

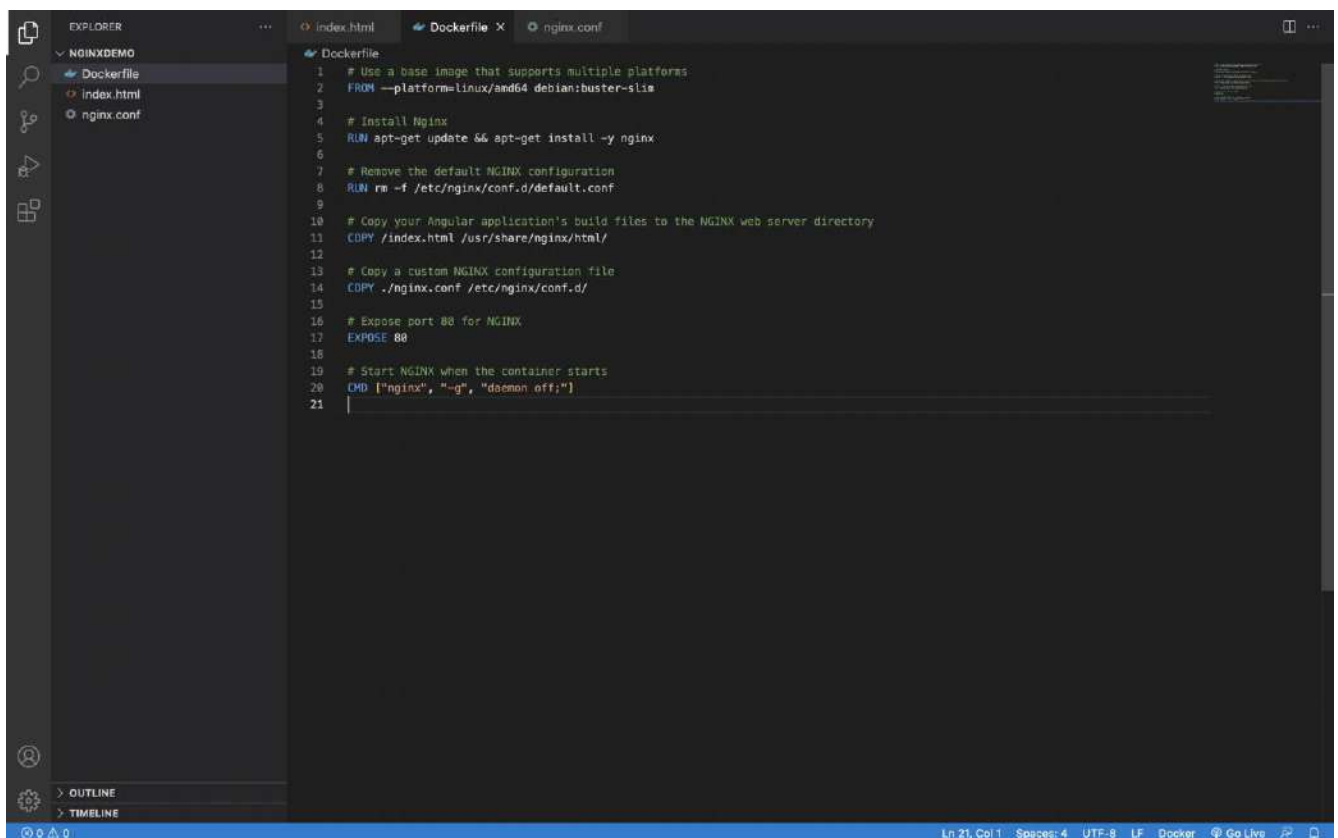
Pushing Docker Image to AWS ECR

Pre-requisites:

1. Docker Desktop
2. AWS CLI
3. Visual Studio Code

Steps:

1. Create folder and open in visual studio code.
2. Create Docker file and build Docker Image in local machine.



The screenshot shows the Visual Studio Code interface with a project named 'NGINXDEMO'. The Explorer sidebar on the left shows the file structure: 'Dockerfile', 'index.html', and 'nginx.conf'. The main editor area has two tabs open: 'Dockerfile' and 'nginx.conf'. The 'Dockerfile' tab is active, displaying the following content:

```
1 # Use a base image that supports multiple platforms
2 FROM --platform=linux/amd64 debian:buster-slim
3
4 # Install Nginx
5 RUN apt-get update && apt-get install -y nginx
6
7 # Remove the default NGINX configuration
8 RUN rm -f /etc/nginx/conf.d/default.conf
9
10 # Copy your Angular application's build files to the NGINX web server directory
11 COPY /index.html /usr/share/nginx/html/
12
13 # Copy a custom NGINX configuration file
14 COPY ./nginx.conf /etc/nginx/conf.d/
15
16 # Expose port 80 for NGINX
17 EXPOSE 80
18
19 # Start NGINX when the container starts
20 CMD ["nginx", "-g", "daemon off;"]
21
```

The 'nginx.conf' tab is also open but its content is not visible. The status bar at the bottom indicates 'Ln 21, Col 1', 'Spaces: 4', 'UTF-8', 'LF', 'Docker', and 'Go Live'.

