AWS Academy Introduction

Software Architecture

February 20, 2023

Brae Webb

1 Before Class

It's preferable if you already have terraform installed[1]. Please also have one of IntelliJ IDEA, PyCharm, or VSCode with the terraform plugin installed.

2 This Week

This week our goal is to get acquainted with AWS Academy. Throughout the course we will use AWS Academy to learn how to deploy and manage infrastructure with AWS. Specifically, this week you need to:

- Enrol in
  1. AWS Academy Cloud Foundations [10906] course;
  2. AWS Academy Learner Lab - Foundation Services [10909] course; and
- Navigate the AWS Academy interface.
- Enter the AWS Console from an AWS Academy lab.
- Use the AWS Console to provision an EC2 instance with a simple web interface.

We then will start getting experience using an Infrastructure as Code tool, specifically, Terraform, to deploy a service to AWS. Specifically, this week you need to:

- Authenticate Terraform to use the AWS learner lab.
- Configure a single server website in Terraform and deploy.
- Create a Terraform module for deploying arbitrary single server websites.

Info

This week's practical has a lot of content so dont feel pressured to complete the tasks and instead it might be better to focus on the tutors walkthrough. Next week we will be deploying our todo application which will recover this content in more depth and you will have the entire practical to complete the task.

3 AWS Academy

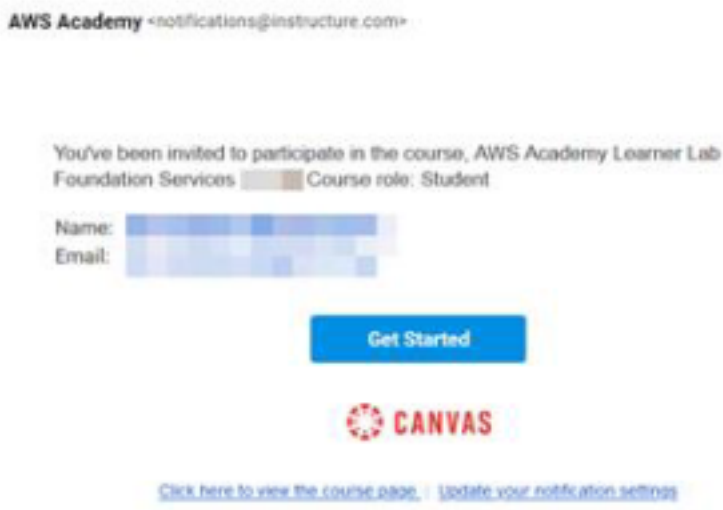
AWS Academy is an educational platform to support teaching AWS services. In this course, we will be using it in two ways:

[1] https://learn.hashicorp.com/tutorials/terraform/install-cli

1. The Cloud Foundations course will be used as supplementary material to help cement your ability to use AWS. The use of Cloud Foundations is completely optional.
2. The Learner Lab provides access to an environment which will be used in these practicals to learn AWS.

4 Enrol in AWS Academy

1. Set up your AWS Academy account by responding to your email invitation and clicking **Get Started**. The email invitation will come from AWS Academy. Check your junk/spam folders.



2. Go to `https://www.awsacademy.com/LMS_Login` to login.
   (a) Press **Student Login**.
   (b) Use the email address that received the email invitation.

