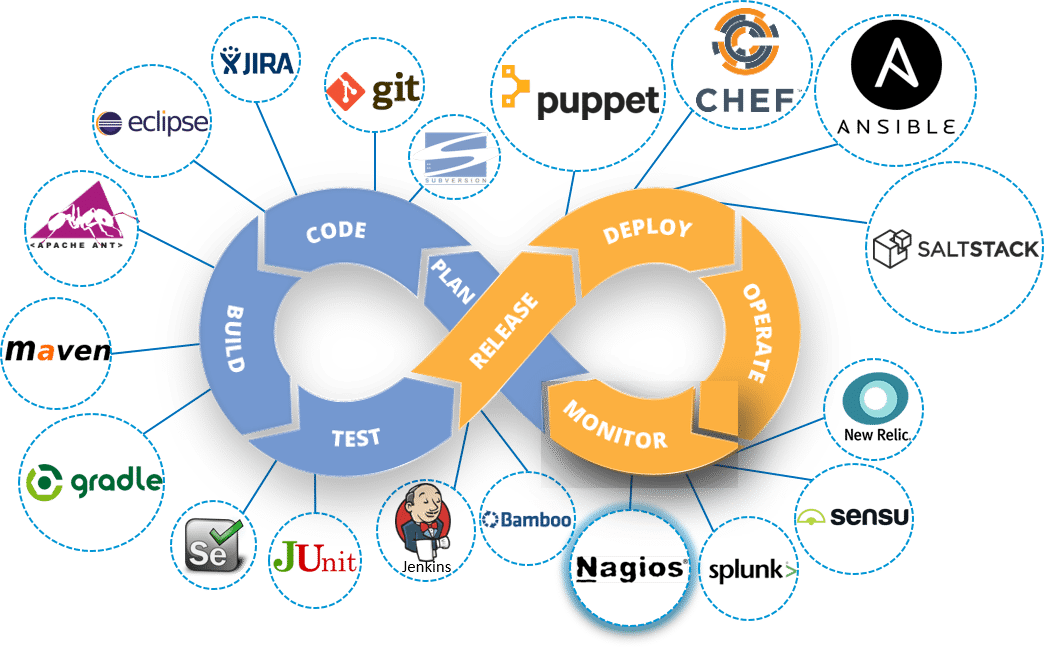
**Assignment-1                                              Marks: 25**

Explain the role of continuous monitoring in DevOps and demonstrate the use of Nagios as a continuous monitoring tool by configuring a host machine with Nagios server.

**Solution :**

## ****Continuous Monitoring:****

Let me first tell you where Continuous Monitoring lies in the DevOps life-cycle, consider the diagram below:



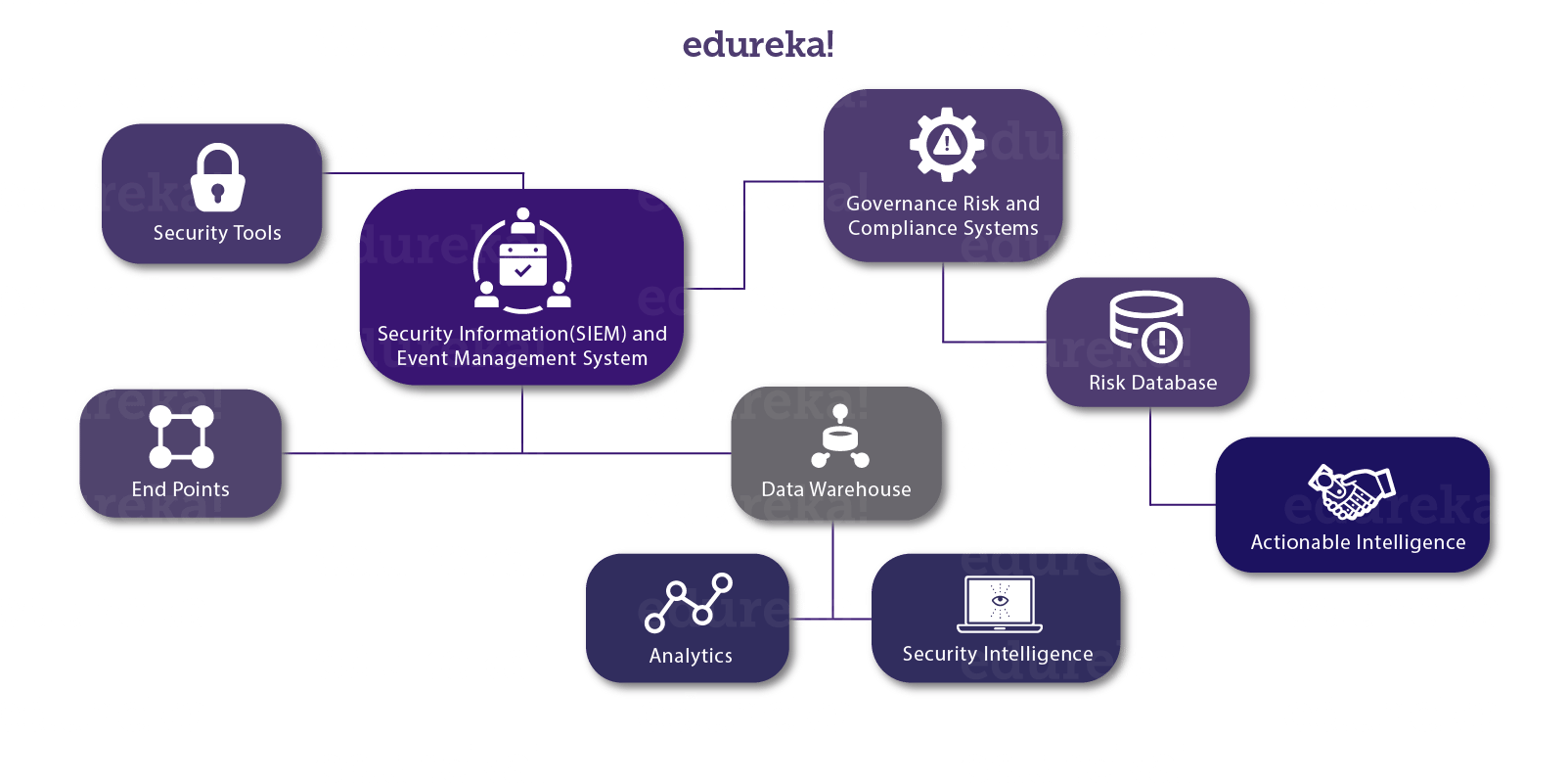
Looking at the diagram you must be thinking this is the last stage in the DevOps lifecycle, but this is not the case. There is no end to a DevOps life cycle, and that is the reason for the infinity symbol. Continuous Monitoring comes into the picture, once the application is deployed on the production servers.

Continuous Monitoring is all about the ability of an organization to detect, report, respond, contain and mitigate the attacks that occur, in its infrastructure.

Continuous Monitoring is actually not new, it’s been around for some time. For years our security professionals are performing static analysis from – system log, firewall logs, IDS logs, IPS logs etc. But, it did not provide proper analysis and response. Today’s Continuous Monitoring approache gives us the ability to aggregate all of the events that I discussed above, co-relate them, compare them and then estimate the organization’s risk posture.

If we take all these pieces and ensure the interlinking between them. This is the crux of Continuous Monitoring.

Let me explain this with a use-case. Consider the diagram below:

Now, let me explain you the above diagram:

1. We have various security tools, like Firewall, IDS, End Point Protection etc. they are connected with a ‘Security Information and Event Management system.
2. In order to achieve Continuous Monitoring, we need to have all the parts talking to each other, let me explain that to you.
3. So we have security tools and series of ‘End Points’,  this can include client and servers, routers, switches, mobile devices and so on.
4. These two groups can then talk to a Security Information and Event Management system (SIEM), through a common language and in more automated fashion.
5. Connected to this SIEM there are two important components, first one is a Data Warehouse. Now to this Data Warehouse, we will connect ‘Analytics’ and ‘Security Intelligence’.
6. Security intelligence (SI) is the information relevant to protecting an organization from external and insider threats as well as the processes, policies and tools designed to gather and analyze that information.
7. This SIEM is also connected to a ‘Governance Risk and Compliance System’ it basically provides dashboarding.
8. To this ‘Governance Risk and Compliance System’  we attach a risk database. This gives us ‘Actionable Intelligence’.
9. Actionable Intelligence is nothing but information that can be acted upon, with the further implication that actions *should*be taken.

So here, we are Monitoring the events on the on going basis and determining what level of risk we are experiencing. With this, we can correlate the events at the SIEM. We can perform Network Behavior and Anomaly Detection at the ‘Analytics Engine’. This is what Continuous Monitoring is all about:

*The integration of an organization security tools, the aggregation, normalization and correlation of the data that is produced by the security tools. The analysis of that data, based on the organization’s risk goals and threat knowledge, and near real-time response to the risks identified.*

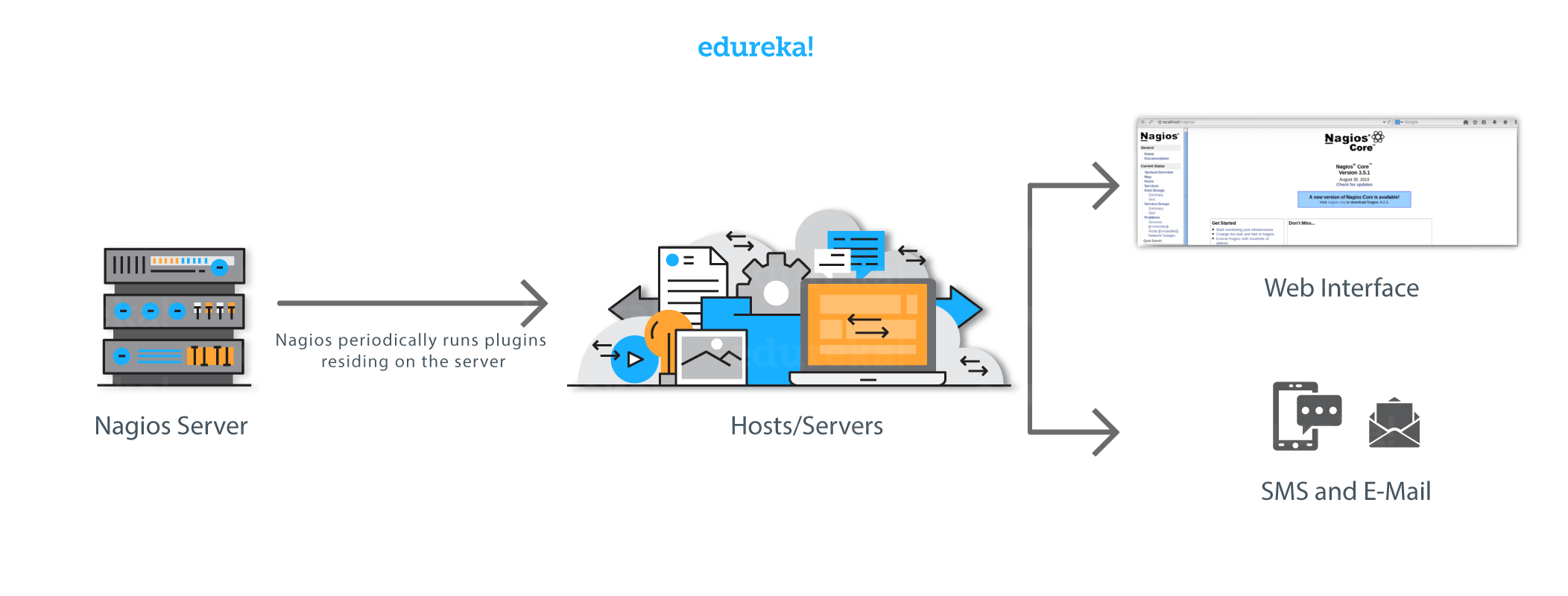
‘If you can’t measure it you can’t manage it’. I hope you know what I am talking about.

Next up in this Nagios tutorial, I will be introducing you to one of the most famous monitoring tool ‘Nagios’.

**Nagios:**

Nagios is used for Continuous monitoring of systems, applications, services, and business processes etc in a DevOps culture. In the event of a failure, Nagios can alert technical staff of the problem, allowing them to begin remediation processes before outages affect business processes, end-users, or customers. With Nagios, you don’t have to explain why an unseen infrastructure outage affect your organization’s bottom line.

Let me explain to you how Nagios works. Consider the diagram below:

Nagios runs on a server, usually as a daemon or a service.

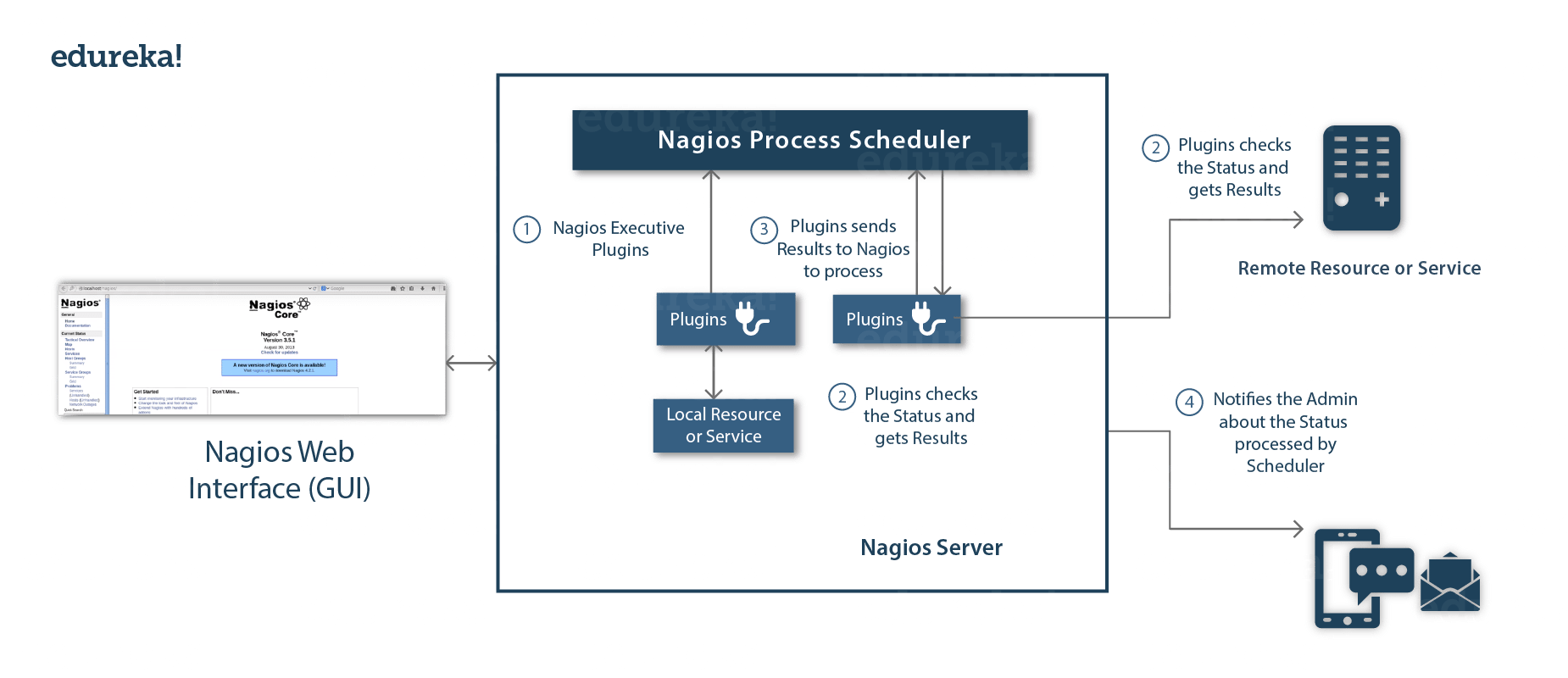
It periodically runs plugins residing on the same server, they contact hosts or servers on your network or on the internet. One can view the status information using the web interface. You can also receive email or SMS notifications if something happens.  
The Nagios daemon behaves like a scheduler that runs certain scripts at certain moments. It stores the results of those scripts and will run other scripts if these results change.

