

1. Install docker and then create a container (Nginx) with docker CLI (volume, network, port).

→ Docker is an open-source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify the delivery of distributed applications and have become increasingly popular as organizations shift to cloud-native development and hybrid multi-cloud environments.

Developers can create containers without Docker, but the platform makes it easier, simpler, and safer to build, deploy and manage containers. Docker is essentially a toolkit that enables developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation through a single API.

→ To install docker we will be using the convenience script.

◆ Let's invoke the following commands one after another:

- `curl -fsSL https://get.docker.com -o get-docker.sh`
- `DRY_RUN=1 sh ./get-docker.sh`
 - 'DRY_RUN=1' is optional, we can use it if we want to learn what steps the script will execute during installation.

```
rtkeshkarma@pop-os:~$ curl -fsSL https://get.docker.com -o get-docker.sh
rtkeshkarma@pop-os:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 93d2499759296ac1f9c510605fef85052a2c32be
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-certificates curl >/dev/null
+ sh -c curl -fsSL "https://download.docker.com/linux/debian/gpg" | gpg --dearmor --yes -o /usr/share/keyrings/docker-archive-keyring.gpg
+ sh -c echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian bullseye stable" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommends docker-ce-cli docker-scanner-plugin docker-ce >/dev/null
+ version_gte 20.10
+ [ -z ]
+ return 0
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras >/dev/null
+ sh -c docker version
Client: Docker Engine - Community
Version:      20.10.11
API version:  1.41
Go version:   go1.16.9
Git commit:   dea9396
Built:        Thu Nov 18 00:37:11 2021
OS/Arch:      linux/amd64
Context:      default
Experimental: true

Server: Docker Engine - Community
Engine:
Version:      20.10.11
```

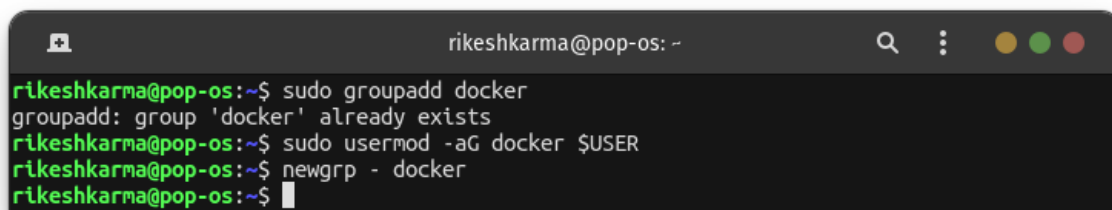
◆ Now we can verify our installation by using the following command:

- `docker version`

```
rikeshkarma@pop-os:~$ docker version
Client: Docker Engine - Community
 Version: 20.10.11
 API version: 1.41
 Go version: go1.16.9
 Git commit: dea9396
 Built: Thu Nov 18 00:37:11 2021
 OS/Arch: linux/amd64
 Context: default
 Experimental: true

Server: Docker Engine - Community
 Engine:
  Version: 20.10.11
  API version: 1.41 (minimum version 1.12)
  Go version: go1.16.9
  Git commit: 847da18
  Built: Thu Nov 18 00:35:17 2021
  OS/Arch: linux/amd64
  Experimental: false
 containerd:
  Version: 1.4.12
  GitCommit: 7b11cfaabd73bb80907dd23182b9347b4245eb5d
 runc:
  Version: 1.0.2
  GitCommit: v1.0.2-0-g52b36a2
 docker-init:
  Version: 0.19.0
  GitCommit: de40ad0
```

- After we verify docker installation, let's add a docker group, add our current user to the docker group and switch the session to the docker group running the following commands:
 - `sudo groupadd docker`
 - `sudo usermod -aG docker $USER`
 - `newgrp - docker`



```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo groupadd docker
groupadd: group 'docker' already exists
rikeshkarma@pop-os:~$ sudo usermod -aG docker $USER
rikeshkarma@pop-os:~$ newgrp - docker
rikeshkarma@pop-os:~$
```

→ Now to create an Nginx container we need to run the command below in the terminal:

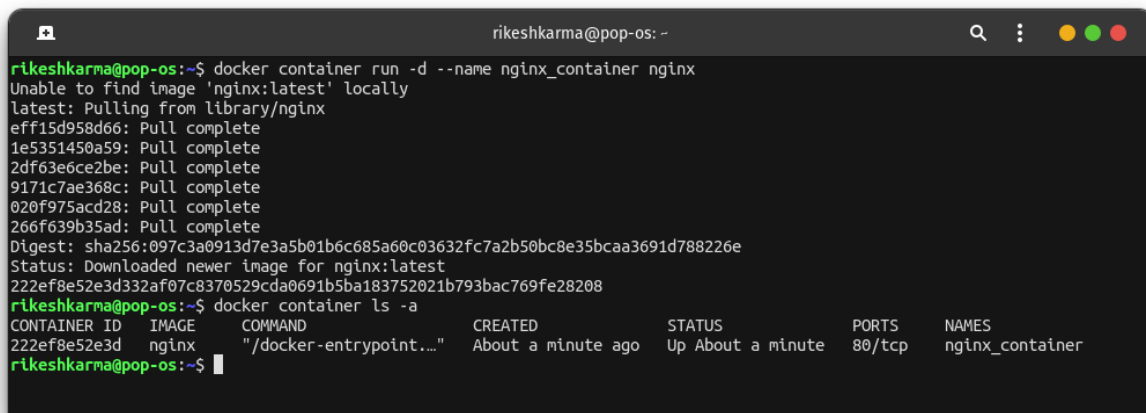
- ◆ `docker container run -d --name nginx_container -v /home/rikeshkarma/docker/index:usr/share/nginx/html --network bridge -p 123:80 nginx`
 - **-d:-** This option is used to run the container in the detached mode.
 - **--name:-** This option is used to provide the name to the container, in our case, `nginx_container` is the name we have provided to our

container. If we don't use this option, the docker will assign the random alphanumeric string to the container.

