**IaC – DevOps with Terraform**

**Objective:** Get practical experience in DevOps (Development and Operation) and Infrastructure-as-Code through the automated cloud infrastructure creation and deployment on AWS using Terraform framework.

**Tasks:**

1. Describing the virtual infrastructure (VM, network adapter, security group, firewall rules)
2. Deployment of the virtual infrastructure on AWS EC2

**Lab environment**:

* Visual Studio Code with the following plugins:
  + Hashicorp Terraform
* Terraform
* AWS CLI

*Contents*

[**Step 1: Opening sample Terraform project**](#_heading=h.a8o9e8u2r436) **3**

[**Step 2: Understanding Terraform Code**](#_heading=h.2s8eyo1) **5**

[**Step 3: Updating key VM settings**](#_heading=h.dx16y91vkx8b) **8**

[**Step 4: Configuring AWS credentials for AWS CLI**](#_heading=h.17dp8vu) **11**

[**Step 5: Deploying AWS Virtual Infrastructure**](#_heading=h.h9cq32j4e31s) **13**

[**Step 6: Checking Deployed Resources**](#_heading=h.26in1rg) **14**

[**Step 7: Removing AWS EC2 Resources**](#_heading=h.vaz7f3ldzaop) **17**

[**Step 8: Advanced Self Task**](#_heading=h.tl6bjof94iqf) **17**

[**References**](#_heading=h.qnlscwkseuxw) **17**

***Terraform*** by HashiCorp is an open-source **infrastructure-as-code (IaC)** software framework that allows DevOps engineers to programmatically provision and manage the virtual computing resources and cloud infrastructure.

Infrastructure-as-code is an IT practice to manage computing infrastructure through programming. This approach to resource allocation allows developers to logically manage, monitor and provision virtual computing resources - as opposed to configuring each required resource manually.

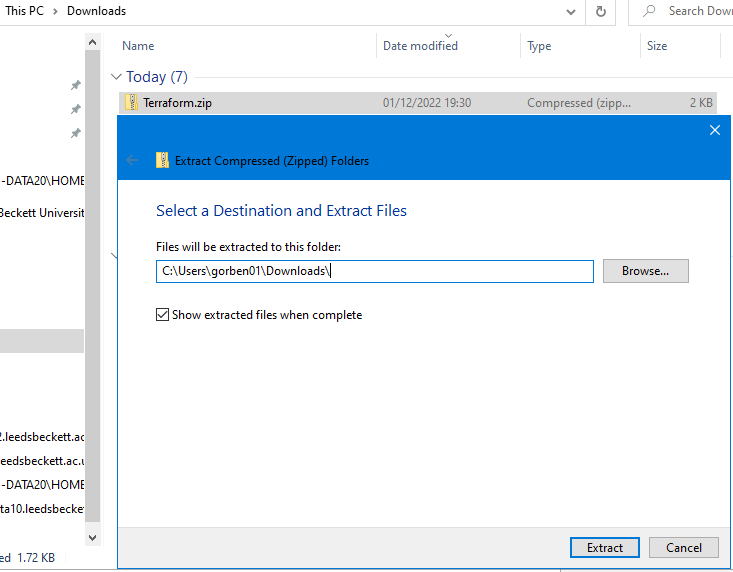
Terraform allows to describe computing infrastructure by using a JSON-like configuration language called HCL (HashiCorp Configuration Language) and automatically deploy it in the private or public virtualised environment, e.g. AWS, Azure, etc.

# 

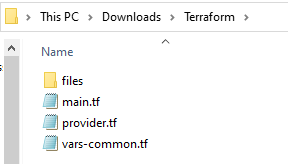
# Step 1: Opening sample Terraform project

In this lab we are using Visual Studio Code IDE with the Terraform plugin installed. Do not forget to delete AWS resources after use.

1. Download and unzip a sample Terraform project (Terraform.zip) from the VLE to the Download folder (e.g. C:\Users\c1234567\Downloads\, where c1234567 is your student ID).