*Plugins:*These are compiled executables or scripts (Perl scripts, shell scripts, etc.) that can be run from a command line to check the status or a host or service. Nagios uses the results from the plugins to determine the current status of the hosts and services on your network.

Let’s now discuss it’s architecture.

**Nagios Architecture:**

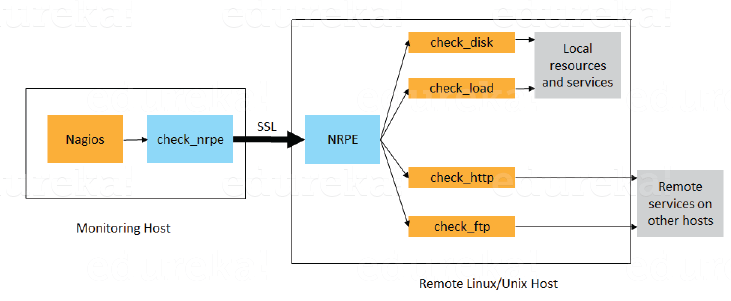
* Nagios is built on a server/agents architecture.
* Usually, on a network, a Nagios server is running on a host, and Plugins interact with local and all the remote hosts that need to be monitored.
* These plugins will send information to the Scheduler, which displays that in a GUI.



I also need to explain you NRPE (Nagios Remote Plugin Executor).

The NRPE addon is designed to allow you to execute Nagios plugins on remote Linux/Unix machines. The main reason for doing this is to allow Nagios to monitor “local” resources (like CPU load, memory usage, etc.) on remote machines. Since these public resources are not usually exposed to external machines, an agent like NRPE must be installed on the remote Linux/Unix machines.

Consider the diagram below:



* The check\_nrpe plugin, resides on the local monitoring machine.
* The NRPE daemon, runs on the remote Linux/Unix machine.
* There is a SSL (Secure Socket Layer) connection between monitoring host and remote host as shown in the diagram above.

Now in this Nagios tutorial, it is time for some **Hands-on**.

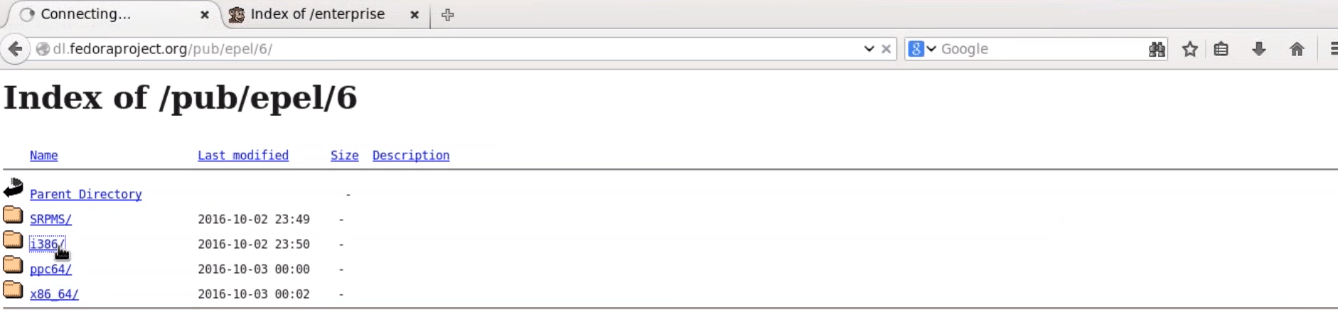
Let’s start by installing Nagios Core

**Install Nagios Core:**

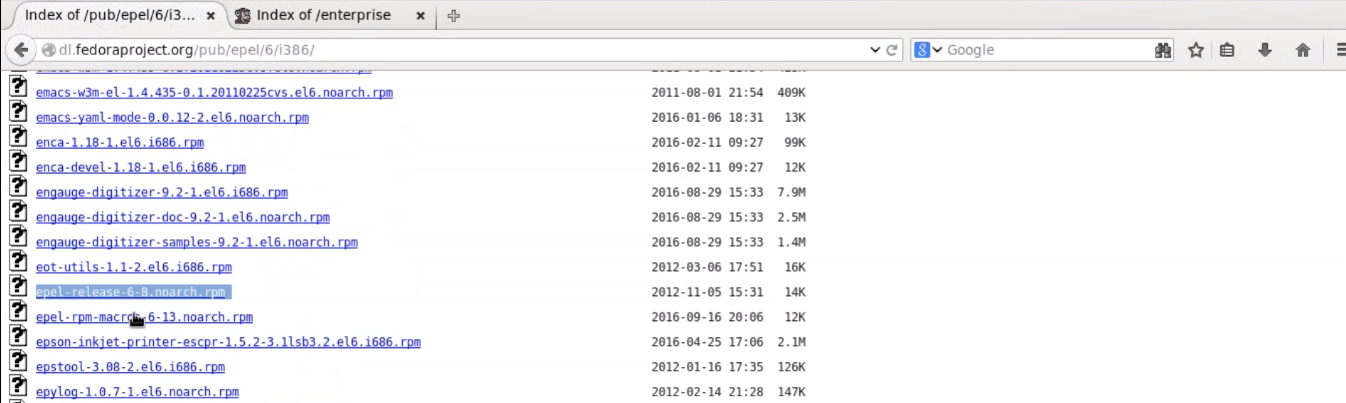
The complete process to install Nagios can be summarized in four steps:

1. Install Required Packages In The Monitoring Server
2. Install Nagios Core, Nagios Plugins And NRPE (Nagios Remote Plugin Executor)
3. Set Nagios Password To Access The Web Interface
4. Install NRPE In Client

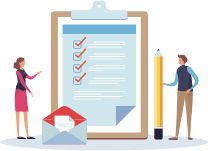
**Step – 1: Install Required Packages On The Monitoring Server:**



Click on i386, and then you will be redirected to a page.



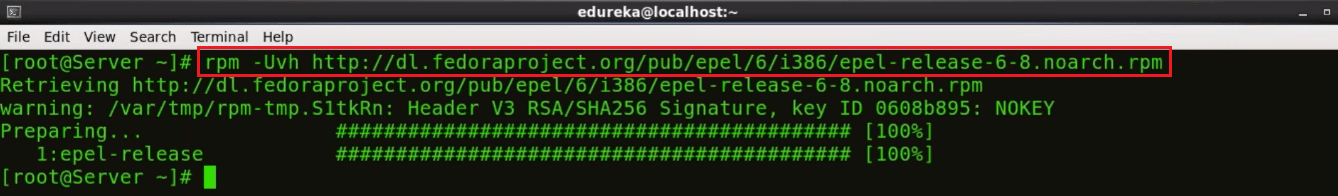
Since I am using CentOS 6, so I will right click and copy the link location of ‘*epel-release-6-8.noarch.rpm*‘, as shown in the above screenshot.

[[](https://www.edureka.co/devops-certification-training)](https://www.edureka.co/devops-certification-training" \t "_blank)

**[DevOps Certification Training Course](https://www.edureka.co/devops-certification-training" \t "_blank)**

[Explore Curriculum](https://www.edureka.co/devops-certification-training" \t "_blank)

Open the terminal and use *rpm -Uvh*command and paste the link.

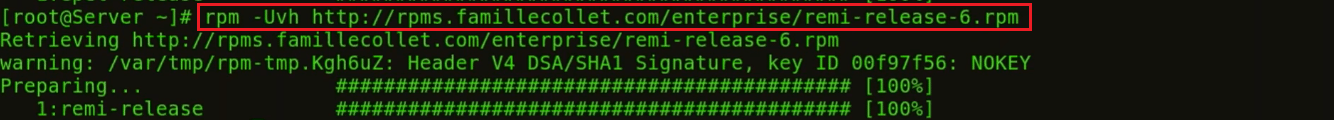


We need to download one more repository, for that visit the website ‘<http://rpms.famillecollet.com/enterprise/>‘



Right-click and copy the link location for ‘*remi-release-6.rpm*‘

Again open the terminal and use *rpm -Uvh* command and paste the link.



Fine, so we are done with the pre-requisites. Let’s proceed to the next step.

**Step – 2: Install Nagios Core, Nagios Plugins And NRPE (Nagios Remote Plugin Executor):**

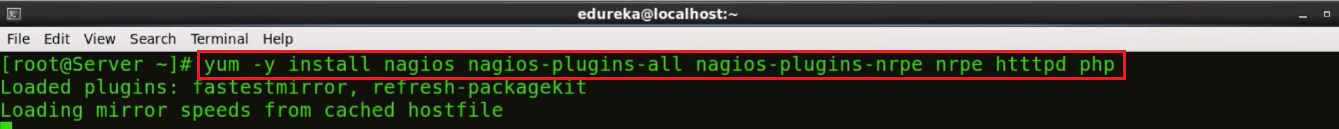
Execute the below command in the terminal:

yum -y install nagios nagios-plugins-all nagios-plugins-nrpe nrpe httpd php

This will install Nagios, Nagios Plugins, Plugins for NRPE, NRPE, Apache and PHP

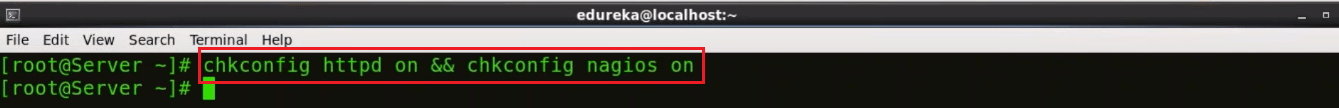
Apache web server is required to monitor the current web server status.

Php is used to process dynamic content of the site date.



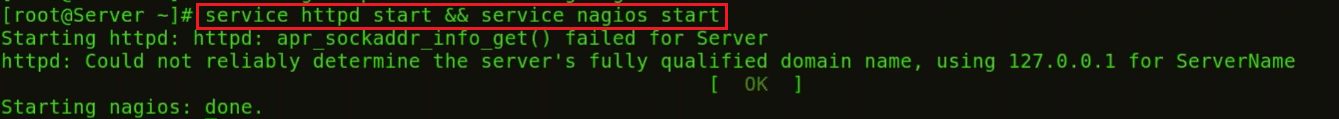
Next, we need to enable Apache and Nagios service:

chkconfig httpd on && chkconfig nagios on



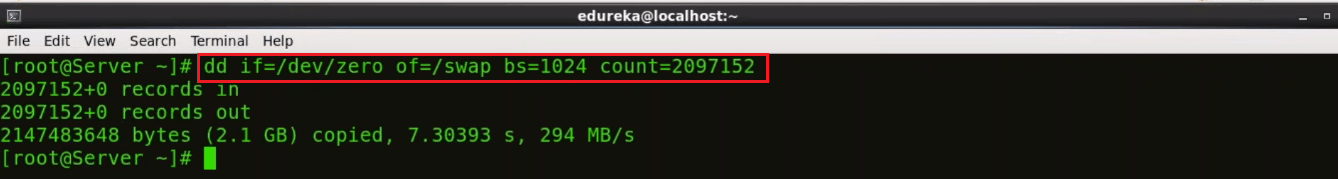
Our next step is to start Nagios and Apache:

service httpd start && service nagios start



Now, I will enable swap memory of at least 1GB. It’s time to create the swap file itself using the dd command:

dd if=/dev/zero of=/swap bs=1024 count=2097152

Swap is basically used to free some, not so frequently accessed information from RAM, and move it to a specific partition on our hard drive.

Now that you have created the swap partition, use the command *mkswap* to setup the swap partition. This is going to prepare the swap file by creating a linux swap area.

mkswap /swap

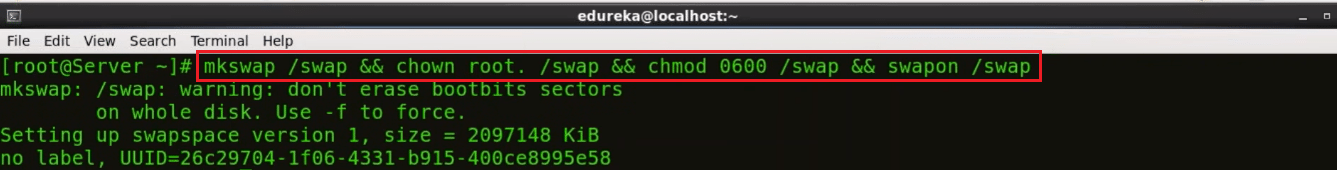
To prevent the file from being world-readable, you should set up the correct permissions on the swap file:

chown root. /swap

chmod 0600 /swap

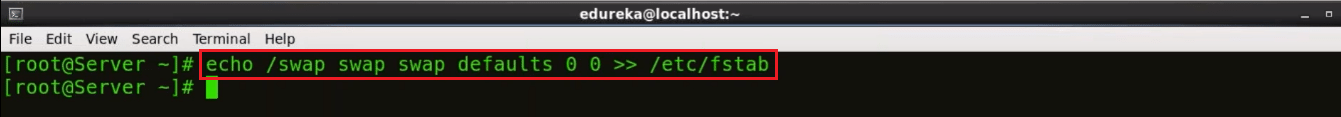
If you see no errors, your swap space is ready to use. To activate it immediately, type:

swapon /swap



This file will last on the virtual private server until the machine reboots. You can ensure that the swap is permanent by adding it to the fstab file.

echo /swap swap swap defaults 0 0 >> /etc/fstab



The operating system kernel can adjust how often it relies on swap through a configuration parameter known as **swappiness**.