- **-v:-** This option is used to mount the volume from the host to the container.
- **--network:-** This option will help to choose a network or we can create a new network too.
- **-p:-** This option will help to assign port from host to container (host:container)
- **Nginx:-** This is the name of the Nginx image.

◆ We can list the containers and see the Nginx should be up and running. We can run the following command to see the list of containers:

- *docker container ls -a*



```
rireshkarma@pop-os: ~  
rireshkarma@pop-os:~$ docker container run -d --name nginx_container nginx  
Unable to find image 'nginx:latest' locally  
latest: Pulling from library/nginx  
eff15d958d66: Pull complete  
1e5351450a59: Pull complete  
2df63e6ce2be: Pull complete  
9171c7ae368c: Pull complete  
020f975acd28: Pull complete  
266f639b35ad: Pull complete  
Digest: sha256:097c3a0913d7e3a5b01b6c685a60c03632fc7a2b50bc8e35bcaa3691d788226e  
Status: Downloaded newer image for nginx:latest  
222ef8e52e3d332af07c8370529cda0691b5ba183752021b793bac769fe28208  
rireshkarma@pop-os:~$ docker container ls -a  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES  
222ef8e52e3d   nginx    "/docker-entrypoint..." About a minute ago Up About a minute 80/tcp       nginx_container  
rireshkarma@pop-os:~$
```

- We can see that the Nginx server is running with default port **80** which is actually configured in the Nginx image Dockerfile. If we want to access the Nginx server, we must know the IP address of the Nginx server too. And in docker, each container has its host and IP.
- We can run the following command to display the properties of the container and file out the IP address.
 - *docker container inspect nginx_container | grep IPAddress*
 - ◆ **inspect:-** This command is used to display information on one or more containers.
 - ◆ **grep IPAddress:-** This is not the docker command, however, this is Linux specific command to find out any word from the command's output.

```
rireshkarma@pop-os: ~$ docker container inspect nginx_container | grep IPAddress
    "SecondaryIPAddresses": null,
    "IPAddress": "172.17.0.2",
    "IPAddress": "172.17.0.2",
rireshkarma@pop-os: ~$
```

- From the above command and the snapshot, we can see the result. And we can observe that the container's IP is **172.17.0.2**.

2. Create a docker container with a docker-compose file for mysql8 and also volume mount and port.

→ Firstly let's get Docker Compose on our system by running following commands sequentially:

◆ `sudo curl -L`

`"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`

◆ `sudo chmod +x /usr/local/bin/docker-compose`

→ We can test if our installation was setup correctly by running:

◆ `docker-compose --version`

```
rireshkarma@pop-os: ~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
[sudo] password for rakeshkarma:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 633      100 633    0     0  1461      0  --:--:-- --:--:-- --:--:-- 1461
100 12.1M    100 12.1M    0     0 1480k      0  0:00:08 0:00:08 --:--:-- 2054k
rireshkarma@pop-os: ~$ sudo chmod +x /usr/local/bin/docker-compose
rireshkarma@pop-os: ~$ docker-compose -- version
docker-compose version 1.29.2, build 5becea4c
docker-py version: 5.0.0
CPython version: 3.7.10
OpenSSL version: OpenSSL 1.1.0l 10 Sep 2019
rireshkarma@pop-os: ~$
```

→ Now we can create a file called `docker-compose.yml`. This file is essentially an instructions sheet for Docker where we tell docker what to do while creating the container. We write the following lines of code on the `docker-compose.yml` file as on the snapshot below:

```
docker-compose.yml x
docker-compose.yml
1  version: '3.3'
2
3  services:
4    db:
5      image: mysql:8
6      command: --default-authentication-plugin=mysql_native_password
7      restart: always
8      environment:
9        MYSQL_ROOT_PASSWORD: sql_testpassword
10     volumes:
11       - my-test-db:/var/lib/mysql
12     ports:
13       - '3306:3306'
14   # Names our volume
15   volumes:
16     my-test-db:
17
```

→ Now after this we can start our container by running the following command in the terminal inside the directory where we created our *docker-compose.yml* file.