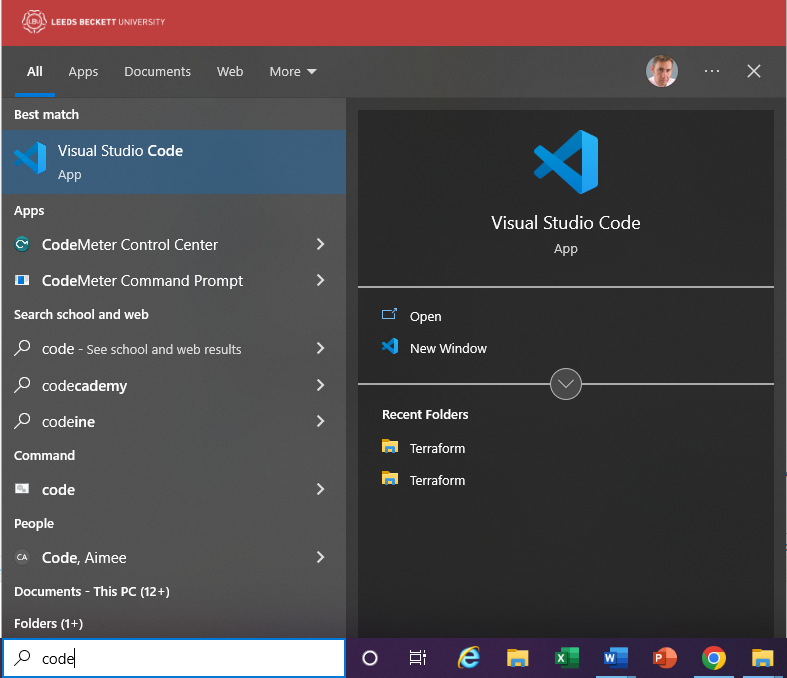


Make sure you have the following file structure after you unzip the project (avoid having ‘Terraform’ folder inside of another ‘Terraform’ folder):

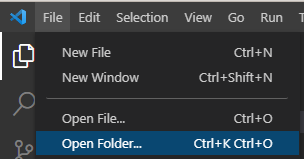


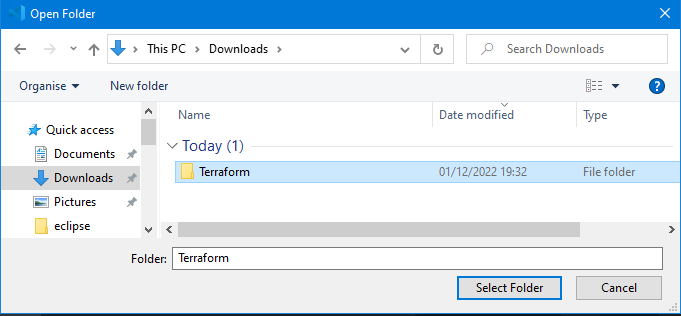
**Note:** Check, that your Download folder is on C: drive (e.g. C:\Users\c1234567\Downloads\), not on P: drive (!!!the project will not work from P:). If you do not have ‘Downloads’ on C:, then unzip the file to C:\Users\c1234567\**Documents**\.