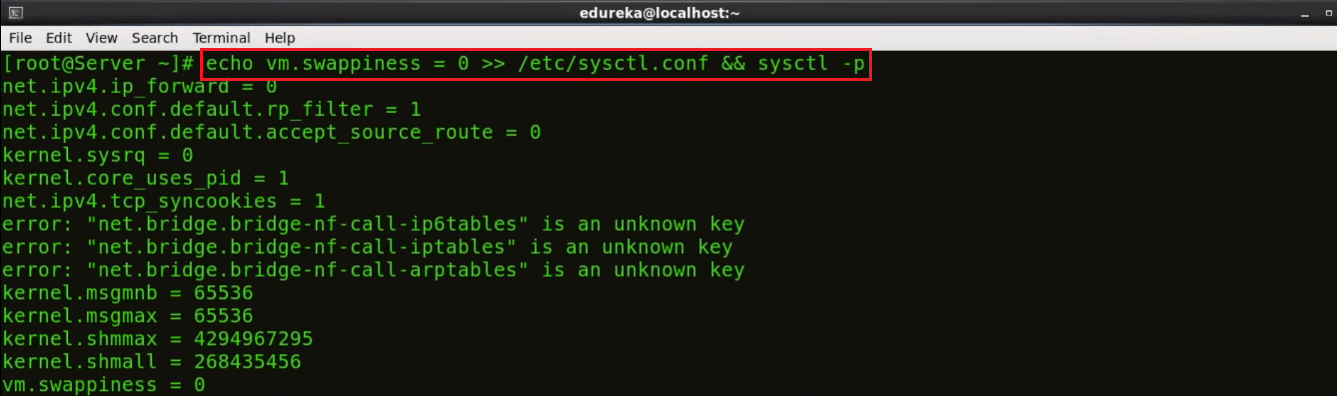
To find the current swappiness settings, type:

cat /proc/sys/vm/swappiness

Swapiness can be a value from 0 to 100. Swappiness near 100 means that the operating system will swap often and usually, too soon. Although swap provides extra resources, RAM is much faster than swap space. Anytime something is moved from RAM to swap, it slows down.

A swappiness value of 0 means that the operating will only rely on swap when it absolutely needs to. We can adjust the swappiness with the sysctl command. To make your VPS automatically apply this setting every time it boots up, you can add the setting to the /etc/sysctl.conf file:

echo vm.swappiness = 0 >> /etc/sysctl.conf && sysctl -p



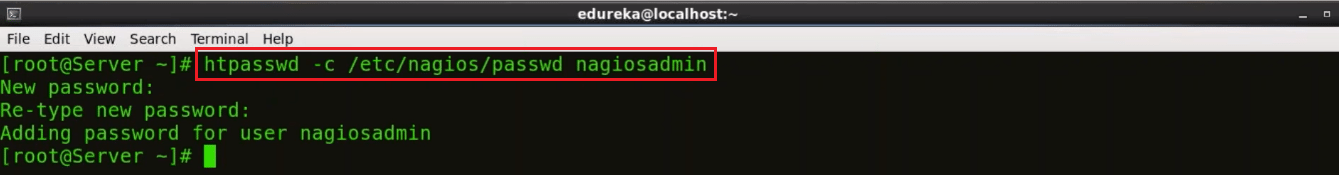
Finally, we are done with the second step.

Let’s proceed further and set Nagios password to access the web interface.

**Step – 3: Set Nagios Password To Access The Web Interface:**

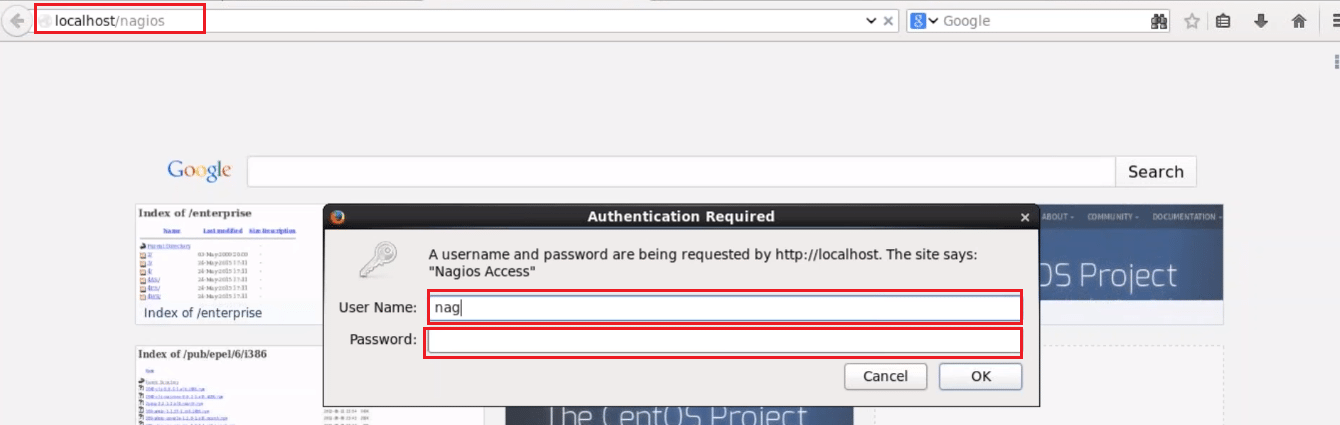
Set the password to access the web interface, use the below command:

htpasswd -c /etc/nagios/passwd nagiosadmin



Type the password and confirm it by retyping it.

Now, open the browser. Here, type your public IP or hostname/nagios. Consider the example below:



Here, give the user name and password. By default, the user name is *nagiosadmin,*and password is what you have set in the previous step. Finally, press OK.

After this, you will directed to Nagios Core dashboard.

**Assignment-2(choice question)                        Marks: 25**

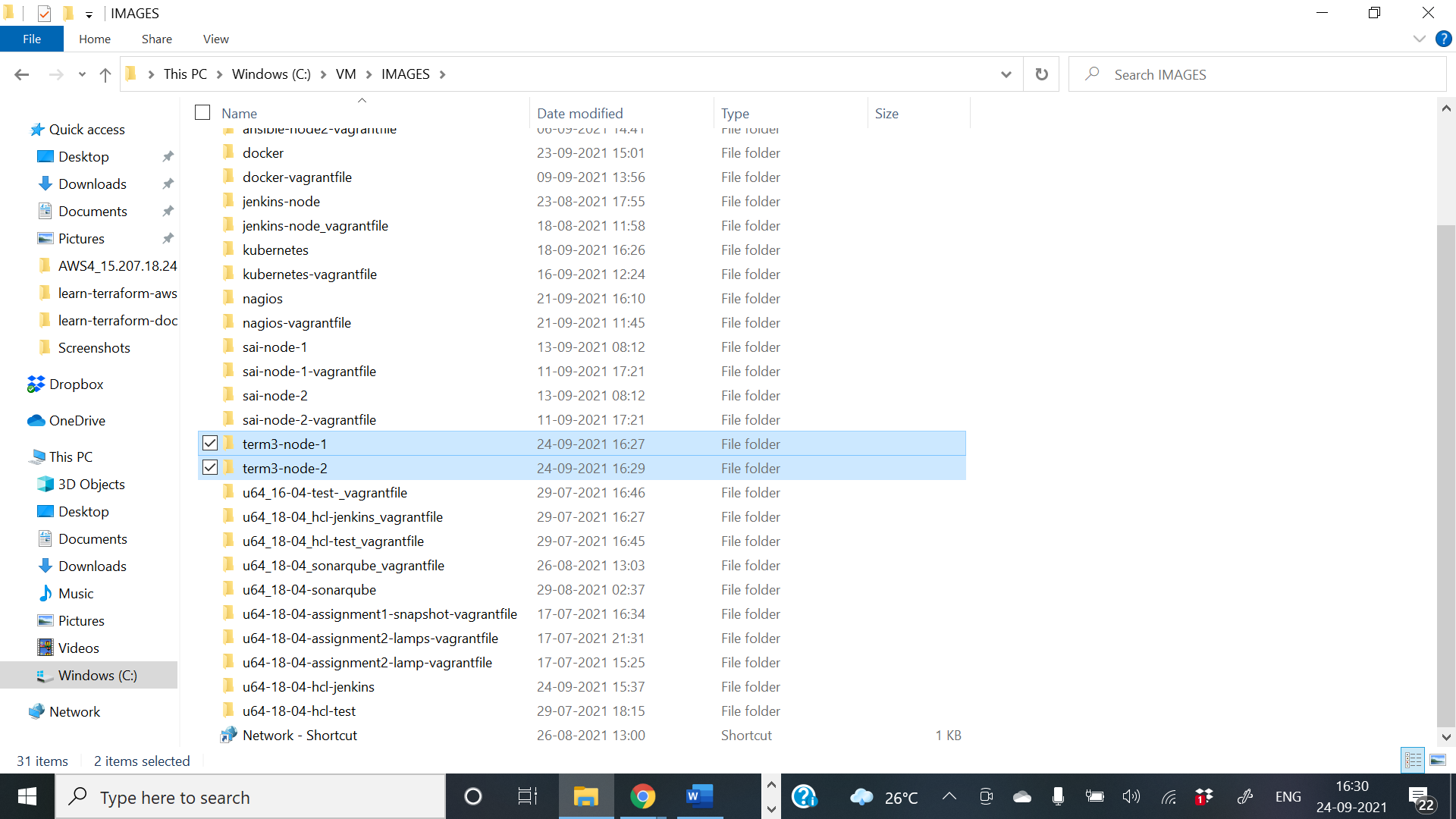
Create 2 new ansible nodes as per below specifications

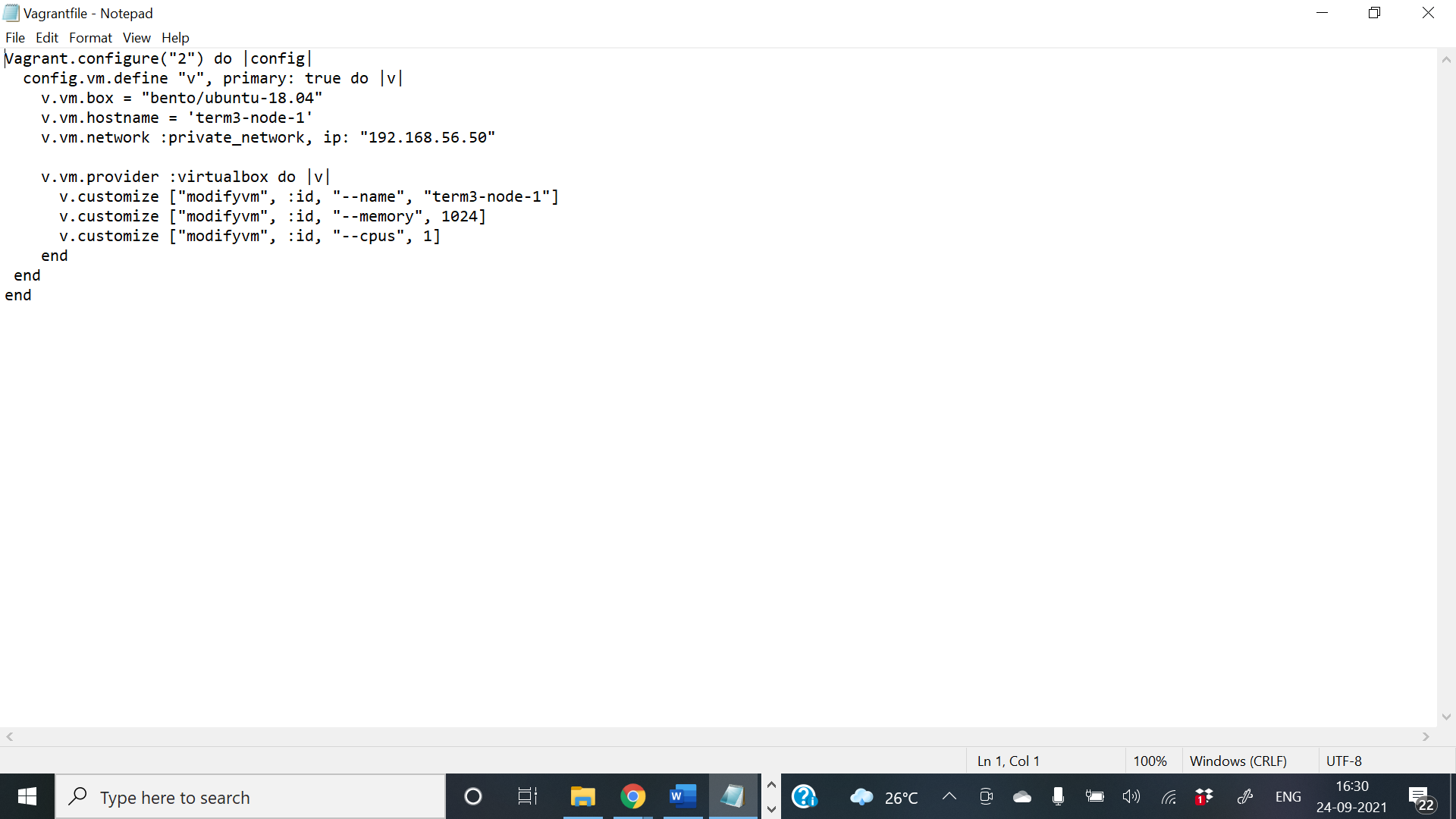
          IP                               Host name                               OS

