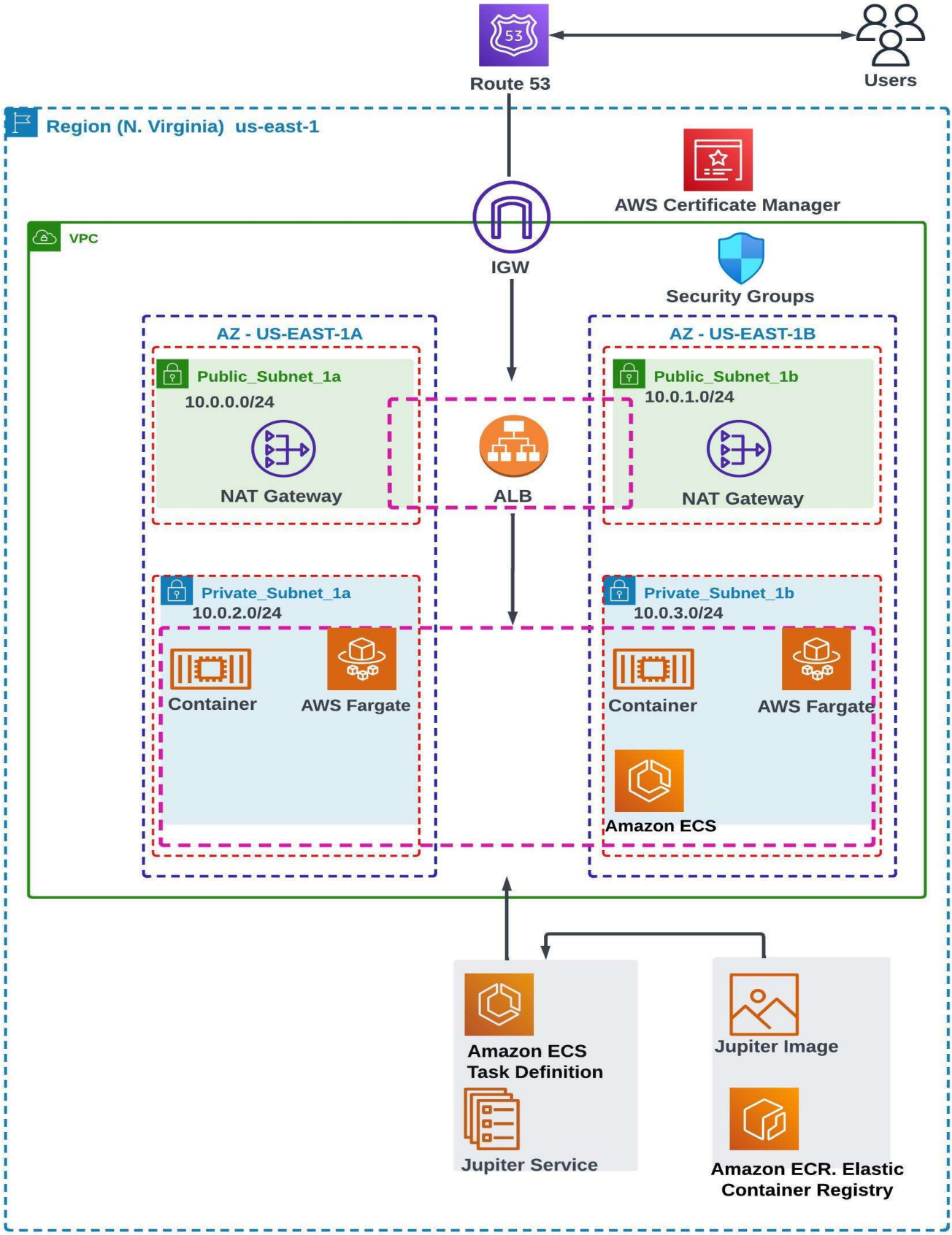
# DEPLOYMENT OF JUPITER WEB APP USING DOCKER, AMAZON ECR AND AMAZON ECS ON AWS



## RESOURCES TO COMPLETE THIS PROJECT ARE:

1.VPC WITH PUBLIC AND PRIVATE SUBNETS

2.NAT GATEWAYS

3.SECURITY GROUPS

4.APPLICATION LOAD BALANCER

5.AMAZON ELASTIC CONTAINER REGISTRY ECR

6.AMAZON ELASTIC CONTAINER SERVICE

7.CERTIFICATE MANAGER

8.ROUTE 53

## DEVOPS TOOLS AND RESOURCES

1.DOCKER

2.DOCKER HUB

3.GIT

4.GIT HUB

5.VISUAL STUDIO CODE

6.POWERSHELL

## PREREQUISITES FOR THIS PROJECT ARE:

1.Install Visual Studio code on your local machine/computer

2.Install Git on your machine

3.Sign up for a free account on GitHub

4.Create a private key pair using PowerShell and upload the public SSH Key to GitHub

5.Create a new GitHub Repository to store the Docker file

6.Clone the GitHub Repository to your local machine

7.Create a Docker hub account

8.Download and install Docker on your machine

9.Create Docker file

10.Build the container image

11.Start the container

12.Create a Repository in your Docker Hub account

13.Push the image to your Docker Hub Repository

14.Install AWS CLI on your computer

15.Create IAM User and Named Profile

16.Create an Amazon ECR Repository to store your image

17.Push the image to your Amazon ECR Repository

18.Create a VPC with Public and Private Subnets in two Availability Zones (AZ)

19.Create Security Groups

20.Create an Application Load Balancer (ALB) that will distribute traffic to the containers

21.Create an ECS Cluster

22.Create a Task Definition

23.Create a Service

24.We will use Amazon Route 53 to register our Domain name and create a record set.

25.We will request an SSL/TLS Certificate from Amazon Certificate Manager (ACM) to enable us to communicate securely to our website

26.Important to note that AWS resources such as NAT Gateways, Application Load Balancers, and Bastion Hosts must reside and interact with the public subnets.

## STEP 1: CREATION OF GITHUB ACCOUNT

I already have a GitHub account setup for my previous projects which I will also be using for this project, so I will not be creating a new account, therefore I will be skipping this step.

My remote GitHub repository **"https://github.com/Ehikioya911/Docker-Project"** will be used for storing codes and project.

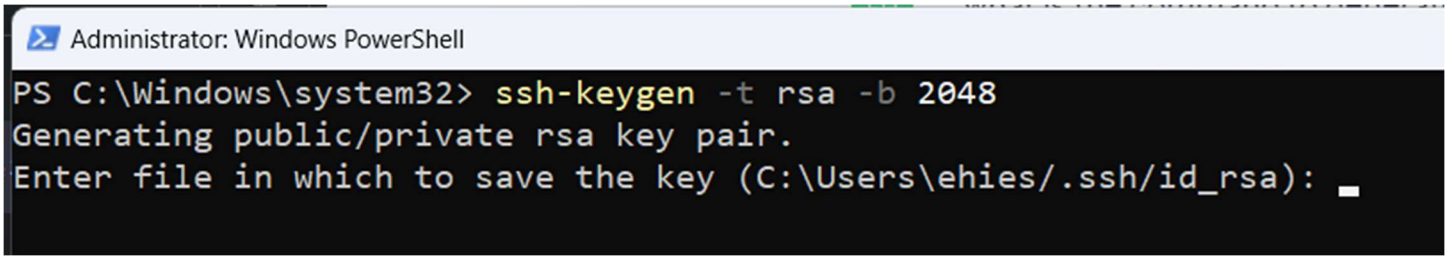
## STEP 2: CREATION OF KEY PAIR

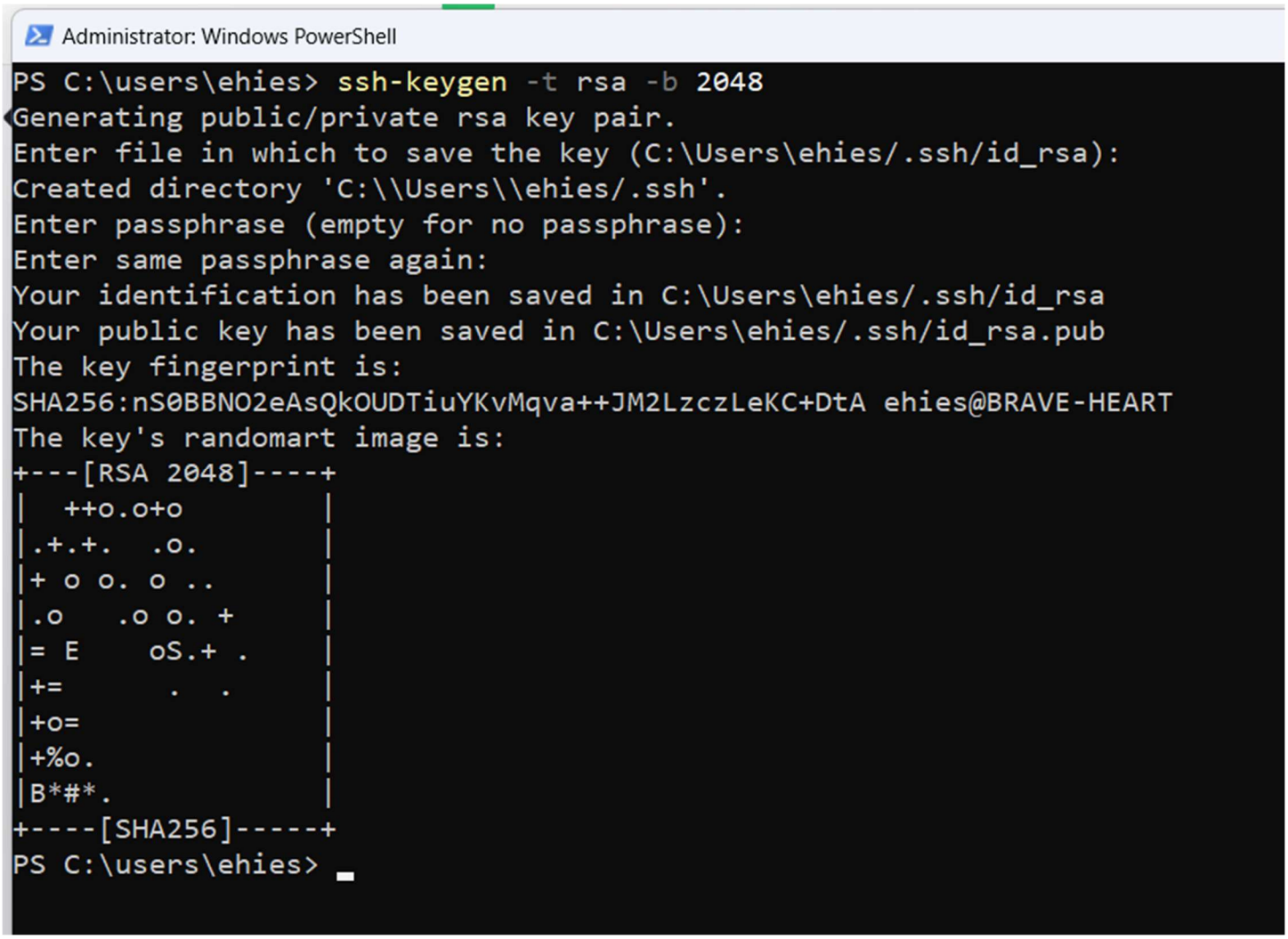
Let's create a key pair that will be used to clone our remote GitHub Repository to our local machine.

I'm currently using a Windows 11 Pro machine for this project, for that reason we will be using PowerShell to generate our public and private SSH key pair.

Let's click on the start menu > type PowerShell in the search box > click on PowerShell app to open the PowerShell terminal.

Run the command ssh-keygen -t rsa -b 2048.




**Key pair was successfully generated.**

## Now let's navigate to the appropriate folder C:\users\ehies\.ssh in our local machine to verify if our key pair is present.

Open Windows PowerShell

**cd** into the **.ssh** directory

PS C:\users\ehies> cd .ssh

**ls** to list files.

PS C:\users\ehies> cd .ssh> ls

Mode LastWriteTime Length Name

-a---- 7/20/2023 10:15 PM 1823 id_rsa

-a---- 7/20/2023 10:15 PM 400 id_rsa.pub

```
PS C:\users\ehies>
PS C:\users\ehies> cd .ssh
PS C:\users\ehies\.ssh> ls


    Directory: C:\users\ehies\.ssh


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         6/28/2023   2:24 PM           1823 id_rsa
-a----         6/28/2023   2:24 PM            400 id_rsa.pub


PS C:\users\ehies\.ssh>
```

## SSH Key pair is present in our local machine.

## STEP 3: UPLOADING OF PUBLIC SSH KEY TO GITHUB

Recall that we created our SSH public key and private key pairs in our previous step, now we must upload the public SSH key from our local machine to GitHub, to enable us to clone our remote GitHub repository to our local Windows 11 machine.

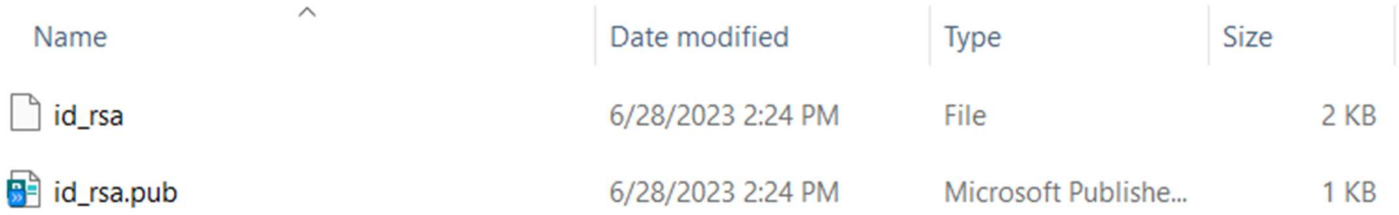
## The prerequisites for this process are:

(a)Install Git on your local machine.

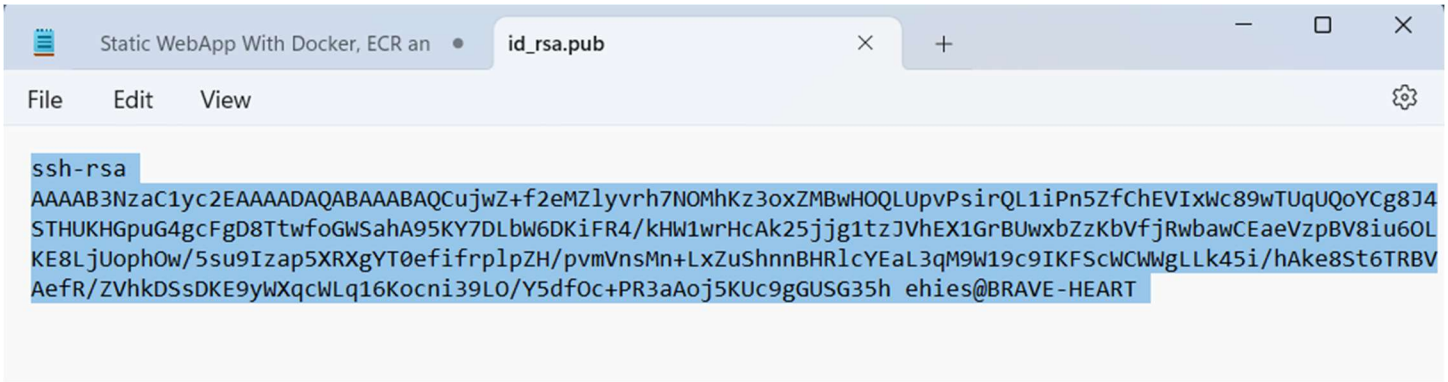
(b)Install Visual Studio Code on your local machine.

(c)Upload the public ssh key pair to GitHub account that was previously created.

## Now, let's navigate to file explorer and navigate to our SSH key pair location C:\users\ehies\.ssh

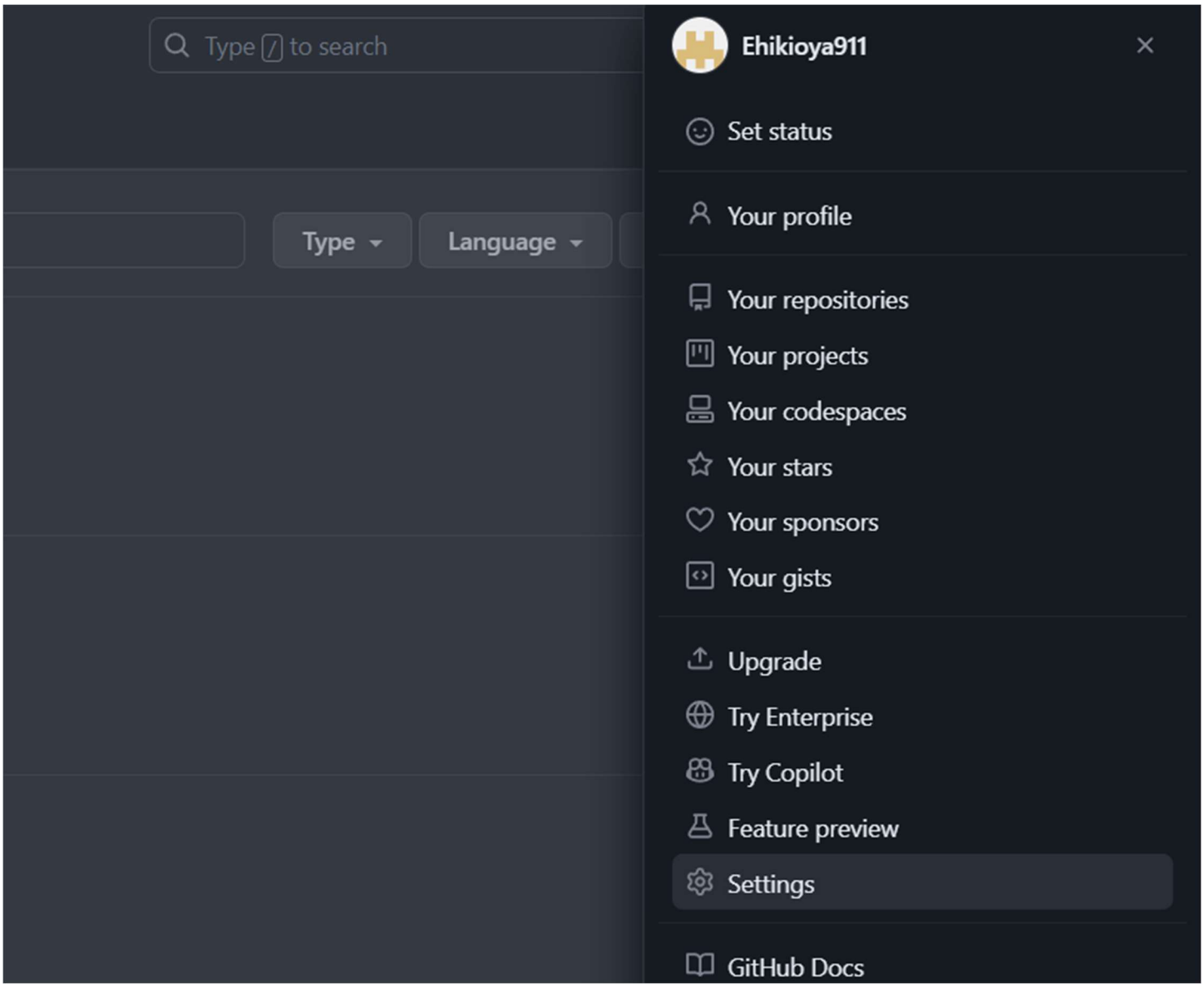


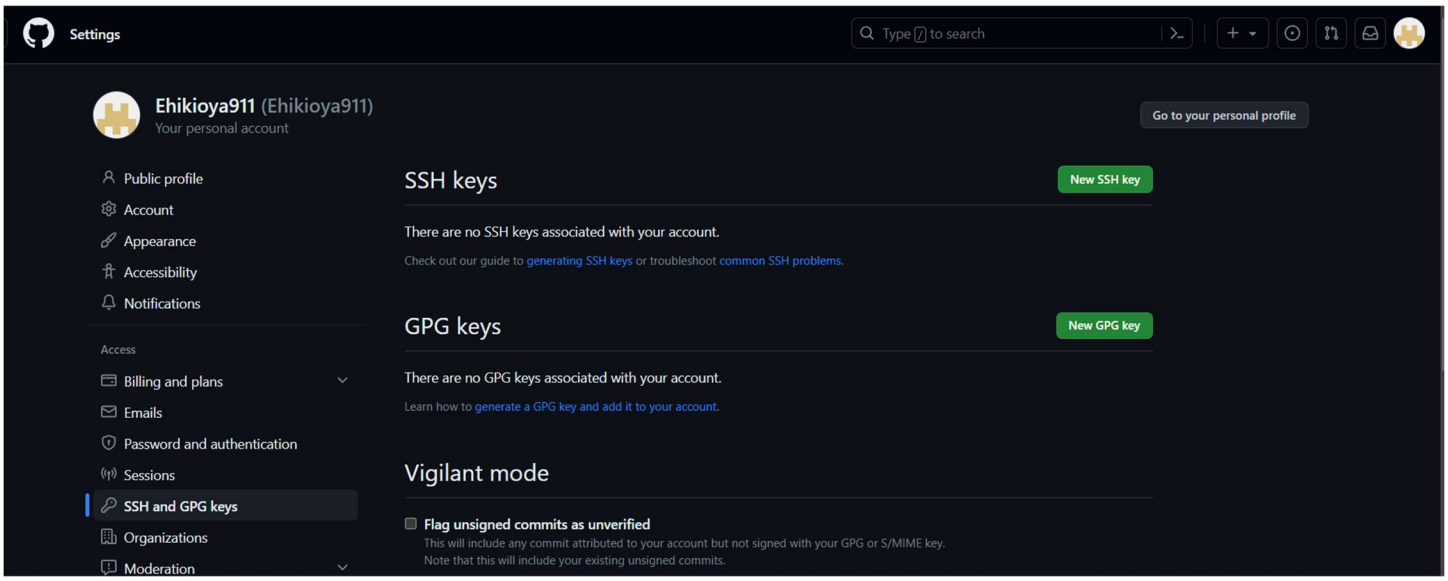

## Now, let's select the public key "id_rsa_pub" > right click on it > open with notepad > copy key pair



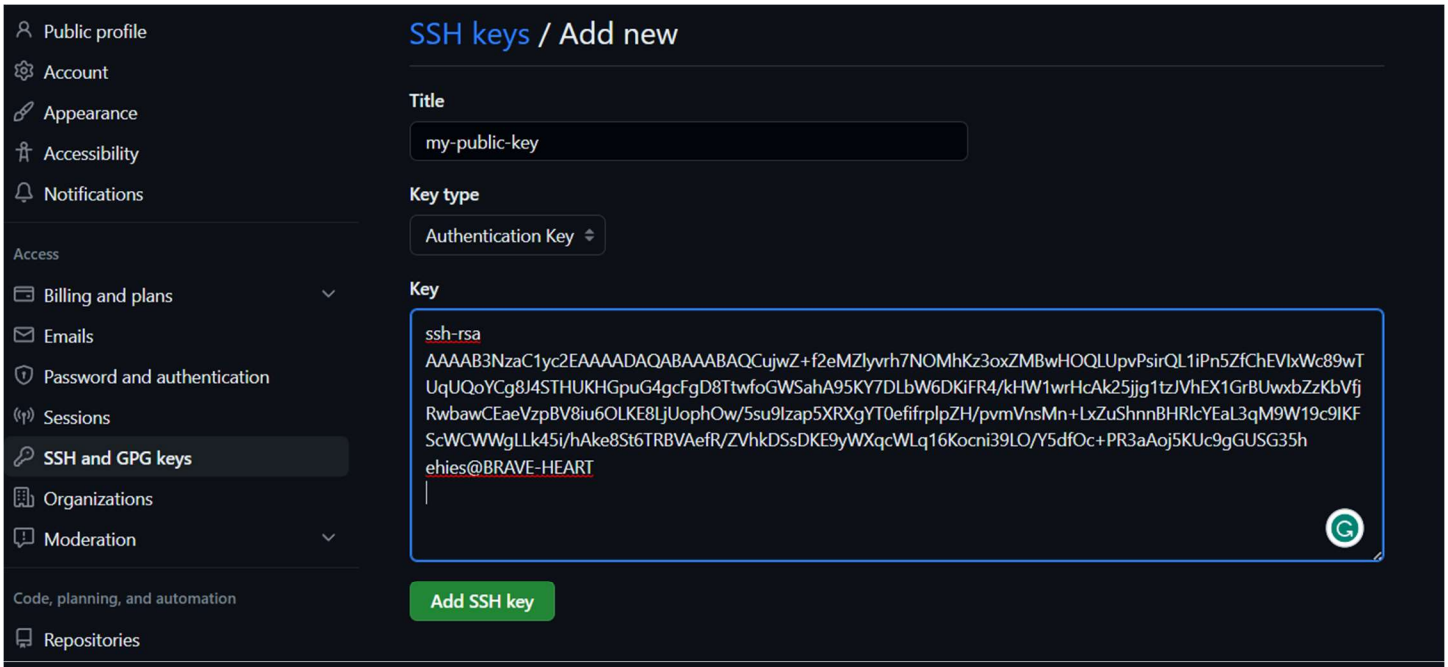

**Login to your GitHub account**

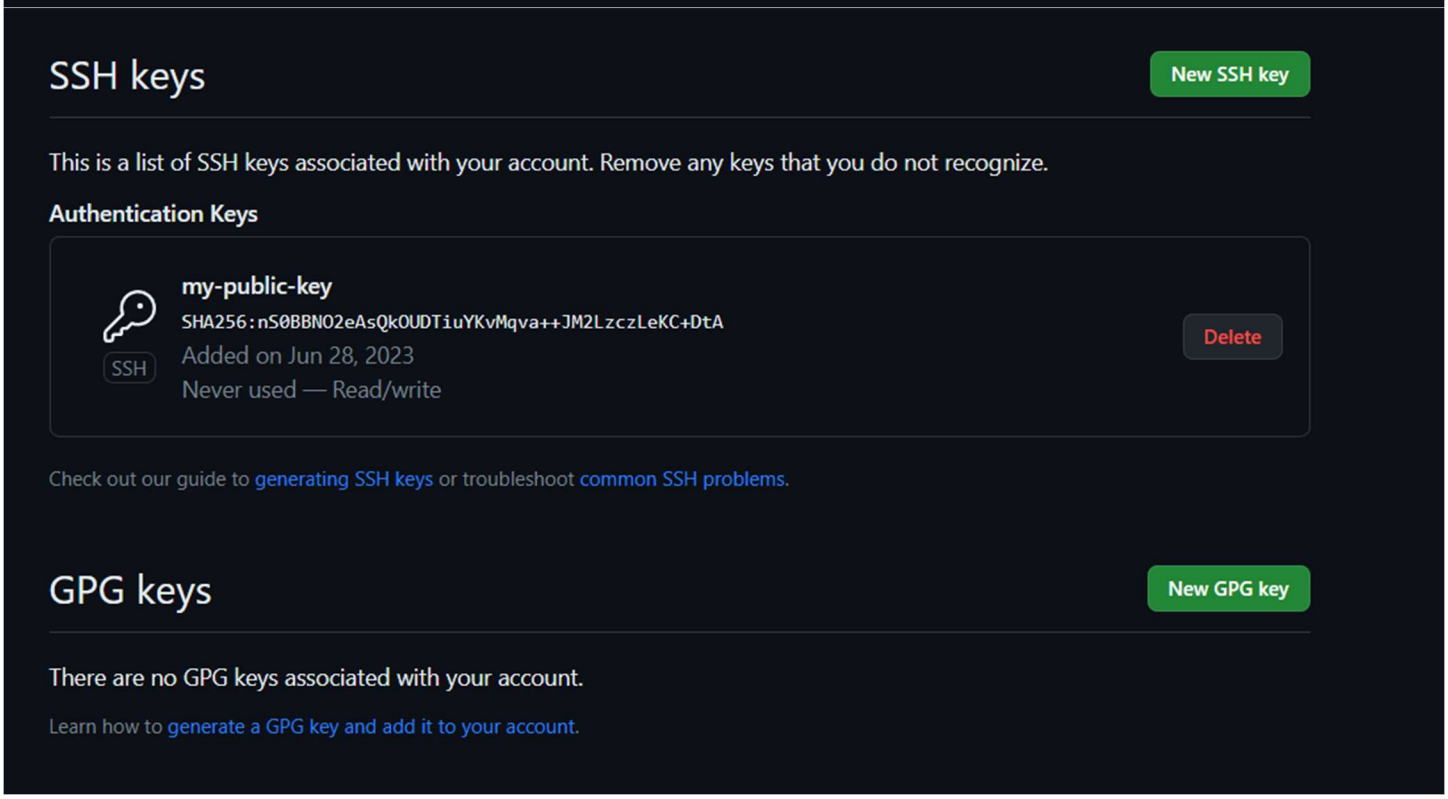
Navigate to the right-hand top corner.
Click on the **circular image logo** to open a drop-down menu.
Select **Settings**
Select **SSH and GPG keys**
Select **New SSH Key** tab


Type / to search
Type
Language
Ehikioya911
Set status
Your profile
Your repositories
Your projects
Your codespaces
Your stars
Your sponsors
Your gists
Upgrade
Try Enterprise
Try Copilot
Feature preview
Settings
GitHub Docs


Settings
Type / to search
Ehikioya911 (Ehikioya911)
Your personal account
Go to your personal profile
Public profile
Account
Appearance
Accessibility
Notifications
Access
Billing and plans
Emails
Password and authentication
Sessions
SSH and GPG keys
Organizations
Moderation
SSH keys
New SSH key
There are no SSH keys associated with your account.
Check out our guide to generating SSH keys or troubleshoot common SSH problems.
GPG keys
New GPG key
There are no GPG keys associated with your account.
Learn how to generate a GPG key and add it to your account.
Vigilant mode
Flag unsigned commits as unverified
This will include any commit attributed to your account but not signed with your GPG or S/MIME key.
Note that this will include your existing unsigned commits.

Let’s give a **Title** key name and let’s call it **“my-public-key”**
Paste the **SSH key**
Click on **Add SSH key** tab to confirm addition of SSH public key to GitHub.




This is how we add our public SSH key to GitHub, and now that our key has been uploaded, we can now clone our remote GitHub repository to our local computer.

## STEP 4:

## INSTALLATION OF GIT

Git is already downloaded and installed on my computer which I have been working with for my previous projects, therefore I will not be re-installing it, and so will be skipping this step.

However, I will give a brief illustration of the installation steps below.

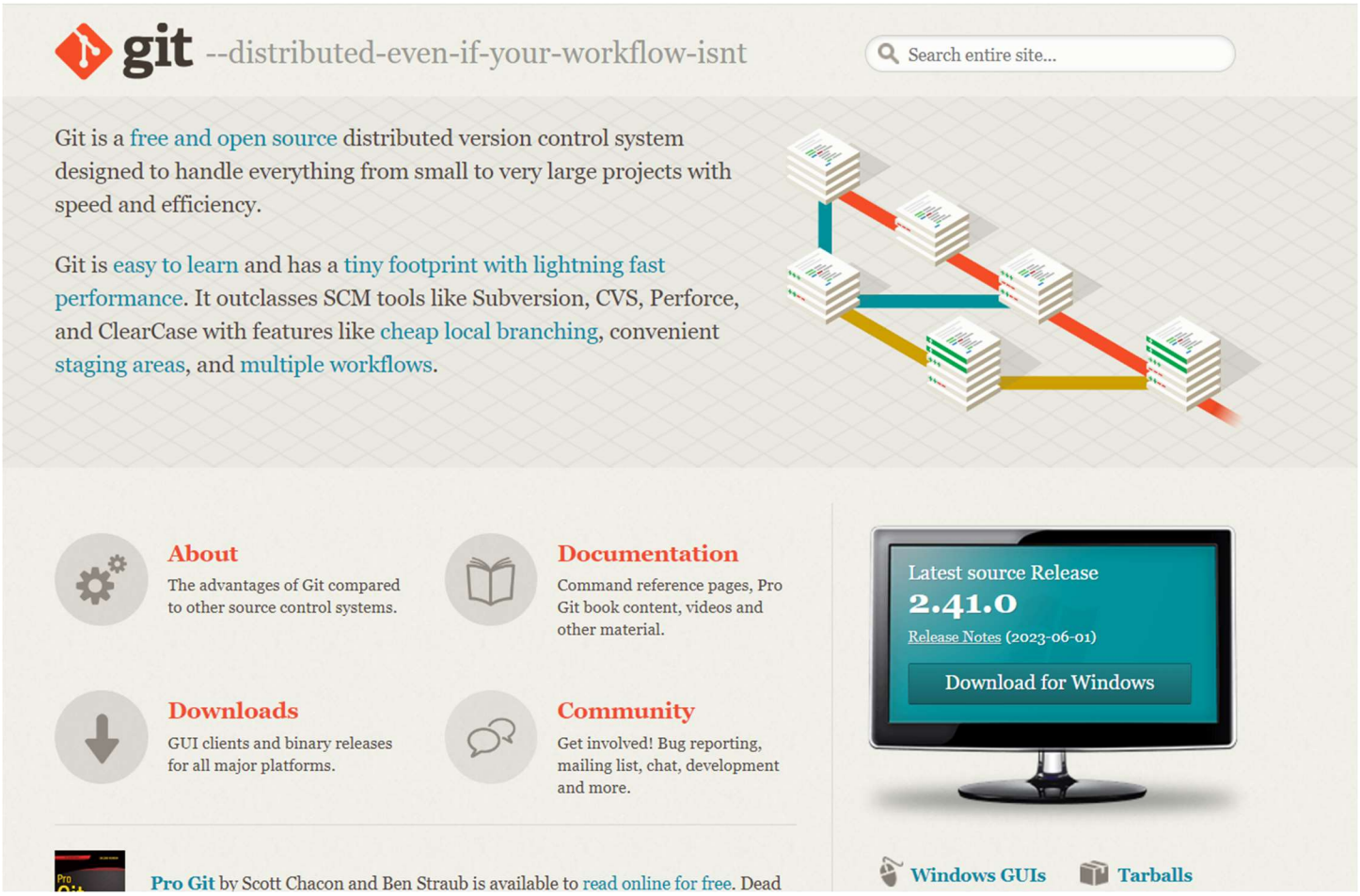
Open www.google.com or any other browser of choice

Open the URL tab and enter www.git-scm.com.

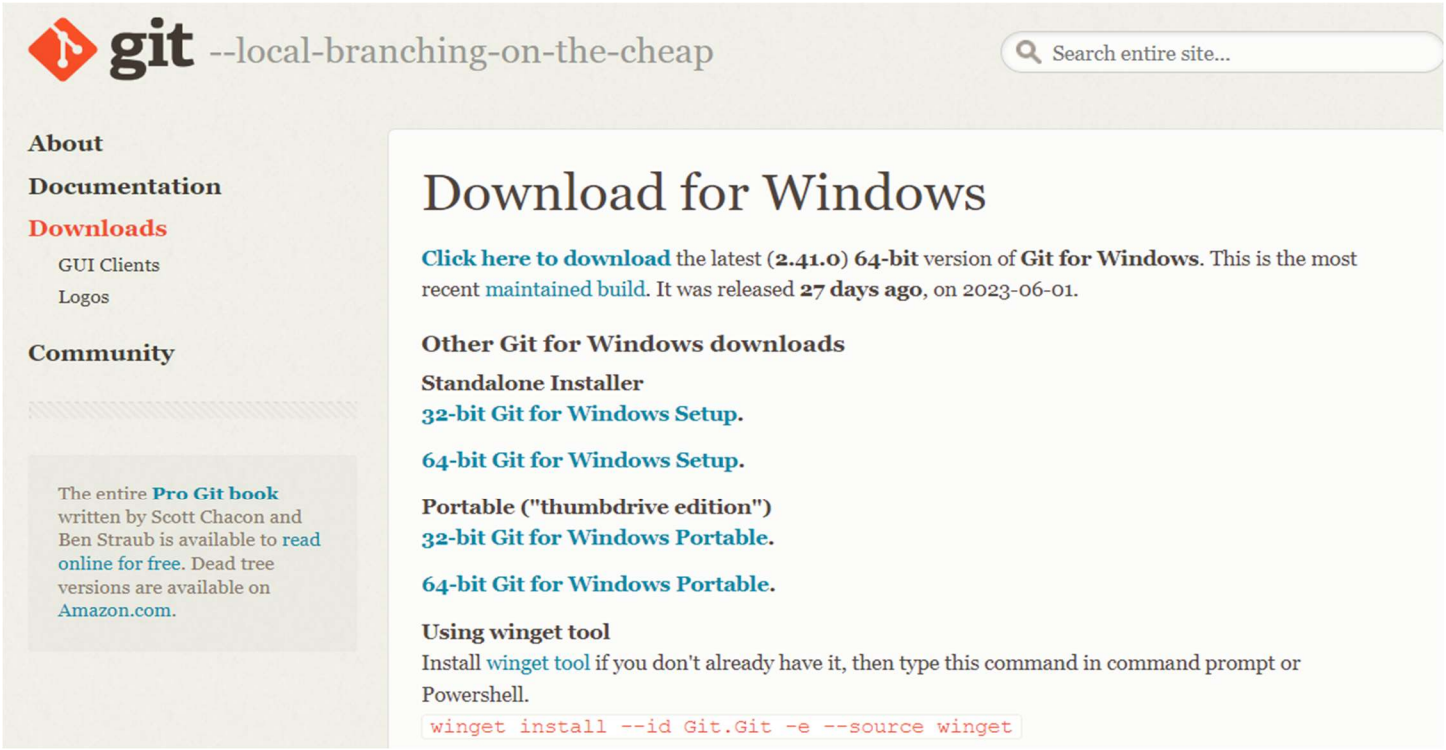
There are options to download git for Windows, Mac, and Linux/Unix

Windows OS is what I am currently running for this project, so we must proceed with the Git download for Windows.

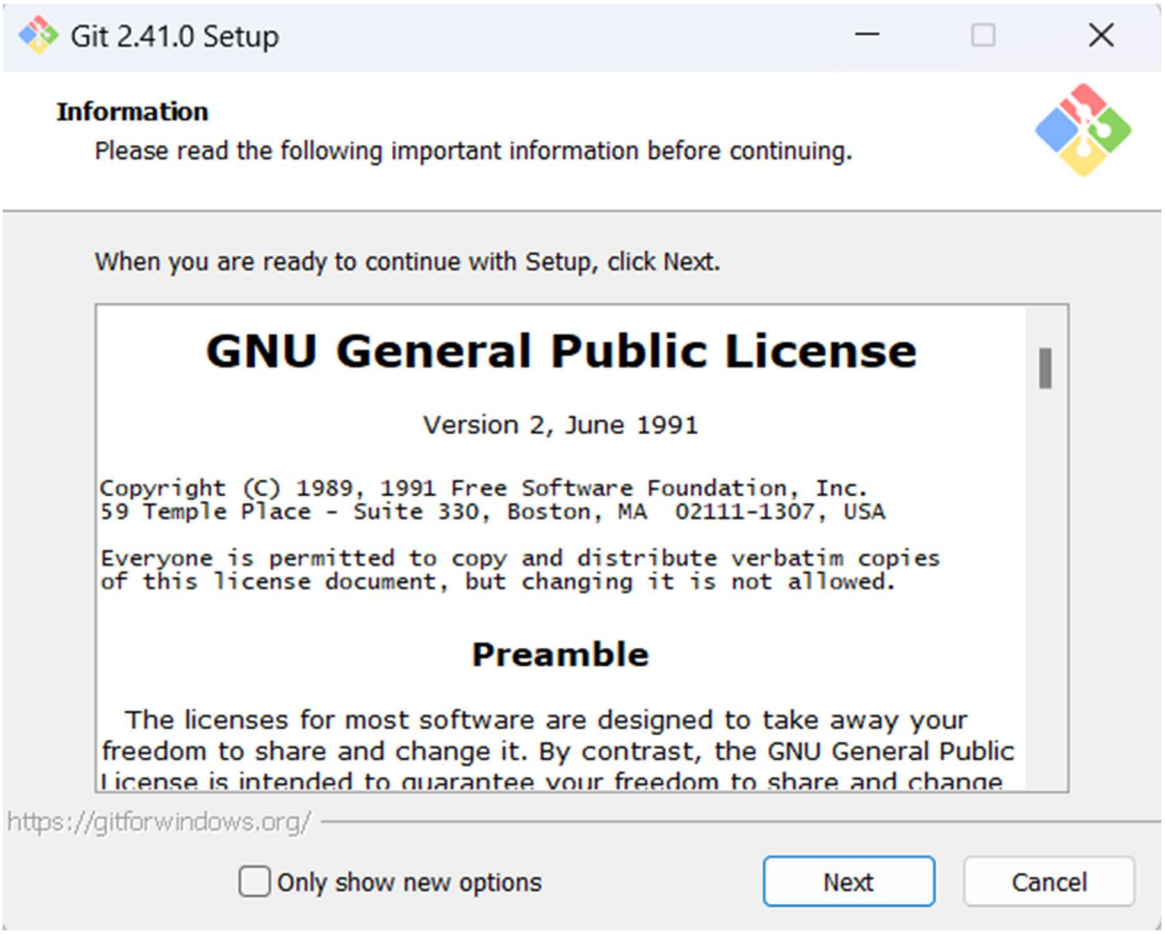
Click on download for Windows version 2.41.0 on the mini green screen Select the 64-bit Git for Windows setup.

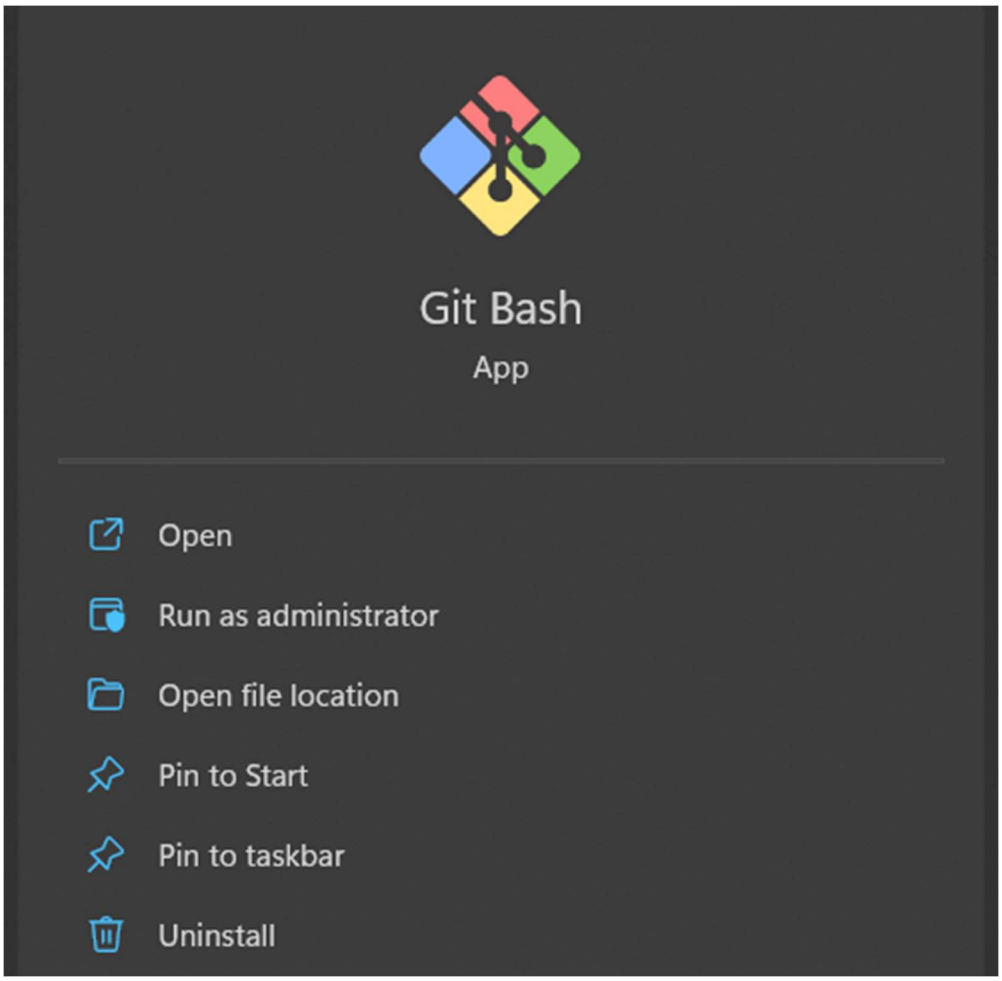


**Navigate to the downloads directory to select the executable download Git file to begin installation.**




Click on NEXT till the end to complete installation**.**



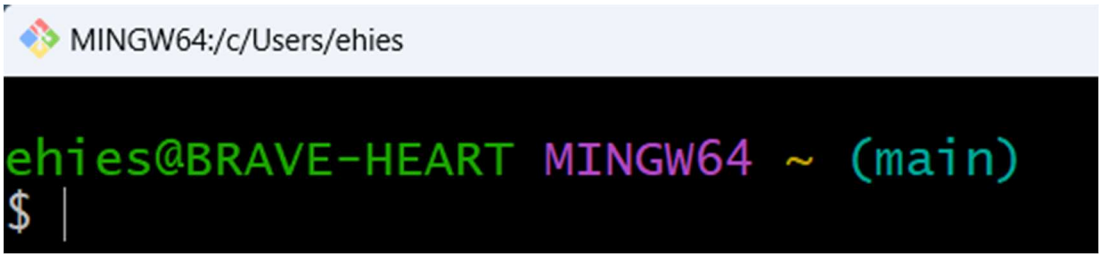



Let's verify if Git was successfully installed on our computer.

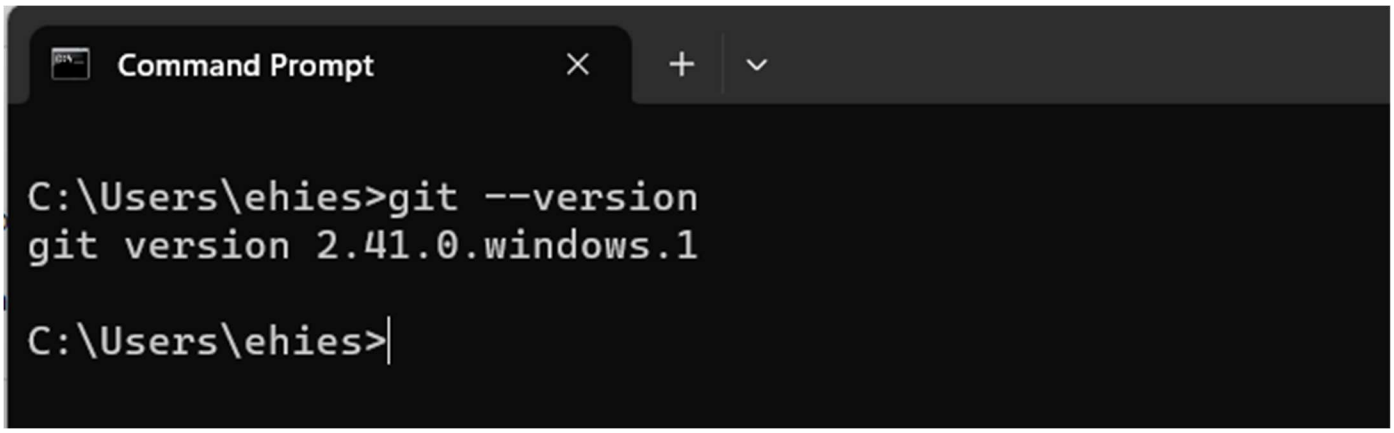
Click on the start menu.

Type git in the search box

Click on Git Bash app to open the Git Bash terminal.




Another way to check if Git was successfully installed is via the command line interface (CLI)
Click on the start menu > type cmd in the search box to open the command line terminal.
Type git --version and press enter.

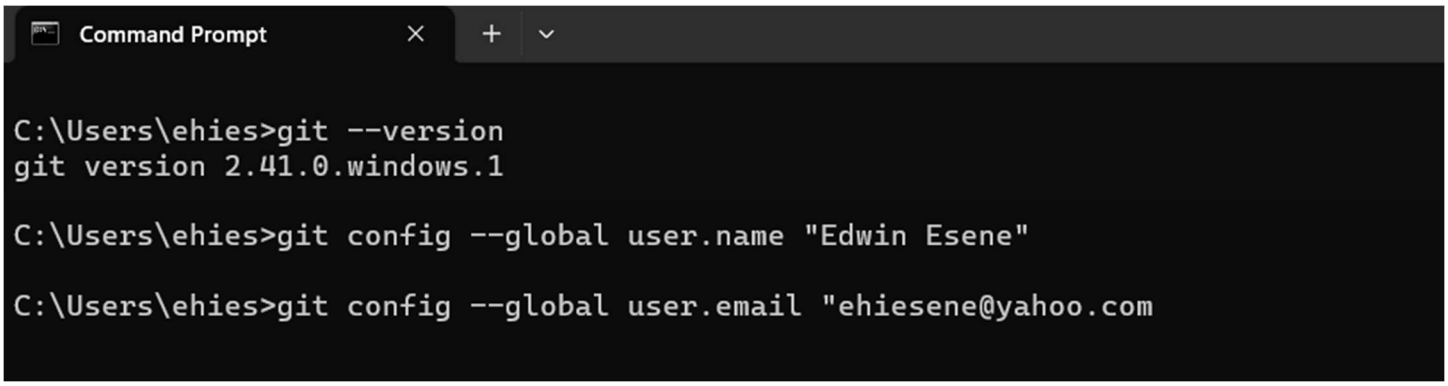



Now that we have verified that Git was successfully installed on our computer, our final step here will be to configure our git username and email.

For that let's copy and run the commands below on our terminal.

**git config --global user.name "Edwin Esene"**

**git config --global user.email "abcdefg@yahoo.com**




Press "Enter" to finalize the configuration of our git username and email.

**NOTE: If you are following along with me, please remember to use your credentials when modifying your name and email as well.**

## STEP 5:

## INSTALLATION OF VISUAL STUDIO CODE (vs-code)

Visual Studio Code is the Text editor we will be using for this project. It was previously downloaded and installed on my computer which I have been working with for my previous projects, therefore I will not be re-installing it, and so will be skipping this step.

However, I will give a brief illustration of the installation steps below.

Open www.google.com or any other browser of choice:

(1) Enter visual studio code and hit enter

(2) Select the link https://code.visualstudio.com/

(3) Click on download tab on the right-hand corner

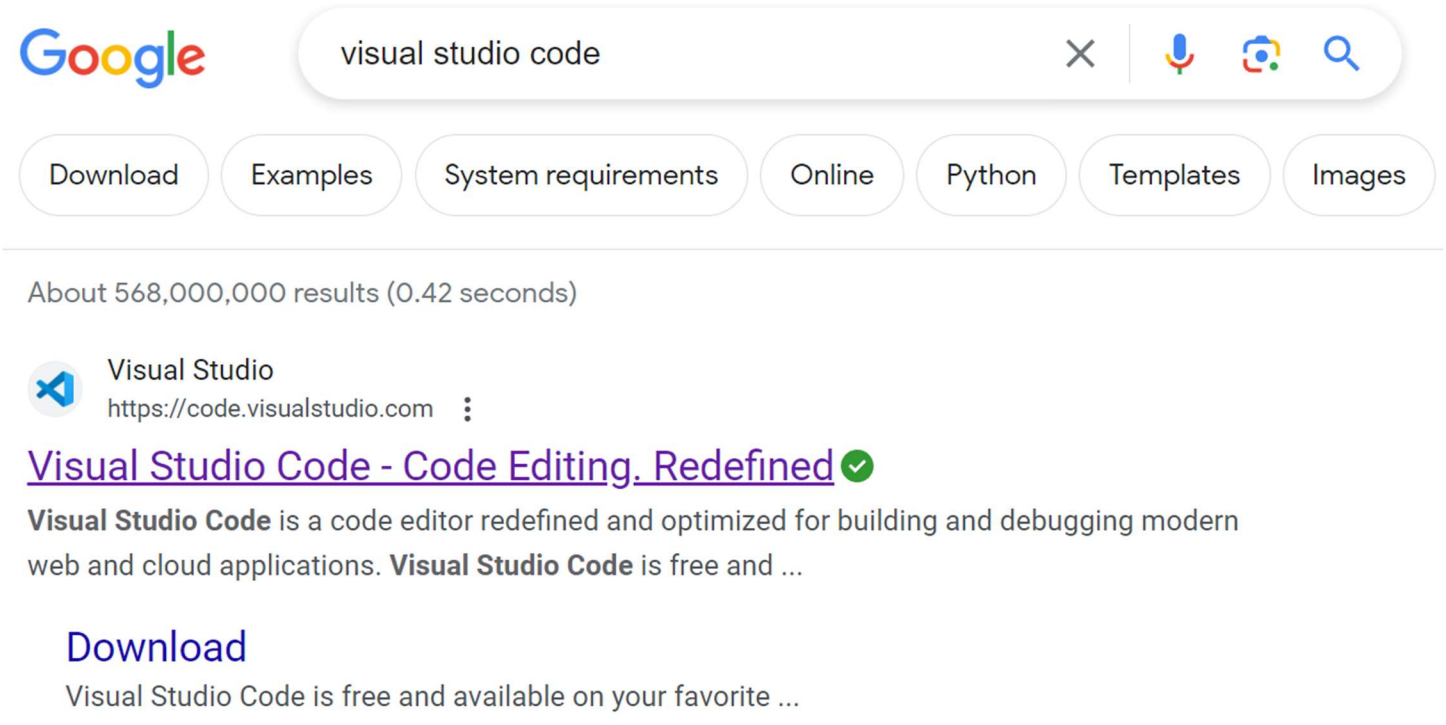
(4) Select your operating system (Windows, Linux, or Mac) and then click to download

(5) Windows OS, is what we are using for this project, so we need to proceed with the Windows download

(6) Navigate to the download folder/directory to select the executable file

(7) Double click on the file to begin the installation

(8) Accept the license agreement and click on "next" until it is finished.



# Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

**Windows**
Windows 8, 10, 11

User Installer x64 x86 Arm64
System Installer x64 x86 Arm64
.zip x64 x86 Arm64

CLI x64 x86 Arm64

**.deb**
Debian, Ubuntu

**.rpm**
Red Hat, Fedora, SUSE

.deb x64 Arm32 Arm64
.rpm x64 Arm32 Arm64
.tar.gz x64 Arm32 Arm64
Snap Snap Store

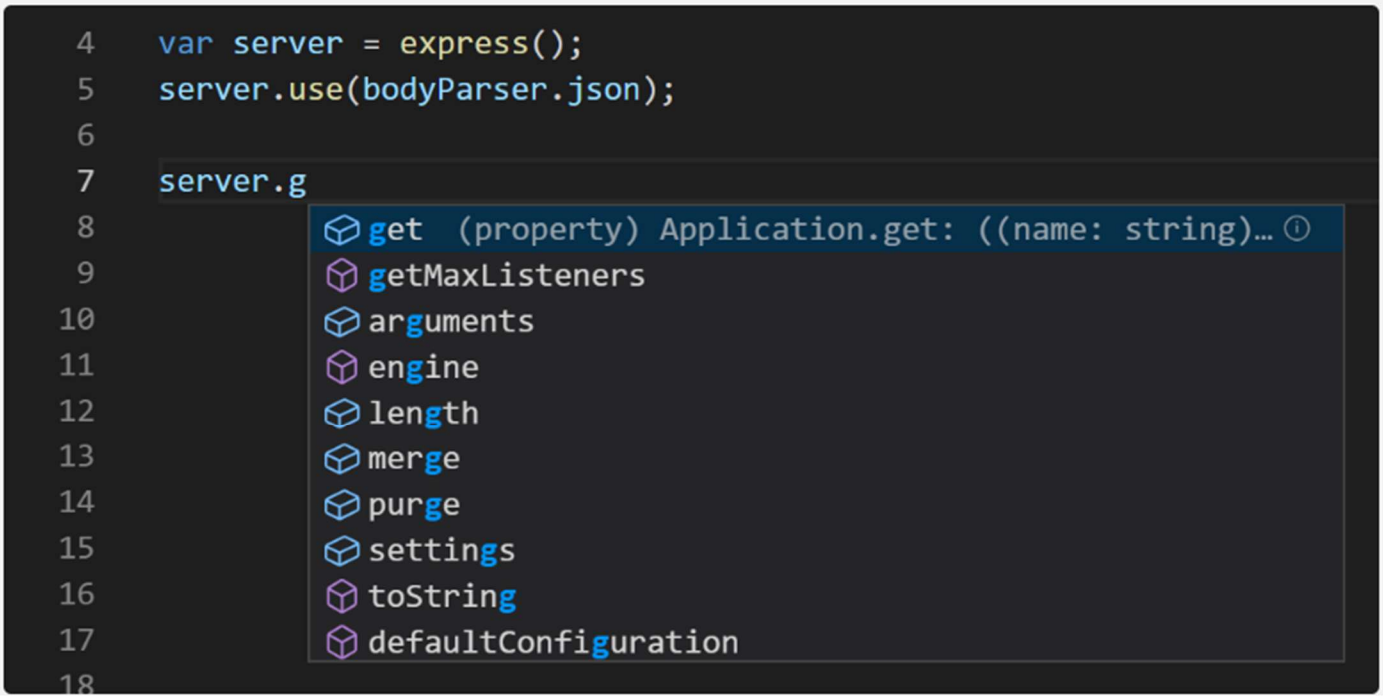
CLI x64 Arm32 Arm64

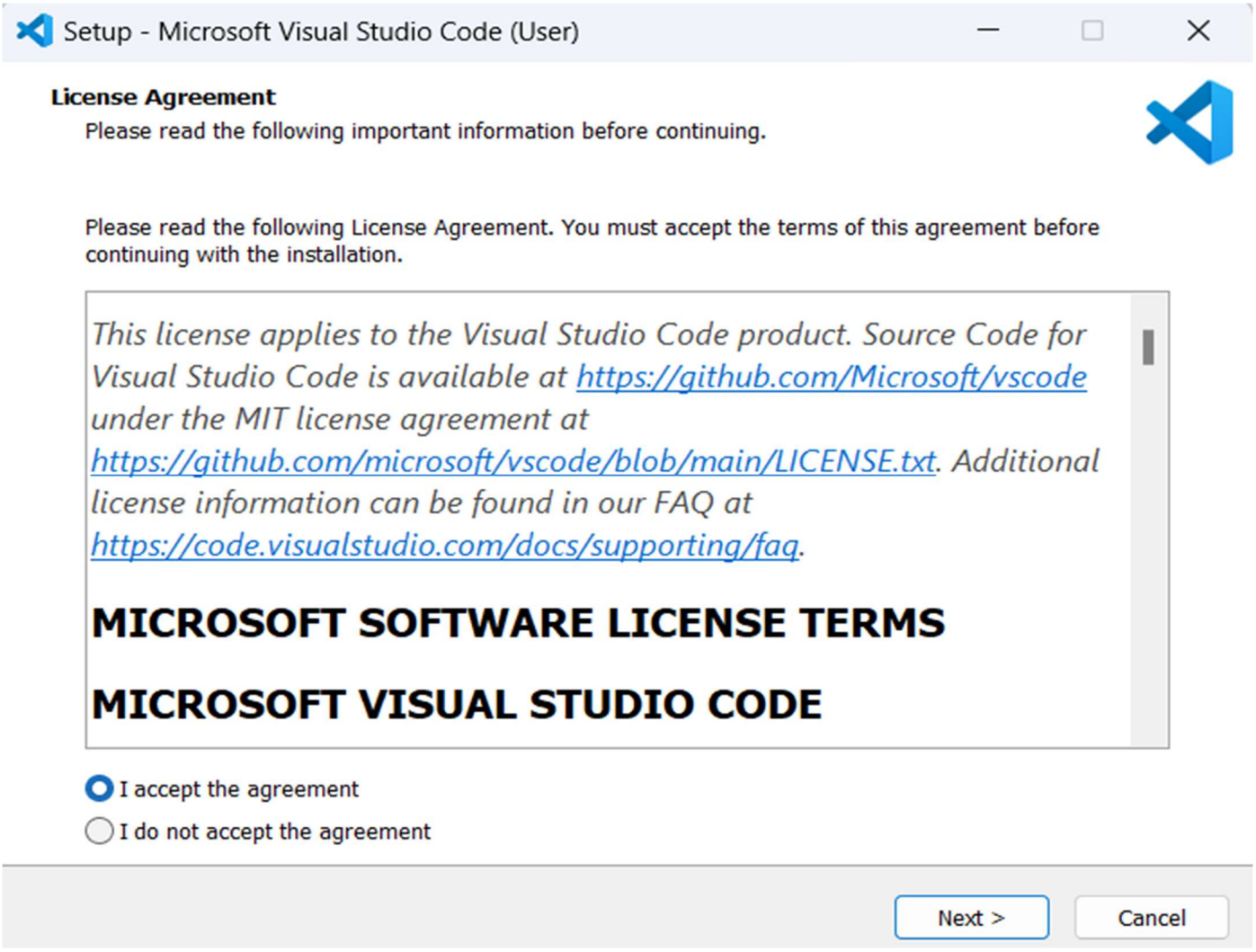
**Mac**
macOS 10.11+

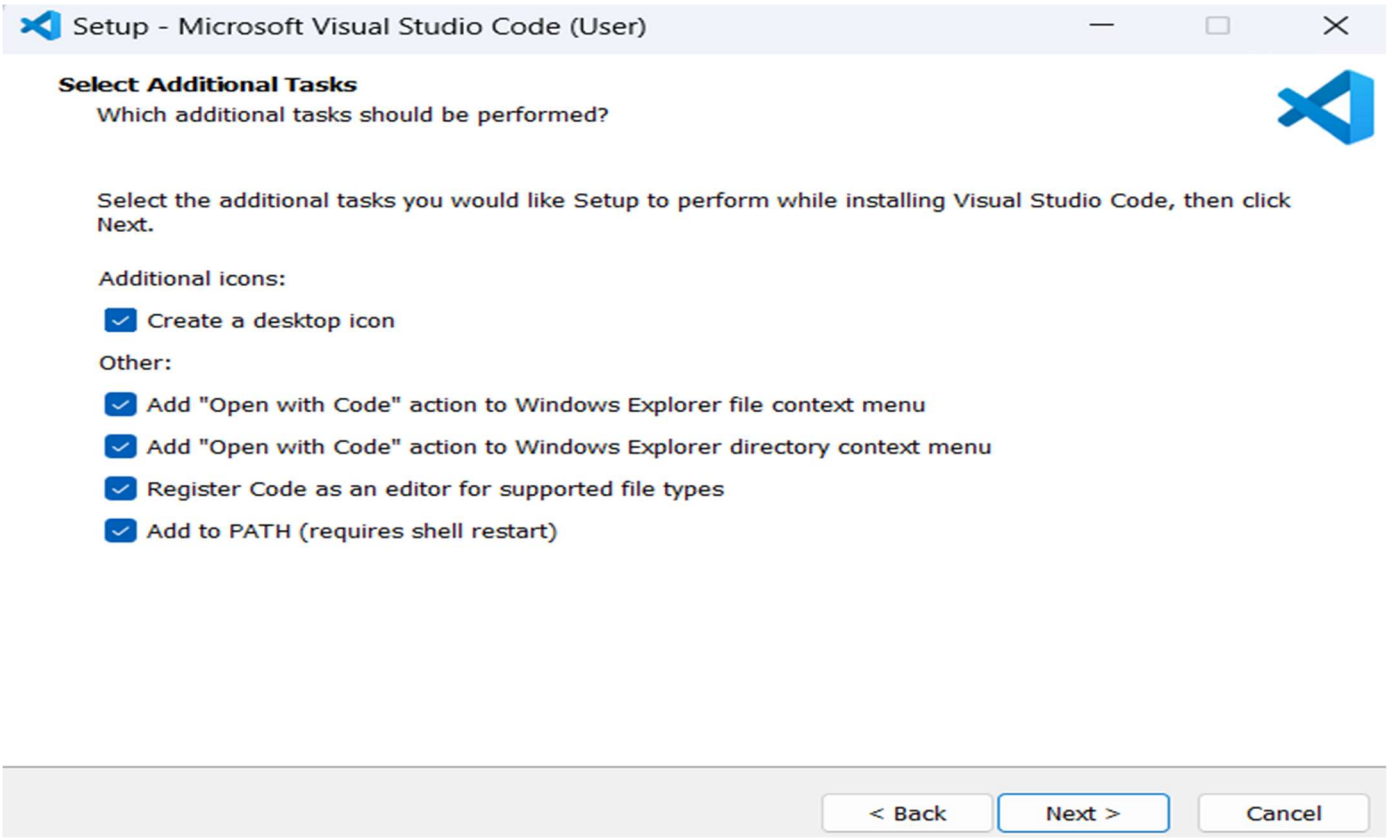
.zip Intel chip Apple silicon Universal

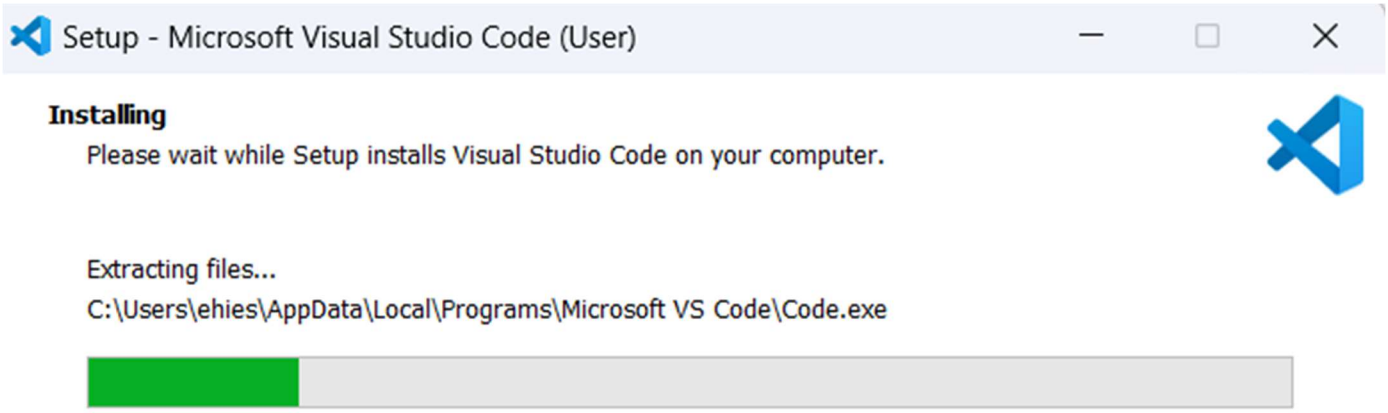
CLI Intel chip Apple silicon

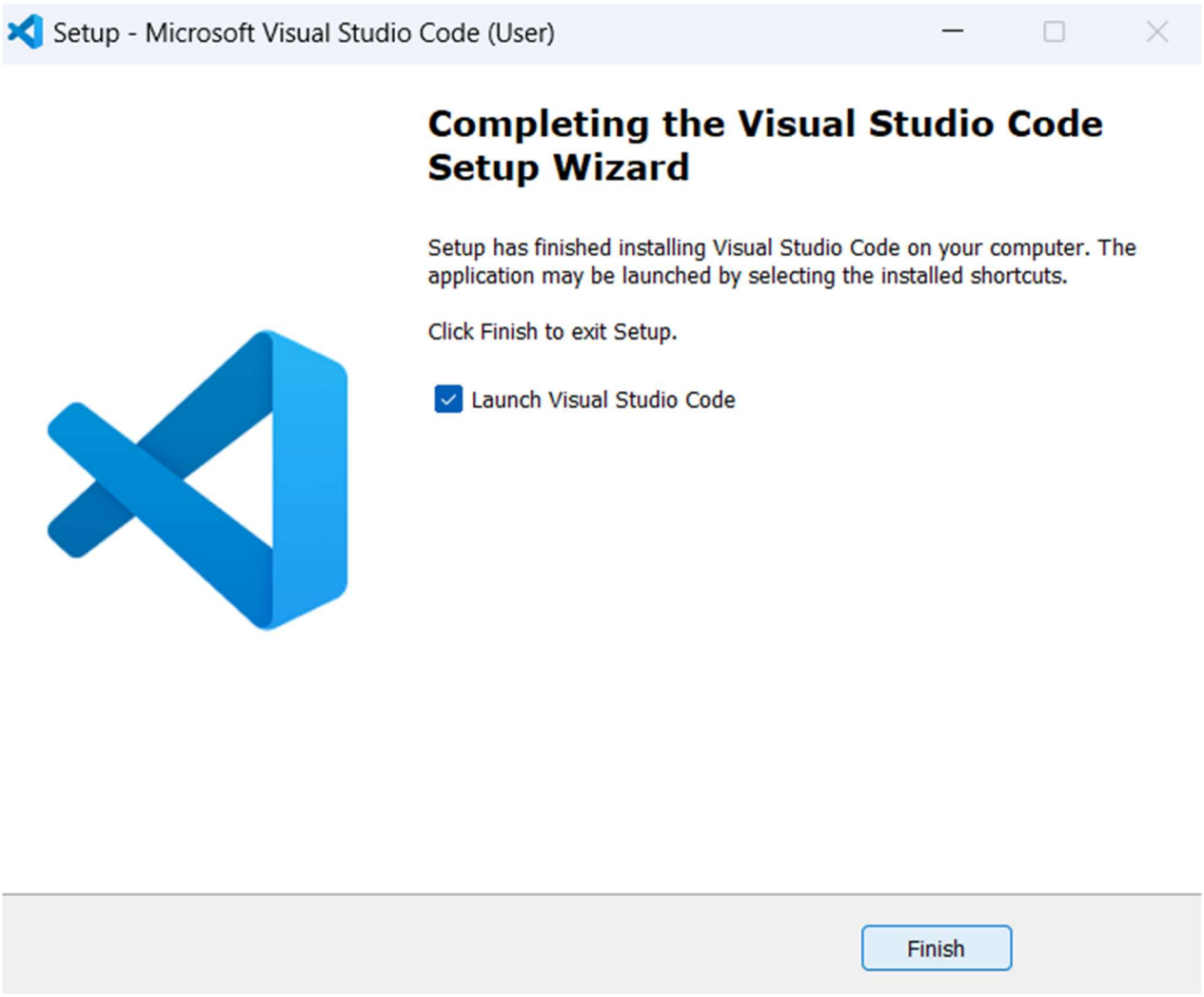
# Visual Studio Code in Action




Setup - Microsoft Visual Studio Code (User)
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
This license applies to the Visual Studio Code product. Source Code for Visual Studio Code is available at https://github.com/Microsoft/vscode under the MIT license agreement at https://github.com/microsoft/vscode/blob/main/LICENSE.txt. Additional license information can be found in our FAQ at https://code.visualstudio.com/docs/supporting/faq.
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT VISUAL STUDIO CODE
I accept the agreement
I do not accept the agreement
Next >
Cancel


Setup - Microsoft Visual Studio Code (User)
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.
Additional icons:
Create a desktop icon
Other:
Add "Open with Code" action to Windows Explorer file context menu
Add "Open with Code" action to Windows Explorer directory context menu
Register Code as an editor for supported file types
Add to PATH (requires shell restart)
< Back
Next >
Cancel




Visual Studio Code has been successfully installed on my machine.