◆ *docker-compose up*

```
rikeshkarma@pop-os:~/Desktop/mysql8_container$ docker-compose up
Pulling db (mysql:8)...
8: Pulling from library/mysql
a10c77af2613: Pull complete
b76a7eb51ffd: Pull complete
258223f927e4: Pull complete
2d2c75386df9: Pull complete
63e92e4046c9: Pull complete
f5845c731544: Pull complete
bd0401123a9b: Pull complete
3ef07ec35f1a: Pull complete
c93a31315089: Pull complete
3349ed800d44: Pull complete
6d01857ca4c1: Pull complete
4cc13890eda8: Pull complete
Digest: sha256:aeecae58035f3868bf4f00e5fc623630d8b438db9d05f4d8c6538deb14d4c31b
Status: Downloaded newer image for mysql:8
Creating mysql8_container_db_1 ... done
Attaching to mysql8_container_db_1
db_1 | 2021-11-19 08:27:37+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.27-1debian10 started.
db_1 | 2021-11-19 08:27:37+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
db_1 | 2021-11-19 08:27:37+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.27-1debian10 started.
db_1 | 2021-11-19 08:27:37+00:00 [Note] [Entrypoint]: Initializing database files
db_1 | 2021-11-19T08:27:37.290868Z 0 [Warning] [MY-010918] [Server] 'default_authentication_plugin' is deprecated and will be removed in a future release. Please use authentication_policy instead.
db_1 | 2021-11-19T08:27:37.290880Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.27) initializing of server in progress as process 42
db_1 | 2021-11-19T08:27:37.354643Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
```

```

db_1 | 2021-11-19 08:30:34+00:00 [Note] [Entrypoint]: Stopping temporary server
db_1 | 2021-11-19T08:30:34.958625Z 10 [System] [MY-013172] [Server] Received SHUTDOWN from user root. Shutting down mysqld (Version: 8.0.27).
db_1 | 2021-11-19T08:31:11.411210Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 8.0.27) MySQL Community Server - GPL.
db_1 | 2021-11-19 08:31:11+00:00 [Note] [Entrypoint]: Temporary server stopped
db_1 | 2021-11-19 08:31:11+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start up.
db_1 | 2021-11-19T08:31:12.271303Z 0 [Warning] [MY-010918] [Server] 'default_authentication_plugin' is deprecated and will be removed in a future release. Please use authentication_policy instead.
db_1 | 2021-11-19T08:31:12.271325Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.27) starting as process 1
db_1 | 2021-11-19T08:31:12.310193Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
db_1 | 2021-11-19T08:31:14.075917Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
db_1 | 2021-11-19T08:31:15.375272Z 0 [Warning] [MY-013746] [Server] A deprecated TLS version TLSv1 is enabled for channel mysql_main
db_1 | 2021-11-19T08:31:15.375320Z 0 [Warning] [MY-013746] [Server] A deprecated TLS version TLSv1.1 is enabled for channel mysql_main
db_1 | 2021-11-19T08:31:15.376472Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
db_1 | 2021-11-19T08:31:15.376527Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
db_1 | 2021-11-19T08:31:15.575885Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
db_1 | 2021-11-19T08:31:15.601174Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqldx.sock
db_1 | 2021-11-19T08:31:15.601268Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.27' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.

```

→ We have created a container with a docker-compose file for mysql8. We can observe that the MySQL instance is running on *localhost:3306*.

3. Create a Dockerfile for nodejs, react and Nginx and also create a container with docker-compose file and map the port and volume in the same network.

→ To create a Dockerfile for node js:

◆ Firstly I created a simpler Node.js project using the following commands:

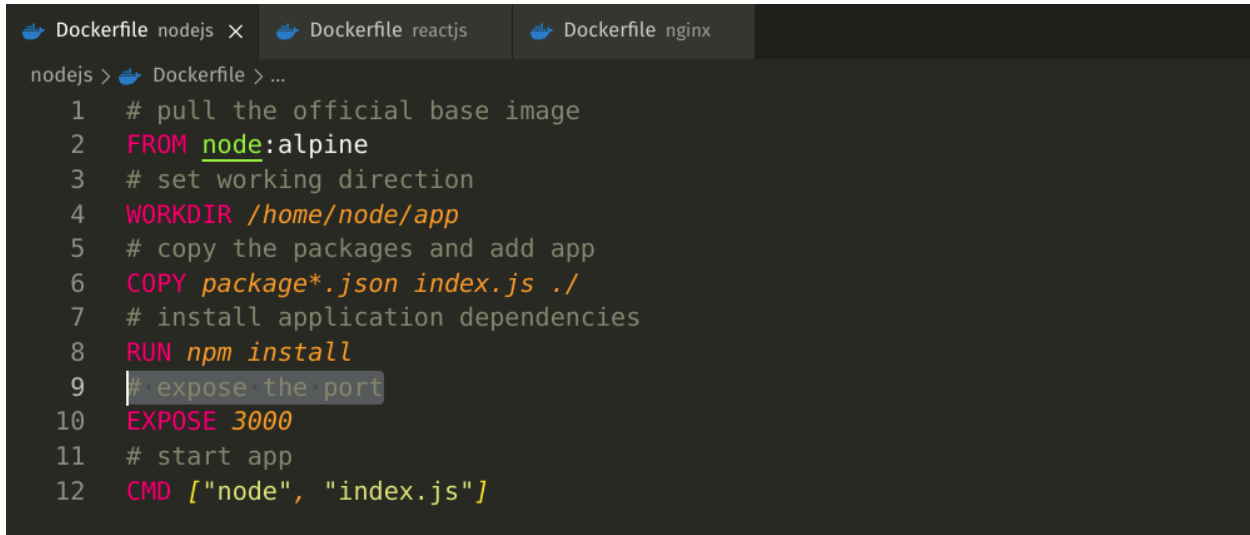
- To initialize the project:
 - *npm init -y*
- To install express.js
 - *npm install express*
- Then I create a file named “index.js” and write this code:

```

index.js > ...
1 // load express module with `require` directive
2 var express = require('express')
3 var app = express()
4
5 // define request response in root URL (/)
6 app.get('/', function (req, res) {
7   res.send('Hello World!')
8 })
9
10 // launch listening server on port 3000
11 app.listen(3000, function () {
12   console.log('app listening on port 3000!')
13 })

```

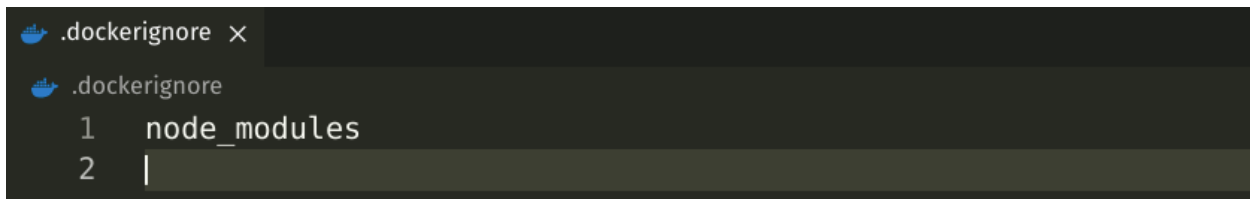
- Then we can run the following command to test if it works:
 - `node index.js`
- ◆ Now after knowing our project is running the file. Let's create a Dockerfile,
 - Let's create a file named "*Dockerfile*" in the project directory.
 - Then write this code inside the Dockerfile:



The screenshot shows a code editor with three tabs: "Dockerfile nodejs", "Dockerfile reactjs", and "Dockerfile nginx". The "Dockerfile nodejs" tab is active, showing the following content:

```
nodejs > Dockerfile > ...
1 # pull the official base image
2 FROM node:alpine
3 # set working direction
4 WORKDIR /home/node/app
5 # copy the packages and add app
6 COPY package*.json index.js ./
7 # install application dependencies
8 RUN npm install
9 # expose the port
10 EXPOSE 3000
11 # start app
12 CMD ["node", "index.js"]
```

- We can also create *.dockerignore* if we want to which will ignore the files/ directory we don't want to push to docker hub. This file works like *.gitignore*.