Open terminal and run following commands

For Building Docker Image command is : `docker build -t imagename .`

For running Container command is : `docker -p hostport:containerport imagename`

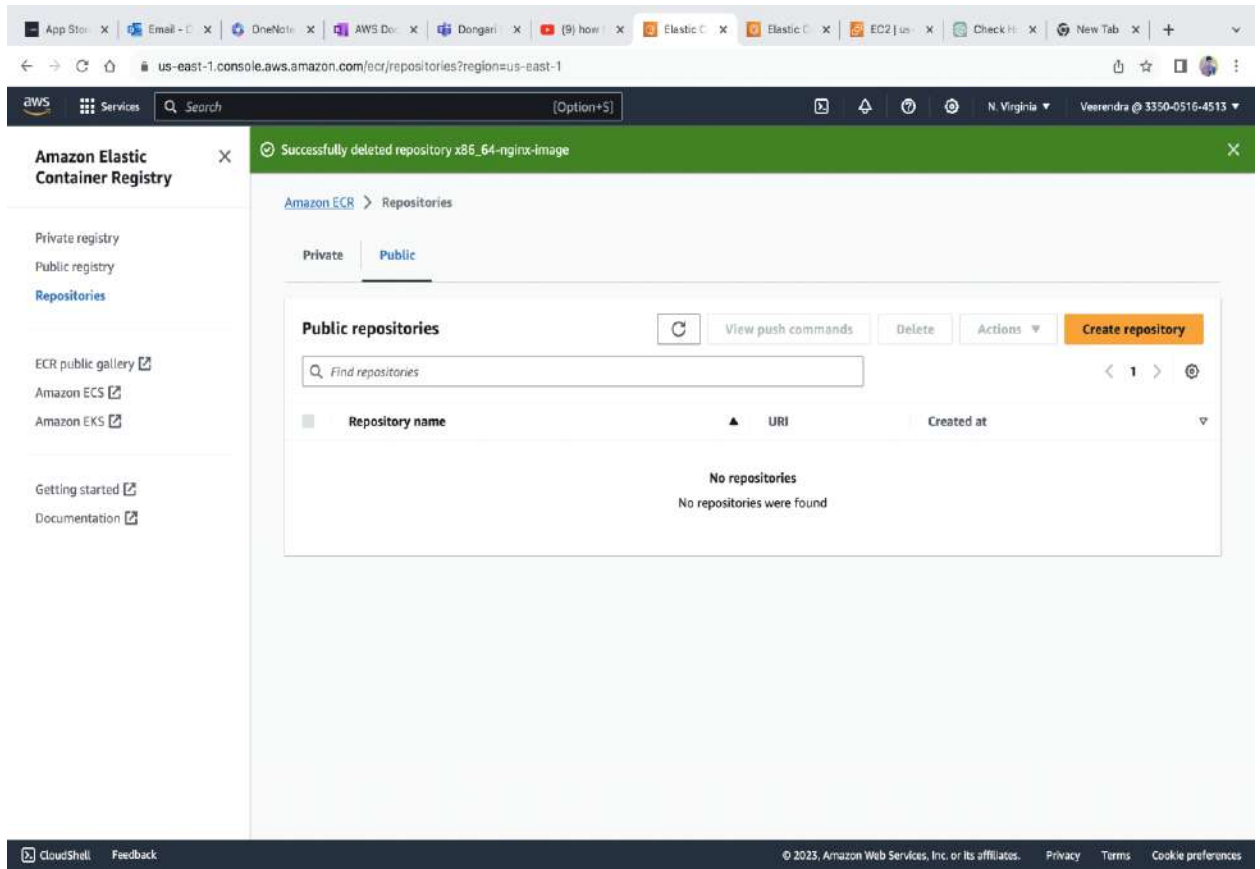
```
/dev/td/13118: command not found: compdef
SA28363898c02HD4JKQ6L5 NginxDemo % docker build --platform linux/amd64 -t x86_64-nginx-image .

[+] Building 0.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 600B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/debian:buster-slim
=> [1/5] FROM docker.io/library/debian:buster-slim@sha256:d9f318899c8ba5fdb0255ed1e2729c29de90e34230f48a392718a014a198b69
=> [internal] load build context
=> => transferring context: 61B
=> CACHED [2/5] RUN apt-get update && apt-get install -y nginx
=> CACHED [3/5] RUN rm -f /etc/nginx/conf.d/default.conf
=> CACHED [4/5] COPY ./index.html /usr/share/nginx/html/
=> CACHED [5/5] COPY ./nginx.conf /etc/nginx/conf.d/
=> exporting to image
=> exporting layers
=> writing image sha256:fff033f8c4f994bd0113610b7c17c4366615c3c855515de156d5bcfd9fa08212
=> naming to docker.io/library/x86_64-nginx-image

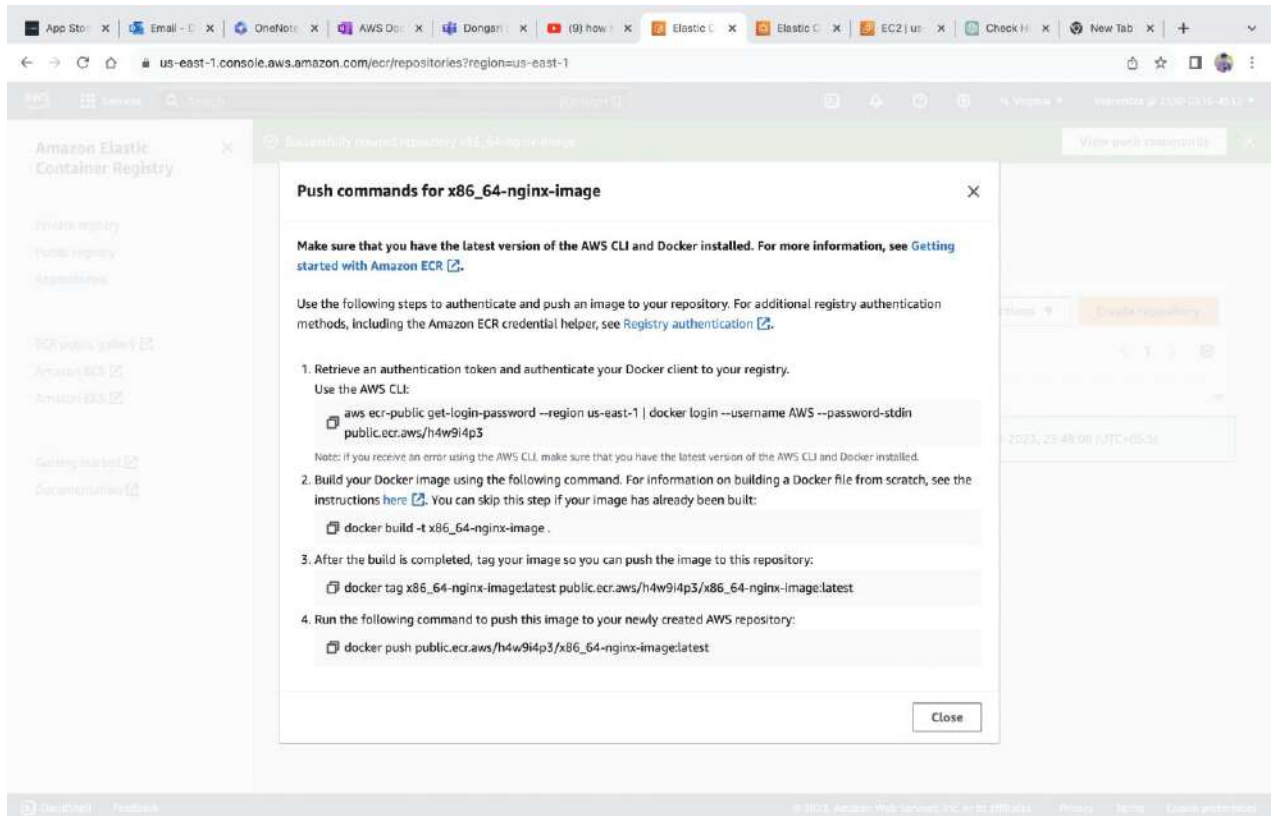
What's Next?
View a summary of image vulnerabilities and recommendations -> docker scout quickfix
SA28363898c02HD4JKQ6L5 NginxDemo % docker run -d -p 80:80 --platform linux/amd64 x86_64-nginx-image
ced748e722c378157268035f6615709797b0fc2eb099a69bc1c4b7b70f3c99a4
SA28363898c02HD4JKQ6L5 NginxDemo %
```