## STEP 6:

## CREATION OF GITHUB REPOSITORY

First, we have to login to our GitHub account to create a GitHub repository.

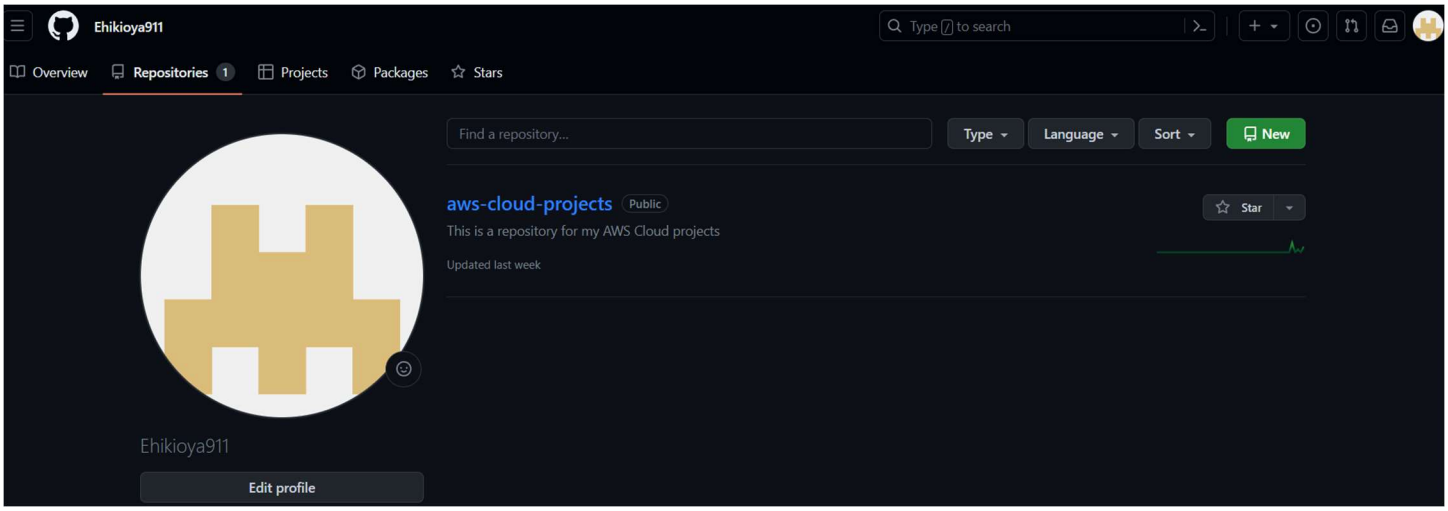
Click on the image on the right-hand top corner to display a drop-down menu.

Select your repositories

The console with previous repositories opens-up

Navigate to the top right of the screen

Click on **New**

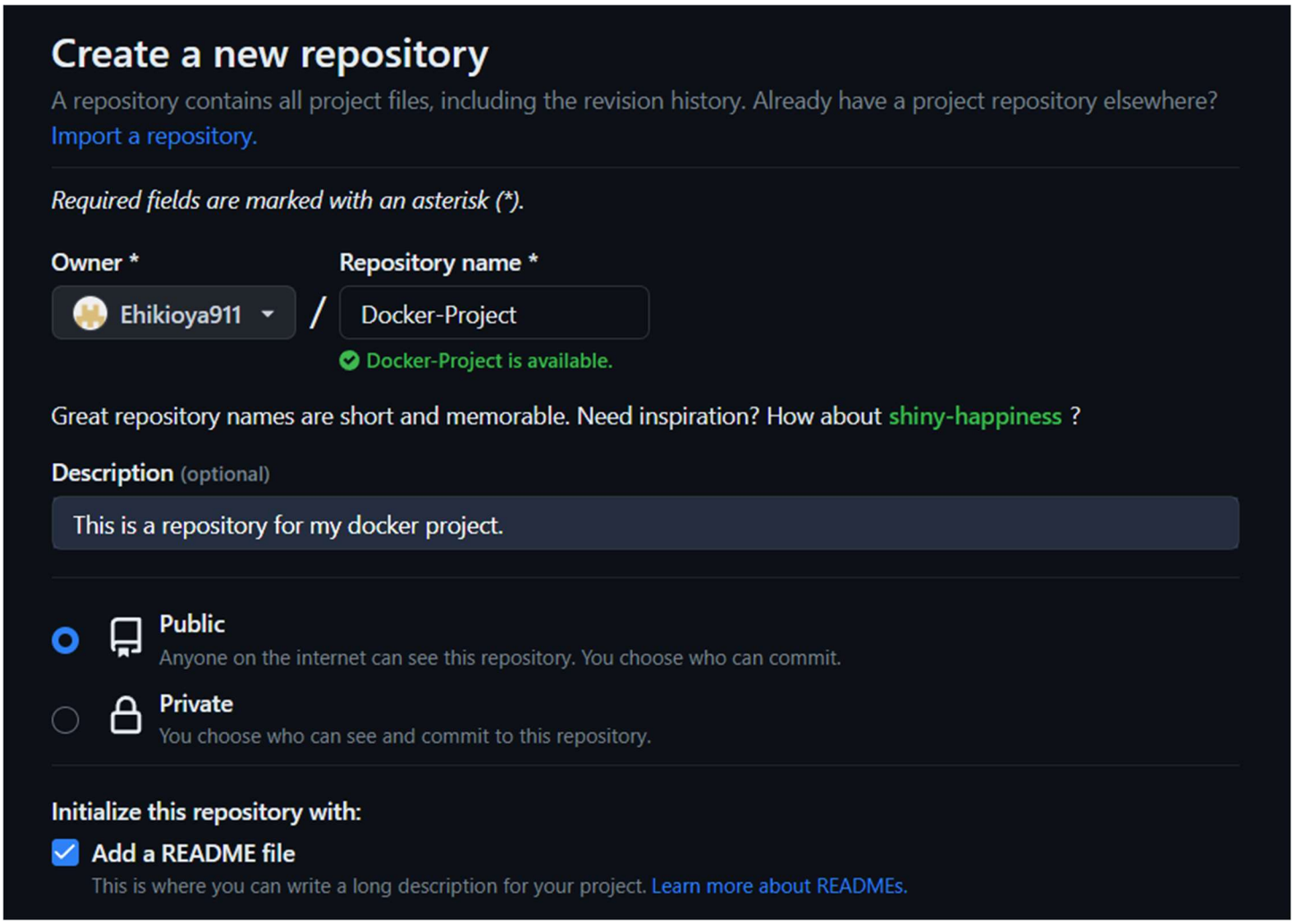



Enter a desired name in the Repository name space and let's call it "**Docker-Project**"

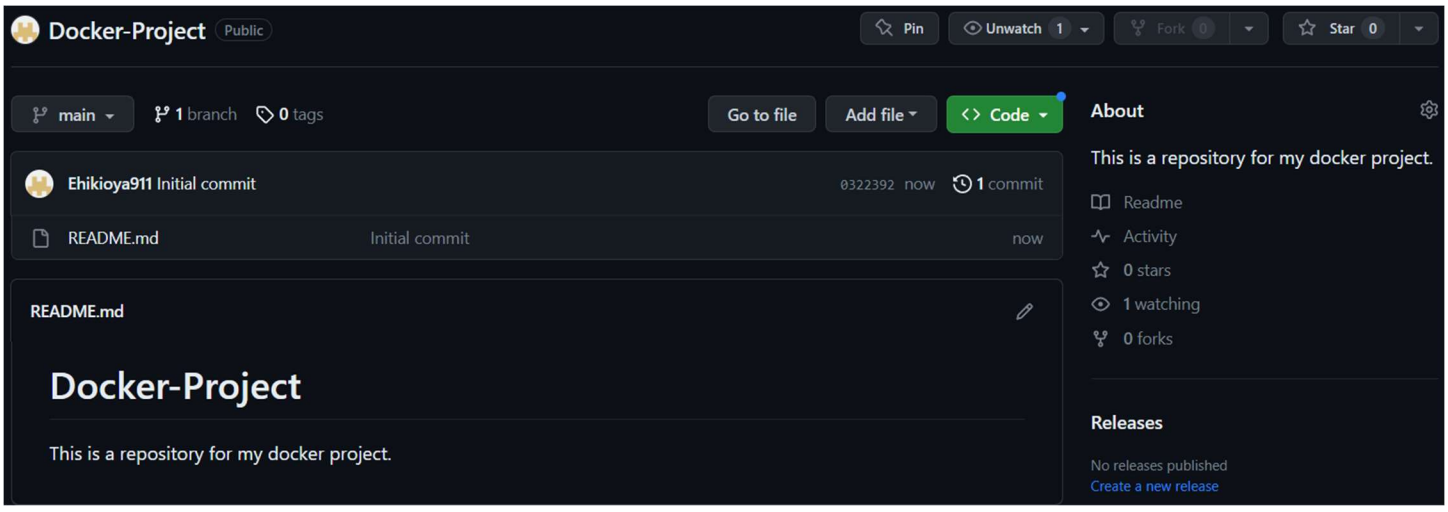
In the description, let's enter "**This is a repository for my docker project**"

We have the option to leave our repository **public** or **private**, but I will select "**Public**"

Select "Add a README file"




Click on create Repository




Docker-Project repository was successfully created.

# STEP 7: CLONING OF GITHUB REPOSITORY TO LOCAL COMPUTER

First, we have to login to our GitHub account to initiate the cloning process.

The repository to be cloned is the Docker-Project repository

Let's open our command prompt to start the process

Now, let's change directory to Desktop, as the location where we want our cloned repository to reside.


We have successfully changed directory to desktop

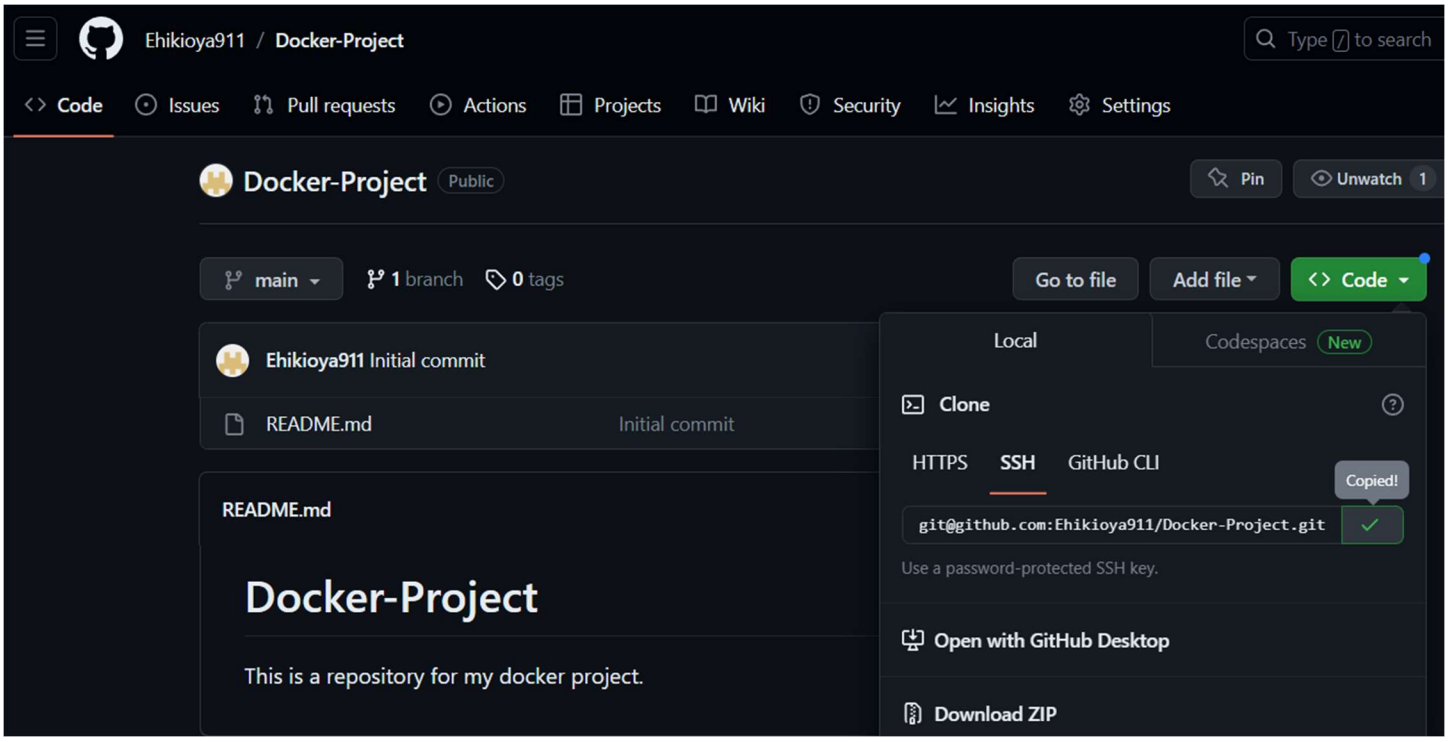
## Next step will be to navigate back to our GitHub repository

Click on the "**Docker-Project**" repository

Click on "**code**"

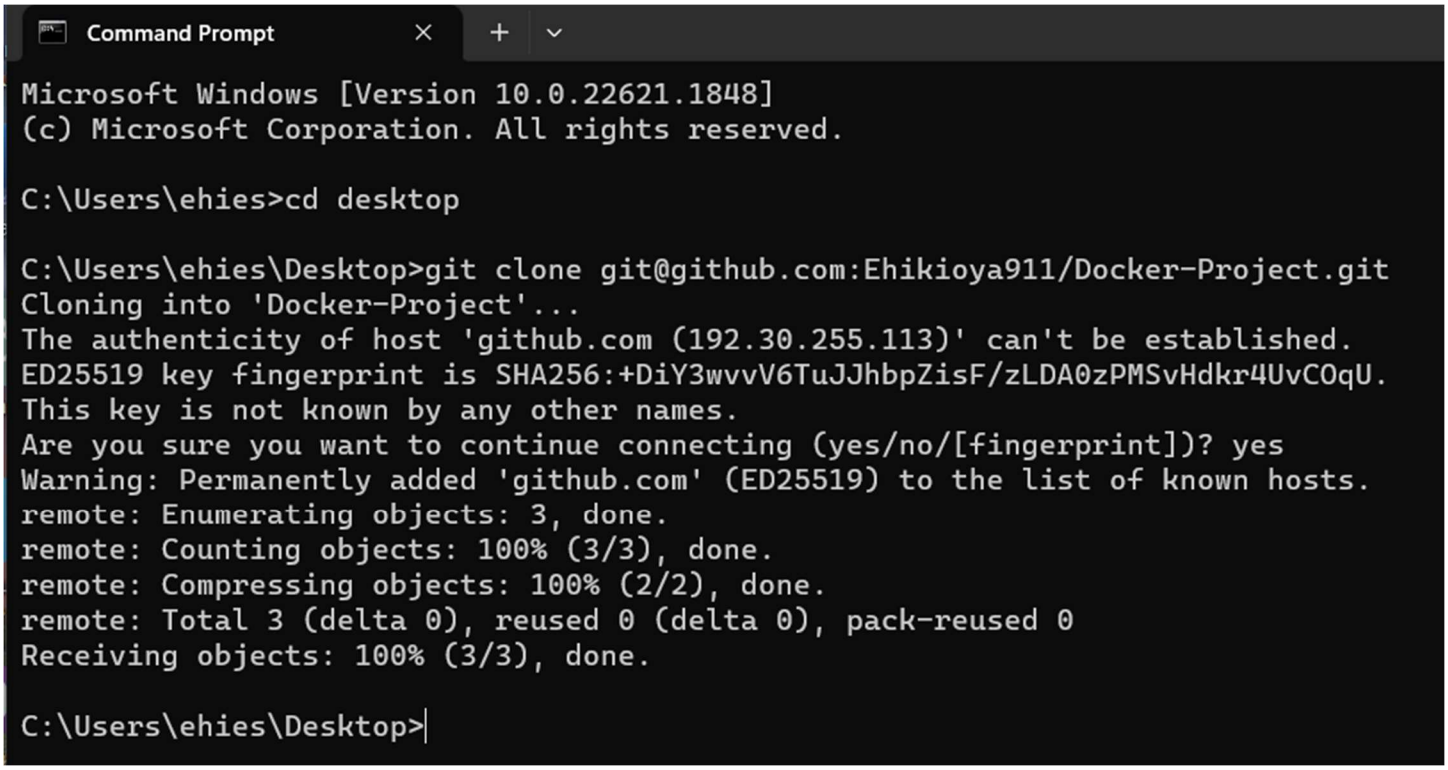
Click on "**SSH**"

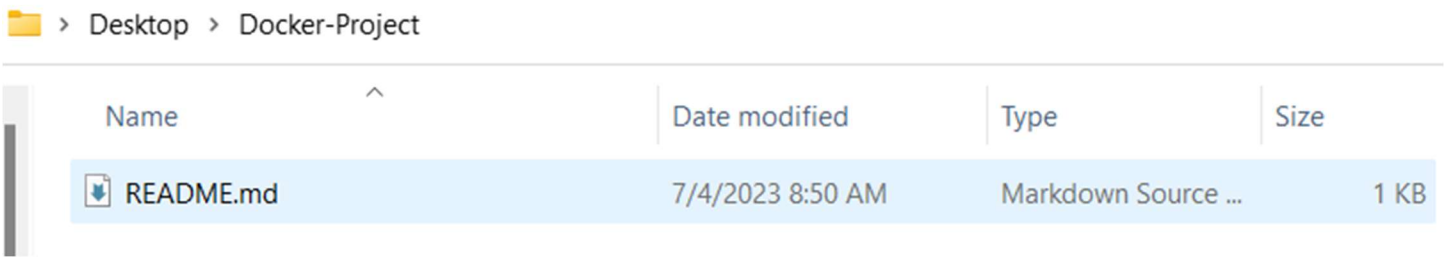
Copy the **repository URL**



Open the command prompt, to run the "**git clone**" command against the copied **URL**

**C:\Users\ehies\Desktop>git clone git@github.com:Ehikioya911/Docker-Project.git**



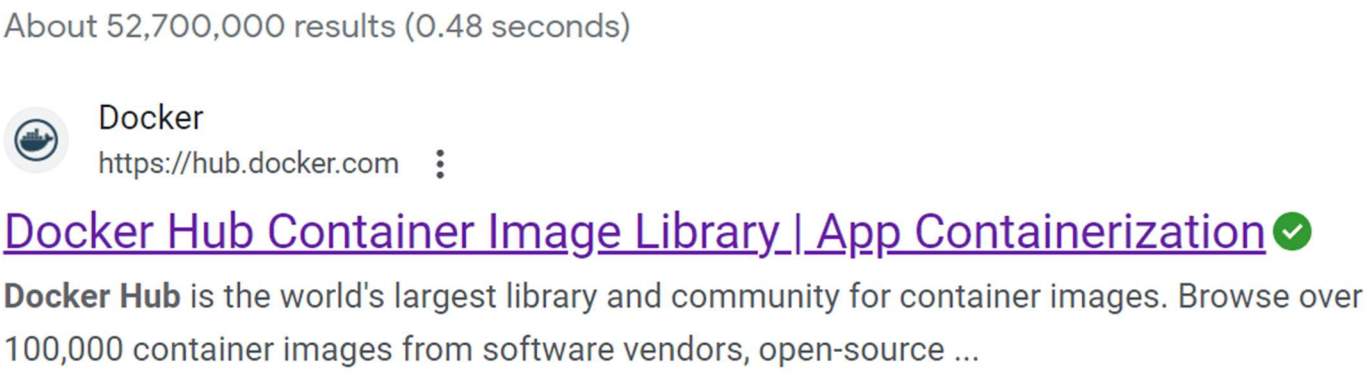




Docker-Project repository has been successfully cloned from GitHub to my desktop as shown from the above screenshot.

## STEP 8: DOCKER HUB ACCOUNT CREATION AND SETUP

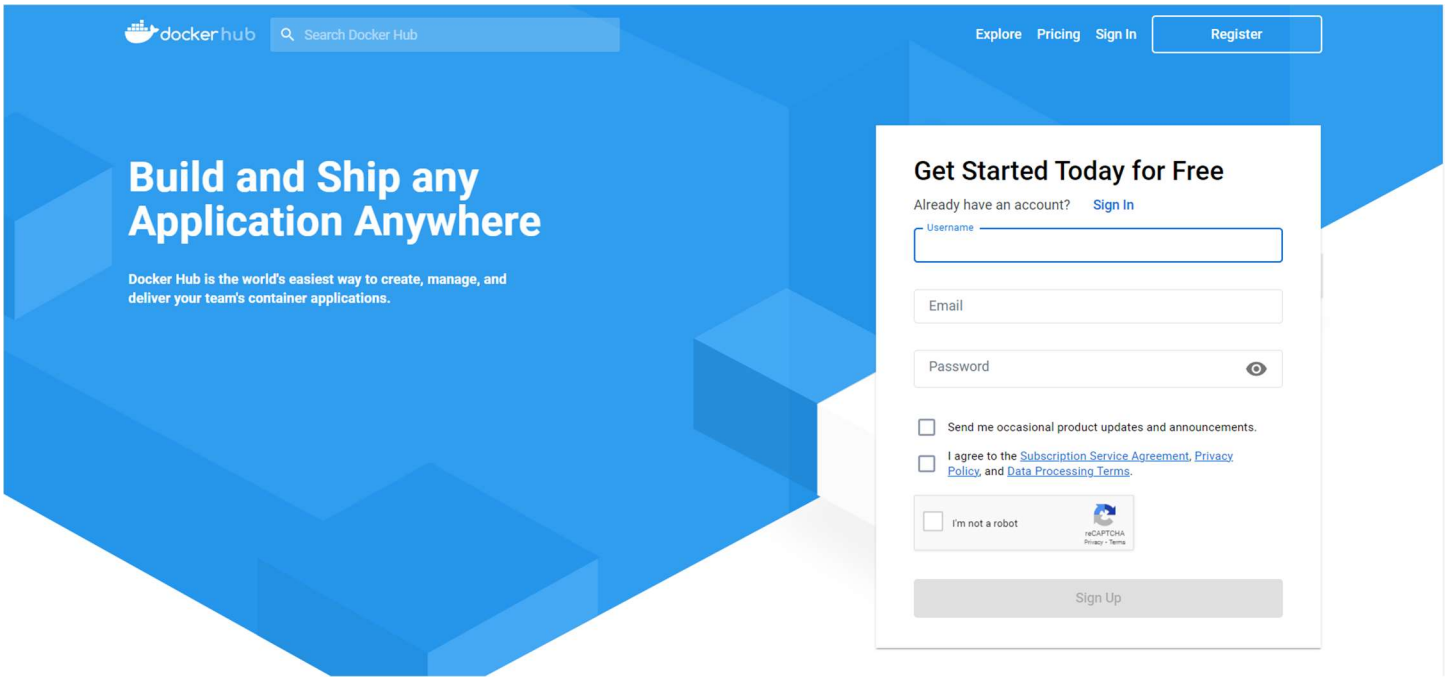
To set up a docker hub account are few simple steps

First, let's open google.com to search for docker hub.




Click on the first link that shows

**https://hub.docker.com**




Now let's navigate to the top right-hand corner of the docker hub home page to click on **Register**

Here, I need to enter a **username** in the field

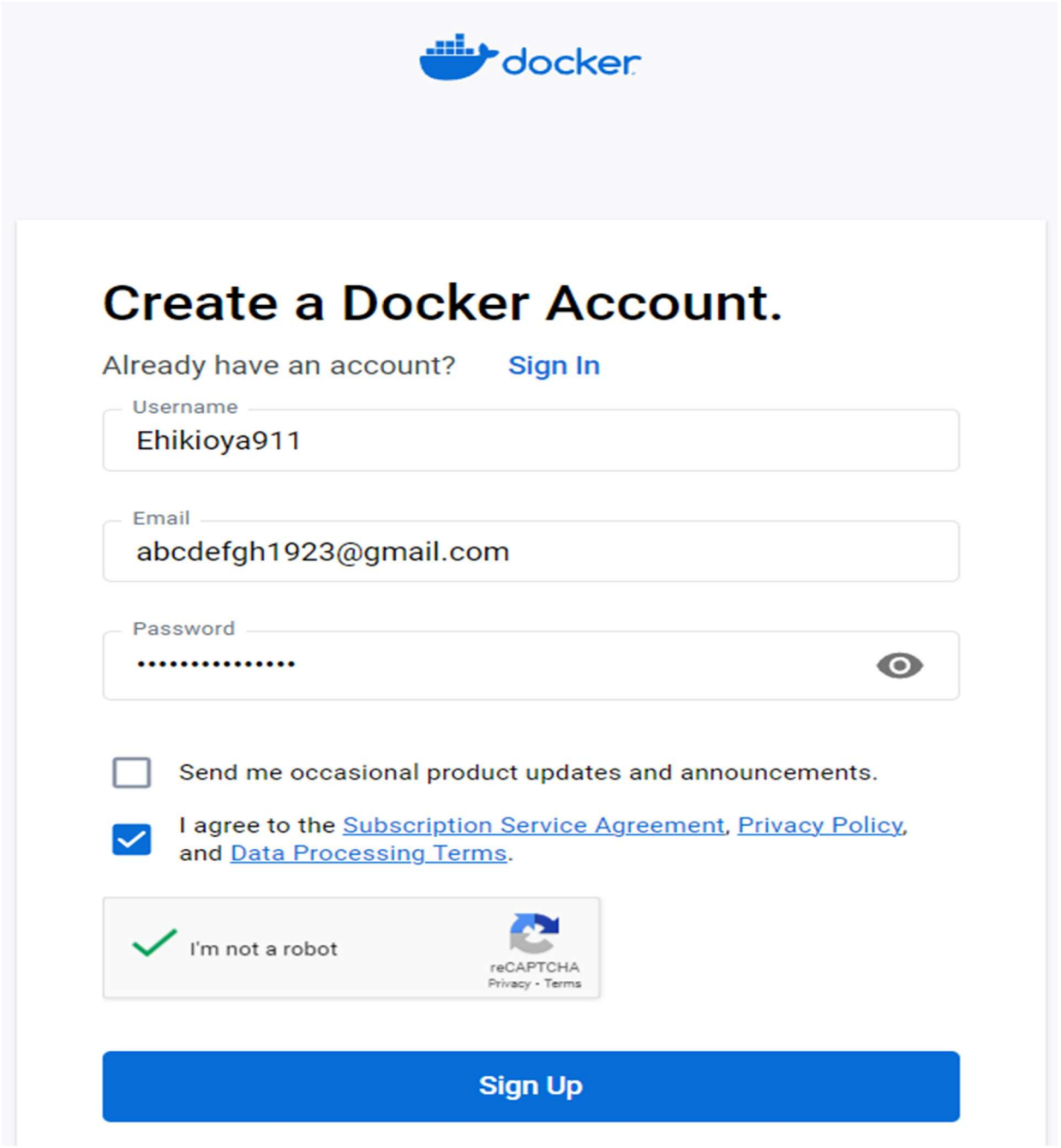
The next step will be to enter my **email** in the email field

Will enter a **password** in the field

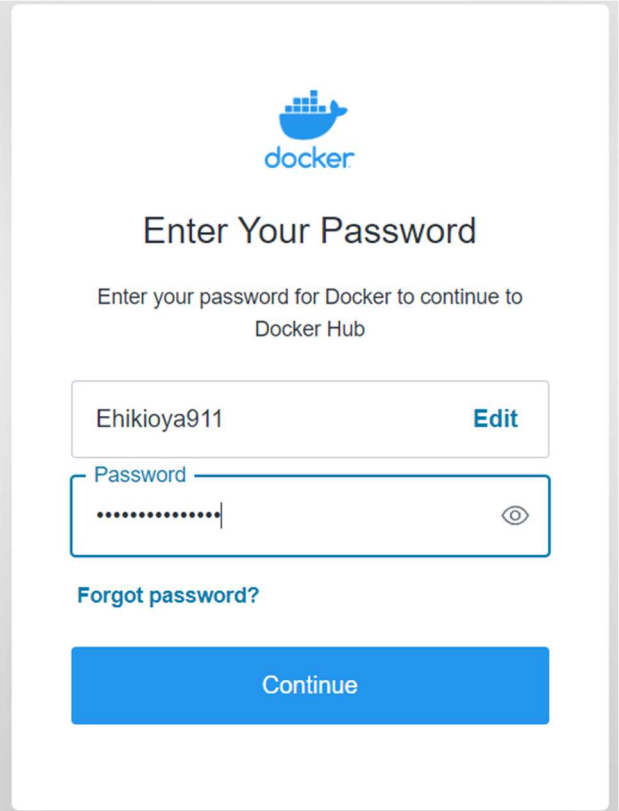
Will select and check **Agree** to the subscription service agreement terms and policy

Will select "**I'm not a robot**" in the CAPTCHA

Next step will be to click on **Sign up**




Click on **Continue** > Enter **username** and **password** > Click **Continue**




**We have successfully created and logged in to our docker hub account, we have the option for an upgrade**

But for this project, it is best to continue with the **personal** plan, which is free.

Now the final step here will be to verify our email account

Sign back into your email account

Locate and open the email from Docker, then click on **"Verify email address"**

## STEP 9: VERIFY AND ENABLE VIRTUALIZATION ON YOUR COMPUTER

Open the command and run "**systeminfo.exe**"

If virtualization is installed, then we are good to continue

But if virtualization is not enabled, below is a link on how to enable virtualization on a Windows Machine.

**https://www.simplilearn.com/enable-virtualization-windows-10-article**

## STEP 10:

## DOCKER DESKTOP INSTALLATION IN MY COMPUTER

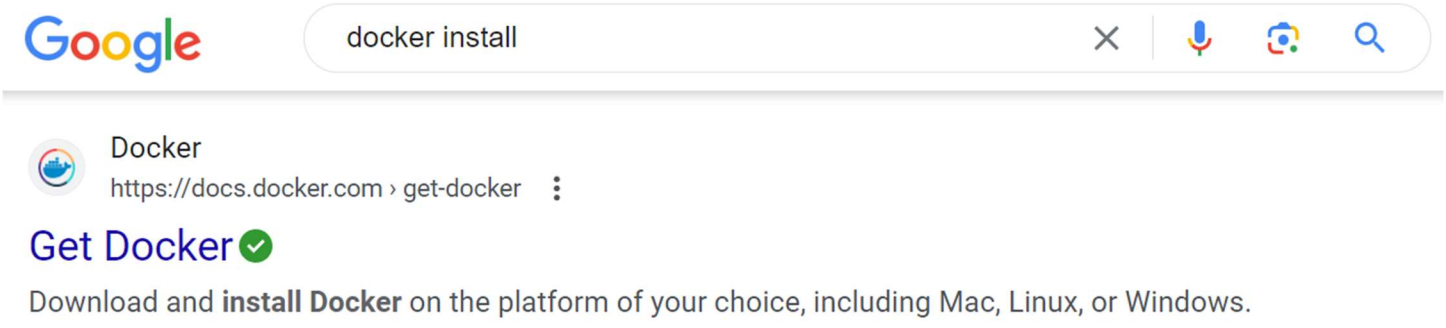
To install docker desktop is just a few simple steps

However, below is a link for a quick step by step installation guide

https://www.youtube.com/watch?v=WDEdRmTCSs8&t=157s

Let's open **google.com,** enter **docker install**

Select the link: **https://docs.docker.com > get docker**




Here we are given the options to **install docker** for **Mac, Linux,** or **Windows**

I will select **Docker Desktop** for **Windows** since I'm running a **Windows 11** operating system.

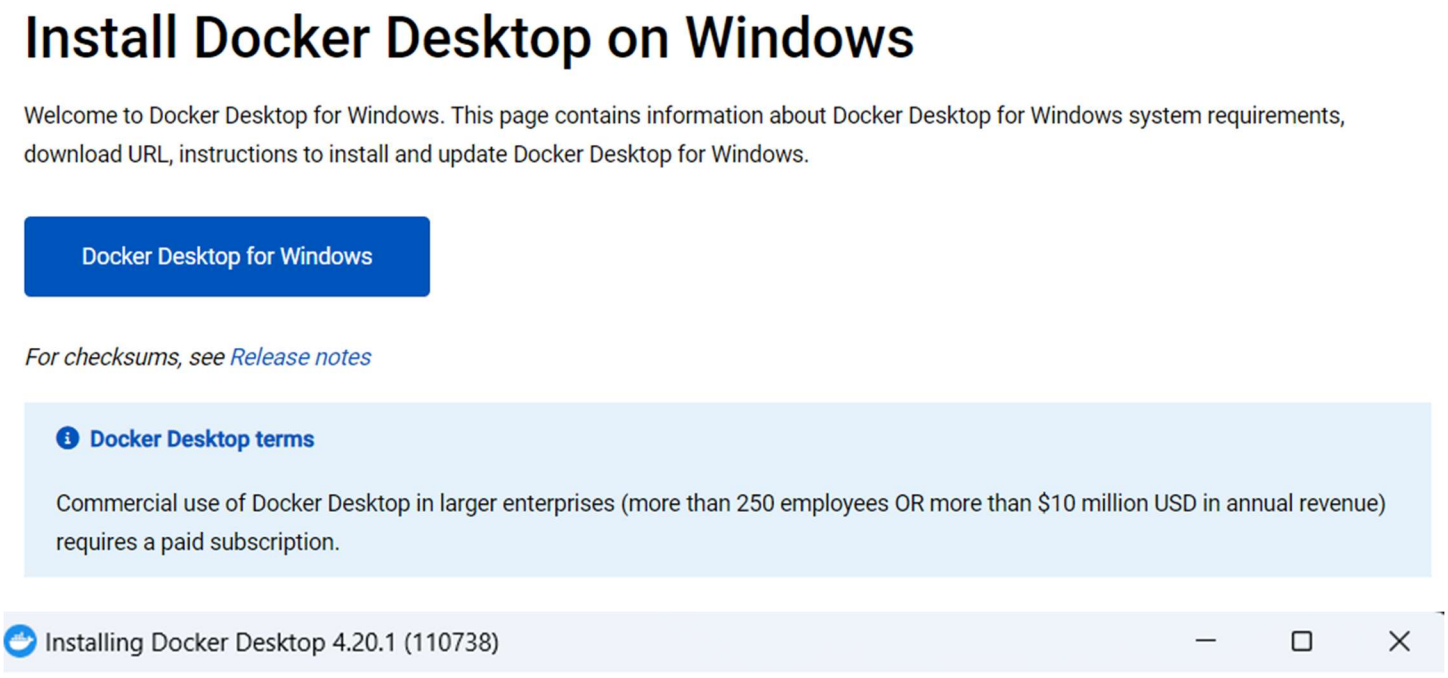
Let's click on "**Download docker desktop for Windows**"

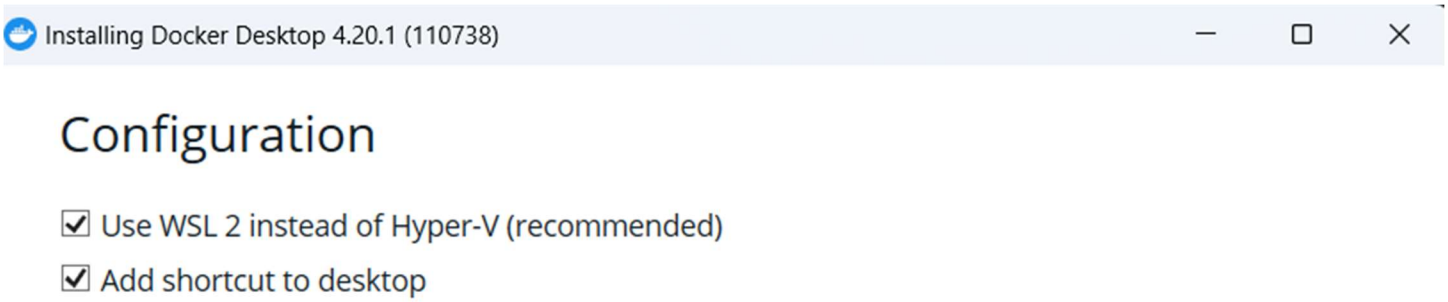
The docker desktop executable file will be downloaded to our download folder

Then, let's navigate to my download folder

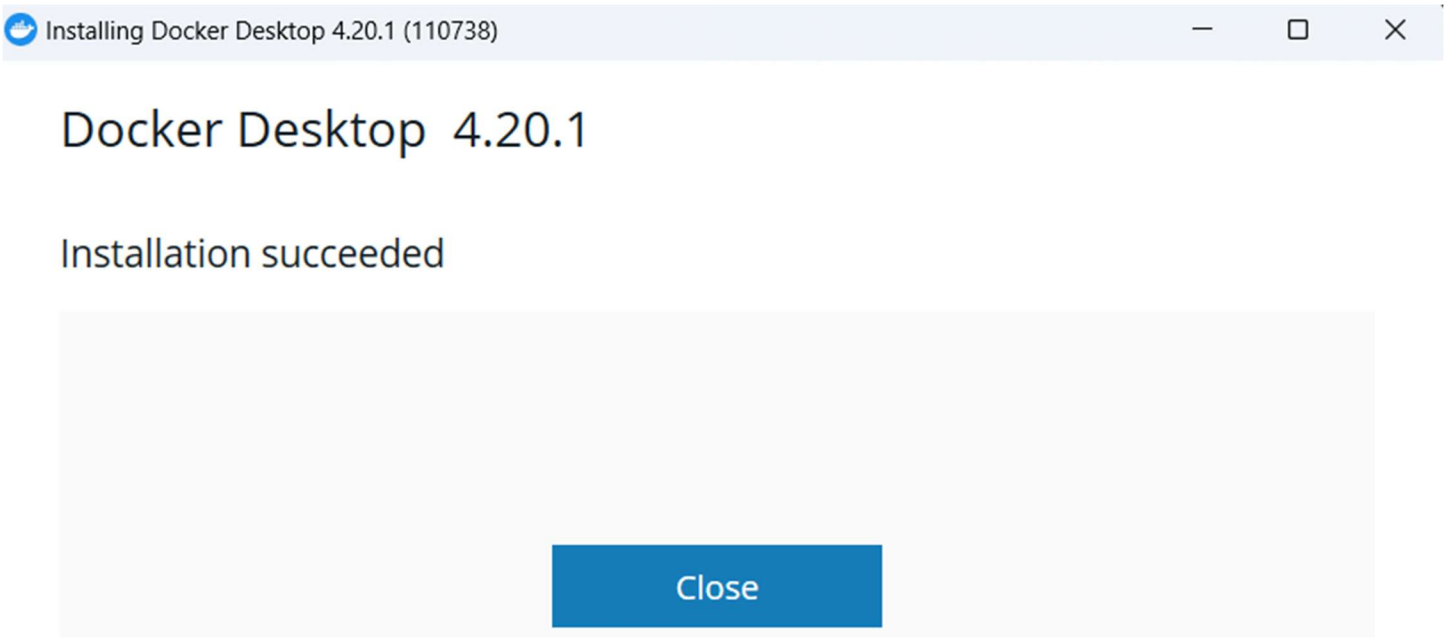
Double click on "Docker Desktop Installer.exe" and begin the installation process

Then click "OK" to use WSL 2 (Windows Subsystem for Linux) configuration instead of Hyper-V

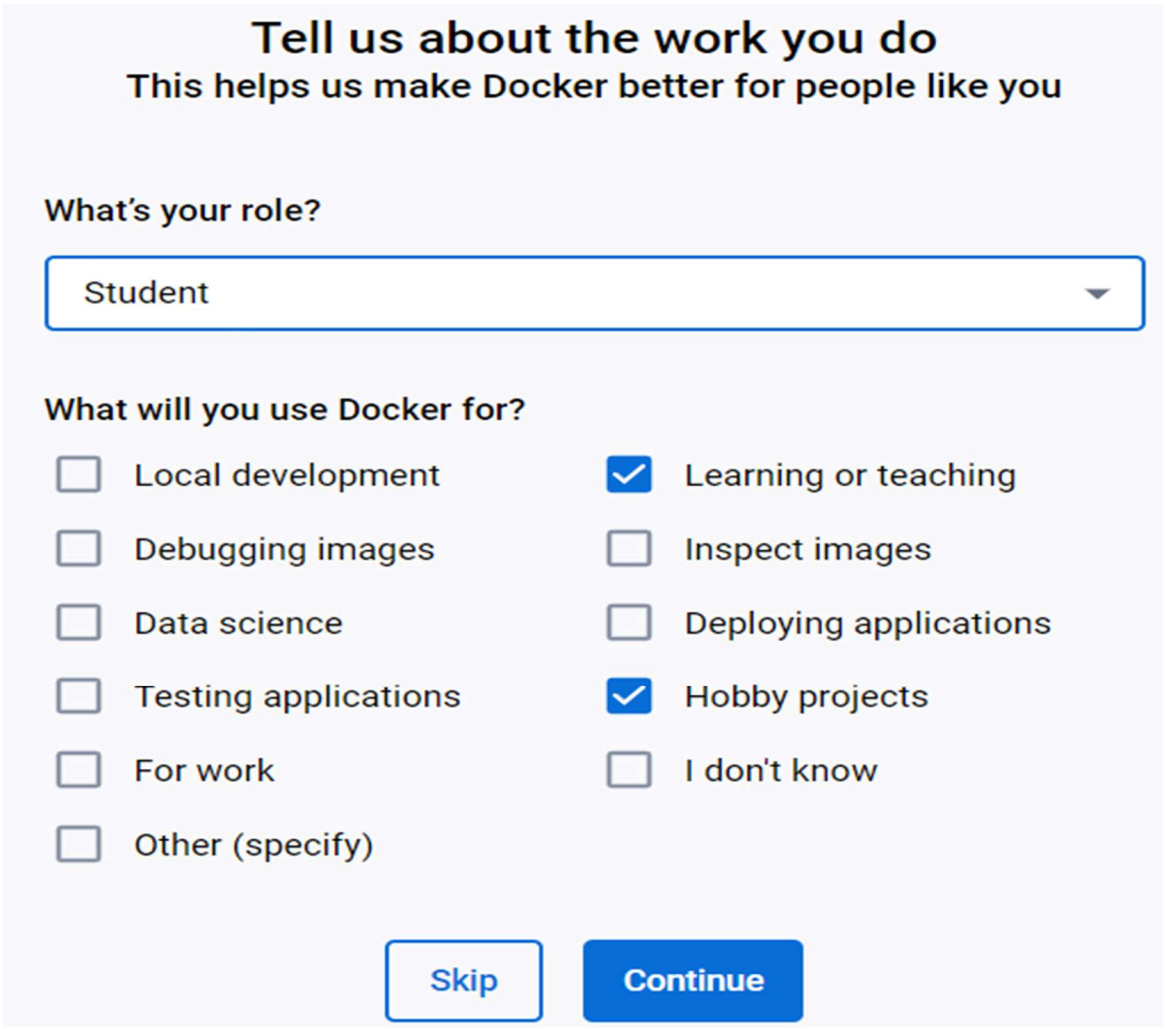


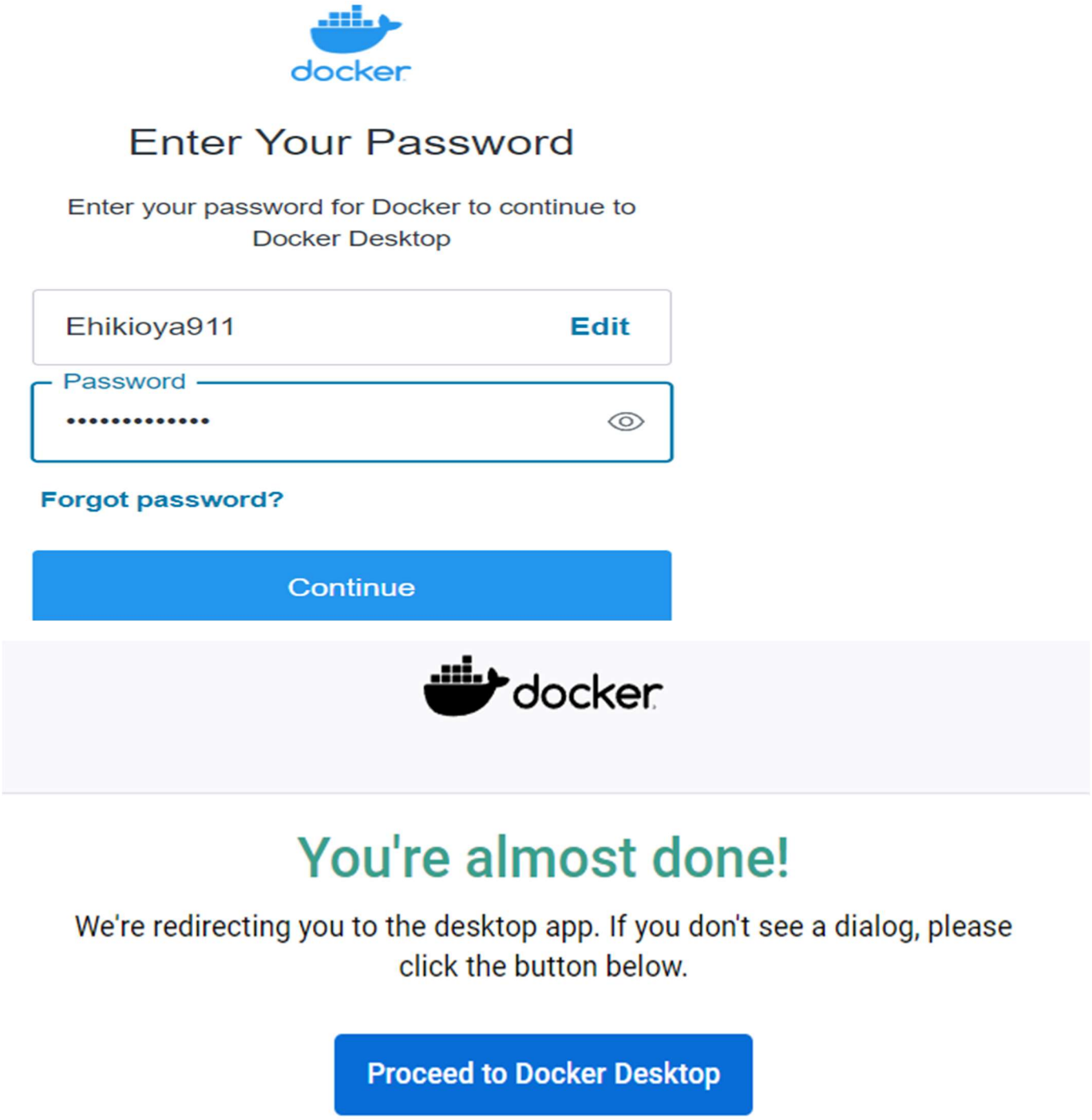




**Click on OK**




**Click on Accept for the docker service terms and agreements**

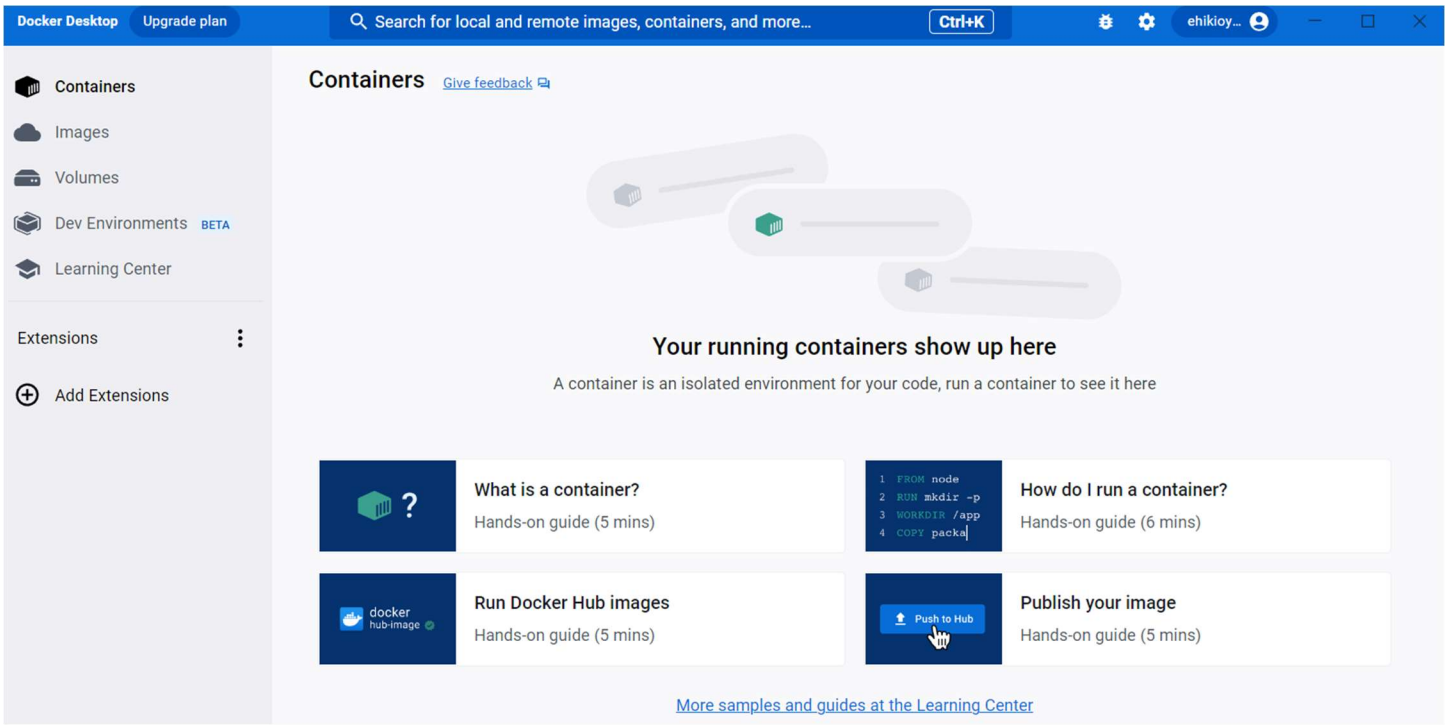





Docker desktop was successfully installed, so let's sign in

The Docker icon is now available on my desktop

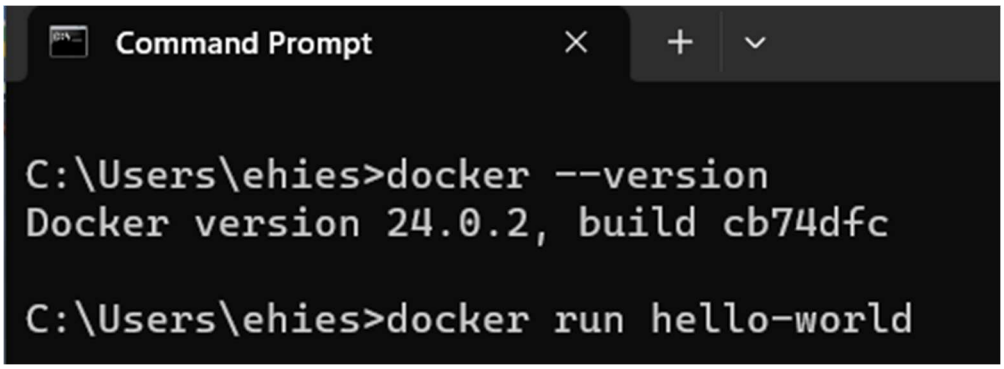
Now, let's double-click on the Docker Desktop icon to open it




**Now, let's verify that docker is running correctly by opening our command prompt to run few commands**

(1) docker --version

(2) docker run hello-world



```
C:\Users\ehies>docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/


C:\Users\ehies>
```

docker is running correctly as we can see from the above screenshots

## STEP 11: CREATE DOCKERFILE

A Dockerfile is a text document that contains all the commands that we are going to create a container image.

In this session, we will create the Dockerfile that will be used to build the container image for the jupiter website.

Recall that our Docker-Project folder was initially created and cloned from our GitHub repository

Let's begin by opening our visual studio code

Next step will be to open our Docker-Project folder using Visual studio code:

=>Visual Studio Code

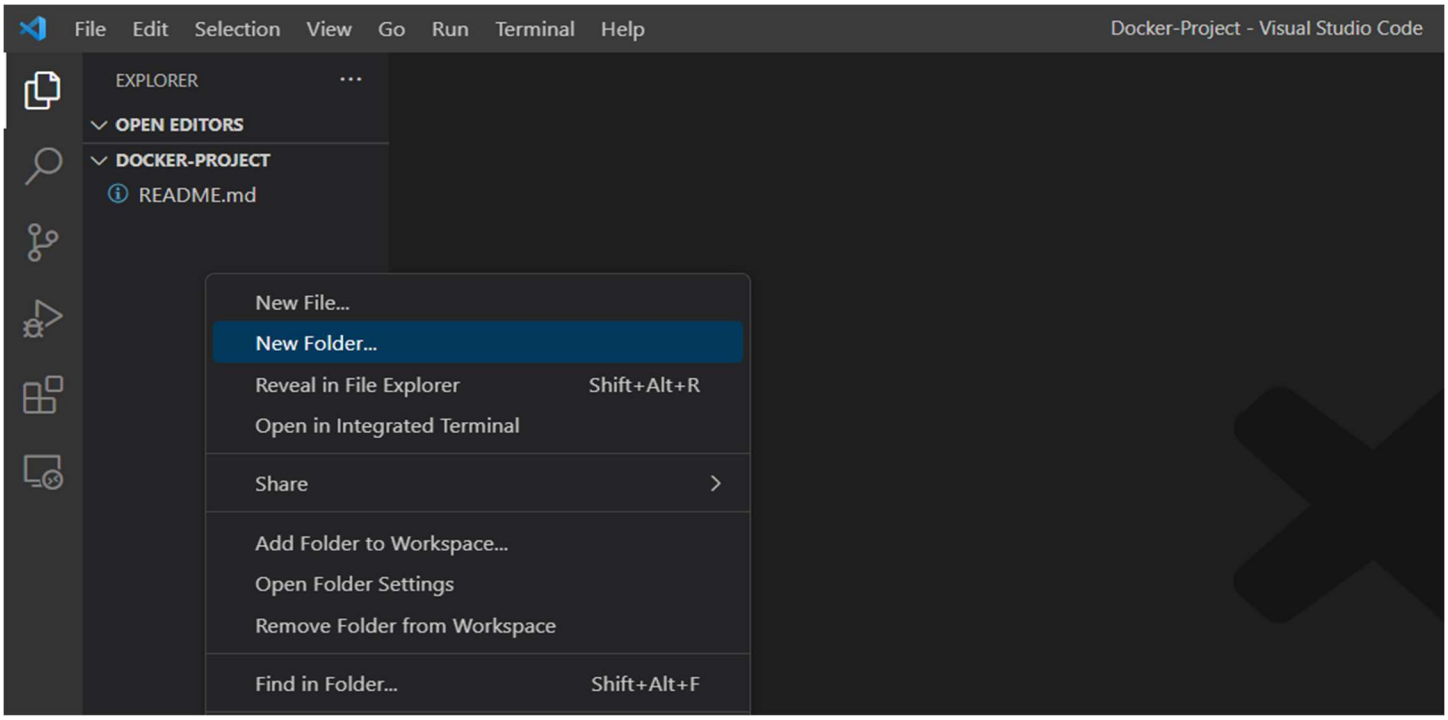
=>Click on file

=>Click on open folder

=>Select desktop

=>Select Docker-Project

=>Select open folder




We have successfully opened the Docker-project folder in VS-Code

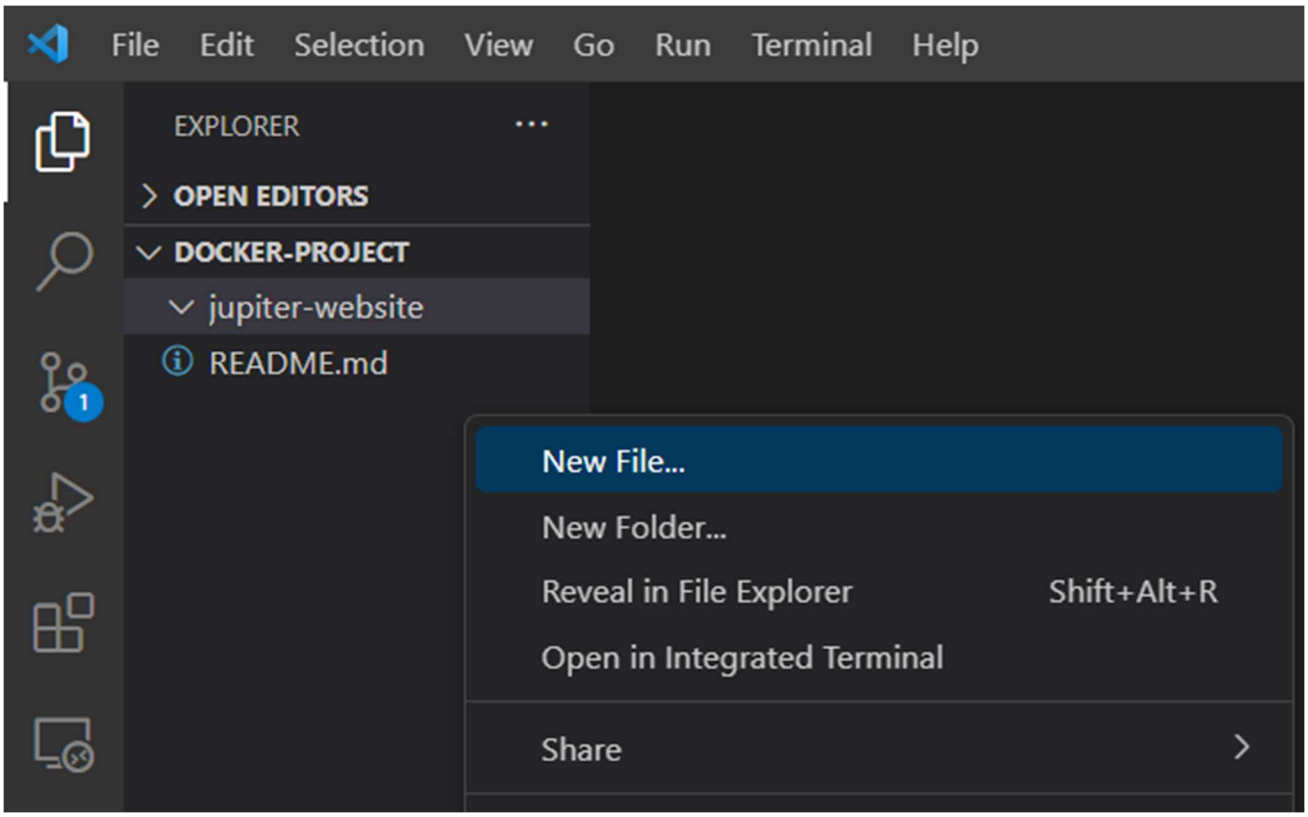
**The next step will be to create a sub-folder in the Docker-Project repository/folder**

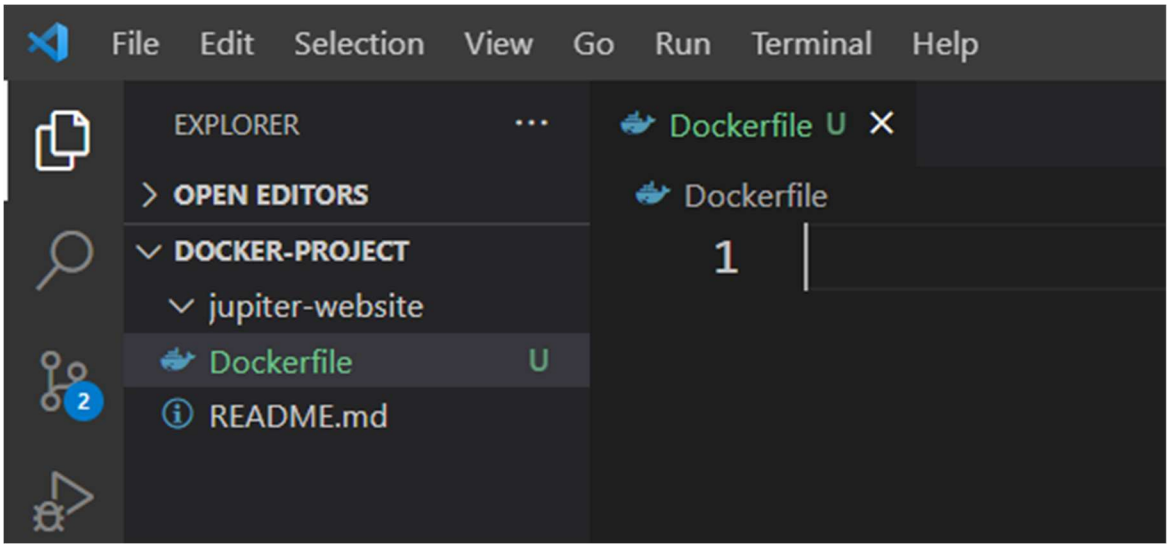
Let's call the sub-folder **jupiter-website**

This new folder will be used to store the Dockerfile for the **jupiter website**

Now let's select the **jupiter website** folder

double click on it and select create **“New file”**, let's name the file as **"Dockerfile"**




The **Dockerfile** is where we will write all our code that will be used to build our container image for the **jupiter website**.