The screenshot shows a code editor with two tabs: ".dockerignore" and ".dockerignore". The ".dockerignore" tab is active, showing the following content:

```
1 node_modules
2 |
```

→ To create a Dockerfile for reactjs:

- ◆ As I have already installed *create-react-app* for a previous assignment, I can skip the installation part here.
- ◆ Now let's create a new React application using the following command:
 - `create-react-app reactjs`
 - Then cd into the reactjs directory and run to the following command to check if its working fine:
 - `cd reactjs`
 - `npm start`
- ◆ Now as our reactjs app is working fine. Let's create a *Dockerfile* for this app same as we did for nodejs:
 - Let's create a file named "*Dockerfile*" in the project directory.
 - Then write this code inside the Dockerfile:

```
Dockerfile nodejs Dockerfile reactjs X Dockerfile nginx
reactjs > Dockerfile > ...
1 # pull the official base image
2 FROM node:alpine
3 # set working direction
4 WORKDIR /app
5 # add `/app/node_modules/.bin` to $PATH
6 ENV PATH /app/node_modules/.bin:$PATH
7 # copy the packages and add app
8 COPY package*.json . ./
9 # install application dependencies
10 RUN npm install
11 # expose the port
12 EXPOSE 4000
13 # start app
14 CMD ["node", "start"]
```

- We can also create `.dockerignore` if we want to which will ignore the files/ directory we don't want to push to docker hub. This file works like `.gitignore`.

```
Dockerfile .dockerignore X
reactjs > .dockerignore
1 node_modules
2 npm-debug.log
3
```

→ To create a Dockerfile for Nginx:

- ◆ For this let's create a directory with the name "Nginx" and inside the directory, let's create a simple HTML file named "index.html". This is our home page when we access the Nginx webserver. Let's write this simple code inside the index.html file.

```
index.html X
nginx > index.html > html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Nginx landing page!!</title>
5   </head>
6   <body>
7     <h1>Welcome to Nginx Web Server</h1>
8     <p>This server is running through Docker container</p>
9   </body>
10 </html>
```


◆ Now let's create a Dockerfile for Nginx like we have done before:

```
Dockerfile nodejs Dockerfile reactjs Dockerfile nginx x
nginx > Dockerfile > ...
1 # pull the official base image
2 FROM nginx:latest
3 # copy the packages and add app
4 COPY ./index.html /usr/share/nginx/html/
5 # expose the port
6 EXPOSE 5000
```

→ After we are done creating Dockerfiles for all the services. Let's now create a *docker-compose.yml* file to combine all three created Dockerfiles.

◆ At the root of this task, let's create a *docker-compose.yml* file.

```
docker-compose.yml x
docker-compose.yml
1 version: '3.8'
2
3 services:
4   nodejs:
5     # Gives location to the Dockerfile
6     build:
7       dockerfile: Dockerfile
8       context: ./nodejs
9     # Restarts the container if any errors
10    restart: always
11    # Names our container for this service
12    container_name: nodejs_app
13    # Mapping our local files/ directory from host path to container path
14    volumes:
15      - ./nodejs:/usr/src/app
16    ports:
17      - "3000:3000"
18
19   reactjs:
20     # Gives location to the Dockerfile
21     build:
22       dockerfile: Dockerfile
23       context: ./reactjs
24     # Restarts the container if any errors
25     restart: always
26     # Names our container for this service
27     container_name: reactjs_app
28     # Mapping our local files/ directory from host path to container path
29     volumes:
30       - ./reactjs:/usr/src/app
31     ports:
32       - "4000:4000"
33
34   nginx:
35     # Gives location to the Dockerfile
36     build:
37       dockerfile: Dockerfile
38       context: ./nginx
39     # Restarts the container if any errors
40     restart: always
41     # Names our container for this service
42     container_name: nginx_app
43     # Mapping our local files/ directory from host path to container path
44     volumes:
45       - ./nginx/index.html:/usr/share/nginx/html/index.html
46     ports:
47       - "5000:5000"
48
49 # To map the ports of all servies to use the same network
50 networks:
51   default:
52     external:
53       name: same
```

→ After creating the docker compose file, now let's create a network so that all the services will be mapped to the same port using the following command:

◆ *sudo docker network create same*

→ Now let's build the docker image by running this command from the root folder:

◆ *sudo docker-compose up -d*

```
rireshkarma@pop-os: ~/Desktop/Qno3
rireshkarma@pop-os:~/Desktop/Qno3$ ls
docker-compose.yml  nginx  nodejs  reactjs
rireshkarma@pop-os:~/Desktop/Qno3$ sudo docker network create same
b5059868c8c9d38b727a44c8985f6b5ff116c7489ca643747230707bc5d727d0
rireshkarma@pop-os:~/Desktop/Qno3$ sudo docker-compose up --build -d
Building nodejs
Sending build context to Docker daemon 20.48kB
Step 1/6 : FROM node:alpine
alpine: Pulling from library/node
97518928ae5f: Pull complete
7001f79e6409: Pull complete
ad6534883285: Pull complete
55d5035a0d7c: Pull complete
Digest: sha256:993bdfb0da7ae8fa4dad7282f797e3e26e88f810d410e0b0409d132d1fb17af3
Status: Downloaded newer image for node:alpine
--> 9e17f47b0a78
Step 2/6 : WORKDIR /home/node/app
--> Running in 56d25d819939
Removing intermediate container 56d25d819939
--> de085cb3cbe8
Step 3/6 : COPY package*.json index.js ./
--> f9b74af082f3
Step 4/6 : RUN npm install
--> Running in 234f74e94957
```