Now login to Amazon Web Service Console

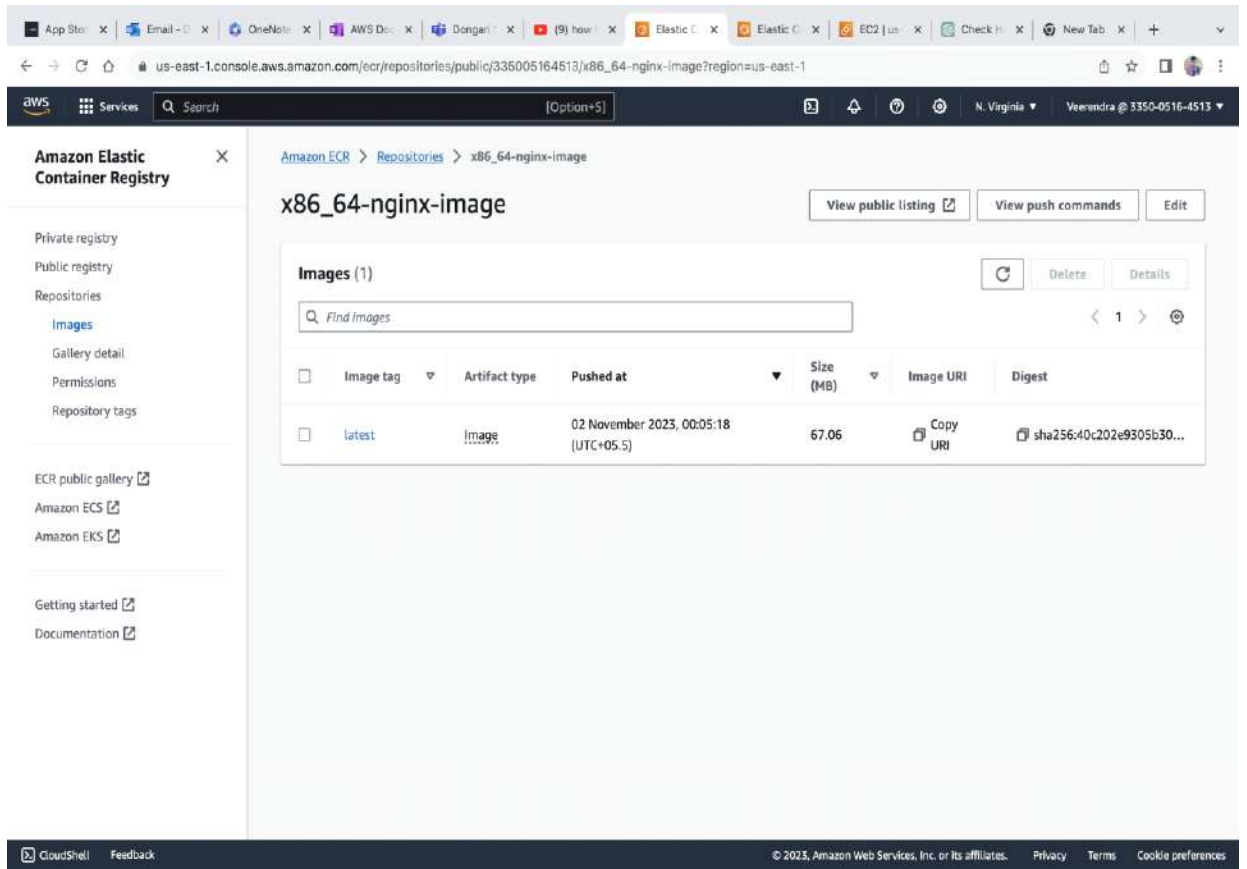
1. Open AWS Elastic Container Registry (ECR)
2. In menu open Repositories and create Repository