Let's copy the following command and paste it in our **Dockerfile**

#!/bin/bash

sudo su

yum update -y

yum install -y httpd

cd /var/www/html

wget https://github.com/edwinesene/jupiter/archive/refs/heads/main.zip

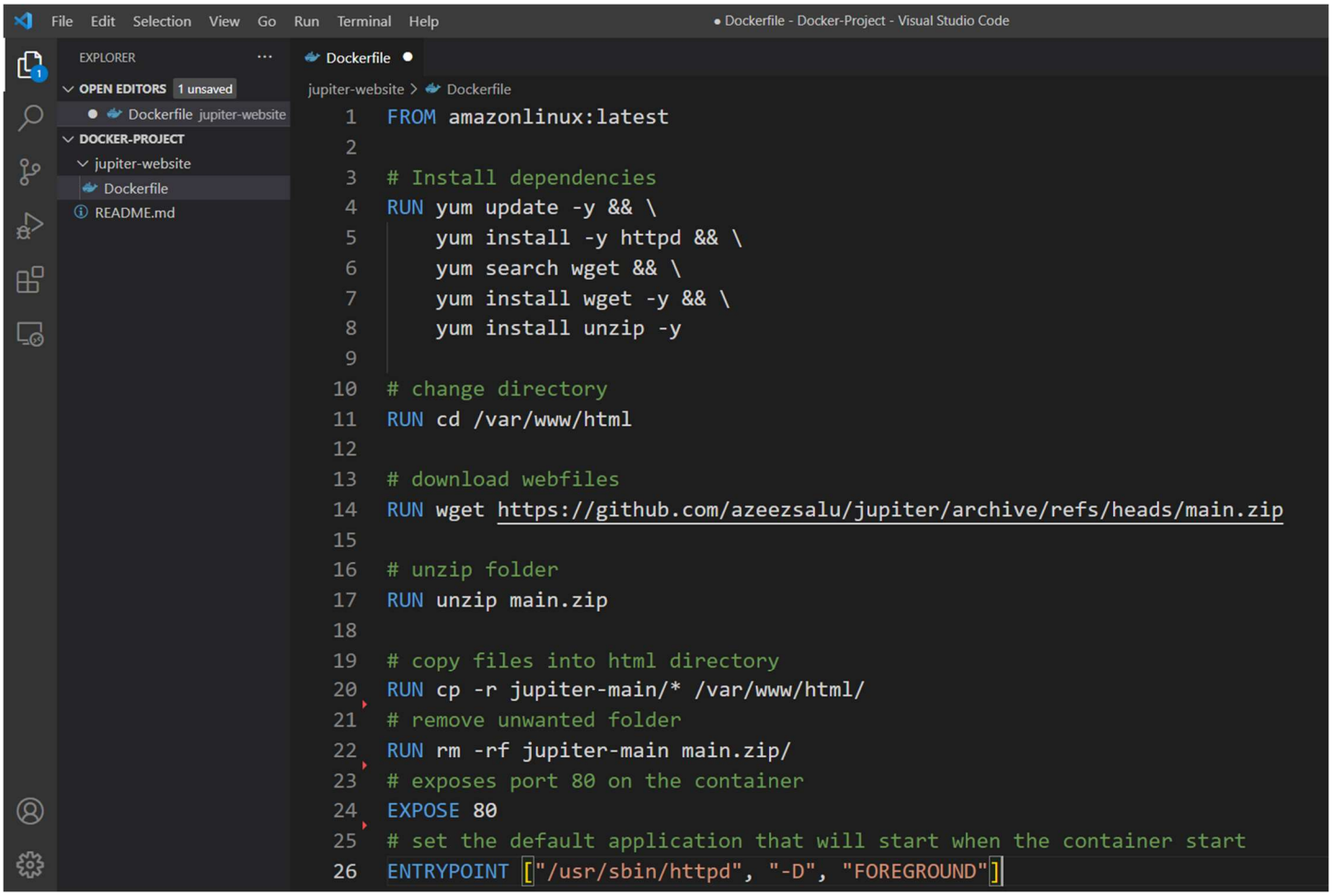
unzip main.zip

cp -r jupiter-main/* /var/www/html/

rm -rf jupiter-main main.zip

systemctl enable httpd

systemctl start httpd




**It is important to save the script in our Dockerfile, which is a format that Docker understands.**

You must start with a "**FROM**" when creating a Dockerfile as we can see below:

**FROM amazonlinux:latest**

```
# Install dependencies
RUN yum update -y && \
   yum install -y httpd && \
   yum search wget && \
   yum install wget -y && \
   yum install unzip -y

# change directory
RUN cd /var/www/html

# download webfiles
RUN wget https://github.com/azeezsalu/jupiter/archive/refs/heads/main.zip

# unzip folder
RUN unzip main.zip

# copy files into html directory
RUN cp -r jupiter-main/* /var/www/html/

# remove unwanted folder
RUN rm -rf jupiter-main main.zip/

# exposes port 80 on the container
EXPOSE 80
# set the default application that will start when the container start
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

Click on File and save all.

**The next step will be to push our update to our GitHub repository**

Open Visual Studio Code

Select **Source control** tab on the left

Enter a short message, I will enter "**create dockerfile**"

Click on **commit**

Click on **“sync”** to push the update to our **GitHub repository**

Now let's navigate to our GitHub repository to verify if Dockerfile was successfully updated

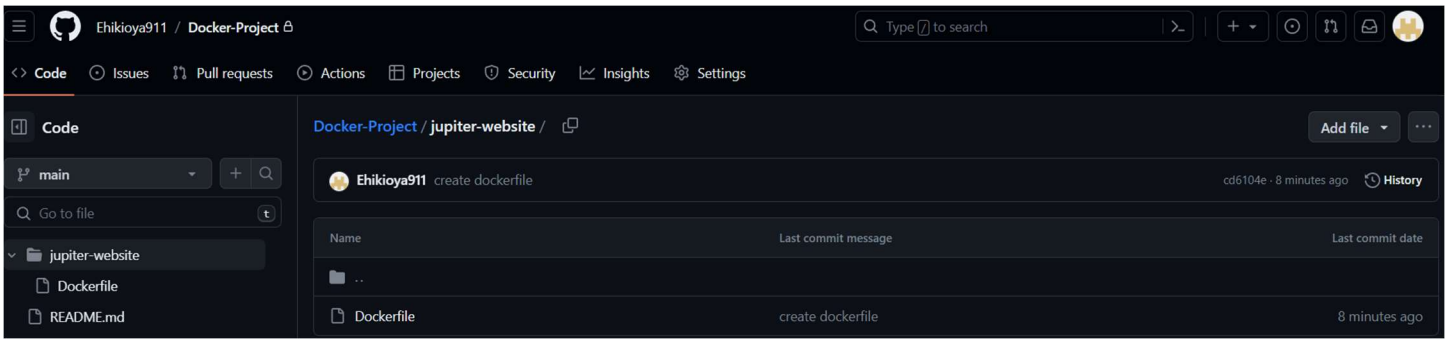
Navigate to GitHub account

Open your repositories

click on the Docker-Project repository

Click on the jupiter website

Click on Dockerfile




We can see from the screenshot that the **Dockerfile** was successfully pushed to our GitHub repository

## STEP 12: BUILD CONTAINER IMAGE

**We are going to use the Dockerfile we created in the last session to create a container image for the jupiter website.**

Let's begin by opening our Docker-project folder in VS-Code.

Click on the **jupiter-website folder** to expose the **Dockerfile**

**Right-click** on the **Dockerfile** and select **“Open in integrated Terminal”**

This will open an integrated Windows PowerShell terminal with Visual Studio code

Run **“ls”** to list the content of the **Dockerfile**

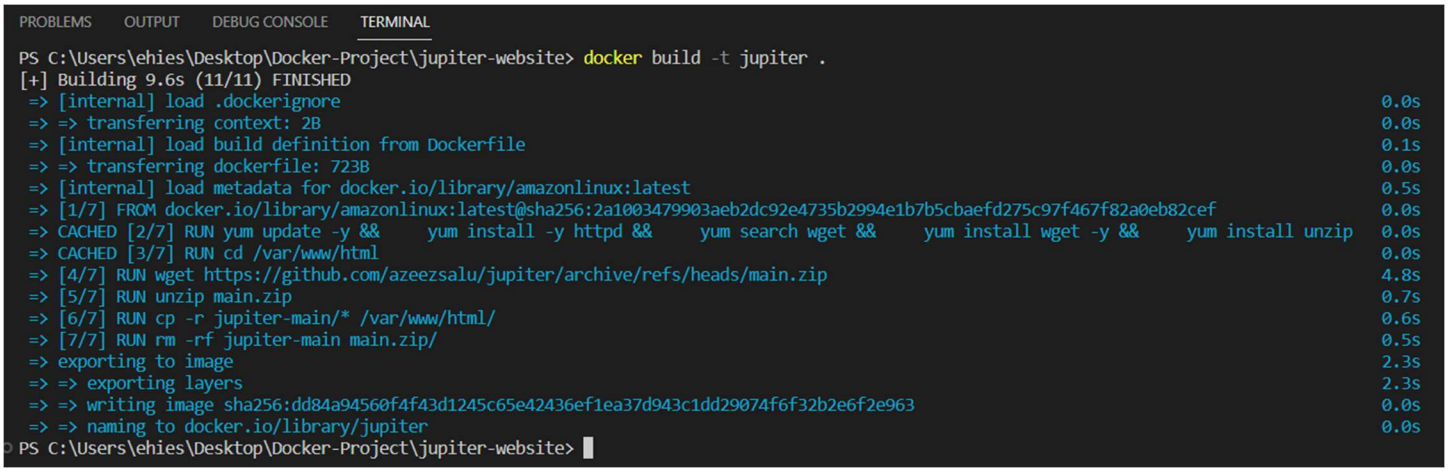
Very important to ensure that our integrated PowerShell terminal is launched in the jupiter-website folder

let's run ls to see what is in the jupiter website directory

**PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> ls**


**To build the image, run the command below:**

**docker build -t jupiter (where -t represents tag)**




Jupiter container image successfully built.

## STEP 13: START CONTAINER

Previously, we created a Dockerfile to build a jupiter image.

Let's use the **jupiter image** to start a **container**

1.Let’s open our Docker-Project folder in Visual Studio Code

2. Click on the **jupiter-website folder** to expose the **Dockerfile**

3. Right-click on the **Dockerfile** and select **“Open in integrated Terminal”**

4.Run "**docker image ls**" to list all available container images

PS C:\Users\ehies\Desktop\Docker-Project> docker images ls

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker image ls
REPOSITORY                                                TAG       IMAGE ID       CREATED        SIZE
763176333159.dkr.ecr.us-east-1.amazonaws.com/jupiter      latest    dd84a94560f4   7 weeks ago    266MB
ehikioya911/jupiter                                       latest    dd84a94560f4   7 weeks ago    266MB
jupiter                                                   latest    dd84a94560f4   7 weeks ago    266MB
2048-game                                                 latest    9c441596990f   7 weeks ago    182MB
```

**5.** Run **“docker images jupiter”** to list the specific **jupiter** container image

PS C:\Users\ehies\Desktop\Docker-Project> docker images jupiter

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker images jupiter
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
jupiter      latest    dd84a94560f4   11 hours ago   266MB
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
```

**6.Enter docker run -dp 80:80 jupiter to start the container**

**PS C:\Users\ehies\Desktop\Docker-Project> docker run -d -p 81:81 jupiter**

-d means the container should run in detached mode

-p means the container port

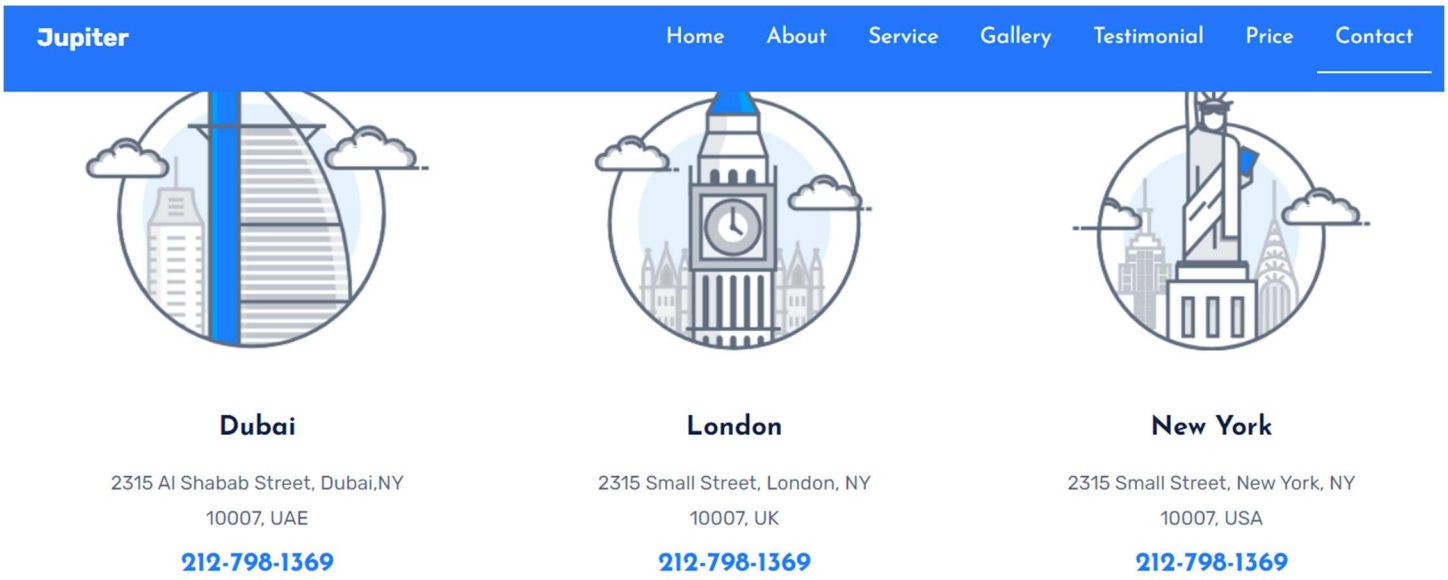
81:81 is the used port

```
PS C:\Users\ehies\Desktop\Docker-Project> docker run -dp 80:80 jupiter
ebdae81c3fe36c68b52481718ce5de4c8c632f8e04ace9071811438f0f535d6d
PS C:\Users\ehies\Desktop\Docker-Project>
```

The container successfully launched and running

To verify that the jupiter website is running successfully

Open a web browser > enter [http://localhost:80](http://localhost:80)




The Jupiter website was successfully launched.

Let’s list all docker containers that are running behind the scenes

PS C:\Users\ehies\Desktop\Docker-Project> **docker ps**

```
PS C:\Users\ehies\Desktop\Docker-Project>
PS C:\Users\ehies\Desktop\Docker-Project> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                    NAMES
63703836e4f0   jupiter   "/usr/sbin/httpd -D …"   54 minutes ago   Up 54 minutes   80/tcp, 0.0.0.0:81->81/tcp   busy_chebyshev
PS C:\Users\ehies\Desktop\Docker-Project>
```

**Now, let's stop the container**

Now, we will run **docker stop + container id**

**PS C:\Users\ehies\Desktop\Docker-Project> docker stop 63703836e4f0**

```
PS C:\Users\ehies\Desktop\Docker-Project> docker stop 63703836e4f0
63703836e4f0
PS C:\Users\ehies\Desktop\Docker-Project>
```

**Now if we run docker ps again,**

**PS C:\Users\ehies\Desktop\Docker-Project> docker ps**

```
PS C:\Users\ehies\Desktop\Docker-Project> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\ehies\Desktop\Docker-Project>
```

we can see that the jupiter container is gone

# STEP 14: CREATE A REPOSITORY IN DOCKER HUB

## The repository will be used to store container images

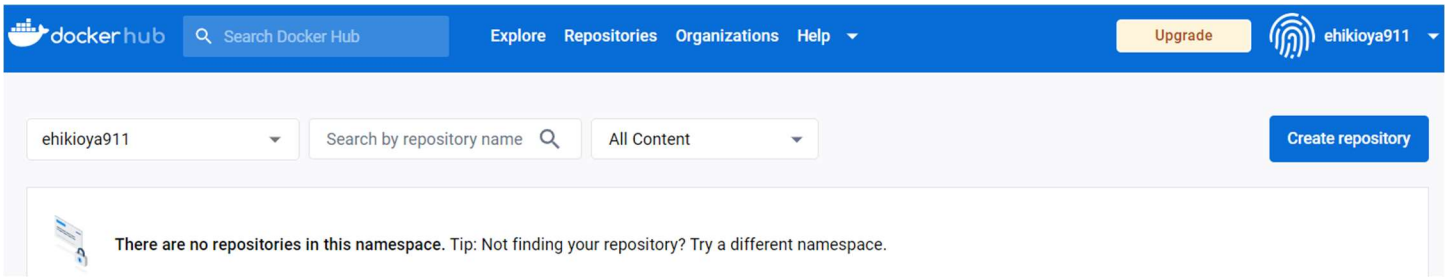
Navigate to hub.docker.com

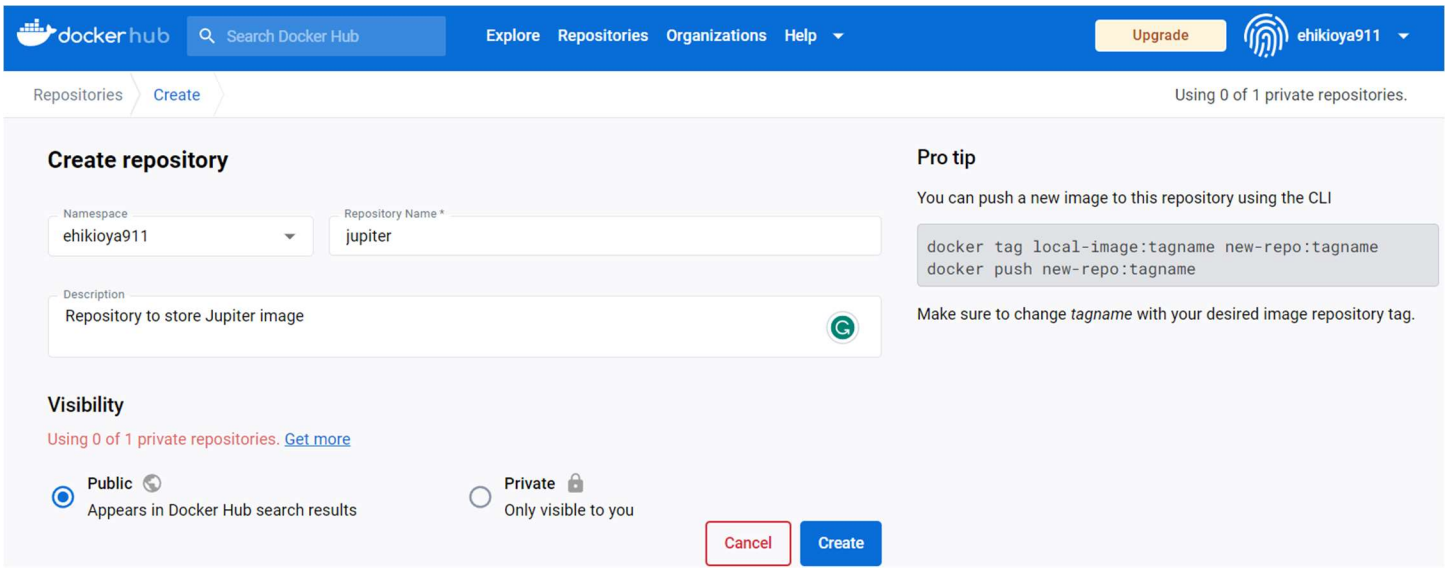
Sign into account

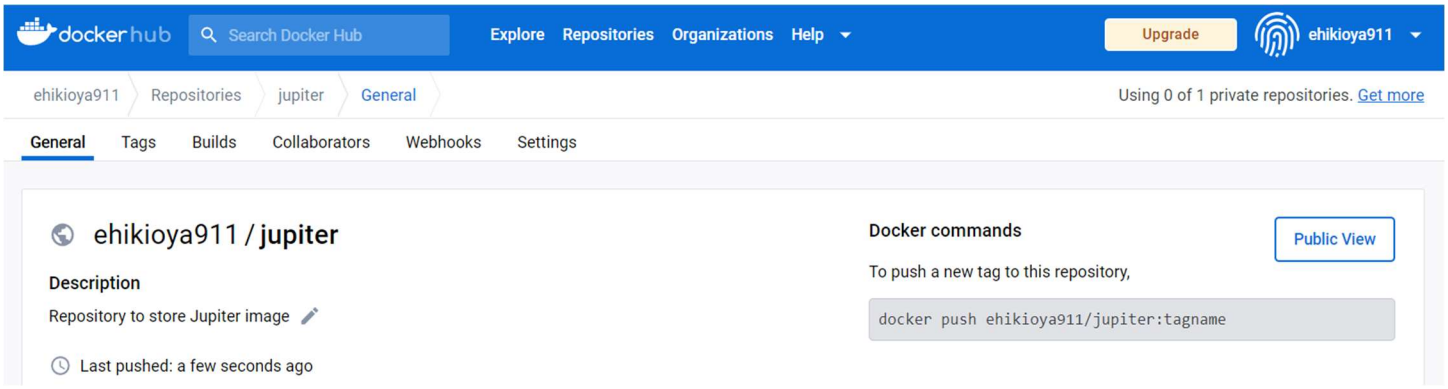
Click on repository

Click on create repository

Enter repository name and description > Click on create repository




The repository "**jupiter**" was successfully created in docker hub.

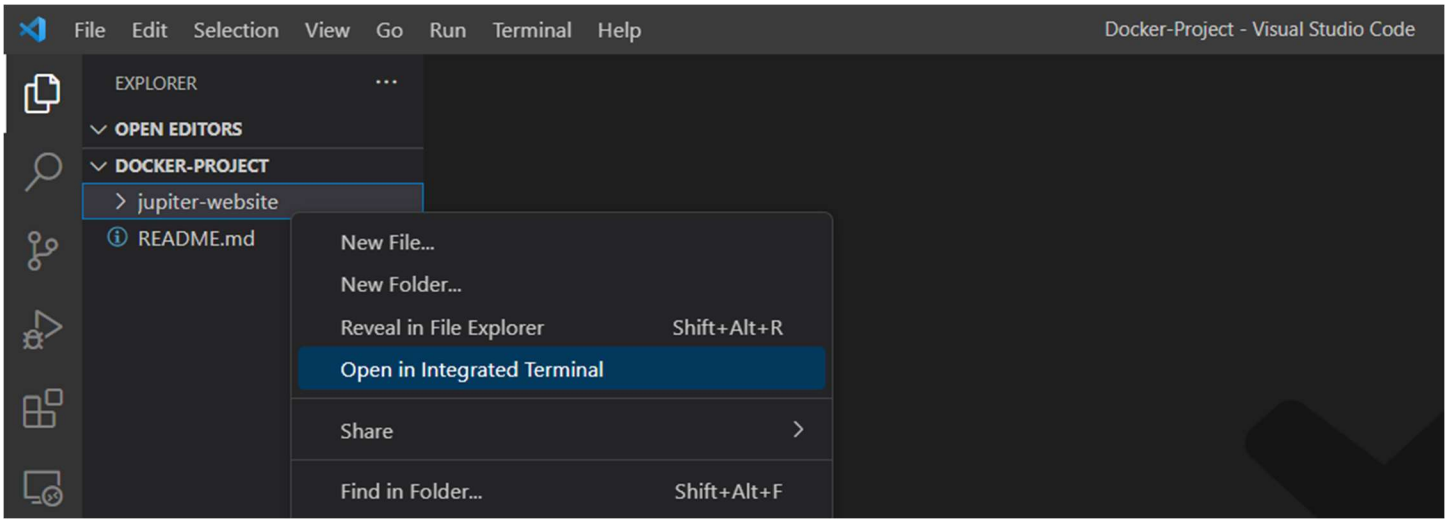
## STEP 15: PUSH CONTAINER IMAGE TO DOCKER HUB REPOSITORY

## Let's push the Jupiter image to our docker hub repository

First, let’s open our visual studio code

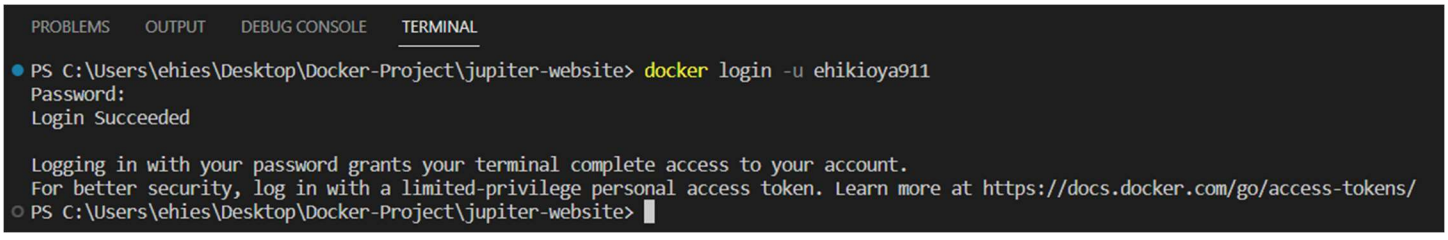
Now, let’s load our Docker-Project folder

Then we need to right click on jupiter-website and open integrated terminal




Run docker login -u username

Enter password


Now, we need to list the docker images we currently have

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> **docker image ls**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
jupiter       latest    dd84a94560f4   17 hours ago    266MB
```

Run **“docker images jupiter”** to list only the **jupiter** container images

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker images jupiter

```
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker images jupiter
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
jupiter       latest    dd84a94560f4   17 hours ago    266MB
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
```

**Now let's use the docker tag command to give the jupiter image a new name**

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker tag jupiter ehikioya911/jupiter

```
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker tag jupiter ehikioya911/jupiter
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker image ls
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
ehikioya911/jupiter   latest    dd84a94560f4   18 hours ago    266MB
jupiter               latest    dd84a94560f4   18 hours ago    266MB
```

We have successfully tagged the docker image

**Next step will be to push the image to the docker repository**

**PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker push ehikioya911/jupiter**

```
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker push ehikioya911/jupiter
Using default tag: latest
The push refers to repository [docker.io/ehikioya911/jupiter]
eb776baf2926: Pushed
160e767444b4: Pushed
16cf6dafdab5: Pushed
d957fd84e554: Pushed
a431e7961bc9: Pushed
b96fdda2a387: Pushed
ccaa10561e98: Mounted from library/amazonlinux
latest: digest: sha256:b1636ec38a30a780e0d9b2f609d1756f98c6d9341fbd5df4ba99c79013d7897d size: 1787
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
```

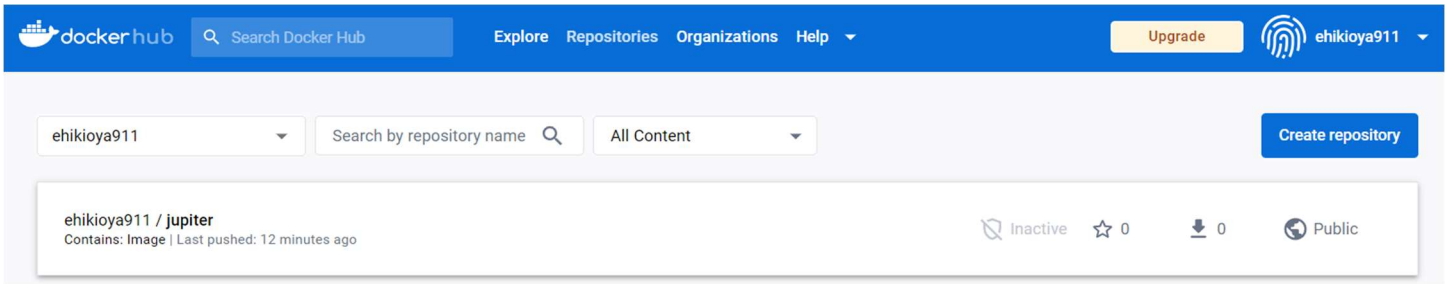
Successfully pushed the ehikioya911/jupiter container image to the docker hub repository.

Now, let's navigate o to our docker hub repository to verify if the image was successfully pushed

login to **docker hub**

click on **“Repository”**

The image was successfully pushed to our docker repository as shown from the screenshot below.



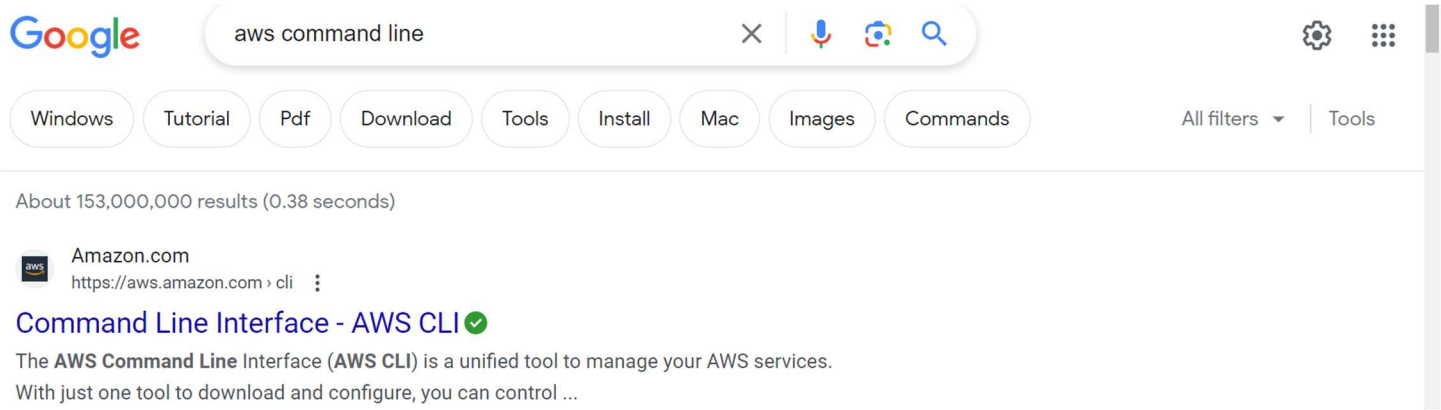

## STEP 16: INSTALL AWS CLI

## The AWS CLI was previously installed on my Windows machine

However, let's quickly walk through the process for the purpose of this project.

Go to browser and enter **https://google.com**

Search for **"AWS command line"**

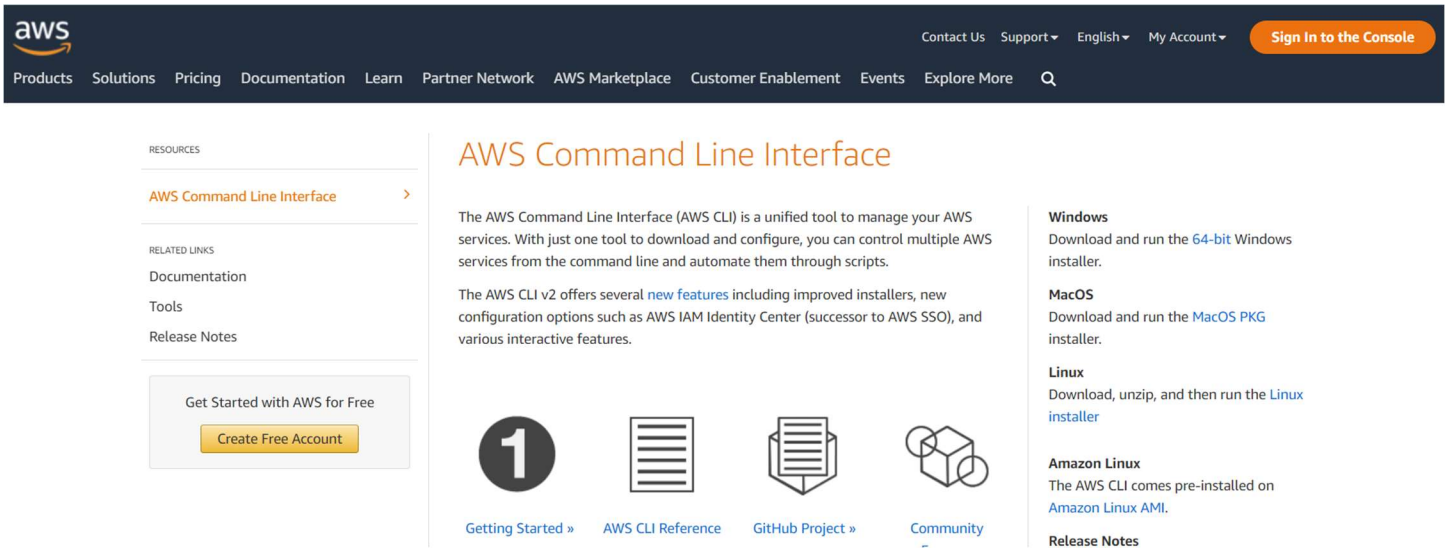



Click on the first URL **https://aws.amazon.com > cli**

Click on (1) Getting started

Navigate to the left-hand side of your screen to select the **"Get started"** drop down arrow

Click on Install/Update

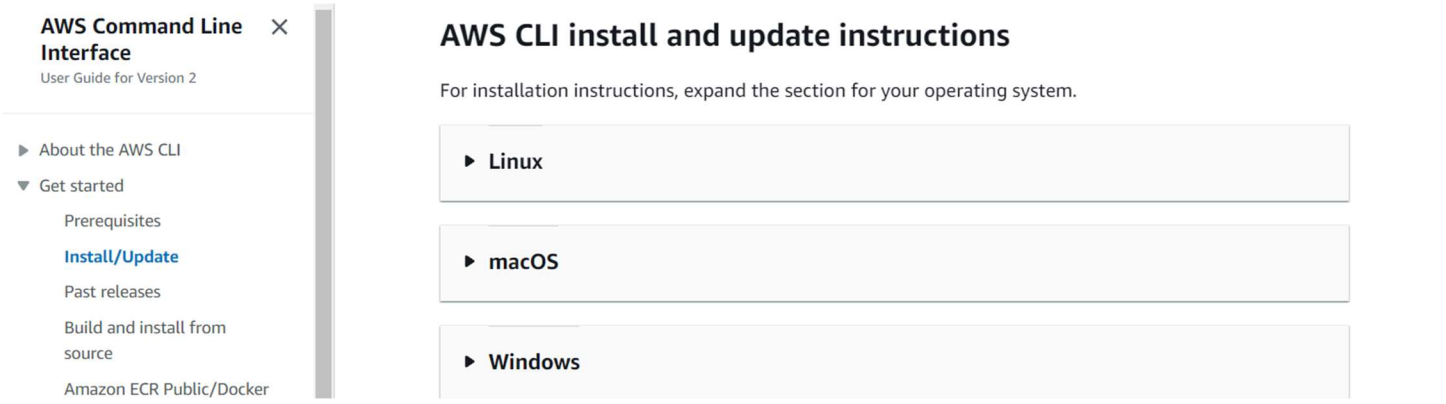



Scroll down towards the bottom of the screen to "AWS CLI install and update instructions"

Select the desired Operating System

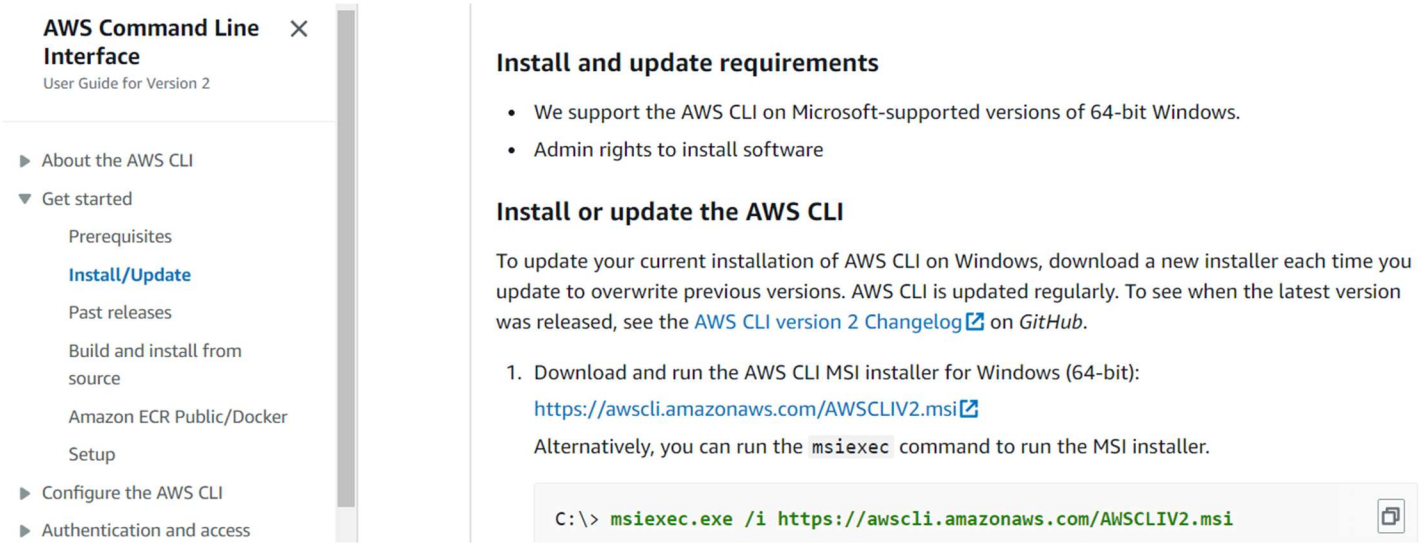
**Windows, MacOS or Linux**

I will select the Windows drop down since my Operating System is Windows

Click this link to download the executable file

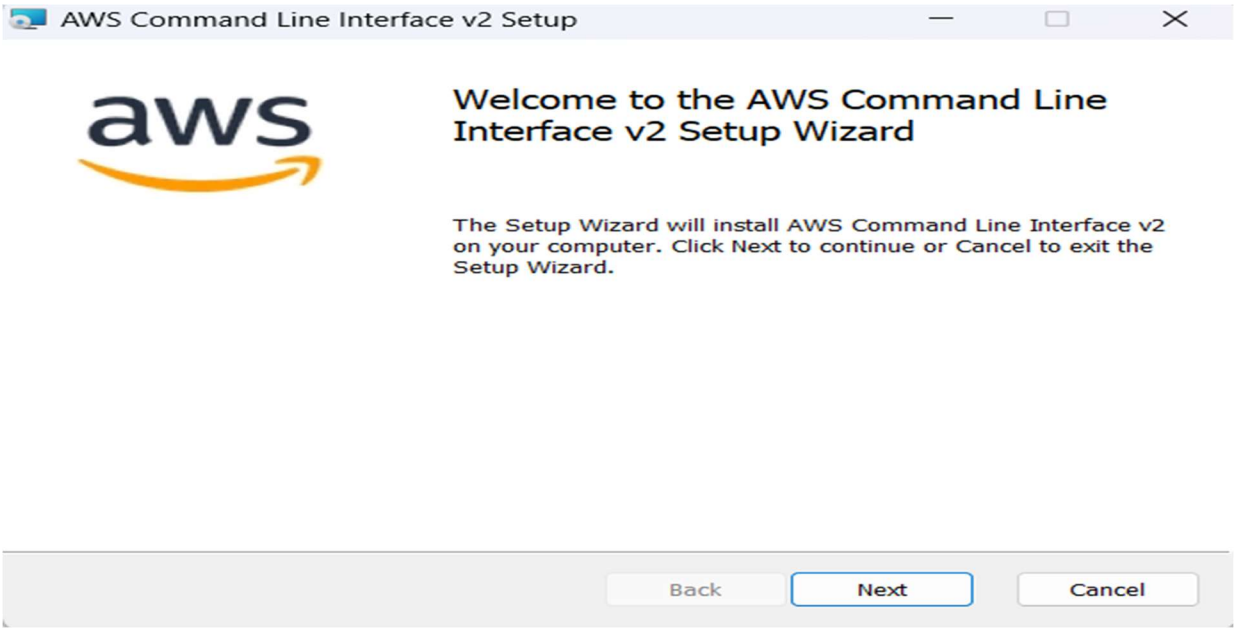
https://awscli.amazonaws.com/AWSCLIV2.msi



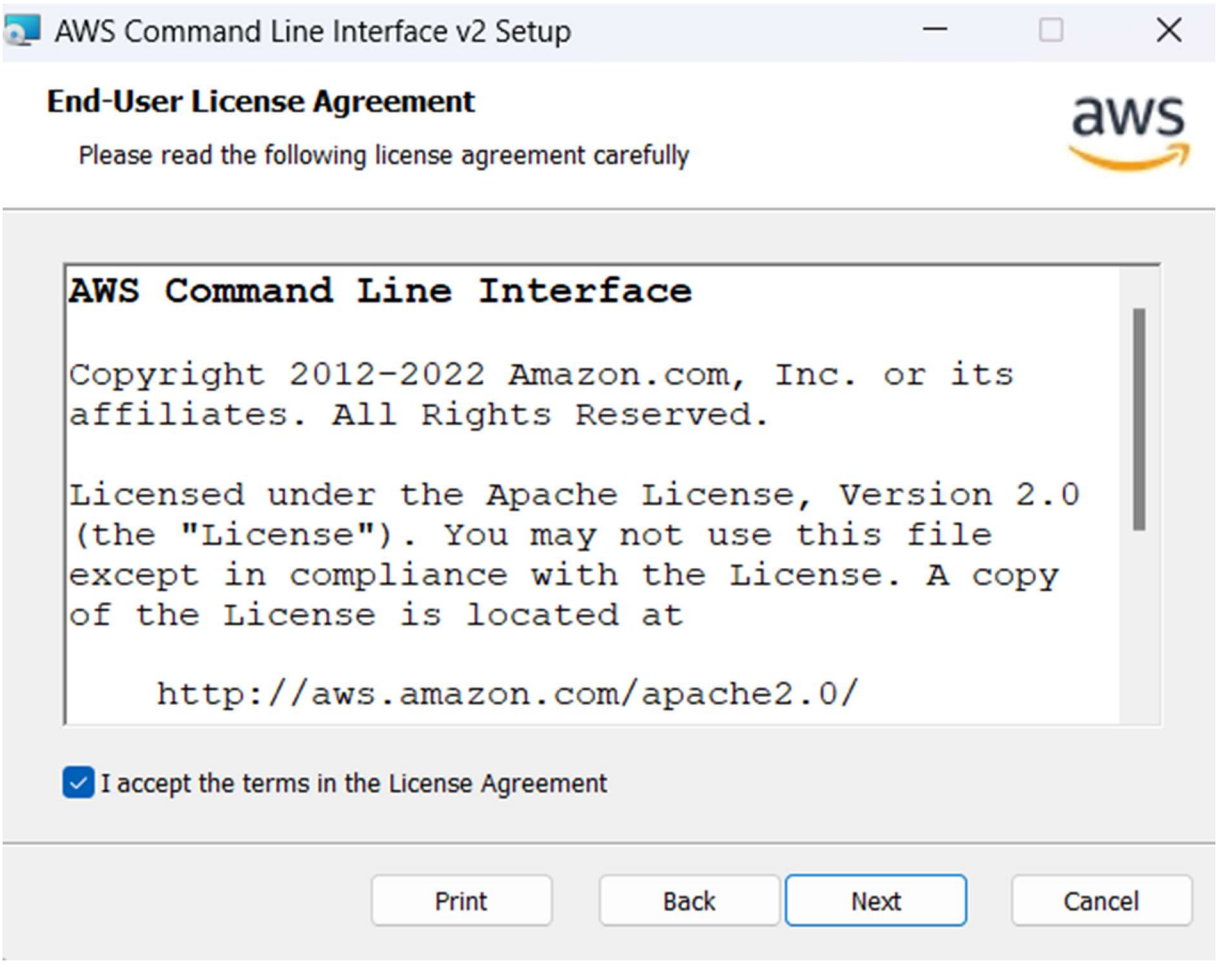

**Navigate to your download directory to open the executable file AWSCLIV2.msi**

The installation wizard will open

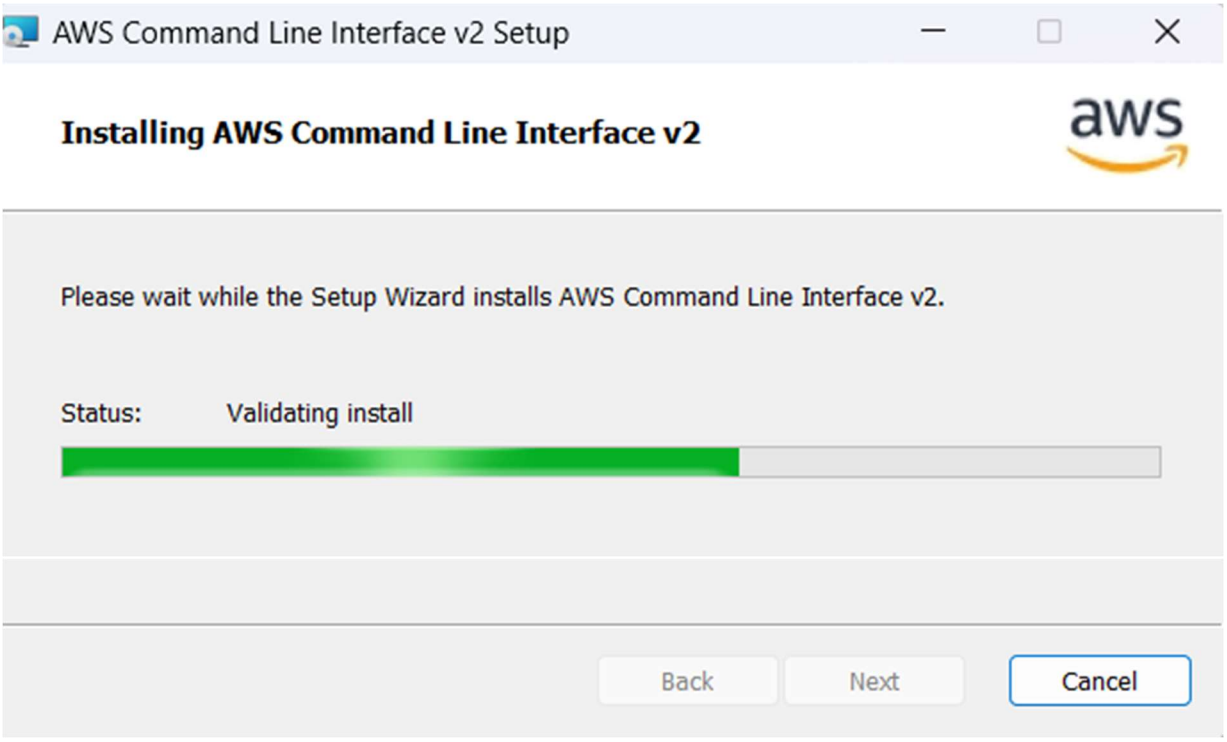
Click on NEXT

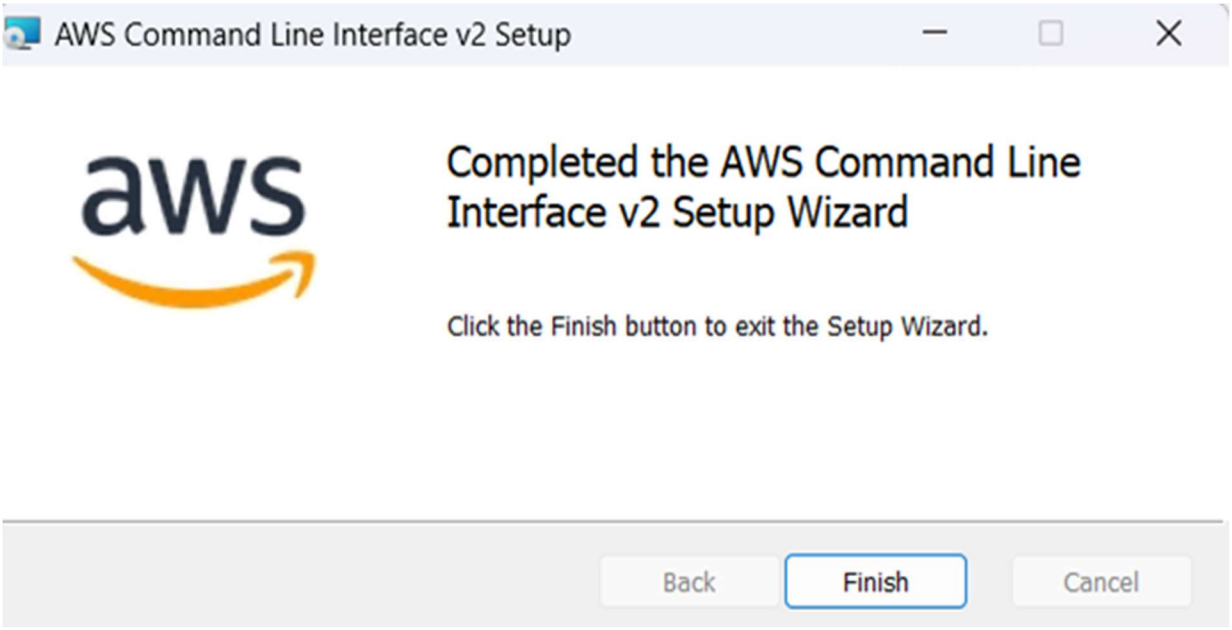



Accept the terms in the license agreement




Click on "NEXT" and follow the steps till "Finish" to install cli

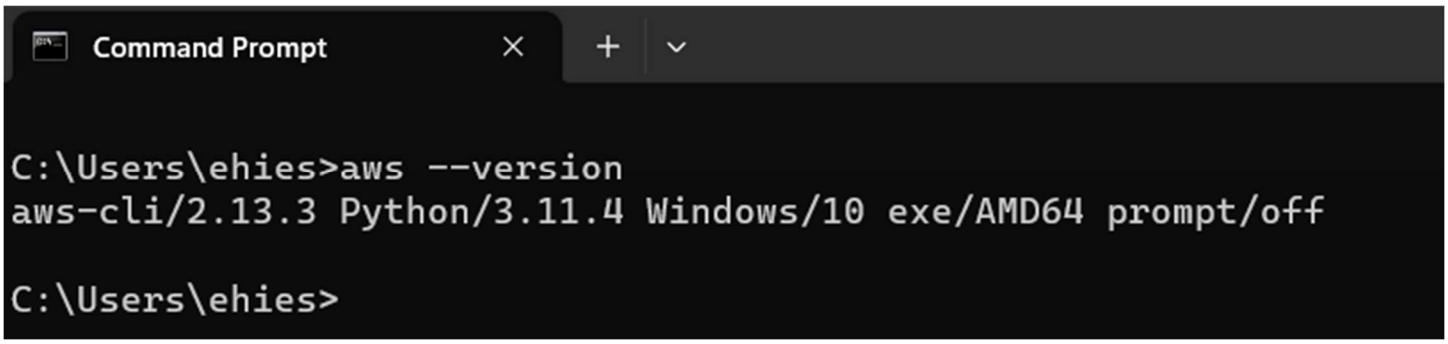





**Let's verify to ensure the AWSCLI was successfully installed**

Let’s open the command prompt

Run the command "aws --version"




The AWSCLI was successfully installed

## STEP 17: CREATION OF IAM USER CREDENTIALS

The main task here will be to push the container image we created in our previous stage to the Elastic Container Registry (ECR) in our AWS account.

The Elastic Container Registry (ECR) is a service that allows us to store our container images in AWS.

Note that the Elastic Container Registry (ECR) is similar to docker hub for storing container images.

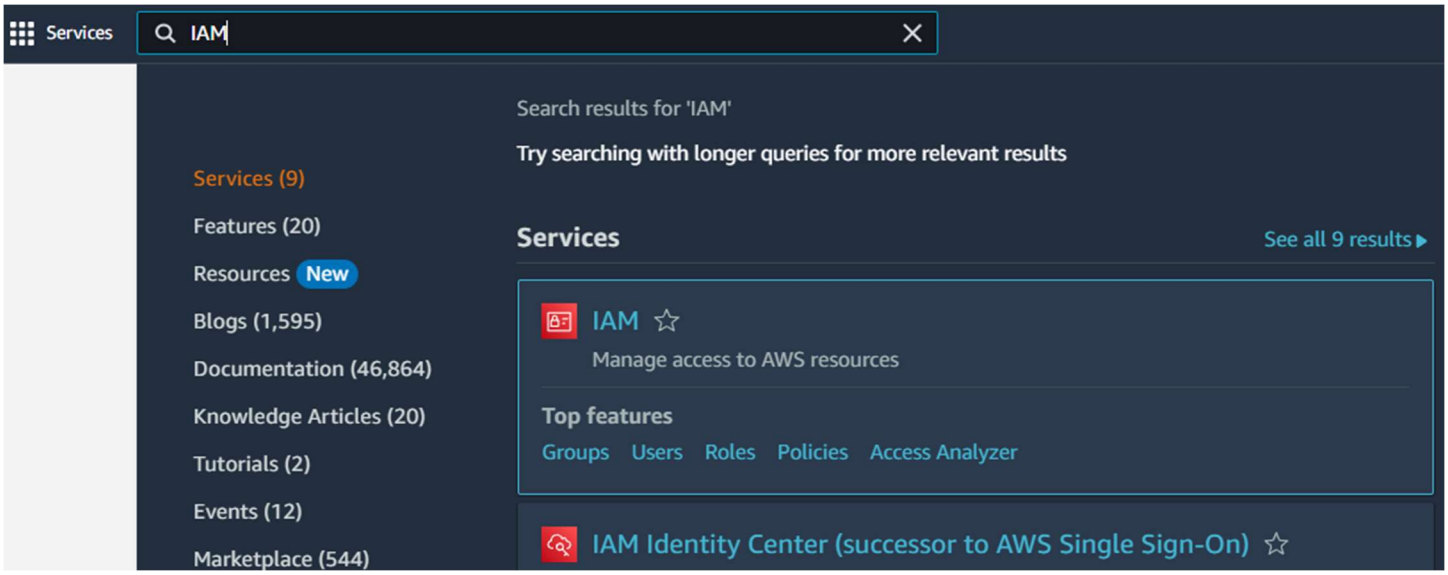
To push an image to the ECR, first we must create an **IAM** user with **programmatic access**

Then we will use the (1) AWS CLI (2) The user access key (3) The secret access key to authenticate with AWS to push the container images to ECR.

After we have pushed the container images to ECR, we will then use it to run ECS Fargate containers.

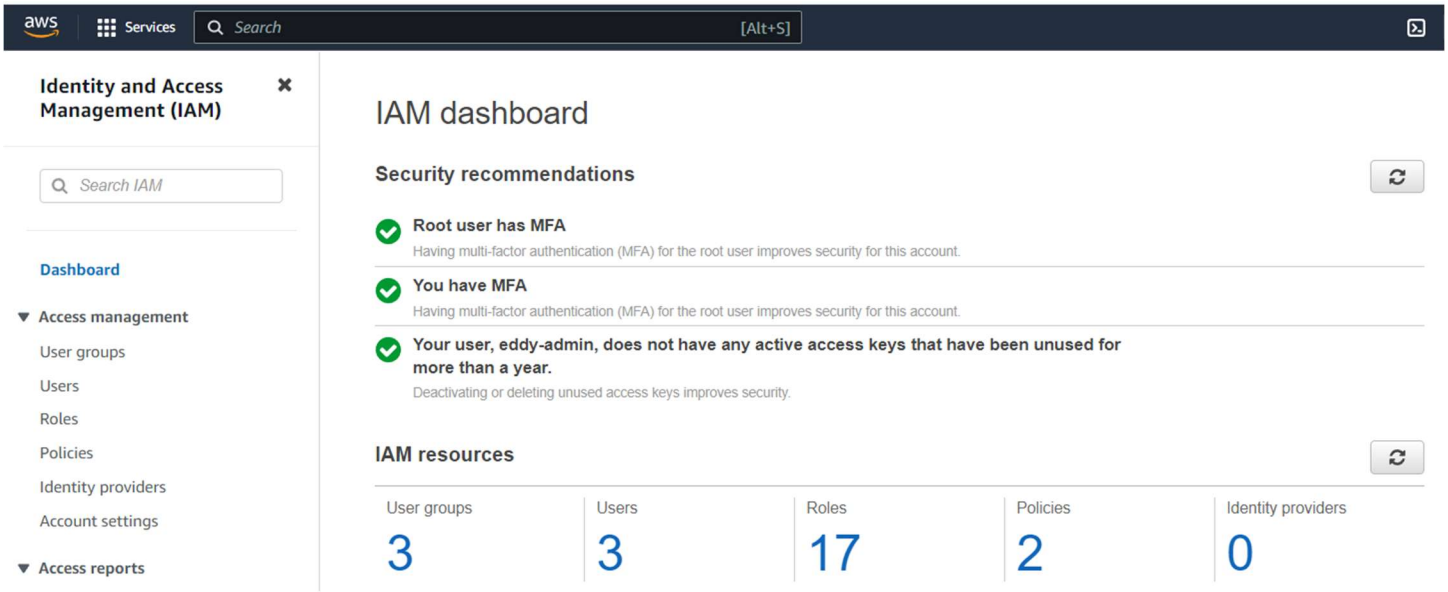
Let's login to our AWS management console to create the **IAM programmatic access**

Navigate to Services on the left-hand corner and enter **IAM** in the search box



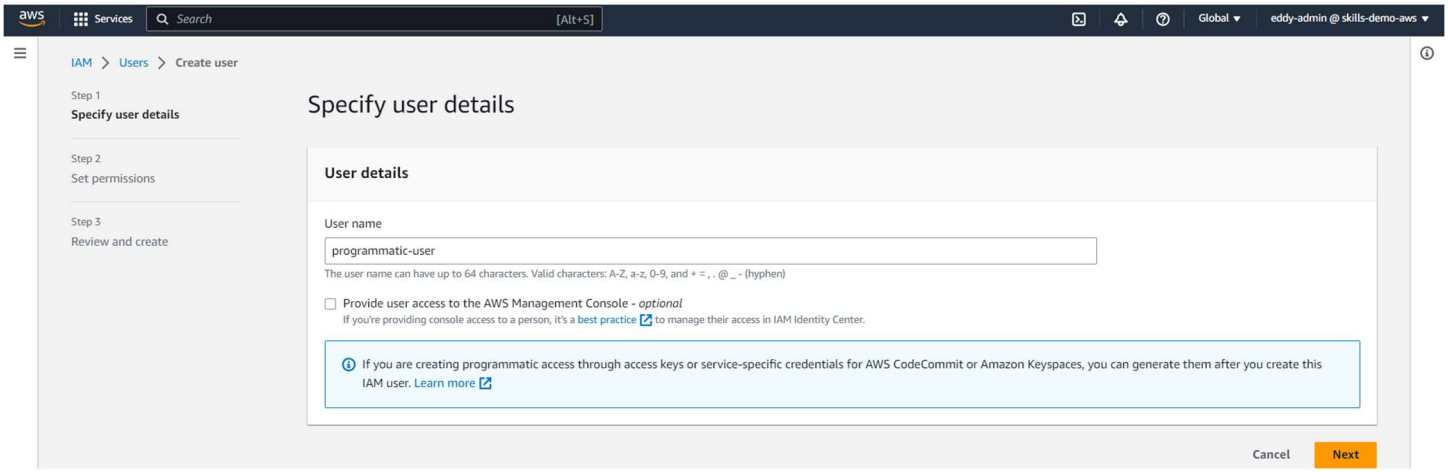

Select Users

Click Add Users

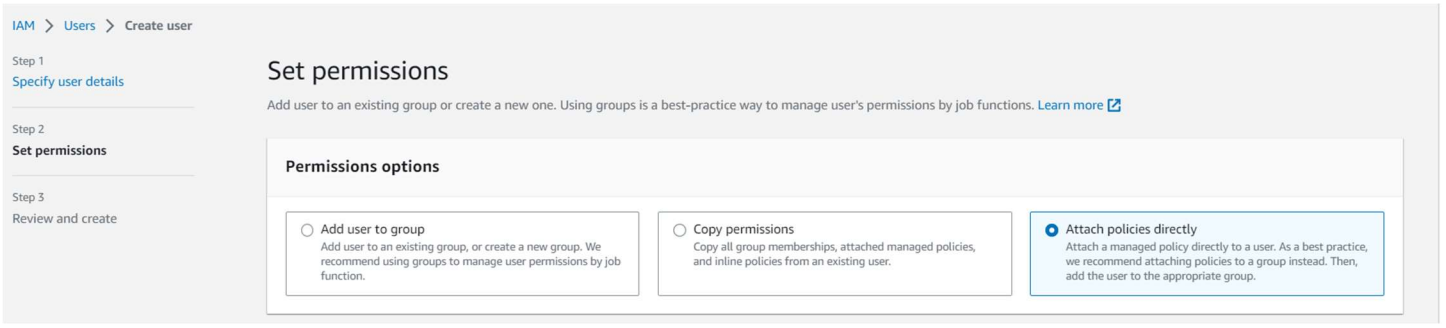


Name User, and I will call this user “**programmatic-user**”

Click Next

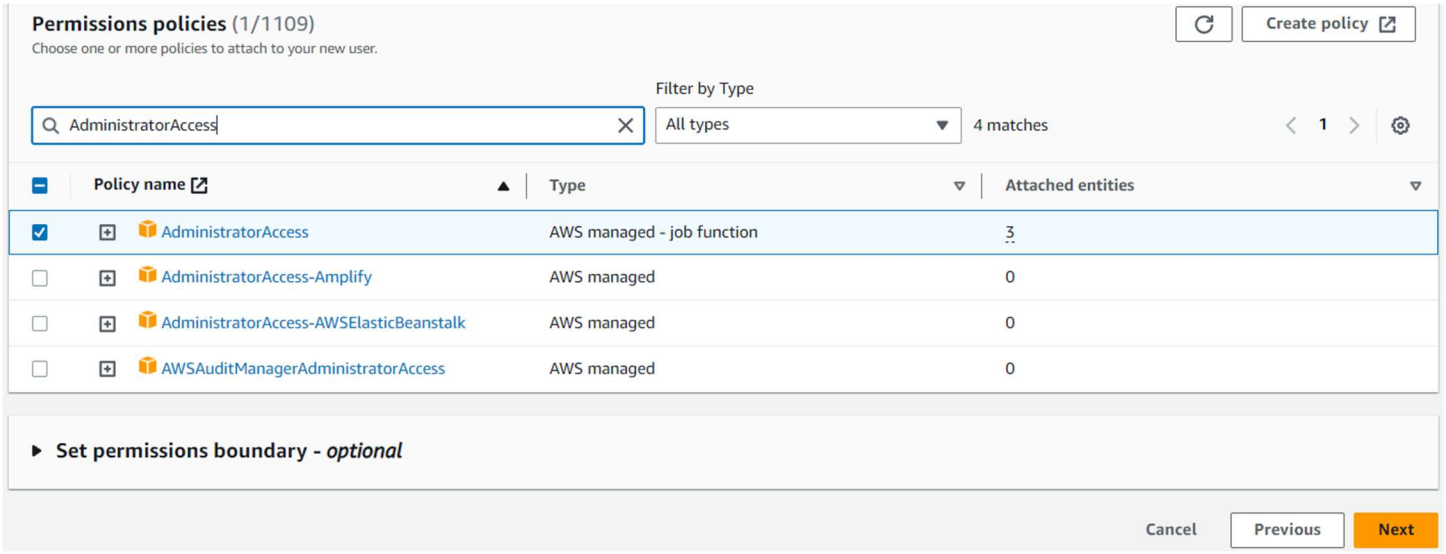



Click on Attach policies directly

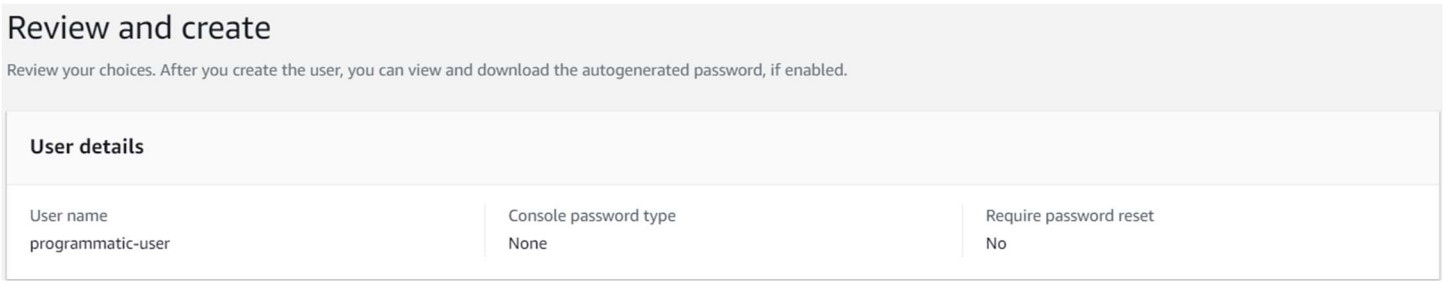



Select **AdministratorAccess**

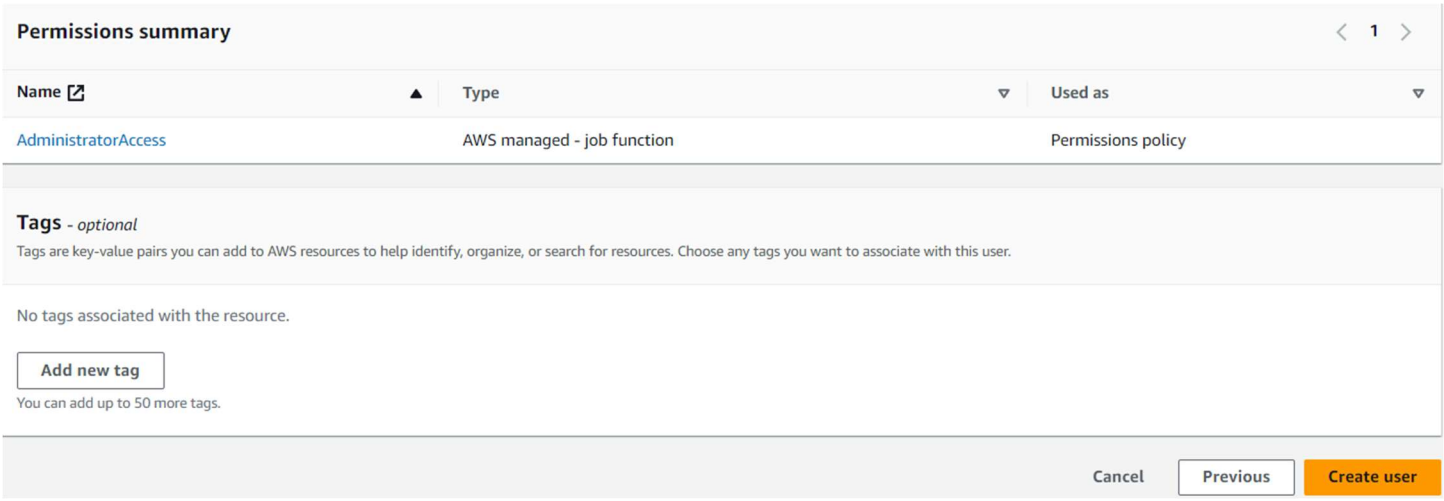
Click on **Next**

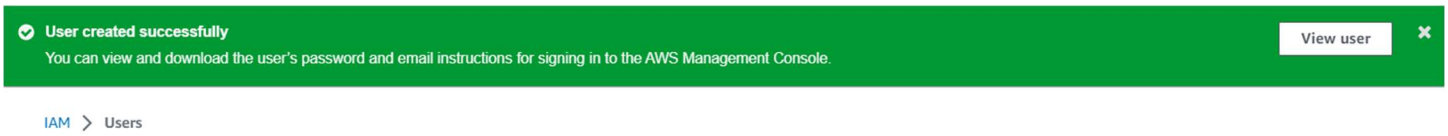


Review and create user




Click on **“Create user”**

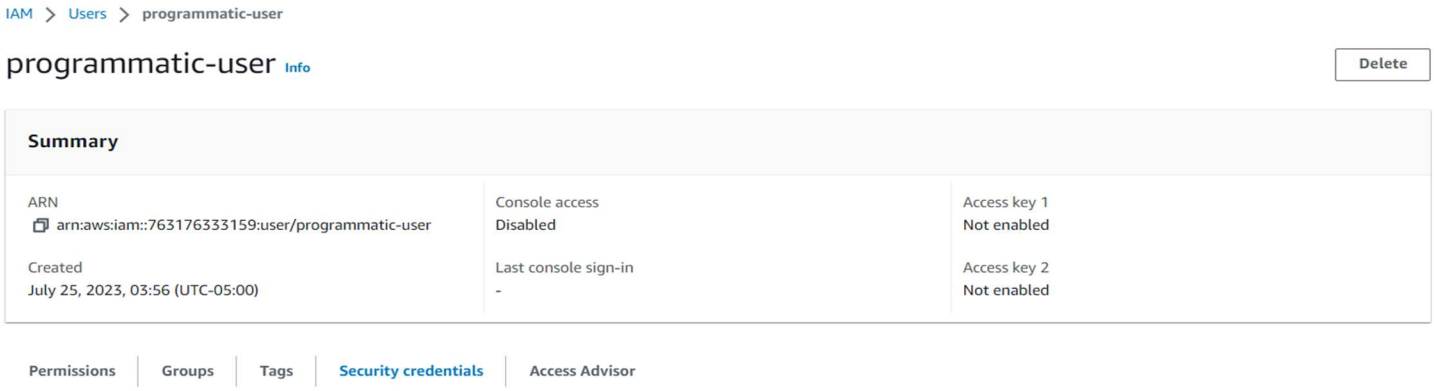



programmatic-user successfully created

Now let's click on programmatic-user

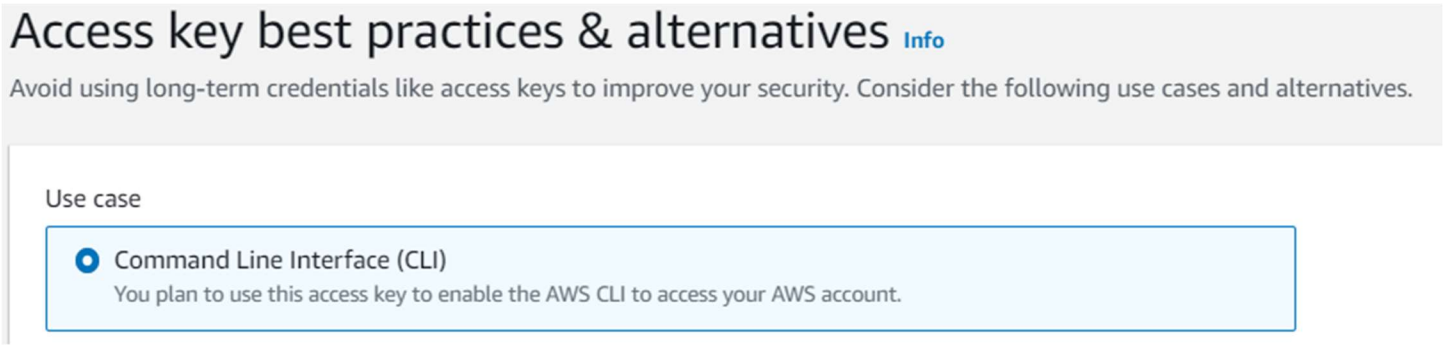
Select Security credentials



Scroll down the page

Select **“Create access key”**

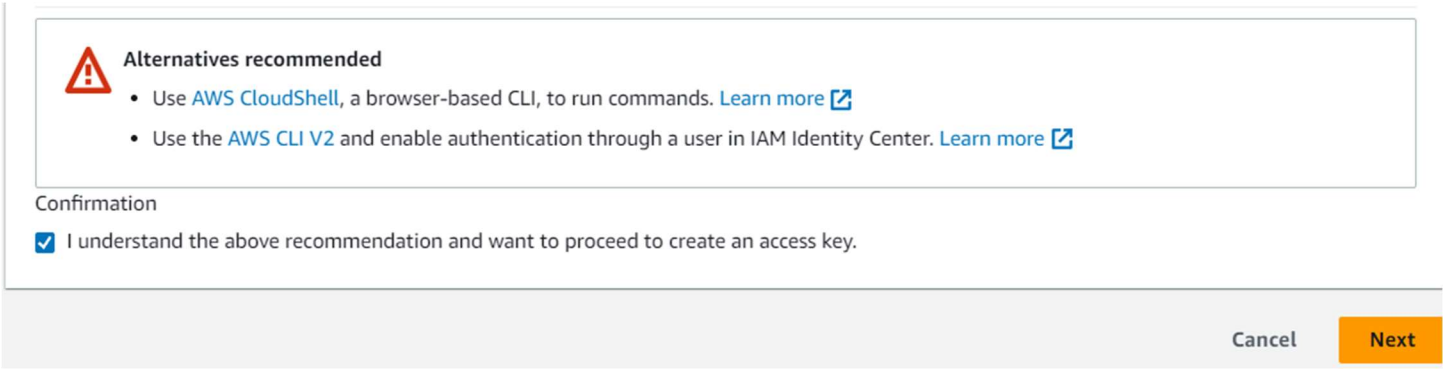

Select the command line interface (CLI)