1. Run Visual Studio Code IDE (not Visual Studio Community/Enterprise):



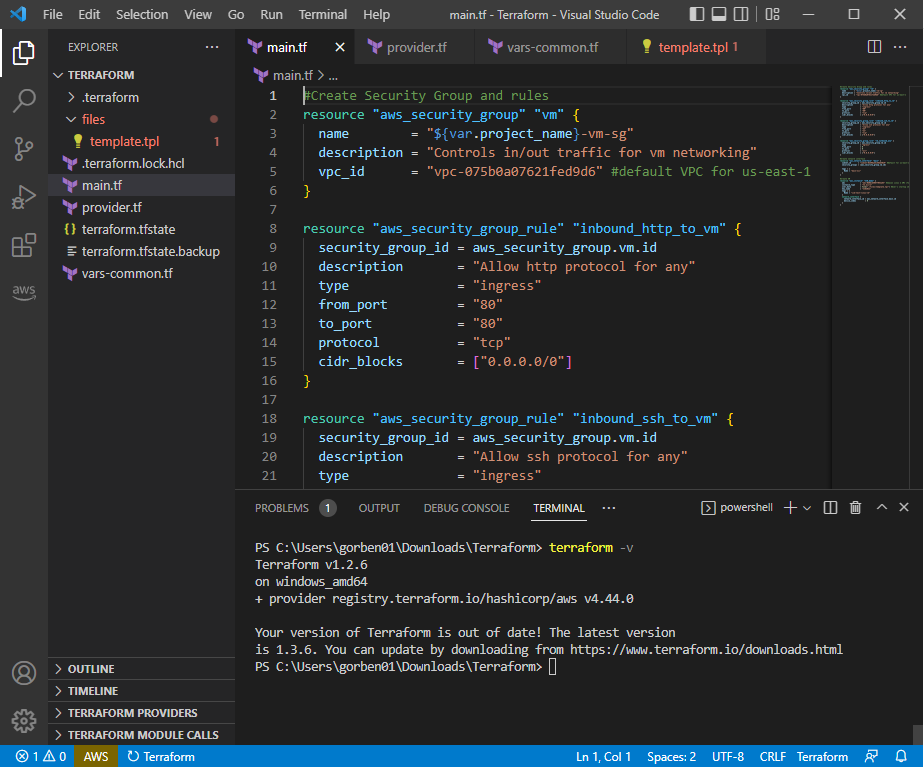
1. In the Visual Studio Code select File->Open Folder and browse to the unzipped ‘Terraform’ folder in Downloads:





# Step 2: Understanding Terraform Code

1. The sample project includes three Terraform configuration files, describing the virtual instance we are going to deploy (main.tf, provider.tf and vars-common.tf) and the startup shell script (files/template.tpl) installing and running Apache web server and creating a simple web page. Explore these files.



1. File ‘provider.tf’

This file simply defines a cloud provider we are going to deploy our VM with (AWS in our example):

provider "aws" {

profile = "default"

region = var.aws\_region

}

The AWS region is described in aws\_region variable which is defined in vars-common.tf file.

1. File ‘vars-common.tf’

This file defines common variables used in the project, e.g. name of the project and AWS region we would like to deploy our VM in:

variable "project\_name" {

default = "CCD\_DevOps\_Example\_c1234567"

}

#Set your aws\_region

variable "aws\_region" {

default = "us-east-1"

}

Replace c1234567 with your student id.

1. File ‘main.tf’

This is the main Terraform config file describing the computing infrastructure we are going to create and deploy.

**#Create Security Group and rules for inbound and outbound network traffic**

resource "aws\_security\_group" "vm" {

name = "${var.project\_name}-vm-sg"

description = "Controls in/out traffic for vm networking"

vpc\_id = "vpc-075b0a07621fed9d6" **#id of the Virtual Private Network - you will need to update it**

}

resource "aws\_security\_group\_rule" "inbound\_http\_to\_vm" {

security\_group\_id = aws\_security\_group.vm.id

description = "Allow http protocol for any"

type = "ingress"

from\_port = "80"

to\_port = "80"

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

resource "aws\_security\_group\_rule" "inbound\_ssh\_to\_vm" {

security\_group\_id = aws\_security\_group.vm.id

description = "Allow ssh protocol for an

type = "ingress"

from\_port = "22"

to\_port = "22"

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

resource "aws\_security\_group\_rule" "vm\_outbound\_any" {

security\_group\_id = aws\_security\_group.vm.id

type = "egress"

from\_port = 0

to\_port = 0

protocol = "all"

cidr\_blocks = ["0.0.0.0/0"]

}

**#Create network interface**

resource "aws\_network\_interface" "main" {

subnet\_id = "subnet-0e498a34c58db50e9" **#subnet id for the specific availability zone, e.g. us-east-1b - you will need to update it**

security\_groups = [aws\_security\_group.vm.id]

tags = {

Name = "main-nic"

}

}

**#Create VM**

resource "aws\_instance" "CCD\_demo" {

ami = "ami-0b0dcb5067f052a63" **#Amazon Linux 2 AMI (for us-east-1 N.Virginia) - you might need to update it if you deploy your VM in a region other than us-east-1**

instance\_type = "t2.micro"

user\_data = file("./files/template.tpl") **#User's startup shell script - get content from files/template.tpl file**

key\_name = "your\_keypair\_name" **#Use the name of YOUR key pair**

tags = {

Name = "ccd-test-linux-vm"

}

network\_interface {

network\_interface\_id = aws\_network\_interface.main.id

device\_index = 0

}

}

Our computing infrastructure we are going to create and deploy is a single virtual web server using Amazon Linux OS to be deployed in us-east-1 (us-east-1b availability zone).

