1 - Creating an ec2 instance and connecting it to our machine:

First step is to create an ec2 instance and connect it to my machine, in order to do that I must:

* Create the ec2 instance
* Relax the security groups
* Connect my machine to the ec2 instance via ssh

To do so we can use terraform and ansible:

* To create the instance we create a folder called terraform where we include the following in provider.tf where the access and secret keys as well as the token are the ones specified for aws access:

provider "aws" {

  # Configuration options

  region     = "us-east-1"

access\_key = “ASIARXVVIWXW34LGJN4V”

secret\_key = “C4Gy/vMu1MlWLd4G7LWw1EyhISTNZyYAmVylOJ7c”

token = “FwoGZXIvYXdzEOH//////////wEaDJ24AGFygNxLzjJr+yK9ATJlvxeyrGbysO/NEPxxcvpIrS0eL+QQ8r8WuwFyj/MfvY5AOSOmEdDYgr/I8KYfukDSiiXsWDqvHL5xHrFRItbp/qNMCBagFD8SfxFi2SOJVYNXxEDyxuriJw06o0HxNuFZdVk/JWIhDl4774VkcB5h9iVyh1Infan+aQ6Wiqh9opGtIgCLp2CcAOKNe8uScsxGm7b9+zeD2tH1dSpjJQTF5myaHHAvEsFLAN4p/BkwlxRcjTo91kcbH0MFuCjh3rCgBjItAS4e+NP8jBW7BN3hhXR4npANt/cDsJdboI3hIT/GulElg+f+tXK3Q7f0jsjh”

}

terraform {

  required\_providers {

    aws = {

      source = "hashicorp/aws"

      version = "4.40.0"

    }

  }

}

* To connect via ssh we add the following to the same provider.tf where the public key is generated via :

Resource “aws\_key\_pair” "deployer" {

  key\_name   = "cep1"

public\_key = " ssh-rsa  mohieddinekhale@ip-172-31-25-28”

}

* Adjusting the instance’s settings like the security group, ami, instance type and the egress using the key that was already created:

resource "aws\_security\_group" "sg" {

  name        = "sgcep1"

  description = "Allow All Traffic"

  ingress {

    from\_port        = 0

    to\_port          = 0

    protocol         = "-1"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  egress {

    from\_port        = 0

    to\_port          = 0

    protocol         = "-1"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

}

resource "aws\_instance" "web" {

  ami           = "ami-08c40ec9ead489470"

  instance\_type = "t2.micro"

  tags = {

    Name = "CEP1"

  }

  key\_name = "cep1"

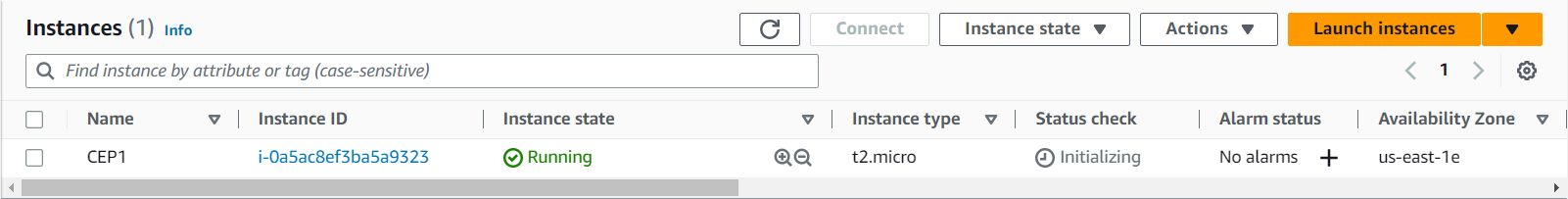
  security\_groups = ["sgcep1"]