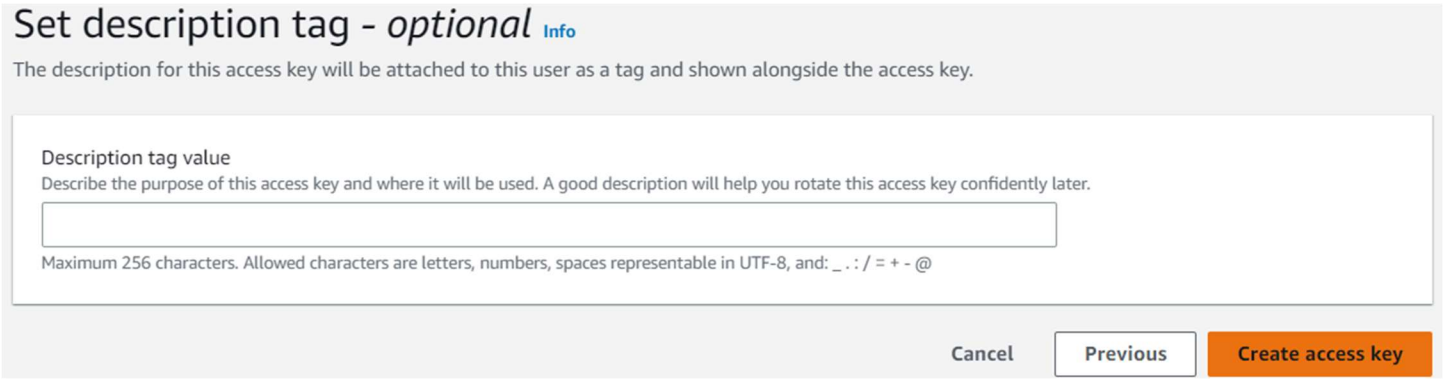

Then scroll down to the bottom of the page

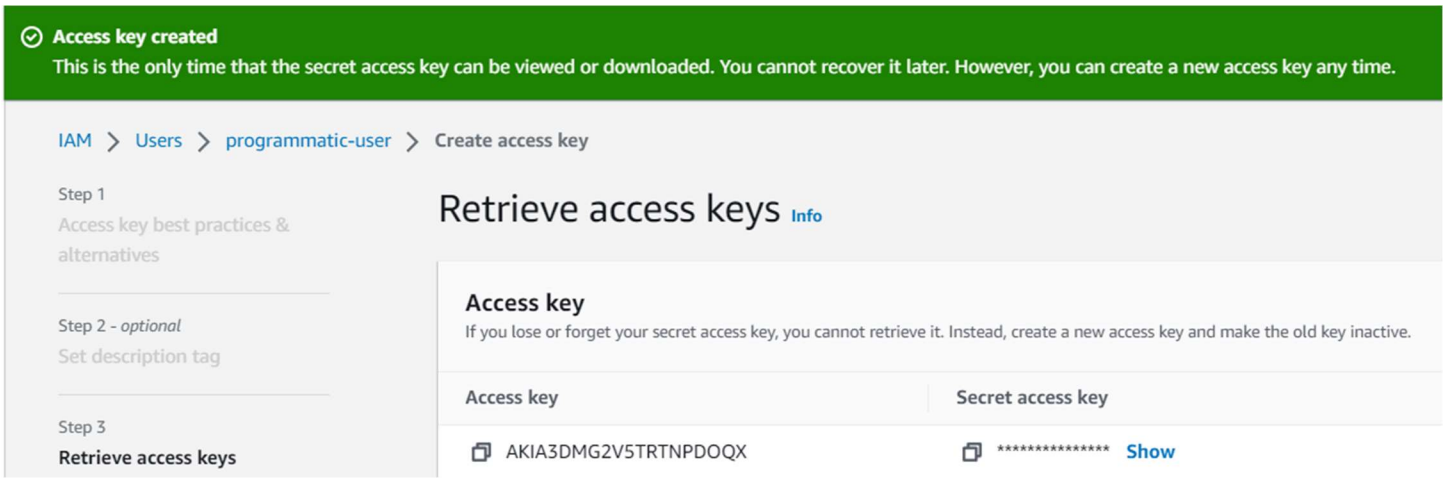
Check the confirmation checkbox

Click Next




Click on create access key





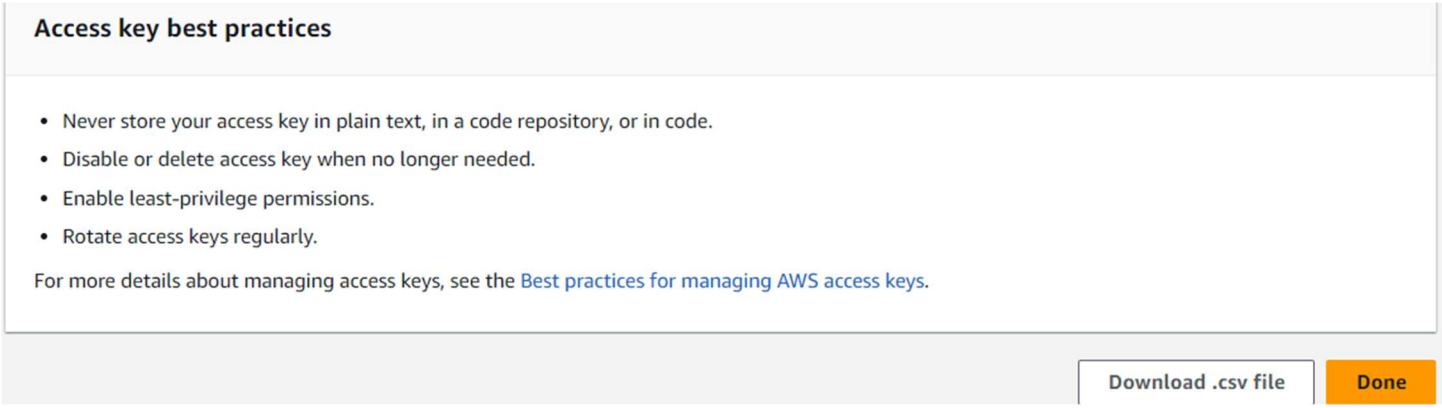

Access keys successfully created

Click on download .csv file

Navigate to the file explorer

Select the download folder to verify that the .csv file was successfully downloaded

Click **“Done”**




We successfully created the programmatic user and programmatic user access key

Next still will be to use the programmatic-user Access key ID and Secret Access key to authenticate into AWS, to push the container image to Elastic Container Registry (ECR)

## STEP 18: CREATE A PROFILE

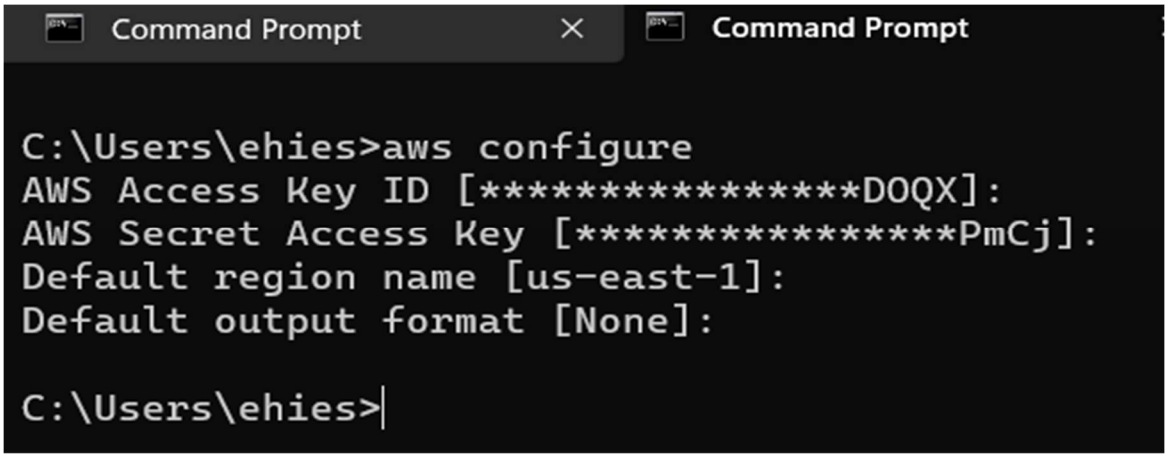
In the previous step, we created an IAM user with programmatic access.

The main task here will be to configure the user's Access key and Secret Access key on our computer.

Very important to know that configuring the user's security credentials on our computer will allow us to authenticate with our AWS account programmatically.

Let's begin by opening the command prompt

Run "**aws configure**" and press enter




You will be prompted to enter the

AWS Access Key ID [None]:********** Copy key and paste

Then press **Enter**

You will be again prompted to enter the

AWS Secret Access Key [None]:********** Copy key and paste

Then press Enter

You will be prompted a third time to enter the

Default region name [None]:********** I will enter us-east-1 and press Enter

Note that the AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) are self-sustaining, Region-based services that are available globally.

But you can also enter your Default region name, for example **us-east-1** if you desire to do so.

File Edit View

```
[default]
region = exit
output = exit
[profile programmatic-user]
region = Global
```

You will be finally prompted to enter the

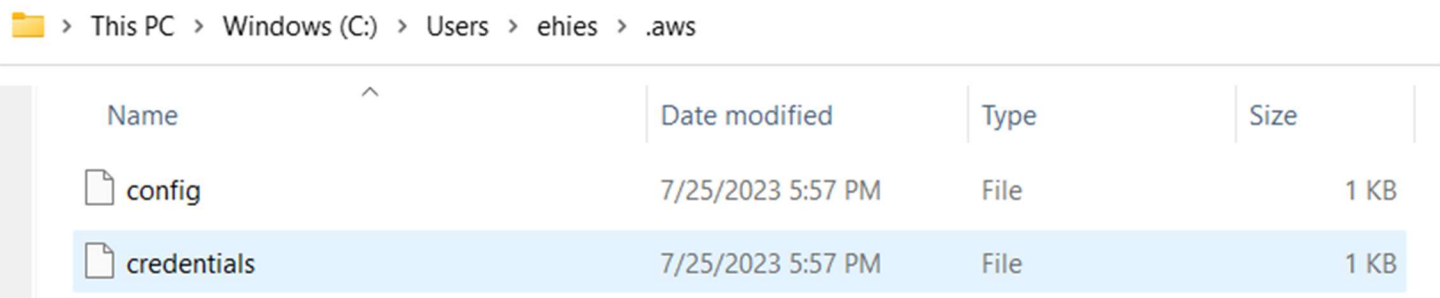
Default output format [None]:

Simply press Enter and proceed without entering anything

This is how we configure a user's credential on our computer, and we will use this user's credentials to authenticate to our AWS account programmatically.

Now let's verify that the programmatic-user's credentials were successfully stored on my computer, let's navigate to

**File Explorer > This PC > Windows (C:) > Users > ehies > .aws**




Double-click on config file and select open with Notepad to view user's default region (us-east-1)

Double-click on credentials file and select open with Notepad to view user's default credentials that shows the default aws_access_key_id and aws_secret_access_key

```
File    Edit    View

[default]
aws_access_key_id = exit
aws_secret_access_key = exit
[programmatic-user]
aws_access_key_id = AKIA3DMG2V5TRTNPDOQX
aws_secret_access_key = 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
```

Important to know that whenever you want to update the credentials, it can be done by running aws configure in the command prompt, then enter the:

aws_access_key_id and aws_secret_access_key or the config and credentials files can be updated directly, then click file and select save.

This is how you configure IAM Credentials on your computer to authenticate into your AWS account.

## STEP 19:

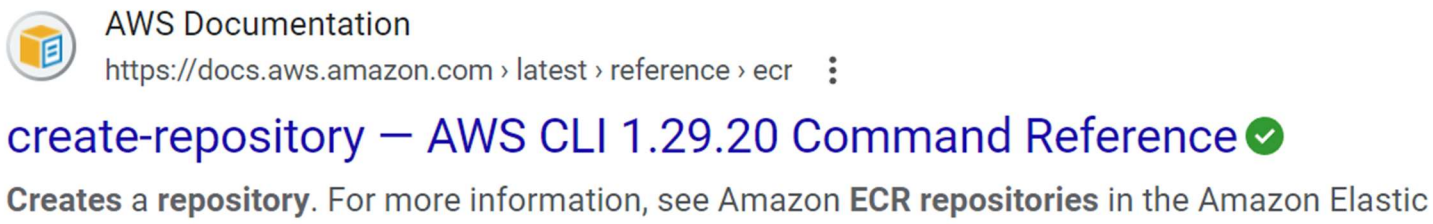
## CREATE AN AMAZON ECR REPOSITORY TO STORE CONTAINER IMAGES

The main task here will be to create an **ECR** repository to store our containers in **AWS**.

Note that before we can push our container images to AWS, first we must create a repository in Elastic Container Registry (**ECR**) to enable us push and store the container images, like what we did in docker hub.

To create a repository in ECR, we will use the AWS CLI (Command line interface)
For that, let's open google search box and enter "**AWS CLI** create **ECR** repository" press "**Enter**"

**Select the link with "https://docs.aws.amazon.com > latest > reference > ecr"**




Once the documentation page is open,

Scroll down to "Examples" and copy the "Example 1" command

Paste the command on a notepad to edit

Hit the back space bottom to remove the backward slash " \ " that breaks the command into 2 lines

The reason for this is to have the command in just a single line instead of two.

## Examples ¶

### Note:

To use the following examples, you must have the AWS CLI installed and configured. See the Getting started guide in the *AWS CLI User Guide* for more information.

Unless otherwise stated, all examples have unix-like quotation rules. These examples will need to be adapted to your terminal's quoting rules. See Using quotation marks with strings in the *AWS CLI User Guide* .

**Example 1: To create a repository**

The following `create-repository` example creates a repository inside the specified namespace in the default registry for an account.

```
aws ecr create-repository \
    --repository-name project-a/nginx-web-app
```

Now, let's edit the **--repository-name** and replace it with "**jupiter**"

**aws ecr create-repository --repository-name jupiter**

```
aws ecr create-repository --repository-name project-a/nginx-web-app
```

```
aws ecr create-repository --repository-name jupiter
```

Then, let's specify the region we are using for this project

**"aws ecr create-repository --repository-name jupiter --region us-east-1"**

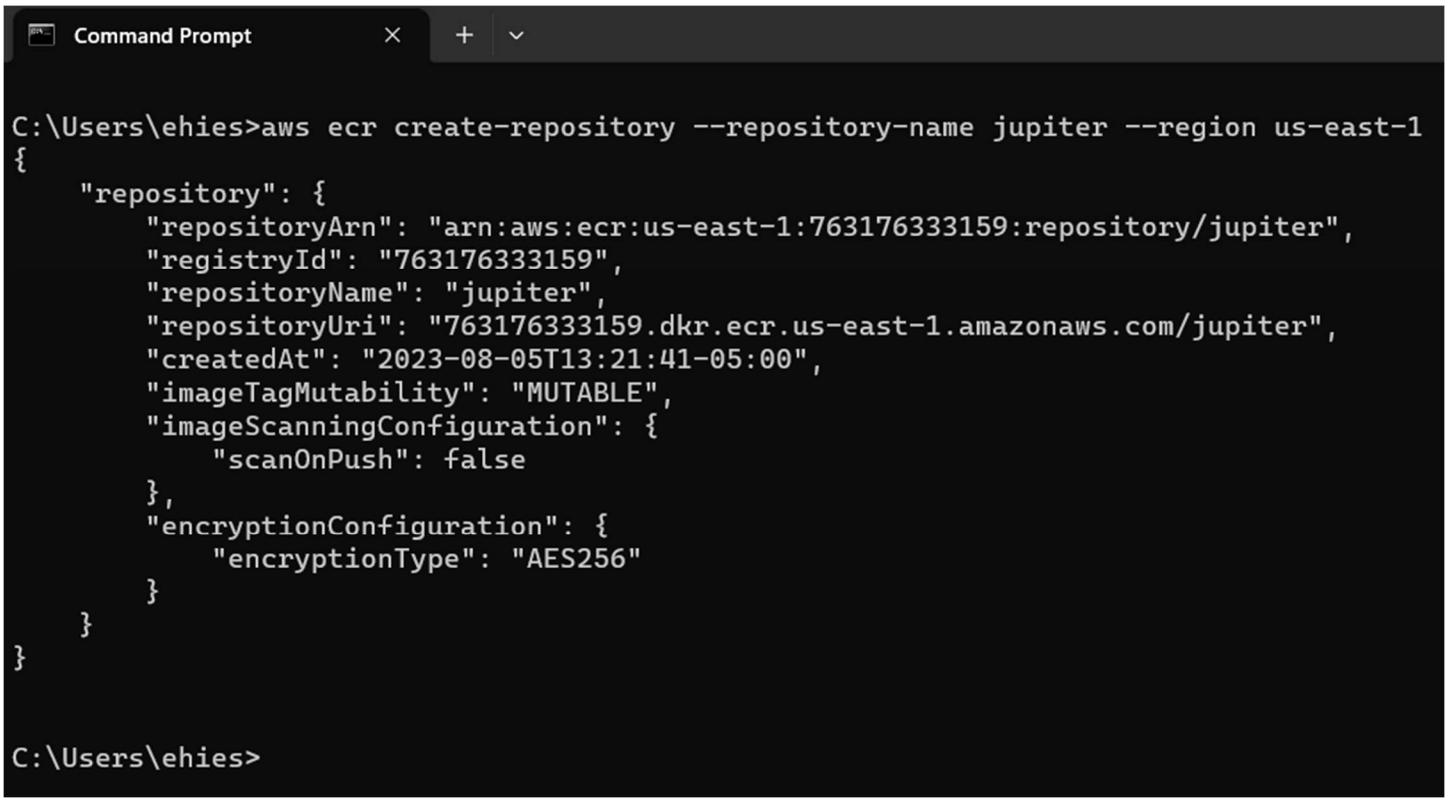
```
aws ecr create-repository --repository-name jupiter --region us-east-1
```


```
aws ecr create-repository --repository-name jupiter --region us-east-1
```


Let's copy the entire command,

Let's open the **command prompt "cmd"** on our machine

Paste and run the command

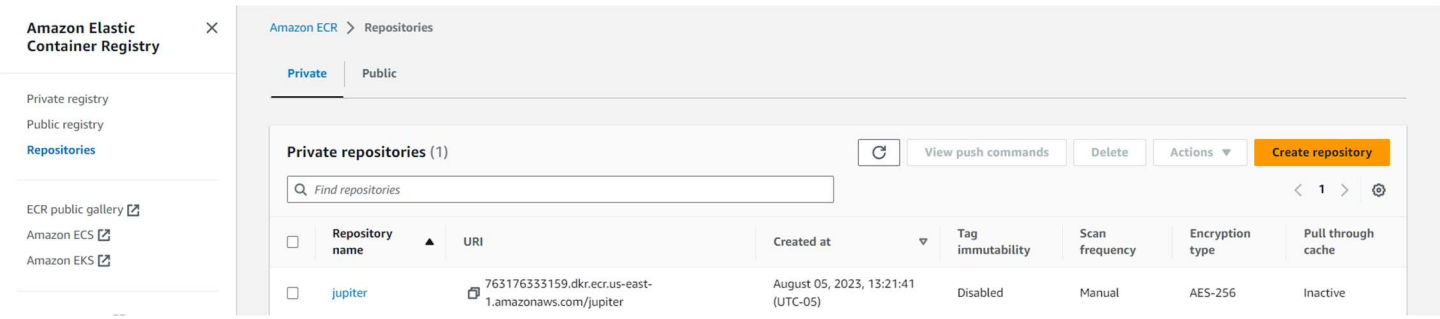



Now, let's copy and paste the output of the command below

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:763176333159:repository/jupiter",
        "registryId": "763176333159",
        "repositoryName": "jupiter",
        "repositoryUri": "763176333159.dkr.ecr.us-east-1.amazonaws.com/jupiter",
        "createdAt": "2023-08-05T13:21:41-05:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": false
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```


Now let's go to our **AWS management console** to verify if our repository is in **ECR**

Login to console and navigate to **Elastic Container Registry** to verify if the "**jupiter**" repository was successfully created in **ECR**




The repository was successfully created in ECR

## STEP 20: PUSH THE CONTAINER IMAGE TO THE YOUR ECR REPOSITORY

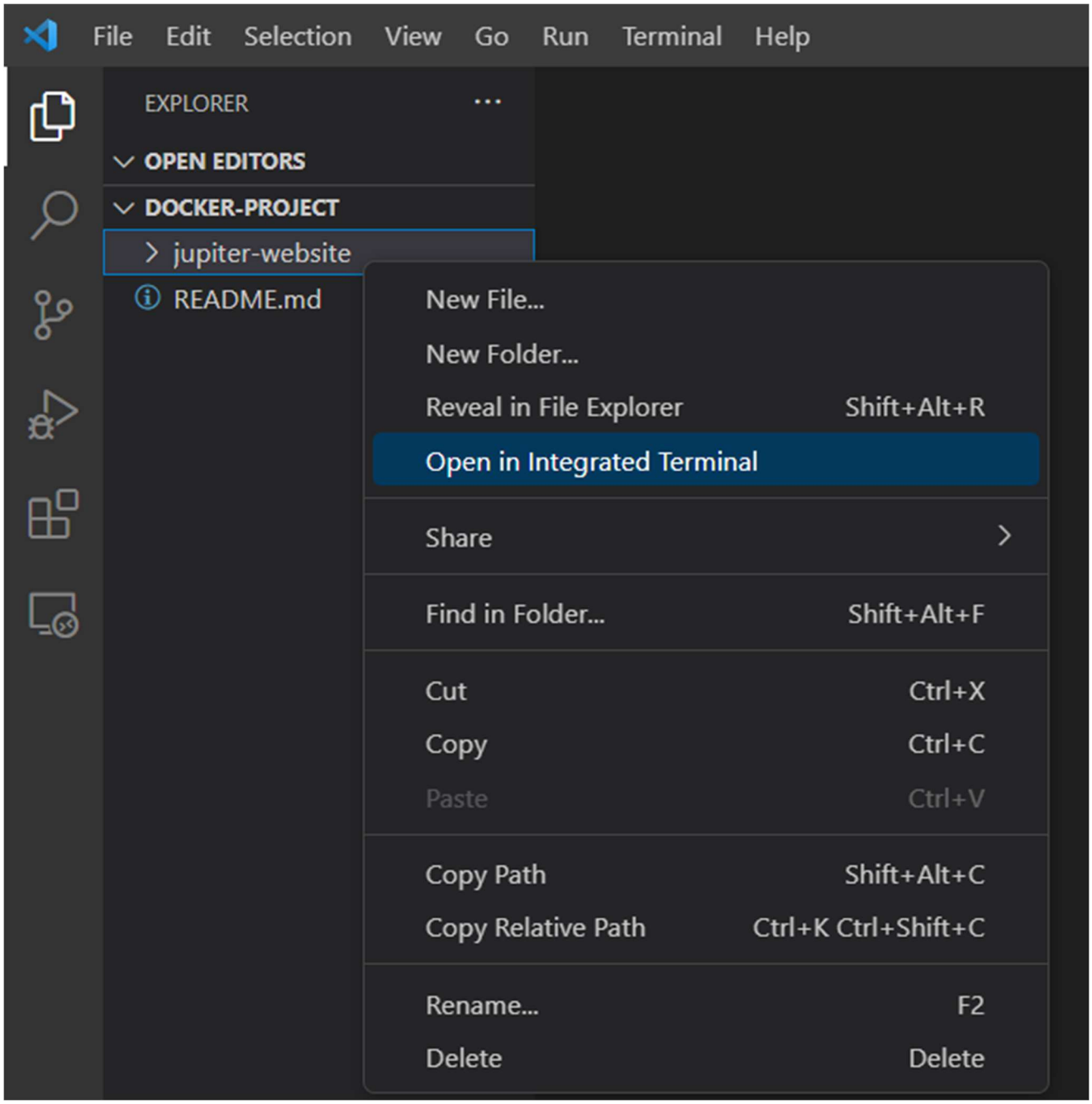
First let's launch **Visual Studio Code**

Second, let's open our **Docker-Project** folder in **Visual Studio Code**

Place the mouse pointer on the **jupiter-website**,

Right click and select open in **integrated terminal**

Run "**docker image ls**" to list the images we currently have in our docker repository




Let’s quickly run the **“docker image ls”** to list the docker images we currently have

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> **docker image ls**

The first image is the image that we renamed that we pushed to **docker hub**

The second image is the one that we originally created

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
ehikioya911/jupiter     latest    dd84a94560f4   2 weeks ago     266MB
jupiter                 latest    dd84a94560f4   2 weeks ago     266MB
2048-game               latest    9c441596990f   2 weeks ago     182MB
hello-world             latest    9c7a54a9a43c   3 months ago    13.3kB
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
```

To push our image to the **ECR repository**, the first thing we should do will be to tag the image

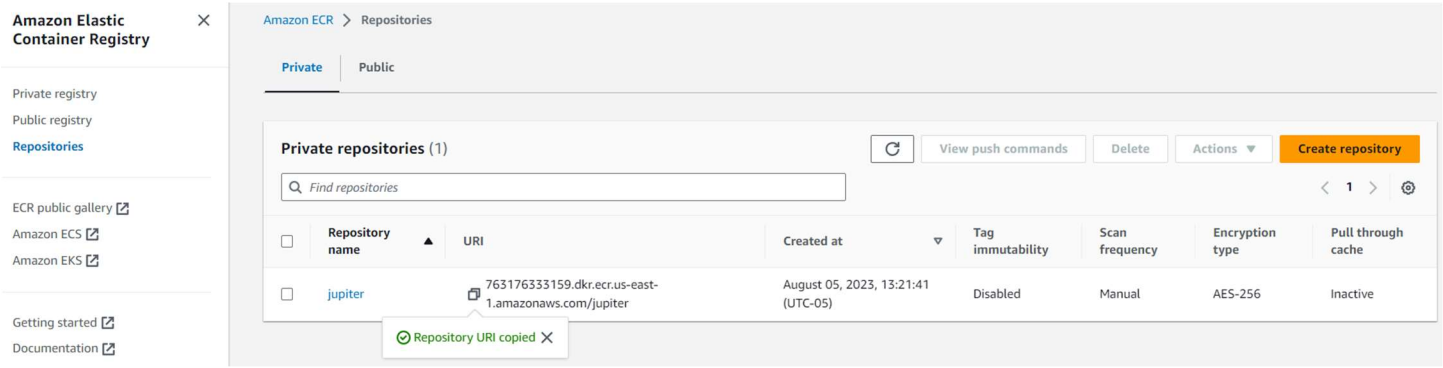
To tag the image, we need to run "**docker tag**" space and name of image the image you want to tag

I will enter "**jupiter**" since that's the name of the image we want to tag, then press space

Now, we need to enter the **"URI"** of the repository that we created

**Navigate to the management console,**

Click on repository and copy the **"URI"**




Paste the copied **URI** after the repository name

Then press "**Enter**" to run

```
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker tag jupiter 763176333159.dkr.ecr.us-east-1.amazonaws.com/jupiter
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
```

We have successfully tagged the image.

**To list the docker images, enter "docker image ls"**


We can see the image we tagged is present in the output.

To enable us to push the **jupiter** container image to **ECR**, like what we did to docker hub, we must sign in to **ECR**.

The command to sign in to **ECR** is given below:

**aws ecr get-login-password | docker login --username AWS --password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com**

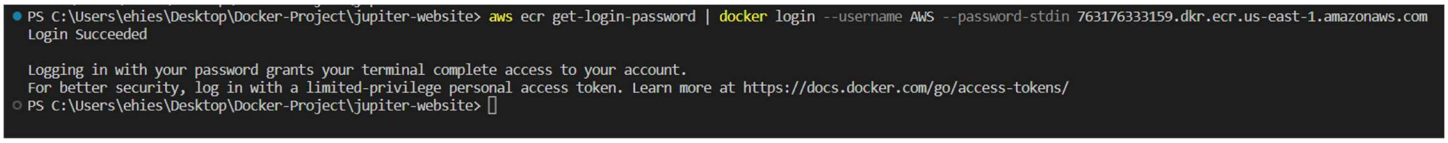
**IMPORTANT:**

We must edit the above command by updating our "**aws_account_id**" and "**region**"

The **acconut_id** and **region** information can be found in the displayed output when we **ran** the "**docker image ls**" command in our previous step.

Now, let's edit and copy the entire command and run it in our terminal.

**aws ecr get-login-password | docker login --username AWS --password-stdin 763176333159.dkr.ecr.us-east-1.amazonaws.com**




Login to ECR was Succeeded and now we can push our jupiter container image to the repository we initially created in ECR.

The next step will be to push our jupiter image to the **ECR** repository by following the steps below:

Run the "**docker push**" command plus the **URI** of our **ECR** repository

Login to AWS account to extract the **URI,** navigate to ECR, then copy the URI of your repository

Let's open Visual Studio Code

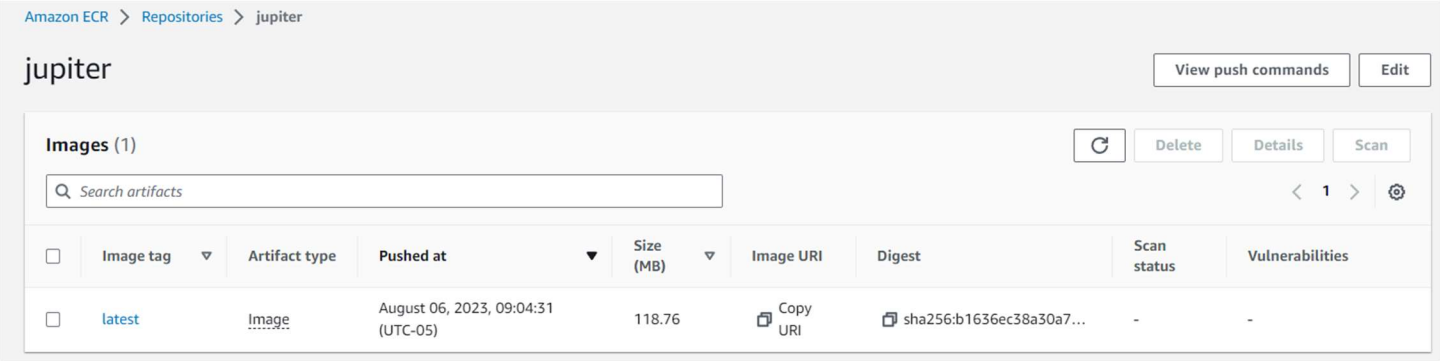
**run "docker push 763176333159.dkr.ecr.us-east-1.amazonaws.com/jupiter"**

