Docker & AWS ECS

CONTAINERS & DOCKER

- Containers are an abstraction at the app layer that packages code and dependencies together.
- Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.
- *Docker is an open-source project that automates the deployment of software applications inside* ***containers*** *by providing an additional layer of abstraction and automation of* ***OS-level virtualization*** *on Linux.*

SETTING UP DOCKER

- https://docs.docker.com/docker-for-mac/install/
- https://docs.docker.com/docker-for-windows/install/

Once you are done installing Docker, test your Docker installation by running the following:

```
$ docker run hello-world
```

STEPS TO DEPLOY IN AWS ECS

- Write a docker file in root directory of your application

Dockerfile

hello-world > Dockerfile

```
1   FROM node:10.15.1
2   RUN mkdir -p /usr/src/app
3   WORKDIR /usr/src/app
4   COPY . .
5   RUN npm install
6   EXPOSE 3000
7   CMD [ "npm", "start" ]
```

STEPS TO DEPLOY IN AWS ECS

- Build the docker image:

  $docker build -t demo .

```
C:\Users\kavya\OneDrive\Desktop\inclass-react\hello-world>docker build -t demo .
Sending build context to Docker daemon  669.2kB
Step 1/7 : FROM node:10.15.1
 ---> 8fc2110c6978
Step 2/7 : RUN mkdir -p /usr/src/app
 ---> Using cache
 ---> 4a230b105cfa
Step 3/7 : WORKDIR /usr/src/app
 ---> Using cache
 ---> e63d3f287919
Step 4/7 : COPY . .
 ---> Using cache
 ---> d2a92bdd3698
Step 5/7 : RUN npm install
 ---> Using cache
 ---> e43f915ae56a
Step 6/7 : EXPOSE 3000
 ---> Using cache
 ---> 214802f308e6
Step 7/7 : CMD [ "npm", "start" ]
 ---> Using cache
 ---> 0c86f524c8a5
Successfully built 0c86f524c8a5
Successfully tagged demo:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context w
ill have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.
```

STEPS TO DEPLOY IN AWS ECS

- Check for image id:

  $docker images

```
C:\Users\kavya\OneDrive\Desktop\inclass-react\hello-world>docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
607531887380.dkr.ecr.us-east-1.amazonaws.com/demo         latest    0c86f524c8a5   24 hours ago     1.13GB
demo                                                      latest    0c86f524c8a5   24 hours ago     1.13GB
hello                                                     latest    0c86f524c8a5   24 hours ago     1.13GB
607531887380.dkr.ecr.us-west-1.amazonaws.com/microservicesmongodeploy   latest    014218057d50   2 weeks ago      717MB
```

STEPS TO DEPLOY IN AWS ECS

- Run the container with that image id (optional)

  $docker run -p 3000:3000 0c86f524c8a5

```
C:\Users\kavya\OneDrive\Desktop\inclass-react\hello-world>docker run -p 3000:3000 0c86f524c8a5

> hello-world@0.1.0 start /usr/src/app
> react-scripts start

Starting the development server...
```

STEPS TO DEPLOY IN AWS ECS

- Create a Repository in AWS ECR:

  Use push commands to push the image to ecr:

  $(aws ecr get-login --no-include-email --region us-east-1)

