Day-48 – Deploy App Using ECS And ECR

Today will be a great learning for sure. I know many of you may not know about the term "ECS". As you know, 90 Days of DevOps Challange is mostly about 'learning new', let's learn then ;)

What is Amazon ECS?
Elastic Container Service (ECS) is a container management service that is highly scalable and fast. Managing containers and performing operations like start/stop is very easy on ECS. Containers in ECS are defined in a task definition within a service and service is a configuration that runs and maintains a specified number of tasks in a cluster. Tasks can be run on a serverless infrastructure that is managed by AWS Faregate or on a cluster of Amazon EC2 instances that is managed by the user.

What is Amazon EKS?
Elastic Kubernetes Service, EKS, is a managed service that can be used to run Kubernetes on AWS. There is no need to install, operate, and maintain the Kubernetes control plane or nodes while using EKS. To ensure high availability, EKS runs Kubernetes control plane instances across multiple Availability Zones. When the nodes are unhealthy, EKS automatically replaces them. EKS provides scalability and security to the applications.

What is meant by Orchestration?

Orchestration is the automated configuration, management, and coordination of computer systems, applications, and services. Orchestration helps IT to more easily manage complex tasks and workflows.

Difference between EKS and ECS?
1] Networking: - EKS allows up to 750 pods per instance whereas ECS accommodates only up to a maximum of 120 tasks per instance.

2] Namespaces: - EKS has the concept of namespaces which isolates workloads running in the same cluster, whereas ECS does not have such concept in it.NameSpaces provide a lot of Advantages.

3]Ease of Use: - ECS is very straight forward and does not have a lot of components to learn whereas EKS is more complex as it uses Kubernetes which is altogether a vast technology to learn and requires expertise for deployments.

4]Pricing and costs: - Amazon EKS consists of a control plane and infrastructure,AWS Farget or AWS EC2 to host containerized applications. The Pricing of EKS or ECS is dependent of infrastructure (EKS or ECS) being used to host containerized applications.

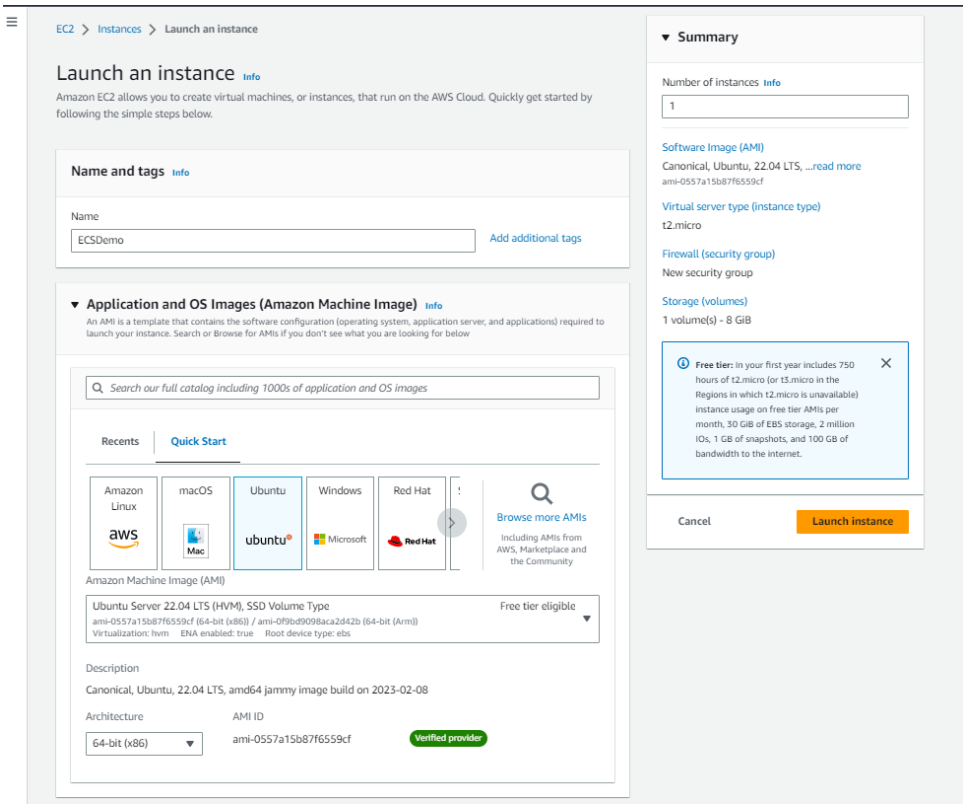
5] portability or compatibility: - Both EKS or ECS are managed services of AWS.ECS is an AWS proprietary service whereas EKS is a Kubernetes as-a-platform service by AWS.All the applications running on EKS clusters can be run on any other Kubernetes cluster with a little or no change, whereas deploying applications on ECS means using the proprietary container platform of AWS.

Task:

Set up ECS (Elastic Container Service) by setting up Nginx on ECS.

Here for Demo purpose, we use base OS I.e., Linux **Ubuntu**.

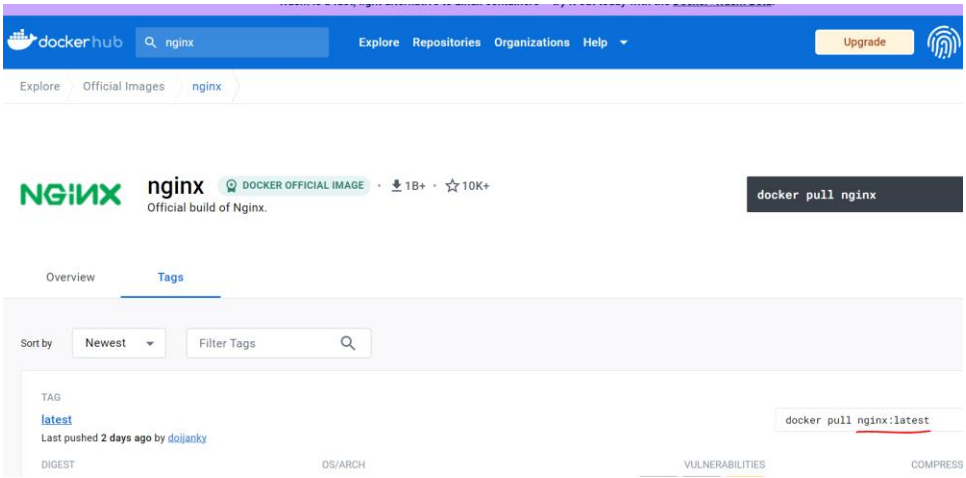
Create a new EC2 Instance.