```
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website> docker push 763176333159.dkr.ecr.us-east-1.amazonaws.com/jupiter
Using default tag: latest
The push refers to repository [763176333159.dkr.ecr.us-east-1.amazonaws.com/jupiter]
eb776baf2926: Pushed
160e767444b4: Pushed
16cf6dafdab5: Pushed
d957fd84e554: Pushed
a431e7961bc9: Pushed
b96fdda2a387: Pushed
ccaa10561e98: Pushed
latest: digest: sha256:b1636ec38a30a780e0d9b2f609d1756f98c6d9341fbd5df4ba99c79013d7897d size: 1787
PS C:\Users\ehies\Desktop\Docker-Project\jupiter-website>
```

The jupiter container image was successfully pushed to our ECR repository

Now let's go to our AWS account to verify that the jupiter container image is in our ECR repository

In the **ECR** repository, hit on refresh and click on the "**jupiter**" to view container image.




The jupiter image was successfully pushed to the ECR repository as we can see in the above screenshot

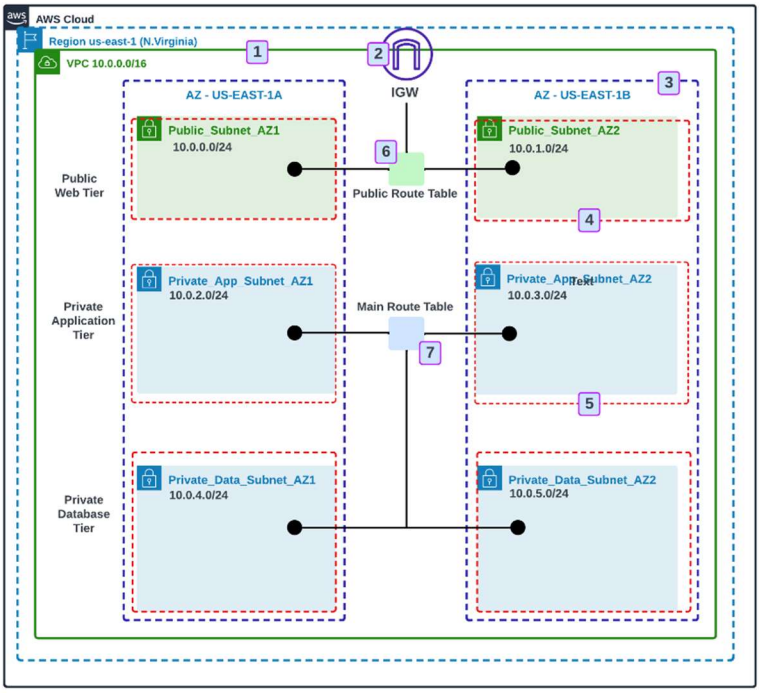
The **tag** for the Image is **latest.**

This is how we can successfully push an image that we created in our local machine to the Amazon

Elastic Container Registry (**ECR**)

This jupiter container image will be used to create ECS containers that will deploy our jupiter website.

# STEP 21: BUILD A 3-TIER AWS NETWORK VPC FROM SCRATCH




1. VPC with public and private subnets in 2 availability zones
2. An internet gateway is used to allow communication between instances in the VPC and the internet
3. We are using 2 Availability zones for high availability and fault tolerance
4. Resources such as NAT gateways, Bastion Hosts and Load balancers are hosted in the public subnets
5. We will deploy the web servers and database in the private subnets to secure them from direct direct internet traffic
6. The Public Route Table is associated with the public subnets and routes traffic to the internet through the internet gateway
7. The Main Route Table is associated with the private subnet

A 3-Tier architecture is made up of 3 separate tiers or layers, which are the Presentation or Web layer, The Application layer, and Database layer.

On the first tier (**tier 1**) is where we have the public subnets, and the public subnets will host resources such as Nat Gateways, Load Balancers and Bastion Hosts

On the second tier (**tier 2**) is where we have the private subnets, and the private subnets will host our web servers (EC2 instances)

On the third tier (**tier 3**) we have another private subnet that will host our database.

We will create duplicate tier 1, tier 2, and tier 3 subnets across two or more availability zones (AZ) for **high availability** and **fault tolerance** architecture.

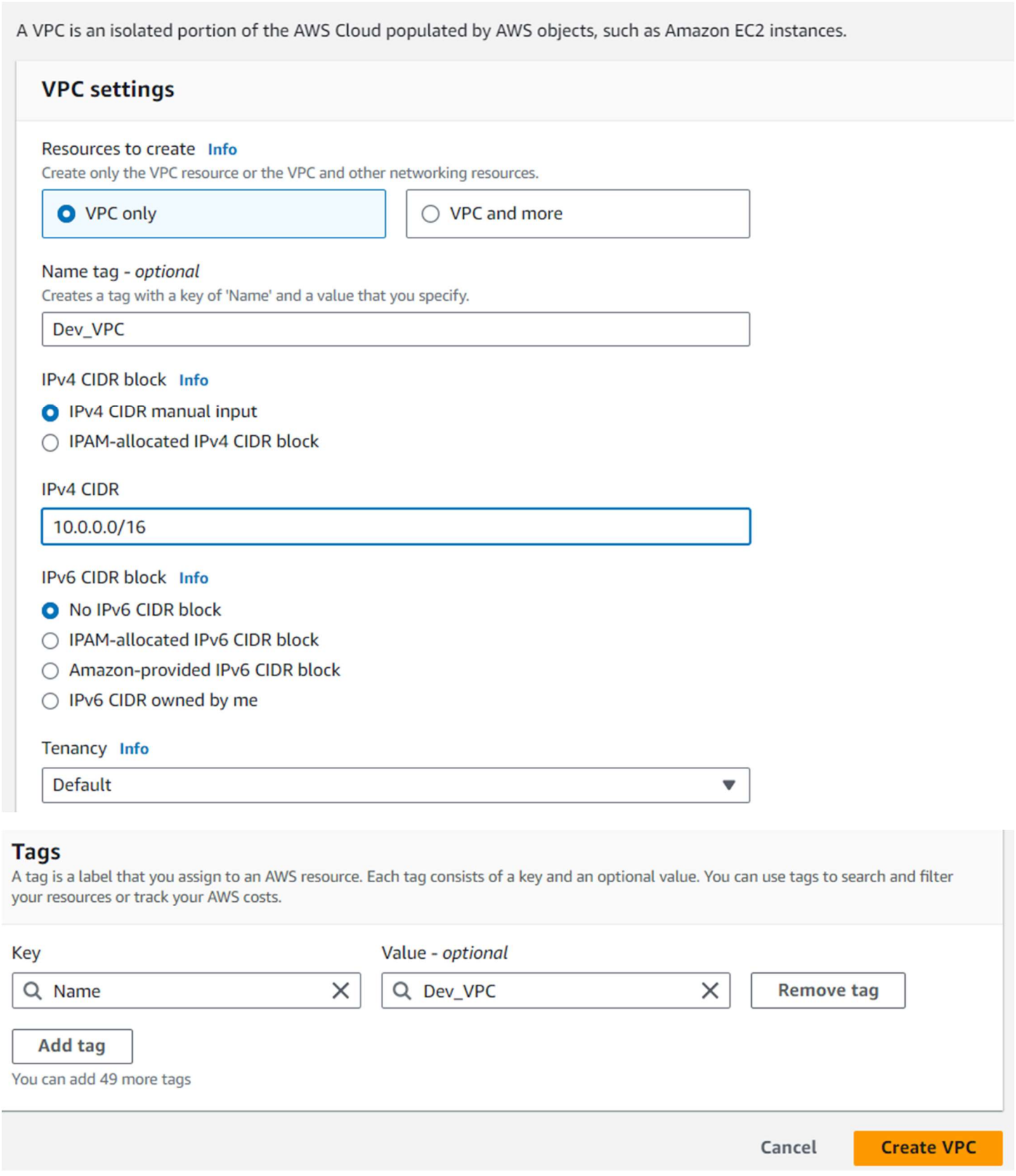
Then, we will create an Internet gateway (**IGW**) and NAT gateway (**NGW**) so that the resources hosted in our private subnets will have access to the internet.

### VPC CREATION

Let's navigate to the AWS management console to create our VPC.

We will create our VPC in the us-east-1 (**N. Virginia**) Region

We must also specify the range of IPv4 CIDR block address for the VPC, and my specified CIDR range will be **10.0.0.0/16**




**Dev_VPC** successfully created as we can see from the images above.

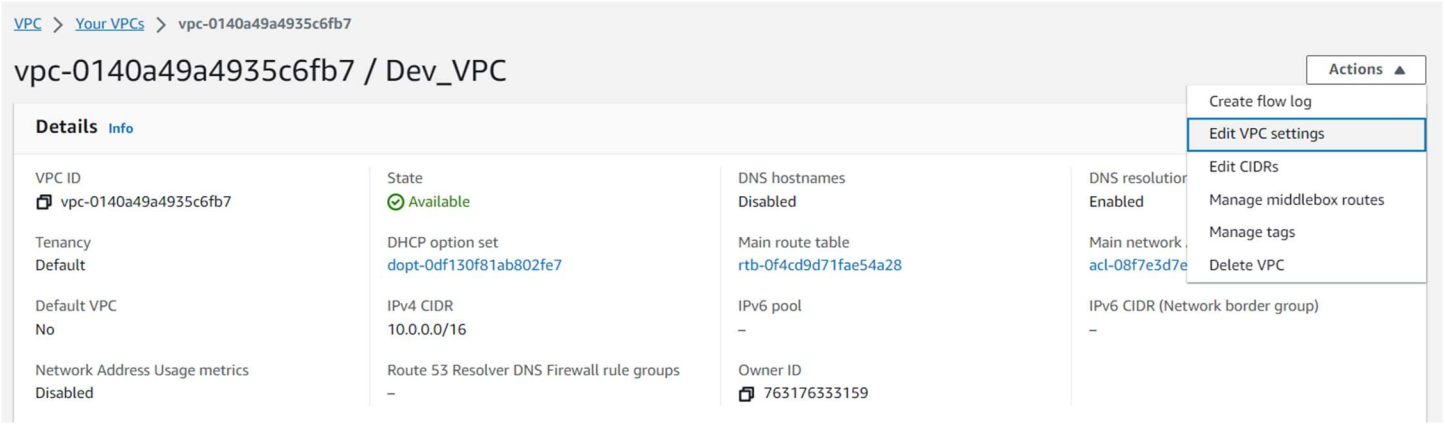
Now, let's proceed to enable the **DNS hostname** of our newly created **Dev_VPC**

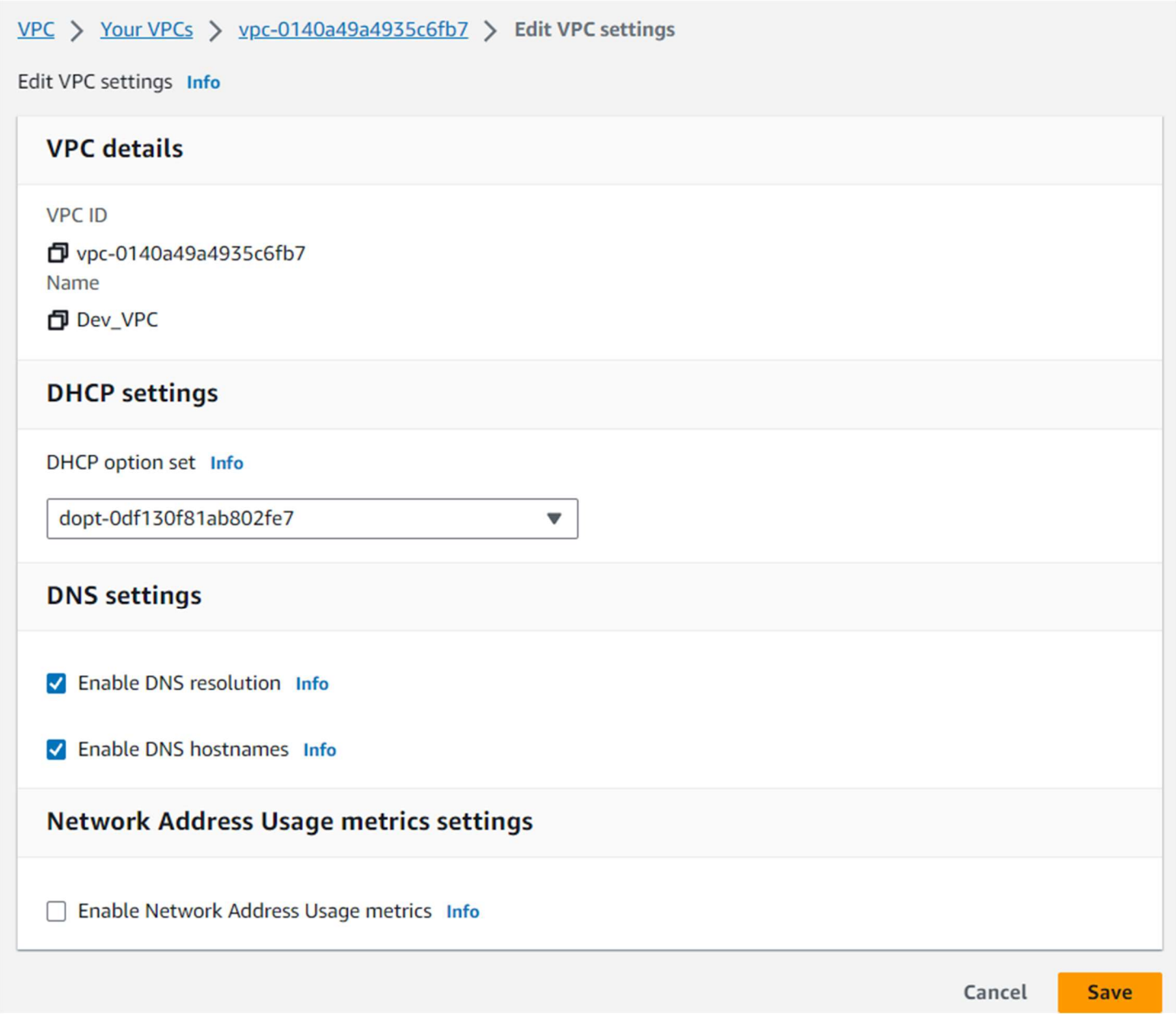
Select the **Dev_VPC**

Navigate to the top right-hand corner

Select the "Actions" drop down menu

Select edit **VPC setting >** Select enable **DNS hostnames** > **SAVE** settings



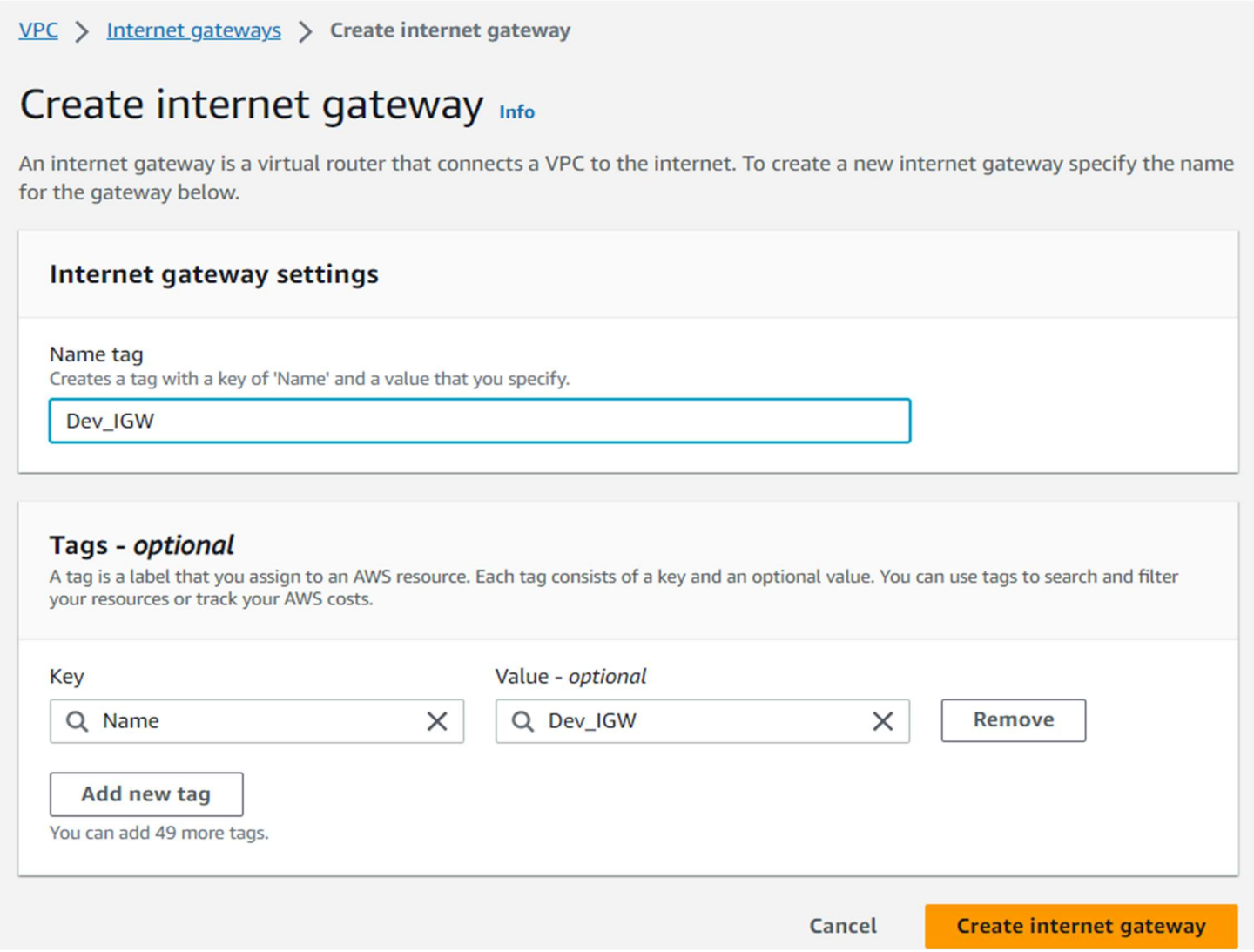



**INTERNET GATEWAY (IGW) CREATION**

Select Internet **Gateway**

Let's name it **Dev_IGW**

Select **Create internet gateway**




**ATTACH INTERNET GATEWAY TO VPC**

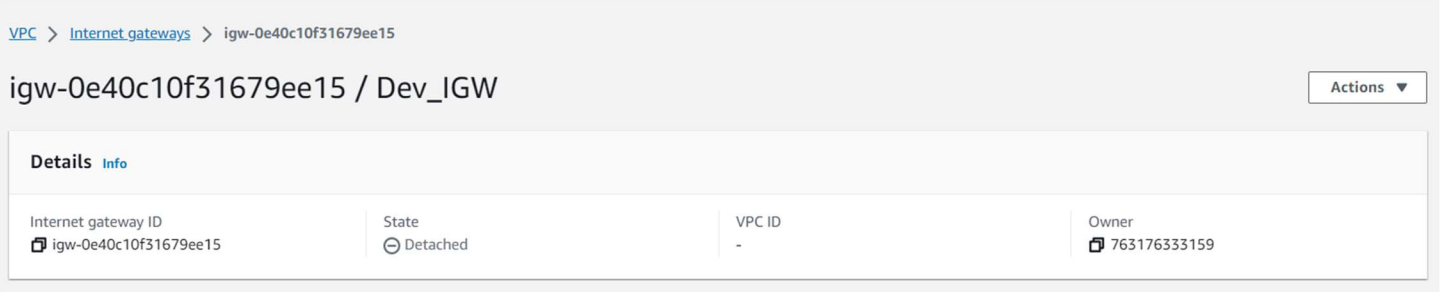
Next, we will attach the Internet gateway **(IGW)** to the **Dev_VPC**

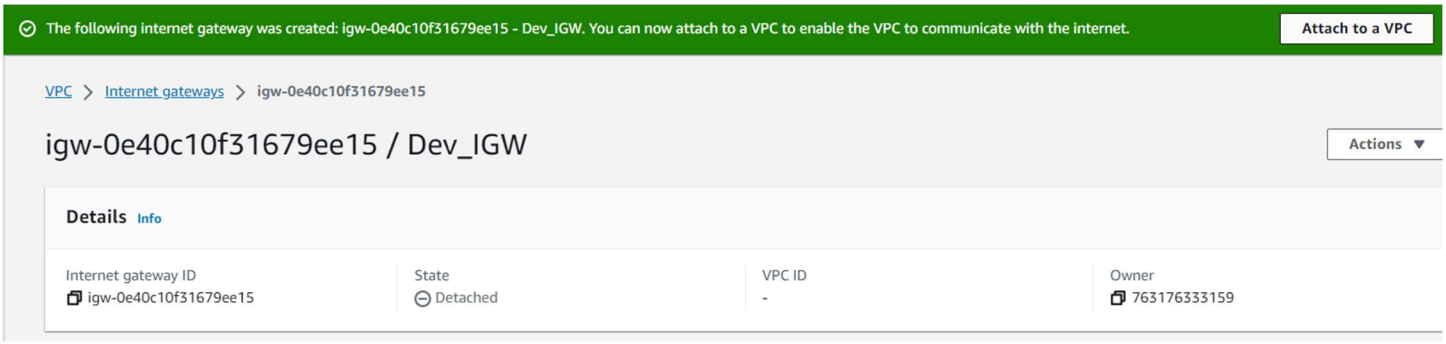
Navigate to the right-hand corner of the screen and click on **Attach to a VPC**

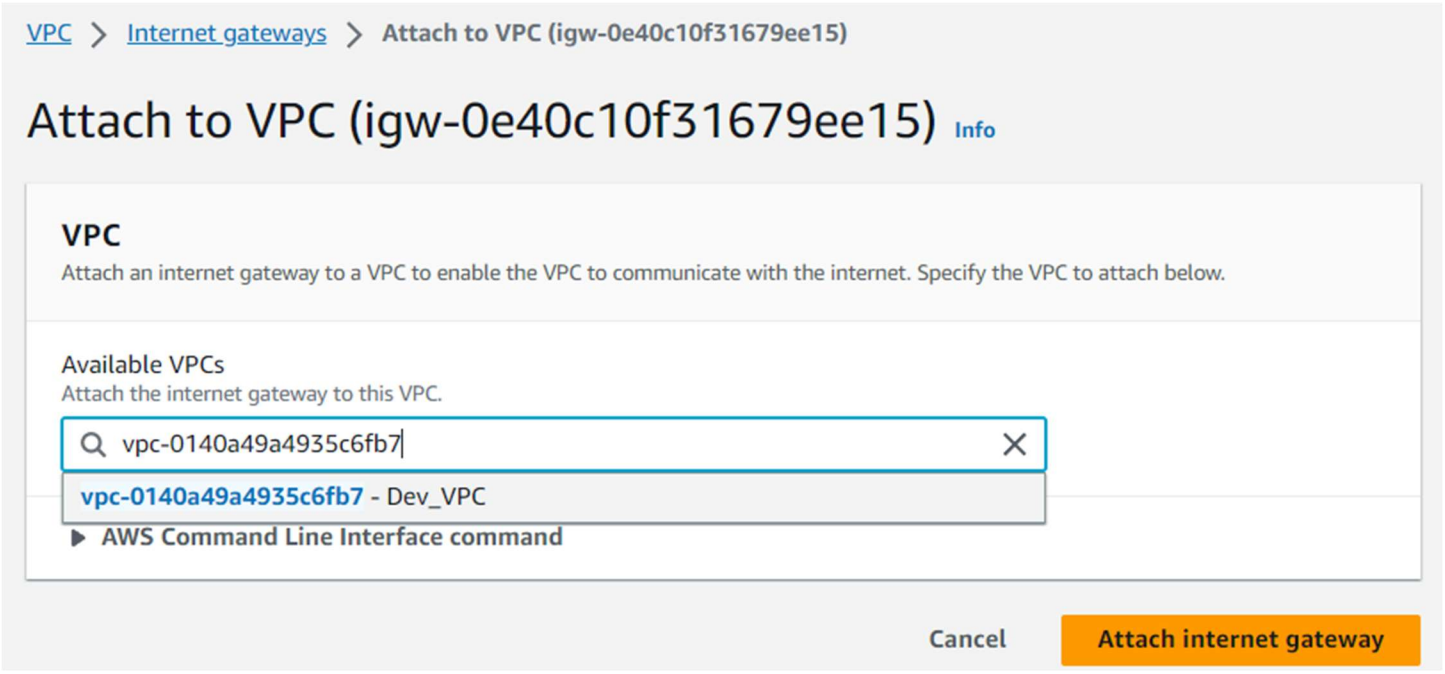
Click on the space to select **Dev_VPC**

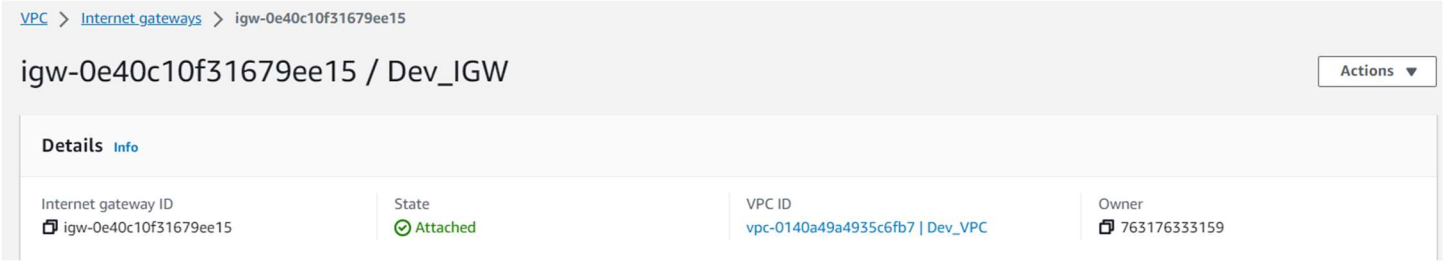
Click on **Attach internet gateway**

**IMPORTANT:** You can only attach one internet gateway to a VPC.






We have successfully attached our **Dev_IGW** to **Dev_VPC**

**CREATION OF 2 PUBLIC SUBNETS**

The next step will be to create 2 public subnets in 2 different availability zones AZs for high availability.

In case an AZ goes down, all traffic will be routed to the second AZ as a disaster recovery solution.

First step here will be to select "**Subnets**" on the left side of the screen

Select the VPC dropdown to select the **Dev_VPC**

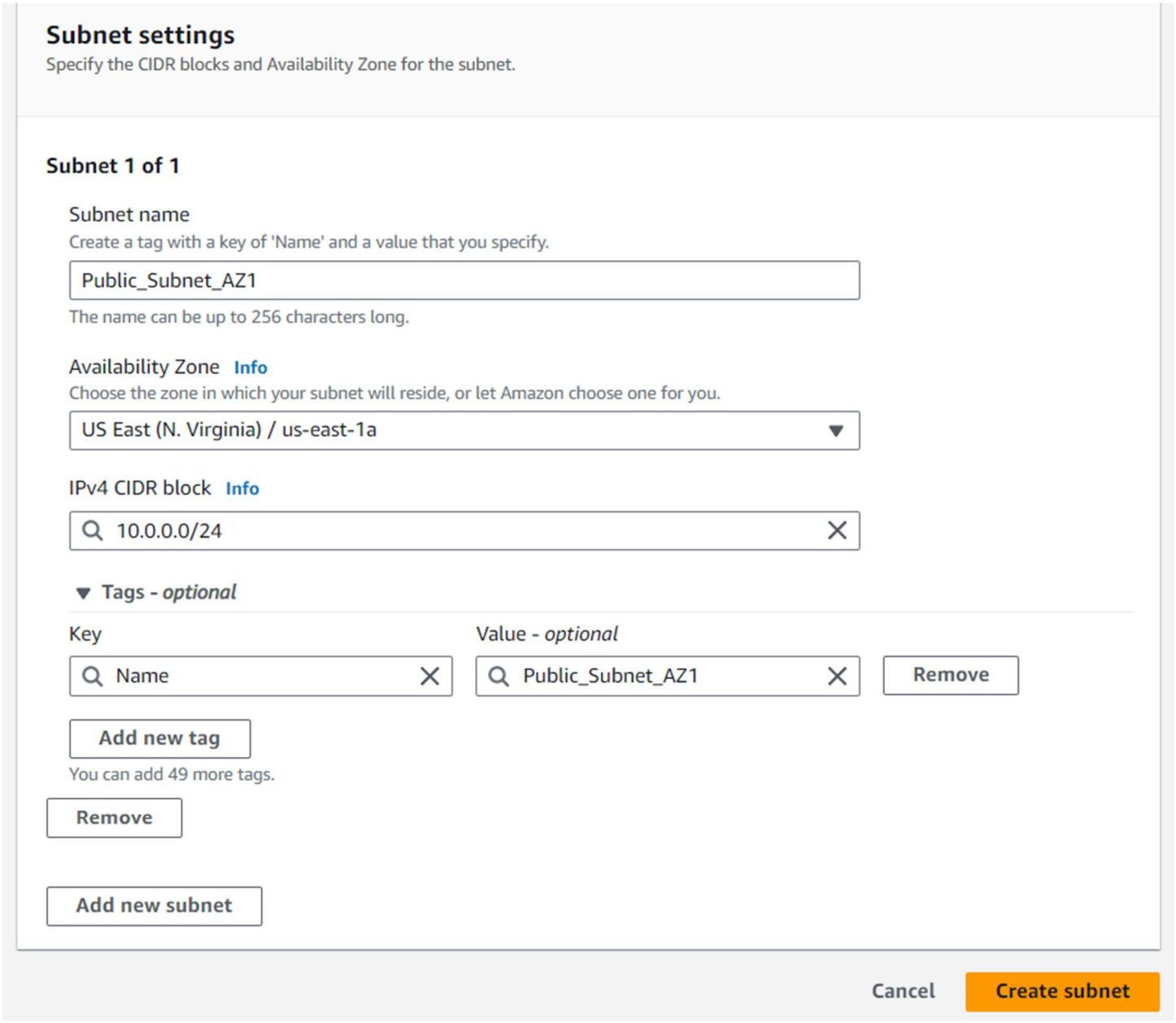
Let's name the Subnet as **Public_Subnet_AZ1**

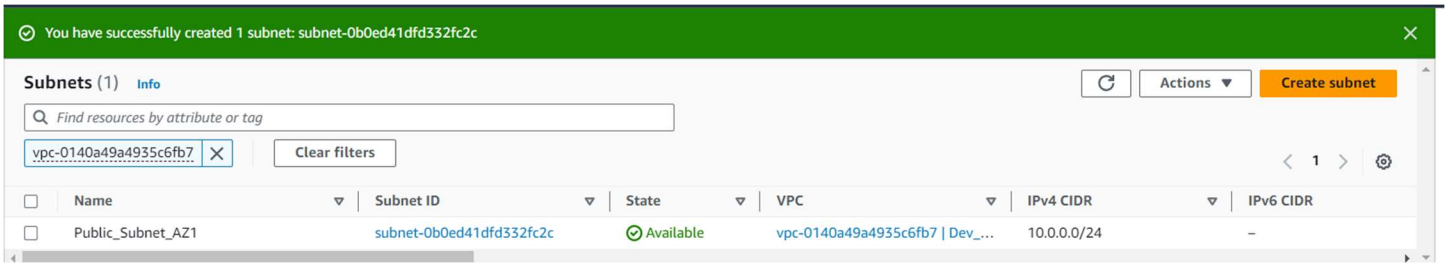
Let's select the availability zone AZ as **us-east-1a**

The IPV4 CIDR address block will be **10.0.0.0/24**

Click on **create subnet**

**Public_Subnet_AZ1** successfully created as we can see from the screenshot below




**Let's create another Public Subnet Public_subnet_AZ2**

Next step here will be to select **Create subnets**

Select the VPC dropdown to again select the **Dev_VPC**

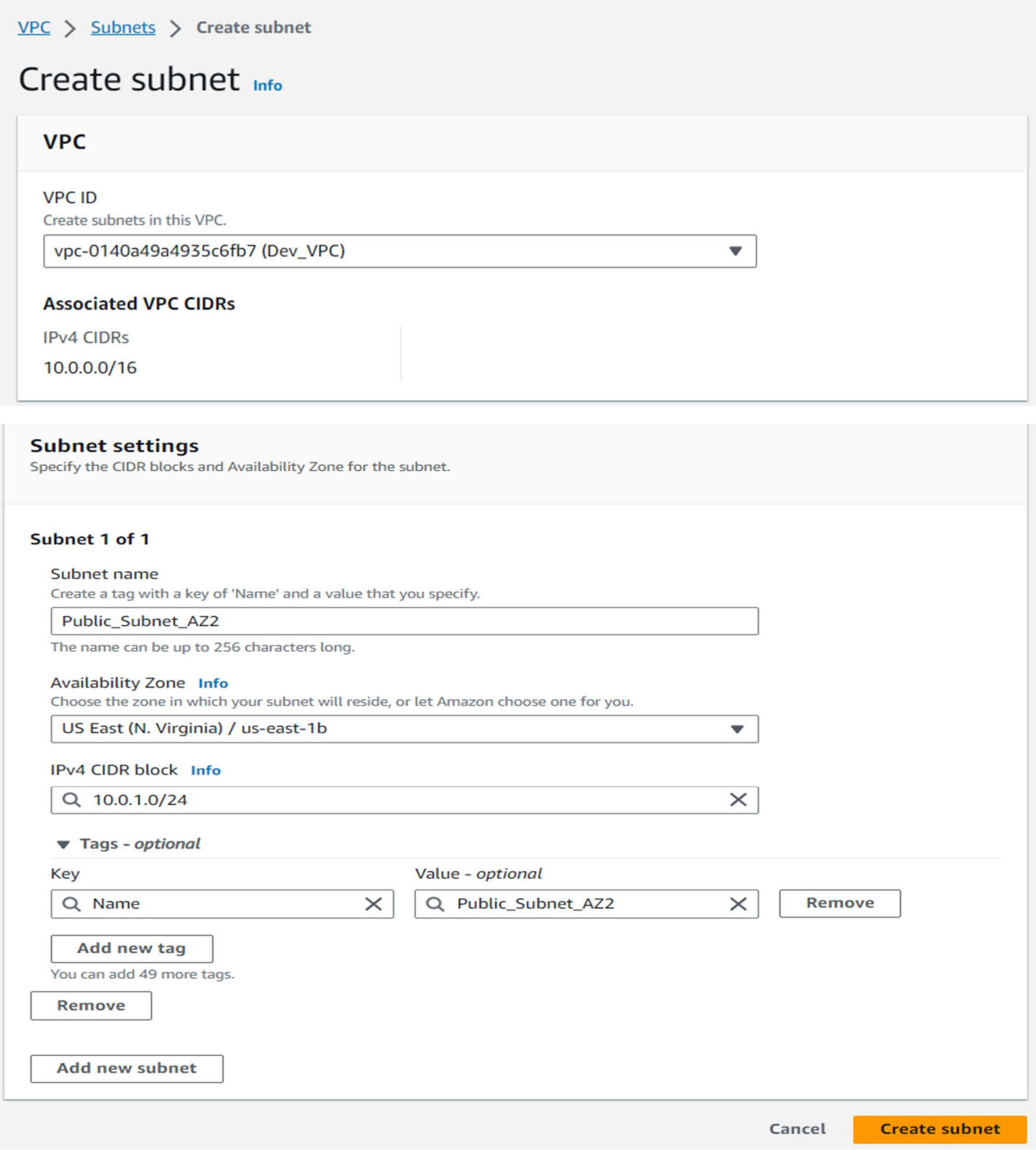
Let's name the Subnet as **Public_Subnet_AZ2**

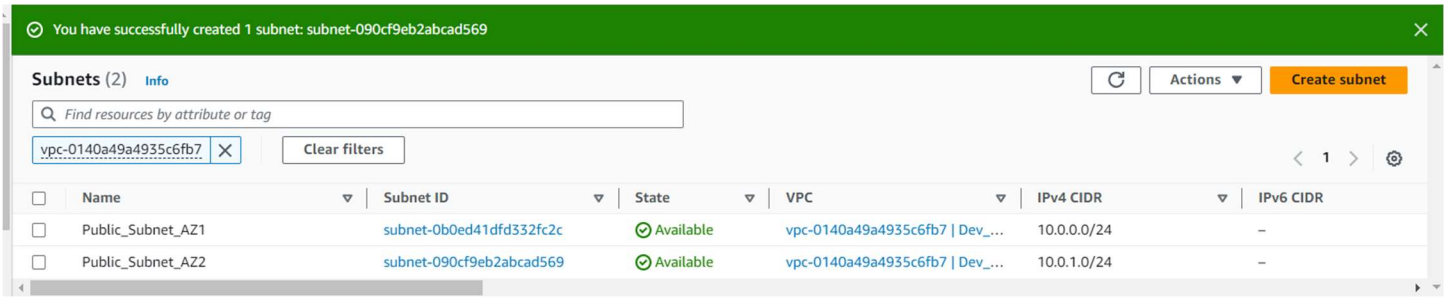
Let's select the availability zone AZ as **us-east-1b**

The IPV4 CIDR address block will be **10.0.1.0/24**

Click on **Create subnet**

**Public_Subnet_AZ2** successfully created as we can see from the screenshot below






## CONFIGURE AUTO ASSIGN IPV4 ADDRESS

My next task will be to configure **"Auto Assign IP address"** to both subnets

The importance of configuring "Auto Assign IP address" is that whenever an EC2 instance is created in these subnets, it automatically assigns public IPs to the created EC2 instances.

Click on the Public subnet "**Public_Subnet_AZ1**" to expand it

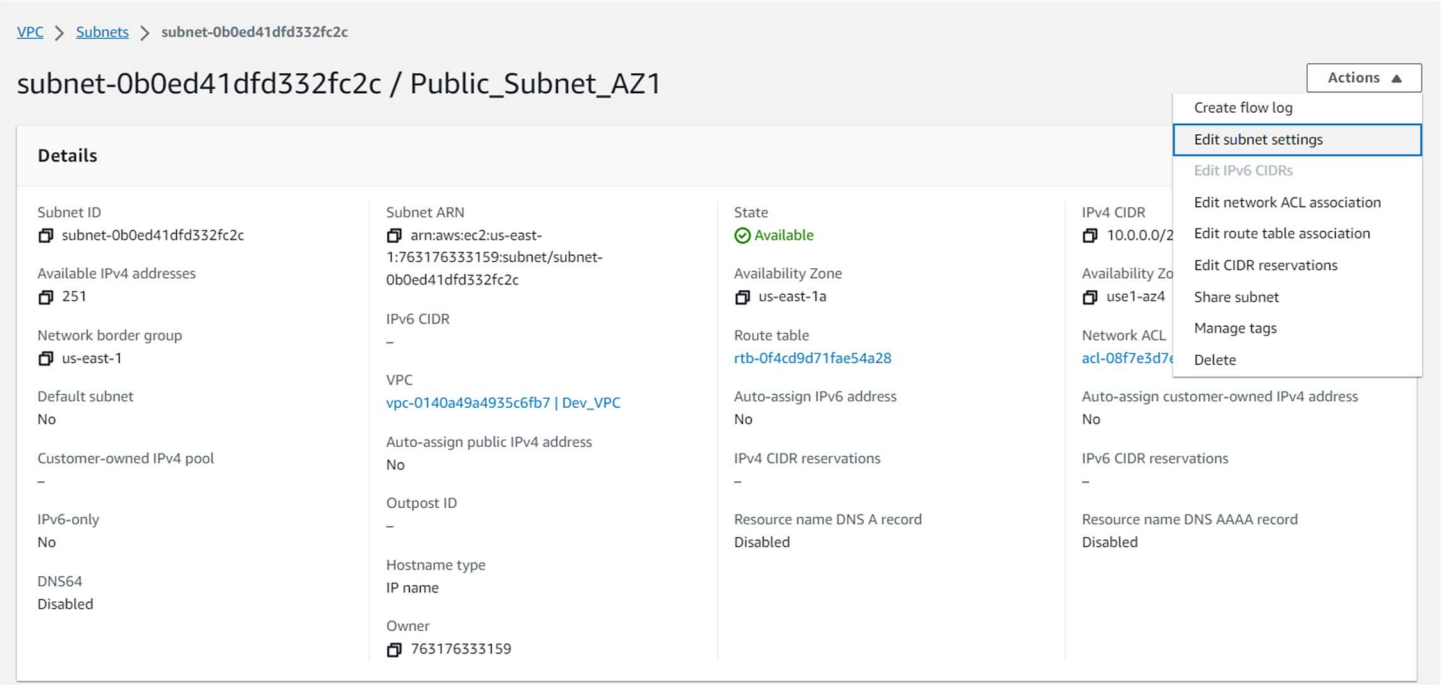
Click on "**Actions**" dropdown on the top right corner of the screen

Select "**Edit subnet settings**"

Select "**Enable auto-assign public IPv4 address**"

Scroll down to the bottom of the page and click "**Save**"

We have successfully configured "**Enable Auto Assign IPV4 address**" for the "**Public_Subnet_AZ1**"



# Edit subnet settings Info

## Subnet

| Subnet ID | Name |
| --- | --- |
| subnet-0b0ed41dfd332fc2c | Public_Subnet_AZ1 |

## Auto-assign IP settings Info

Enable the auto-assign IP settings to automatically request a public IPv4 or IPv6 address for a new network interface in this subnet.

- [x] Enable auto-assign public IPv4 address Info
- [ ] Enable auto-assign customer-owned IPv4 address Info
  Option disabled because no customer owned pools found.

## Resource-based name (RBN) settings Info

Specify the hostname type for EC2 instances in this subnet and optional RBN DNS query settings.

- [ ] Enable resource name DNS A record on launch Info
- [ ] Enable resource name DNS AAAA record on launch Info

Hostname type Info

- ( ) Resource name
- (•) IP name

## DNS64 settings

Enable DNS64 to allow IPv6-only services in Amazon VPC to communicate with IPv4-only services and networks.

- [ ] Enable DNS64 Info

Cancel Save

**You have successfully changed subnet settings:**

- Enable auto-assign public IPv4 address

**Let's repeat the same process to configure "Enable Auto Assign IPV4 address" for the "Public_Subnet_AZ2"**

Click on the Public subnet "**Public_Subnet_AZ2**" to expand it

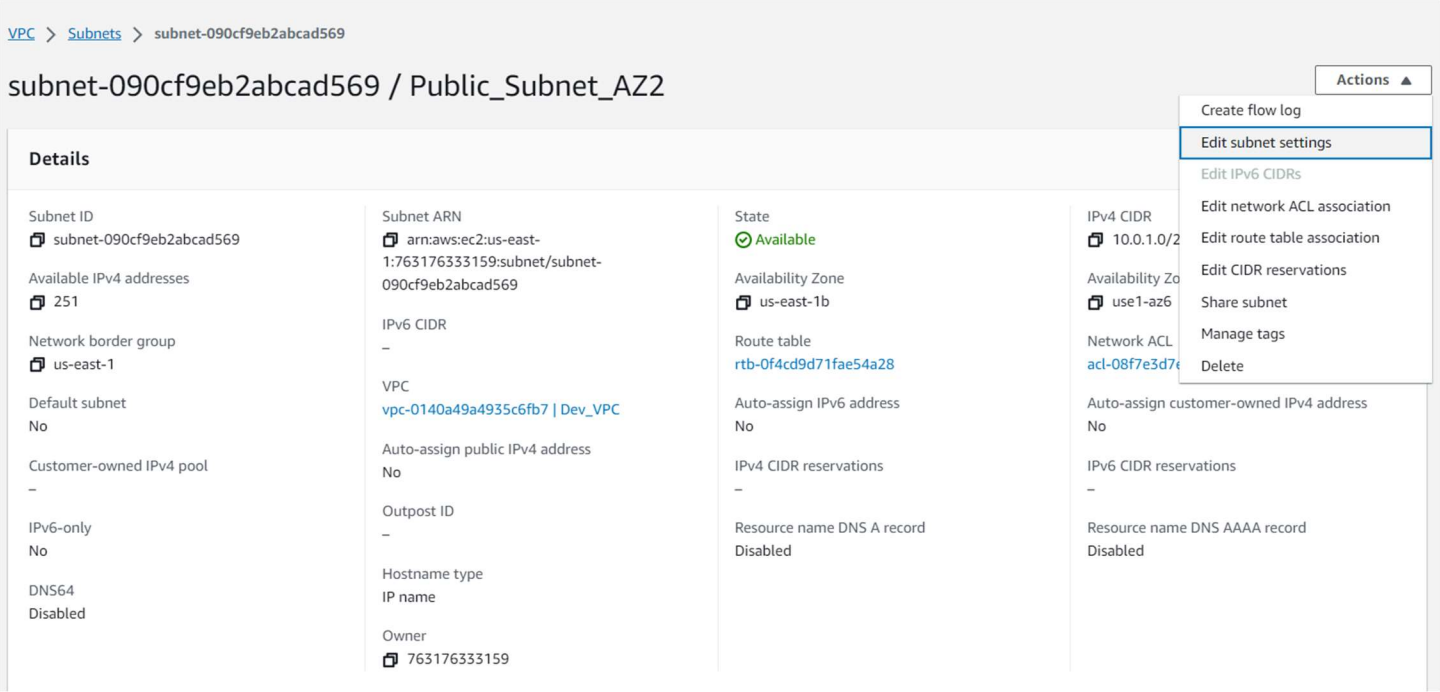
Click on "**Actions**" dropdown on the top right corner of the screen

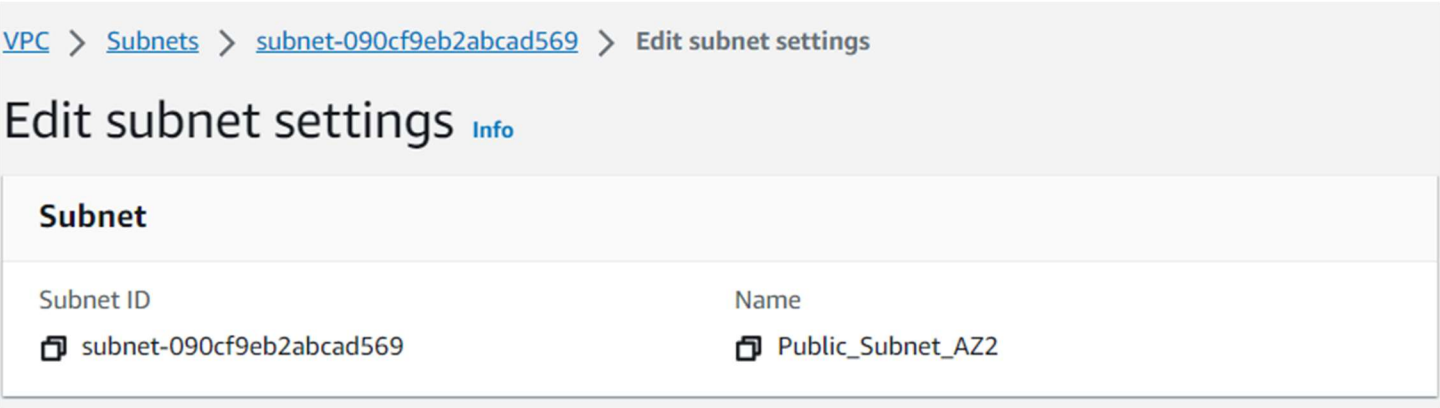
Select "**Edit subnet settings**"

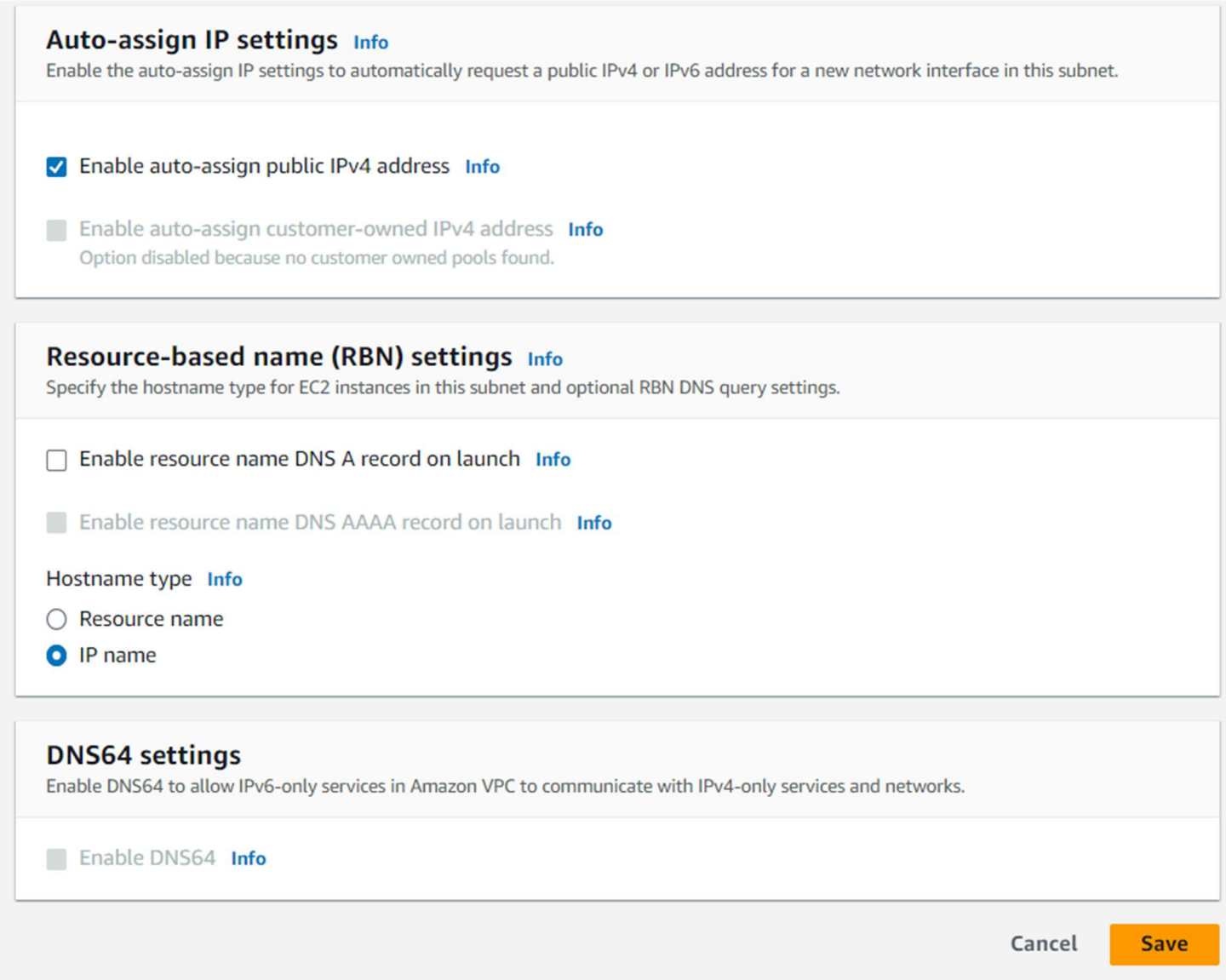
Select "**Enable auto-assign public IPv4 address**"

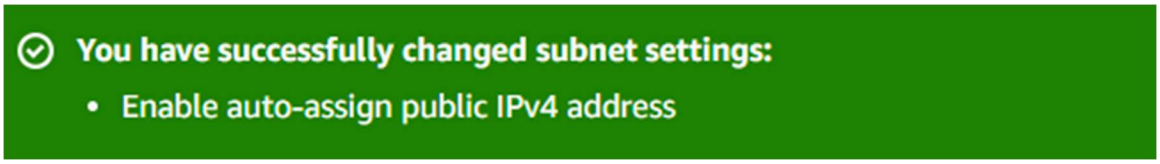
Scroll down to the bottom of the page and click "**Save**"

We have successfully configured "**Enable Auto Assign IPV4 address**" for the "**Public_Subnet_AZ2**"






**MAIN ROUTE TABLE**

**IMPPORTANT:** There already exists a **Main Route Table** that is associated with our **Dev_VPC** by default when the **Dev_VPC** was created, and it is called the **Main Route Table** and it is **private by default.**

**ROUTE TABLE CREATION**

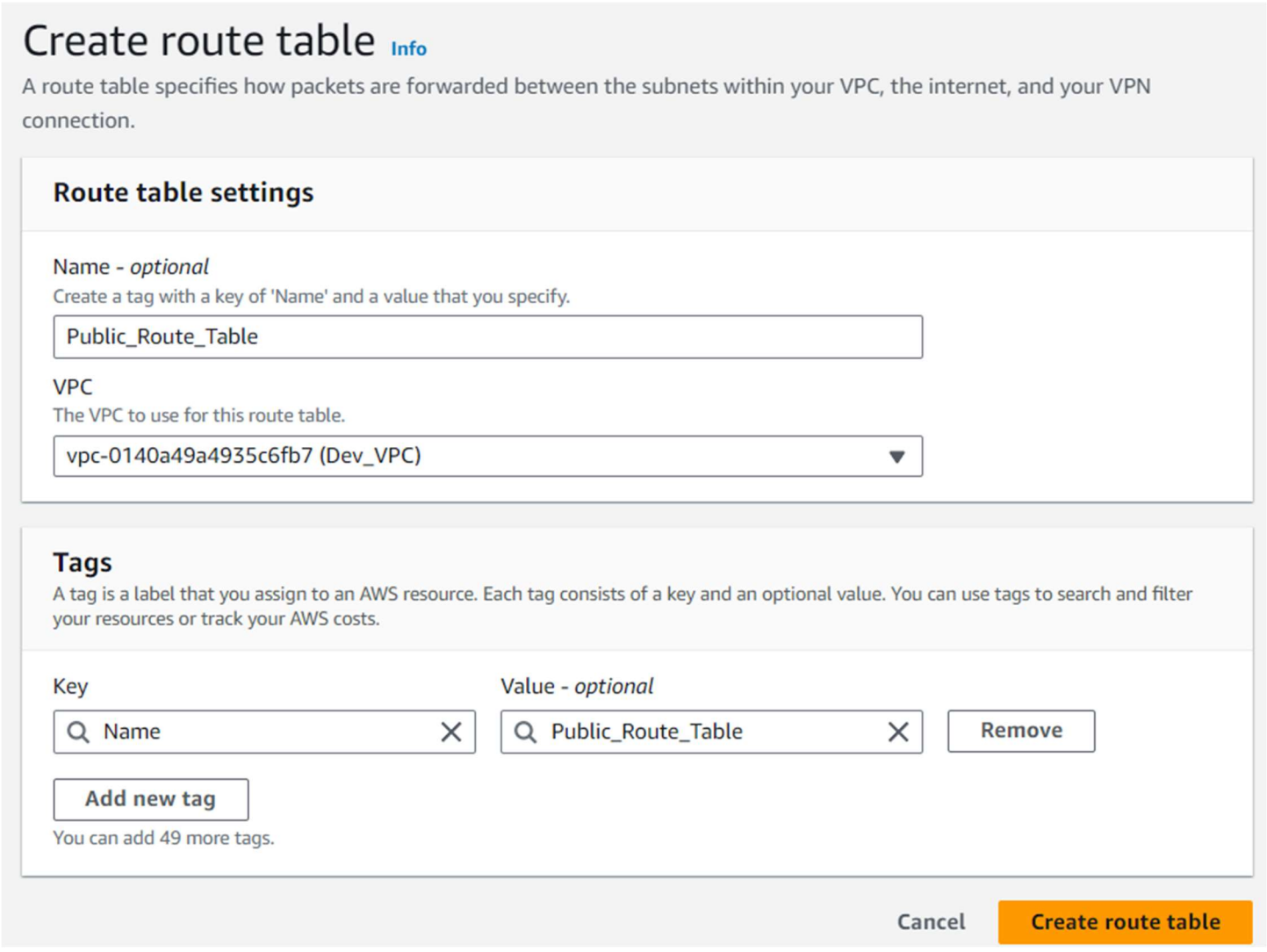
Let's Select "**Create route table**"

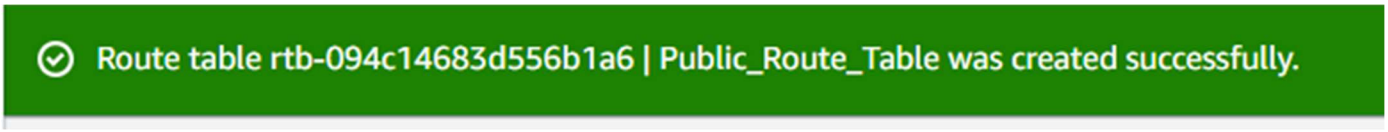
Let's call it "**Public_Route_Table**"

Let's select our **Dev_VPC**

Then click "**Create route table**"

The **Pubic_Route_Table** was successfully created



**EDIT ROUTE TABLE**

Now, we need to edit the route table by adding a **public** route to the route table

The **public** route will allow the route table to route traffic to the internet via the **Internet Gateway (IGW)**

**Follow the steps below:**

Navigate to Route tables

Click on **Route tables**

Click on **Public_Route_Table**

Make sure you have selected the "**Routes**" tab

Click on "**Edit routes**"

Click on "**Add routes**"

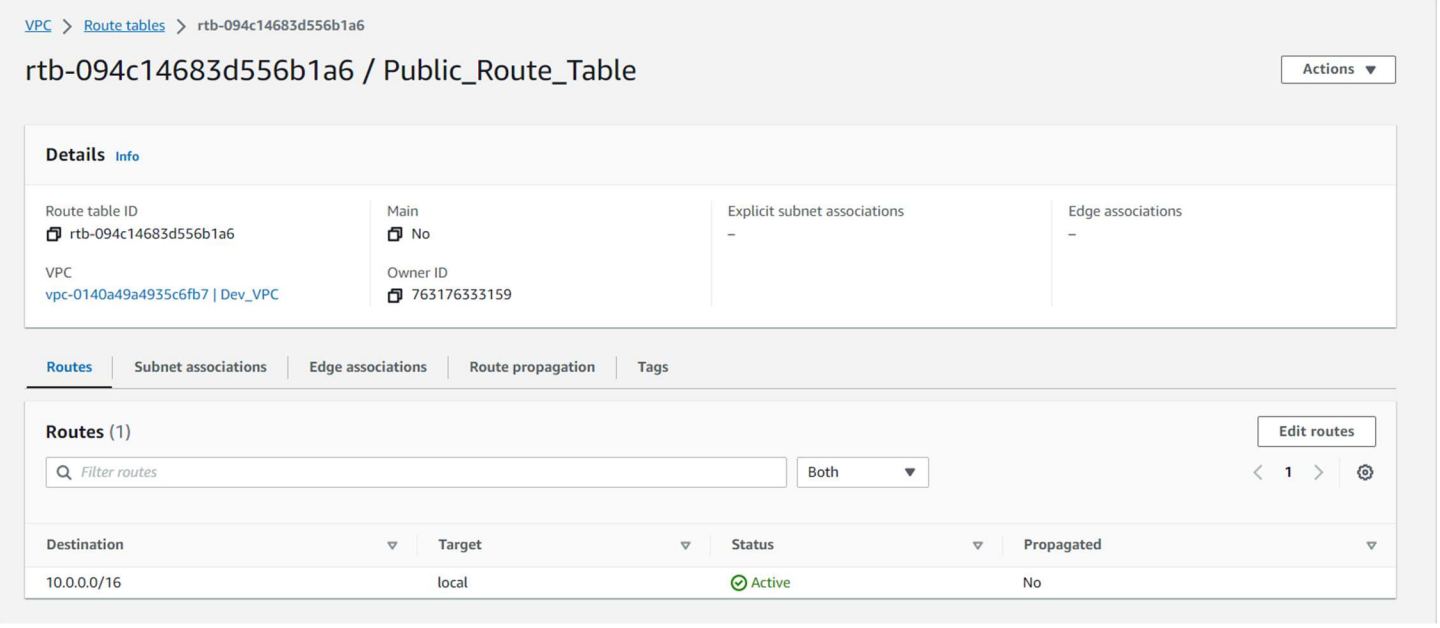
Click on the search box under "**Destination**" and enter **0.0.0.0/0**, meaning "**ANYWHERE**"

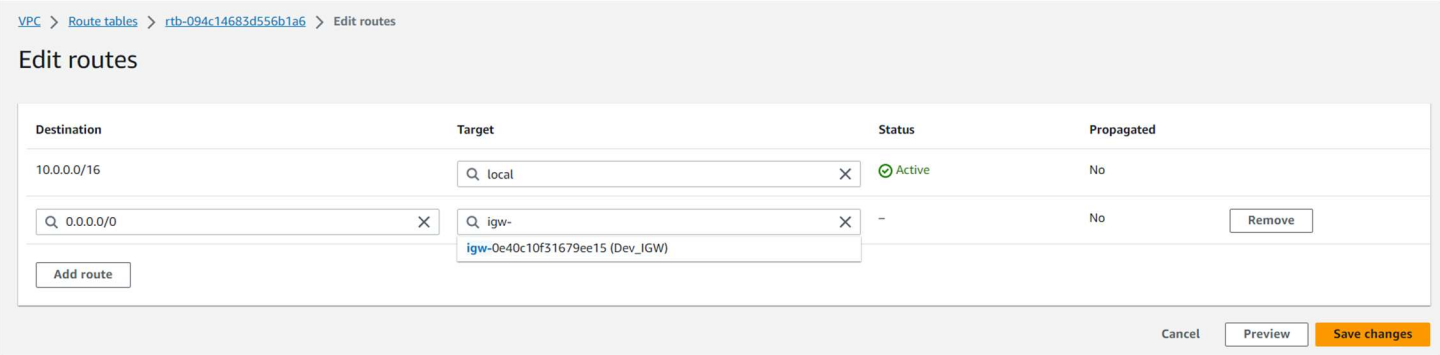
Click on the search box under "**Target**" and select "**Internet gateway"**

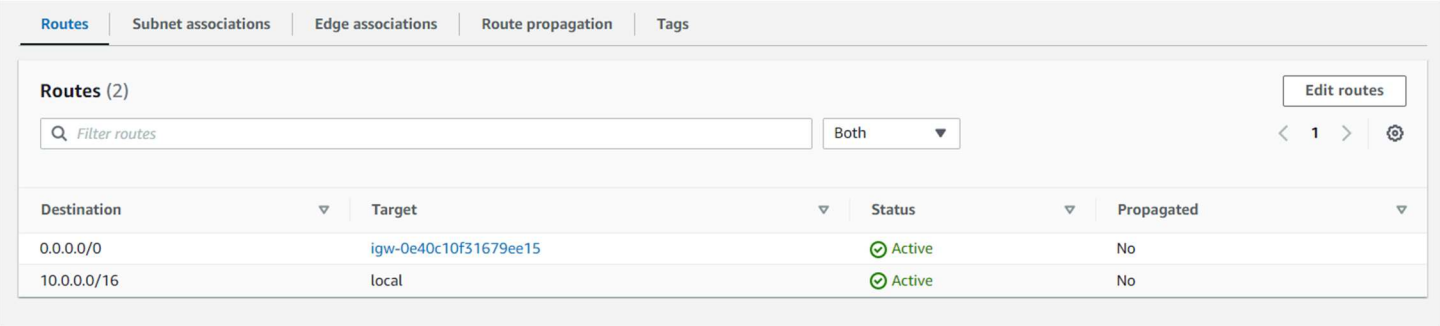
Click to select the **Dev_IGW**

Then click on "**Save Changes**"

**Public_route_table** successfully **edited** and **added** to the route table as shown below.





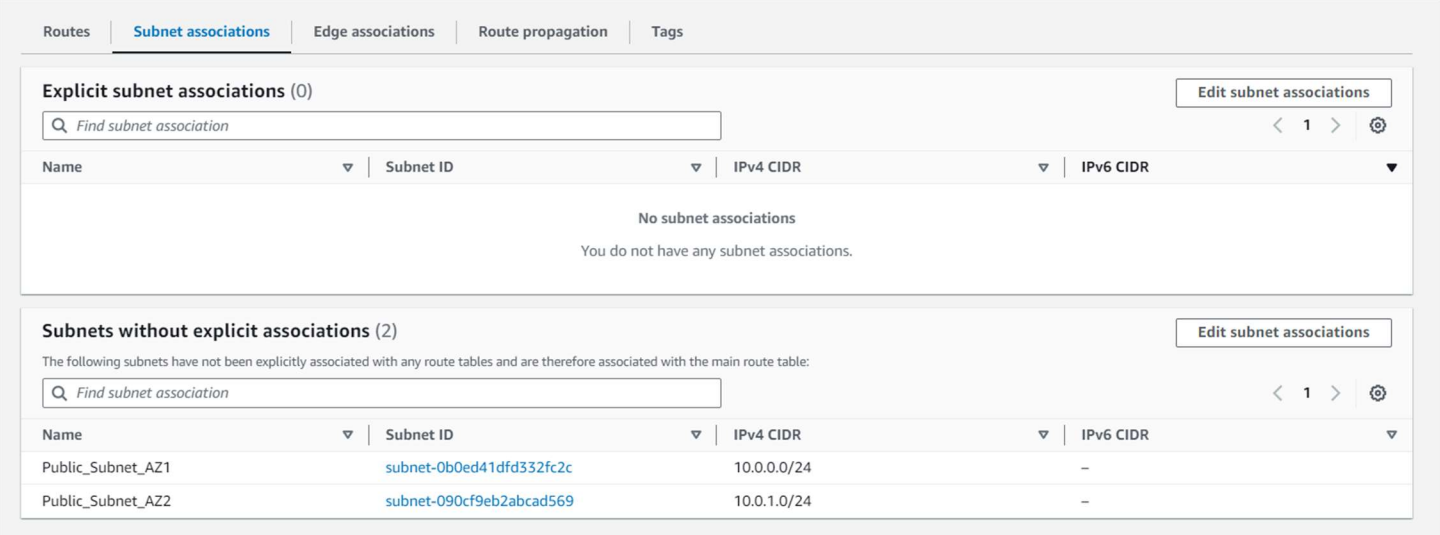



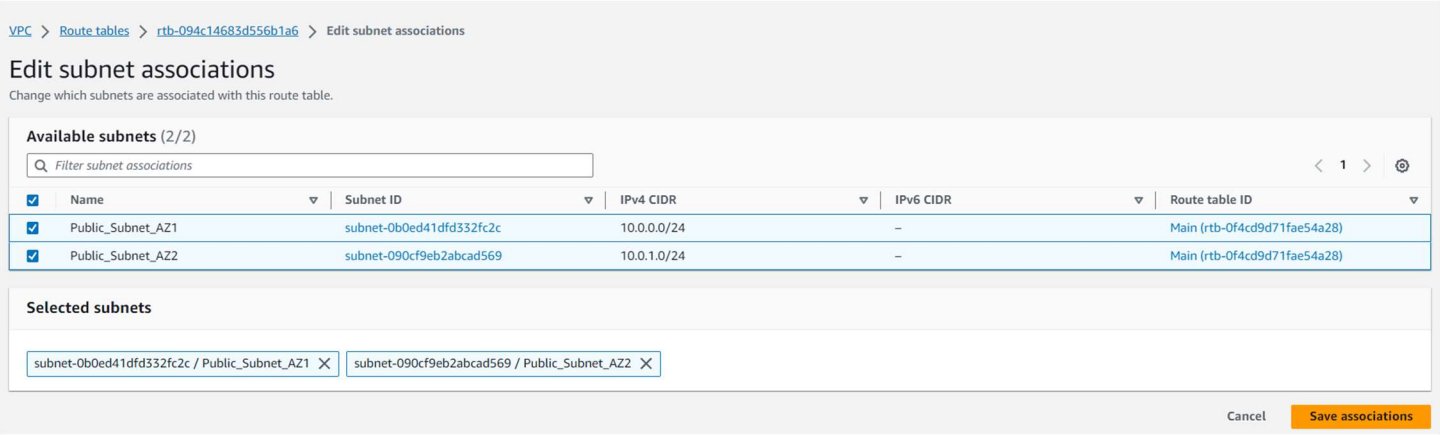

## EDIT SUBNET ASSOCIATION

Let's edit the subnet association of the "**Public_Route_Table**" linked to the 2 public subnets (**Public_Subnet_AZ1**) and (**Public-Subnet_AZ2**)

Simply click on "**Subnet associations**", proceed to click on "**Edit subnet associations**"

Select the public subnets (**Public-Subnet-AZ1**) and (**Public_Subnet_AZ2**) then click on "**Save associations**"







We have successfully updated the subnet associations for the Public_Route_Table.

**CREATION OF 4 PRIVATE SUBNETS**

Next, we need to create 4 private subnets in 2 different availability zones AZs for high availability.

The CIDR blocks assigned to the subnets must be derived from my main VPC CIDR block **10.0.0.0/16**

We can do this by specifying a subnet name, the availability zone and the IPv4 CIDR block for each subnet.

In case an AZ goes down, all traffic will be routed to the second AZ as a disaster recovery solution.

First step here will be to select "**Subnets**" on the left side of the screen

Select the VPC dropdown, then select the **Dev_VPC**

Let's name the Subnet as **Private_App_Subnet_AZ1**

Let's select the availability zone as **us-east-1a**

The IPV4 CIDR address block will be **10.0.2.0/24**

Click on **“Create subnet”**

VPC > Subnets > Create subnet

# Create subnet Info

## VPC

VPC ID
Create subnets in this VPC.

vpc-0140a49a4935c6fb7 (Dev_VPC)

### Associated VPC CIDRs

IPv4 CIDRs

10.0.0.0/16

## Subnet settings

Specify the CIDR blocks and Availability Zone for the subnet.

### Subnet 1 of 1

Subnet name
Create a tag with a key of 'Name' and a value that you specify.

Private_App_Subnet_AZ1

The name can be up to 256 characters long.

Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

US East (N. Virginia) / us-east-1a

IPv4 CIDR block Info

10.0.2.0/24

▼ Tags - *optional*

| Key | Value - *optional* | |
|---|---|---|
| Name | Private_App_Subnet_AZ1 | Remove |

Add new tag

You can add 49 more tags.

Remove

Add new subnet

Cancel Create subnet

**Private_App_Subnet_AZ1** was successfully created as we can see from the screenshot above.

The CIDR blocks assigned to the subnets must be derived from my main VPC CIDR block **10.0.0.0/16**

We can do this by specifying a subnet name, the availability zone and the IPv4 CIDR block for each subnet.

In case an AZ goes down, all traffic will be routed to the second AZ as a disaster recovery solution.

First step here will be to select "**Subnets**" on the left side of the screen

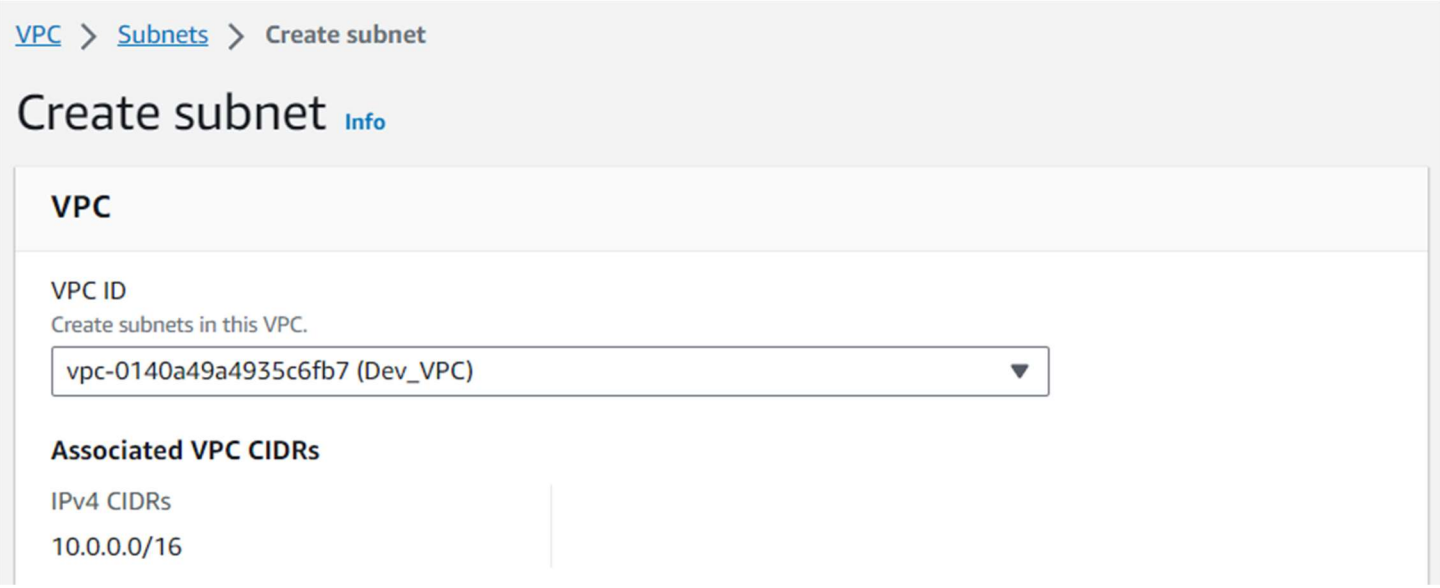
Select the VPC dropdown, then select the **Dev_VPC**

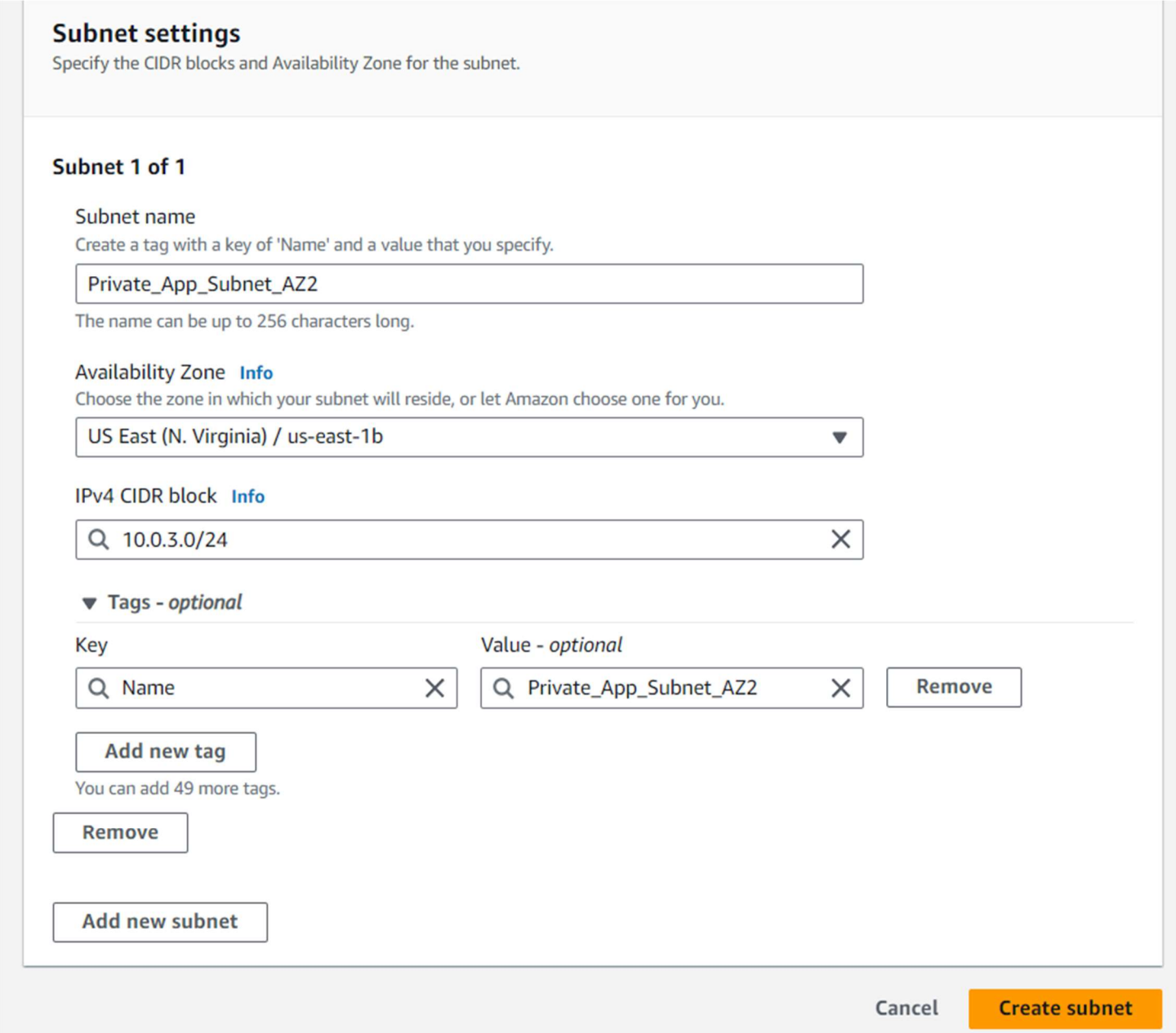
Let's name the Subnet as **Private_App_Subnet_AZ2**

Let's select the availability zone as **us-east-1b**

The IPV4 CIDR address block will be **10.0.3.0/24**

Click on **Create subnet**






**Private_App_Subnet_AZ2** was successfully created as we can see from the screenshot above.

First step here will be to select "**Subnets**" on the left side of the screen

Select the VPC dropdown, then select the **Dev_VPC**

Let's name the Subnet as **Private_Data_Subnet_AZ1**

Let's select the availability zone as **us-east-1a**

The IPV4 CIDR address block will be **10.0.4.0/24**

Click on "**Create subnet**"

VPC > Subnets > Create subnet

# Create subnet Info

## VPC

VPC ID
Create subnets in this VPC.

vpc-0140a49a4935c6fb7 (Dev_VPC)

### Associated VPC CIDRs

IPv4 CIDRs

10.0.0.0/16

## Subnet settings

Specify the CIDR blocks and Availability Zone for the subnet.

### Subnet 1 of 1

Subnet name
Create a tag with a key of 'Name' and a value that you specify.

Private_Data_subnet_AZ1

The name can be up to 256 characters long.

Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

US East (N. Virginia) / us-east-1a

IPv4 CIDR block Info

10.0.4.0/24

▼ Tags - *optional*

| Key | Value - *optional* | |
|---|---|---|
| Name | Private_Data_subnet_AZ1 | Remove |

Add new tag

You can add 49 more tags.

Remove

Add new subnet

Cancel Create subnet

**Private_Data_Subnet_AZ1** successfully created as we can see from the screenshot above.

The CIDR blocks assigned to the subnets must be derived from my main VPC CIDR block **10.0.0.0/16**

We can do this by specifying a subnet name, the availability zone and the IPv4 CIDR block for each subnet.

In case an AZ goes down, all traffic will be routed to the second AZ as a disaster recovery solution.

First step here will be to select "**Subnets**" on the left side of the screen

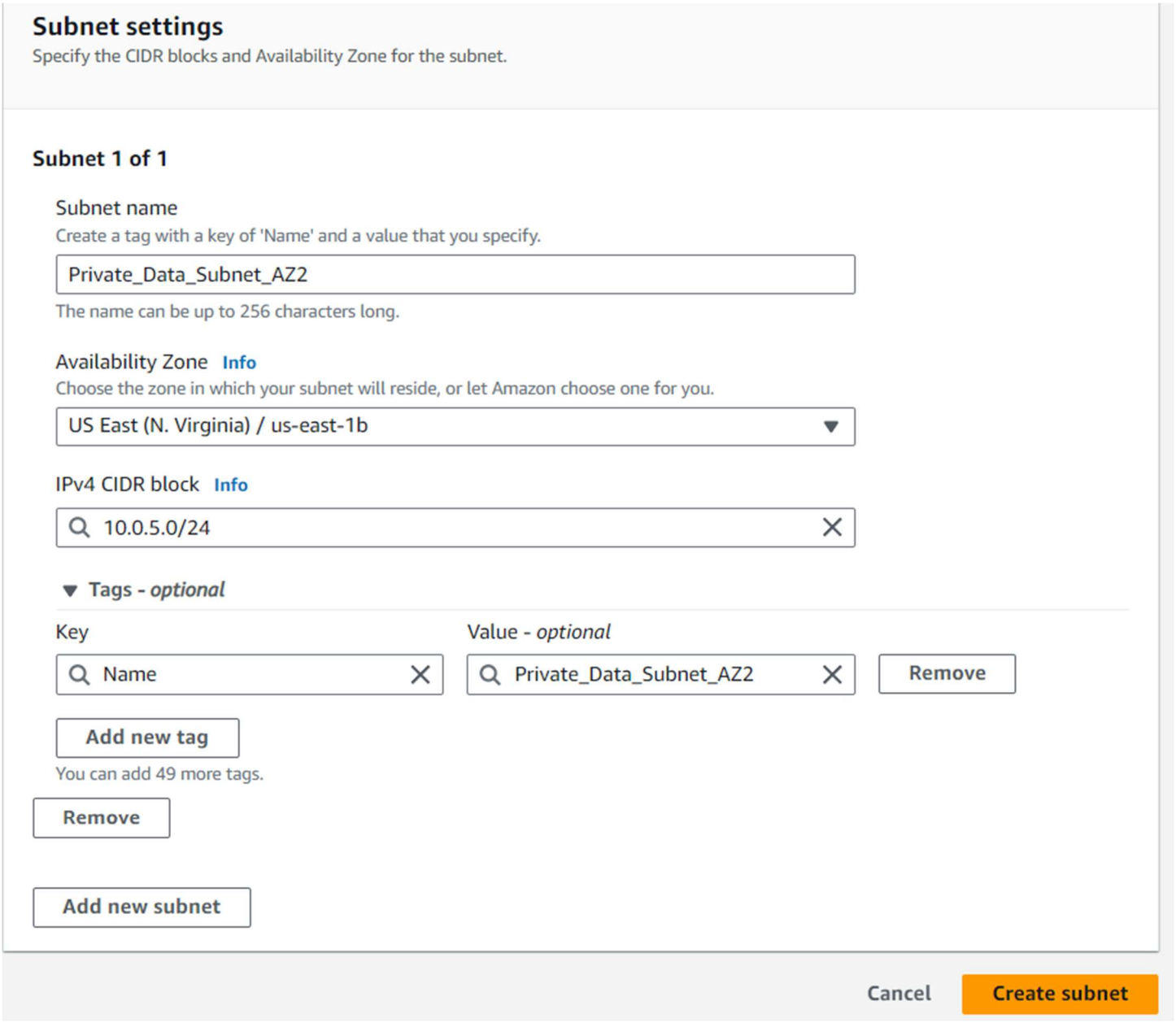
Select the VPC dropdown to select the **Dev_VPC**

Let's name the Subnet as **Private_Data_Subnet_AZ2**

Let's select the availability zone as **us-east-1b**

The IPV4 CIDR address block will be **10.0.5.0/24**

Click on "**Create subnet**"




**Private_Data_Subnet_AZ2** successfully created as we can see from the screenshot above.

## VERY IMPORTANT! VERY IMPORTANT!! VERY IMPORTANT!!!

The key difference between a **PUBLIC SUBNET** and a **PRIVATE SUBNET** lies in their accessibility and routing configuration.

These concepts are fundamental to creating a secure and functional network architecture within AWS.

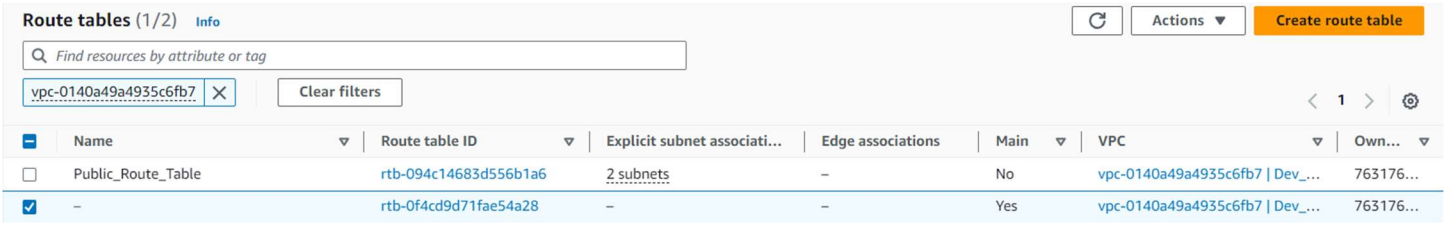
**Public subnets** can send and receive traffic directly to and from the internet, while private subnets rely on **NAT gateways or instances** for outbound internet access and are generally

isolated from incoming internet traffic. This segregation helps enhance security and control, in the **AWS VPC architecture**.

When you create a Route Table and add a ROUTE (**ex. 0.0.0.0/0 and igw-0e40c10f31679ee15**) to that route table, to enable it route traffic to the internet, any subnet, or subnets you associate with that route table automatically becomes public.

**PUBLIC SUBNETS AND PUBLIC ROUTE TABLE**

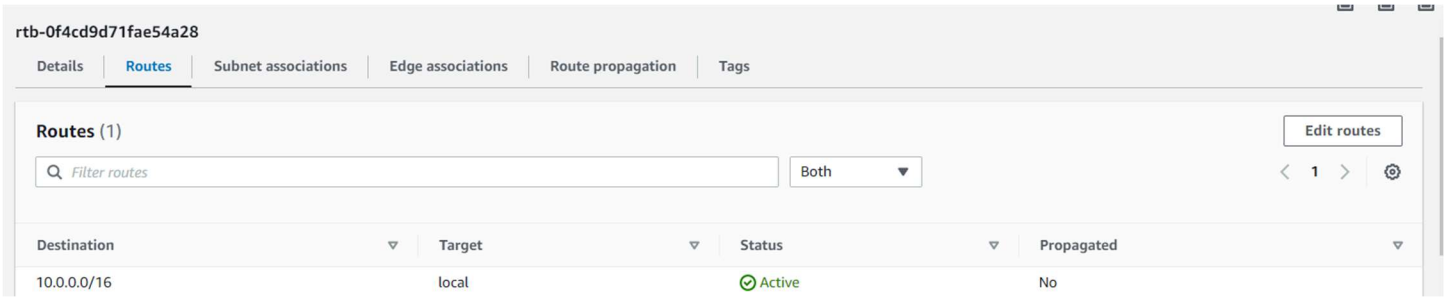
**Route Table:**
Public subnets typically have a **route table** that directs traffic to the internet gateway. This allows resources in the public subnet to communicate with resources outside the **VPC** over the internet.

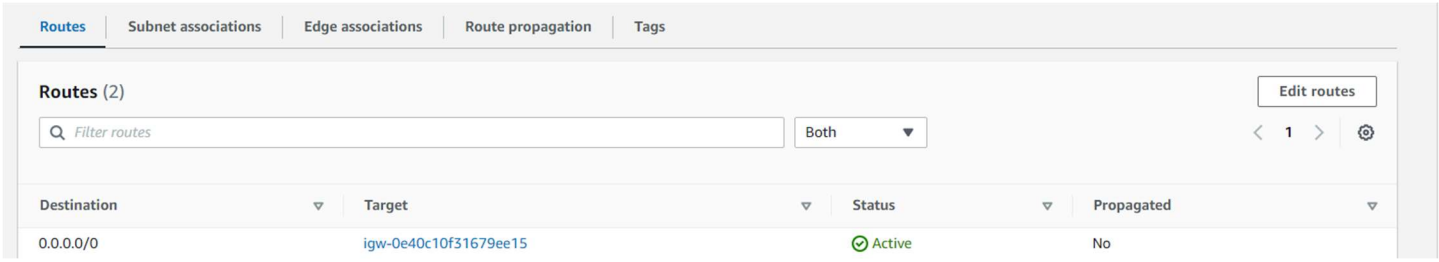


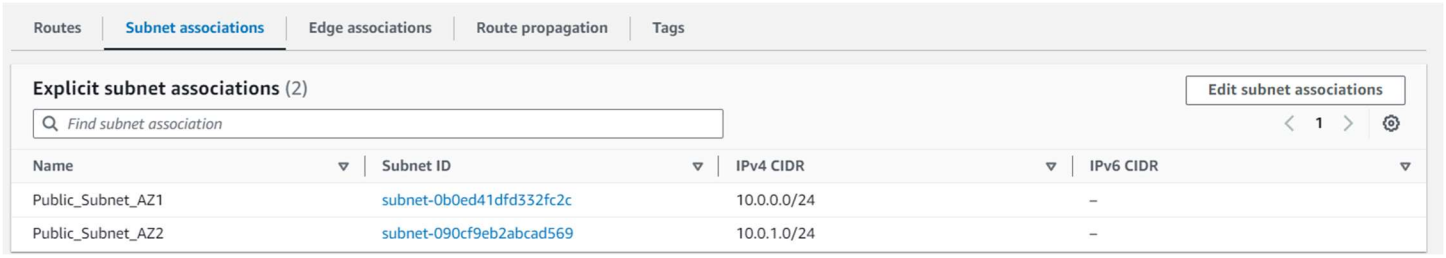

**Internet Accessibility:**
A **public subnet** is configured to have direct access to the internet. Instances launched in a public subnet can send and receive traffic to and from the internet if they have an Elastic IP address or are associated with a Network Address Translation (**NAT**) gateway or instance.