5 Exploring the Interface

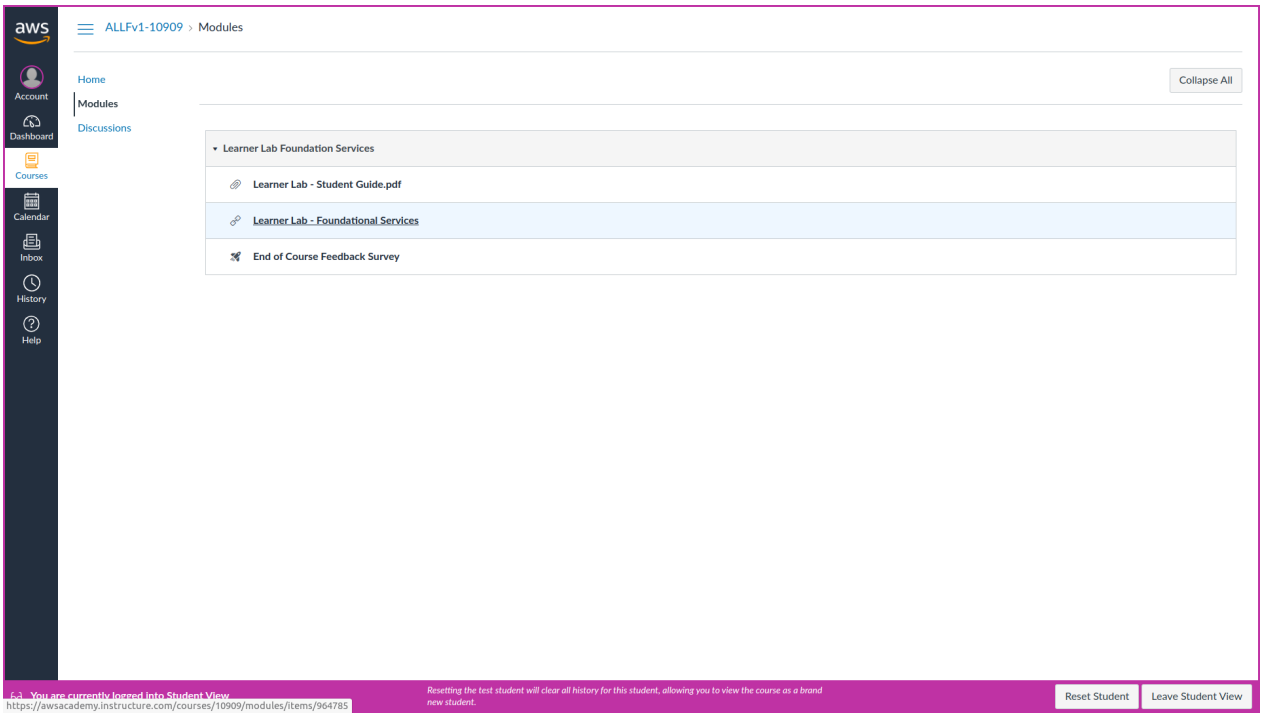
Aside

We will just be looking at the learner lab today, please ask on the slack if you need help using the cloud foundations course.

The following steps will enter the learner lab:

1. Once you have enrolled in the course, you should see this course page (minus everything in pink):

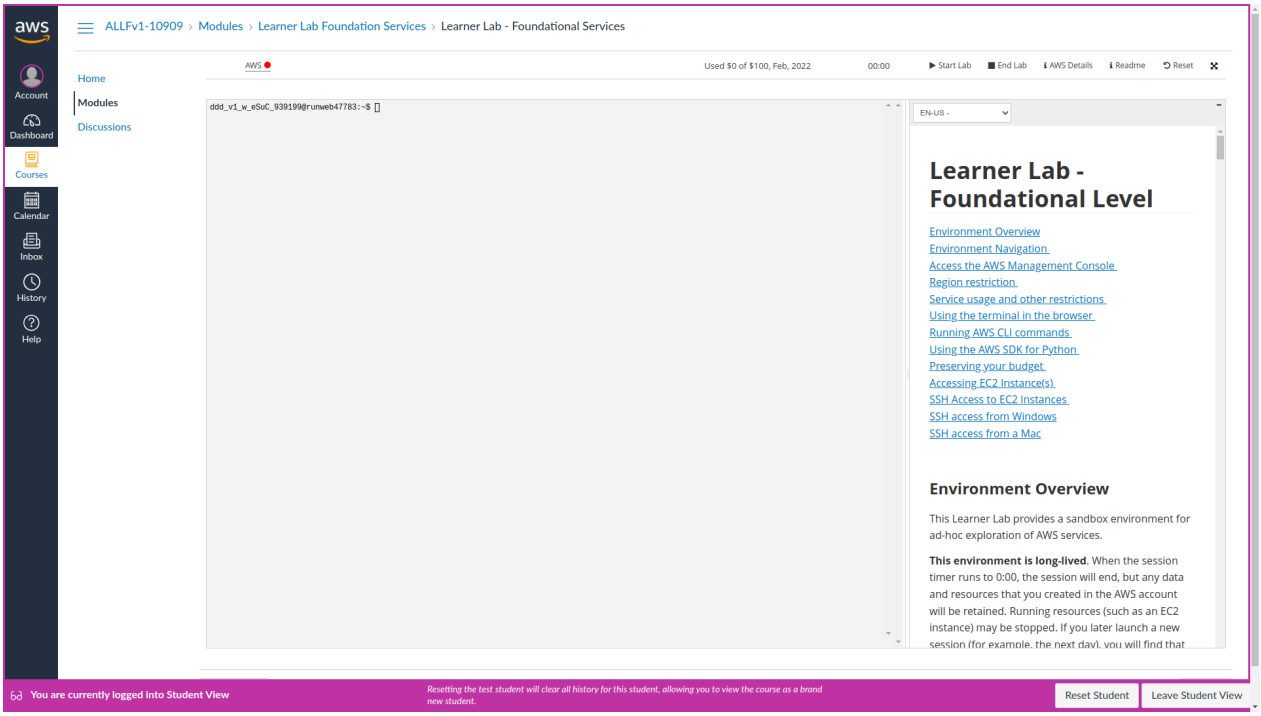
2. Navigate to the `Modules` tab and select the link for "Learner Lab - Foundational Services". You may also open and browse the "Learner Lab - Student Guide.pdf" link which covers some of the content of this guide.