Here we use Containerization tool I.e., **Docker** to make a container for an App.

```
ubuntu@ip-172-31-61-108:~$ sudo apt-get update && sudo apt-get install docker.io
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 38 not upgraded.
Need to get 66.8 MB of archives.
After this operation, 287 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-1ubuntu3 [34.4 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 runc amd64 1.1.0-0ubuntu1.1 [4242 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 containerd amd64 1.5.9-0ubuntu3.1 [28.1 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 dns-root-data all 2021011101 [5256 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dnsmasq-base amd64 2.86-1.1ubuntu0.1 [354 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 docker.io amd64 20.10.12-0ubuntu4 [34.0 MB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 ubuntu-fan all 0.12.16 [35.2 kB]
Fetched 66.8 MB in 2s (34.2 MB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 63605 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.6-1_amd64.deb ...
Unpacking pigz (2.6-1) ...
Selecting previously unselected package bridge-utils.
Preparing to unpack .../1-bridge-utils_1.7-1ubuntu3_amd64.deb ...
Unpacking bridge-utils (1.7-1ubuntu3) ...
Selecting previously unselected package runc.
Preparing to unpack .../2-runc_1.1.0-0ubuntu1.1_amd64.deb ...
Unpacking runc (1.1.0-0ubuntu1.1) ...
Selecting previously unselected package containerd.
Preparing to unpack .../3-containerd_1.5.9-0ubuntu3.1_amd64.deb ...
Unpacking containerd (1.5.9-0ubuntu3.1) ...
Selecting previously unselected package dns-root-data.
Preparing to unpack .../4-dns-root-data_2021011101_all.deb ...
Unpacking dns-root-data (2021011101) ...
Selecting previously unselected package dnsmasq-base.
Preparing to unpack .../5-dnsmasq-base_2.86-1.1ubuntu0.1_amd64.deb ...
Unpacking dnsmasq-base (2.86-1.1ubuntu0.1) ...
Selecting previously unselected package docker.io.
```

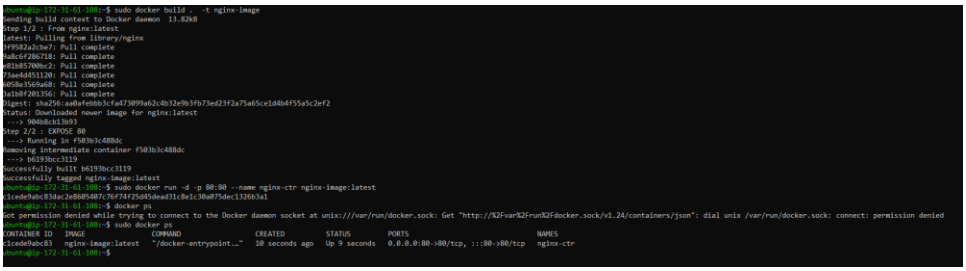
Copy the latest Nginx image from Docker Hub.



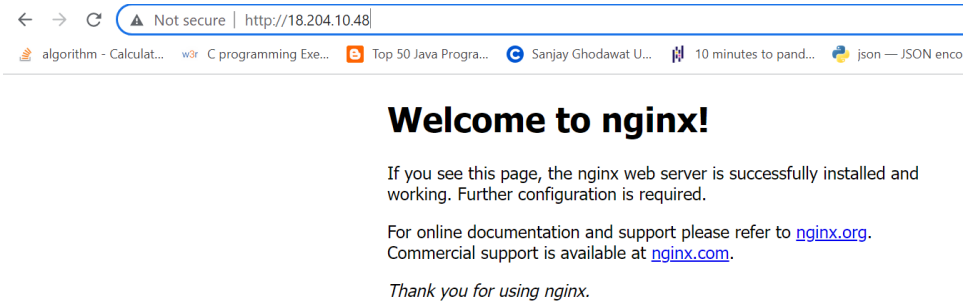
Make a DockerFile.

```
ubuntu@ip-172-31-61-108:~$ vi Dockerfile
ubuntu@ip-172-31-61-108:~$ cat Dockerfile
From nginx:latest
EXPOSE 80
```

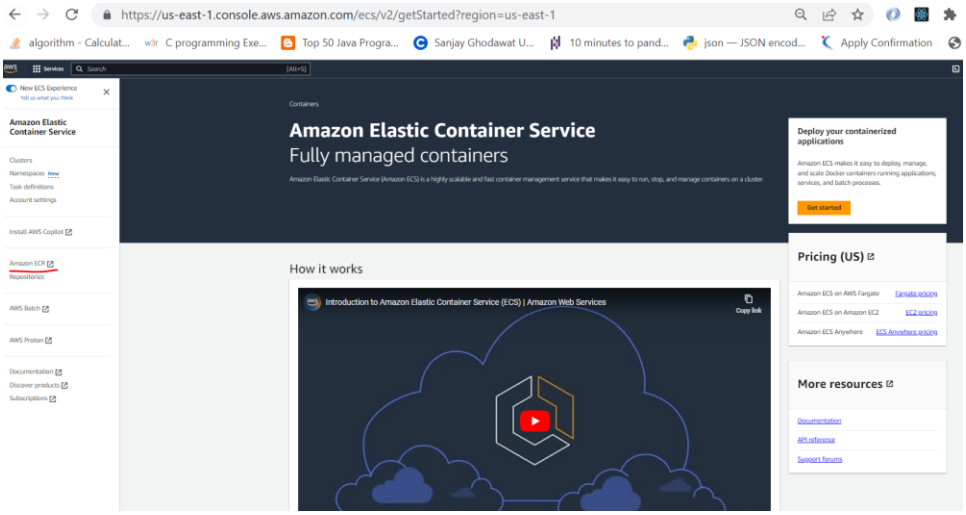
Build and Run Docker File and generate a Docker Image and Docker Container.

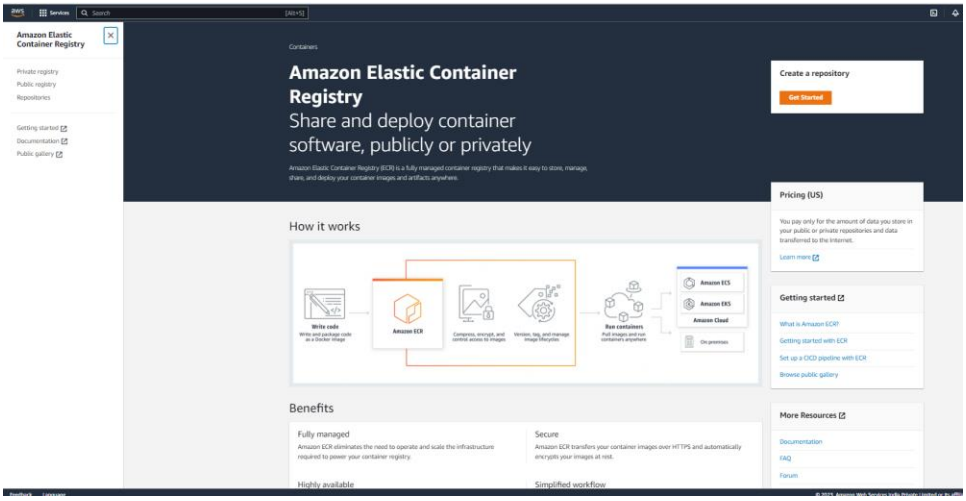


Copy the EC2 Public Address and Hit on Browser and First Check Your App is Running on Local Machine.