→ Now after the completion of the process. We can check to see if the containers are running:

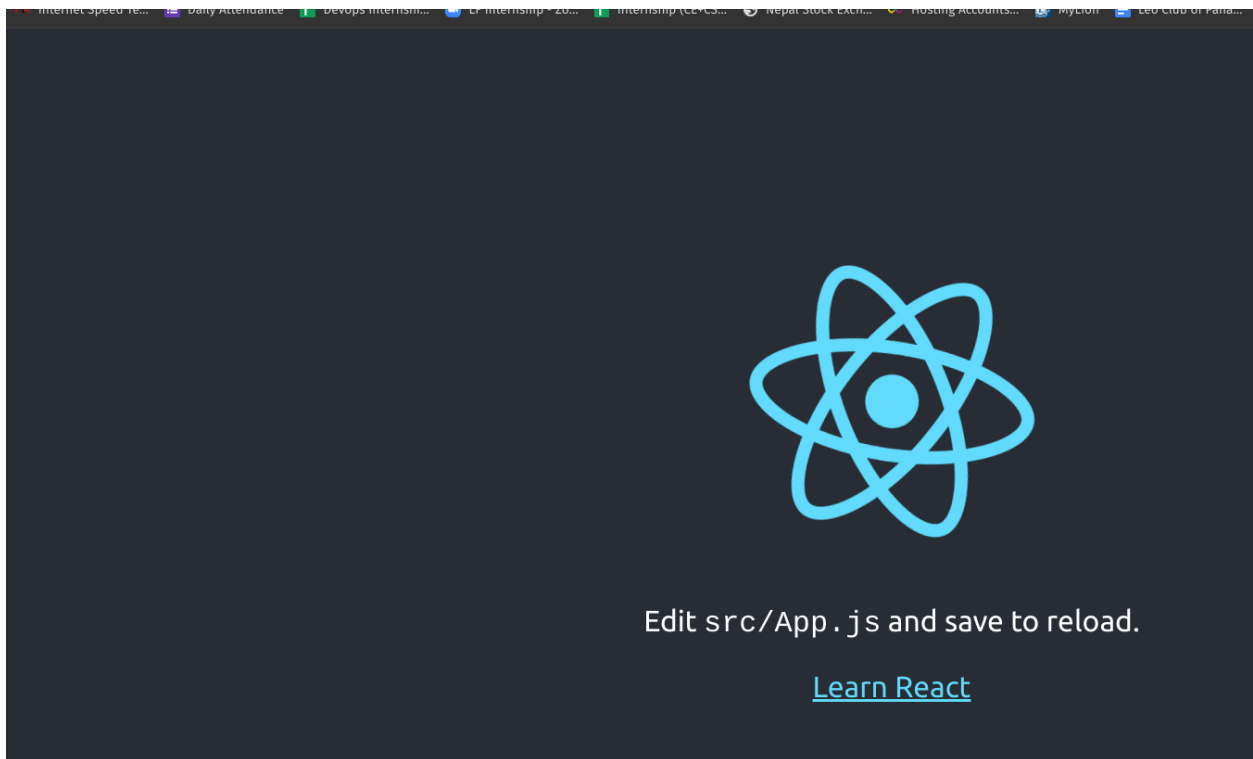
```
rireshkarma@pop-os: ~/Desktop/Qno3
Successfully tagged qno3_reactjs:latest
Building nginx
Sending build context to Docker daemon 3.072kB
Step 1/3 : FROM nginx:latest
--> ea335ee17ab
Step 2/3 : COPY ./index.html /usr/share/nginx/html/
--> 30f94d6d4e36
Step 3/3 : EXPOSE 5000
--> Running in 5384c4ede41f
Removing intermediate container 5384c4ede41f
--> 8dc1155cec3b
Successfully built 8dc1155cec3b
Successfully tagged qno3_nginx:latest
Creating nodejs_app ... done
Creating reactjs_app ... done
Creating nginx_app ... done
rireshkarma@pop-os:~/Desktop/Qno3$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS      PORTS                               NAMES
63a8ff93bee3   qno3_nginx  "/docker-entrypoint.s..." 13 minutes ago Up 13 minutes 80/tcp, 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp  nginx_app
ba9fa38e5a4b   qno3_reactjs  "docker-entrypoint.s..." 13 minutes ago Restarting (1) 55 seconds ago  nodejs_app
6761fd89a9eb   qno3_nodejs  "docker-entrypoint.s..." 13 minutes ago Up 13 minutes 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp  nodejs_app
1420af7ab04f   mysql:8      "docker-entrypoint.s..." 21 hours ago Up 2 hours 0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp  mysql8_container_db_1
```

→ As we can observe all three services are running, we can browse them on the ports to see if all three services are running.