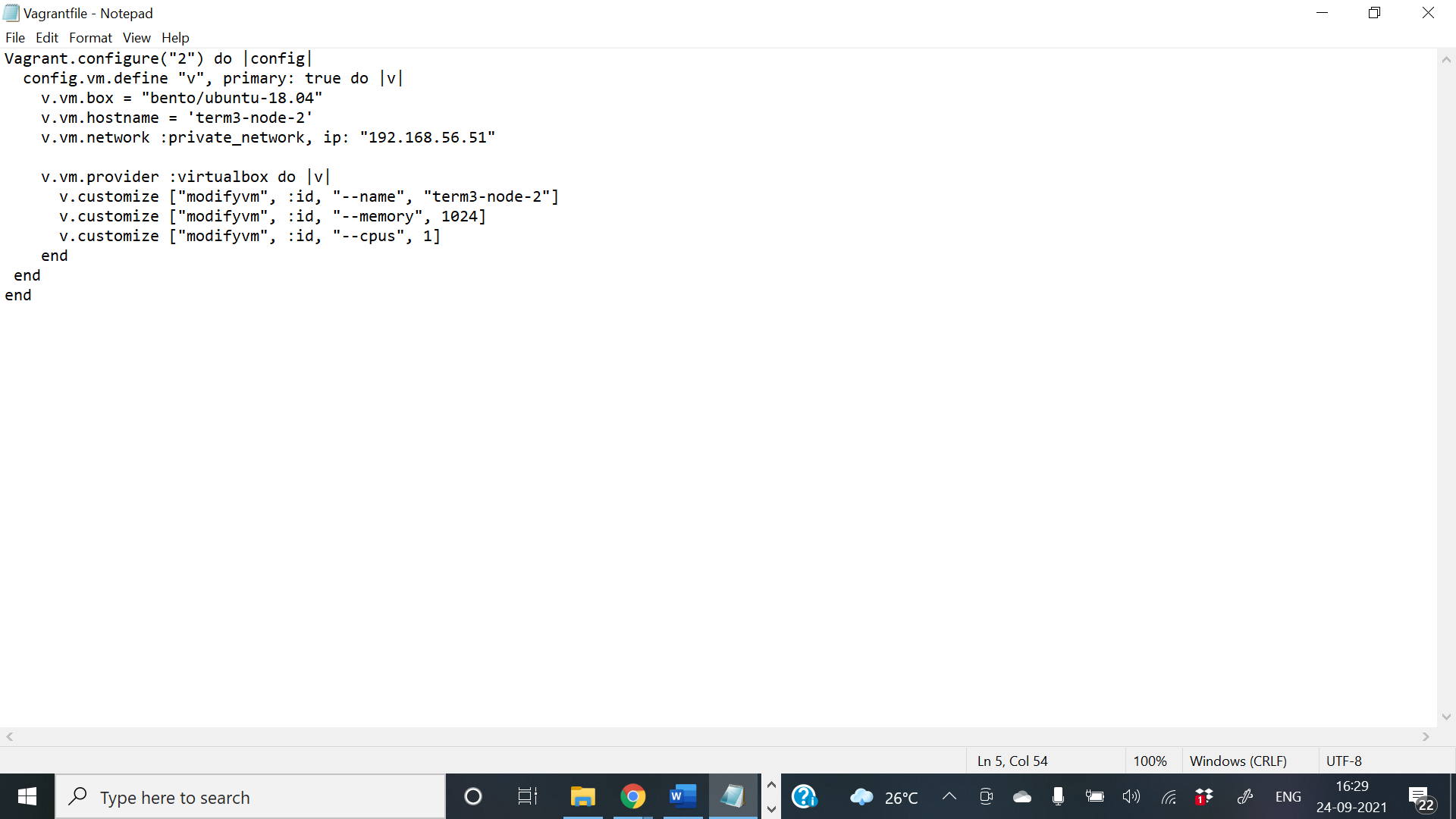
192.168.56.50                  term3-node-1                ubuntu 18.04

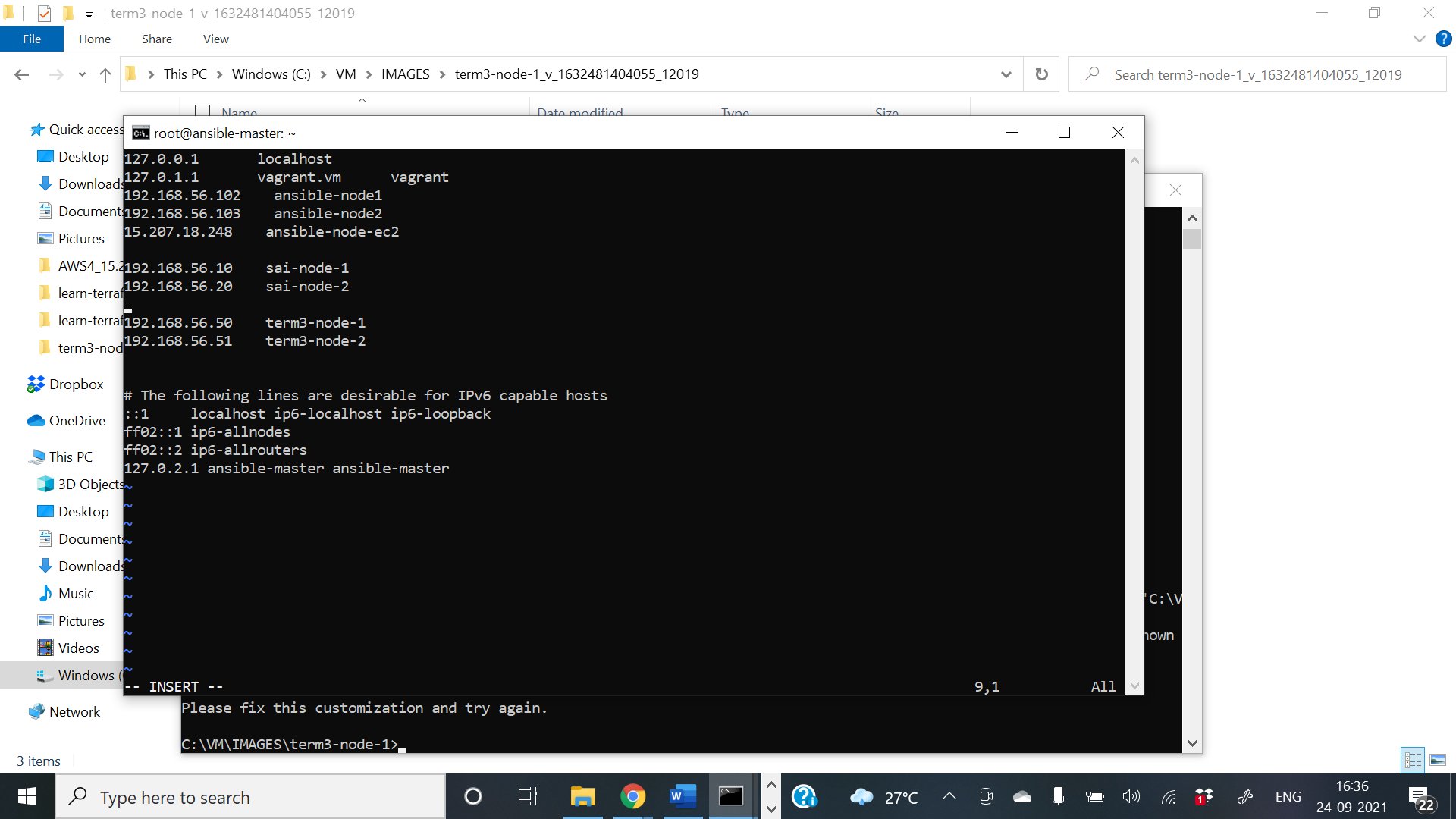
192.168.56.51                  term3-node-2                 ubuntu 18.04

Create an ansible role using ansible-galaxy and deploy the same to both the ansible nodes.

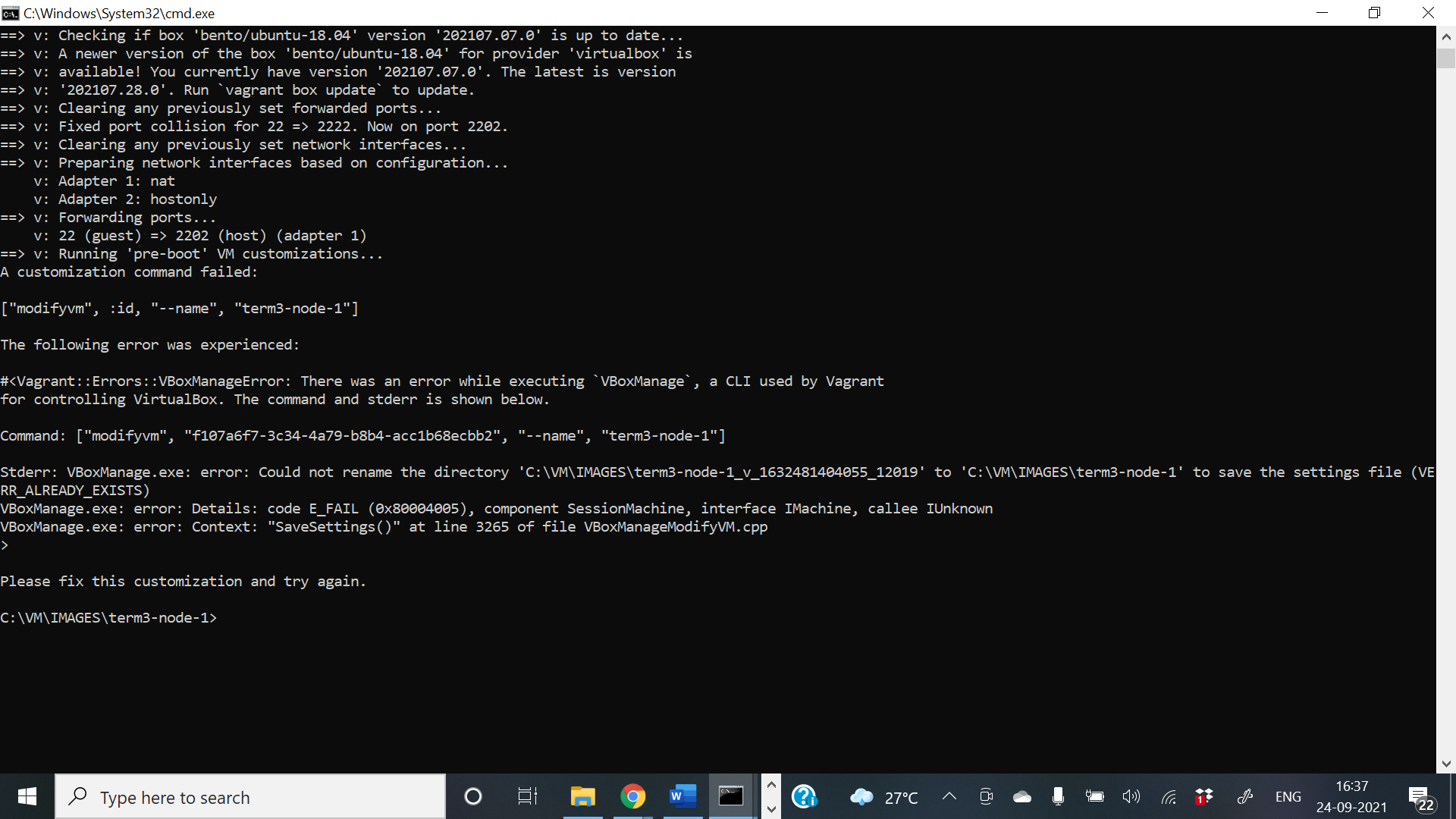


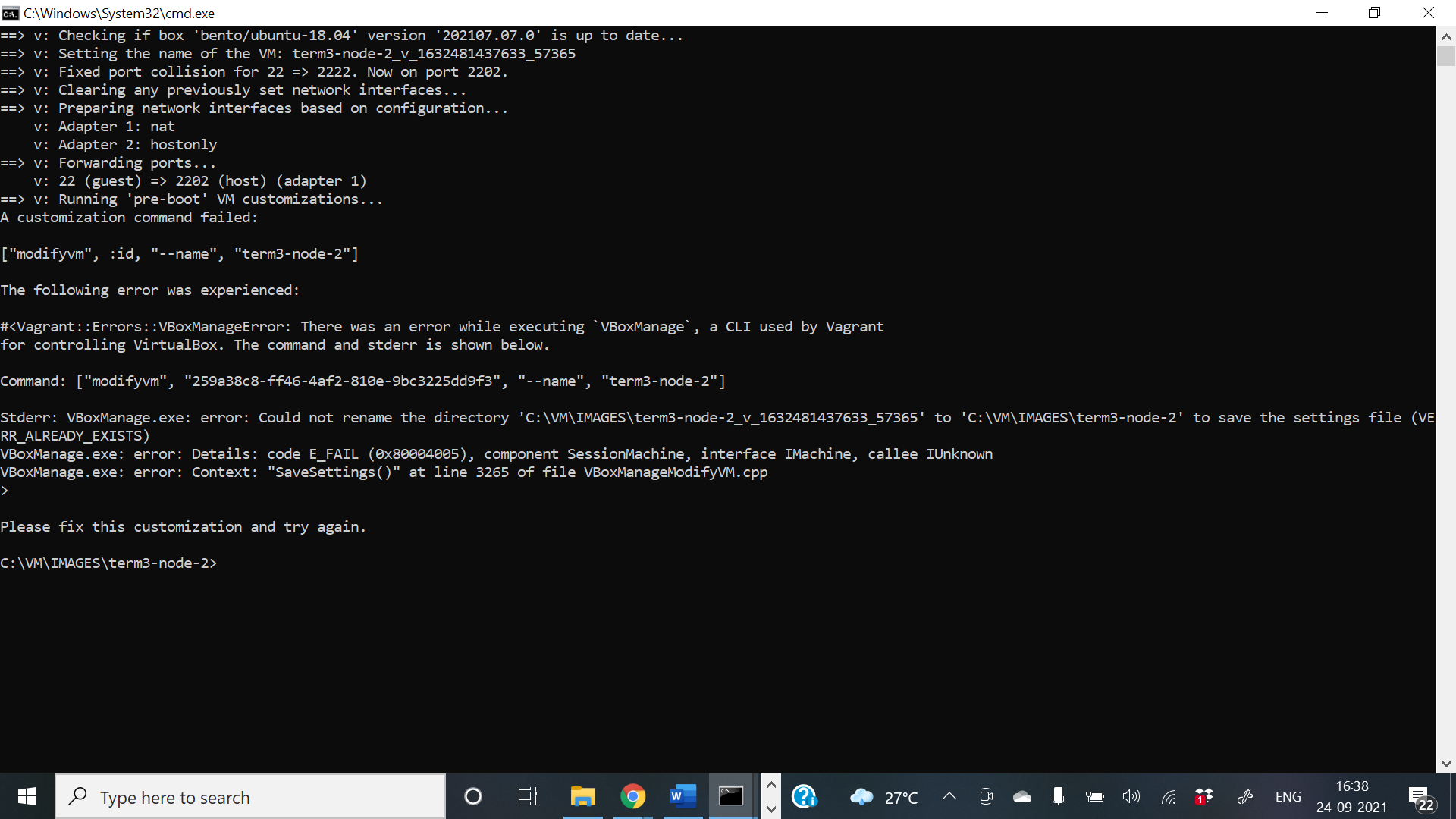






Error in term3-node-1 and term3-node-2





**Assignment-3                                                                Marks: 25**

Explain the role of Terraform as Infrastructure as Code and provisioning tool and demonstrate the process to perform the basic operations of Terraform to provision EC2 instance in AWS.

**Solution :**

**Introduction to Terraform:**

Terraform is an infrastructure as code (IaC) tool that allows you to build, change, and version infrastructure safely and efficiently. This includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc. Terraform can manage both existing service providers and custom in-house solutions.

**Key Features :**

**Infrastructure as Code**:

You describe your infrastructure using Terraform's high-level [configuration language](https://www.terraform.io/docs/language/index.html) in human-readable, declarative configuration files. This allows you to create a blueprint that you can version, share, and reuse.