}

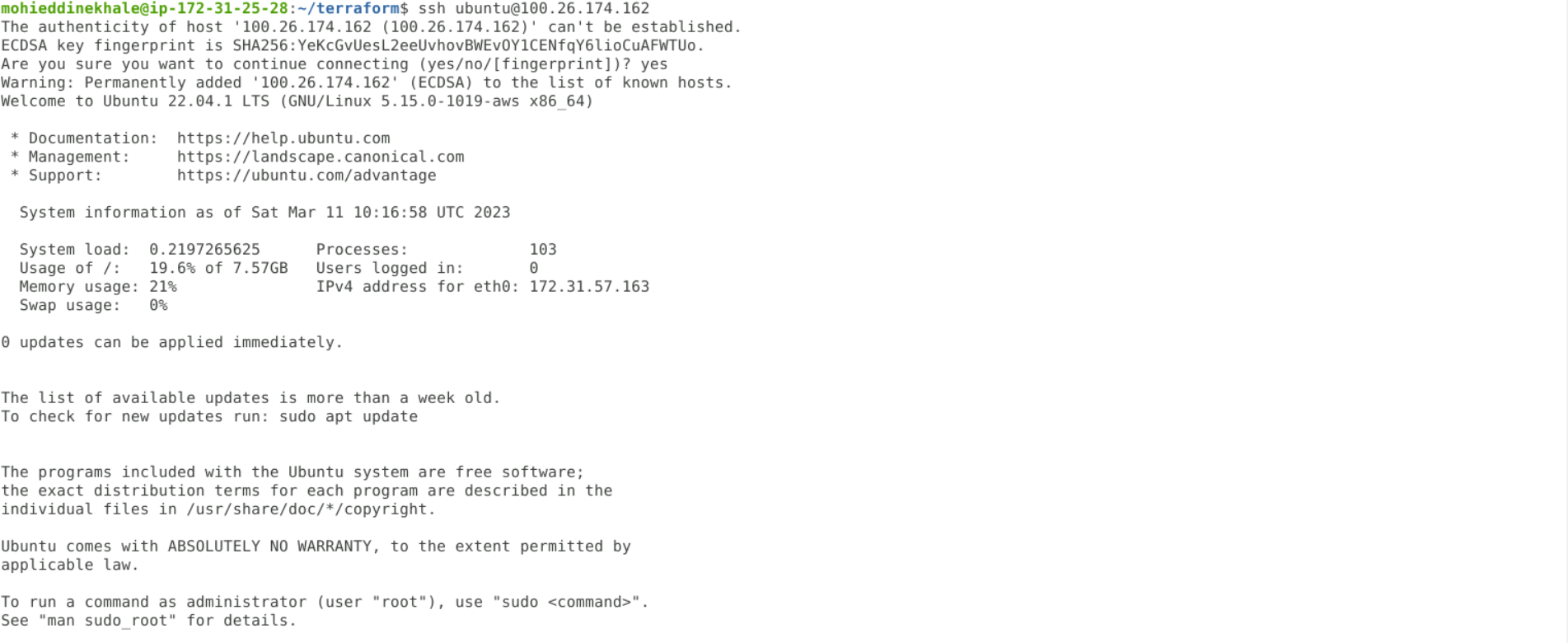
In order to be able to execute this tf file on the ec2 instance we need to use:

terraform apply

The result is the following:



To verify the connection between our machine and the ec2 instance:



In order to be able to use ansible to install Jenkins on our ec2 instance we need to add our instance to our ansible hosts by adding this line:

project ansible\_host= 34.201.24.37 ansible\_user=ubuntu

To the following file: /etc/ansible/hosts

2 – Downloading Jenkins on the ec2 instance:

With this being done our ec2 instance is created and its connection to our machine is confirmed

The next step is to download Jenkins using the following jenkins.yaml file:

|  |
| --- |
| - name: "course end project"    hosts: project    become: yes    tasks:            - name: "install java"              apt:                      name: openjdk-11-jdk                      update\_cache: yes            - name: "install Python"              apt:                      name: python3            - name: "configure system to install jenkins"              shell:                      cmd: 'curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null'            - name: "configure system to install jenkins -1"              shell:                      cmd: 'echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null'            - name: "install Jenkins"              apt:                      name: jenkins                      update\_cache: yes |

Then we run the following command: ansible-playbook jenkins.yaml.

3 - Use the required plugins to run the build creation on a containerized platform:

Jenkins has a number of plugins that can be used to create and manage Docker containers, including the Docker Pipeline plugin and the Docker Build and Publish plugin.

To use these plugins, we’ll need to install them on our Jenkins instance and configure our build pipeline to create and manage Docker containers as part of the build process.