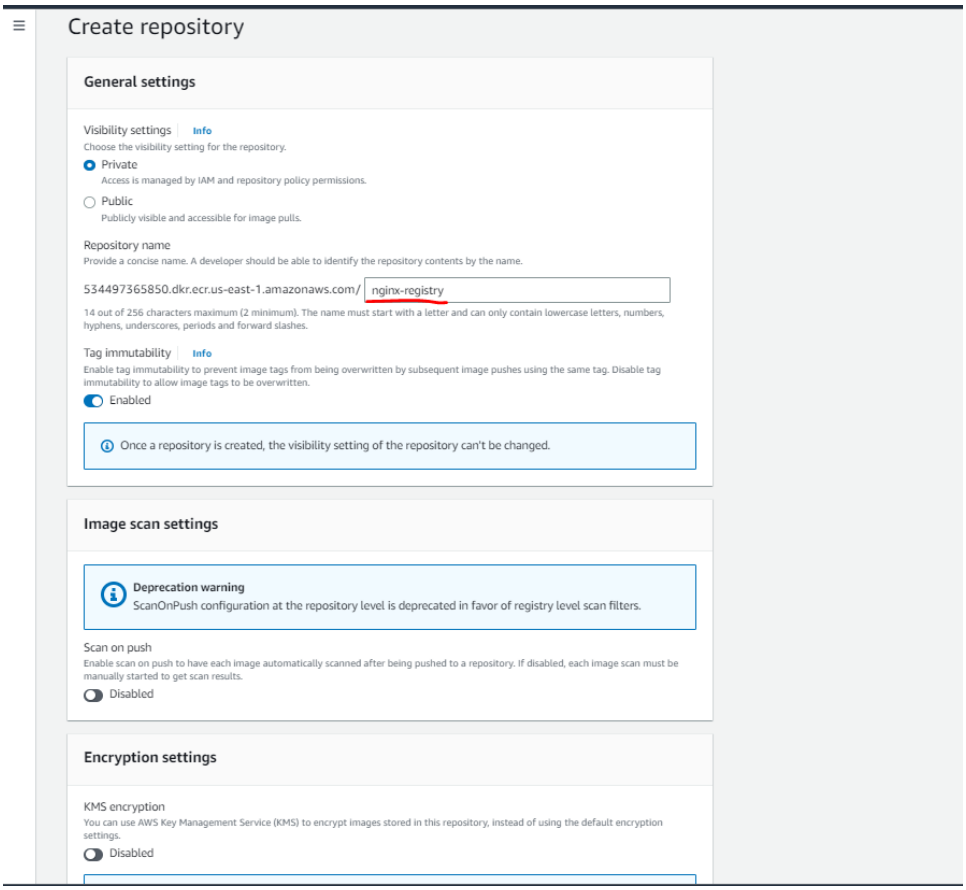


If Your App is Perfectly Run on Local, then you can Push This image to ECR (Elastic Container Registry)

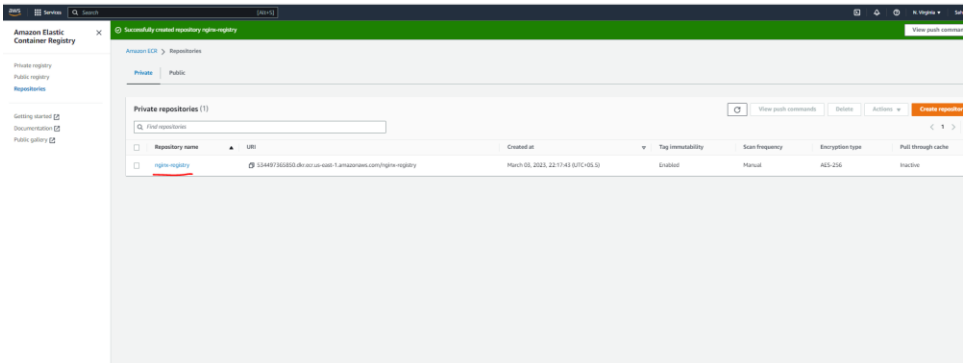




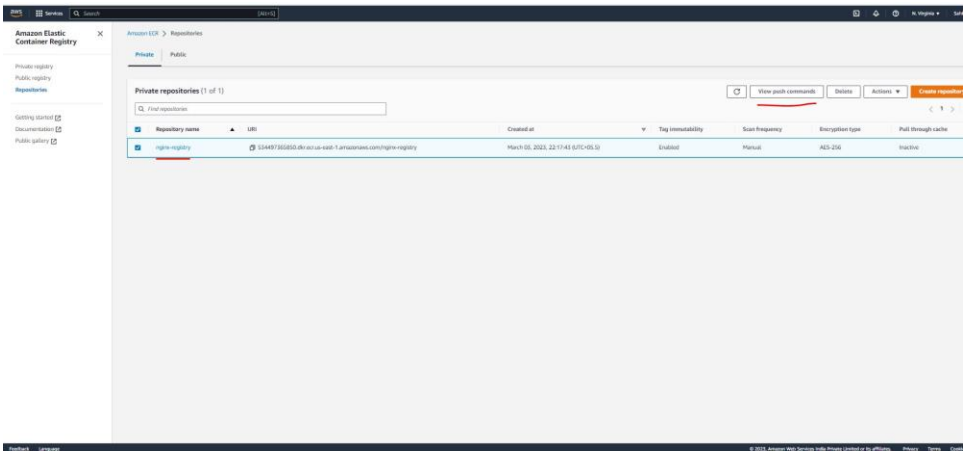
First Make a Registry in ECR,Follow the Below Steps to Create a Registry.



ECR Repository is Created Successfully.



Click on View Push Commands, Tō see all Push Commands.