◆ NodeJS



◆ ReactJS



◆ Nginx



4. Now push the created docker images to the docker hub repository.

→ To push the created docker images to the docker hub repository:

◆ First login to the docker hub in CLI using the following command:

- `sudo docker login`
- Fill in your credentials and log in.


```
rikeshkarma@pop-os:~$ sudo docker login
[sudo] password for rikeshkarma:
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker
ID, head over to https://hub.docker.com to create one.
Username: rikeshkarma
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
rikeshkarma@pop-os:~$
```

- ◆ Now first let's push our node js image to the docker hub repository by using the following command:
 - Before we push we need to tag the locally created image to the docker hub, which means we need to tag the image with the docker hub username using the following command:
 - `docker tag qno3_nodejs rikeshkarma/qno3_nodejs`

```
rikeshkarma@pop-os: ~  
rikeshkarma@pop-os:~$ docker tag qno3_nodejs rikeshkarma/qno3_nodejs  
rikeshkarma@pop-os:~$ sudo docker push rikeshkarma/qno3_nodejs  
[sudo] password for rikeshkarma:  
Using default tag: latest  
The push refers to repository [docker.io/rikeshkarma/qno3_nodejs]  
88065d552f36: Pushed  
1d5dac004c0a: Pushed  
c641545c01a0: Pushed  
291200a56550: Mounted from library/node  
17c7cb9c9d9c: Mounted from library/node  
13f414425f62: Mounted from library/node  
1a058d5342cc: Mounted from library/node  
latest: digest: sha256:01ada9543958b416bbc3397e5e61936baa1156c36b4c066be0fedfb3fbcdd194 size: 1784  
rikeshkarma@pop-os:~$
```

- `sudo docker push rikeshkarma/qno3_nodejs`


 rikeshkarma / qno3_nodejs

This repository does not have a description

Last pushed: a minute ago

Tags and Scans

This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
latest		a minute ago	a minute ago

[See all](#)

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions.

Upgrade to Pro

[Learn more](#)

Docker commands


To push a new tag to this repository,

`docker push rikeshkarma/qno3_nodejs:tagname`

[Public View](#)

- ◆ Now first let's push our react js image to the dockerdocker hub repository by using the following command:
 - Before we push we need to tag the locally created image to the docker hub, which means we need to tag the image with the docker hub username using the following command:
 - `docker tag qno3_reactjs rikeshkarma/qno3_reactjs`
 - `sudo docker push rikeshkarma/qno3_reactjs`

```
rireshkarma@pop-os: ~$ docker tag qno3_reactjs rakeshkarma/qno3_reactjs
rireshkarma@pop-os: ~$ sudo docker push rakeshkarma/qno3_reactjs
Using default tag: latest
The push refers to repository [docker.io/rakeshkarma/qno3_reactjs]
ecf674a8fa07: Pushed
a00c736320ef: Pushed
d5da76cbff0c: Pushed
291200a56550: Mounted from rakeshkarma/qno3_nodejs
17c7cb9c9d9c: Mounted from rakeshkarma/qno3_nodejs
13f414425f62: Mounted from rakeshkarma/qno3_nodejs
1a058d5342cc: Mounted from rakeshkarma/qno3_nodejs
latest: digest: sha256:7eb814ccaf3b67955975c21d2b06e7d83ab2995e31417ef0c71ccf85cd974ef4 size: 1787
rireshkarma@pop-os: ~$
```

 rakeshkarma / qno3_reactjs

This repository does not have a description

Last pushed: a minute ago


Docker commands

To push a new tag to this repository,

`docker push rakeshkarma/qno3_reactjs:tagname`

Tags and Scans

This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
latest		a minute ago	a minute ago

See all

VULNERABILITY SCANNING - DISABLED

Enable

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions.

Upgrade to Pro

Learn more

Public View

- ◆ Now first let's push our node js image to docker hub repository by using the following command:
 - Before we push we need to tag the locally created image to the docket hub, which means we need to tag the image with the docker hub username using the following command:
 - `docker tag qno3_nginx rakeshkarma/qno3_nginx`

```
rireshkarma@pop-os: ~$ docker tag qno3_nginx rakeshkarma/qno3_nginx
rireshkarma@pop-os: ~$ sudo docker push rakeshkarma/qno3_nginx
Using default tag: latest
The push refers to repository [docker.io/rakeshkarma/qno3_nginx]
a64486a117ab: Pushed
8525cde30b22: Mounted from library/nginx
1e8ad06c81b6: Mounted from library/nginx
49eeddd2150f: Mounted from library/nginx
ff4c72779430: Mounted from library/nginx
37380c5830fe: Mounted from library/nginx
e1bbcf243d0e: Mounted from library/nginx
latest: digest: sha256:7dd41d9f0a6143dbd32fa391c7a256cd2ee636b27ce422d5979b40c785e4a867 size: 1777
rireshkarma@pop-os: ~$
```

- `sudo docker push rikeshkarma/qno3_nginx`

The screenshot shows the Docker Hub interface for the repository `rikeshkarma/qno3_nginx`. The repository is public and has no description. It was last pushed a few seconds ago. The Docker commands section shows the command `docker push rikeshkarma/qno3_nginx:tagname`. The Tags and Scans section shows a table with one tag, `latest`, which was pushed a few seconds ago. The Automated Builds section shows that automated builds are disabled and provides instructions on how to enable them by connecting to GitHub or Bitbucket.

TAG	OS	PULLED	PUSHED
latest	linux	a few seconds ago	a few second...

