_target_group.todo_app.arn  
  }  
}
```


45. Run terraform validate, format and plan to check that everything is correct and the expected changes will get applied

```
$> terraform validate
```

```
$> terraform fmt
```

```
$> terraform plan
```





Fix an issues

46. Apply the changes and validate that they got deployed as expected in the AWS Console

```
$> terraform apply --auto-approve
```

47. If you go to the DNS address of the load balancer you should be able to see the application

48. Last thing we will do is to create an output, so the DNS address is available from the command line

Create a new terraform file called outputs.tf

```
$> touch outputs.tf
```

49. Add a new output to the file called “endpoint” and give it the value referencing the load balancer dns_name variable

```
output "endpoint" {  
  value = aws_lb.todo_app.dns_name  
}
```

50. Run terraform apply one last time and you should see the dns endpoint printed to your screen

```
$> terraform apply --auto-approve
```

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.  
  
Outputs:  
  
endpoint = todo-app-lb-721677880.us-east-1.elb.amazonaws.com
```

Remember that once you’ve completed and tested all your changes, please tear down your resources so you don’t spend all your credit?

```
$> terraform destroy
```



Extra

You should have a basic understanding of terraform and how that works, if you finish early, have a look at how to deploy RDS instances using terraform. It will help you in your next assessment.

Starting point – doco: https://www.terraform.io/docs/providers/aws/r/db_instance.html