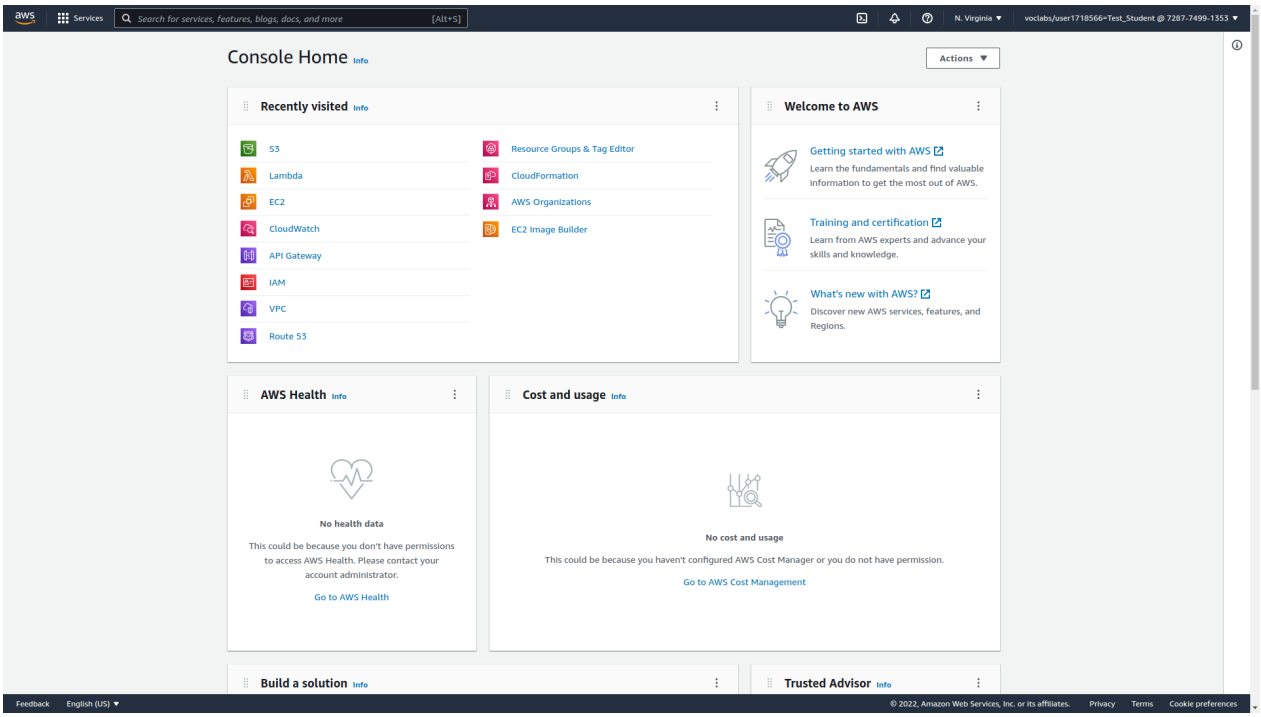
3. Now we have entered the main part of the interface, the learner lab.

   - The AWS text with the currently red circle is the link to open the AWS console.
   - You also see your budget, note that the budget is not updated in real-time.
   - The `00:00` indicates how long your lab has been activated. A lab can only remain active for 6 hours, after which it will be closed, unless you press start lab again before the 6 hours expires.

- AWS details will become important later but are not needed now.
- The README button will re-open the text panel currently on the right of the terminal interface.
- The terminal interface is not yet important.
- The right-hand has a lot of important information including what AWS services are available in the learner labs environment, please read it.