5. Install docker swarm and manage the cluster of all created containers.

- ➔ Docker Swarm is a clustering tool that turns a group of Docker hosts into a single virtual server. Docker Swarm ensures availability and high performance for your application by distributing it over the number of Docker hosts inside a cluster. Docker Swarm also allows you to increase the number of container instances for the same application.
- ➔ The Docker swarm has already been installed automatically while we install the docker engine, so we don't need to do it manually. Now we just need to initialize it, for initializing we can invoke the following command:
 - ◆ `sudo docker swarm init --advertise-addr <host-ip>`
 - ◆ After using this command the docker swarm should be initialized.
 - ◆ Let's verify that in case using the following command:
 - `sudo docker system info | grep Swarm`

```
rikeshkarma@pop-os: ~$ sudo docker swarm init --advertise-addr 192.168.254.141
Swarm initialized: current node (pl9bofs7jwozgj09ksjrsa7hs) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-60rzrw7udns95Lrhneb1rd0vygq34quak0jrrqp7ssc7jrms8-7gsq9bi0n81zjbec7efgl0mb6
    192.168.254.141:2377

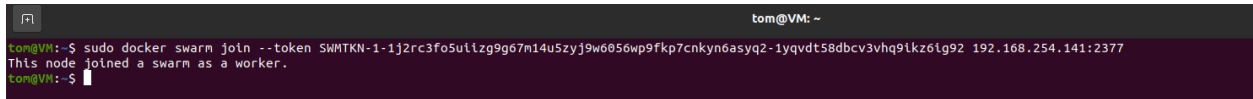
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

rikeshkarma@pop-os: ~$ sudo docker system info | grep swarm
rikeshkarma@pop-os: ~$ sudo docker system info | grep Swarm
Swarm: active
rikeshkarma@pop-os: ~$
```

→ Firstly we need to join the worker's node to manage the clusters of all created containers, so let's do it::

- ◆ Initially, we need to install Docker Engine in the worker node. toun the command that we got when we used swarm init in the VM so as to add a worker in the swarm. The command generated was:

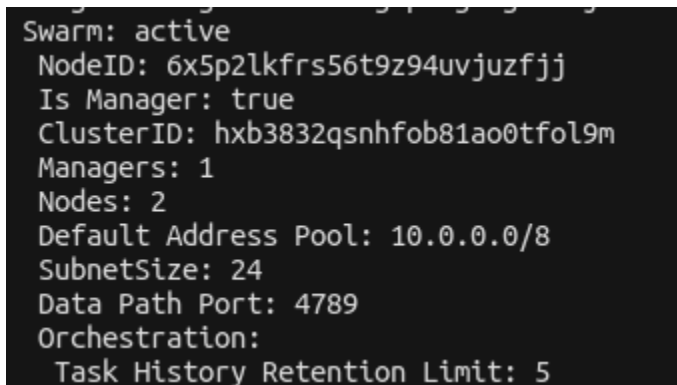
- `docker swarm join --token SWMTKN-1-1j2rc3fo5uizg9g67m14u5zyj9w6056wp9fkp7cnkyn6asyq2-1yqvdt58dbcv3vhq9ikz6ig92 192.168.254.141:2377`



```
tom@VM: ~  
tom@VM:~$ sudo docker swarm join --token SWMTKN-1-1j2rc3fo5uizg9g67m14u5zyj9w6056wp9fkp7cnkyn6asyq2-1yqvdt58dbcv3vhq9ikz6ig92 192.168.254.141:2377  
This node joined a swarm as a worker.  
tom@VM:~$
```

- ◆ Now we can see that the node has been added to swarn as a worker if we view the Docker info in manager mode using the following command:

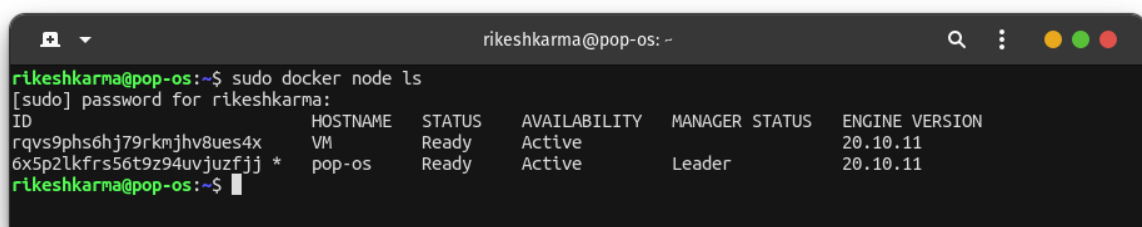
- `sudo docker info`



```
Swarm: active  
NodeID: 6x5p2lkfrs56t9z94uvjuzfjj  
Is Manager: true  
ClusterID: hxb3832qsnhfob81ao0tfol9m  
Managers: 1  
Nodes: 2  
Default Address Pool: 10.0.0.0/8  
SubnetSize: 24  
Data Path Port: 4789  
Orchestration:  
Task History Retention Limit: 5
```

- We can see that now there are **2 nodes**.
- ◆ We can check the information about the nodes using the following command:

- `sudo docker node ls`



```
rikeshkarma@pop-os:~$ sudo docker node ls  
[sudo] password for rikeshkarma:  
ID                HOSTNAME        STATUS      AVAILABILITY    MANAGER STATUS  ENGINE VERSION  
rqvs9phs6hj79rknhv8ues4x  VM             Ready       Active           Leader           20.10.11  
6x5p2lkfrs56t9z94uvjuzfjj *  pop-os         Ready       Active           Leader           20.10.11  
rikeshkarma@pop-os:~$
```

- We can see that there is a VM as a node as well. We can also see that our host machine is leader and the VM is the worker (empty MANAGER STATUS) also both are active.

→ Swarm services use a declarative model, which means that you define the desired state of the service, and rely upon Docker to maintain this state.

→ Now let's create a swarm service using the following command:

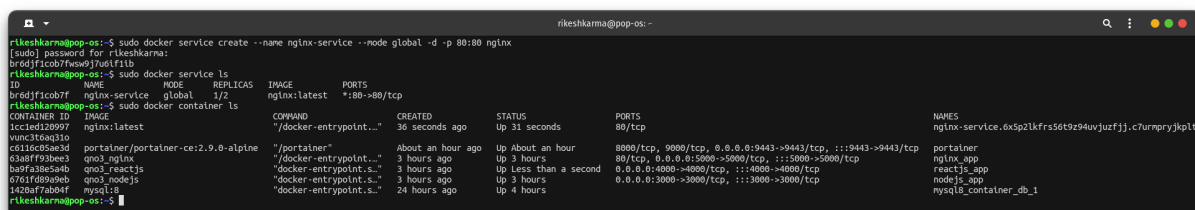
◆ `sudo docker service create --name Nginx-service --mode global -d -p 80:80 Nginx`

- **--mode:-** This option will help to create the container in our desired mode. As our model for this command is global, which means two containers will be created on the 2 nodes we have i.e manager node and the worker node.