Step-by-step we perform the following:

1. Create a new security group in us-east-1 Virtual Private Cloud (VPC);
2. Add firewall rules to the security group for inbound and outbound traffic enabling: (i) inbound HTTP traffic (local port 80) to provide access to a web server on the VM, (ii) inbound SSH traffic (local port 22) to allow external management via SSH, and (iii) all outbound network connections.
3. Create a new network interface which applies the above security group;
4. Create a new Linux VM by: (i) defining VM image (ami), which correspond to Amazon Linux 2 OS; (ii) specifying instance type/size, i.e. hardware resources as t2.micro; (iii) providing path to user’s startup shell script which will download and install Apache web server and create a simple web page; (iv) attaching the networking interface created above.

# Step 3: Updating key VM settings

In the above code you MUST replace the following settings with yours, which are different for different users and/or AWS regions:

key\_name = "your\_keypair\_name"

vpc\_id = "vpc-075b0a07621fed9d6"

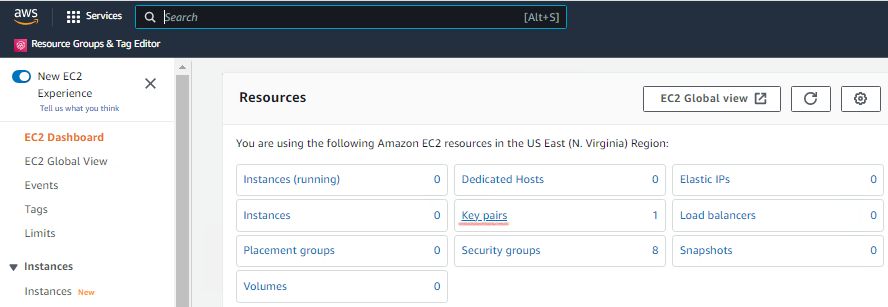
subnet\_id = "subnet-0e498a34c58db50e9"

ami = "ami-0b0dcb5067f052a63"

**key\_name:**

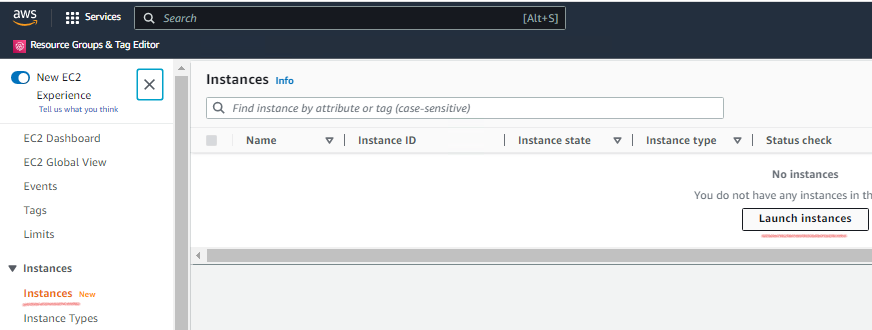
You need to specify the name of YOUR key-pair (instead of "your\_keypair\_name") which you previously created and will need to use to connect to the VM via SSH.

Check your security key-pair name (or create a new one if you lost the .pem file) in the EC2 Dashboard:

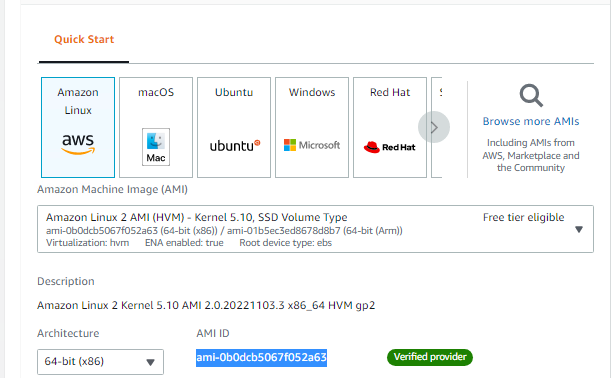


**ami, vpc\_id, subnet\_id:**

You can notice desired ami, vpc\_id and subnet\_id in AWS EC2 console by going through ‘Launch Instances Wizard’.