**Use Cases:**
Public subnets are commonly used for resources that need to be publicly accessible, such as web servers, load balancers, or Bastion hosts. They serve as the entry point for incoming traffic from the internet.
From the screenshots below, the route is (**0.0.0.0/0 and igw-0e40c10f31679ee15**) which gives direct access to the internet.

From the screenshots below, the Explicit Subnet Associations that is linked to the route (**0.0.0.0/0 and igw-0e40c10f31679ee15)** with direct access to the internet are **Public_subnet_AZ1** and **Public_Subnet_AZ2.**



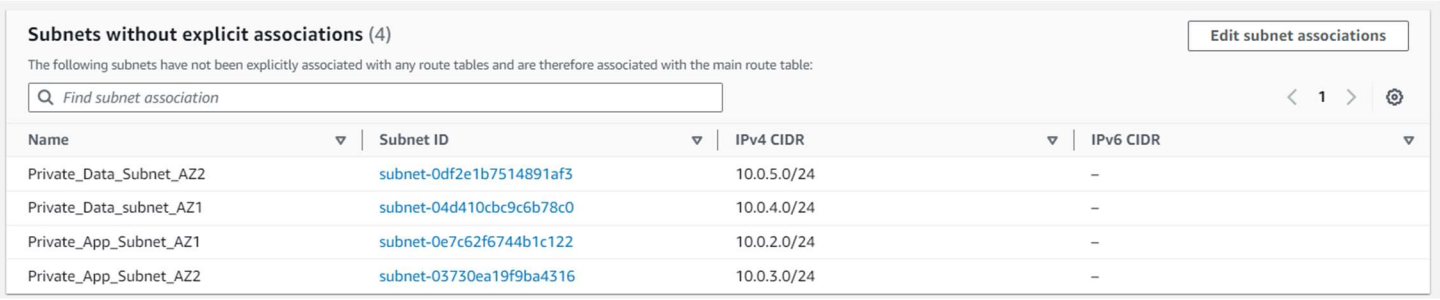




**PRIVATE SUBNET AND PRIVATE ROUTE TABLE**

**Route Table:**
Private subnets often have a different route table compared to public subnets. Traffic routed from the private subnets and destined for the internet is typically routed through a **NAT gateway or NAT instance** hosted in the public subnet.

The **main route table** only routes traffic locally within the **VPC,** and it is created by **default** when a **VPC** is created.

When you have a route table that does not have a route to the Internet **(ex. 10.0.0.0/16)** in our case, any subnet linked with that route table becomes private by default.




The Main route table **(10.0.0.0/16)** is private by default, and we created 4 private subnets that we did not "**explicitly**" associate with any **route table**, therefore the 4 private subnets will be automatically associated with the **Main Route Table** which is **private by default.**

**TO SHOW THAT INFORMATION IN DETAIL:**

Select the "**Main route table"**

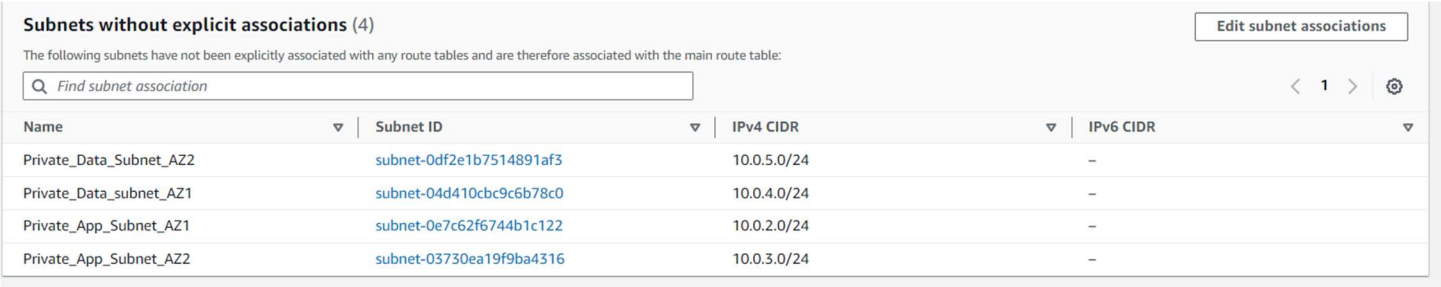
Select **"Subnet association"** tab and scroll to the bottom of the page

Here you will see that there are 4 private subnets shown as:

(1) Subnets without explicit associations (4)

(2) The message on the fine line states that:

**The following subnets have not been explicitly associated with any route tables and are therefore associated with the main route table:**




Private subnets are used for resources that should not be directly accessible from the internet, such as databases, application servers, and internal services. To enable them to access the internet for software updates or other purposes, a **NAT gateway** or **NAT instance** is used to proxy the traffic.

The Instances or resources in a private subnet cannot communicate with the internet directly, and incoming traffic from the internet cannot reach instances in a private subnet without the use of a **NAT gateway** or **NAT instance**.

In the future steps, we will be creating **NAT gateways** and **Private Route table**

Then we will explicitly associate the Private Subnets with the public route table.

At this moment, we would allow the 4 Private subnets to be associated with the **Main Route Table** which is the private by default.

## STEP 22: CONFIGURATION OF NAT GATEWAY #1

**Let's configure the first NAT Gateway**

**1. NAT_Gateway_AZ1**

The main function of the **NAT Gateway** is to allow resources in the **private subnets** to have **access** to the **internet** for software updates, package installation, downloads etc.

**STEPS**

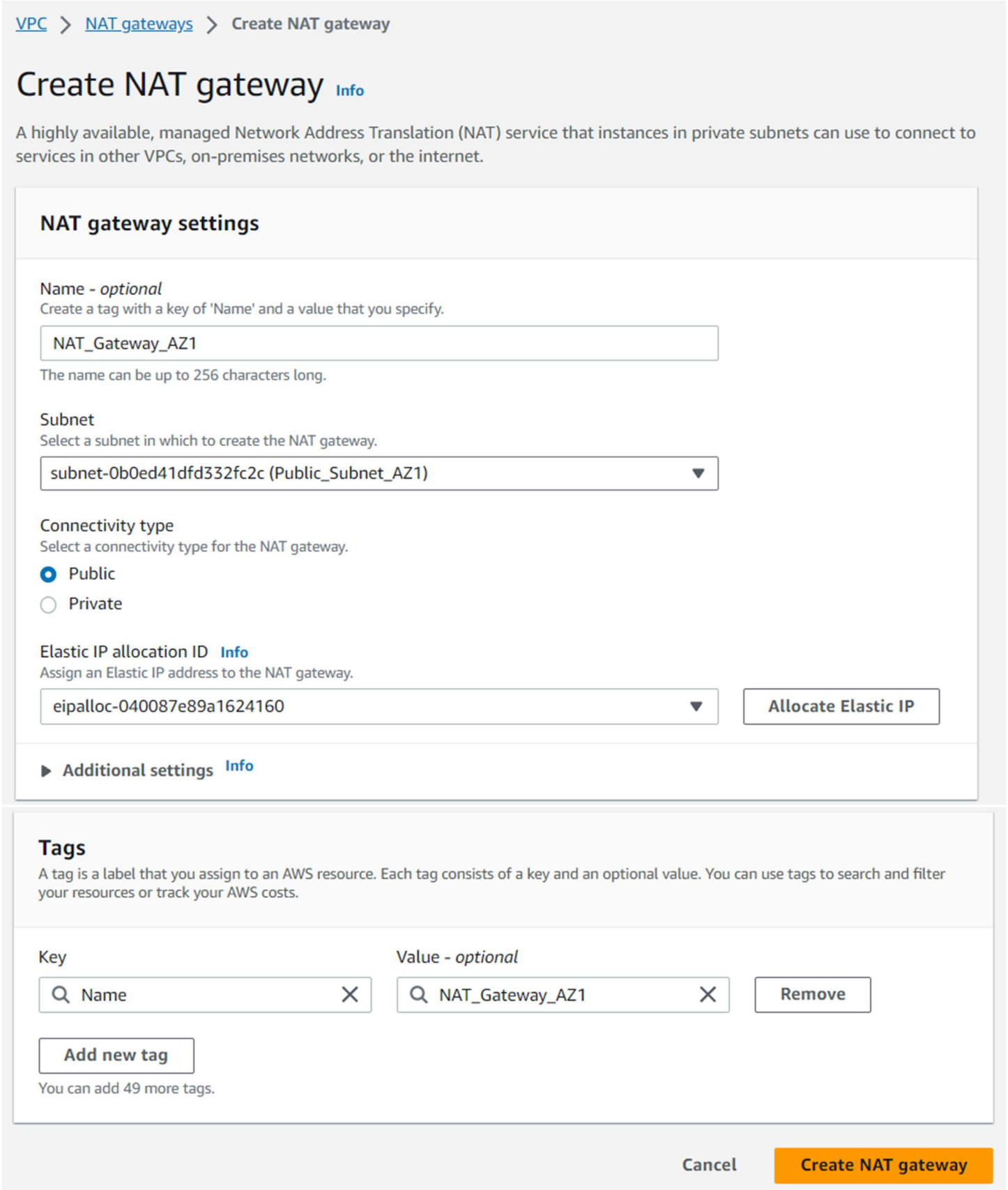
Navigate to **VPC**, scroll down to select **NAT gateways,** and click on "**Create NAT gateway**"

Let's specify the public subnets where we want the NAT gateways to reside when created.

Let's select our first public subnet "**Public_Subnet_AZ1**" for this configuration, and then select connectivity type as public.

We must click on "**Allocate Elastic IP**" to allocate an Elastic IP address to the NAT gateway.

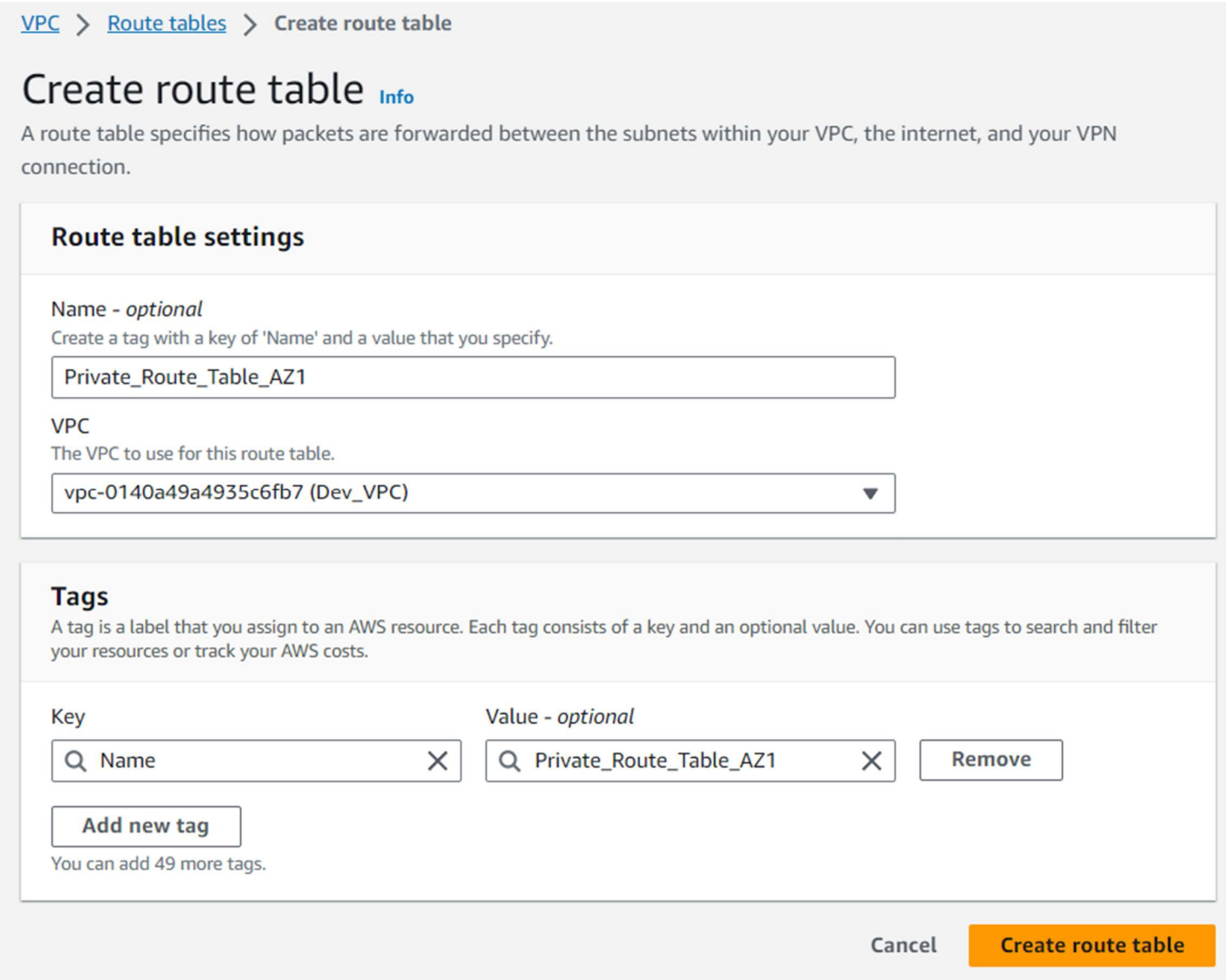
Click "**Create NAT Gateway**"




The first NAT gateway **(NAT_Gateway_AZ1)** was successfully deployed in the public subnet **(Public_Subnet_AZ1)** and configured to receive outgoing traffic from **(Private_App_Subnet_AZ1)** and **(Private_Data_Subnet_AZ1)** then route the traffic towards the Internet Gateway **(IGW)**

## STEP 23: CREATION OF ROUTE TABLE

**(a)** Create the private Route Table **(Private_Route_Table_AZ1)**

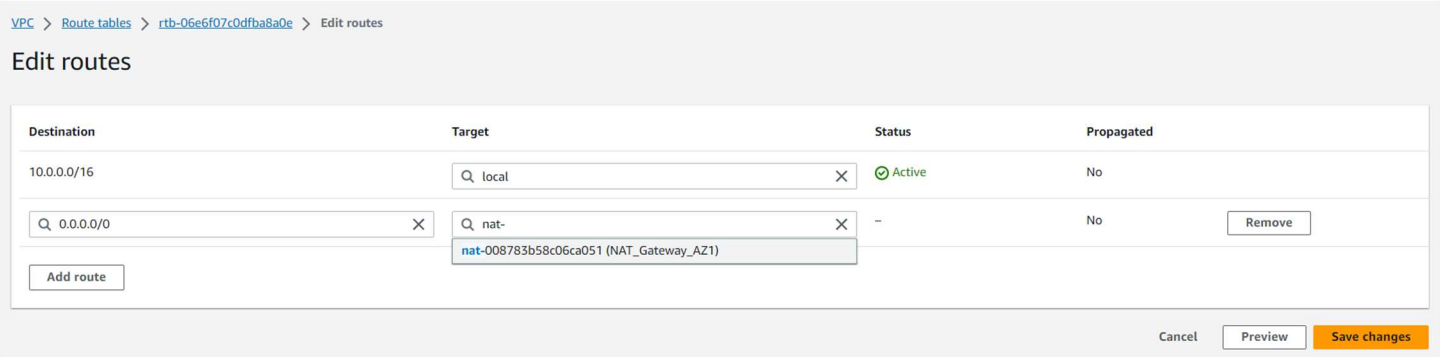



**(b)** Edit the Private route table **(Private_Route_Table_AZ1**) to route traffic to the first NAT gateway **(NAT_Gateway_AZ1)** that resides in the public subnet **(Public_Subnet_AZ1)**

For this configuration, simply click on the Route table **(Private_Route_Table_AZ1)**

Select "**Edit routes**" > Select "Add route" in the blank field, enter **0.0.0.0/0**

Then scroll down to select NAT gateway **(NAT_Gateway_AZ1)** as target, then click **"Save changes"**




**(c)** Edit subnet association

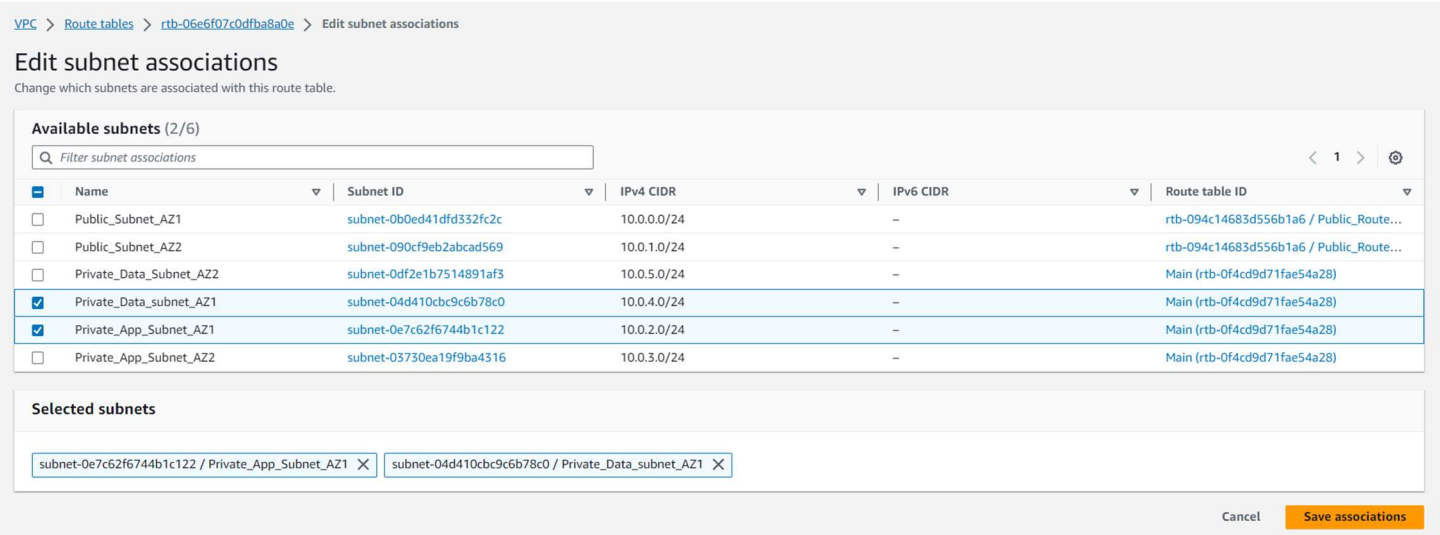
To edit the subnet association for the route table **(Private_Route_Table_AZ1)**

Click on "**Subnet associations**"

Click on "**Edit subnet associations**"

Select the private subnets **(Private_App_Subnet_AZ1)** and **(Private_Data_Subnet_AZ1)**

Then click on "**Save associations**"




We have successfully associated **our (Private_App_Subnet_AZ1)** and **(Private_Data_Subnet_AZ1)** to the route table **(Private_Route_Table_AZ1)**

## STEP 24: CONFIGURATION OF NAT GATEWAY #2

Let’s configure the first NAT Gateway

1. **NAT_Gateway_AZ2**

The main function of the **NAT Gateway** is to allow resources in the **private subnets** to have access to the internet for software updates, package installation, downloads etc.

**STEPS:**

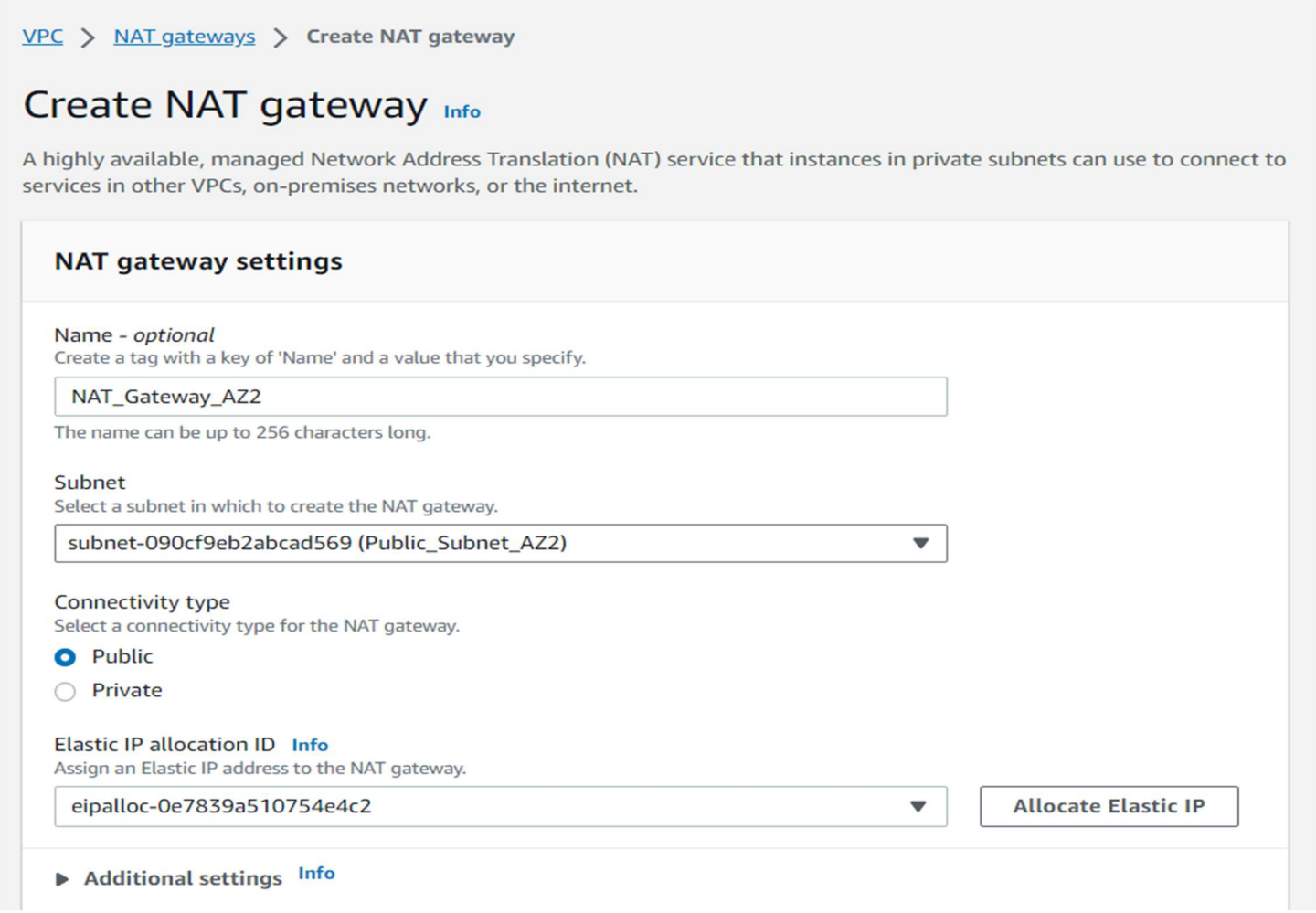
Navigate to **VPC,** scroll down to select **NAT gateways**, and click on create NAT gateway.

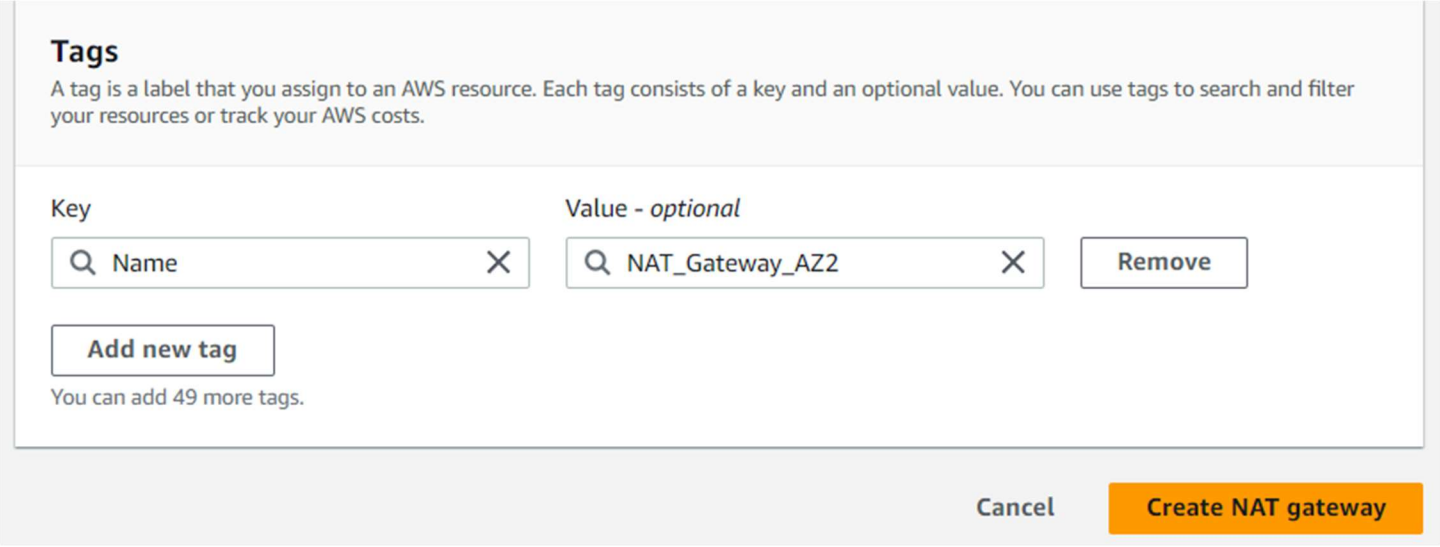
Let’s specify the public subnets where we want the NAT gateways to reside when created.

Let’s select our second public subnet "**Public_Subnet_AZ2**" for this configuration, and then select connectivity type as public.

We must click on "**Allocate Elastic IP**" to allocate an Elastic IP address to the NAT gateway.

Click "**Create NAT gateway**"

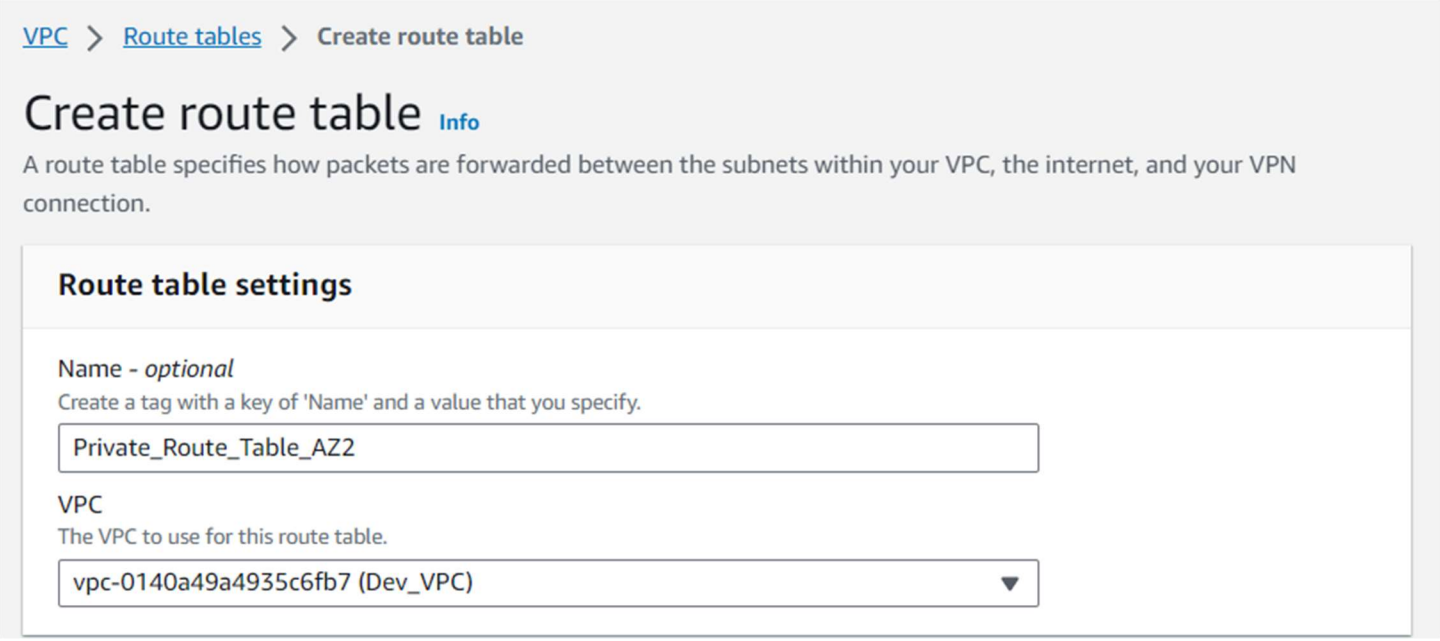


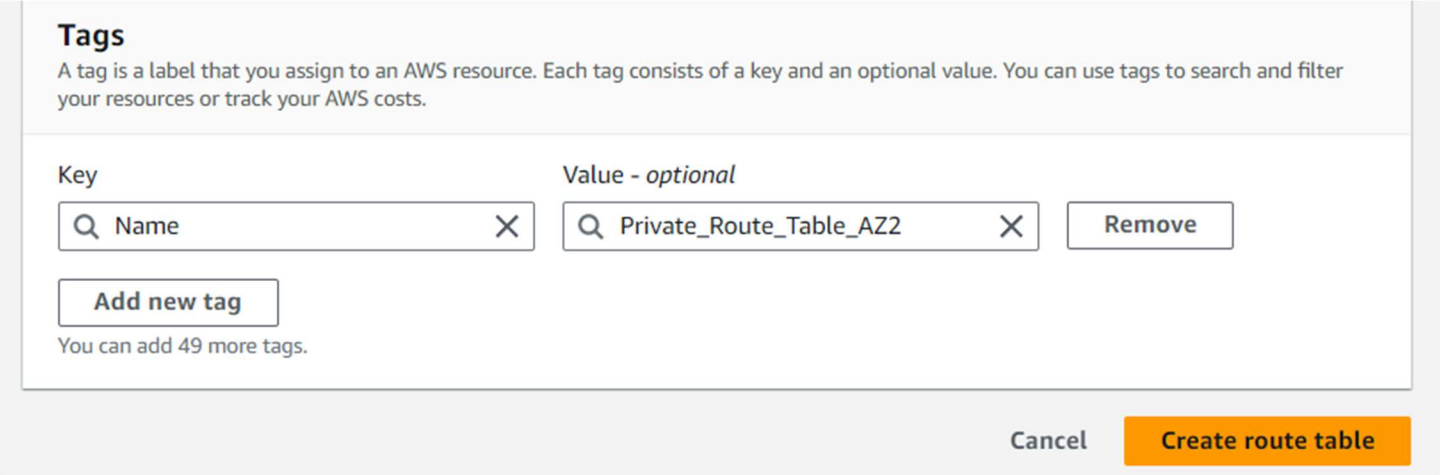



The second NAT gateway **(NAT_Gateway_AZ2)** was successfully deployed in the public subnet **(Public_Subnet_AZ2)** and configured to receive outgoing traffic from **(Private_App_Subnet_AZ2)** and **(Private_Data_Subnet_AZ2)** then route the traffic towards the Internet Gateway **(IGW)**

## STEP 25: CREATION OF ROUTE TABLE

**(a)** Create the private Route Table **(Private_Route_Table_AZ2)**





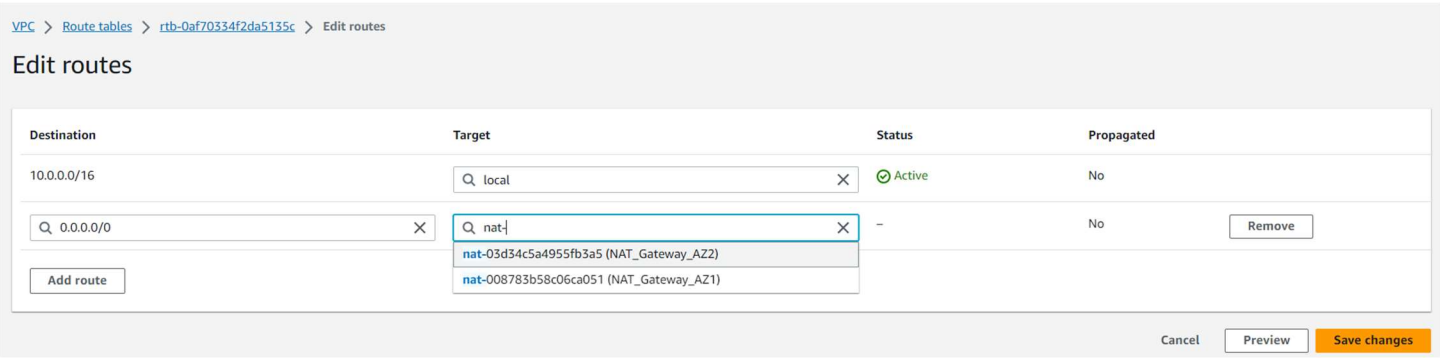

(b) Edit the Private route table **(Private_Route_Table_AZ2)** to route traffic to the second NAT gateway

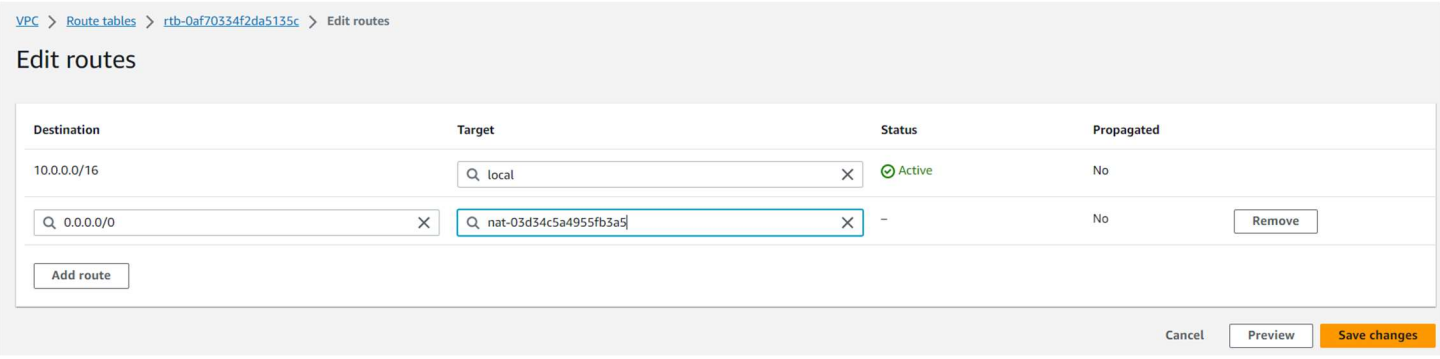
**(NAT_Gateway_AZ2)** that resides in the public subnet **(Public_Subnet_AZ2)**

For this configuration, simply click on the Route table **(Private_Route_Table_AZ2)**

Select Edit routes, select add route, in the blank field, enter **0.0.0.0/0**

Then scroll down to select NAT gateway **(NAT_Gateway_AZ2)** as target, then click "**Save changes**"



**(c) Edit subnet association**

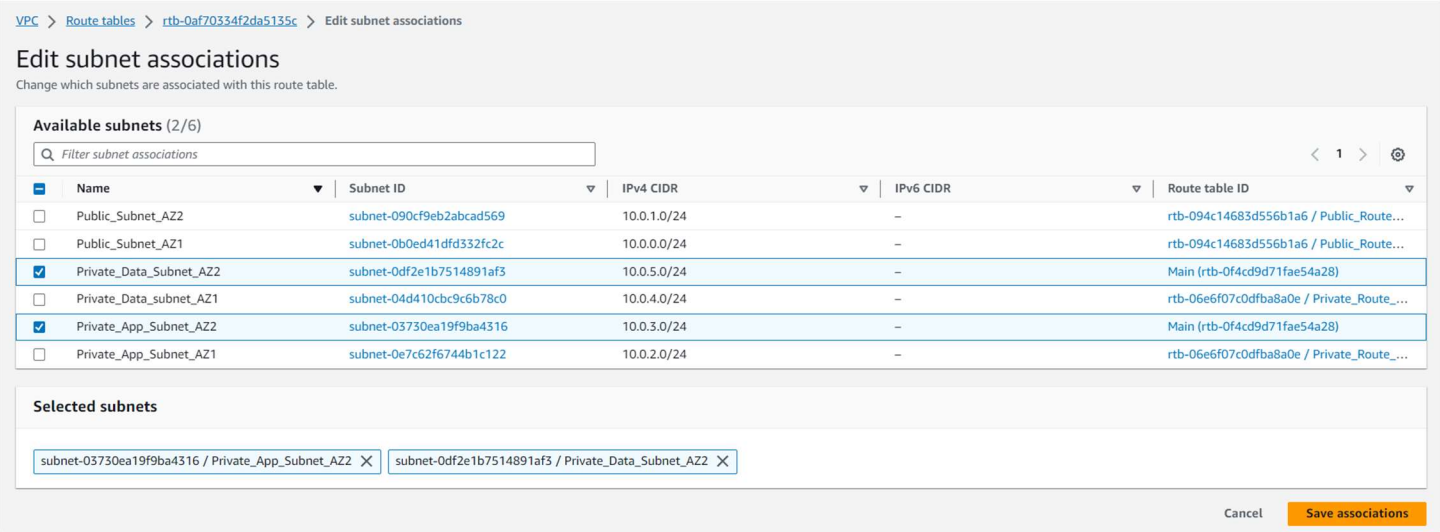
To edit the subnet association for the route table **(Private_Route_Table_AZ2)**

Click on "**Subnet associations**"

Click on "**Edit subnet associations**"

Select the private subnets **(Private_App_Subnet_AZ2)** and **(Private_Data_Subnet_AZ2)**

Then click on "**Save associations**"




We have successfully associated our **(Private_App_Subnet_AZ2)** and **(Private_Data_Subnet_AZ2)** to the route table **(Private_Route_Table_AZ2)**

# STEP 26:

# CREATE SECURITY GROUPS

We will create two **security groups** for this project

(1) The first security group to create will be **ALB Security Group**

We will open **port 80** from anywhere on the internet

We will open **port 443** from anywhere on the internet

**Port = 80 and 443**

**Source = 0.0.0.0/0**

Let’s navigate to the EC2 dashboard, scroll down to select **Security groups**

Click on create security group and let's name it **"ALB_security_group"**

Click on description info and let's enter **"ALB_security_group"**

Click on VPC info and let's select the **Dev_VPC**

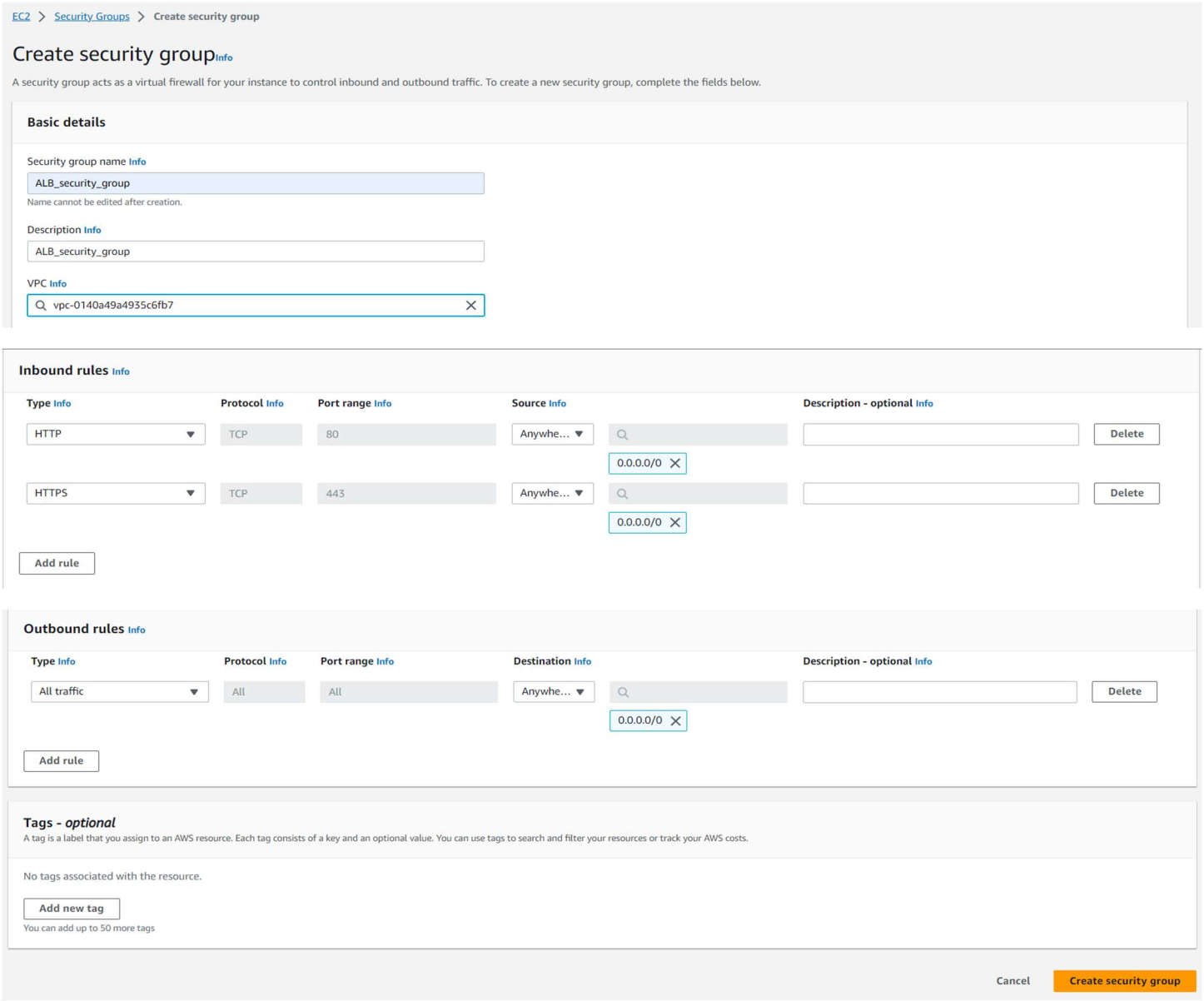
Click on **“Inbound rules”** > Click on “**Add rules”**

Click and configure **HTTP port 80** from anywhere **0.0.0.0/0**

Click and configure **HTTPS port 443** from anywhere **0.0.0.0/0**

Outbound rule configuration as default

Click on **"Create security group"**




**ALB_security_group** was successfully created

(2) The second security group will be **Container security group**

We will open **port 80** and the source of traffic will be from the **ALB security group**

We will open **port 443** and the source of traffic will be from the **ALB security group**

**Port = 80 and 443**

Source **= ALB Security group**

Let’s navigate to the EC2 dashboard, scroll down to select **Security groups.**

Click on create security group and let's name it **"Container_security_group"**

Click on description info and let's enter **"Container_security_group"**

Click on VPC info and let's select the **Dev_VPC**

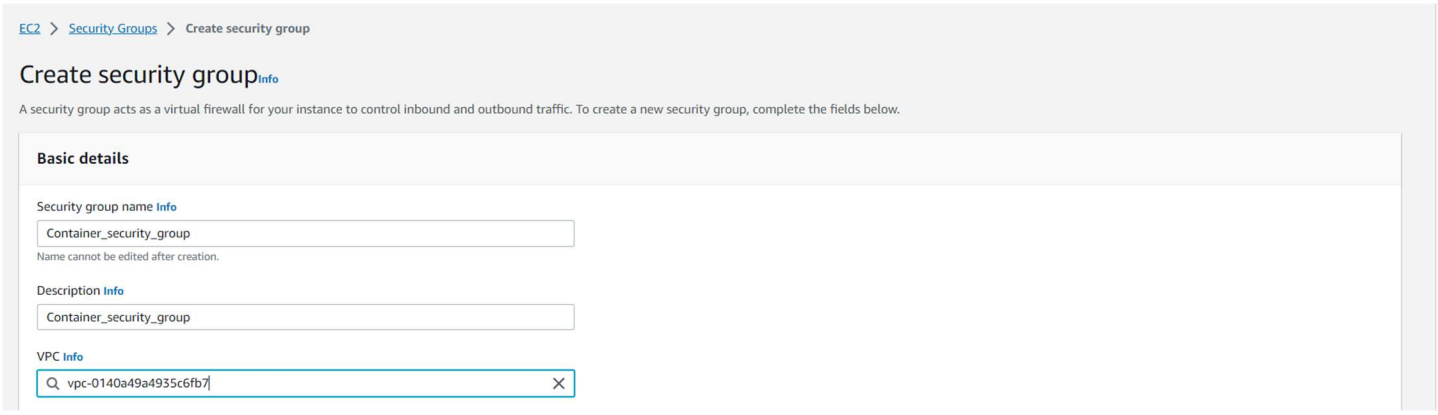
Click on “**Inbound rules”** > Click on **“Add rules”**

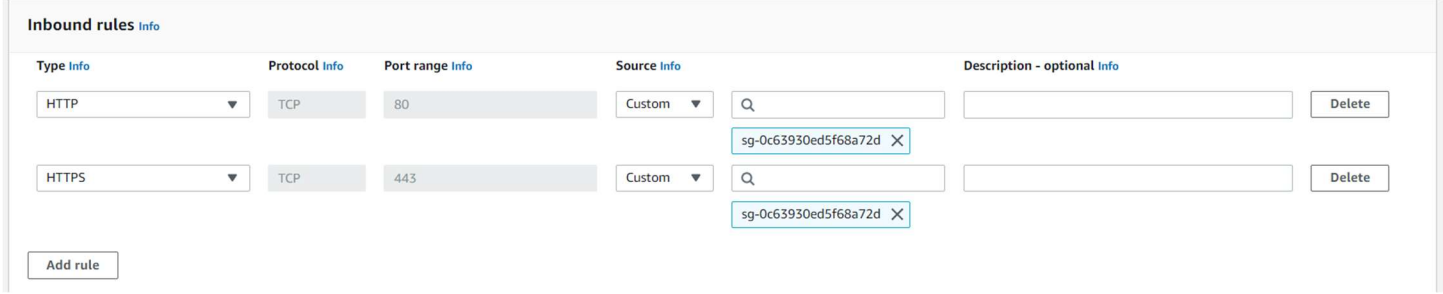
Click and configure HTTP **port 80** from **ALB_security_group**

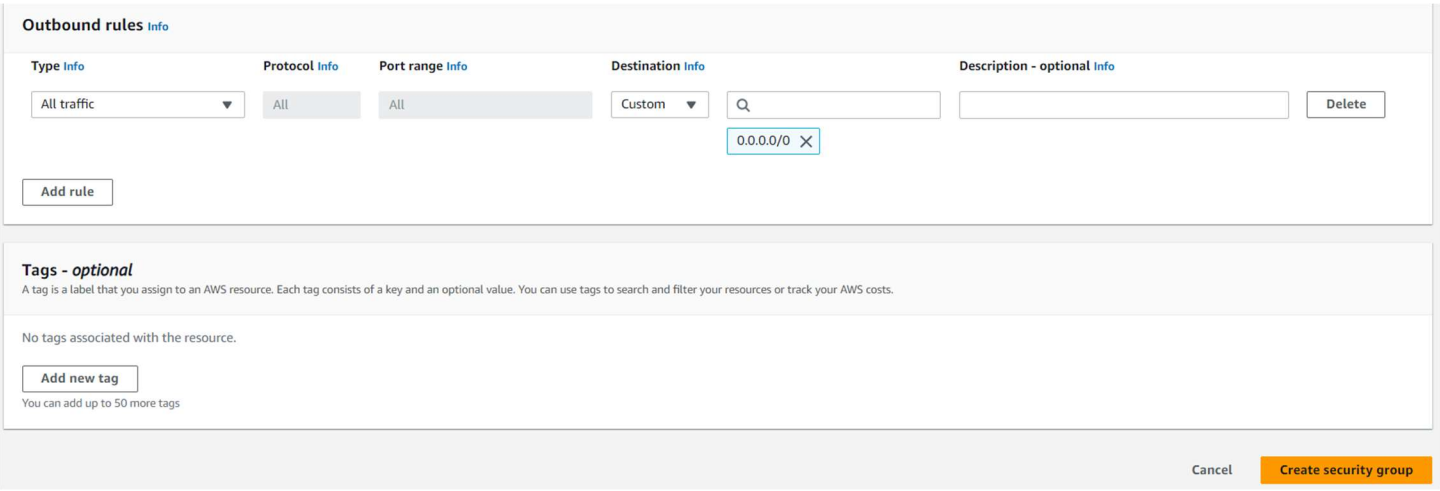
Click and configure HTTPS **port 443** from **ALB_security_group**

Leave outbound rule configuration as default

Click on **"Create security group"**






**Container_security_group** was successfully created

## STEP 27:

## CREATION OF TARGET GROUPS FOR ALB

Go to the **EC2** service.

In the navigation pane, click on **"Target Groups**" under "Load Balancing."

Click on the **"Create target group"**

Specify a name for the target group

Configure the health checks for your target group.

Health checks help the load balancer to determine the availability of targets.

Set the protocol, path, and interval for the health checks.

Specify the targets for your target group. This depends on the target type you selected earlier.

If you choose "**Instance"** you can select one or more instances to include in the target group.

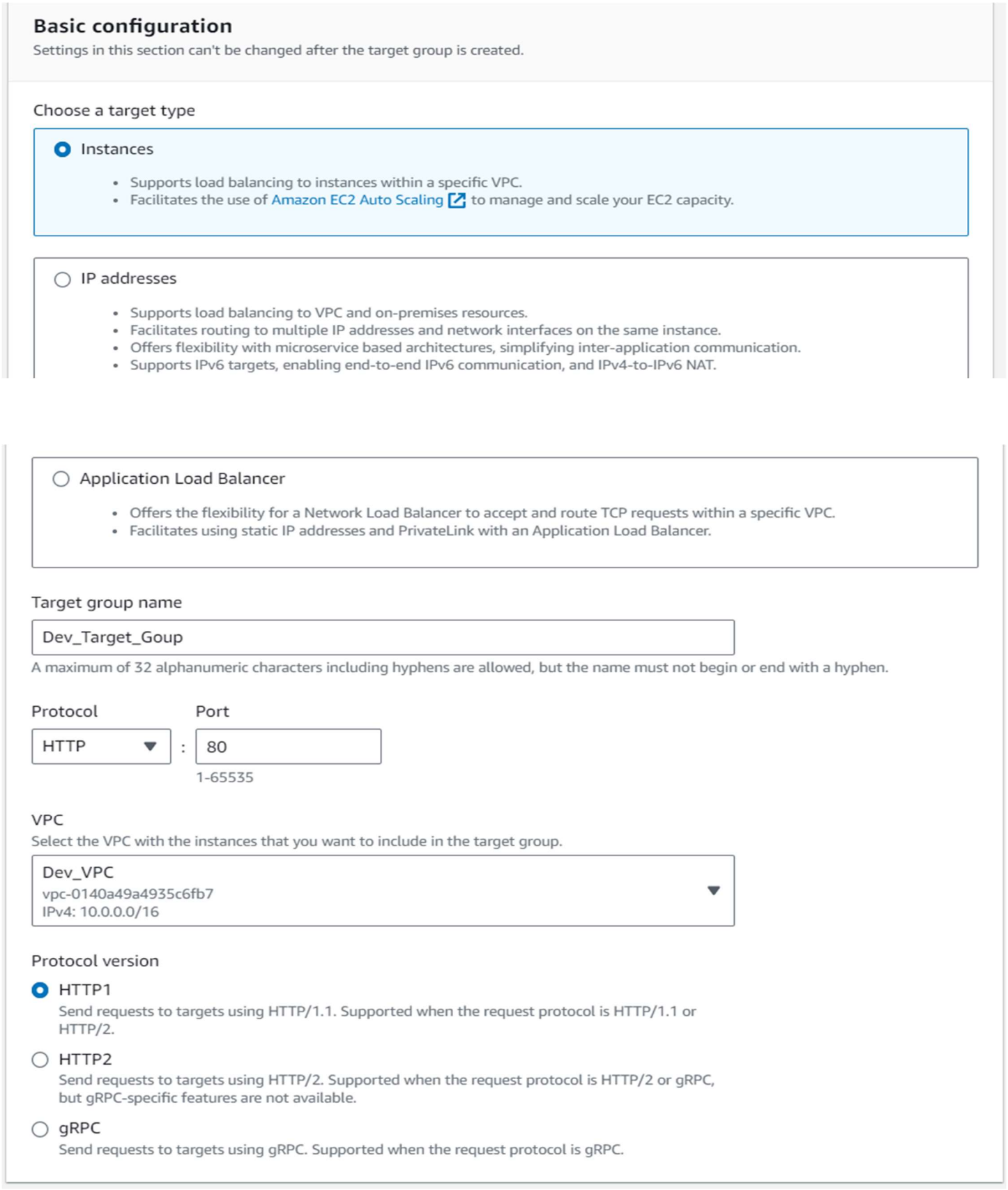
Once the target group is created, you can associate it with your Application Load Balancer to route traffic to the targets based on the specified rules.

**IMPORTANT:**

1.This target will be deleted when we are done creating the Application Load Balancer ALB in this project.

2. When we create the ECS service, we will create a Target Group that the ALB will route traffic.

Now, let's proceed to create our **Target Group**



## Health checks

The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.

Health check protocol

HTTP

Health check path

Use the default path of "/" to ping the root, or specify a custom path if preferred.

/

Up to 1024 characters allowed.

▶ **Advanced health check settings**

## Attributes

Certain default attributes will be applied to your target group. You can view and edit them after creating the target group.

▶ **Tags - *optional***

Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.

Cancel Next

## Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.

**Available instances** (0)

Filter resources by property or value

1

| Instance ID | Name | State | Security groups | Zone | Su |
|---|---|---|---|---|---|

No Available instances

**0 selected**

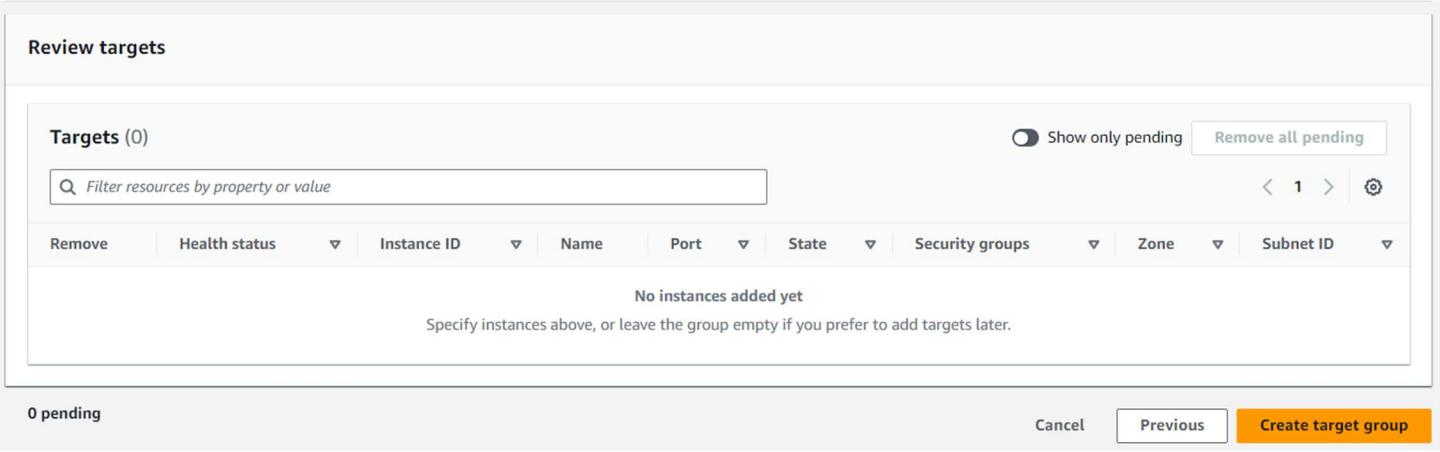
Ports for the selected instances

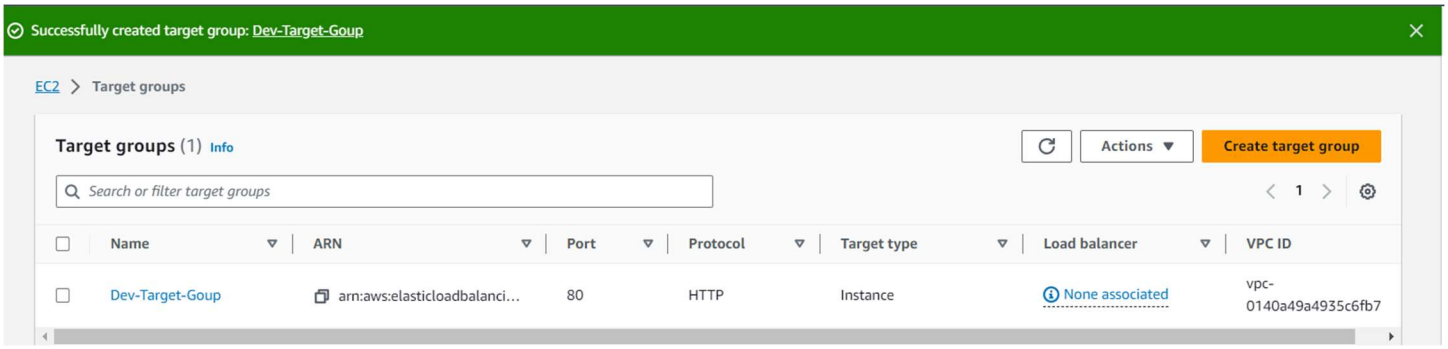
Ports for routing traffic to the selected instances.

80

1-65535 (separate multiple ports with commas)

Include as pending below




Target group was successfully created.

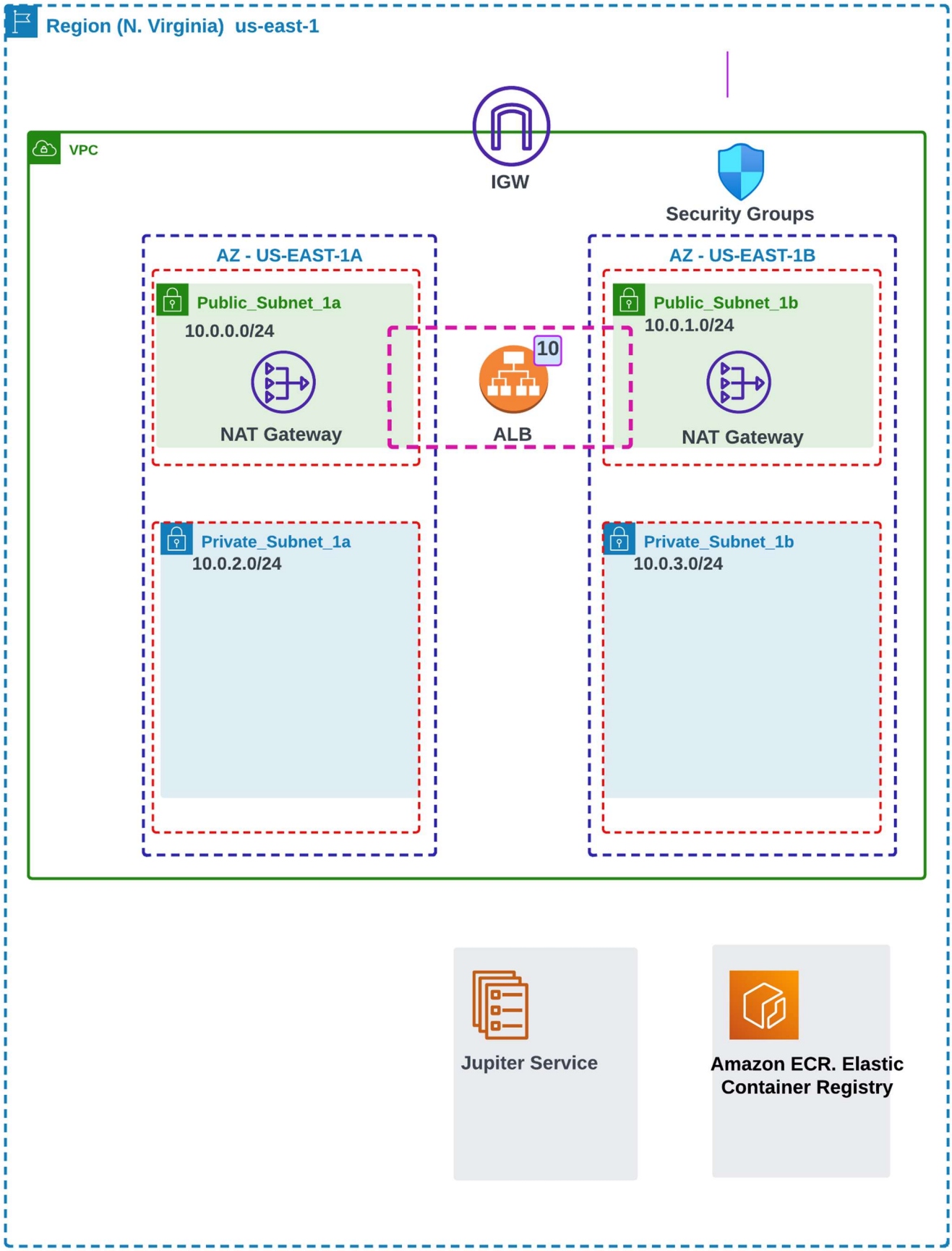
## STEP 28:

## CREATION OF APPLICATION LOAD BALANCER (ALB)

The main task here will be to create an Application Load Balancer "**ALB"** that will be used to route traffic to the Fargate containers in the **Private_App_Subnets**

**IMPORTANT:** Very important that the Application Load balancer **(ALB)** should be associated with Public Subnets **(Public_Subnet_AZ1)** and **(Public_Subnet_AZ2)**

The reason for this is because, Internet traffic flows into both subnets through the internet gateway **(IGW),** the ALB being associated with the public subnets, will then be contaminated with the traffic. The **ALB** will then wire this traffic from both Public Subnet straight to the **(Private_App_Subnet_AZ1)** and **(Private_Data_Subnet_AZ1)** and to the corresponding **(Private_App_Subnet_AZ2)** and **(Private_Data_Subnet_AZ2)** as well.




10 Application Load Balancer is used to distribute Web traffic to the containers

Let's follow the steps below to create our **Application Load Balancer**

Navigate to **EC2**

Navigate to **Load Balancers**

Click on **“Create Load Balancers”**

Select **Application Load Balancer**

Click on **Create**

Enter Load balancer name

Scroll down to the Network Mapping section

Click on the drop down to select **Dev_VPC**

Select the **us-east-1a** Availability Zone (AZ)

Click on the dropdown to select the **Public_Subnet_AZ1**

Select the **us-east-1b** Availability Zone (AZ)

Click on the dropdown to select the **Public_Subnet_AZ2**

Scroll down to **Security group**

Remove the default security group

Select **ALB_security_group**

Click on **Listener HTTP:80**

Under default action, select the **Dev-Target-Group**

Leave all other configurations as default

Click on **"Create Load Balancer"**

# Create Application Load Balancer Info

The Application Load Balancer distributes incoming HTTP and HTTPS traffic across multiple targets such as Amazon EC2 instances, microservices, and containers, based on request attributes. When the load balancer receives a connection request, it evaluates the listener rules in priority order to determine which rule to apply, and if applicable, it selects a target from the target group for the rule action.

▶ **How Elastic Load Balancing works**

## Basic configuration

Load balancer name

Name must be unique within your AWS account and can't be changed after the load balancer is created.

Dev-ALB

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Scheme | Info

Scheme can't be changed after the load balancer is created.

- (•) Internet-facing
  An internet-facing load balancer routes requests from clients over the internet to targets. Requires a public subnet. Learn more
- ( ) Internal
  An internal load balancer routes requests from clients to targets using private IP addresses.

IP address type | Info

Select the type of IP addresses that your subnets use.

- (•) IPv4
  Recommended for internal load balancers.
- ( ) Dualstack
  Includes IPv4 and IPv6 addresses.

## Network mapping Info

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC | Info

Select the virtual private cloud (VPC) for your targets or you can create a new VPC. Only VPCs with an internet gateway are enabled for selection. The selected VPC can't be changed after the load balancer is created. To confirm the VPC for your targets, view your target groups.

Dev_VPC
vpc-0140a49a4935c6fb7
IPv4: 10.0.0.0/16

Mappings | Info

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.

☑ **us-east-1a (use1-az4)**

Subnet

subnet-0b0ed41dfd332fc2c Public_Subnet_AZ1

IPv4 address

Assigned by AWS

☑ **us-east-1b (use1-az6)**

Subnet

subnet-090cf9eb2abcad569 Public_Subnet_AZ2

IPv4 address

## Security groups Info

A security group is a set of firewall rules that control the traffic to your load balancer. Select an existing security group, or you can create a new security group.

Security groups

*Select up to 5 security groups*

ALB_security_group
sg-0c63930ed5f68a72d VPC: vpc-0140a49a4935c6fb7

## Listeners and routing Info

A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

▼ Listener **HTTP:80** Remove

| Protocol | Port | Default action Info |
|---|---|---|
| HTTP | 80 (1-65535) | Forward to: Dev-Target-Goup, Target type: Instance, IPv4, HTTP |