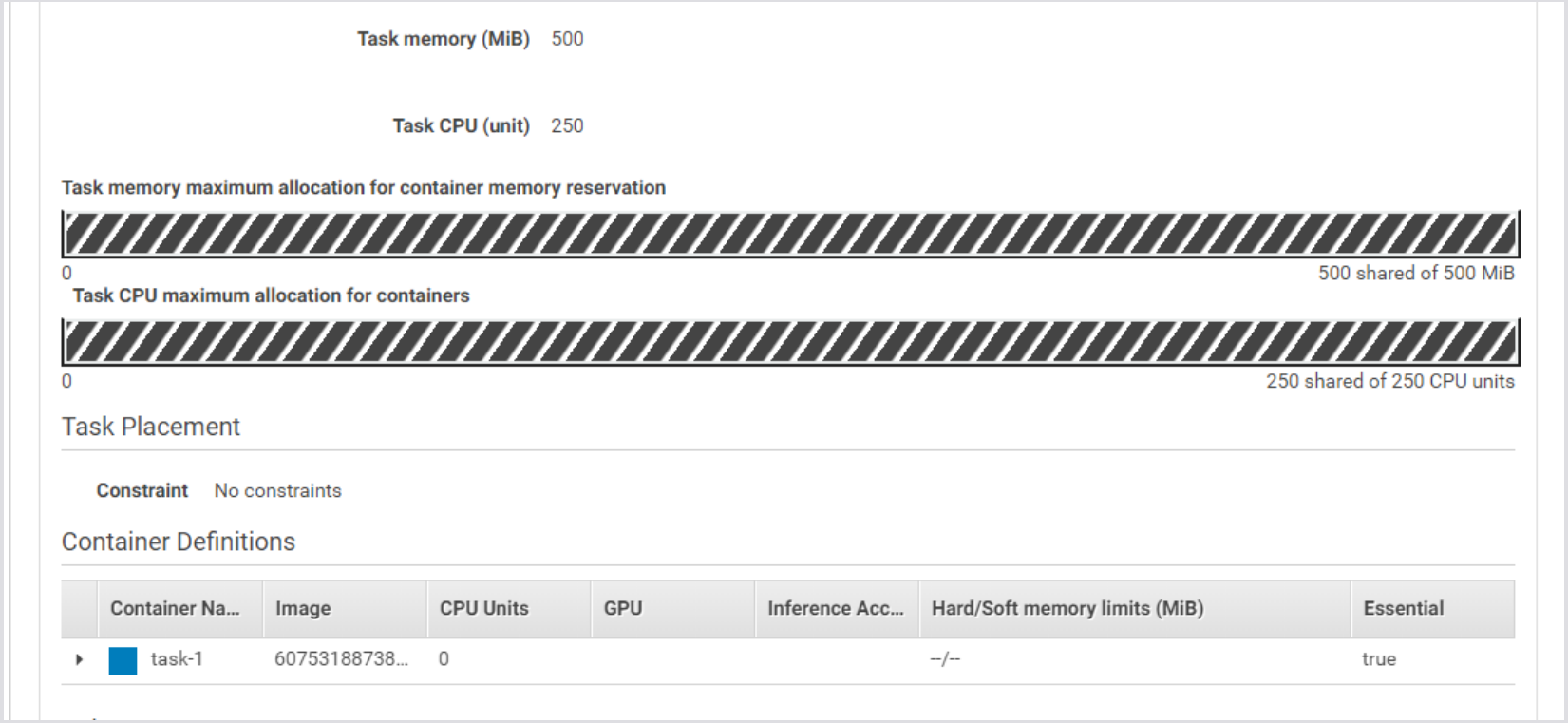
  docker build -t demo .