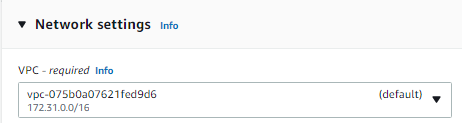


AMI (OS image):



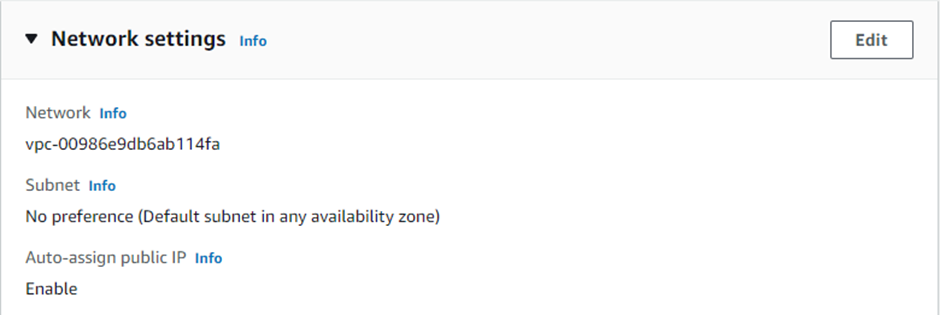
Vpc\_id:

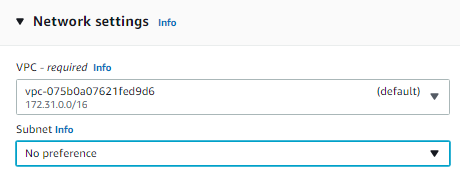
Go to network settings and notice id for VPC (Virtual Private Cloud):

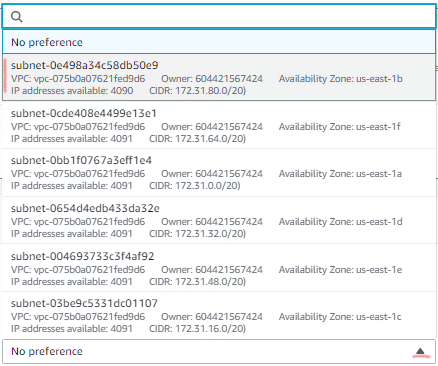


Subnet\_id:

Click on ‘Edit’ -> ’Subnet’ of ‘Network settings’ and notice the subnet\_id:





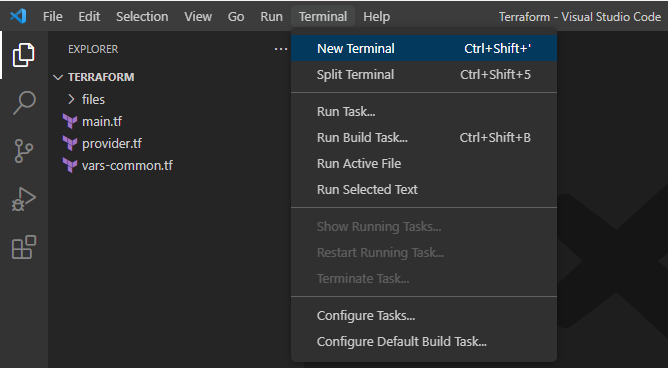


If later you would like to deploy a Windows web server, you will need to change ami and use another startup script, e.g. the power shell script we used in one of previous labs and replace it in the user\_data settings.

user\_data = file("./files/template.tpl")