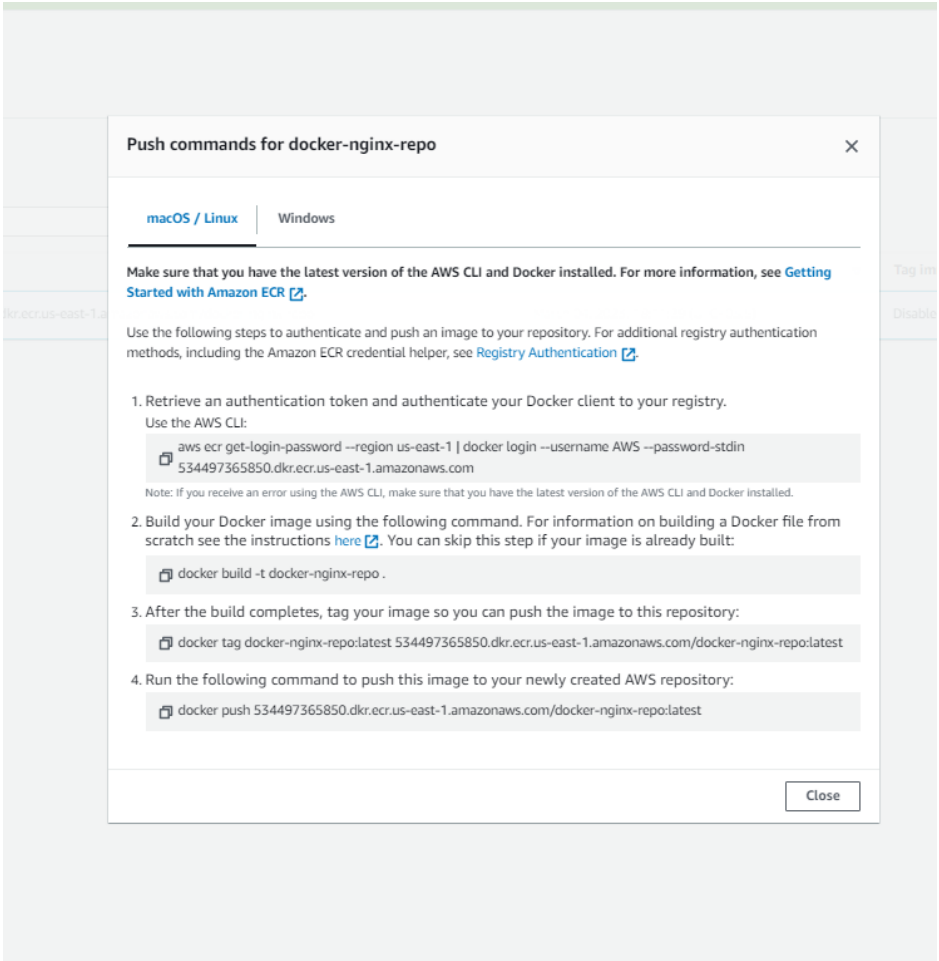


Go to **Ubuntu cmd prompt** and install **awscli**, for communicating to ECR.

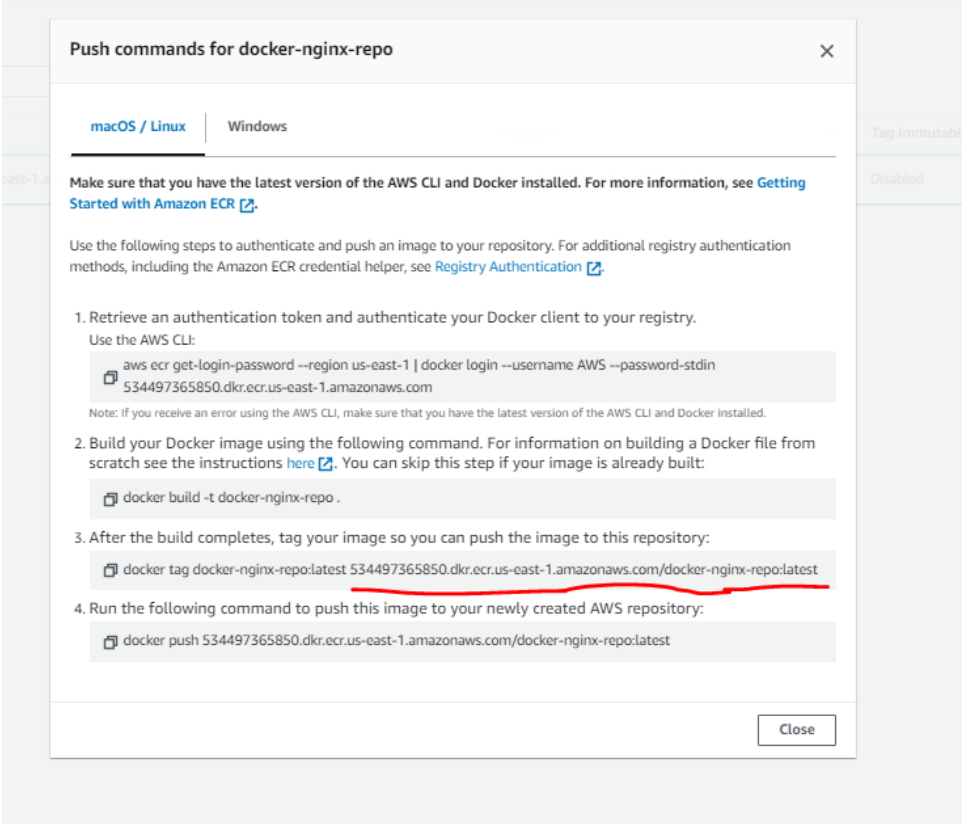
sudo apt install awscli –y **---> Only for Ubuntu OS**

```
ubuntu@ip-172-31-61-108:~$ aws --version
aws-cli/2.11.0 Python/3.11.2 Linux/5.15.0-1028-aws exe/x86_64.ubuntu.22 prompt/off
ubuntu@ip-172-31-61-108:~$
```

ECR Push Commands



Login in to ECR using aws cli

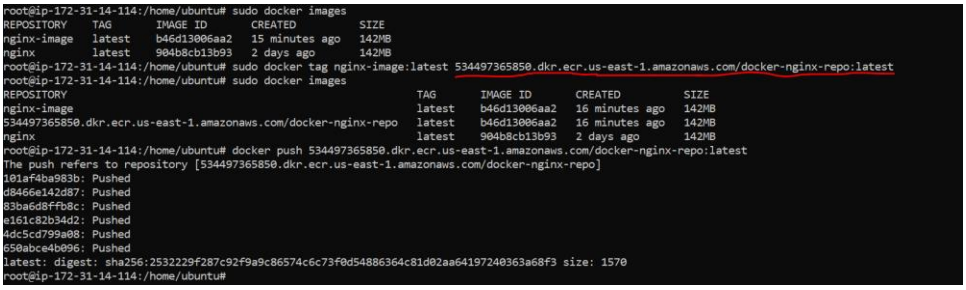


After ECR Login, try to tag an image name.