4. Go ahead and start the lab. It will take a few moments to get ready and the red circle will turn green once it is. Click on the green circle when it is available. This will open the AWS Console in a new tab.[2] If you end up working in a company which uses AWS, welcome to your new home.



[2] Your view will be different as you will not yet have any recently visited services.

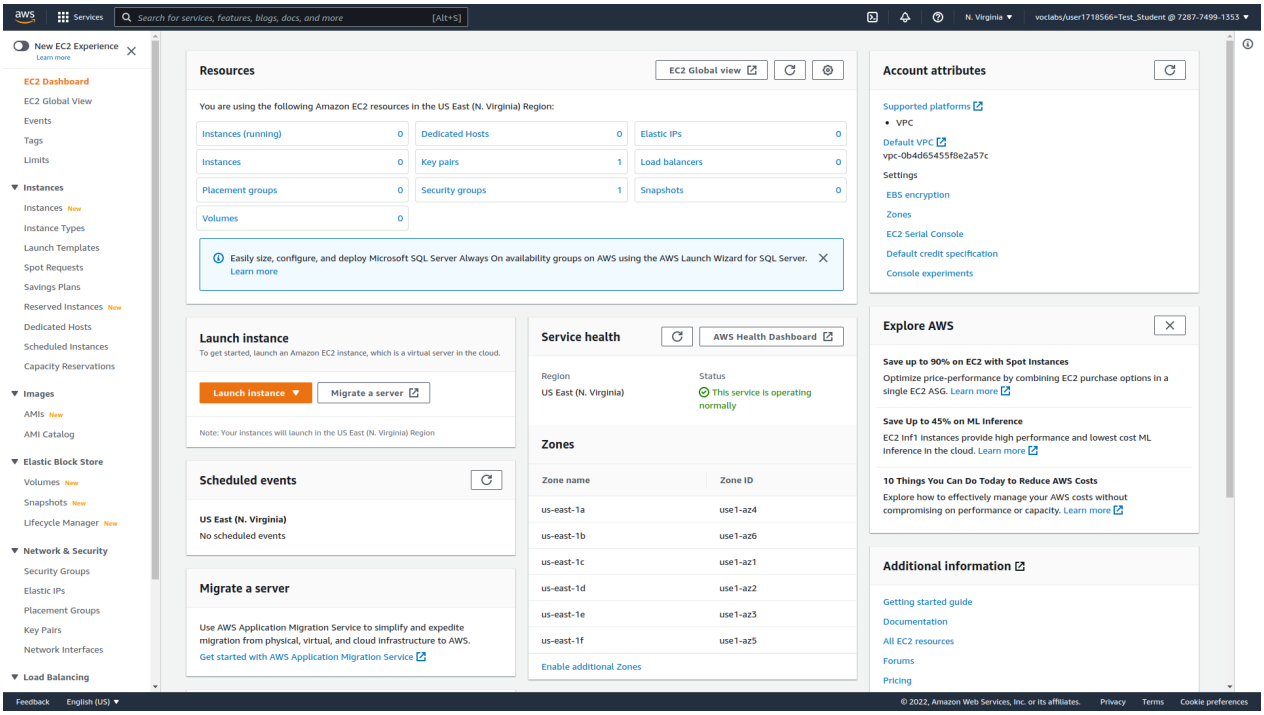
Aside

A short introduction to AWS: Amazon Web Services (AWS) is an Infrastructure as a Service (IaaS) and Software as a Service (SaaS) provider. They offer a collection of services which are helpful for development. For example, they offer virtual compute resources, database storage options, and networking to tie it all together. Services are offered with a pay as you go model, meaning you only pay for the seconds you use a service. We will get acquainted with some simple services offered by AWS now.

6 AWS EC2

Today we are going to focus on using AWS's EC2 service. Elastic Compute Cloud (EC2) is the primary compute service offered by AWS. It allows you to create a linux virtual machine on Amazon's infrastructure. You have full control over this machine and can configure it for whatever purpose you need.

Navigate to the search bar in the top left and find the EC2 service. You might find this interface overwhelming. It is important to note that since EC2 is one of the primary services offered by AWS, many smaller services we do not need are bundled into the service.



Today, we will just need the `Instances` dashboard. Navigate to there and select "Launch instances".

6.1 EC2 AMI

First we will need to select an Amazon Machine Image (AMI). An AMI is the template (cookie cutter) which provides instructions on how an instance should be provisioned. Amazon offers a range of built-in AMIs. There are also community AMIs or you can create your own. As we just want a simple server today, we will use one of the built-in AMIs.