To install the Docker Plugin, follow these steps:

* Log in to Jenkins and click on "Manage Jenkins" in the left-hand menu.
* Click on "Manage Plugins" and select the "Available" tab.
* Search for "Docker Plugin" and select it.
* Click on "Install without restart" and wait for the plugin to be installed.
* Repeat the above steps to install the Amazon ECR Plugin.

Once the plugins are installed, we need to configure Jenkins to use Docker to build and test the application. We can do this by creating a Jenkinsfile, which is a text file that defines the steps of the build process.

Since we know that our application will be deployed using a yaml file, the test performed should have a form similar to the following, in order to test the contents of the yaml before deployment:

pipeline {

agent any

stages {

stage('Validate YAML syntax') {

steps {

sh 'yamllint my-deployment.yaml'

}

}

stage('Deploy container') {

steps {

// Include the steps to deploy the container

}

}

// Include additional stages as needed

}

}

4 – Create a Docker image that contains the application I need to deploy on the ec2 instance:

In order to do this, assuming we have a .war file that contains the application we must do the following:

mkdir nginx-docker

cd nginx-docker

Create a new file called Dockerfile and add the following content:

FROM nginx:latest

COPY . /usr/share/nginx/html

This Dockerfile specifies that we want to use the latest version of the official NGINX image as the base image for our container. We then copy the contents of the current directory into the /usr/share/nginx/html directory in the container.

Now we need to build the docker image using docker build

docker build -t my-nginx-image

This command tells Docker to build a new image using the Dockerfile in the current directory, and to tag the image with the name "my-nginx-image".