Syntax is: -

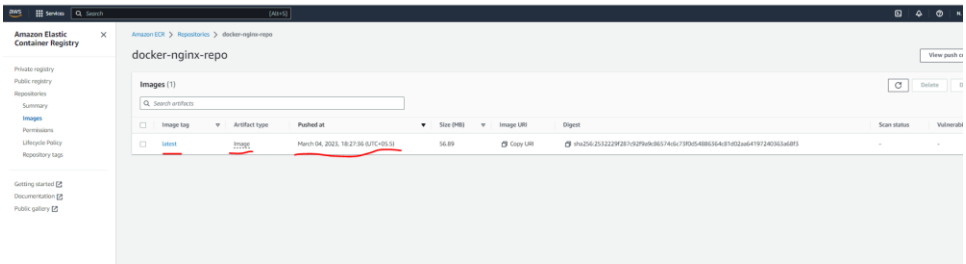
Docker tag <image_name><Above image mentioned Full path>

After that Using Push Command, Push the Docker tagged image to ECR Repository.

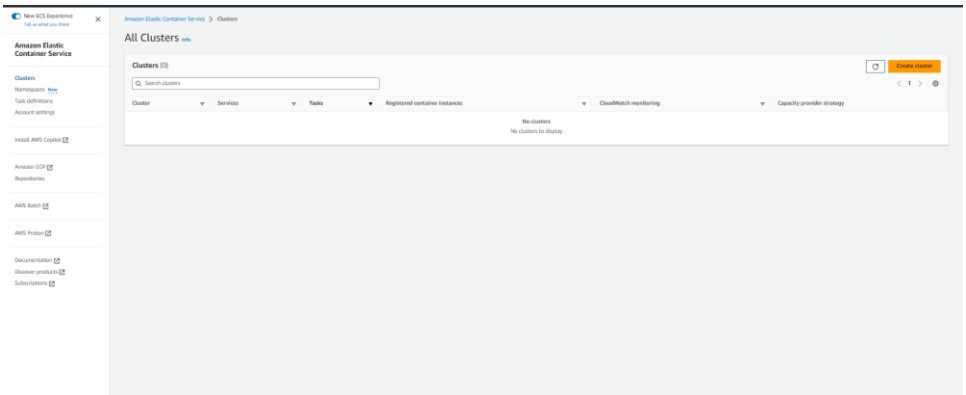


```
root@ip-172-31-14-114:/home/ubuntu# sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
nginx-image   latest    b46d13006aa2   15 minutes ago   142MB
nginx         latest    904b8cb13b93   2 days ago       142MB
root@ip-172-31-14-114:/home/ubuntu# sudo docker tag nginx-image:latest 534497365850.dkr.ecr.us-east-1.amazonaws.com/docker-nginx-repo:latest
root@ip-172-31-14-114:/home/ubuntu# sudo docker images
REPOSITORY                                                          TAG       IMAGE ID       CREATED          SIZE
nginx-image                                                         latest    b46d13006aa2   16 minutes ago   142MB
534497365850.dkr.ecr.us-east-1.amazonaws.com/docker-nginx-repo      latest    b46d13006aa2   16 minutes ago   142MB
nginx                                                               latest    904b8cb13b93   2 days ago       142MB
root@ip-172-31-14-114:/home/ubuntu# docker push 534497365850.dkr.ecr.us-east-1.amazonaws.com/docker-nginx-repo:latest
The push refers to repository [534497365850.dkr.ecr.us-east-1.amazonaws.com/docker-nginx-repo]
101af4ba983b: Pushed
d8466e142d87: Pushed
83ba6d8ffb8c: Pushed
e161c82b34d2: Pushed
4dc5cd799a08: Pushed
650abce4b096: Pushed
latest: digest: sha256:2532229f287c92f9a9c86574c6c73f0d54886364c81d02aa64197240363a68f3 size: 1570
root@ip-172-31-14-114:/home/ubuntu#
```

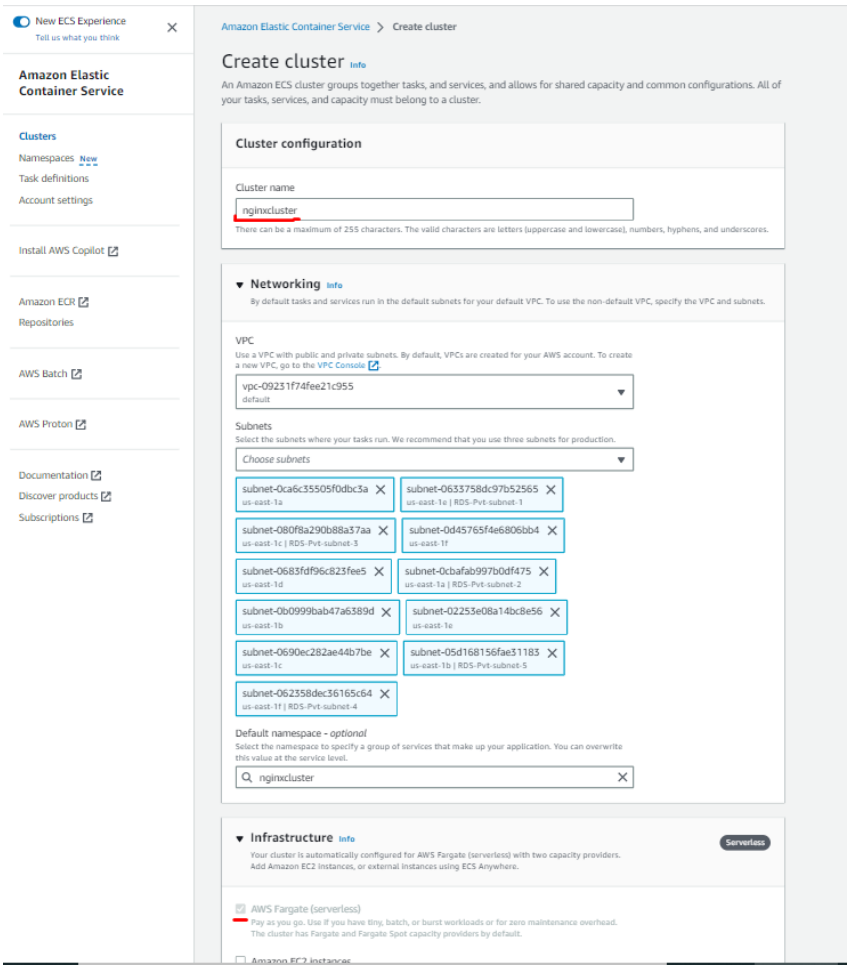
See Our Docker nginx image is Successfully pushed into ECR Repository.