3. After creating Repository select that Repository and view push commands

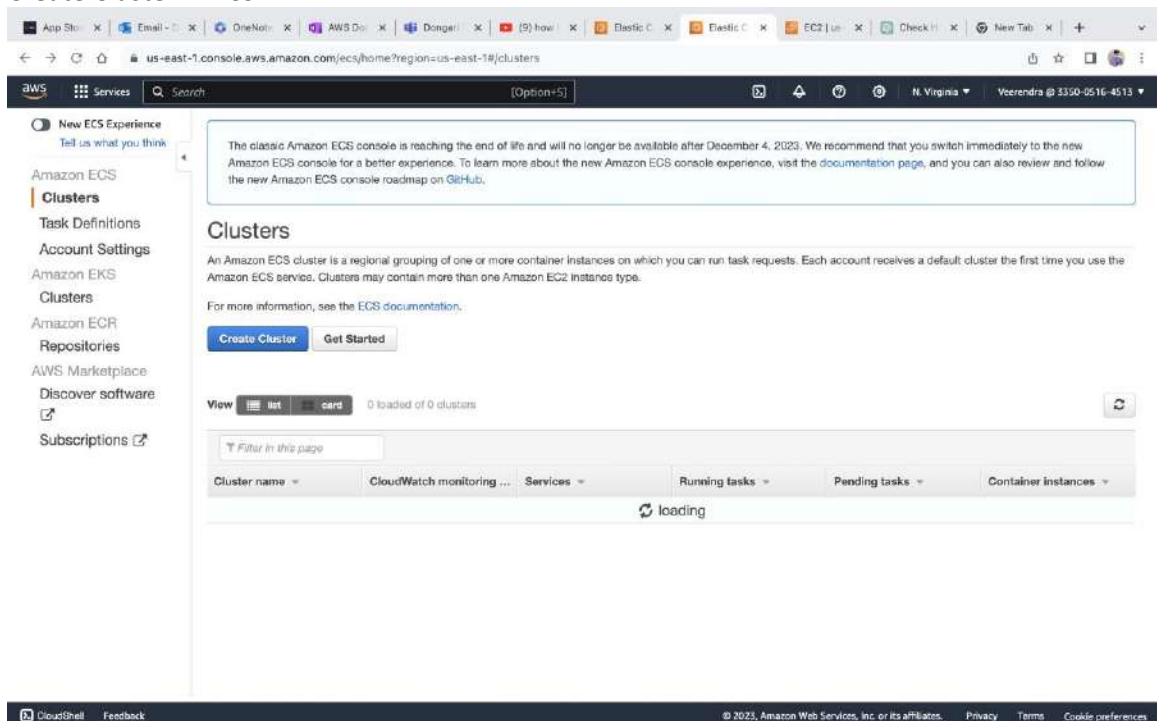


4. Run the above commands in visual studio code.
5. After running the above commands, the docker image is successfully pushed to AWS ECR.