Every AMI has a unique AMI code, something that looks like `ami-0a8b4cd432b1c3063`. We will use this AMI today[3] , it is considered one of the fundamental images. Search for it via the code and select it.

[3]Check the requirements for your chosen website, if your website requires a specific operating system, look for it as a community AMI. Ask for help if you aren't sure.

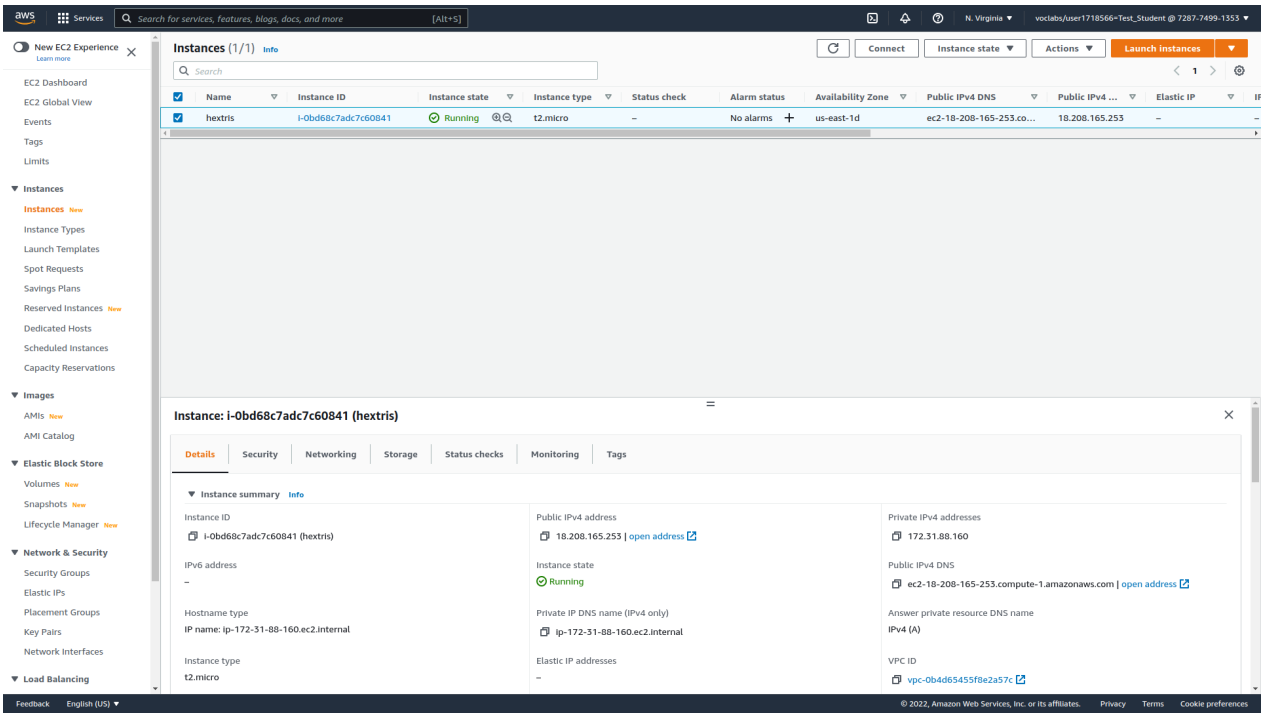
6.2 Instance Settings

The next few settings for configuring your instance are:

1. We do not need a large server, choose `t2.micro`.
2. Keep the 'Configure Instance' settings as default.
3. Keep the 'Add Storage' settings as default.
4. Add a 'Name' tag, call it the name of your website, e.g. `hextris`.
5. Choose 'Create a **new** security group'.
   (a) Give it a meaningful name, e.g. `hextris-http-ssh-access`.
   (b) Change the description to something meaningful, e.g. `Hextris HTTP and SSH access`.
   (c) Add a rule, select the `Type` as `HTTP`.
6. When prompted to choose a key pair (after the clicking `Launch` button), select the existing `vockey | RSA` option.

7 Accessing the Instance

Return to the `Instances` dashboard. You should now see a new instance created, its instance state might not yet be `Running`, if not, wait.



Note the public IPv4 address, we will need to use this to connect to the server.

Notice

For terminal examples in this section, lines that begin with a $ indicate a line which you should type while the other lines are example output that you should expect. Not all of the output is captured in the examples to save on space.