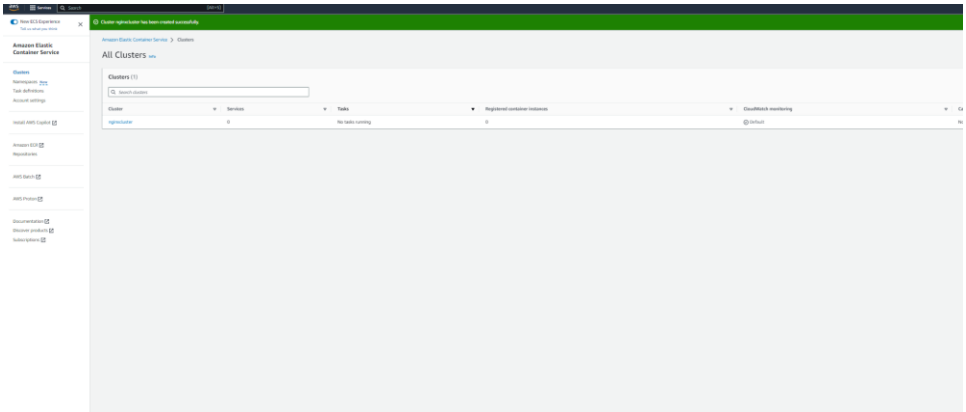
Goto ECS (Elastic Container Service) and Follow the Below Steps to create a Cluster.



Put necessary information and click on next and create.

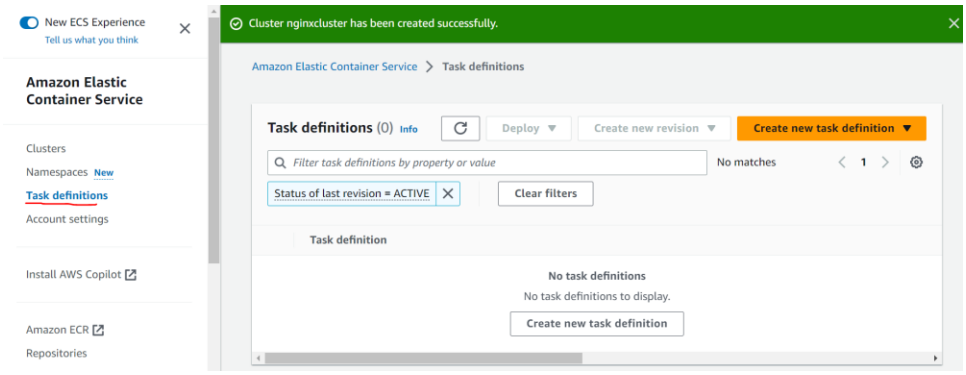


ECS Cluster is Created successfully.

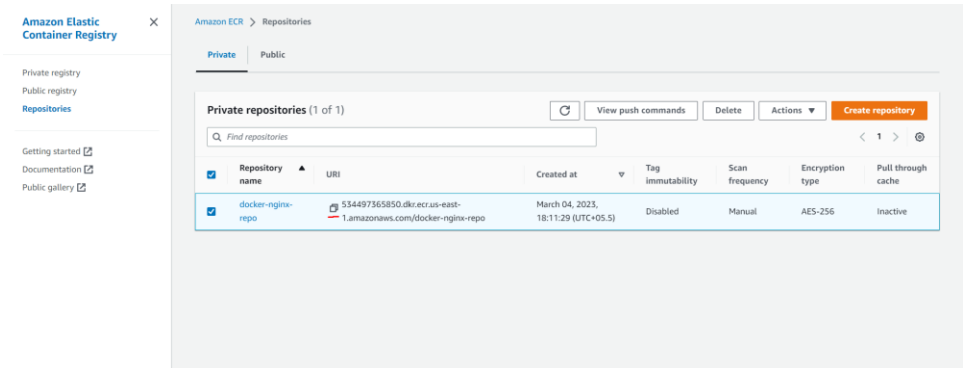


After Creating cluster, you need to define a **TASK**.

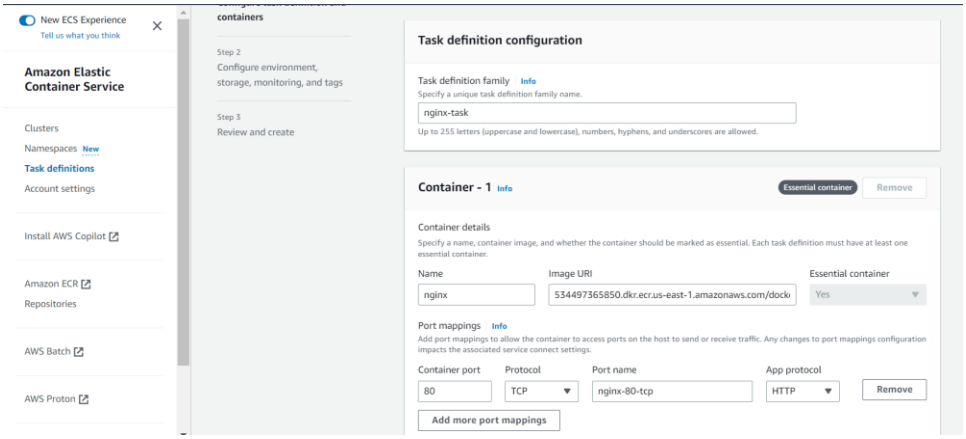
So, click on **Create a new task definition**.



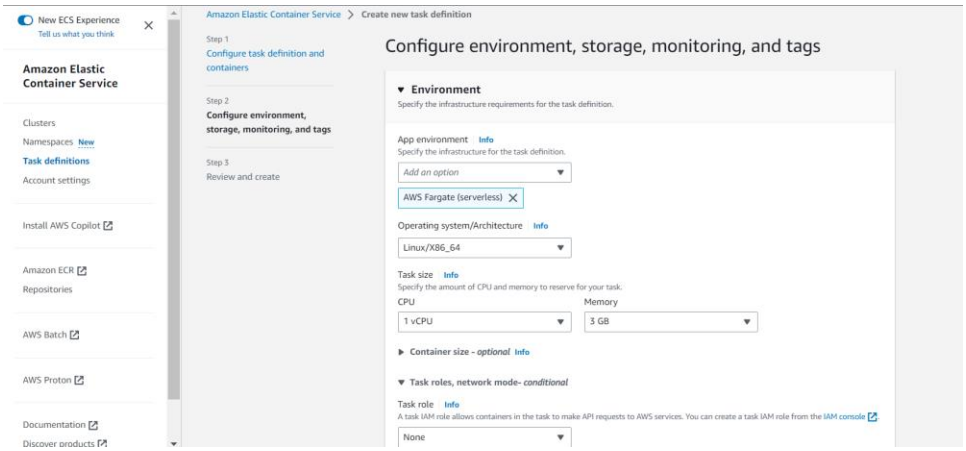
Copy This ECR Image URI and paste in below **container Image URI Section**.