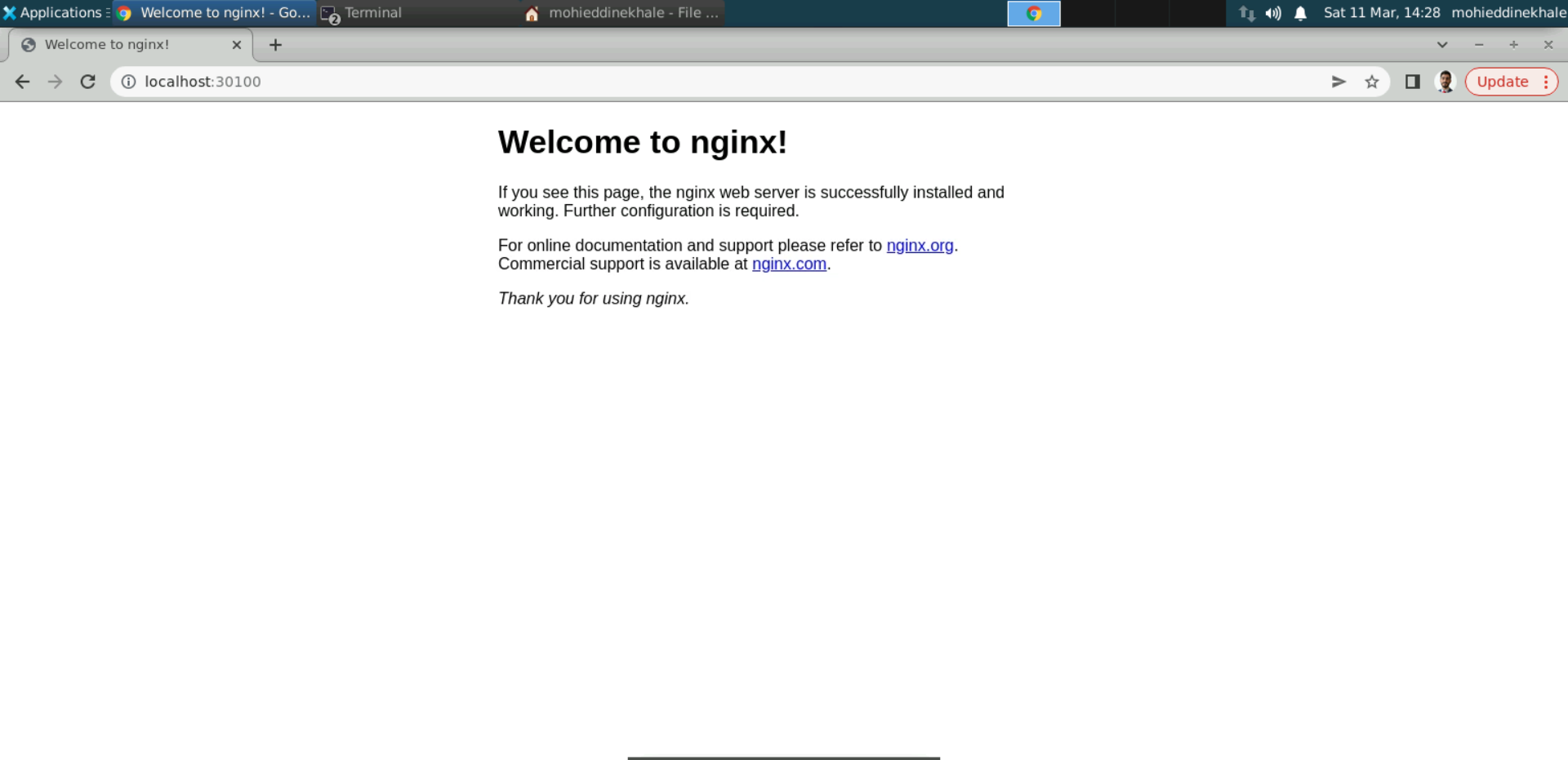


Finally we run the Docker container using the following command:

docker run -d -p 30100:80 my-nginx-image

This command starts a new container using the "my-nginx-image" image, and maps port 30100 on the host machine to port 80 in the container. The "-d" option tells Docker to run the container in the background.

The following image shows what is expected when things go correctly:



With this being done we can create a nginx.yaml file to transfer the docker container from our machine to the ec2 instance:

---

- name: Deploy NGINX container to EC2 instance

hosts: project

become: true

vars:

container\_name: my-nginx-container

image\_name: my-nginx-image

port\_mappings: "30100:80"

jenkins\_url: http://localhost:8080/

jenkins\_job\_name: my-nginx-test

git\_repo\_url: https://github.com/mohieddinekhaled/my-nginx-repo.git

tasks:

- name: Install Docker

apt:

name: docker.io

state: present

- name: Start Docker service

systemd:

name: docker

state: started

- name: Pull Docker image

docker\_image:

name: "{{ image\_name }}"

state: present

- name: Create Docker container

docker\_container:

name: "{{ container\_name }}"

image: "{{ image\_name }}"

state: started

restart\_policy: always

published\_ports: "{{ port\_mappings }}"

- name: Run Jenkins test

uri:

url: "{{ jenkins\_url }}/job/{{ jenkins\_job\_name }}/build?delay=0sec"

method: POST

user: admin

password: admin

return\_content: yes

register: response

until: response.status == 201 or response.status == 302

retries: 10

delay: 10

- name: Login to Docker Hub docker\_login:

registry: https://index.docker.io/v1/

username: myusername

password: mypassword -

- name: Tag Docker image docker\_image:

name: "{{ image\_name }}"

tag: latest

repository: "{{ git\_repo\_url }}"

- name: Push Docker image

docker\_push:

name: "{{ git\_repo\_url }}:latest"

- name: Stop and remove Docker container

docker\_container:

name: "{{ container\_name }}"

state: stopped

force\_kill: yes

- name: Remove Docker container

docker\_container:

name: "{{ container\_name }}"

state: absent

This yaml file pulls the NGINX Docker image and then runs a container based on that image on the EC2 instance, mapping port 80 of the container to port 30100 on the host. This allows users to access the NGINX web server running inside the container by connecting to port 30100 on the EC2 instance and finally runs the Jenkins test that we defined earlier. It also pushes the contents of the container to our private git repository and removes the stack once done.

Now all we need to do is run the following command: ansible-playbook nginx.yaml

5 – Suggestions to improve the project:

* Security: Implement security measures like using SSL/TLS for data encryption, adding firewalls, and implementing user authentication and authorization.
* Monitoring: Consider implementing monitoring tools to ensure the system is running smoothly, such as Prometheus or Grafana.
* Use of containers: Consider using more containers to isolate different components of the application, as this can help with scalability and maintainability.
* Use of microservices architecture: Consider using microservices architecture to create smaller, loosely coupled services that can be easily managed and scaled independently.
* Automated testing: Consider adding more automated testing, such as integration and end-to-end testing, to ensure that the application is working correctly and catch bugs early in the development process.
* Documentation: Create detailed documentation that includes instructions for building, deploying, and maintaining the application, including how to troubleshoot common issues.