1. Return to the learner lab interface.

2. Run the following, replacing `127.0.0.1` with the public IP of your instance. This command uses the `vockey | RSA` key pair to gain SSH access to the machine.

```
$ ssh -i ~/.ssh/labsuser.pem ec2-user@127.0.0.1
```

example:

```
$ ssh -i ~/.ssh/labsuser.pem ec2-user@35.173.230.42
The authenticity of host '35.173.230.42 (35.173.230.42)' can not be established.
ECDSA key fingerprint is SHA256:W7OzzZm6nhwM9tB9Kb7enONPryo1a4259hJmSAZX7HQ.
Are you sure you want to continue connecting (yes/no)?

# At this point you will want to type yes and press enter
Warning: Permanently added '35.173.230.42' (ECDSA) to the list of known hosts.
Connection to 35.173.230.42 closed by remote host.
Connection to 35.173.230.42 closed.
```

8 Installing Hextris

Hextris is very simple to install, using an EC2 interface is perhaps overkill for it. It is an entirely client-side application which means we just have to serve the static files.

First, we will need to enable the basic serving of static files. We can install and start the `httpd` service for this. The AMI we have picked uses the yum package manager, so to install httpd we run:

```
$ sudo yum install httpd
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
.....
.....
Total download size: 1.9 M
Installed size: 5.2 M
Is this ok [y/d/N]:

# enter y to install
.....
.....
Complete!
$ sudo systemctl enable httpd
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to
    /usr/lib/systemd/system/httpd.service.
$ sudo systemctl start httpd
```

All files in the `/var/www/html` directory will now be served when accessed via HTTP. Change to this directory and notice that it is currently empty.

Next, we need to download the static files to the server. We can use git for this, but first git needs to be installed on the machine.

```
$ sudo yum install git
```

Finally, let's double check we are in the `/var/www/html` directory.

```
$ cd /var/www/html
```

And clone the repository into that directory.

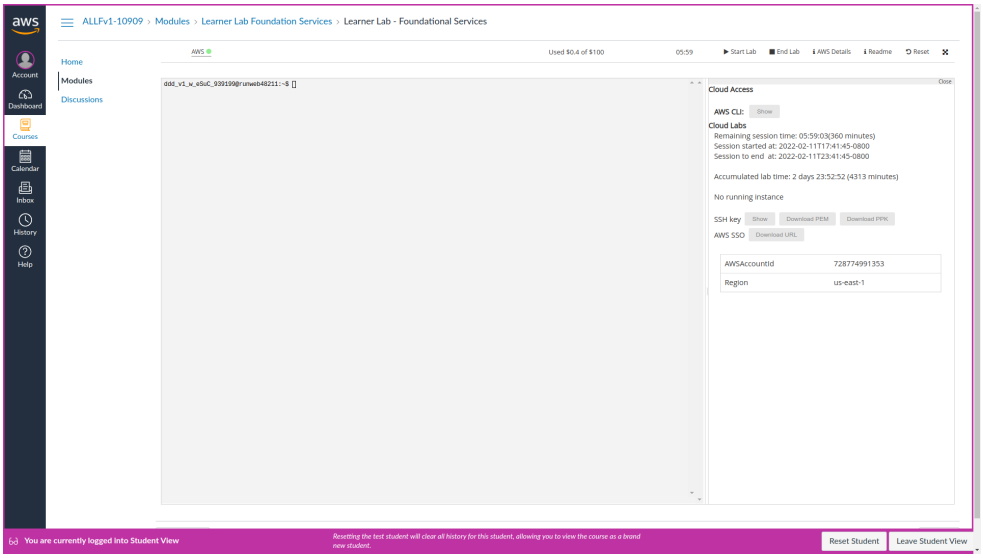
```
$ sudo git clone https://github.com/Hextris/hextris .
```

Now if you navigate to the **http** address of the public IP address (e.g. `http://18.208.165.253`), you should be able to see your newly deployed website.

9 Using Terraform in AWS Learner Labs

Following the steps from the week one practical, start a learner lab in AWS Academy. For this practical, you do not need to create any resources in the AWS Console. The console can be used to verify that Terraform has correctly provisioned resources.