Create target group

Listener tags - *optional*

Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.

Add listener tag

You can add up to 50 more tags.

## Summary

Review and confirm your configurations. Estimate cost

### Basic configuration Edit

Dev-ALB

- Internet-facing
- IPv4

### Security groups Edit

- ALB_security_group
  sg-0c63930ed5f68a72d

### Network mapping Edit

VPC vpc-0140a49a4935c6fb7
Dev_VPC

- us-east-1a
  subnet-0b0ed41dfd332fc2c
  Public_Subnet_AZ1
- us-east-1b
  subnet-090cf9eb2abcad569
  Public_Subnet_AZ2

### Listeners and routing Edit

- HTTP:80 defaults to
  Dev-Target-Goup

### Add-on services Edit

*None*

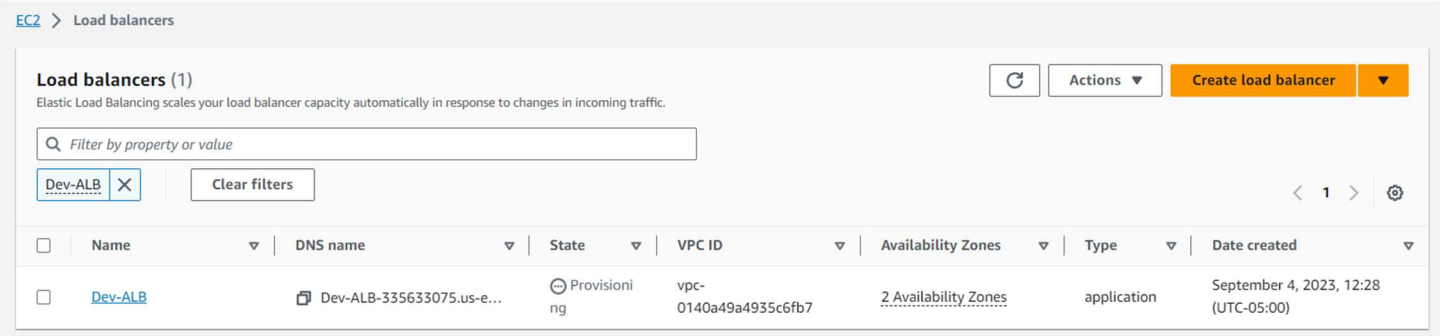
### Tags Edit

*None*

### Attributes

Certain default attributes will be applied to your load balancer. You can view and edit them after creating the load balancer.

Cancel **Create load balancer**




Load balancer was successfully created

Our **Application Load Balancer** will route incoming traffic to the containers that will be created in the **(Private_App_Subnet_AZ1**) and **(Private_App_Subnet_AZ2)**

**IMPORTANT**

When we create the **ECS service** that will create the container, will create a new Listener and Target group that our ALB will route incoming traffic.

At this point we will **delete** the Listener and **Dev-Target-Group**, that was initially created because we don't need it anymore.

**Stage A**

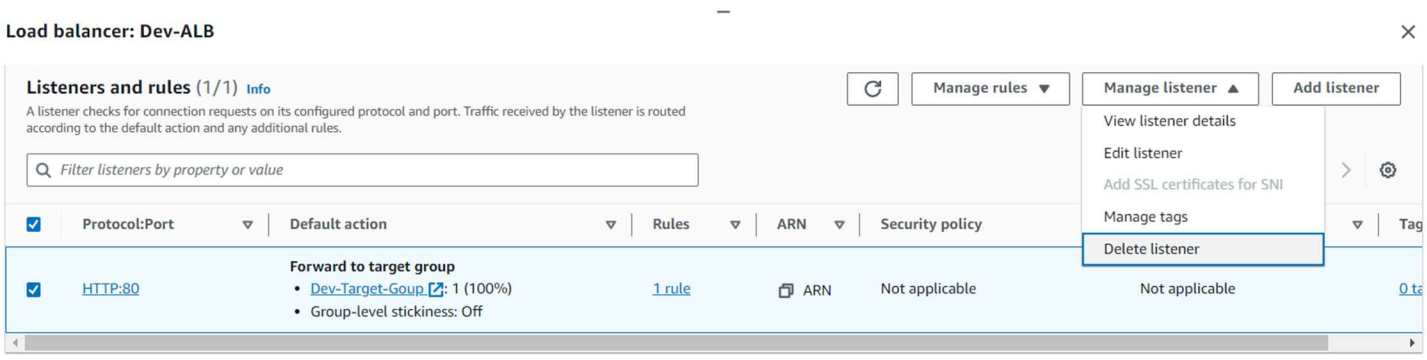
Click on “**Load Balancer”**

Click on “**Listeners and rules”**

Select “**Target group”**

Click on “**Manage Listener”**

Click on “**Delete listener”**



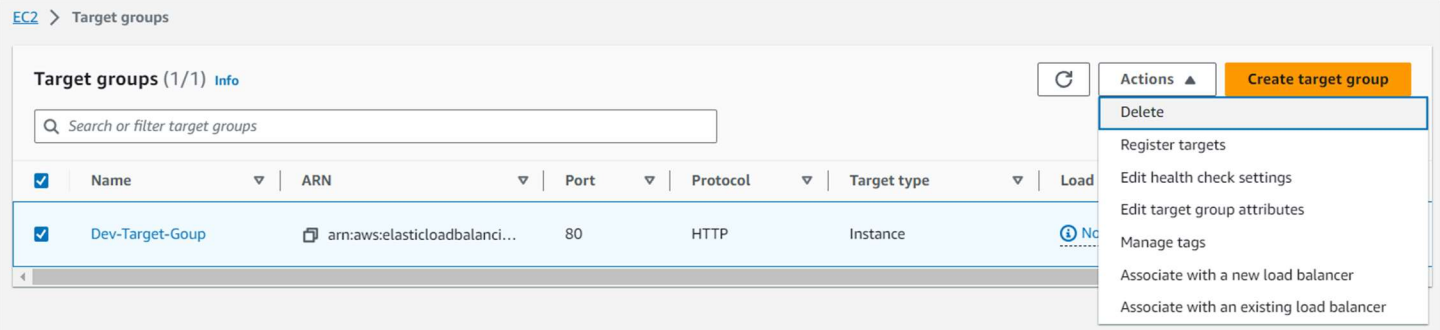
**Stage B**

Click on “**Target Groups”**

Select the “**Dev-Target-Group”**

Click on “**Actions”**

Click on Delete and **“Confirm to delete”**




## STEP 29:

## CREATION OF ECS CLUSTER

Previously, the jupiter image was created and pushed it to the **ECR repository.**

Now, we are ready to run our **ECS Fargate** containers

To run the **ECS Fargate** containers, the first step will be to create an "**ECS cluster**"

Let's navigate to **Elastic Container Service (ECS)** in the AWS management console

Click on **Get Started**

Click on **"Cluster"** on the left pane of the screen

Click on **"Create Cluster"**

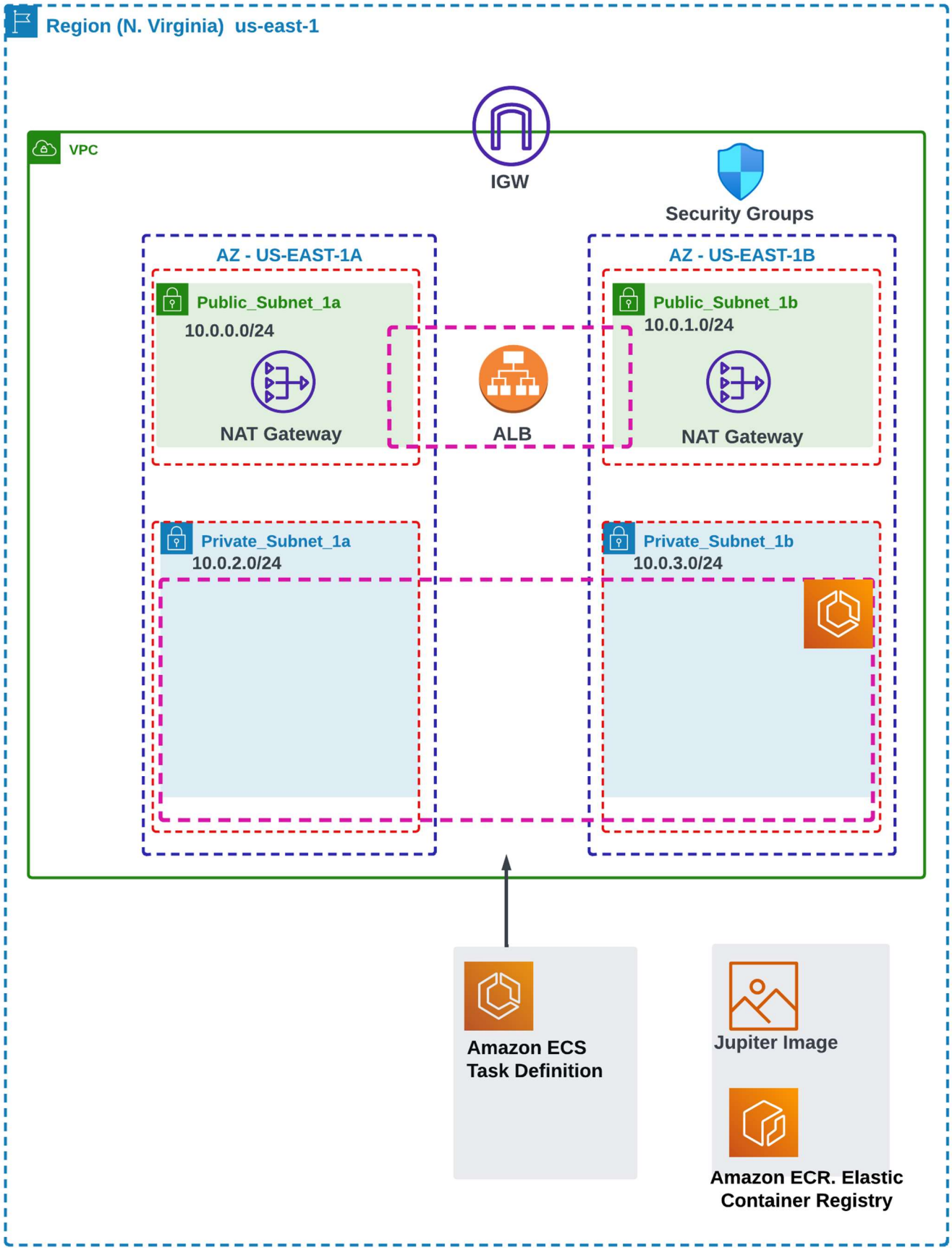
Let's call our cluster "**jupiter-cluster"**

Click on the **"Infrastructure"** dropdown

Select **AWS Fargate** (Serverless)

Scroll down to the bottom of the page

Click on **"Create Cluster"**




11 Create an ECS Cluster

# Create cluster Info

An Amazon ECS cluster groups together tasks, and services, and allows for shared capacity and common configurations. All of your tasks, services, and capacity must belong to a cluster.

## Cluster configuration

Cluster name

jupiter-cluster

There can be a maximum of 255 characters. The valid characters are letters (uppercase and lowercase), numbers, hyphens, and underscores.

Default namespace - *optional*

Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

jupiter-cluster

## Infrastructure Info

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances, or external instances using ECS Anywhere.

Serverless

- [x] AWS Fargate (serverless)
  Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead.
  The cluster has Fargate and Fargate Spot capacity providers by default.
- [ ] Amazon EC2 instances
  Manual configurations. Use for large workloads with consistent resource demands.
- [ ] External instances using ECS Anywhere
  Manual configurations. Use to add data center compute.

## Monitoring - *optional* Info

Container Insights is off by default. When you use Container Insights, there is a cost associated with it.

## Tags - *optional* Info

Tags help you to identify and organize your clusters.

Cancel Create

## STEP 30:

## CREATION OF TASK DEFINITION

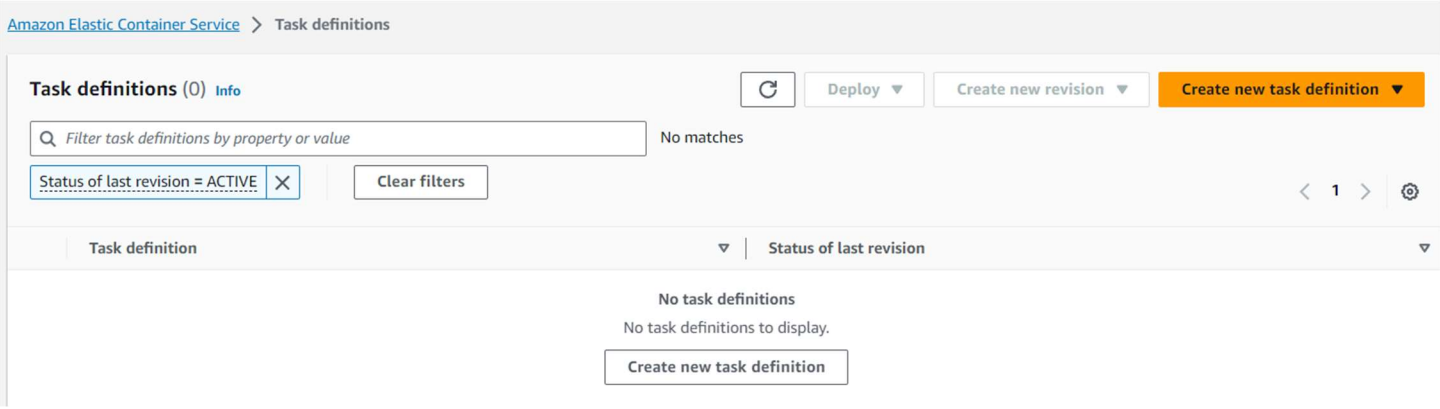
Navigate to the **Amazon Elastic Container Service console (ECS)**

Click on **Task definitions**

Click on **“Create new task definition”**

Give your Task definition a **name**

Let's call the Task definition **"jupiter-task-definition"**





Click on the **“Infrastructure requirements”**

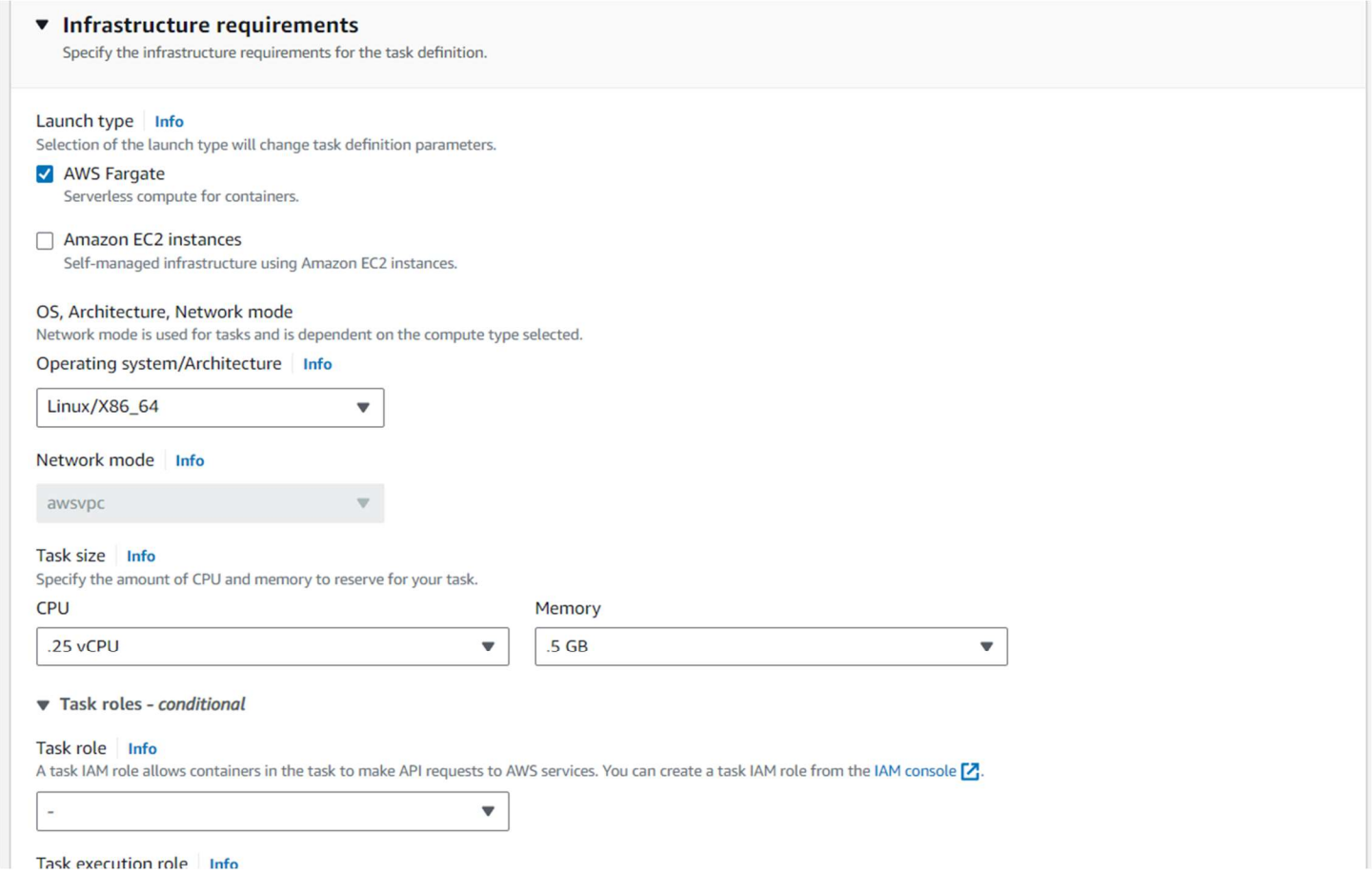
Select **AWS Fargate**

Leave the Operating System Architecture as **Linux/x86_64**

Under Task size

Click on **CPU** dropdown and select **0.25 vCPU** option

Click on memory dropdown and select **0.50 GB**




Scroll down to **Container - 1**

Enter **"jupiter"** as container name

Remember that the **"jupiter"** container image (has already been created and push to the Elastic Container Repository **(ECR)**

Let's enter our container image **URI**

Right click on the open tab and select **"Duplicate"** to open a new tab

Enter ECR to search for **(Elastic Container Service)**

Click on the **"Elastic Container Service"** to open

**IMPORTANT** to remember that our **"jupiter"** container image is in our **ECR** repository

**Copy** the **URI** for jupiter image

**Paste** the **URI** in the space for the **Image URI**

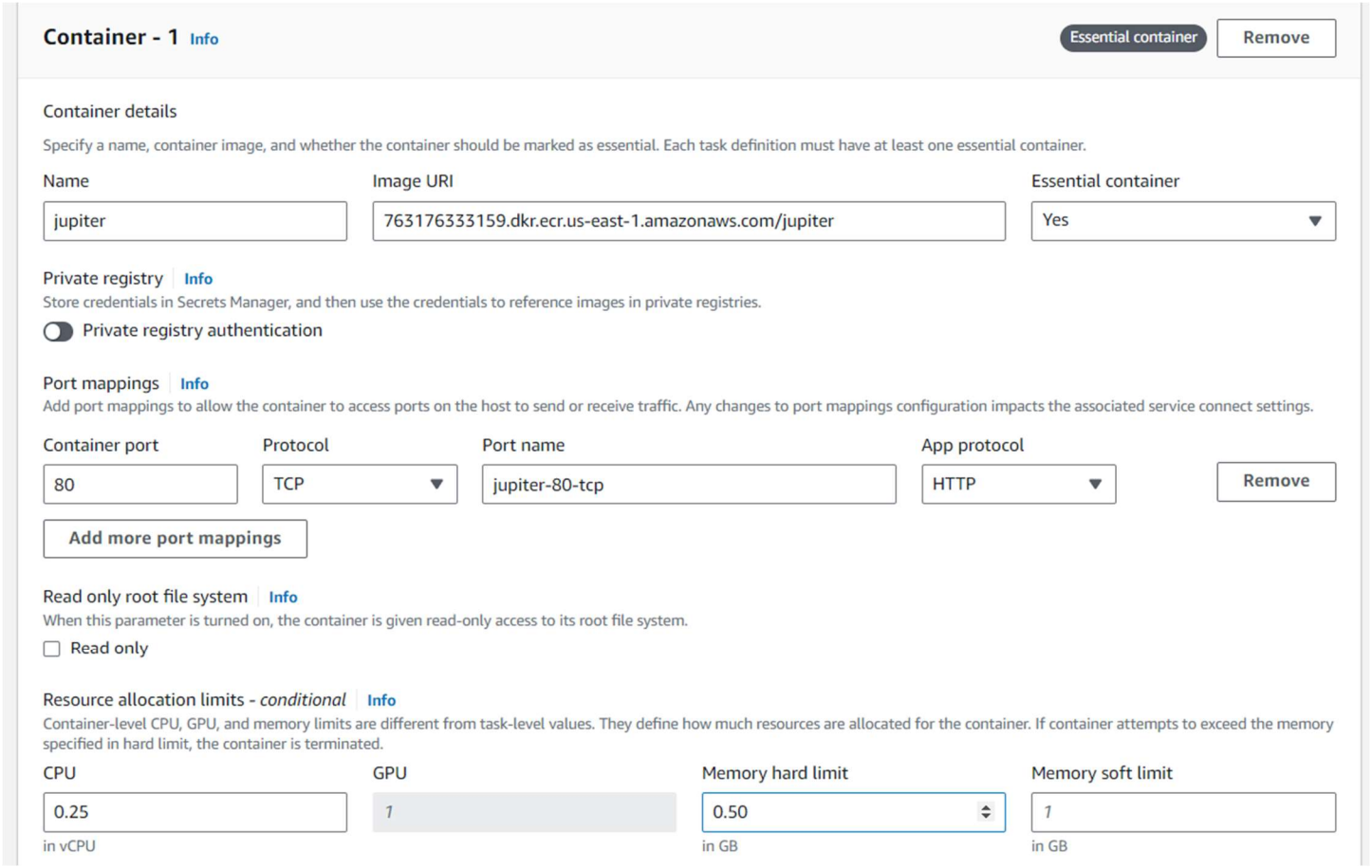
**Under Port mapping:**

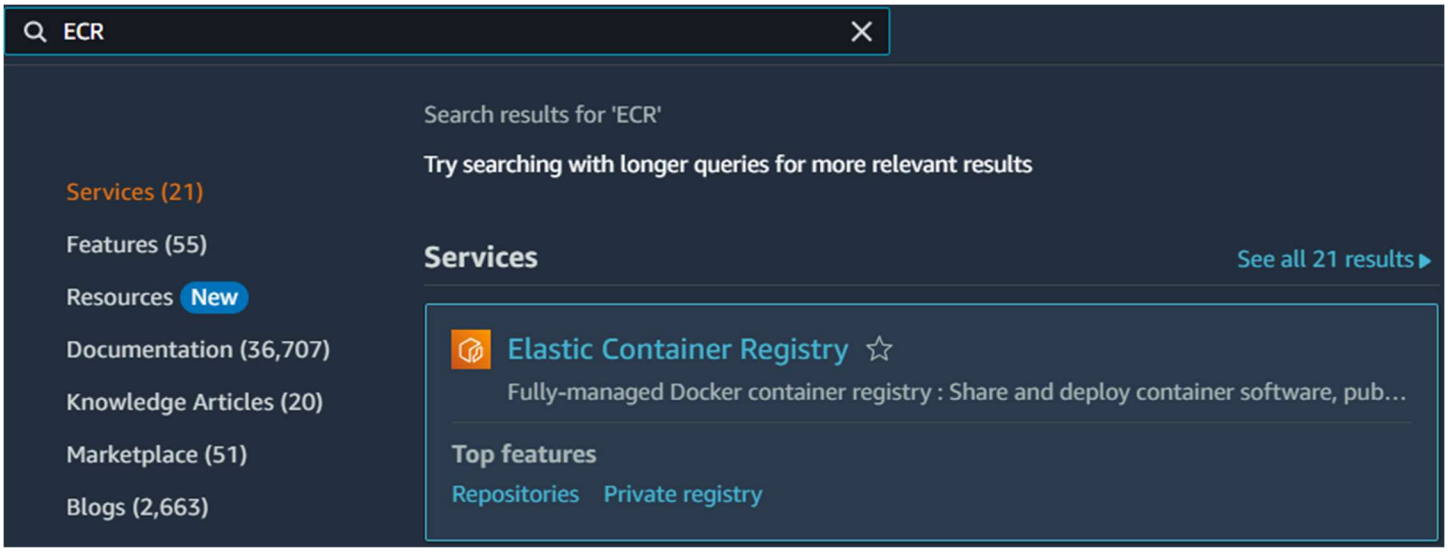
**Container port 80,** protocol will be **TCP,** and port name will be **jupiter-80-tcp**

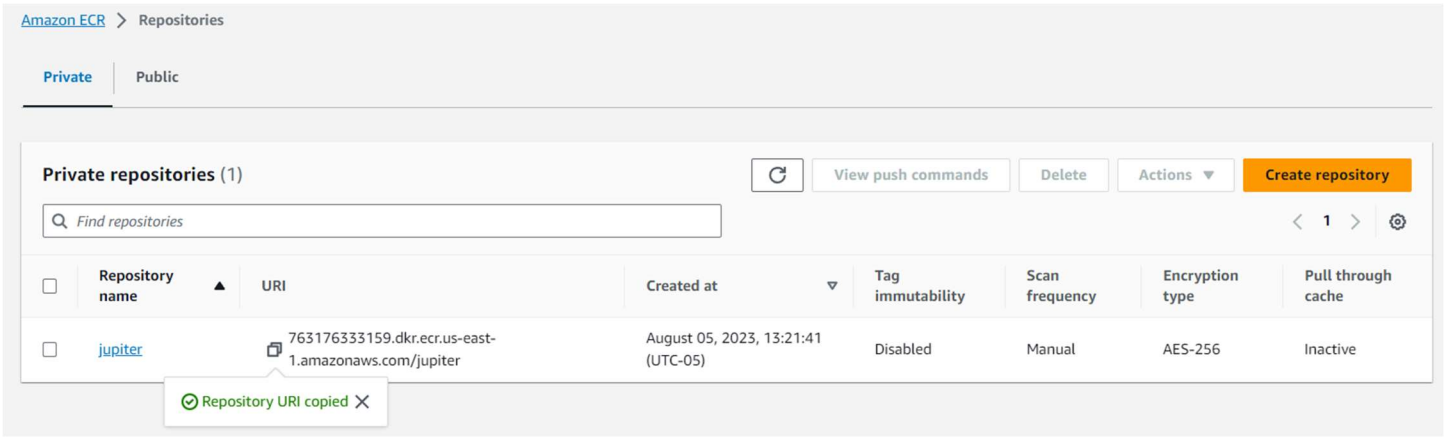
Configure **CPU = 0.25 vCPU** and Memory hard limit = **0.50 GB** (Same as earlier configurations)

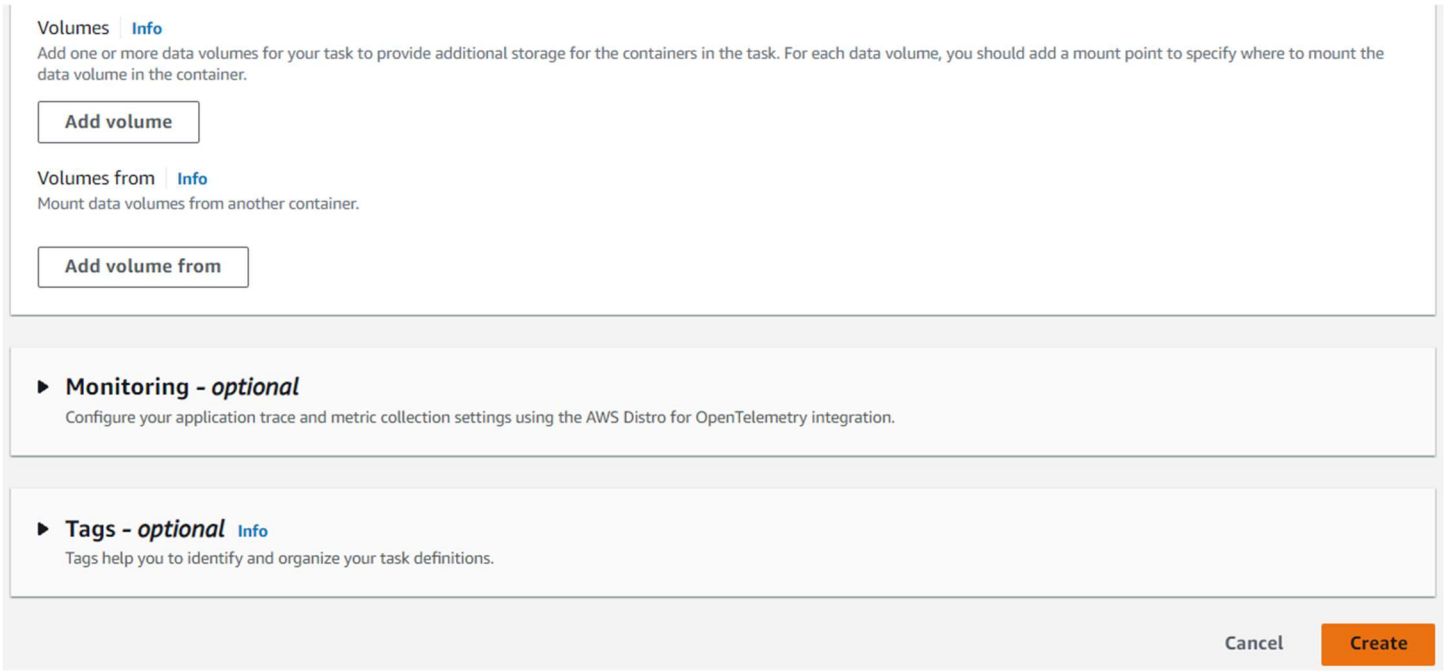
Scroll down to the bottom of the page leaving all other configurations as default

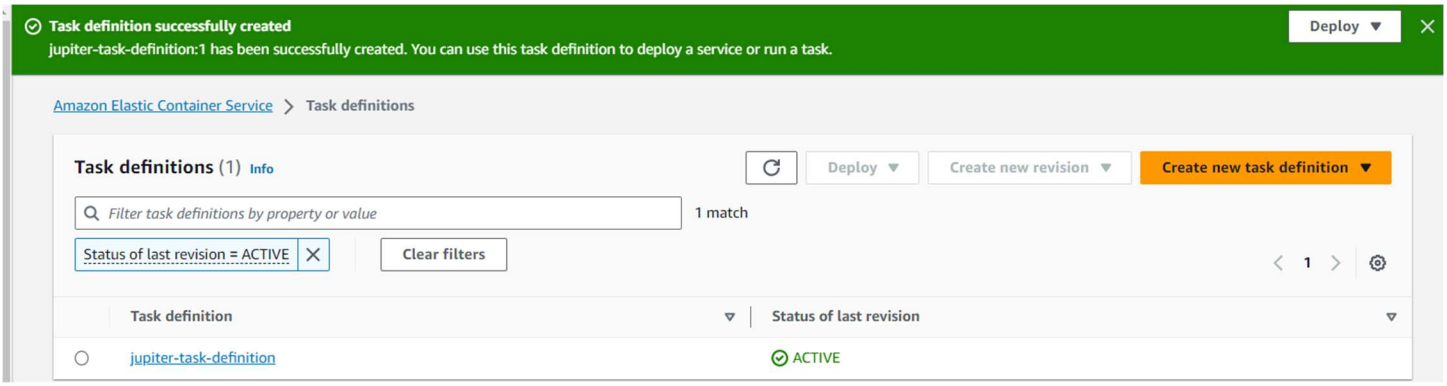
Click on **"Create"**






**jupiter-task-definition** was successfully created.

## STEP 31:

## CREATION OF SERVICE

The main task here will be to deploy a "**Service**" for our **ECS** Cluster

ECS Service is how we start our containers

Let's navigate to the **ECS** console

Click on “**Cluster**”

Click on "**jupiter-cluster**" that we created previously.

Click on the "**Services**" tab

Click on "**Create**"

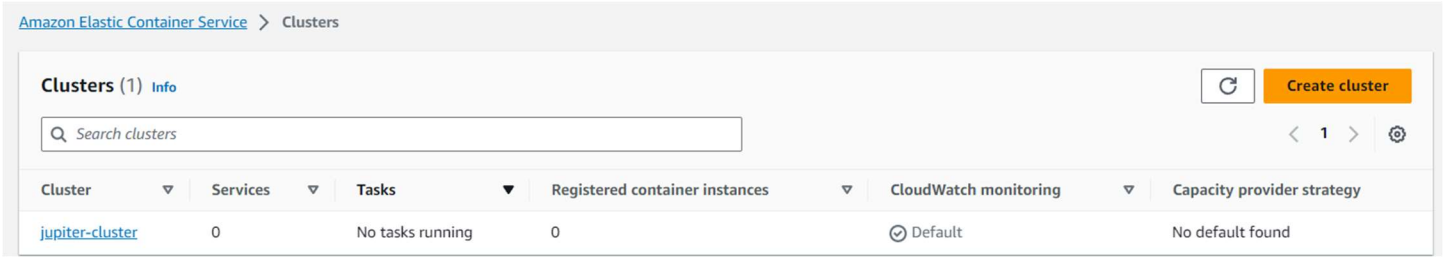
Our existing cluster **"jupiter-cluster**" is automatically selected

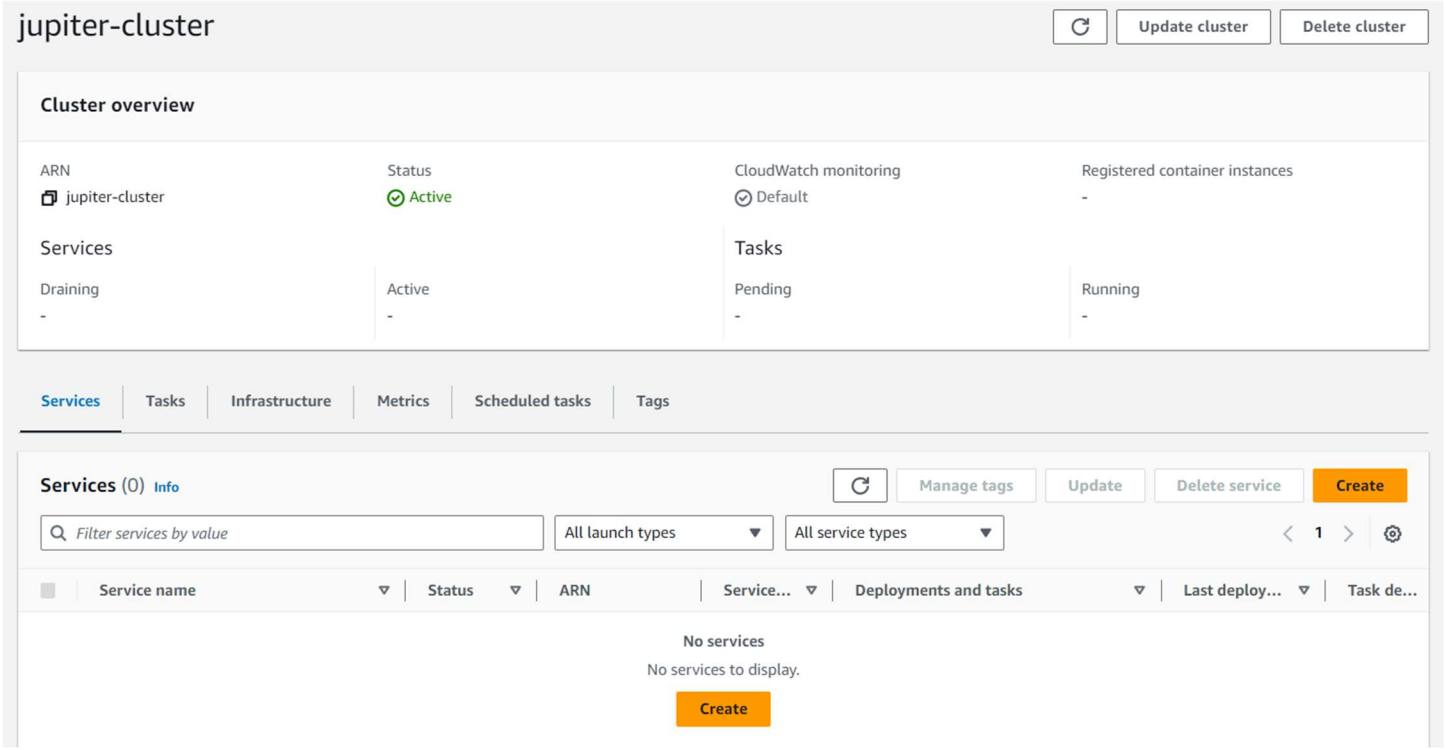
Scroll down to "**Deployment Configuration**"

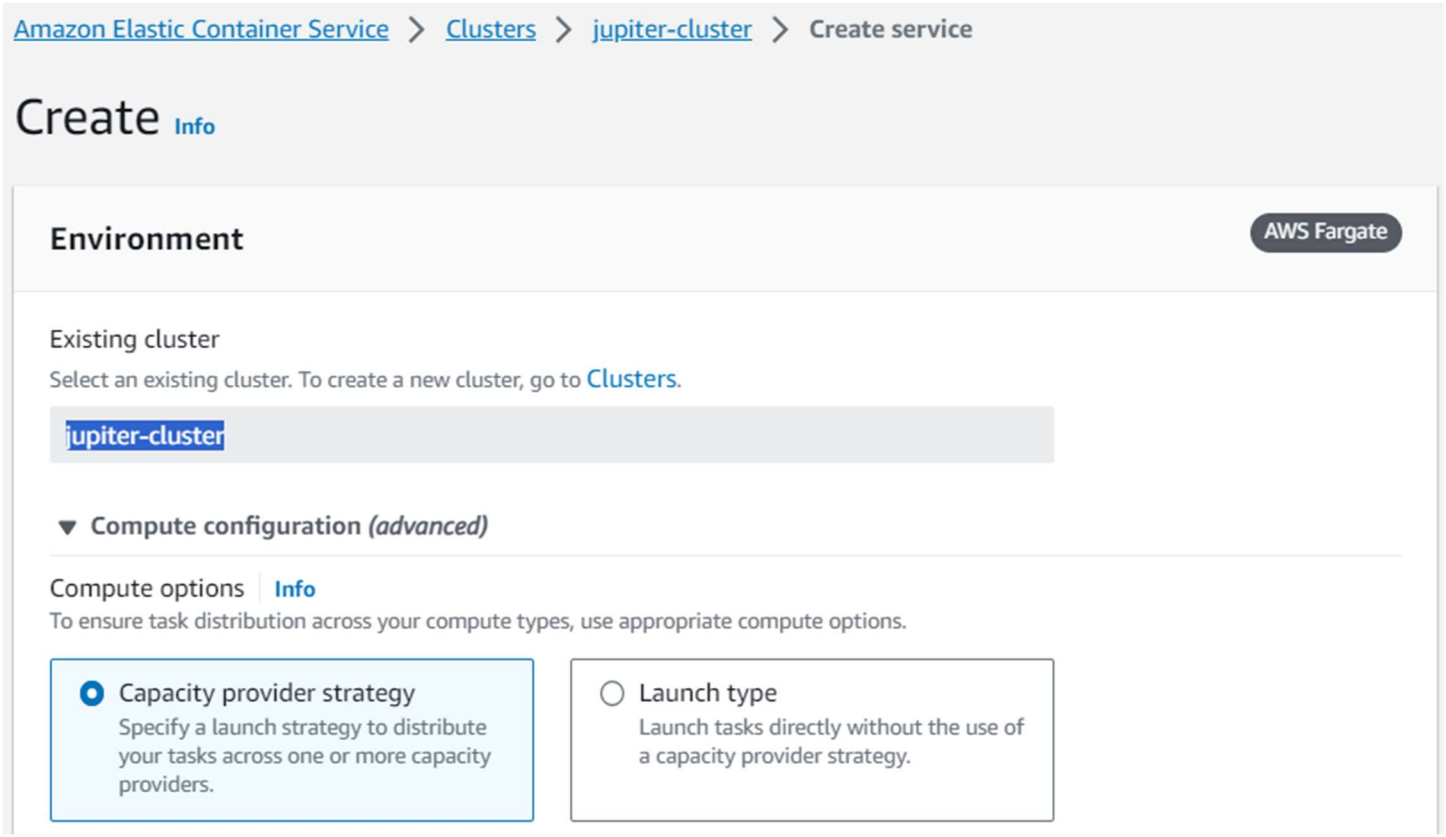
Select "**Service**"

Under Family: select "**jupiter-task-definition**"

Under service name: let's call it "**jupiter-service**"


jupiter-cluster
Update cluster
Delete cluster
Cluster overview
ARN
jupiter-cluster
Status
Active
CloudWatch monitoring
Default
Registered container instances
-
Services
Draining
-
Active
-
Tasks
Pending
-
Running
-
Services
Tasks
Infrastructure
Metrics
Scheduled tasks
Tags
Services (0) Info
Manage tags
Update
Delete service
Create
Filter services by value
All launch types
All service types
1
Service name
Status
ARN
Service...
Deployments and tasks
Last deploy...
Task de...
No services
No services to display.
Create


Amazon Elastic Container Service
Clusters
jupiter-cluster
Create service
Create Info
Environment
AWS Fargate
Existing cluster
Select an existing cluster. To create a new cluster, go to Clusters.
jupiter-cluster
Compute configuration (advanced)
Compute options Info
To ensure task distribution across your compute types, use appropriate compute options.
Capacity provider strategy
Specify a launch strategy to distribute your tasks across one or more capacity providers.
Launch type
Launch tasks directly without the use of a capacity provider strategy.

Capacity provider strategy | Info
Select either your cluster default capacity provider strategy or select the custom option to configure a different strategy.

- Use cluster default
- Use custom (Advanced)

Capacity provider: FARGATE

Base: 0

Weight: 1

Add more

Platform version | Info
Specify the platform version on which to run your service.

LATEST

## Deployment configuration

Application type | Info
Specify what type of application you want to run.

- Service
  Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.
- Task
  Launch a standalone task that runs and terminates. For example, a batch job.

Task definition
Select an existing task definition. To create a new task definition, go to Task definitions.

- [ ] Specify the revision manually
  Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family: jupiter-task-definition

Revision: 1 (LATEST)

Service name
Assign a unique name for this service.

jupiter-service

Service type | Info
Specify the service type that the service scheduler will follow.

- Replica
  Place and maintain a desired number of tasks across your cluster.
- Daemon
  Place and maintain one copy of your task on each container instance.

Under desired tasks: Let's select 2 (**This means we want 2 containers running**)

Select "**Deployment options**" dropdown: but let's leave it as **default**

Click on **Networking**, select the "**Dev-VPC**"

Click on Subnet and select the **"private_App_Subnet_AZ1"** and **"Private_App_Subnet_AZ2"**

Click on security group tab, the select the "**Container_security_group**"

**Remove** the **default security group**, and **disable** the "**Public IP"**

Under Load balancing:

Select "**Application Load Balancer**"

Select "**Use an existing load Balancer**"

Select the "**Dev-ALB**"

**RECALL:** When we initially created the **Dev-ALB**, we **deleted** the **Listener** and **Target group**

This is where our "**Service"** will create a new **Listener** and **Target group** to replace the initially deleted ones

Under listener, it will be **Port 80** and Protocol will be **HTTP**

Let's name our Target group as **Dev-Target-Group** and Protocol will be **HTTP**

Health check path and Health check grace period will be left as default

Click on **Service auto scaling**

Select "**Use service auto scaling**"

Select Minimum number of Task as 1

Select Maximum number of Task as 2

Let's leave Scaling Policy as "**Target Tracking**"

Under Policy name: Let's call it "**jupiter**"

Click on the dropdown under ECS service metric

Select "**ECSServiceAverageCPUUtilization**"

Under Target value: Let's enter 70 (default value)

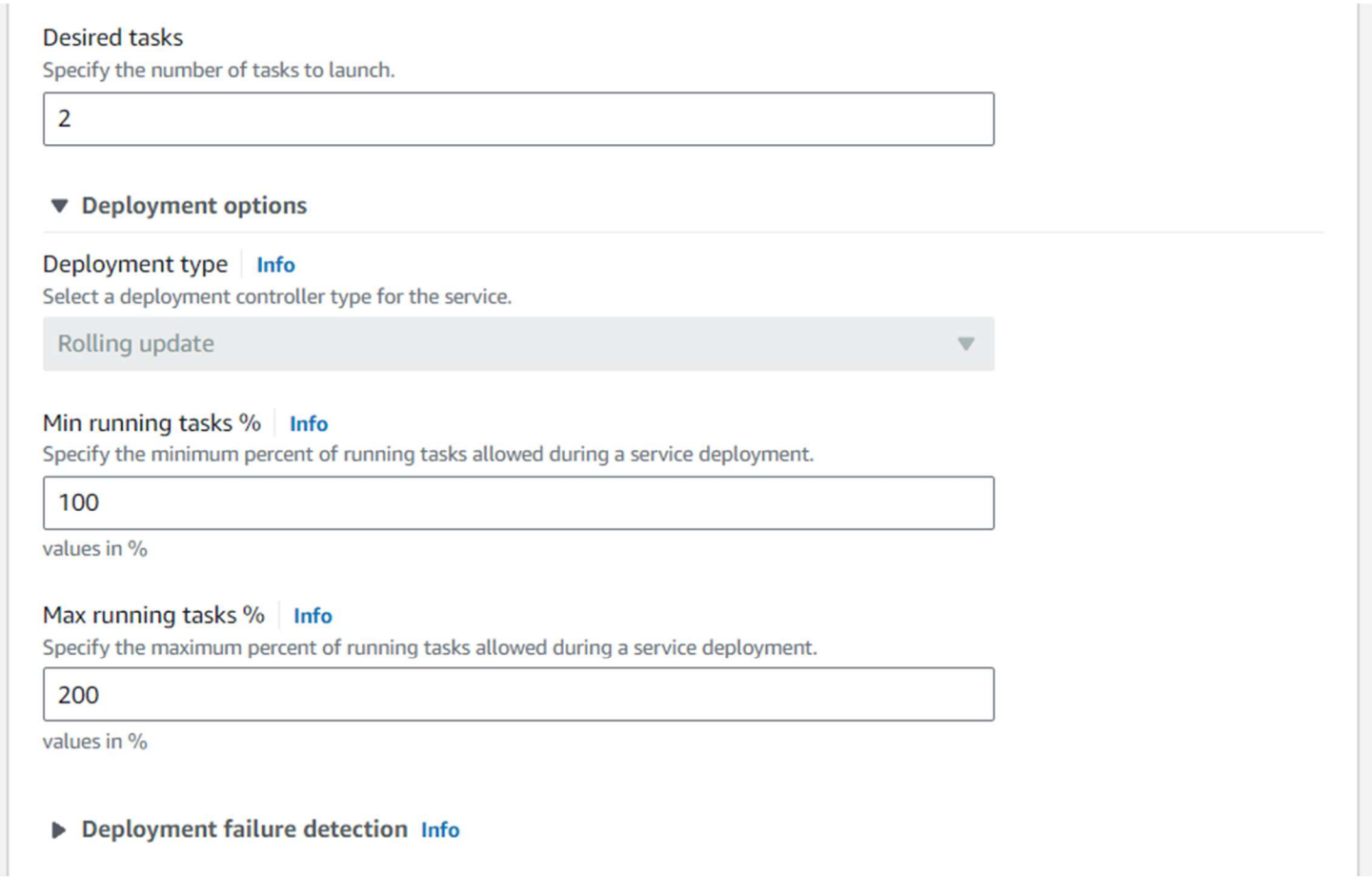
Under Scale-out cooldown period: Let's enter 300 (default value)

Under Scale-in cooldown period: Let's enter 300 (default value)

Let's leave Tags as default

Click on "**Create**"

It will take a few minutes for our "**jupiter-service**" to be created



## Networking

VPC | Info
Choose the Virtual Private Cloud to use.

vpc-0140a49a4935c6fb7
Dev_VPC

Subnets
Choose the subnets within the VPC that the task scheduler should consider for placement.

*Choose subnets*

subnet-0e7c62f6744b1c122
Private_App_Subnet_AZ1
us-east-1a 10.0.2.0/24

subnet-03730ea19f9ba4316
Private_App_Subnet_AZ2
us-east-1b 10.0.3.0/24

Security group | Info
Choose an existing security group or create a new security group.

- (•) Use an existing security group
- ( ) Create a new security group

Security group name
Choose an existing security group.

*Choose security groups*

sg-073c704a1951b07ba
Container_security_group

Public IP | Info
Choose whether to auto-assign a public IP to the task's elastic network interface (ENI).

Turned off

## Load balancing - *optional*

Load balancer type | Info
Configure a load balancer to distribute incoming traffic across the tasks running in your service.

Application Load Balancer

Application Load Balancer
Specify whether to create a new load balancer or choose an existing one.

- ( ) Create a new load balancer
- (•) Use an existing load balancer

Load balancer
Select the load balancer you wish to use to distribute incoming traffic across the tasks running in your service.

Dev-ALB

Choose container to load balance

jupiter 80:80

Listener | Info

Specify the port and protocol that the load balancer will listen for connection requests on.

- (•) Create new listener
- ( ) Use an existing listener

Port: 80

Protocol: HTTP

Target group | Info

Specify whether to create a new target group or choose an existing one that the load balancer will use to route requests to the tasks in your service.

- (•) Create new target group
- ( ) Use an existing target group

Target group name: Dev-Target-Group

Protocol: HTTP

Health check path | Info: /

Health check protocol: HTTP

Health check grace period | Info: 0

seconds

## ▼ Service auto scaling - *optional*

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your service auto scaling configuration at any time to meet the needs of your application.

- [x] Use service auto scaling

Configure service auto scaling to adjust your service's desired count

Minimum number of tasks

The lower boundary to which service auto scaling can adjust the desired count of the service.

1

Maximum number of tasks

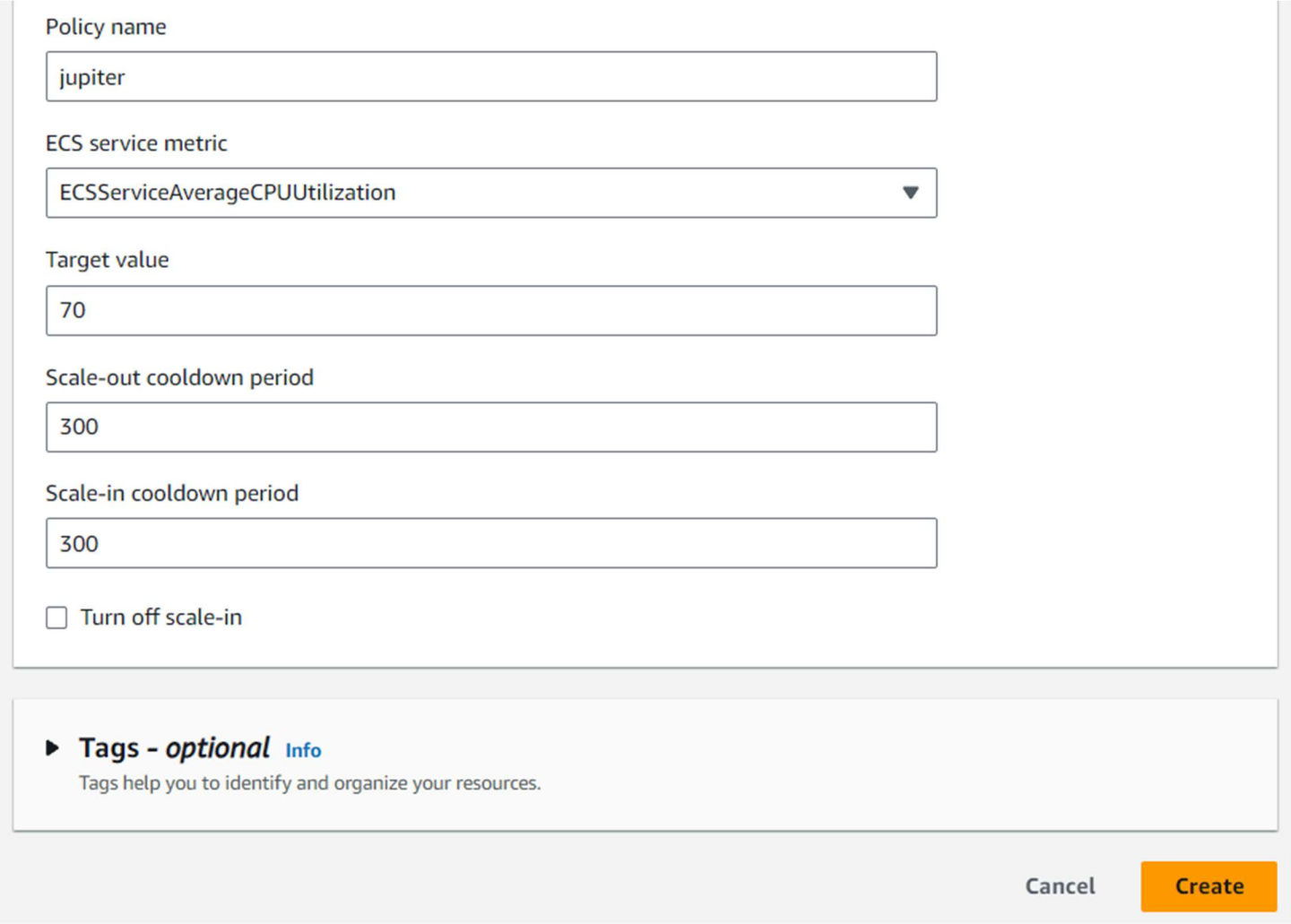
The upper boundary to which service auto scaling can adjust the desired count of the service.

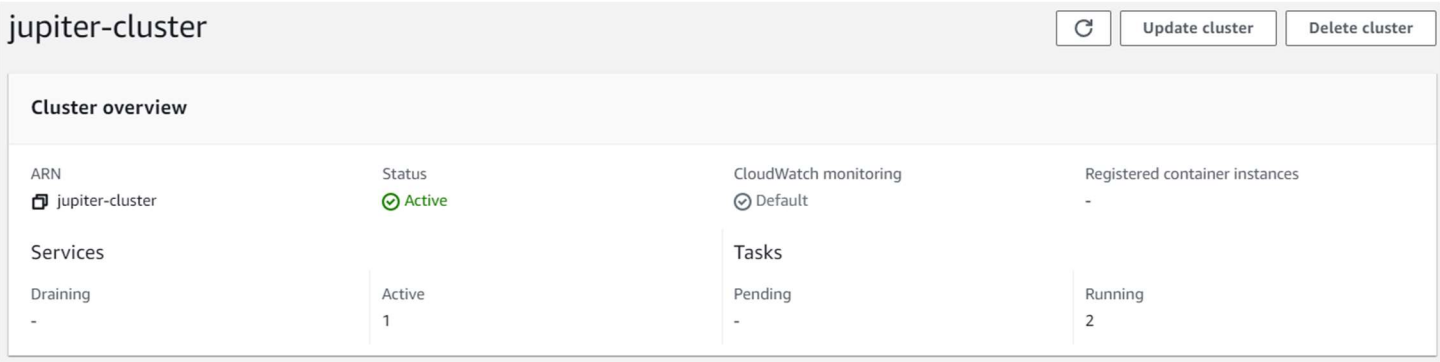
2

Scaling policy type | Info

Create either a target tracking or step scaling policy.

- (•) **Target tracking**
  Increase or decrease the number of tasks that your service runs based on a target value for a specific metric.
- ( ) **Step scaling**
  Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.



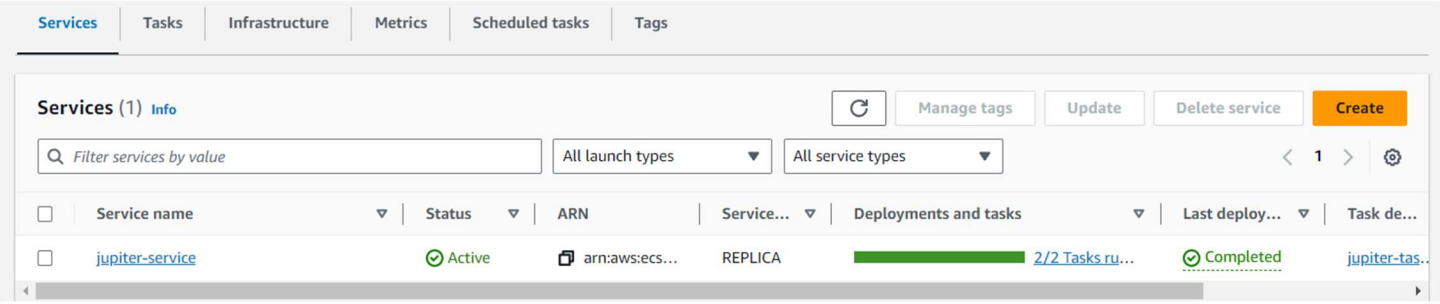




**"jupiter-service"** was successfully created.

Now Click on "**jupiter-service"**

**Verify that:**

**Deployment and task** status is **completed**




Click on **“jupiter-service”**

Under **“Health and metrics”**

**Verify that:**

Deployment Current State = **1 Completed**

Status = **Active**

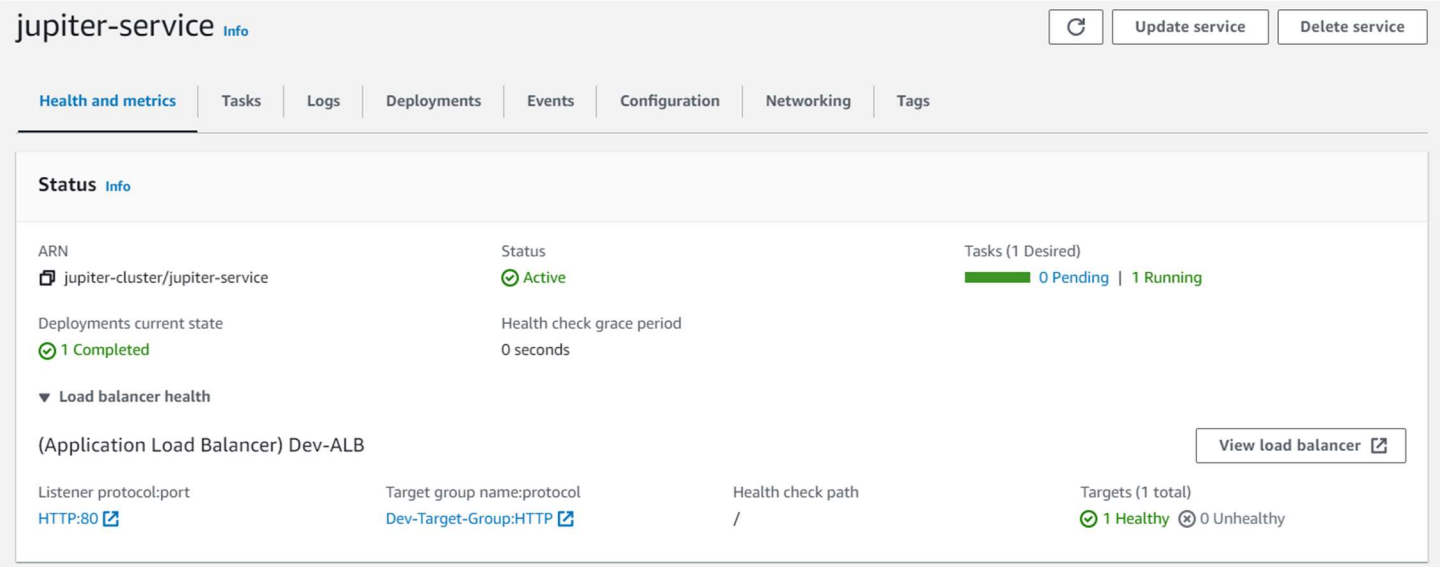
Task = (**Desired = 1 Running)**

Click on the **Dev-Target-Group:HTTP**

Total Targets = 1

Healthy = 1

Unhealthy = 0



For instance, if the "**Task**" in your ECS Service is showing unhealthy,

The troubleshooting steps is shown below

Click on the "**Event**" tab to display all events and activities

For any task that is unhealthy, you will clearly see the notification that states:

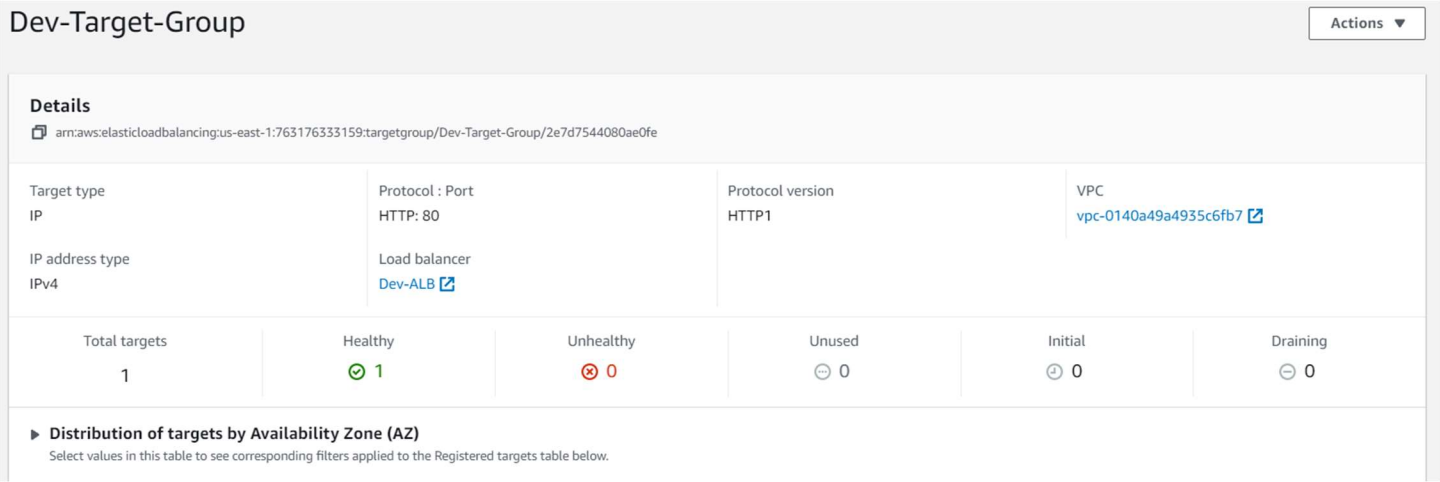
service jupiter-service has started *: **task acd1234567898367...**

Click on the **task number,** and it will open the details and show why the task is failing

Troubleshooting the issue will then be a possibility.

**Healthy is 1**

**Unhealthy is 0**




Now, that our **jupiter-service** is running successfully

The next step will be to use the DNS name of the application load balancer **Dev-ALB** to access our **jupiter website**

Right click on the **open** tab and select "**Duplicate**" to open a new tab

Enter **EC2** to search for services

Scroll down to "**Load Balancers**"

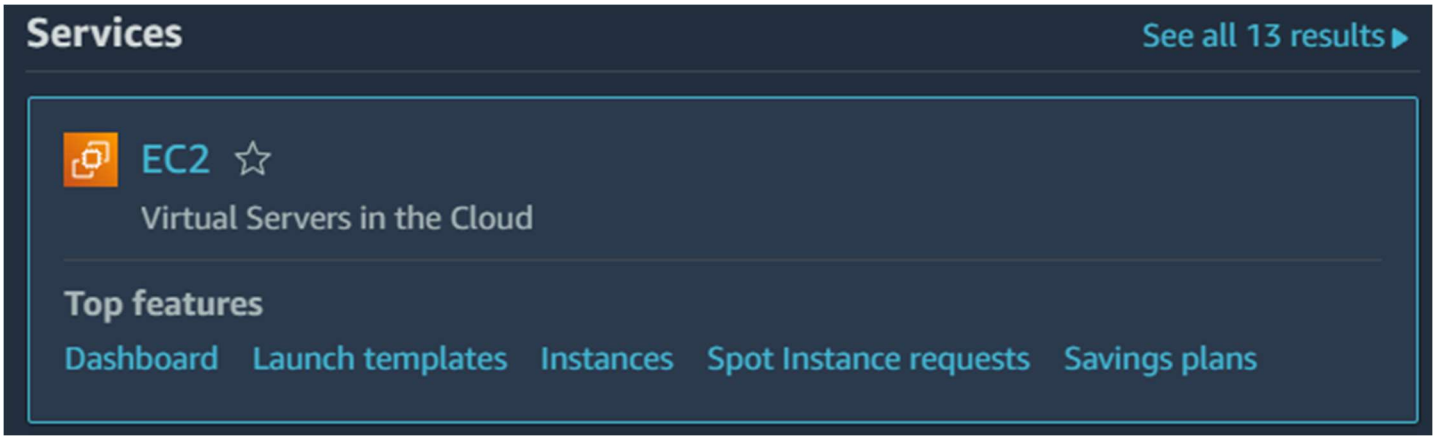
Click on **Dev-ALB**

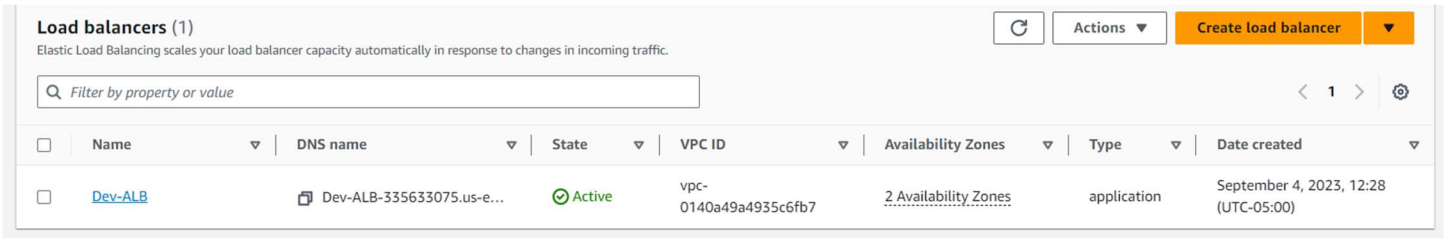
Copy the **DNS name** of the Application Load Balancer

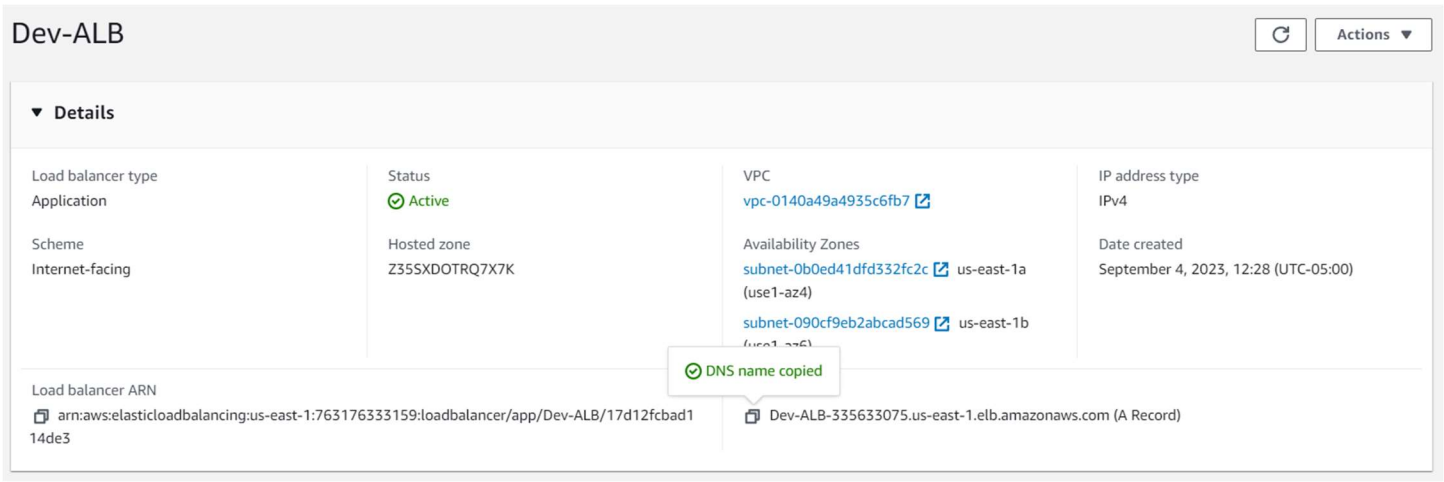
Open a new browser window/tab

Then paste the copied DNS name **"Dev-ALB-335633075.us-east-1.elb.amazonaws.com"**

Press "**Enter**"



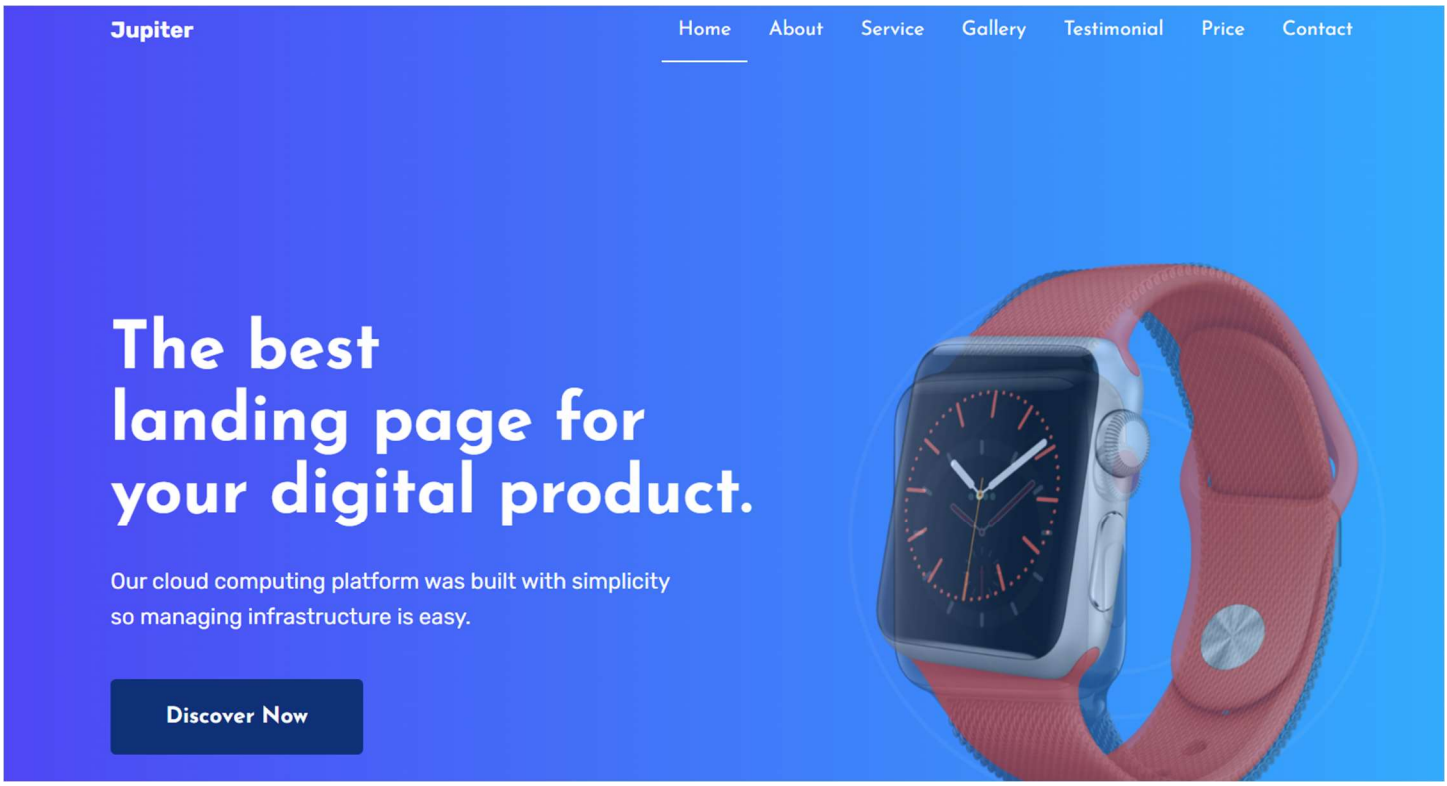




Copy the **DNS name** of the Application Load Balancer

Open a new browser window/tab

Then paste the copied DNS name **"Dev-ALB-335633075.us-east-1.elb.amazonaws.com"**

Press "**Enter**"




We were **successfully** able to access our **jupiter website** which is currently running on **ECS Fargate containers.**

## STEP 32:

## REGISTER A NEW DOMAIN NAME IN ROUTE 53

Navigate to the management console

Enter **Route 53** in the search box

Select Register domain and select "**Get Started**"

Enter your desired domain name to see if it is available

**cloudinformatics.net** is what I will be using for this project, and it is available

Click select and proceed to checkout

You can turn on or off Auto-renew (Optional)

Enter personal information

IMPORTANT: Turn on Privacy protection to protect personal information, then click "**Next**"

Review information and click on check box to accept terms and conditions

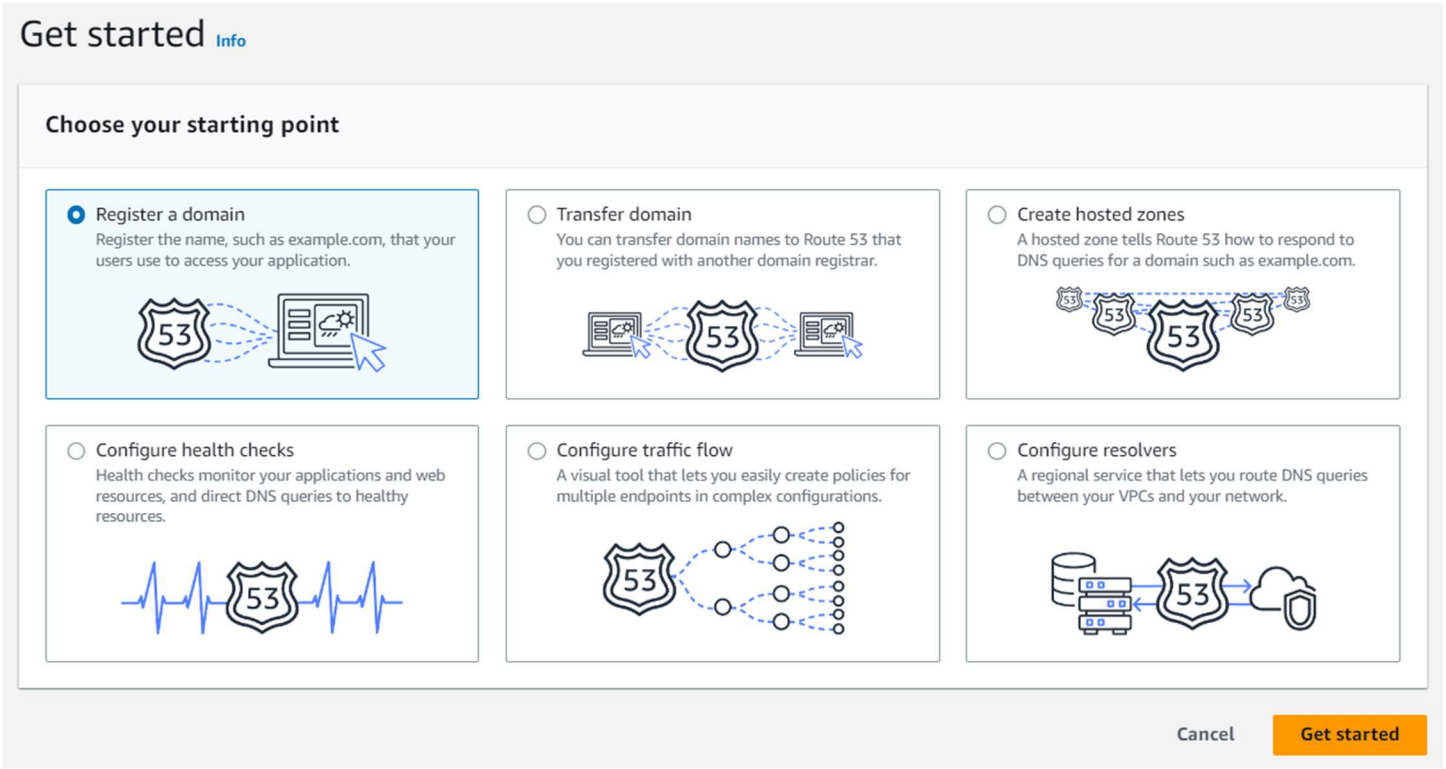