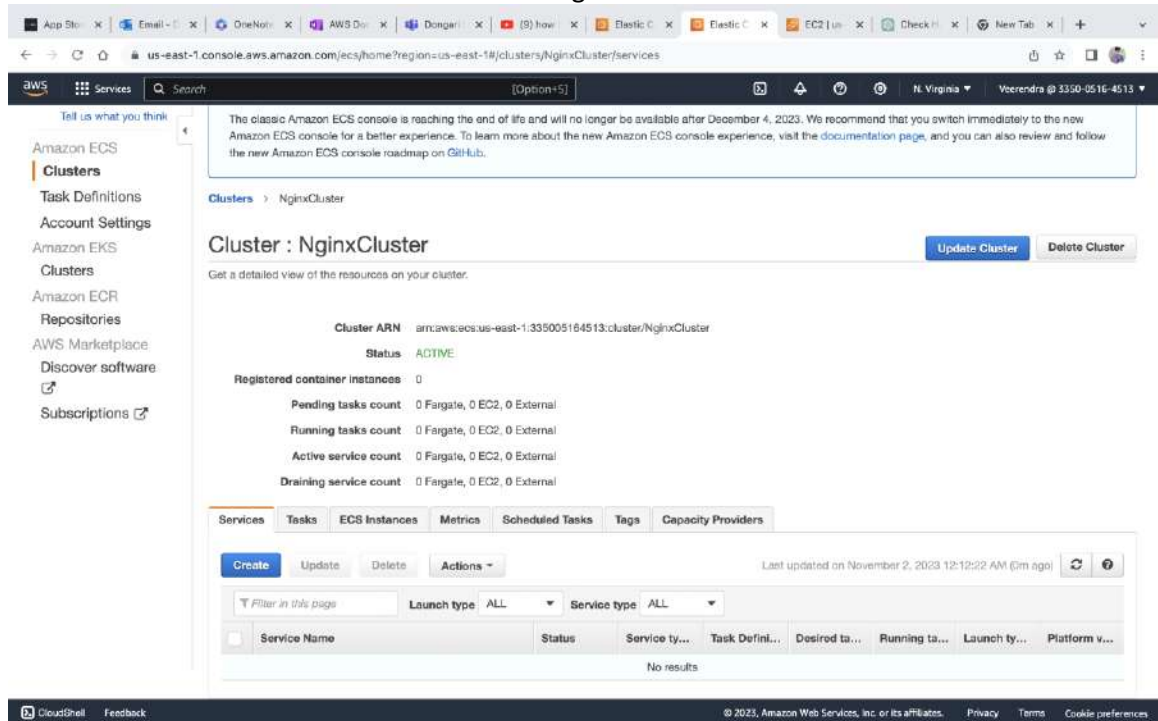


6. Now open AWS Elastic Container Service (ECS)

7. Create Cluster in ECS

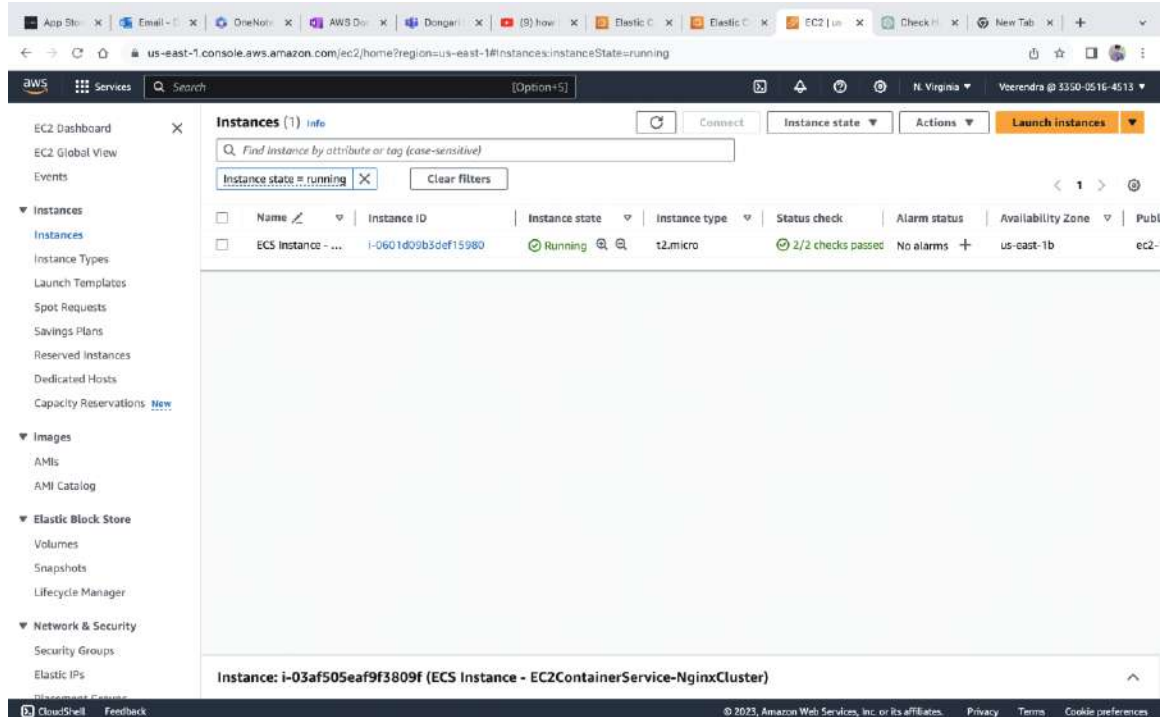


8. Click the create cluster button and make configurations based on the need.