1. Once the learner lab has started, click on 'AWS Details' to display information about the lab.



2. Click on the first 'Show' button next to 'AWS CLI' which will display a text block starting with `[default]`.
3. Create a directory for this week's practical.
4. Within that directory create a `credentials` file and copy the contents of the text block into the file. **Do not share this file contents — do not commit it.**

5. Create a `main.tf` file in the same directory with the following contents:

```
» cat main.tf
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
            version = "~> 3.0"
        }
    }
}

provider "aws" {
    region = "us-east-1"
    shared_credentials_file = "./credentials"
}
```

   The `terraform` block specifies the required external dependencies, here we need to use the AWS provider. The `provider` clock configures the AWS provider, instructing it which region to use and how to authenticate (using the credentials file we created).

6. We need to initialise terraform which will fetch the required dependencies. This is done with the `terraform init` command.

```
$ terraform init
```

   This command will create a `.terraform` directory which stores providers and a provider lock file, `.terraform.lock.hcl`.

7. To verify that we have setup Terraform correctly, use `terraform plan`.

```
$ terraform plan
```

   As we currently have no resources configured, it should find that no changes are required. Note that this does not ensure our credentials are correctly configured as Terraform has no reason to try authenticating yet.

10 Deploying Hextris

If you followed the default instructions in the week one practical, you would have manually deployed the hextris game. Now we'll try to deploy hextris using terraform.

First, we will need to create an EC2 instance resource. The AWS provider calls this resource an `aws_instance`[4]. Get familiar with the documentation page. Most Terraform providers have reasonable documentation, reading the argument reference section helps to understand what a resource is capable of.

[4] https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance

We will start off with the basic information for the resource. We configured it to use a specific Amazon Machine Instance (AMI) and chose the `t2.micro` size. Refer back to the week one practical sheet for a refresher. Add the following basic resource to `main.tf`:

```
   » cat main.tf
resource "aws_instance" "hextris-server" {
    ami = "ami-0a8b4cd432b1c3063"
    instance_type = "t2.micro"

    tags = {
        Name = "hextris"
    }
}
```

To create the server, invoke `terraform apply` will first do `terraform plan` and prompt us to confirm if we want to apply changes.

```
$ terraform apply
```

You should be prompted with something similar to the output below.

```
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_instance.hextris-server will be created
  + resource "aws_instance" "hextris-server" {
      + ami = "ami-0a8b4cd432b1c3063"
      (omitted)
      + instance_type = "t2.micro"
      (omitted)
      + tags = {
          + "Name" = "hextris"
        }
    }

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

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

If the plan looks sensible enter `yes` to enact the changes.

```
  Enter a value: yes

aws_instance.hextris-server: Creating...
aws_instance.hextris-server: Still creating... [10s elapsed]
aws_instance.hextris-server: Still creating... [20s elapsed]
aws_instance.hextris-server: Still creating... [30s elapsed]
aws_instance.hextris-server: Still creating... [40s elapsed]
aws_instance.hextris-server: Creation complete after 47s [id=i-08c92a097ae7c5b18]

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

You can now check in the AWS Console that an EC2 instance with the name `hextris` has been created. Now we have got a server, we should try to configure it to contain hextris. We will use the `user_data` field which configures commands to run when launching the instance. First we need a script to provision the server, if we combine all our commands from the last practical, we will produce this script:

```
» cat deploy.sh
#!/bin/bash
yum install -y httpd
systemctl enable httpd
systemctl start httpd

yum install -y git
cd /var/www/html
git clone https://github.com/Hextris/hextris .
```

Now we can add the following field to our Terraform resource. It uses the Terraform `file` function to load the contents of a file named `deploy.sh` relative to the Terraform directory. The contents of that file is passed to the `user_data` field.

```
user_data = file("${path.module}/deploy.sh")
```

If you run the `terraform plan` command now, you will notice that Terraform has identified that this change will require creating a new EC2 instance. Where possible, Terraform will try to update a resource in-place but since this changes how an instance is started, it needs to be replaced. Go ahead and apply the changes.

Now, in theory[5], we should have deployed hextris to an EC2 instance. But how do we access that instance? We *could* go to the AWS Console and find the public IP address. However, it turns out that Terraform already knows the public IP address. In fact, if you open the Terraform state file (`terraform.tfstate`), you should be able to find it hidden away in there. But we do not want to go hunting through this file all the time. Instead we will use the `output` keyword.

We can specify certain attributes as 'output' attributes. Output attributes are printed to the terminal when the module is invoked directly but as we will see later, they can also be used by other Terraform configuration files.