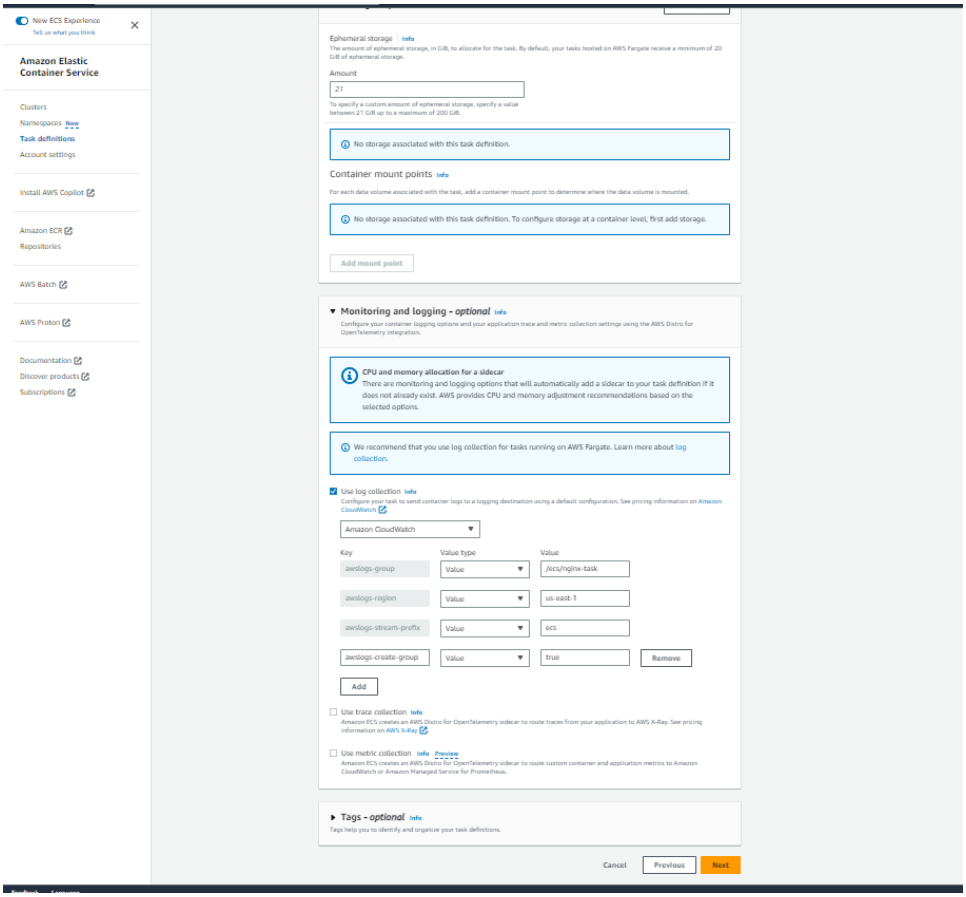


Give Task name and add Container port Name.

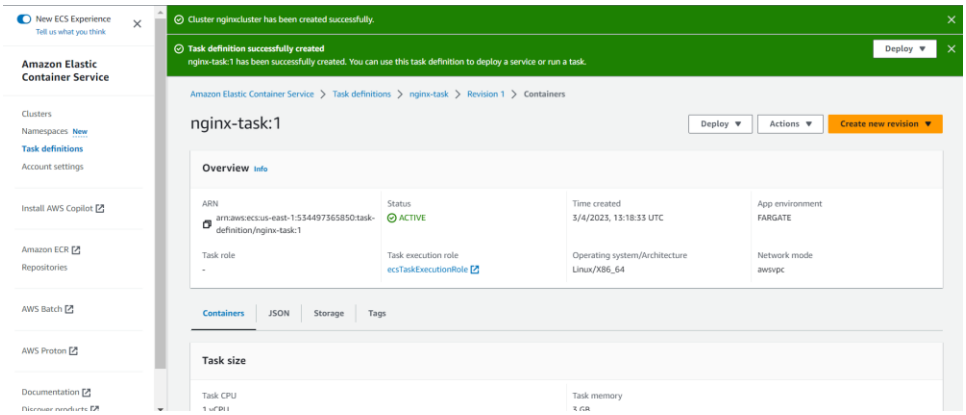


Select **Farget** as an Environment Type.

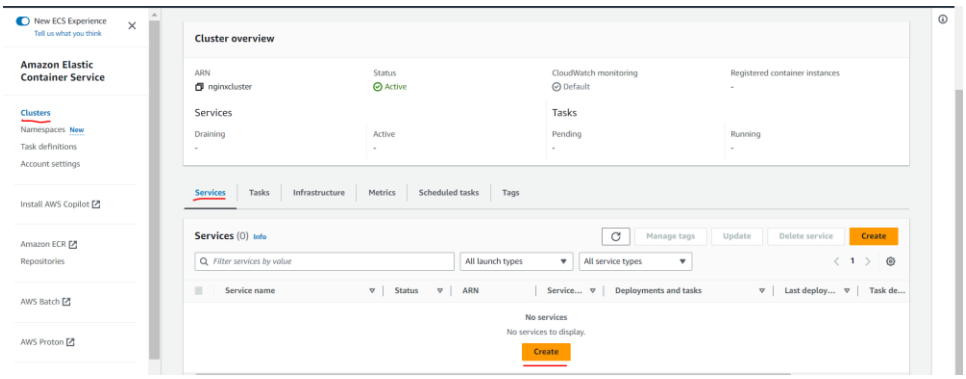


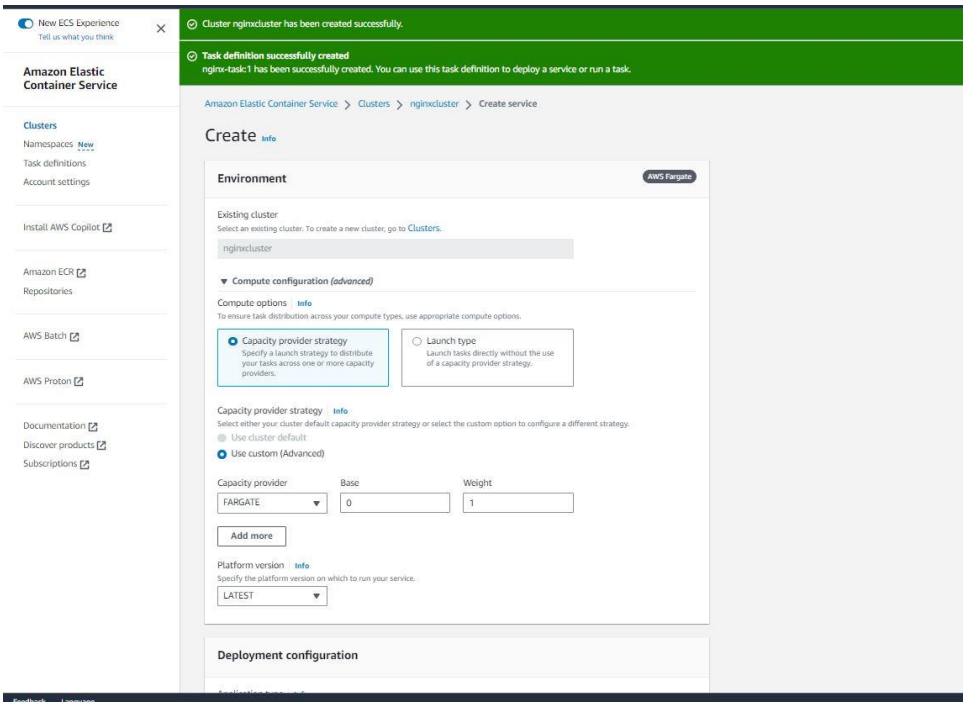


Your **Task Definition** is Created Successfully.