### [»](https://www.terraform.io/intro/index.html" \l "execution-plans)Execution Plans:

Terraform generates an execution plan describing what it will do and asks for your approval before making any infrastructure changes. This allows you to review changes before Terraform creates, updates, or destroys infrastructure.

### [»](https://www.terraform.io/intro/index.html" \l "resource-graph)Resource Graph:

Terraform builds a resource graph and creates or modifies non-dependent resources in parallel. This allows Terraform to build resources as efficiently as possible and gives you greater insight into your infrastructure.

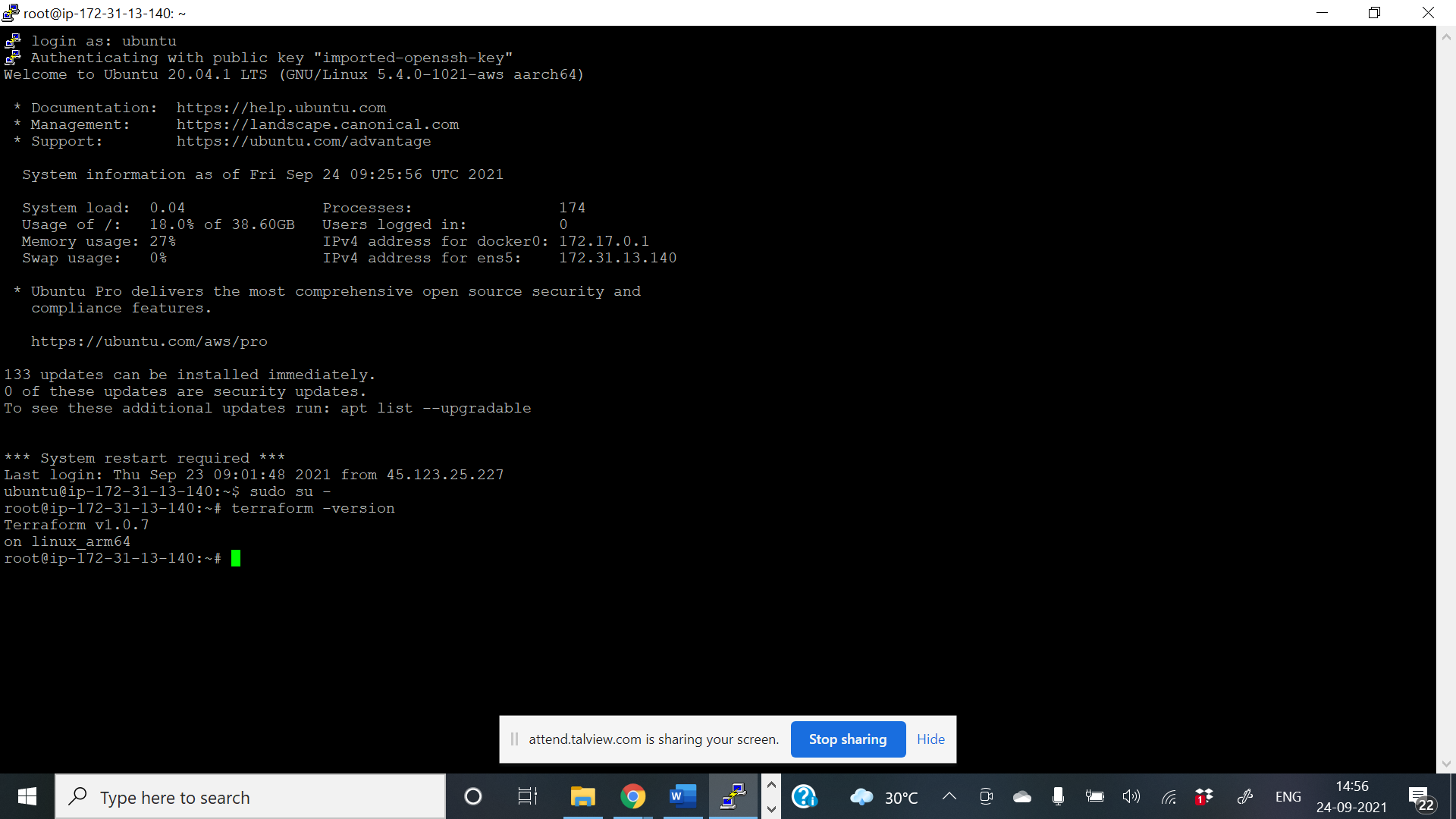
### [»](https://www.terraform.io/intro/index.html" \l "change-automation)Change Automation:

Terraform can apply complex changesets to your infrastructure with minimal human interaction. When you update configuration files, Terraform determines what changed and creates incremental execution plans that respect dependencies.

*Demonstrate the process to perform the basic operations of Terraform to provision EC2 instance in AWS*

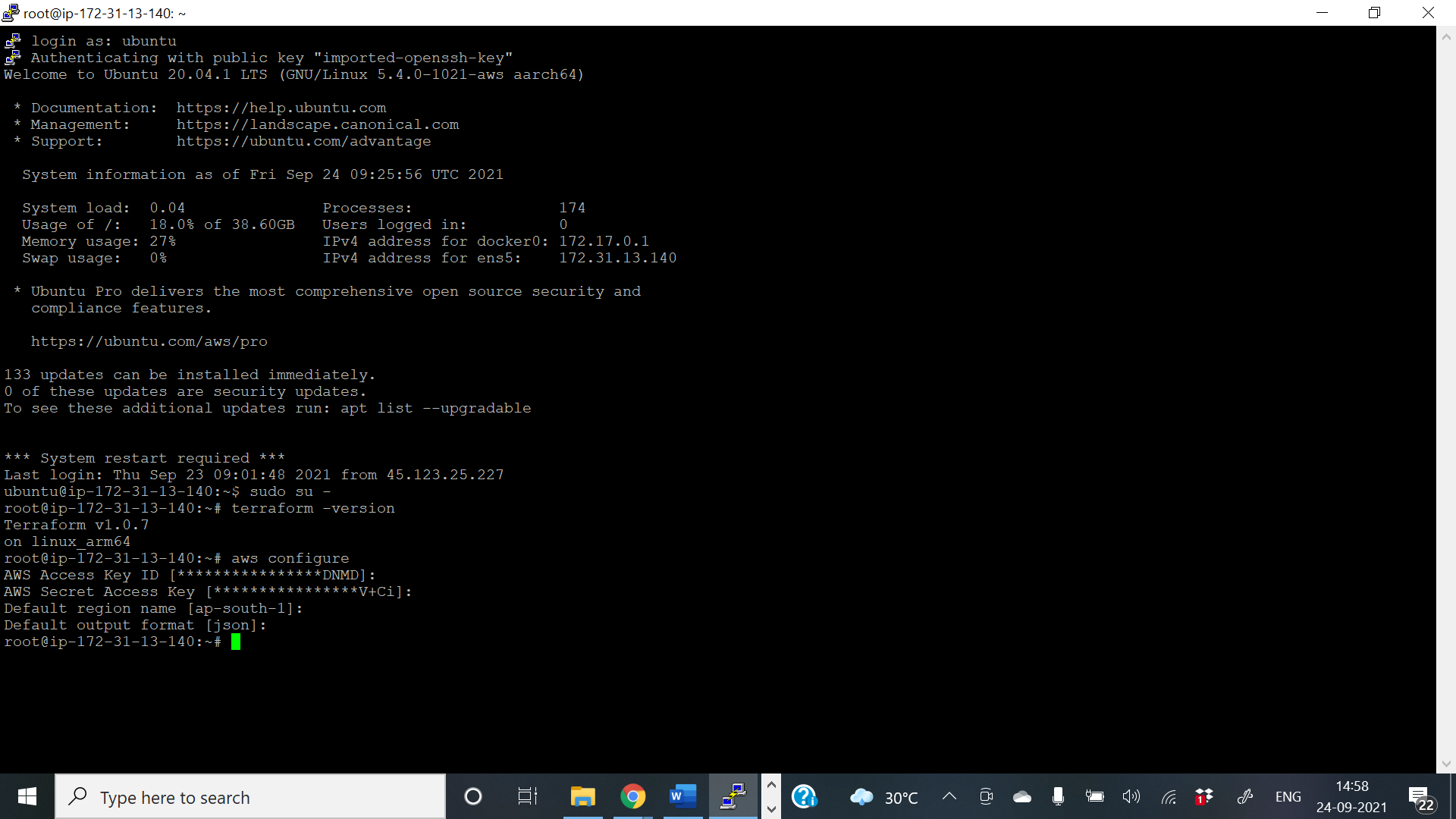
Commands to in install Terraform:

**sudo apt-get update && sudo apt-get install terraform**



**Build Infrastucre :**

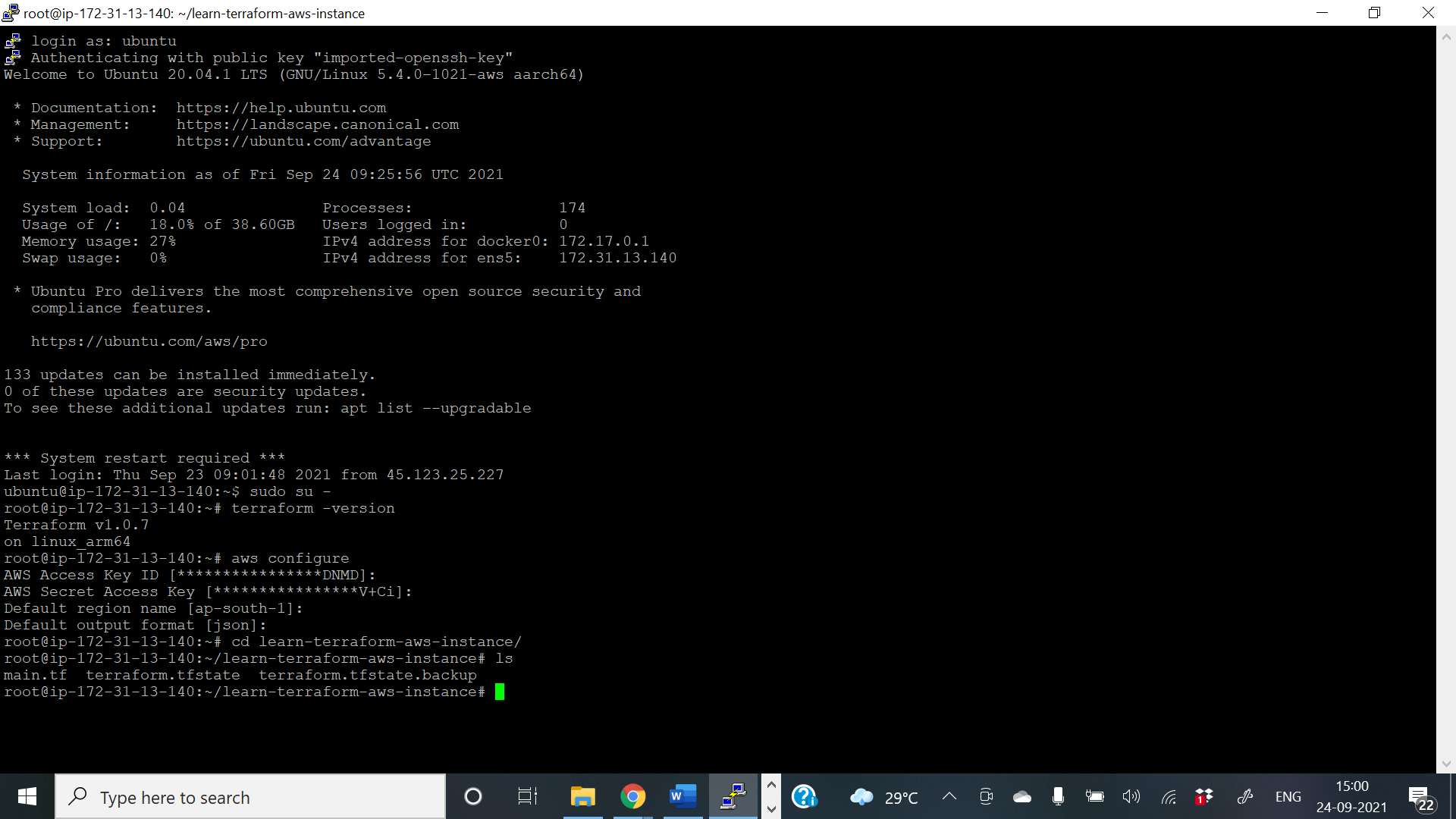
* **aws configure**

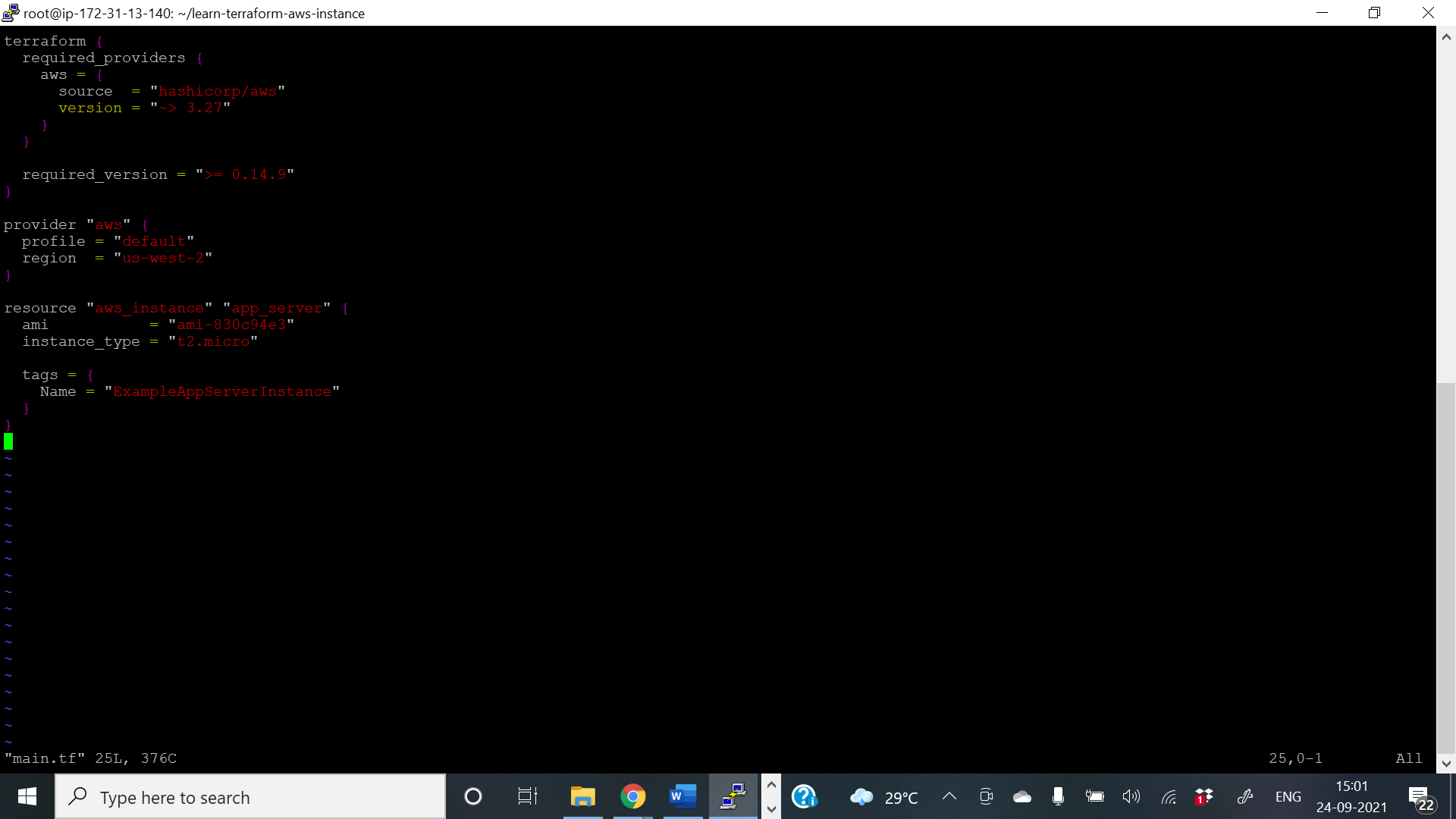


**mkdir learn-terraform-aws-instance**

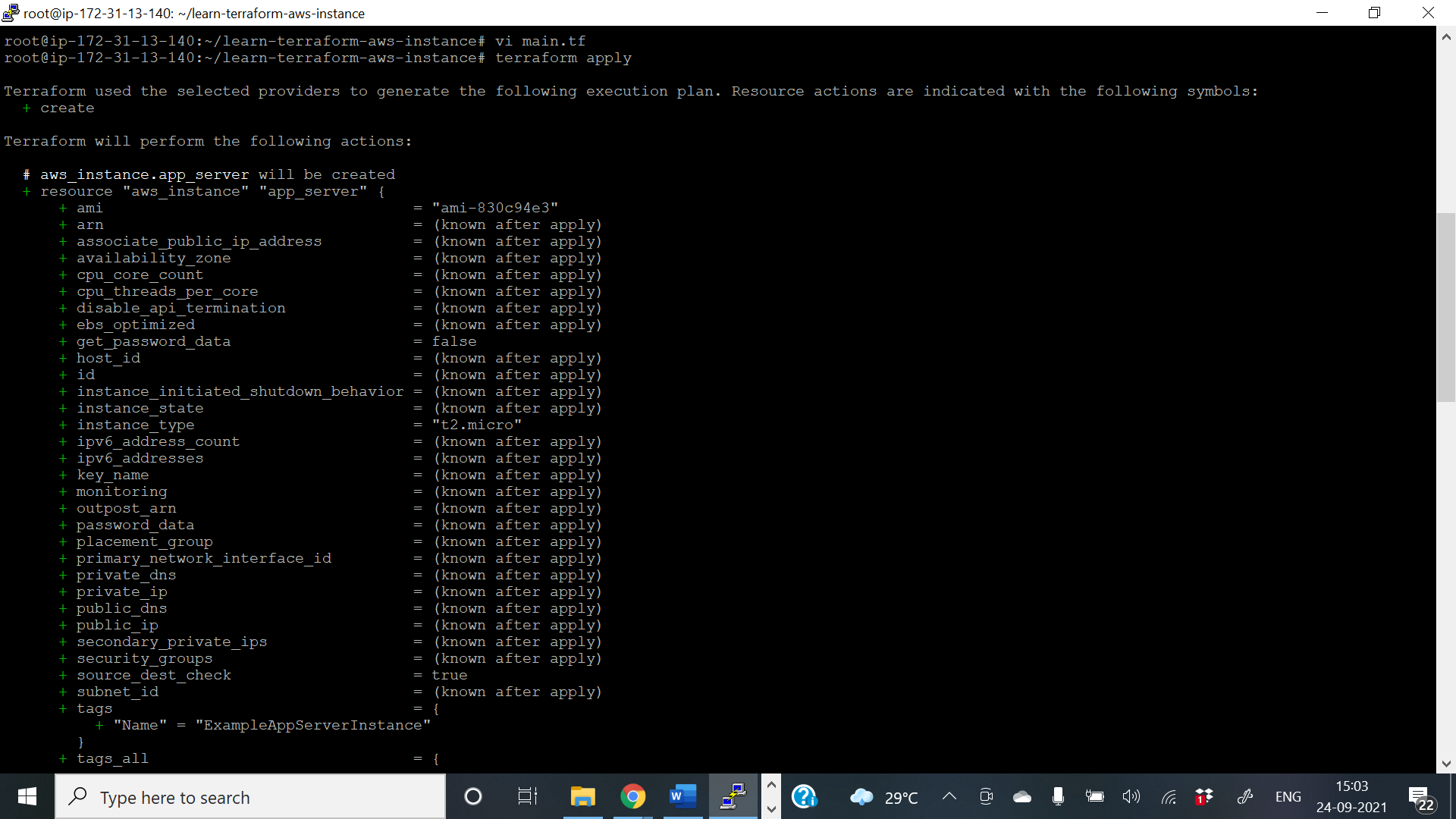
**cd learn-terraform-aws-instance**

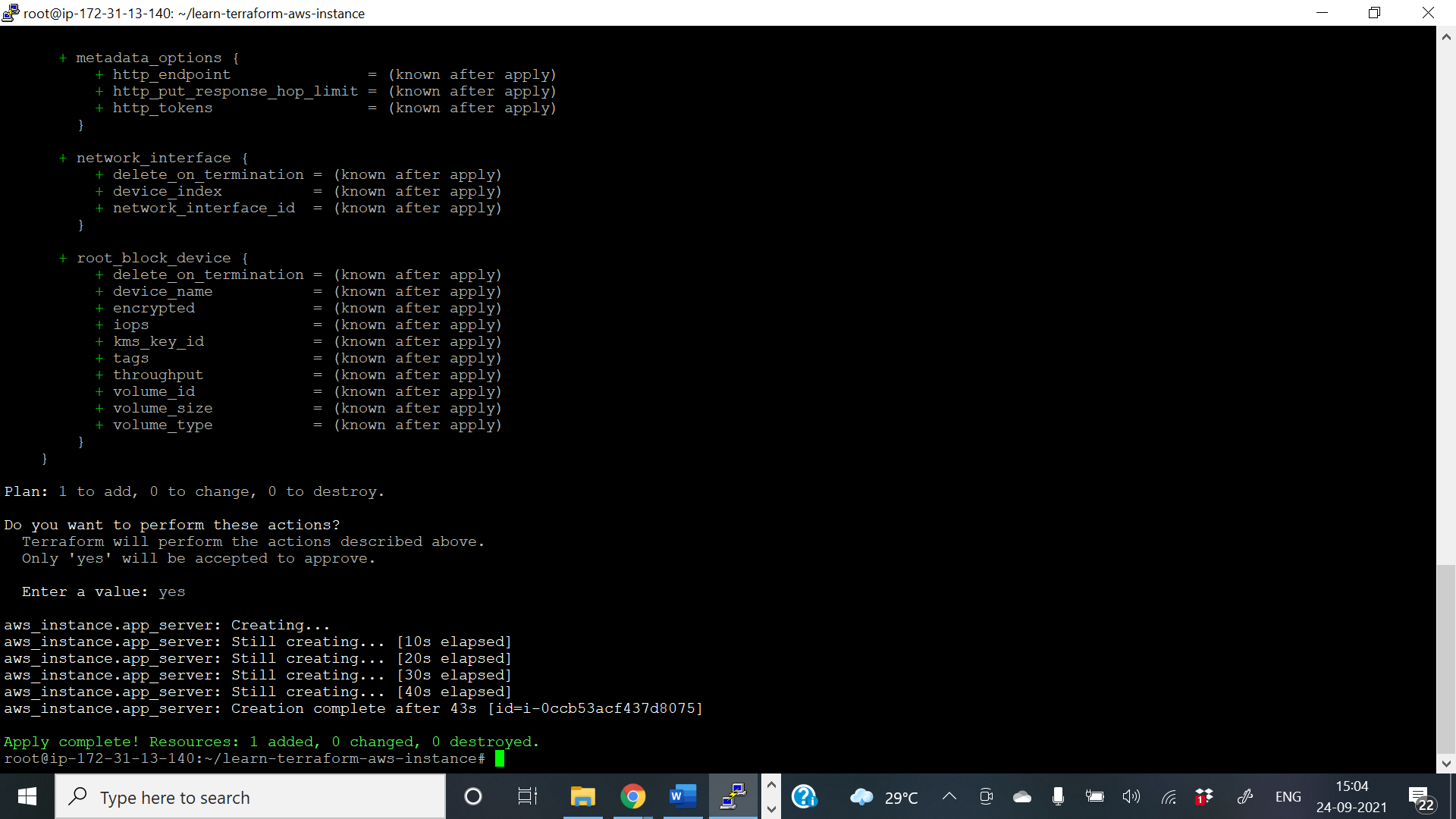
**touch main.tf**

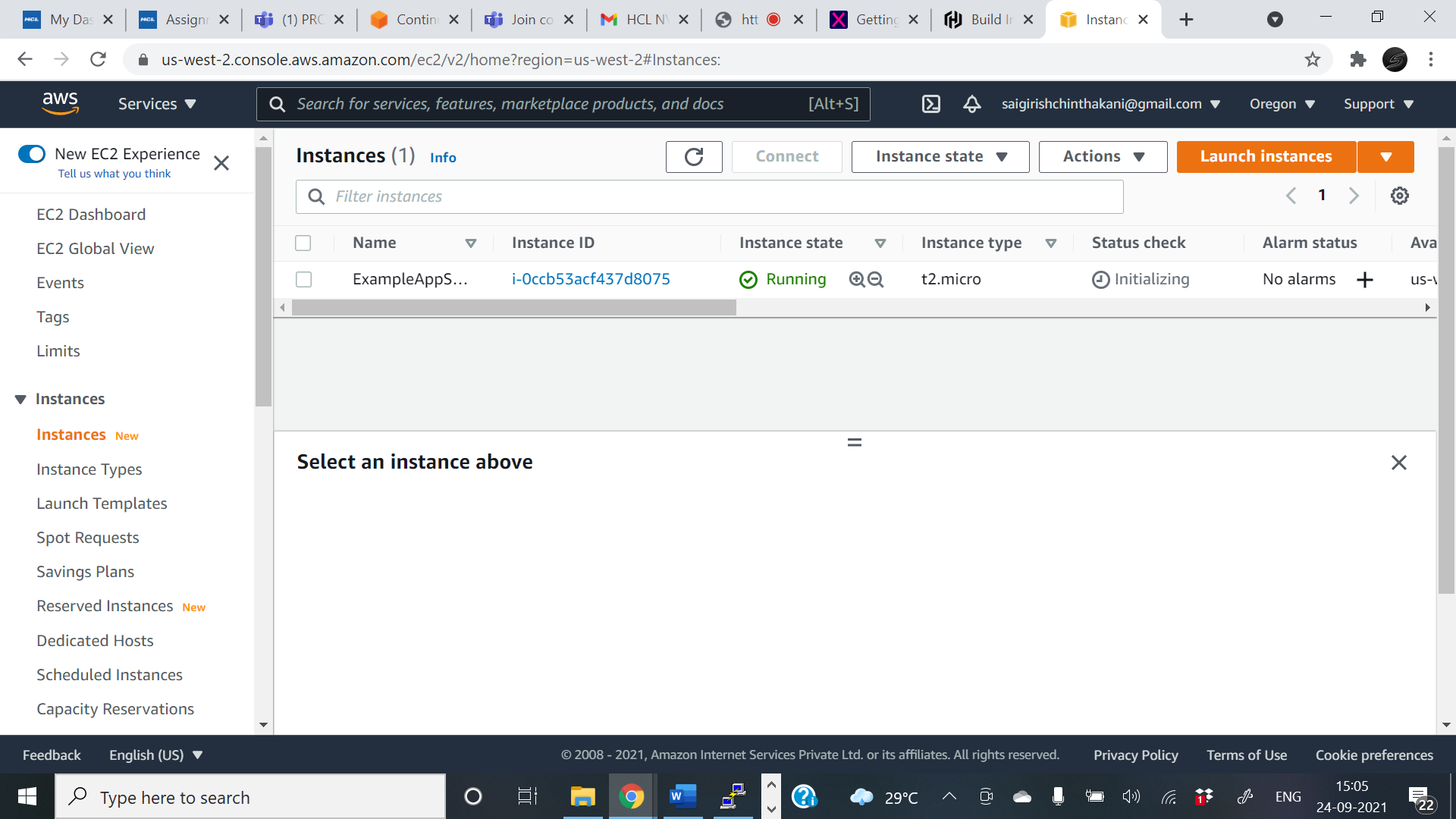




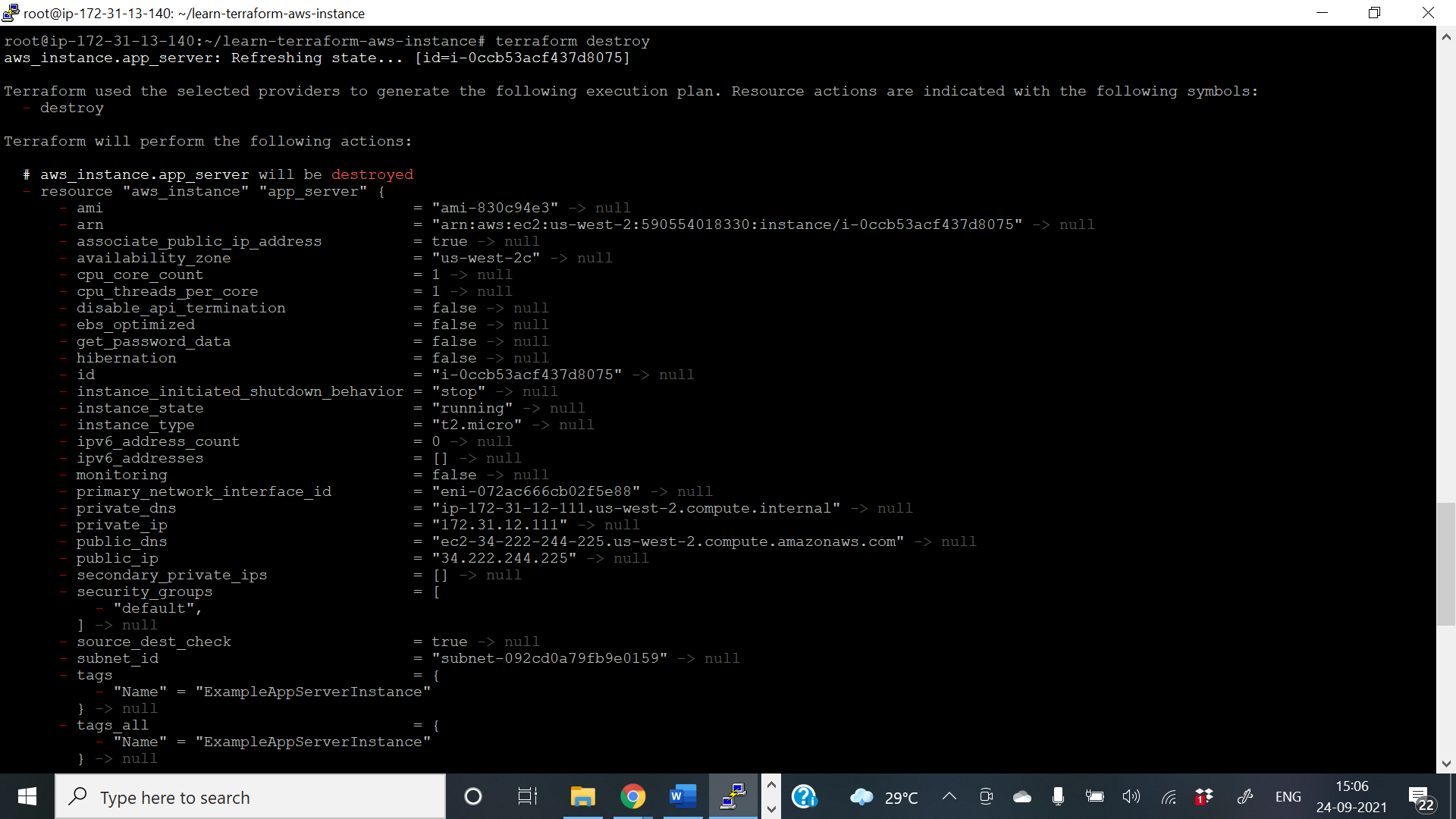
**terraform apply**

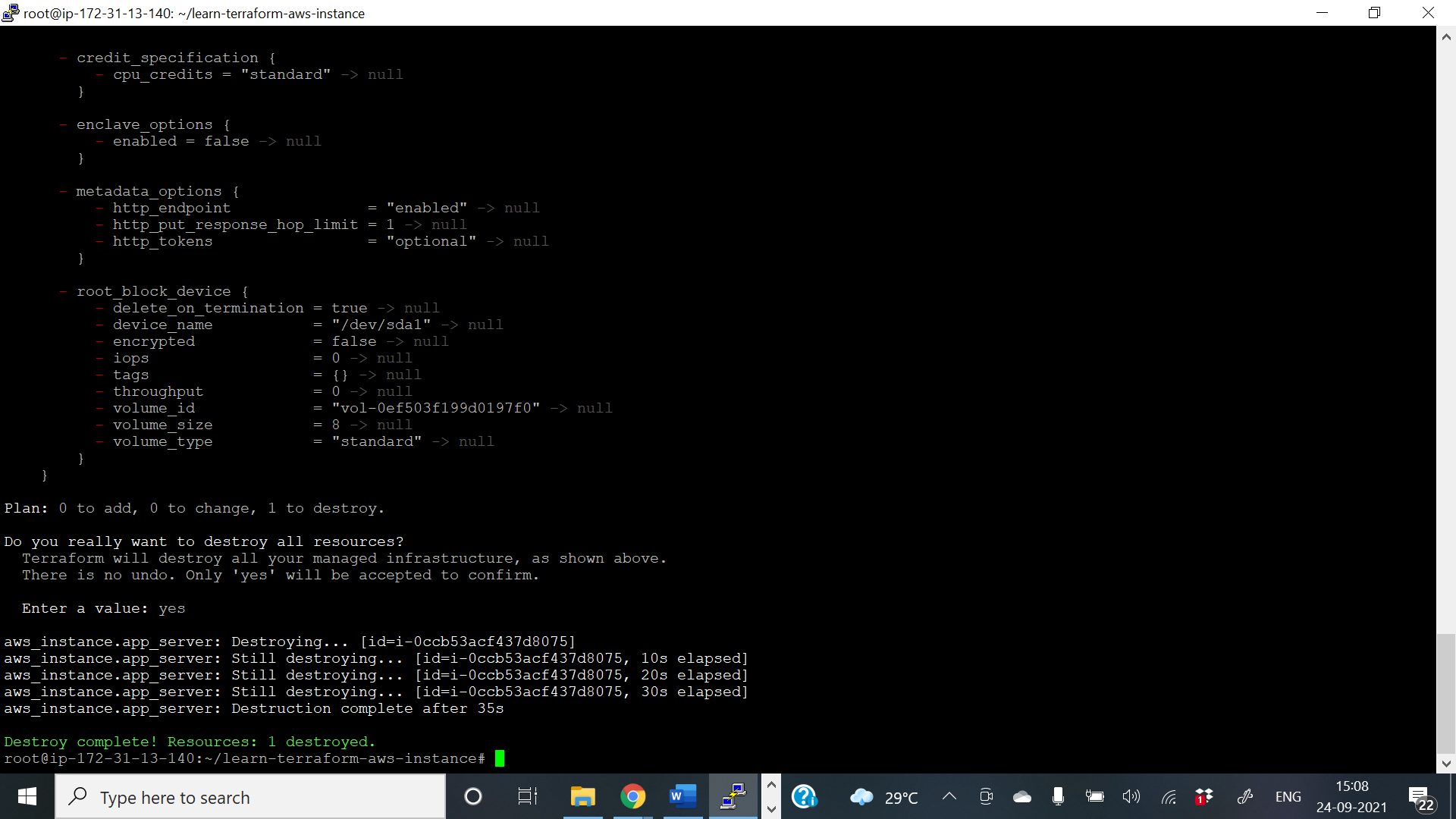


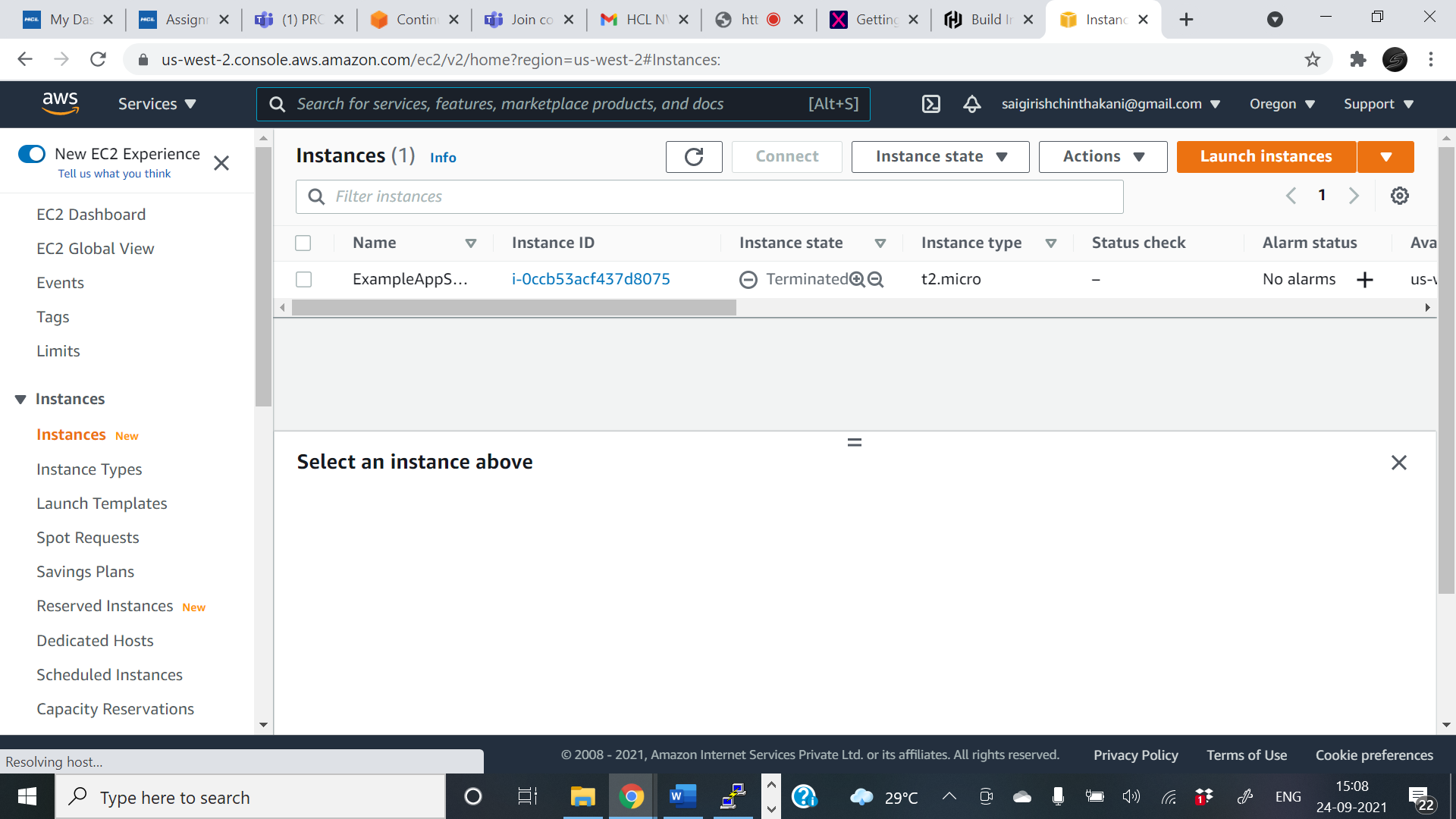




**terraform destroy**







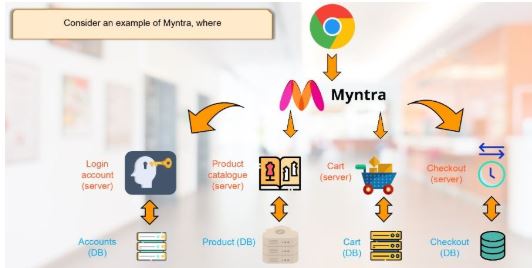
**Assignment-4                                                               Marks: 25**

Explain the concept of docker-compose and create a multi-container application using docker-compose and deploy it using Jenkins to access from web browser.

**Solution :**

## Docker Compose:

To understand Docker Compose, let’s look at Myntra as an example. Myntra is a fashion e-commerce website similar to Amazon. You visit the Myntra website through your web browser and go through several activities, like logging in to your account, browsing a catalog, checking out, and so on. Behind each of these activities or services are different products, such as an account database, product database, cart database, and others that run behind the scenes.



Each of these can be considered a microservice. The more microservices you build into your environment, the more valuable it is to have each of these services in their containers. But as a developer, you should be able to jump from one container to another. This is where you can relate this example to Docker, where Docker Compose can connect different containers as a single service.

Docker Compose is used for running multiple containers as a single service. Each of the containers here run in isolation but can interact with each other when required. Docker Compose files are very easy to write in a scripting language called YAML, which is an XML-based language that stands for Yet Another Markup Language. Another great thing about Docker Compose is that users can activate all the services (containers) using a single command.