[5] hint

```
» cat main.tf
output "hextris-url" {
  value = aws_instance.hextris-server.public_ip
}
```

This creates a new output attribute, `hextris-url`, which references the `public_ip` attribute of our `hextris-server` resource. Note that resources in Terraform are addressed by the resource type (`aws_instance`) followed by the name of the resource (`hextris-server`).

If you plan or apply the changes, it should tell you the public IP address of the instance resource.

```
$ terraform plan
```

```
aws_instance.hextris-server: Refreshing state... [id=i-043a61ff86aa272e0]

Changes to Outputs:
  + hextris-url = "3.82.225.65"
```

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

So let's try and access that url, hmm. That's strange. Something has gone wrong.

11 Security Groups

As you will recall from the practical in week one. An important part of setting up the EC2 instance was configuring the security group to allow traffic from port 80 to reach the instance. Configuring the security group was built into the process with the GUI configuration. When configuring with Terraform, security groups and their attachment to EC2 instances are separate resources. Refer back to the Terraform documentation for details or, as is normally quicker, Google "terraform aws security group".

First, let us create an appropriate security group. Recall that in the GUI configuration, ingress SSH access (port 22) and all egress[6] traffic was automatically configured and we just added ingress port 80. In Terraform the whole state must be configured so we specify two `ingress` blocks one for HTTP (port 80) and one for SSH access (port 22).[7] Additionally, we will create egress for all outgoing traffic.

```
resource "aws_security_group" "hextris-server" {
  name = "hextris-server"
  description = "Hextris HTTP and SSH access"

  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
```

[6] Ingress and egress in networking just means incoming and outgoing respectively.

[7] We do not actually need SSH access as all the server configuration is done when the machine is provisioned thanks to the `user_data`, but we will try to recreate the instance from the week one practical exactly.

```
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 22
    to_port = 22
    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"]
  }
}
```

Note the following:

- `from_port` and `to_port` are the start and end of a range of ports rather than incoming or outgoing. In this example our range is 80-80.
- `protocol` set to -1 is a special flag to indicate all protocols.
- Explaining `cidr` is outside the scope of the course, but the specified block above means to apply to all IP addresses.

You may now apply the changes to create this new security group resource.

Next, we will attach the security group to the EC2 instance. Return to the `aws_instance.hextrix-server` resource and include the following line:

```
security_groups = [aws_security_group.hextris-server.name]
```

Note that EC2 instances can have multiple security groups. Once again notice the structure of resource identifiers in AWS.

Now apply the changes. If you now try to access via the IP address (the IP address may have changed), you should be able to view the hextris website.

12 Tearing Down

One of the important features of Infrastructure as Code (IaC) is all the configuration we just did is stored in a file. This file can, and should be, version controlled and subject to the same quality rules of code files. It also means that if we want to redeploy hextris at any point, we can easily just run the IaC to deploy it.

To try this out, let us first take everything down. We can do this with:

```
$ terraform destroy
```

You should be prompted to confirm that you want to destroy all of the resources in the state. Once Terraform has finished taking everything down, confirm that you can no longer access the website and that the AWS console says the instances have been destroyed.

Now go ahead and apply the changes to bring everything back:

```
$ terraform apply
```

Confirm that this brings the website back exactly as before (with a different IP address). You can now start any lab you want and almost instantly spin back up the website you have configured. That is the beauty of Infrastructure as Code!

Hint: destroy everything again before you leave.

References

[1] D. Poccia, "Now open — third availability zone in the aws canada (central) region." https://aws.amazon.com/blogs/aws/now-open-third-availability-zone-in-the-aws-canada-central-region/, March 2020.

A AWS Networking Terminology

AWS Regions Regions are the physical locations of AWS data centres. When applying Terraform, the changes are being made to one region at a time. In our case we specified the region `us-east-1`. Often you do not need to deploy to more than one region, however, it can help decrease latency and reduce risk from a major disaster. Generally, pick a region and stick with it, we have picked `us-east-1` because it is the least expensive.

Figure 1: AWS Regions as of March 2020 [1]

Availability Zones An AWS Region will consist of availability zones, normally named with letters. For example, the AWS Region located in Sydney, `ap-southeast-2` has three availability zones: `ap-southeast-2a`, `ap-southeast-2b`, and `ap-southeast-2c`. An availability zone is a collection of resources which run on separate power supplies and networks. Essentially minimising the risk that multiple availability zones would fail at once.

VPC Virtual Private Clouds, or VPCs, are virtual networks under your control, if you have managed a regular network before it should be familiar. VPCs are contained within one region but are spread across multiple availability zones.