Click "**Submit**"

Registration request successfully submitted

According to AWS, domain registration might take up to 3 days to complete

But, in my case it took only about 15 minutes to be completed.

My domain name was successfully registered.

## Register domains Info

Pricing for domain names varies by top-level domain (TLD). For more information, view price with different TLDs.

### Search for domain

Check availability for a domain

cloudinformatics.net

Search

### Search result

| Domain | Price/year | |
|---|---|---|
| cloudinformatics.net Exact match | 11.00 USD | Selected |

### Selected domains (1/5)

Domain registration fee

**cloudinformatics.net**
11.00 USD

Remove

Subtotal: **11.00 USD**

The domain registration fee displayed is for 1 year. You can change duration on the next page.

Proceed to checkout

# Review and submit Info

## Step 1: Pricing

Edit

### Domain pricing options

| Domain name | Price | Year | Auto-renew |
|---|---|---|---|
| cloudinformatics.net | 11.00 USD | 1 year | Yes |

Subtotal: **11.00 USD**

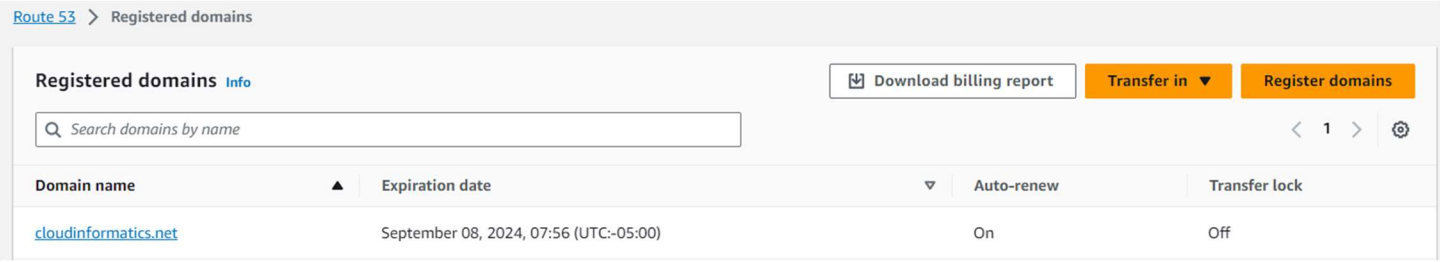
Applicable taxes will be calculated at checkout.

## Terms and conditions

Amazon Route 53 enables you to register and transfer domain names by using your AWS account. Route 53 uses Amazon Registrar and other registrar associates to perform the registration and transfer services. The registrar for your domain will periodically contact the registrant contact that you specified to verify the contact details and to renew registration. For more information, see Amazon Route 53 Domain Name Registration End User Agreement.

☑ I have read and agree to the Amazon Route 53 Domain Name Registration End User Agreement.

Cancel | Previous | Submit




My new domain name **“cloudinformatics.net”** has been successfully registered.

## STEP 33:

## CREATION OF RECORD SET IN ROUTE 53

The main task here will be to create a record set in **Route 53** to point our domain name to the **Application Load Balancer** (ALB)

Navigate to **Route 53** Dashboard

Click **Hosted zone**

Click your domain name. My domain name is **"cloudinformatics.net"**

Click on **"Create Record"**

Under "**Record name**" we are going to enter **"www.cloudinformatics.net"**

Toggle on the **"Alias”** tab

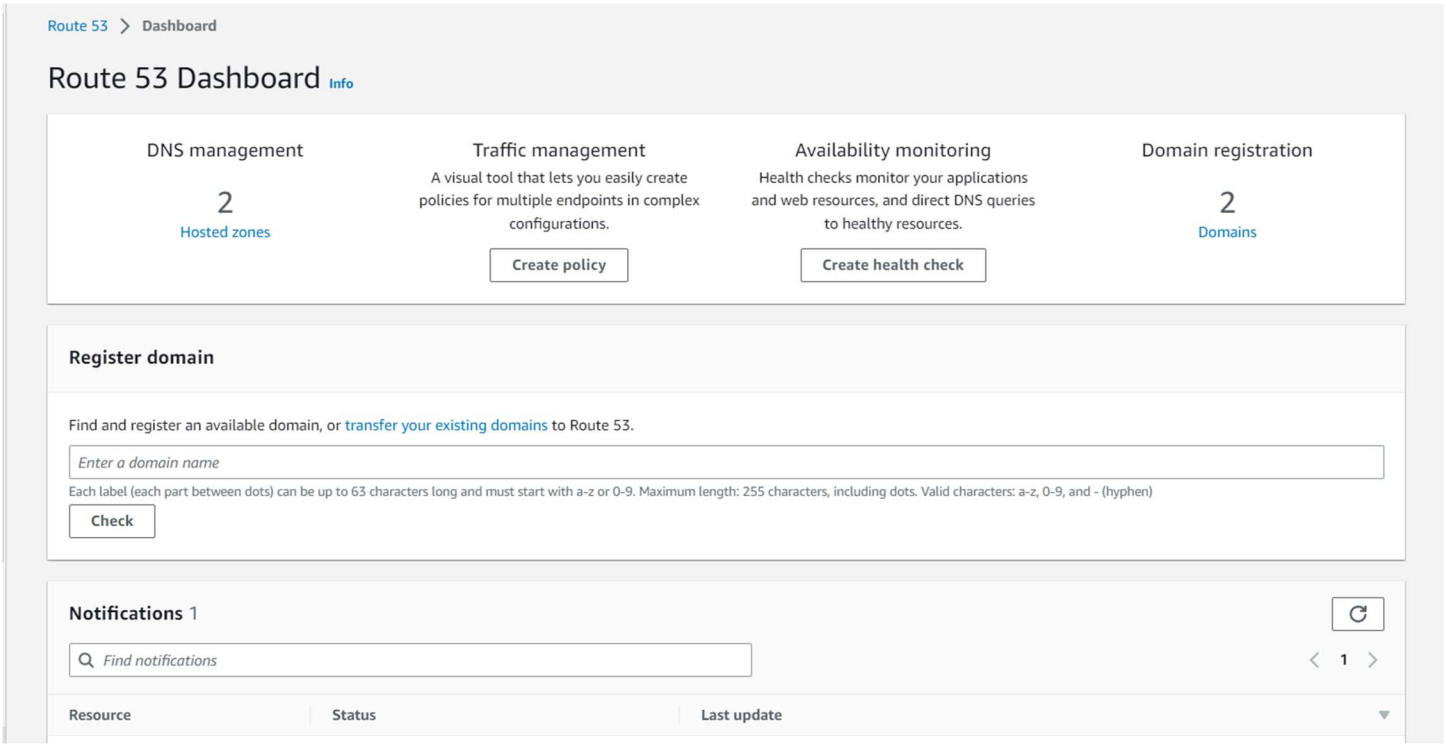
Under "**Route traffic to**" click on the drop down to select "**Endpoint**"

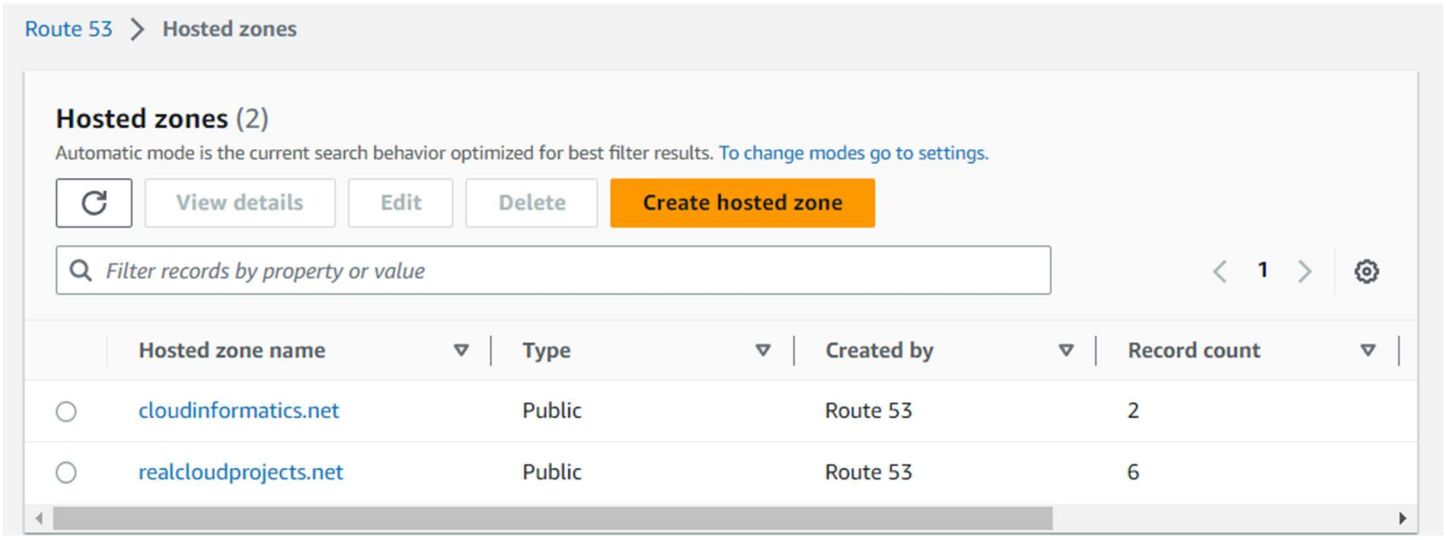
Select "**Alias** to **Application** and **Classic** Load Balancer"

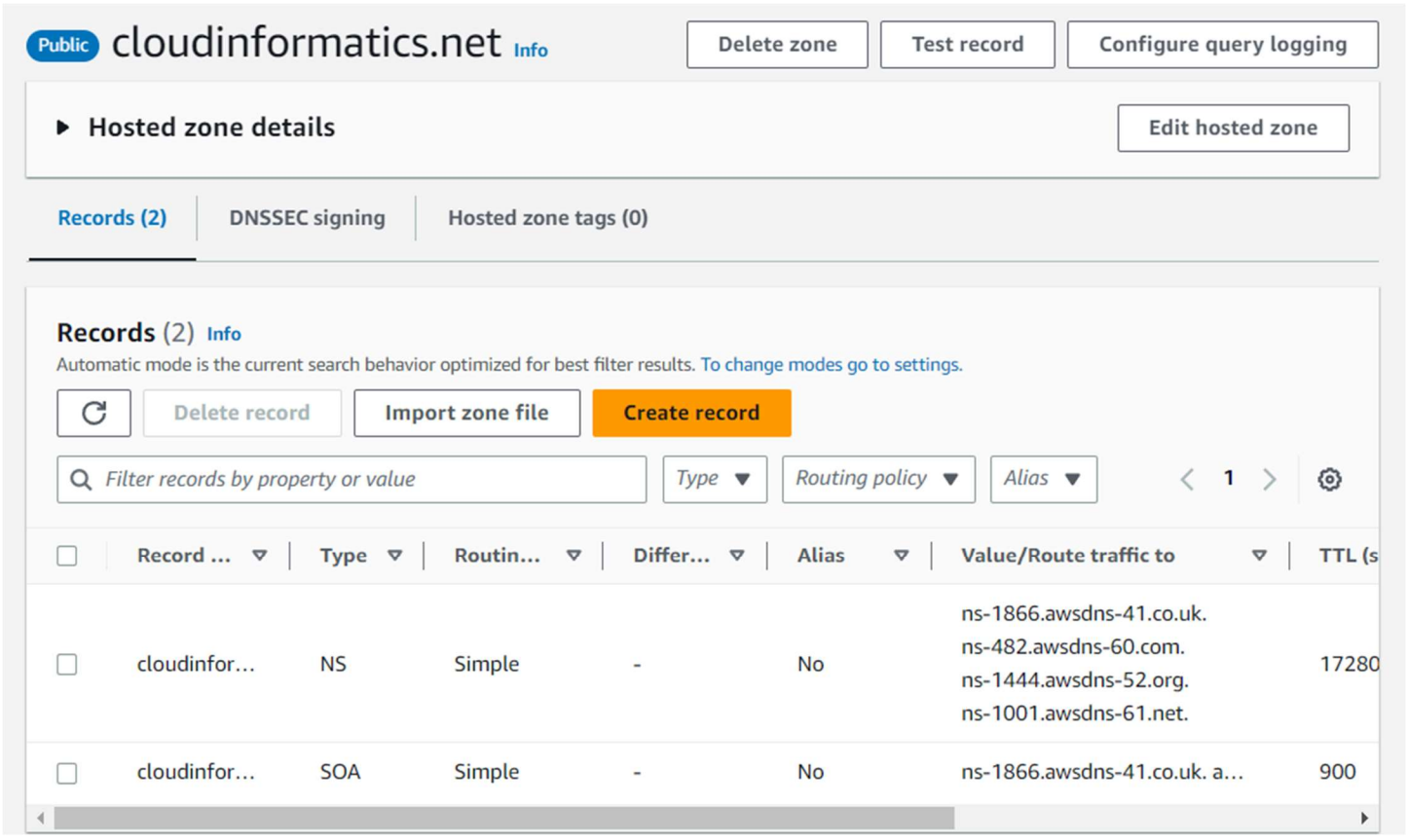
Select a region **US East (N. Virginia)**

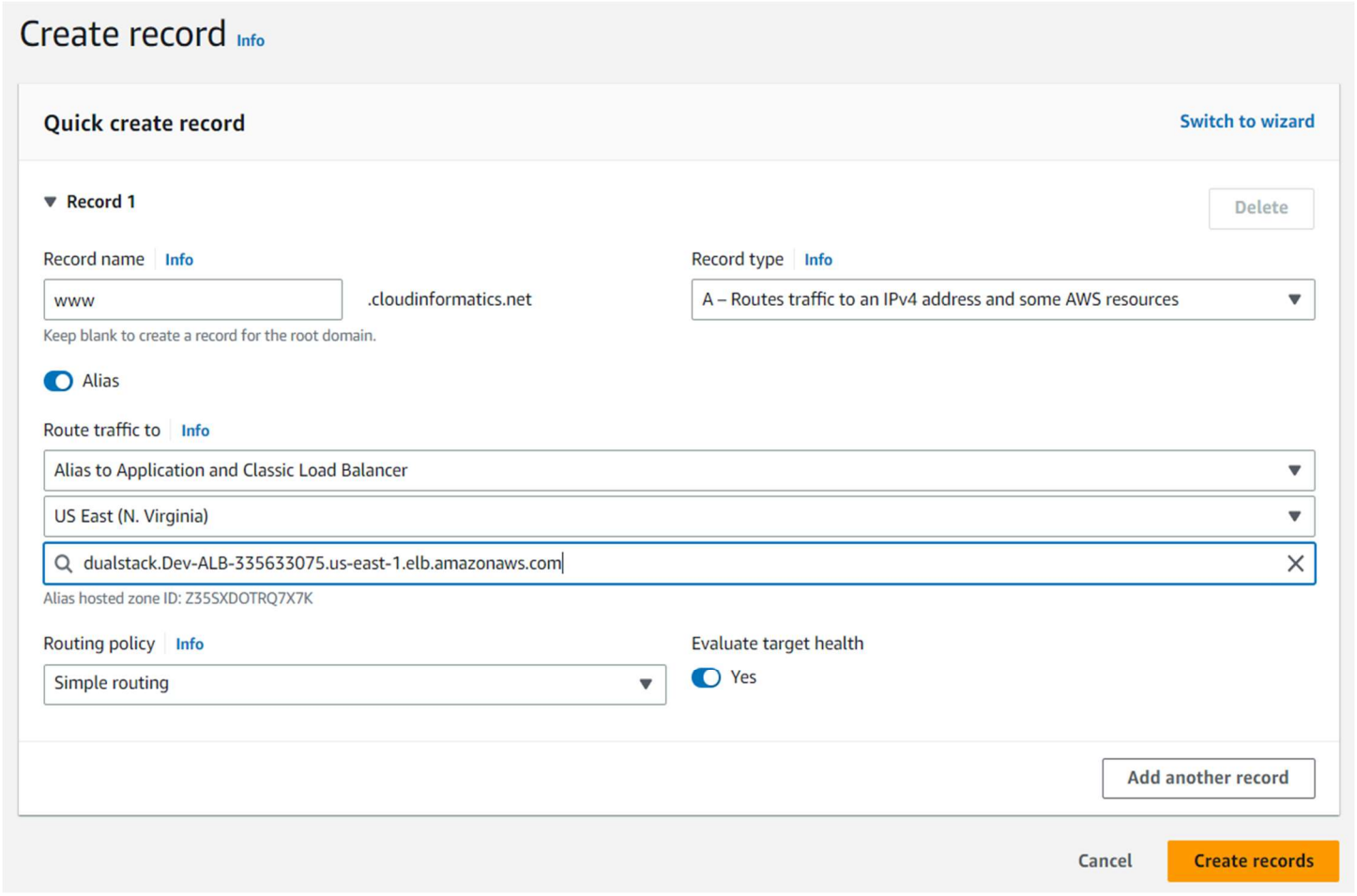
Search for the **Dev-ALB** under the us-east-1 **(N. Virginia)** region

Click on "**Create Records**"


Route 53 > Dashboard
Route 53 Dashboard Info
DNS management
2
Hosted zones
Traffic management
A visual tool that lets you easily create policies for multiple endpoints in complex configurations.
Create policy
Availability monitoring
Health checks monitor your applications and web resources, and direct DNS queries to healthy resources.
Create health check
Domain registration
2
Domains
Register domain
Find and register an available domain, or transfer your existing domains to Route 53.
Enter a domain name
Each label (each part between dots) can be up to 63 characters long and must start with a-z or 0-9. Maximum length: 255 characters, including dots. Valid characters: a-z, 0-9, and - (hyphen)
Check
Notifications 1
Find notifications
1
Resource
Status
Last update


Route 53 > Hosted zones
Hosted zones (2)
Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.
View details
Edit
Delete
Create hosted zone
Filter records by property or value
1
Hosted zone name
Type
Created by
Record count
cloudinformatics.net
Public
Route 53
2
realcloudprojects.net
Public
Route 53
6


Public cloudinformatics.net Info
Delete zone
Test record
Configure query logging
Hosted zone details
Edit hosted zone
Records (2)
DNSSEC signing
Hosted zone tags (0)
Records (2) Info
Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.
Delete record
Import zone file
Create record
Filter records by property or value
Type
Routing policy
Alias
1
Record ...
Type
Routin...
Differ...
Alias
Value/Route traffic to
TTL (s
cloudinfor...
NS
Simple
-
No
ns-1866.awsdns-41.co.uk.
ns-482.awsdns-60.com.
ns-1444.awsdns-52.org.
ns-1001.awsdns-61.net.
17280
cloudinfor...
SOA
Simple
-
No
ns-1866.awsdns-41.co.uk. a...
900

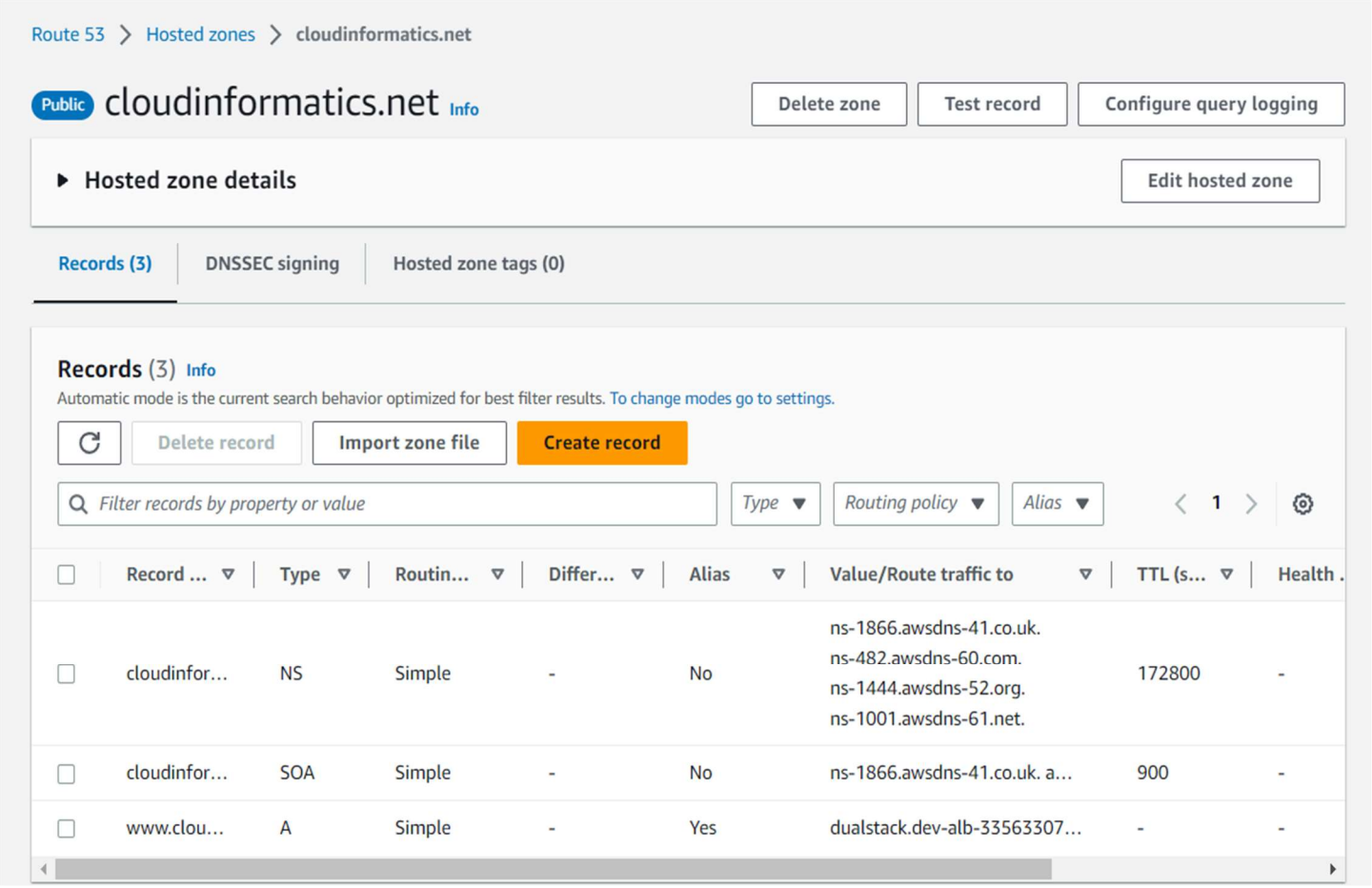

Create record Info
Quick create record
Switch to wizard
Record 1
Delete
Record name Info
www
.cloudinformatics.net
Keep blank to create a record for the root domain.
Record type Info
A – Routes traffic to an IPv4 address and some AWS resources
Alias
Route traffic to Info
Alias to Application and Classic Load Balancer
US East (N. Virginia)
dualstack.Dev-ALB-335633075.us-east-1.elb.amazonaws.com
Alias hosted zone ID: Z35SXDOTRQ7X7K
Routing policy Info
Simple routing
Evaluate target health
Yes
Add another record
Cancel
Create records

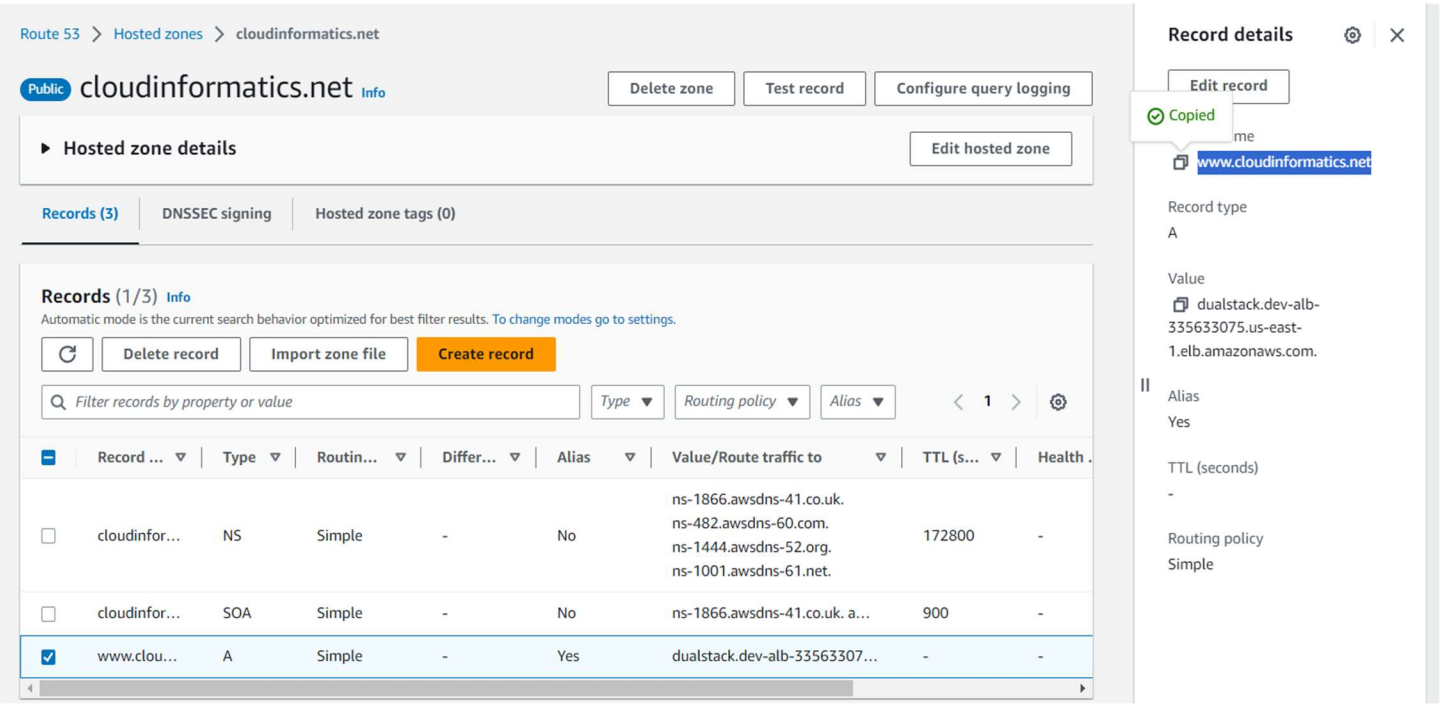
We have **successfully** created a **record set** in **Route 53**, and we have pointed our domain name "**cloudinformatics.net**" to our Application Load Balancer "**Dev-ALB**"

At this point, let's use our domain name to access our **jupiter** website

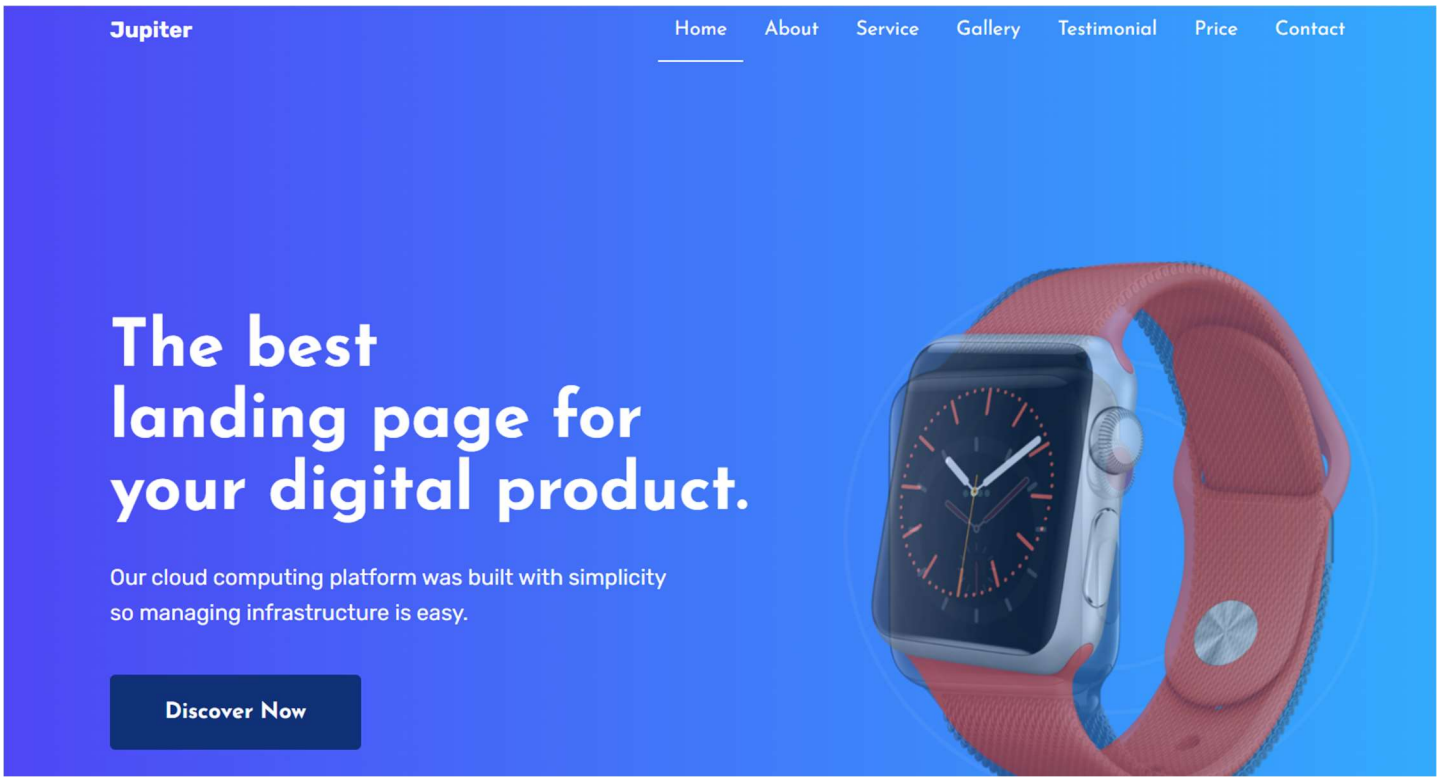
Select the domain name "cloudinformatics.net" navigate to the right side of your screen to copy the "**Record name**"

This is how we create a **record set** in **Route 53** and point the record set to our **Application Load Balancer** (ALB)






Open a new browser tab and paste it there, press "**Enter**"




We have **successfully** accessed the "**jupiter**" website using the domain name "**cloudinformatics.net"**

## STEP 34:

### REGISTRATION OF AN SSL CERTIFICATE IN AWS CERTIFICATE MANAGER (ACM)

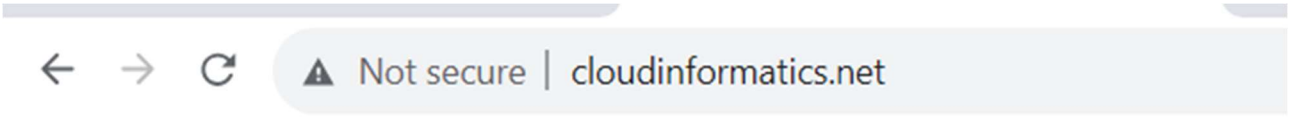
The main task here will be to request a free **SSL certificate** from **AWS Certificate manager (ACM)**

We will use the **SSL Certificate** to encrypt all communication and traffic between our web servers and web browser

**IMPORTANT** to note that whenever we visit a website, and the website is having a "Lock icon"

that signifies that the Website is secure, and traffic between the website and your web browser is secure.

Currently, the communication between our jupiter website and web browser IS NOT secure as we can see from the image below




Therefore, we will use our S**SL Certificate** to secure all **traffic / communication** between our web browser and website and this is also known as **encryption in transit.**

It is important to select the AWS N. Virginia region (**us-east-1**) when requesting a public certificate because CloudFront only recognizes this region as it's ACM certificates issuer.

Enter **Certificate Manager** in the search box to navigate to Certificate Manager

Click on "**Request a certificate**"

Select "**Request a public certificate**"

Under **Fully Qualified Domain Name** (FQDN), we need to enter the domain name that we are requesting the SSL certificate for.

"**cloudinformatics.net**" for this project.

Click on "**Add another name to this certificate**"

This allows you to add a wildcard to your domain name, for example the wildcard **(*.)** allows us to add

"www" to your domain name. For example, "www" + "cloudinformatics.net" = www.cloudinformatics.net

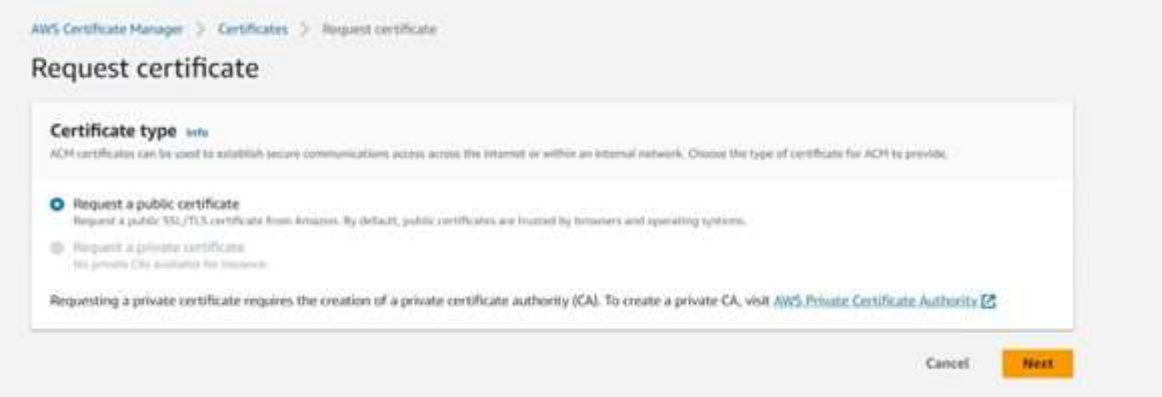
Select "**DNS Validation**" (Recommended)

Leave the "Key Algorithm" as default.

Click on "**Request**"

SSL Certificate successfully requested

Click on the "**certificate Id**" and the status is pending validation

# Request public certificate

## Domain names

Provide one or more domain names for your certificate.

Fully qualified domain name Info

cloudinformatics.net Remove

*.cloudinformatics.net Remove

Add another name to this certificate

You can add additional names to this certificate. For example, if you're requesting a certificate for "www.example.com", you might want to add the name "example.com" so that customers can reach your site by either name.

## Validation method Info

Select a method for validating domain ownership.

- (●) DNS validation - recommended
  Choose this option if you are authorized to modify the DNS configuration for the domains in your certificate request.
- ( ) Email validation
  Choose this option if you do not have permission or cannot obtain permission to modify the DNS configuration for the domains in your certificate request.

## Key algorithm Info

Select an encryption algorithm. Some algorithms may not be supported by all AWS services.

- (●) RSA 2048

---

AWS Certificate Manager > Certificates

**Certificates** (2)

Delete | Manage expiry events | Import | Request

1

| Certificate ID | Domain name | Type | Status | In use | Renewal eligibility | Key algorithm |
|---|---|---|---|---|---|---|
| 2e8f8a7c-0bc6-4aff-971e-316f5c8f8d5c | cloudinformatics.net | Amazon Issued | Pending validation | No | Ineligible | RSA 2048 |

AWS Certificate Manager > Certificates > 2e8f8a7c-0bc6-4aff-971e-316f5c8f8d5c

# 2e8f8a7c-0bc6-4aff-971e-316f5c8f8d5c

Delete

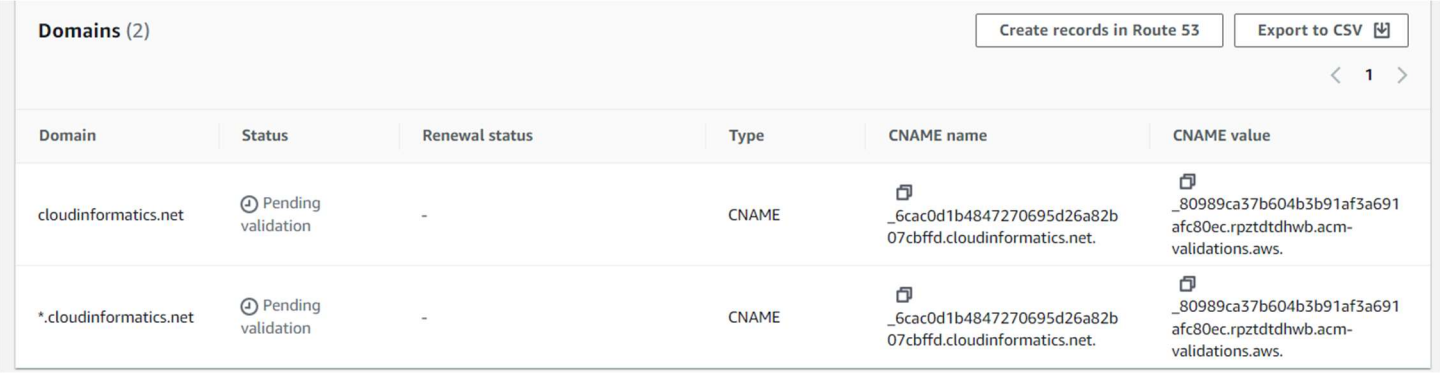
## Certificate status

Identifier
2e8f8a7c-0bc6-4aff-971e-316f5c8f8d5c

Status
Pending validation Info

ARN
arn:aws:acm:us-east-1:763176333159:certificate/2e8f8a7c-0bc6-4aff-971e-316f5c8f8d5c

Type
Amazon Issued




Now we need to create a **record set** in **Route 53** to validate that we are the rightful owner of the certificate

Fortunately, AWS has made this process easy and stress free.

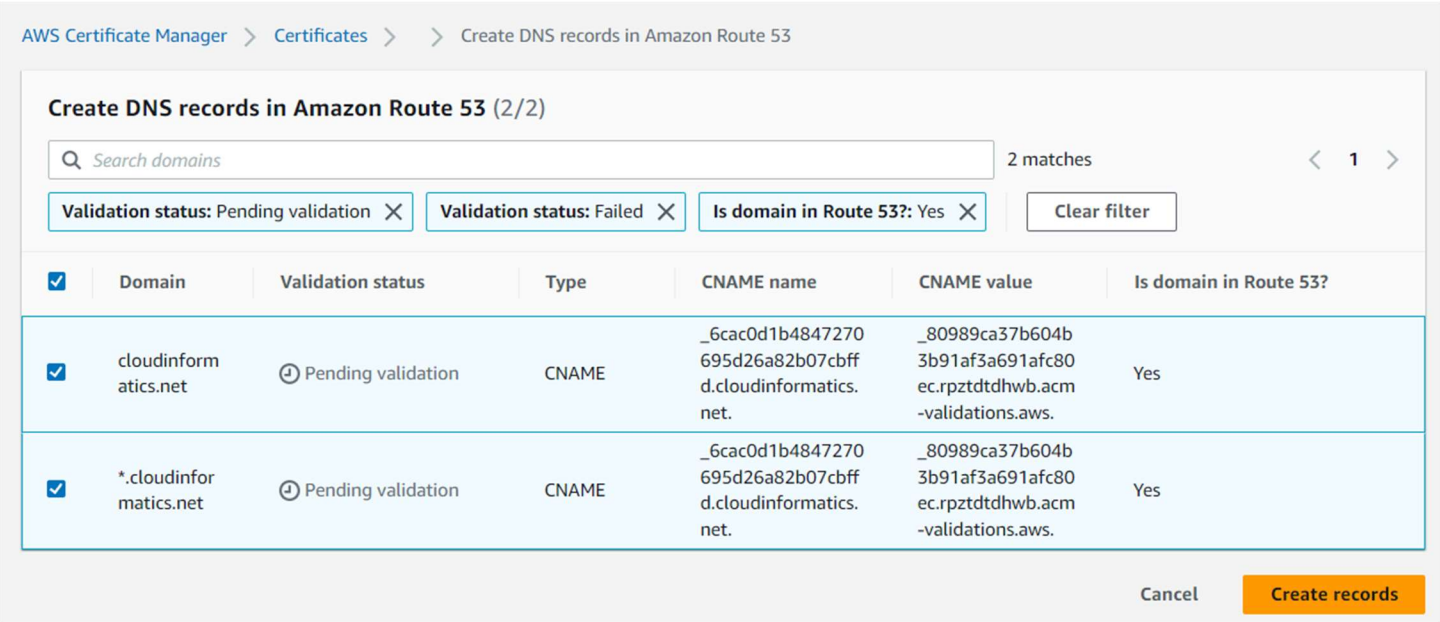
Choose a method of validation, however **DNS validation** is recommended.

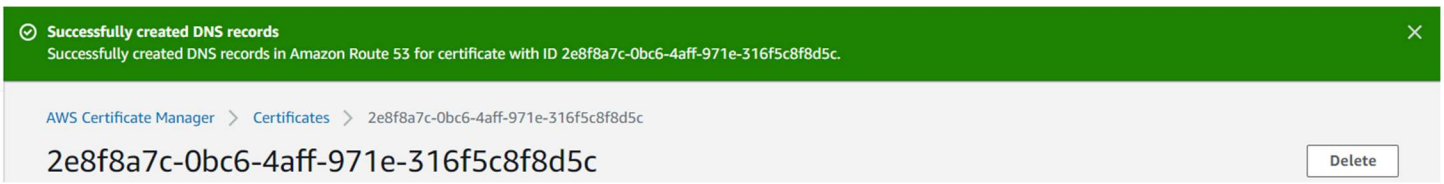
Click on **"Create records in Route 53"**

We must select the **domain name** we are creating the record set for in **Route 53.**

We must select the **wild card** as well

Click on **"Create Records"**

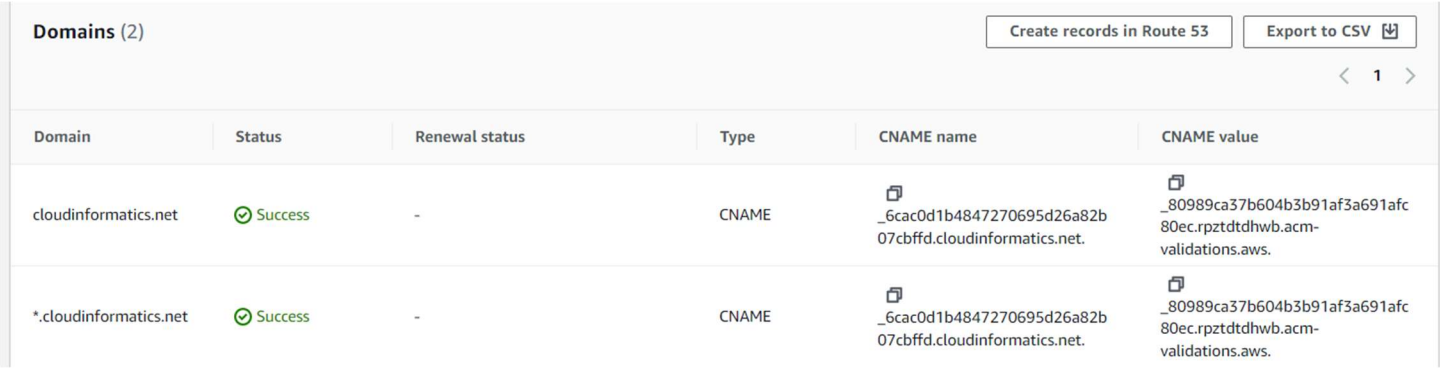


We have successfully created DNS records in Route 53 to validate that this domain name belongs to us.

We need to wait a few minutes for the **validation** process to be completed

Click on the refresh tab to refresh the page.


We can see that the status of the certificate has changed from **pending validation** to "**Issued**"

We can also see that for the 2 domain names that we requested the certificates for, the status has changed from pending validation to "**Success**"

## STEP 35:

## CREATE A HTTPS (SSL) LISTENER FOR AN APPLICATION LOAD BALANCER (ALB)

The main task here will be to use the **SSL Certificate** that we registered earlier to secure all traffic in transit to our website.

Let’s navigate to the **EC2 console**

Then we will scroll down to select "**Load Balancers**"

Click on the **Dev-ALB**

Scroll down to the "**Listeners and rules**" tab

Here, we will create our **HTTPS Listener** that will securely **encrypt** the communication in transit between our end-user and our jupiter website.

In other words, whenever a user visits a website with the "**lock icon”** this means that the communication between the end-user and website is **secure.**

Important to know that the **HTTPS Listener** that we are going to create will enable us to add the "**lock icon**" to our website signifying that the communication or traffic between the user and website is **encrypted**.

Click on "**Add Listener**"

Under Protocol: Port

Click on the dropdown

Select **HTTPS (Port 443)**

Under Action Types

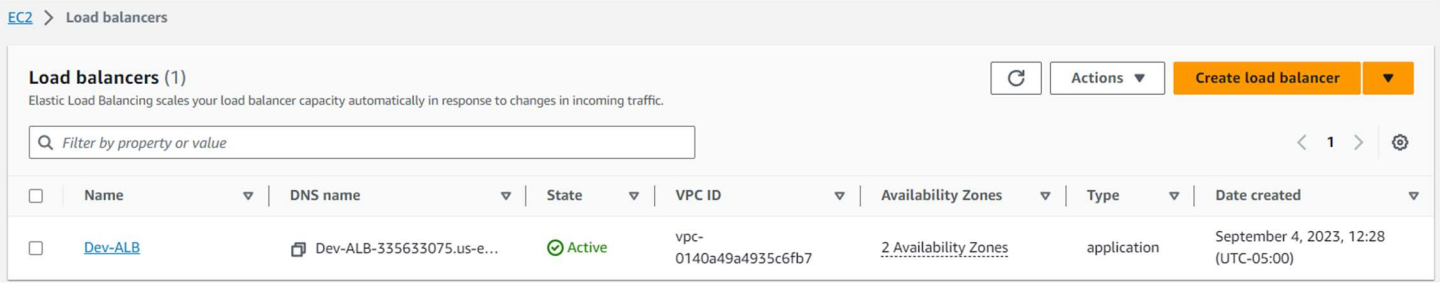
Select “**Forward to Target groups”**

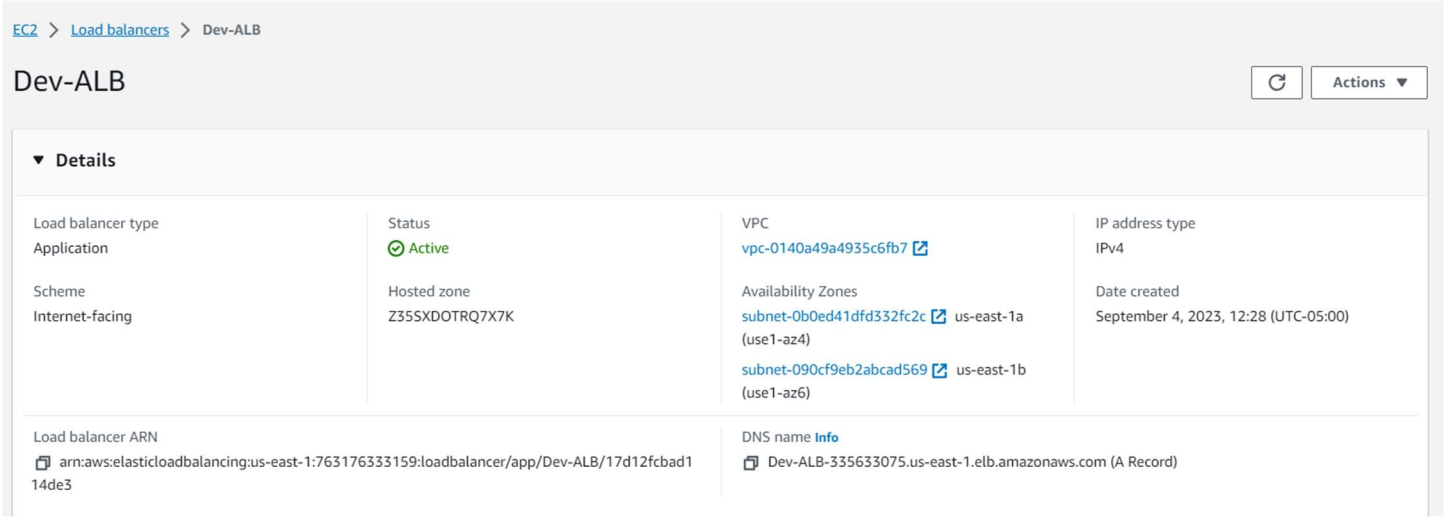
Click on dropdown and select **"Dev-Target-Group"**

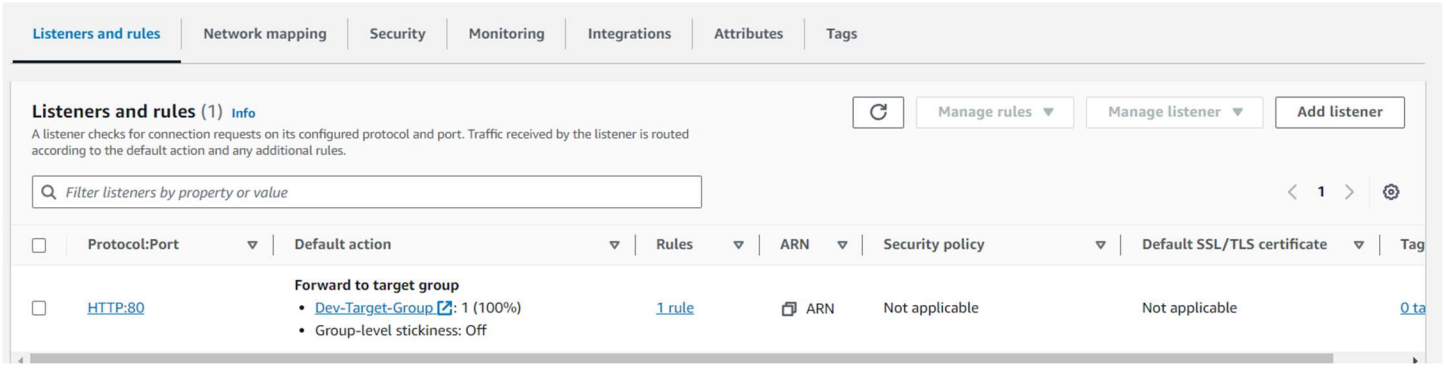
Scroll down to **Default SSL/TLS certificate**

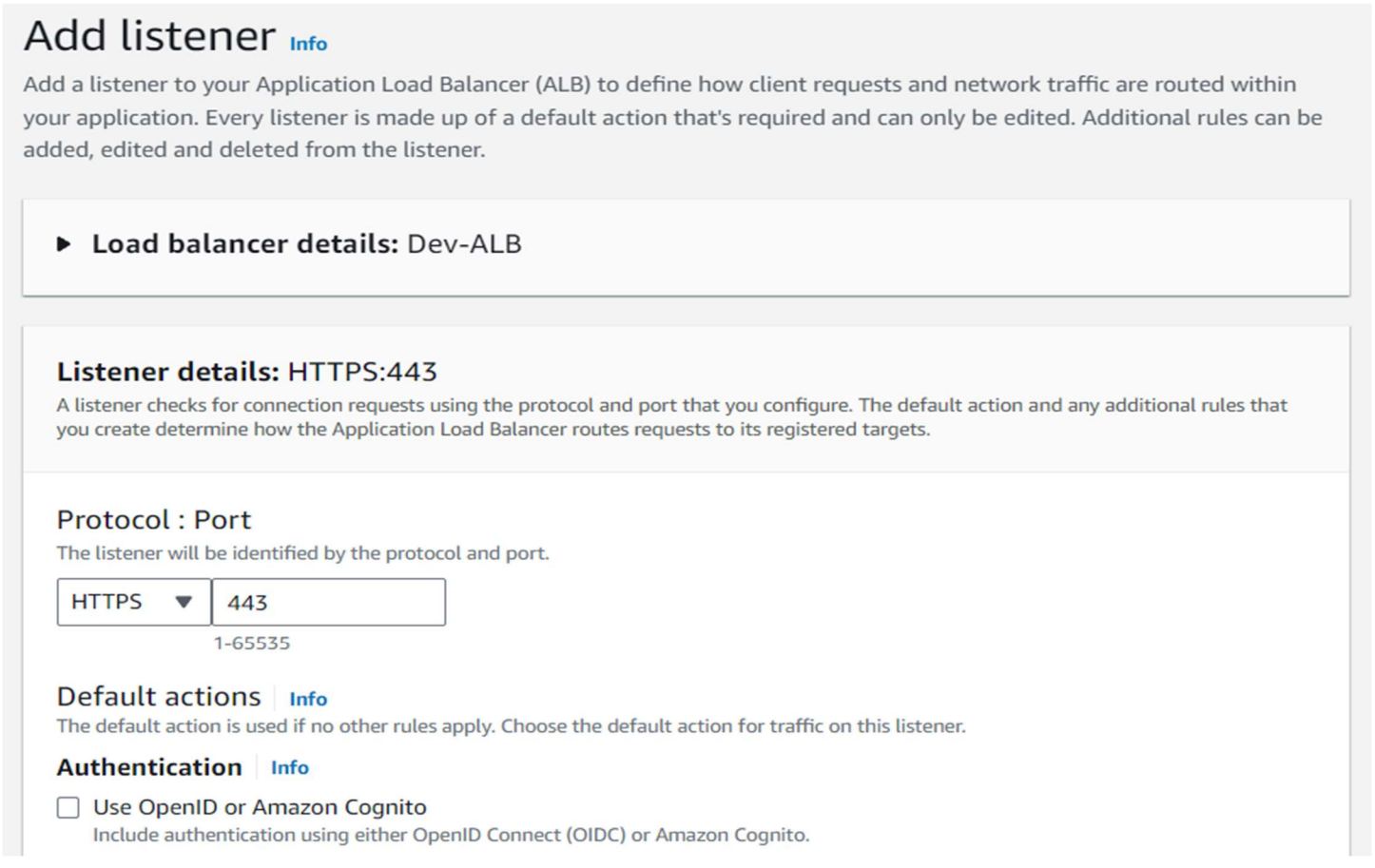
Select from **ACM** to **cloudinformatics.net**

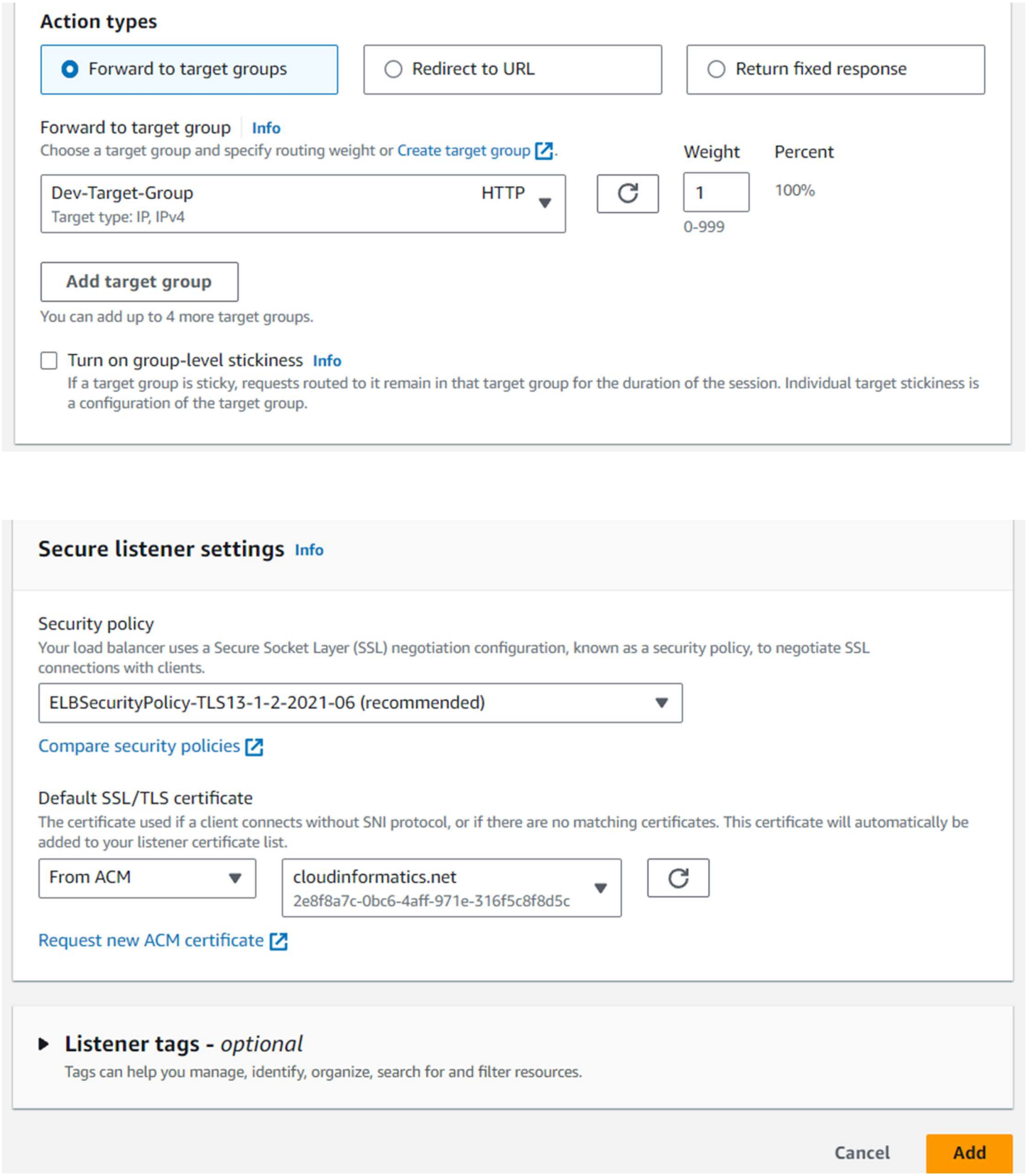
Click on **“Add”**




EC2 > Load balancers > Dev-ALB
Dev-ALB
Actions
Details
Load balancer type
Application
Status
Active
VPC
vpc-0140a49a4935c6fb7
IP address type
IPv4
Scheme
Internet-facing
Hosted zone
Z35SXDOTRQ7X7K
Availability Zones
subnet-0b0ed41dfd332fc2c us-east-1a (use1-az4)
subnet-090cf9eb2abcad569 us-east-1b (use1-az6)
Date created
September 4, 2023, 12:28 (UTC-05:00)
Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:763176333159:loadbalancer/app/Dev-ALB/17d12fcbad114de3
DNS name Info
Dev-ALB-335633075.us-east-1.elb.amazonaws.com (A Record)


Listeners and rules
Network mapping
Security
Monitoring
Integrations
Attributes
Tags
Listeners and rules (1) Info
A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.
Manage rules
Manage listener
Add listener
Filter listeners by property or value
Protocol:Port
Default action
Rules
ARN
Security policy
Default SSL/TLS certificate
Tag
HTTP:80
Forward to target group
Dev-Target-Group: 1 (100%)
Group-level stickiness: Off
1 rule
ARN
Not applicable
Not applicable
0 ta


Add listener Info
Add a listener to your Application Load Balancer (ALB) to define how client requests and network traffic are routed within your application. Every listener is made up of a default action that's required and can only be edited. Additional rules can be added, edited and deleted from the listener.
Load balancer details: Dev-ALB
Listener details: HTTPS:443
A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.
Protocol : Port
The listener will be identified by the protocol and port.
HTTPS
443
1-65535
Default actions Info
The default action is used if no other rules apply. Choose the default action for traffic on this listener.
Authentication Info
Use OpenID or Amazon Cognito
Include authentication using either OpenID Connect (OIDC) or Amazon Cognito.







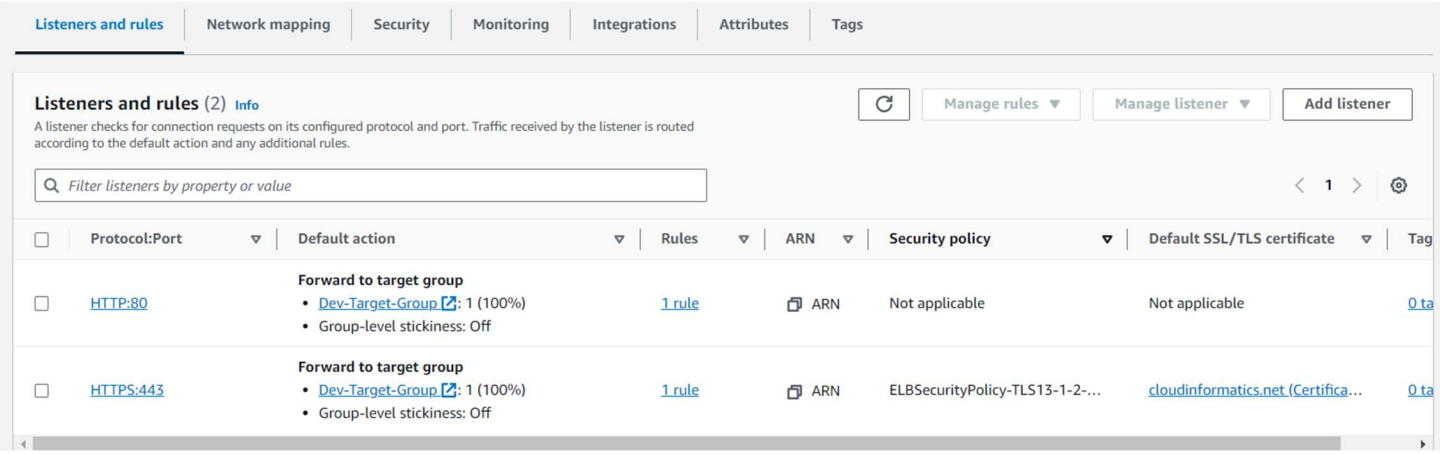
**Listener was successfully created**

Scroll down to **“Listeners and rules”**

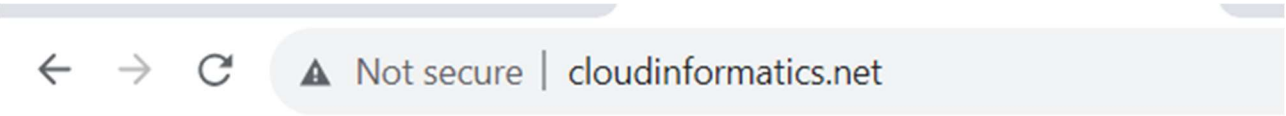
We now have 2 Listeners on **HTTP:80** and **HTTPS:443**

The listener on **HTTPS:443** is forwarding traffic to our target group **"Dev-Target-Group"**

The listener on **HTTP:80** now, is also forwarding traffic to our target group **"Dev-Target-Group**




But the main issue here is that the traffic from **HTTP:80** is **not secure** and is **not encrypted**.




To resolve this issue, we will modify **the listener on HTTP:80,** so that anytime an end-user is trying access our website on **HTTP:80**, the traffic will be redirected to **HTTPS:443**

Select **HTTP:80**

Click on "**Manage listener"**

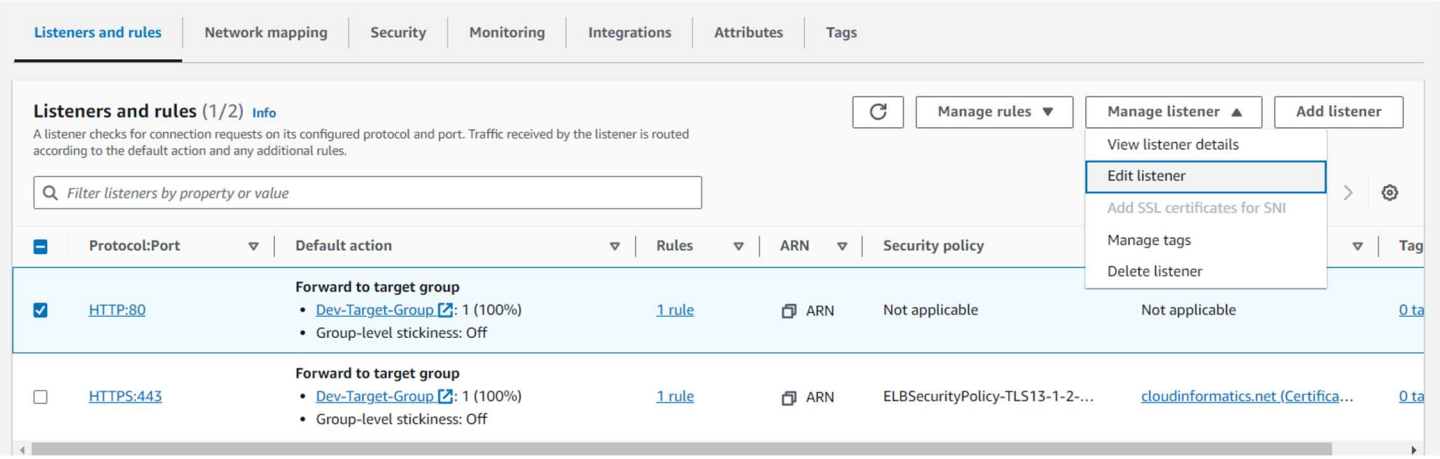
Select **"Edit Listener"**

Under default actions,

Select **"Redirect to URL”**

Select **"Full URL"**

Scroll down and click on **"Save changes"**




# Edit listener

Edit the protocol, port or default actions of your Application Load Balancer (ALB) listener.

▶ **Load balancer details:** Dev-ALB

## Listener details

A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.

Listener ARN

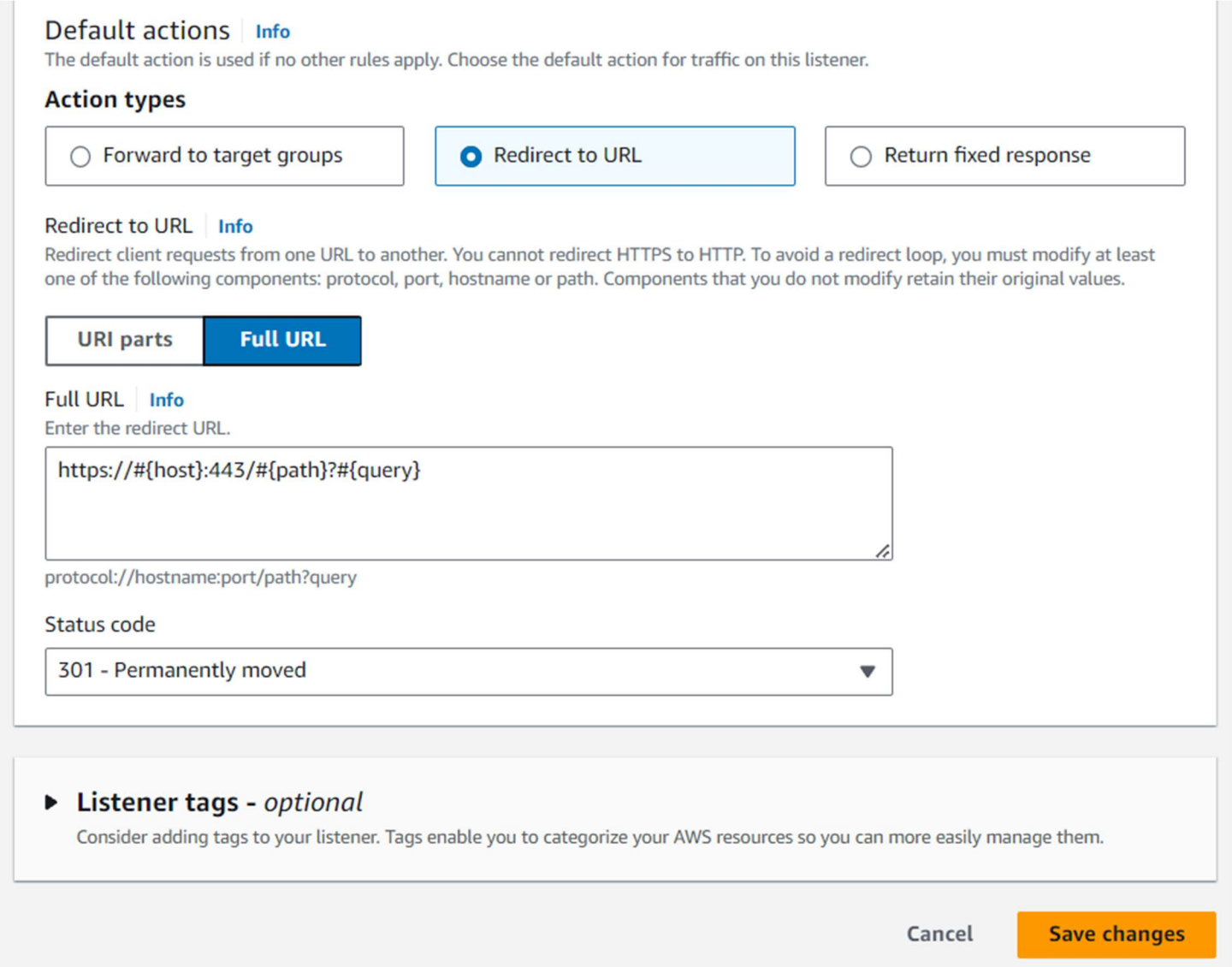
arn:aws:elasticloadbalancing:us-east-1:763176333159:listener/app/Dev-ALB/17d12fcbad114de3/0eaf43cc49fbec21

### Protocol : Port

The listener will be identified by the protocol and port.

HTTP | 80

1-65535





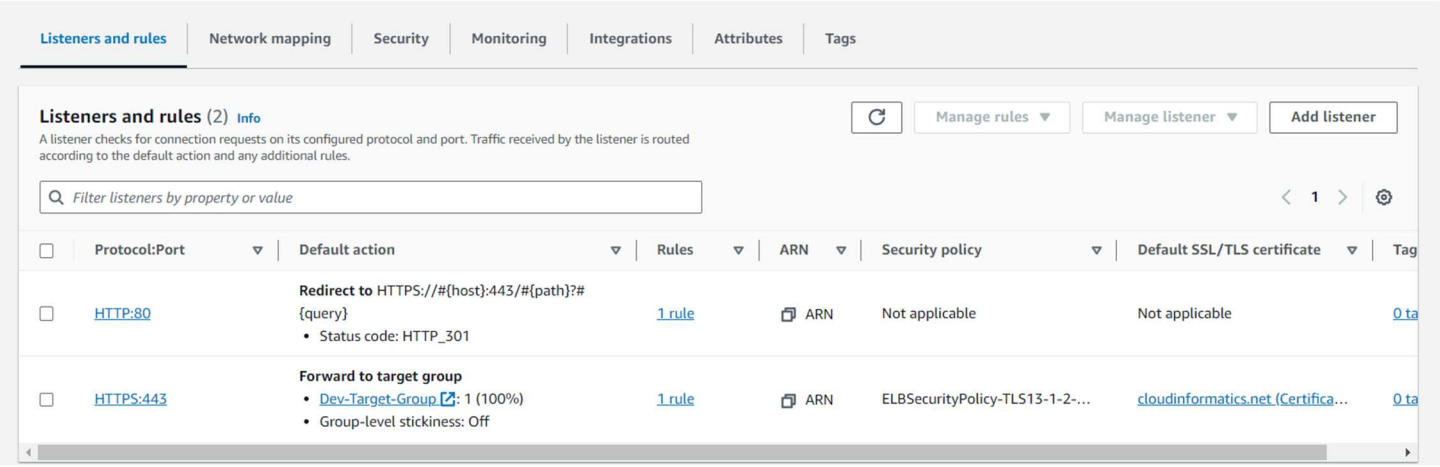
**HTTP:80 was successfully modified**

Click on the **“Dev-ALB”**

Scroll down to **"Listeners and rules"**

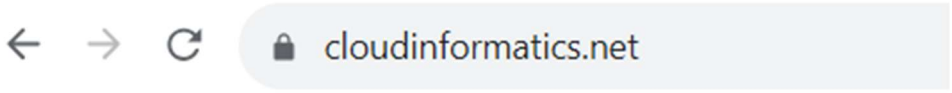
We can see that **HTTP:80** is now **redirecting** traffic to **HTTPS:443**

While **HTTPS:443** in turn is **forwarding** the traffic to our **"Dev-Target-Group"**




Now to verify the configuration works

open a new browser window and enter **"cloudinformatics.net"**




Now we can see the “**lock icon”** which means there is **secure** and **encrypted** communication between our end-users and our website using the **SSL Certificate.**

This is all we need to do to **encrypt** the communication between our **end-users** and our **website.**

**WOWWW!!! FINALLY DONE!!!!!**