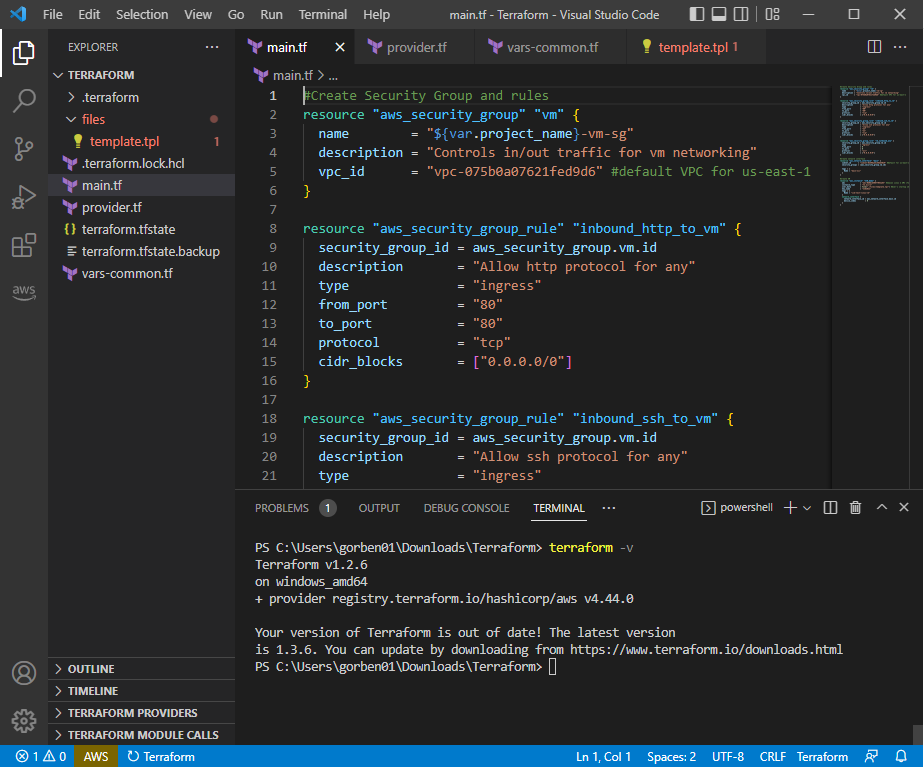
# Step 4: Configuring AWS credentials for AWS CLI

Open a New Terminal Window in Visual Studio Code and run the following commands.



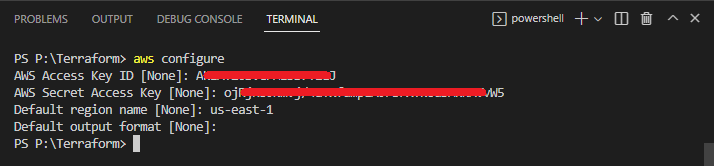
1. First, check if the Terraform framework is installed (you can copy/paste the command; use right click to paste):

.



1. Second, configure YOUR aws credentials (copy/paste your AWS access and secret keys when prompted from the .csv file you downloaded earlier):

aws configure



# Step 5: Deploying AWS Virtual Infrastructure

Run the following commands:

terraform init

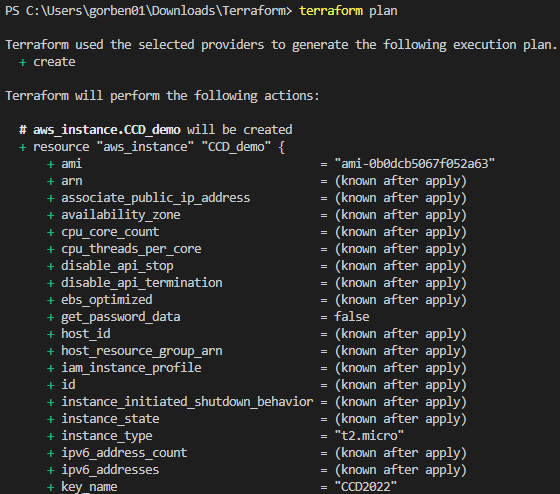


terraform validate



terraform plan

You will see a list of actions to be performed to deploy your infrastructure:

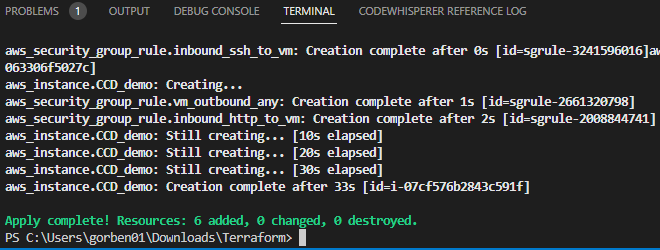


terraform apply



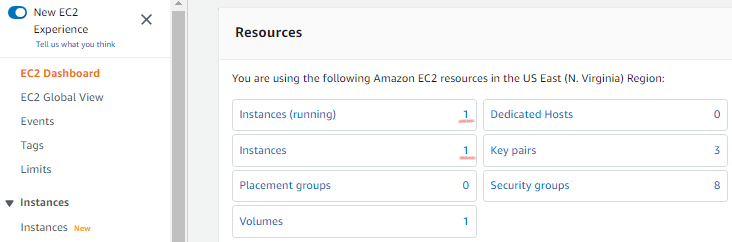
Answer ‘yes’ when you are prompted to perform these actions.