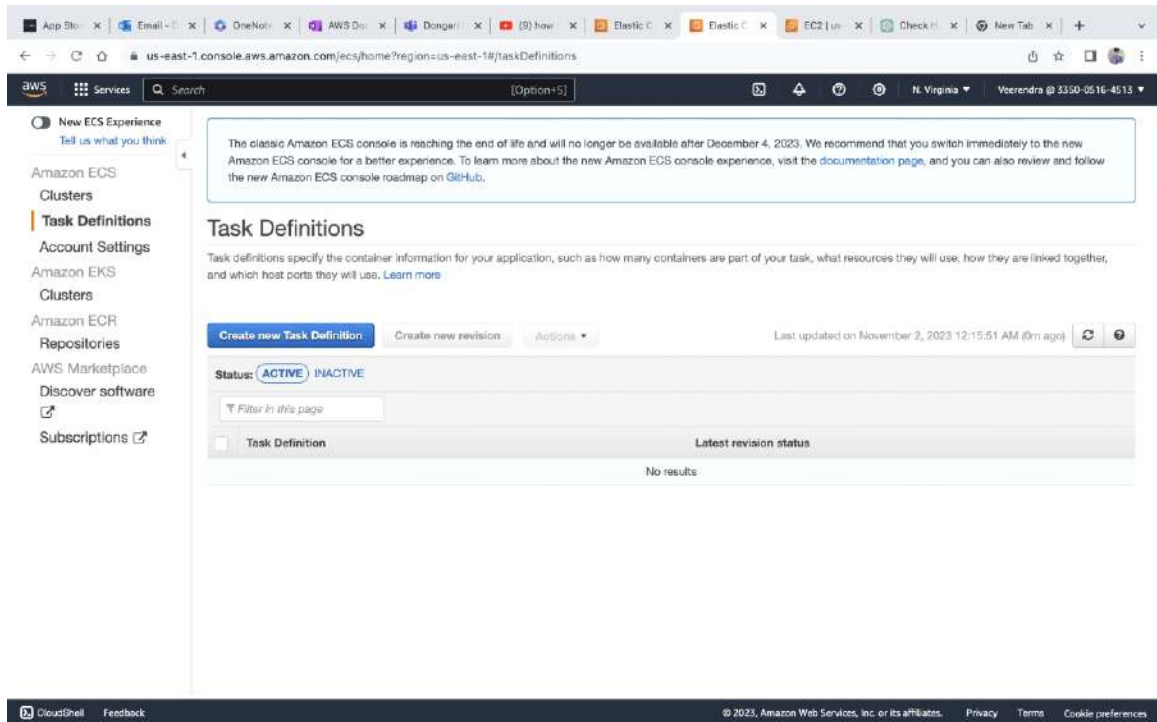


9. After creating the cluster, it will look like in the above screenshot.

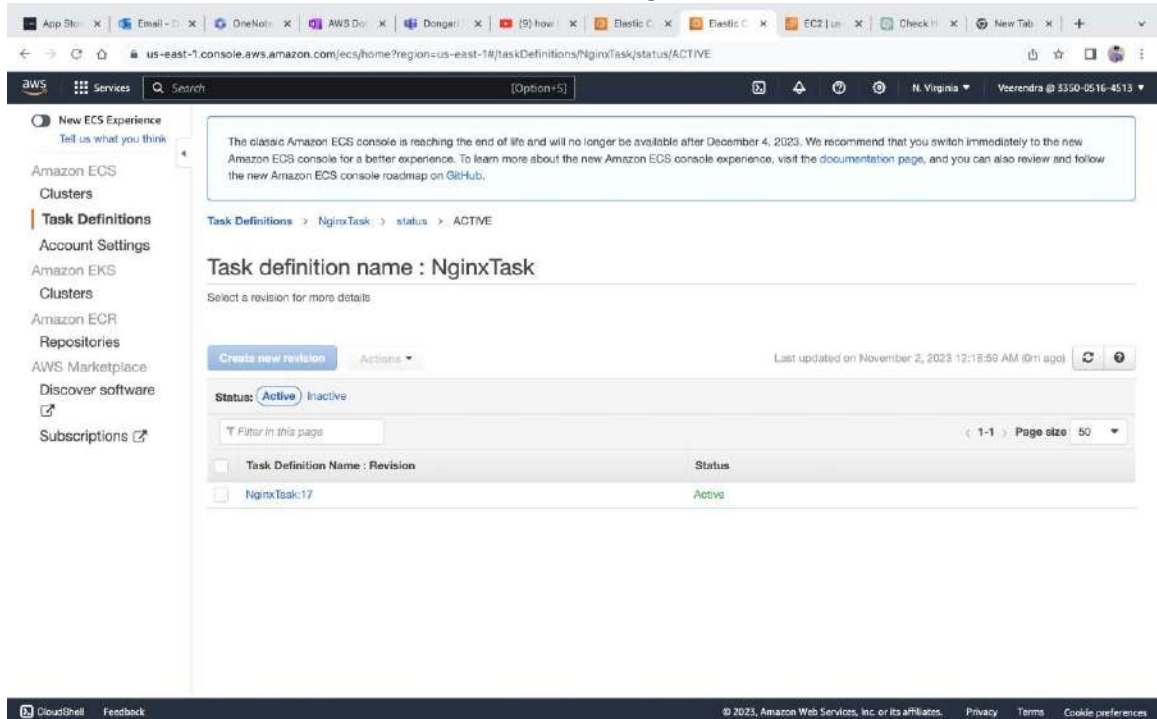
10. After creating the cluster, it also creates one EC2 instance.