**For example:**

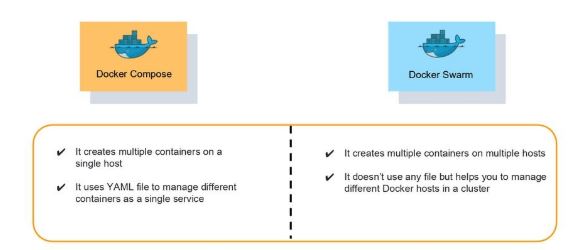
If you have an application that requires an NGINX server and Redis database, you can create a Docker Compose file that can run both the containers as a service without the need to start each one separately.



## ****Benefits of Docker Compose****

* Single host deployment - This means you can run everything on a single piece of hardware
* Quick and easy configuration - Due to YAML scripts
* High productivity - Docker Compose reduces the time it takes to perform tasks
* Security - All the containers are isolated from each other, reducing the threat landscape

Now, you might be thinking that Docker Compose is quite similar to Docker Swarm, but that’s not the case. Here are some of the differences between Docker Compose and Docker Swarm:



## **Basic Commands in Docker Compose**

* Start all services: Docker Compose up
* Stop all services: Docker Compose down
* Install Docker Compose using pip: pip install -U Docker-compose
* Check the version of Docker Compose: Docker-compose-v
* Run Docker Compose file: Docker-compose up -d
* List the entire process: Docker ps
* Scale a service - Docker Compose up -d -scale
* Use YAML files to configure application services - Docker Compose.yml

| **Command** | **Purpose** |
| --- | --- |
| build | Build or rebuild services |
| help | Get help on a command |
| kill | Kill containers |
| logs | View output from containers |
| port | Print the public port for a port binding |
| ps | List containers |
| pull | Pulls service images |
| restart | Restart services |
| rm | Remove stopped containers |
| run | Run a one-off command |
| scale | Set number of containers for a service |
| start | Start services |
| stop | Stop services |
| up | Create and start containers |

The application used in this section is built based on myWebapp communicating with a Postgresql database. When the page is loaded, it will query the Student table for the record with ID and display the name of student on the page.

The myWebApp and Postgresql will be running in two separate containers, and thus making this a multi-container application.