  docker tag demo:latest 607531887380.dkr.ecr.us-east-1.amazonaws.com/demo:latest

  docker push 607531887380.dkr.ecr.us-east-1.amazonaws.com/demo:latest

STEPS TO DEPLOY IN AWS ECS

Create a TASK DEFINITION: WITH CONTAINER IMAGE POINTING TO IMAGE IN ECR



Task memory (MiB) 500

Task CPU (unit) 250

Task memory maximum allocation for container memory reservation

0 — 500 shared of 500 MiB

Task CPU maximum allocation for containers

0 — 250 shared of 250 CPU units

Task Placement

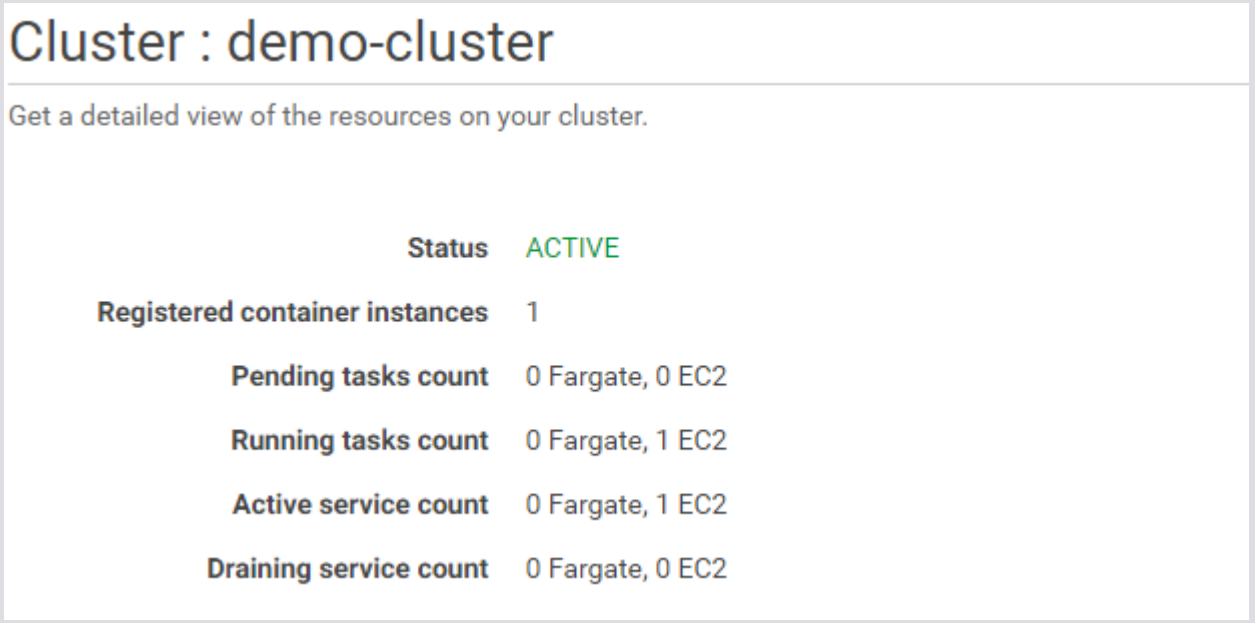
Constraint No constraints

Container Definitions

Container Na...	Image	CPU Units	GPU	Inference Acc...	Hard/Soft memory limits (MiB)	Essential
task-1	60753188738...	0			--/--	true

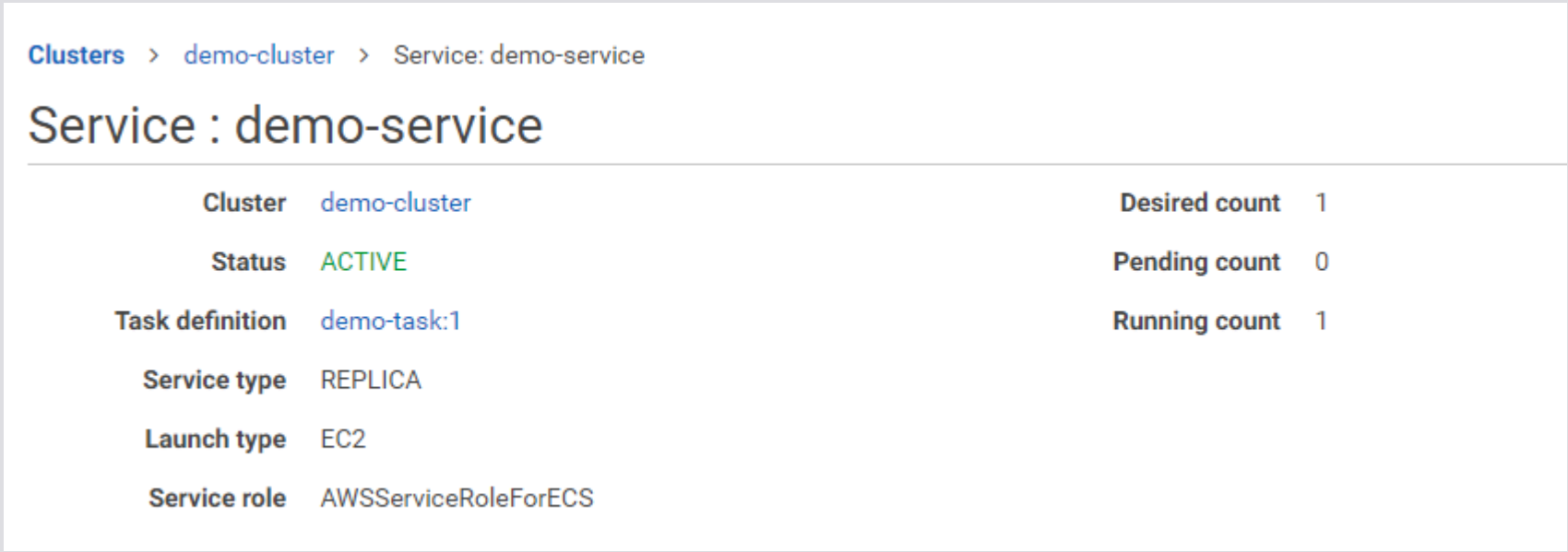
STEPS TO DEPLOY IN AWS ECS

Create a CLUSTER to run a service:



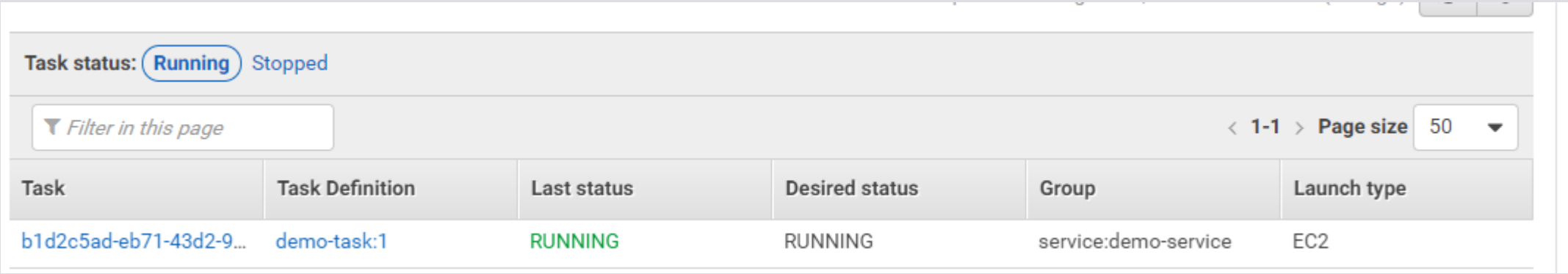
STEPS TO DEPLOY IN AWS ECS

- Create a SERVICE: in the cluster created running the previously created task



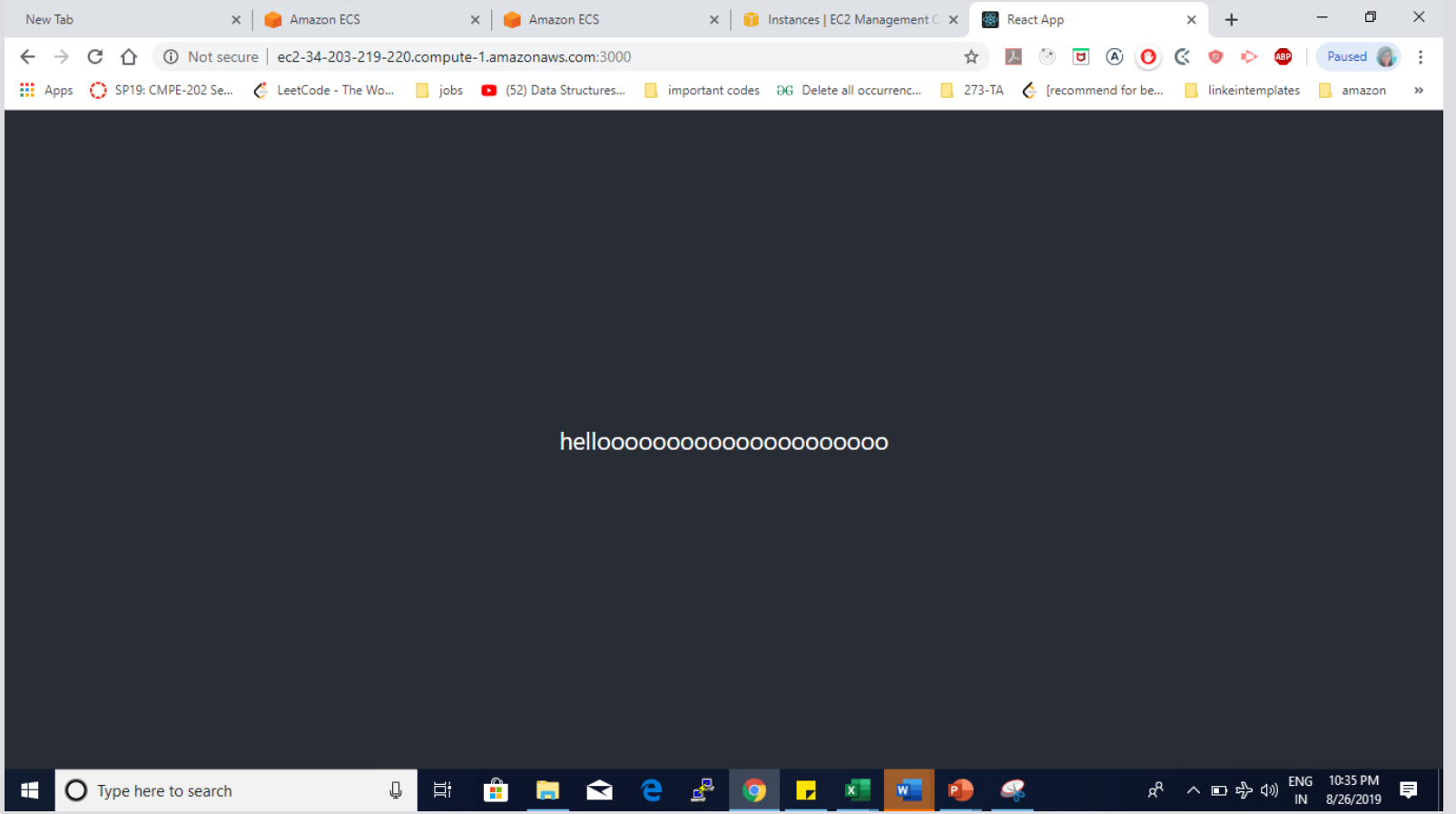
STEPS TO DEPLOY IN AWS ECS

After making sure the task is in the running state , go to the instance and check the inbound rules to allow the exposed port 3000.



Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
Custom TCP Rule	TCP	3000	0.0.0.0/0	
Custom TCP Rule	TCP	3000	::/0	

STEPS TO DEPLOY IN AWS ECS



References:

- https://www.docker.com/
- https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html