11. Now create task definition to run tasks.

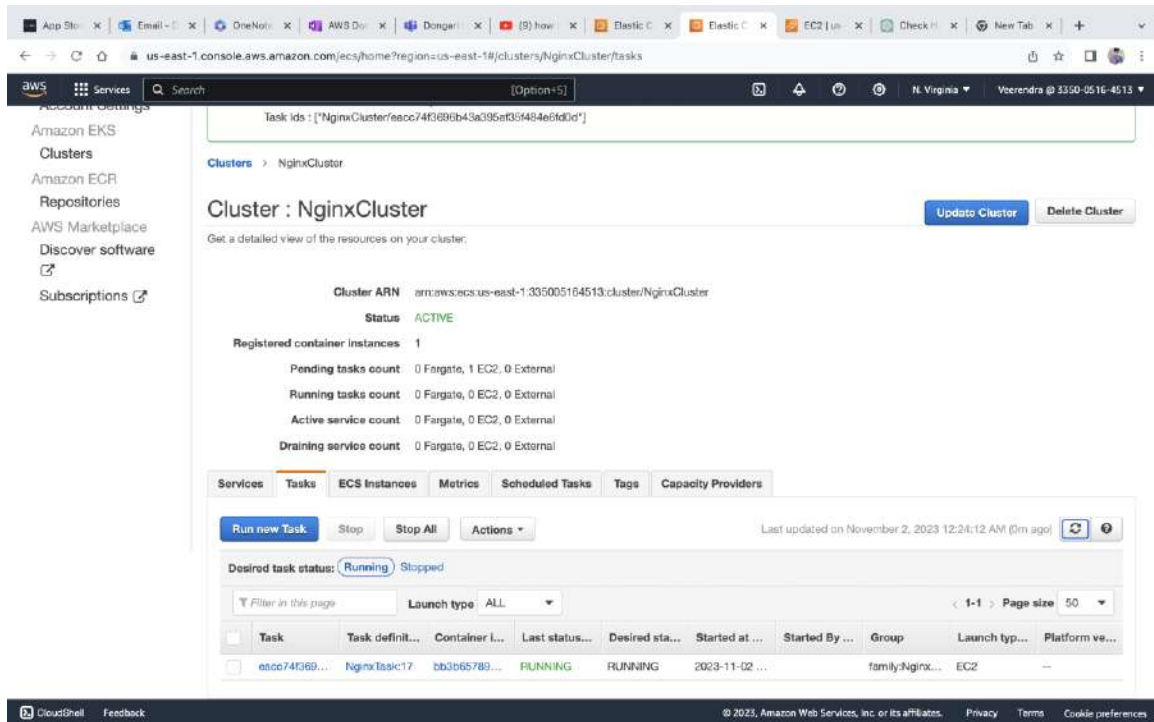


12. Click the create task definition button and make configurations based on the need.



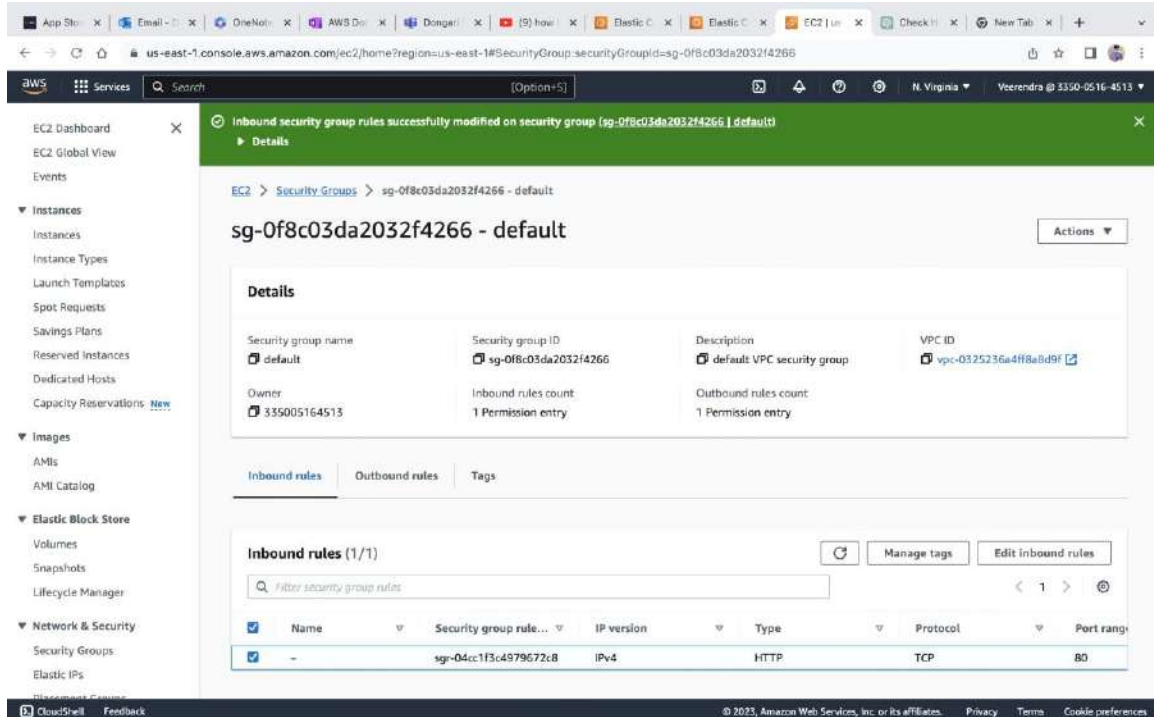
13. After creating the Task Definition, it will look it in the above screenshot.