Wait until you receive a message that the instance was successfully created:



# Step 6: Checking Deployed Resources

1. Log in to AWS <https://aws.amazon.com/> and go to AWS console -> EC2 Dashboard and check your instances:

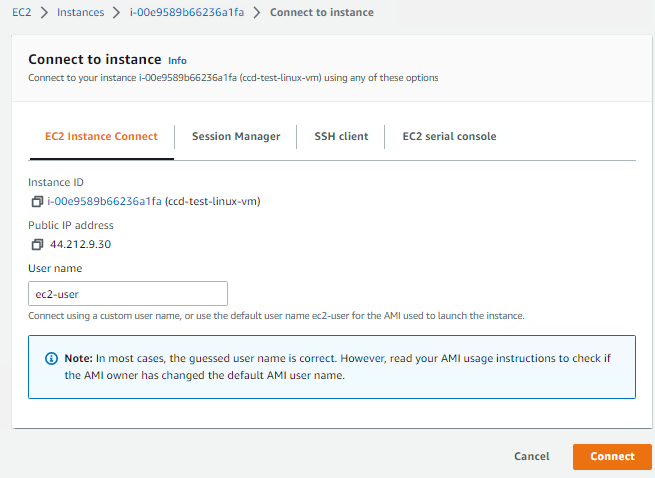


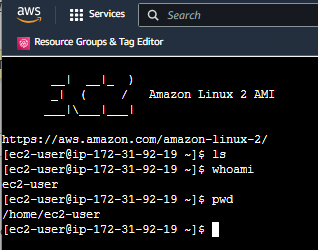
1. Check, that the web server has successfully started and the simple web page we created is available for browsing (make sure you use http instead of https):



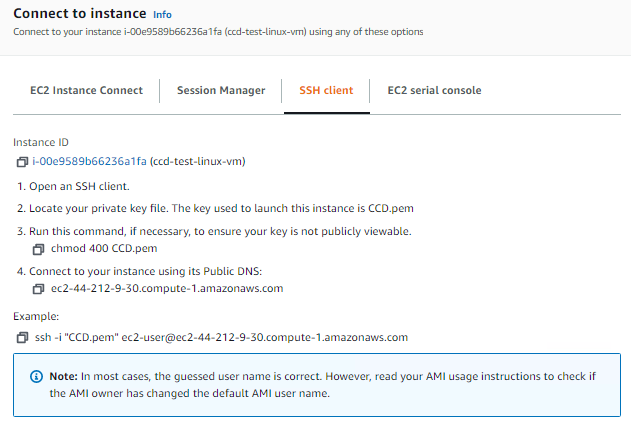


1. Connect to your instance and run some commands in the terminal window:





1. You can also connect to the instance using an SSH client similar to how you did in one of the first CCD labs (do not forget to provide a path to the .pem keypair file which you need to have on your desktop PC).



# Step 7: Removing AWS EC2 Resources

You can terminate the instance from the AWS Console or directly from Visual Code by running (type ‘yes’ when prompted):

terraform destroy



Type ‘yes’ to confirm your choice.

Wait until you receive a message that your AWS resources have been successfully deleted.



Go to AWS EC2 Dashboard and check your resources.

# Step 8: Advanced Self Task

Update the Terraform specification to create a Windows Web Server similar to one we created manually through the AWS GUI (see Week 2). Make sure you replace the startup bash script with the PowerShell script which instals the Internet Information Server on Windows and deploy a simple web page.

# References

Watch this video if you want to install and use Terraform on your home PC/laptop: <https://www.youtube.com/watch?v=hmKC6YagHqY>