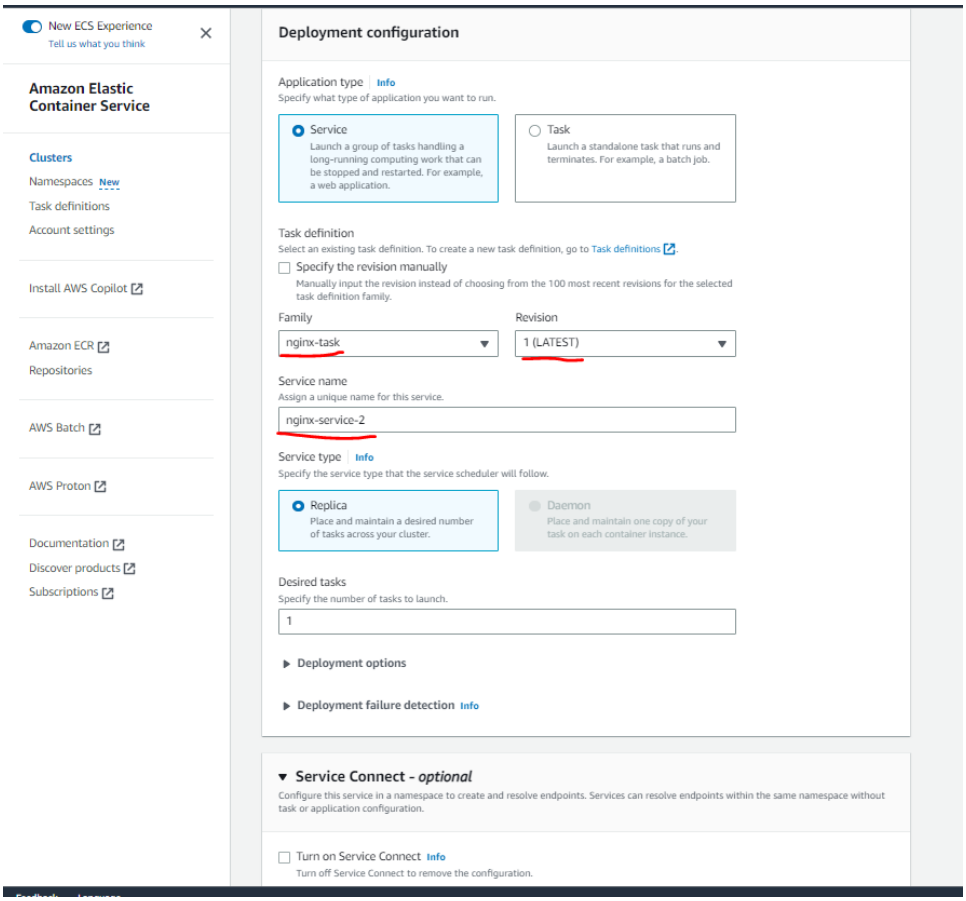


After Creating Task Definitions, Create A service for Deployment of App.

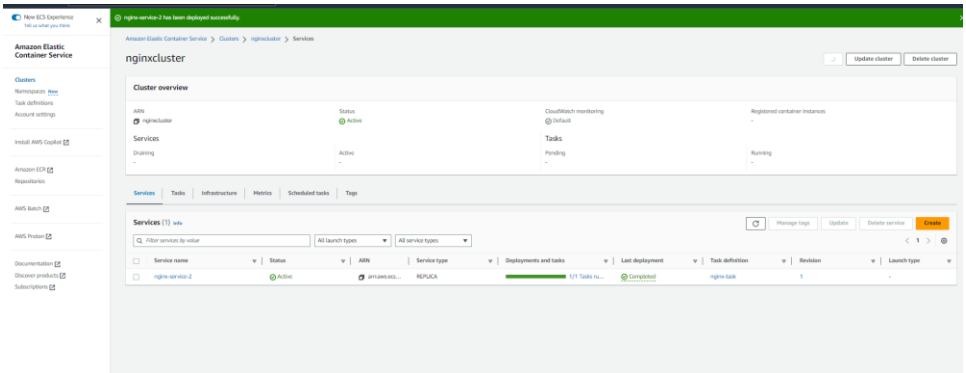




Select Already Created Task Name and Give the Service name.



Your Service is Created.

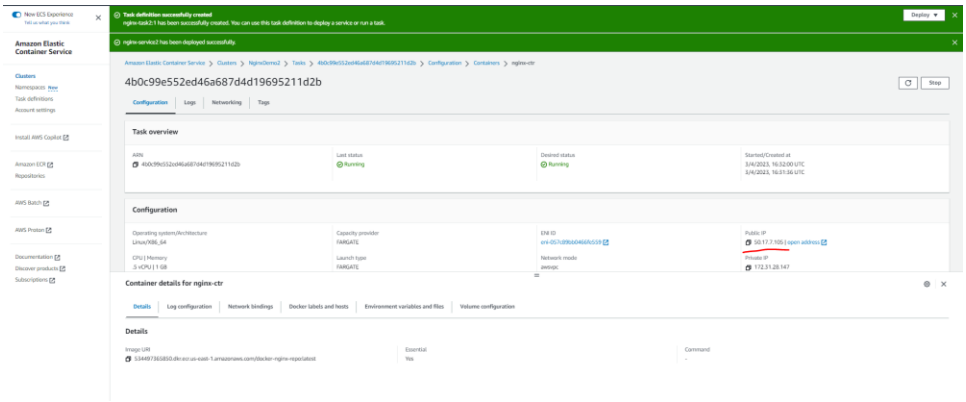


Now, Your App is Deployed.

Check your app is Running, follow below path to copy the Public Address.

Goto Cluster, click on Tasks/Configuration

Inside this Configuration, you will find the Public Address.



Hit this Public Address to your browser.

Hey!!! Your App is Running