14. Now go to cluster that we created and go to tasks and Run new Task.

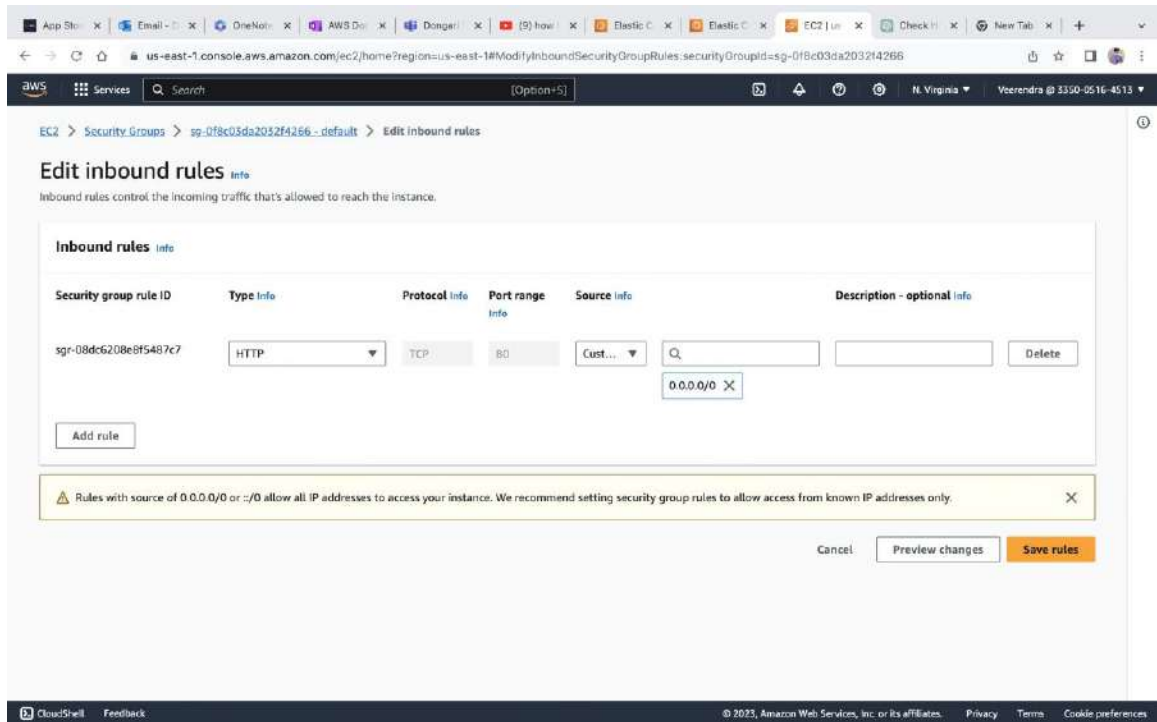


15. After running a task it will create one task like in the above screenshot.

16. Now to go to EC2 instance and select that EC2 instance go to security tab and select security groups.

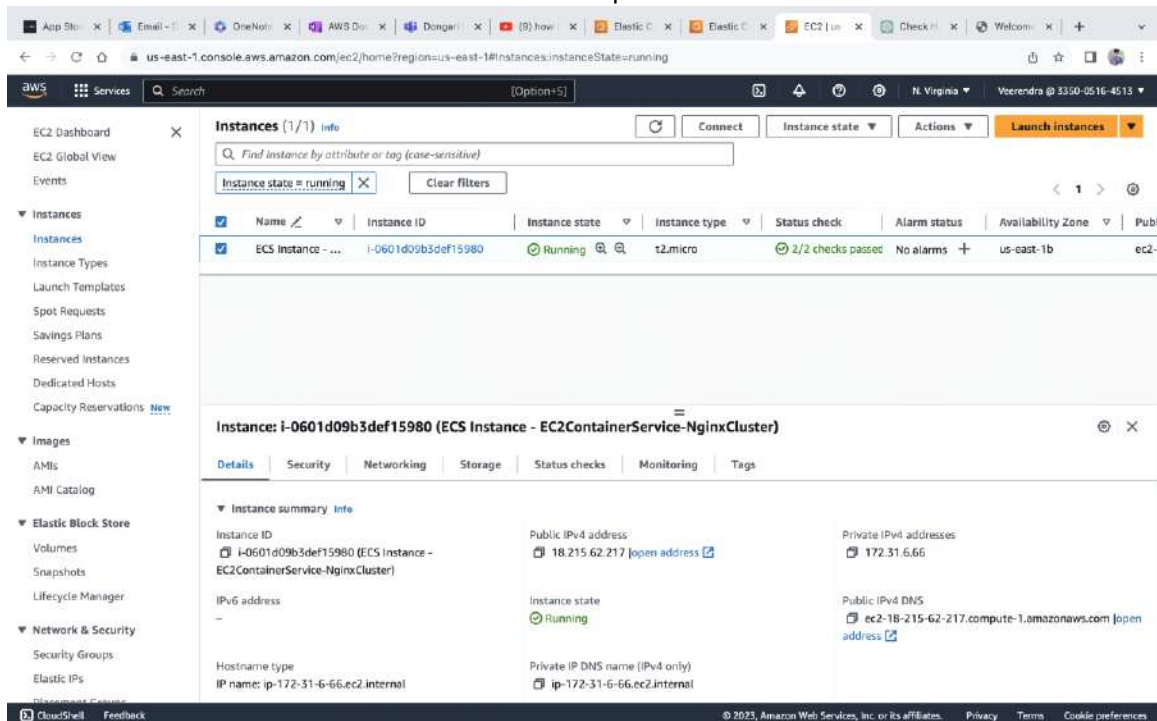


17. Click Edit inbound rules and add configuration.



18. Click on Add rule and add port number and select AnyWhere IPv4 in Source dropdown and click on save rule.

19. After that go to EC2 instance and go to Details tab and copy public IPv4 address and then paste in browser and click enter then it will show the output.



20. This is the process for pushing the local docker image to AWS ECR and running container in AWS