Update the myWebapp

1. adding package to allow app taking to database

cd myWebapp

dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

1. create student model

* Create a Models folder in the project folder
* Create Models/Student.cs with the following code:

using System;

using System.Collections.Generic;

namespace myWebApp.Models

{

public class Student

{

public int ID { get; set; }

public string LastName { get; set; }

public string FirstMidName { get; set; }

public DateTime EnrollmentDate { get; set; }

}

}

1. Create the SchoolContext with the following code

using Microsoft.EntityFrameworkCore;

namespace myWebApp.Data

{

public class SchoolContext : DbContext

{

public SchoolContext (DbContextOptions<SchoolContext> options)

: base(options)

{

}

public DbSet<Models.Student> Students { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Models.Student>().ToTable("Student");

}

}

}

1. Regsister SchoolContext to DI in Startup.cs

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<myWebApp.Data.SchoolContext>(options =>

options.UseNpgsql(Configuration.GetConnectionString("SchoolContext")));

services.AddRazorPages();

}

1. Adding database connectionstring to appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"SchoolContext": "Host=db;Database=my\_db;Username=postgres;Password=example"

}

}

1. Boostrap the table if it does not exist in Program.cs

public static void Main(string[] args)

{

var host= CreateHostBuilder(args).Build();

CreateDbIfNotExists(host);

host.Run();

}

private static void CreateDbIfNotExists(IHost host)

{

using (var scope = host.Services.CreateScope())

{

var services = scope.ServiceProvider;

try

{

var context = services.GetRequiredService<Data.SchoolContext>();

context.Database.EnsureCreated();

// DbInitializer.Initialize(context);

}

catch (Exception ex)

{

var logger = services.GetRequiredService<ILogger<Program>>();

logger.LogError(ex, "An error occurred creating the DB.");

}

}

}

Configuration file

The entry point to Docker Compose is a Compose file, usually called docker-compose.yml. In project directory, create a new file docker-compose.yml in it. Use the following contents:

services:

db:

image: postgres

restart: always

environment:

POSTGRES\_PASSWORD: example

volumes:

- postgres-data:/var/lib/postgresql/data

adminer:

image: adminer

restart: always

ports:

- 8080:8080

app:

build:

context: .

dockerfile: ./Dockerfile

ports:

- 5000:80

depends\_on:

- db

volumes:

postgres-data:

In this Compose file:

* Two services in this Compose are defined by the name db and web attributes; the adminer service is a helper for us to access db
* Image name for each service defined using image attribute
* The postgres image starts the Postgres server.
* environment attribute defines environment variables to initialize postgres server.
  + POSTGRES\_PASSWORD are used set that default user,**postgres** password. This user will be granted superuser permissions for the database **my\_db** in the connectionstring.
* app application uses the db service as specified in the connection string
* The app image is built using the Dockerfile in the project directory
* Port forwarding is achieved using ports attribute.
* depends\_on attribute allows to express dependency between services. In this case, Postgres will be started before app. Application-level health check are still user’s responsibility.

Start application

All services in the application can be started, in detached mode, by giving the command:

docker-compose up -d

An alternate Compose file name can be specified using -f option.

An alternate directory where the compose file exists can be specified using -p option.

This shows the output as:

docker-compose up -d

Starting mywebapp\_adminer\_1 ... done

Starting mywebapp\_db\_1 ... done

Starting mywebapp\_app\_1 ... done

The output may differ slightly if the images are downloaded as well.

Started services can be verified using the command docker-compose ps:

docker-compose ps

Name Command State Ports

------------------------------------------------------------------------------------

mywebapp\_adminer\_1 entrypoint.sh docker-php-e ... Up 0.0.0.0:8080->8080/tcp

mywebapp\_app\_1 ./myWebApp Up 0.0.0.0:5000->80/tcp

mywebapp\_db\_1 docker-entrypoint.sh postgres Up 5432/tcp

This provides a consolidated view of all the services, and containers within each of them.

Alternatively, the containers in this application, and any additional containers running on this Docker host can be verified by using the usual docker container ls command:

docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ee35a9399b80 mywebapp\_app "./myWebApp" 29 minutes ago Up About a minute 0.0.0.0:5000->80/tcp mywebapp\_app\_1

0fc85278791c postgres "docker-entrypoint.s…" 30 minutes ago Up About a minute 5432/tcp mywebapp\_db\_1

a9c725d0e684 adminer "entrypoint.sh docke…" 30 minutes ago Up About a minute 0.0.0.0:8080->8080/tcp mywebapp\_adminer\_1

Service logs can be seen using docker-compose logs command, and looks like:

docker container logs mywebapp\_db\_1

PostgreSQL Database directory appears to contain a database; Skipping initialization

2021-03-16 04:19:51.862 UTC [1] LOG: starting PostgreSQL 13.2 (Debian 13.2-1.pgdg100+1) on x86\_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit

2021-03-16 04:19:51.863 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432

2021-03-16 04:19:51.863 UTC [1] LOG: listening on IPv6 address "::", port 5432

2021-03-16 04:19:51.868 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"

2021-03-16 04:19:51.875 UTC [29] LOG: database system was shut down at 2021-03-16 04:19:04 UTC

2021-03-16 04:19:51.884 UTC [1] LOG: database system is ready to accept connections

2021-03-16 04:20:03.442 UTC [1] LOG: received fast shutdown request

2021-03-16 04:20:03.444 UTC [1] LOG: aborting any active transactions

2021-03-16 04:20:03.446 UTC [1] LOG: background worker "logical replication launcher" (PID 35) exited with exit code 1

2021-03-16 04:20:03.447 UTC [30] LOG: shutting down

2021-03-16 04:20:03.473 UTC [1] LOG: database system is shut down

PostgreSQL Database directory appears to contain a database; Skipping initialization

2021-03-16 04:20:53.597 UTC [1] LOG: starting PostgreSQL 13.2 (Debian 13.2-1.pgdg100+1) on x86\_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit

2021-03-16 04:20:53.597 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432

2021-03-16 04:20:53.597 UTC [1] LOG: listening on IPv6 address "::", port 5432

2021-03-16 04:20:53.601 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"

2021-03-16 04:20:53.606 UTC [26] LOG: database system was shut down at 2021-03-16 04:20:03 UTC

2021-03-16 04:20:53.618 UTC [1] LOG: database system is ready to accept connections

2021-03-16 04:21:31.054 UTC [38] ERROR: invalid input syntax for type timestamp: "" at character 91

2021-03-16 04:21:31.054 UTC [38] STATEMENT: INSERT INTO "Student" ("LastName", "FirstMidName", "EnrollmentDate")

VALUES ('YHH', 'HH', '')

2021-03-16 04:33:09.323 UTC [1] LOG: received fast shutdown request

2021-03-16 04:33:09.325 UTC [1] LOG: aborting any active transactions

2021-03-16 04:33:09.327 UTC [1] LOG: background worker "logical replication launcher" (PID 32) exited with exit code 1

2021-03-16 04:33:09.329 UTC [27] LOG: shutting down

2021-03-16 04:33:09.342 UTC [1] LOG: database system is shut down

PostgreSQL Database directory appears to contain a database; Skipping initialization

2021-03-16 04:49:23.844 UTC [1] LOG: starting PostgreSQL 13.2 (Debian 13.2-1.pgdg100+1) on x86\_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit

2021-03-16 04:49:23.844 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432

2021-03-16 04:49:23.844 UTC [1] LOG: listening on IPv6 address "::", port 5432

2021-03-16 04:49:23.849 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"

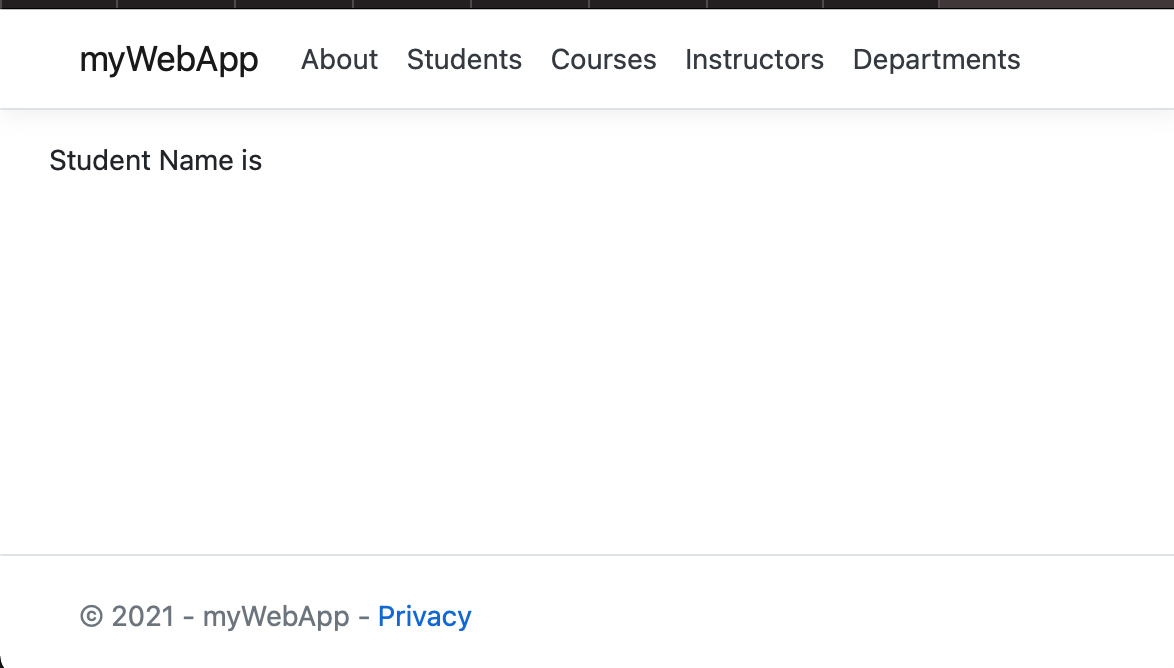
2021-03-16 04:49:23.855 UTC [26] LOG: database system was shut down at 2021-03-16 04:33:09 UTC

2021-03-16 04:49:23.862 UTC [1] LOG: database system is ready to accept connections

Verify application

Let’s access the application. In your browser address bar type http://localhost:5000

you will see the page show no student name since the database is empty.

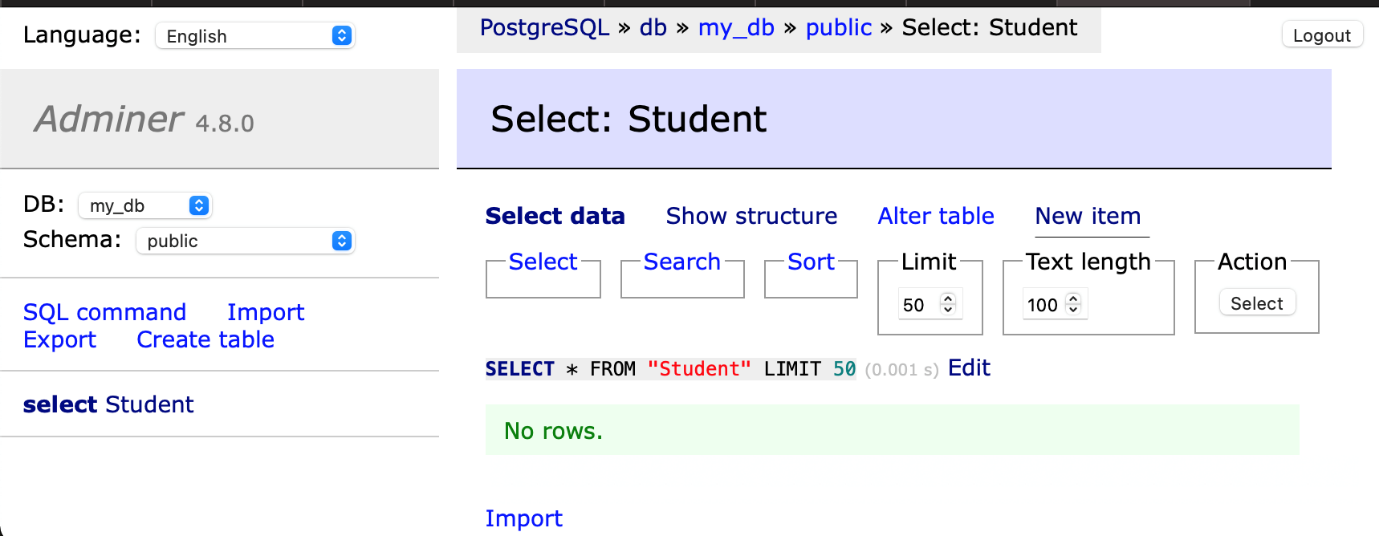


now open another tab with address http://localhost:8080 and you will be asked to login

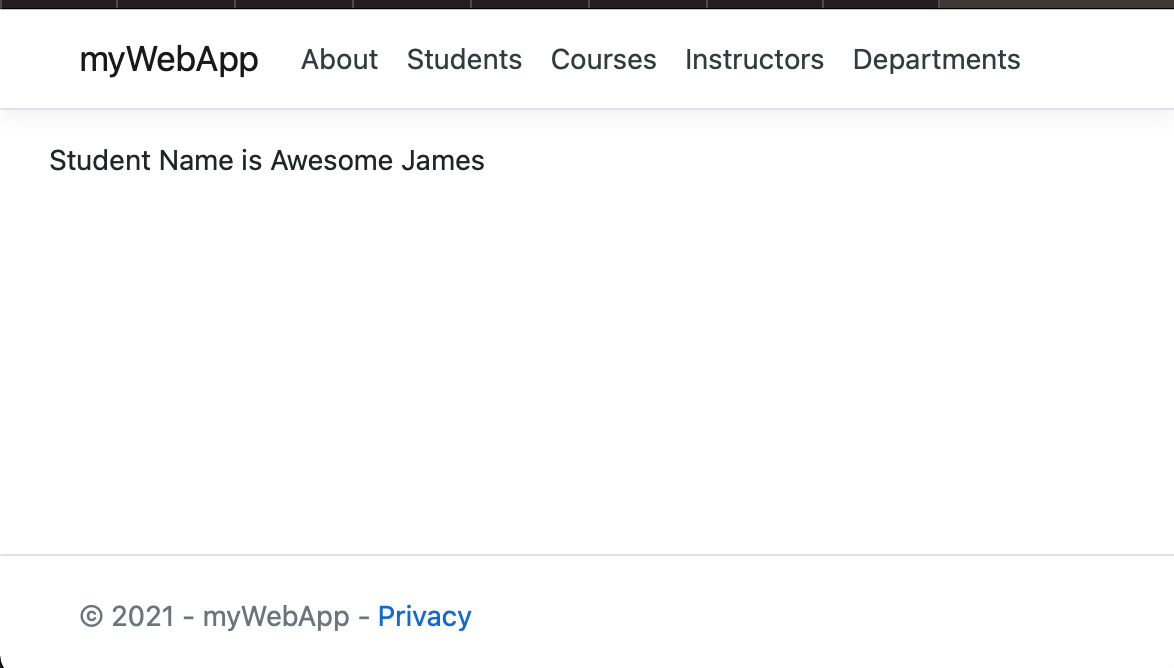


use **postegres** and **example** as username/passowrd to login my\_db.

once you are logged in, you can create a new student record



Now, refresh the app page at http://localhost:5000, the new added student name will be there,



Shutdown application

Shutdown the application using docker-compose down:

docker-compose down

Stopping mywebapp\_app\_1 ... done

Stopping mywebapp\_db\_1 ... done

Stopping mywebapp\_adminer\_1 ... done

Removing mywebapp\_app\_1 ... done

Removing mywebapp\_db\_1 ... done

Removing mywebapp\_adminer\_1 ... done

Removing network mywebapp\_default

This stops the container in each service and removes all the services. It also deletes any networks that were created as part of this application.

**Assignment-5                                                              Marks: 25**

Create a windows-based java/spring-boot project in eclipse and push it to GitLab and deploy the same using Jenkins windows-node to tomcat container.

**Solution :**

I created a java based project