→ After creating the service we can check the services and list the containers running by invoking the following commands:

◆ `sudo docker service ls`

◆ `sudo docker container ls`



```
rakeshkarma@pop-os:~$ sudo docker service create --name nginx-service --mode global -d -p 80:80 nginx
[sudo] password for rakeshkarma:
rakeshkarma@pop-os:~$ sudo docker service ls
ID                NAME              MODE              REPLICAS            IMAGE              PORTS
brdjdjicob7f      nginx-service     global            1/2                  nginx:latest       *:80->80/tcp
rakeshkarma@pop-os:~$ sudo docker container ls
CONTAINER ID   IMAGE              COMMAND                  CREATED            STATUS              PORTS              NAMES
1c4e4e120997   nginx:latest       "/docker-entrypoint..." 36 seconds ago    Up 31 seconds      80/tcp             nginx-service_6x5p2lkfrs56t9z94uvjuzfjj_c7urnpryjkpl
c6116c05ae3d   portainer/portainer-ce:2.9.0-alpine   "/portainer"           About an hour ago  Up About an hour   8080/tcp, 9080/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp   portainer
0a0ff93beec3   qno3/nginx         "/docker-entrypoint..." 3 hours ago       Up 3 hours         80/tcp, 0.0.0.0:15000->5000/tcp, :::5000->5000/tcp              nginx_app
ba9fa38e5a4b   qno3/reactjs       "/docker-entrypoint.s..." 3 hours ago       Up Less than a second  0.0.0.0:4000->4000/tcp, :::4000->4000/tcp                      reactjs_app
0761f089a9eb   qno3/nodejs        "/docker-entrypoint.s..." 3 hours ago       Up 3 hours         0.0.0.0:3000->3000/tcp, :::3000->3000/tcp                      nodejs_app
142ba7fab04f   mysql:8.0          "mysql"                  24 hours ago     Up 4 hours         mysql               mysql0_container_db_1
rakeshkarma@pop-os:~$
```

→ Now we can deploy the docker-compose.yml to the worker node with custom build images we need to invoke the following commands:

◆ But before that, we need to make some changes to the docker-compose.yml file. The snapshot of yml file after making the modifications is below and the file is also available in the task repo:

◆ Added modifications in the docker-compose.yml files are:

- **image:-** links to the previously pushed docker image. (dockerhubID/image-name)
- **networks:-** changes have been name on the restart policy, stacking the network and replicas have been added to all three services. Also, mapping ports of all services drivers has been added as an overlay.


```
docker-compose.yml x
docker-compose.yml
1  version: '3.8'
2
3  services:
4    nodejs:
5      # Gives link to the Docker image
6      image: rikeshkarma/qno3_nodejs
7      ports:
8        - "3000:3000"
9      networks:
10       - stack
11      deploy:
12        replicas: 1
13        restart_policy:
14          condition: on-failure
15
16      reactjs:
17        # Gives link to the Docker image
18        image: rikeshkarma/qno3_reactjs
19        ports:
20          - "4000:4000"
21        networks:
22          - stack
23        deploy:
24          replicas: 1
25          restart_policy:
26            condition: on-failure
27
28      nginx:
29        # Gives link to the Docker image
30        image: rikeshkarma/qno3_nginx
31        networks:
32          - stack
33        deploy:
34          replicas: 1
35          restart_policy:
36            condition: on-failure
37        ports:
38          - "5000:5000"
39
40      # To map the ports of all servies to use the same network
41      networks:
42        stack:
43          driver : overlay
```

→ We assign the service to run containers in the choice of our node by changing this on the docker-compose.yml file: (following to run on the manager mode)

◆ *deploy:*

placement:

constraints:

- *node.role == manager*

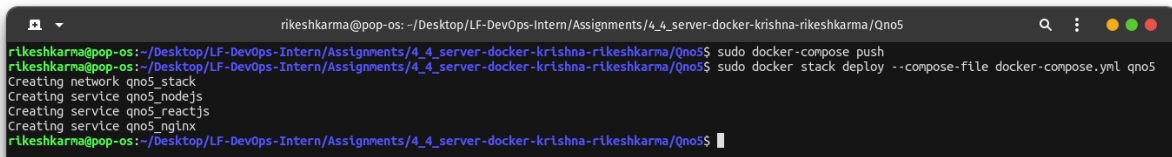
→ Now let's naming deploy to the worker node i.e in the VM.

◆ First, we need to push the compose file to the Docker hub.

- For this let's navigate to the directory where the docker-compose.yml file is located.
- Then run the following command in the terminal:
 - *sudo docker-compose push*

◆ Now to deploy stack with composer file use the following command:

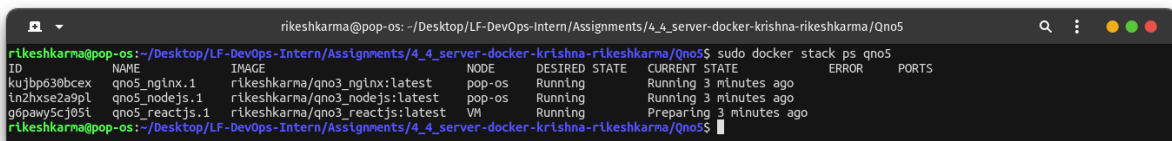
- *sudo docker stack deploy --compose-file docker-compose.yml qno5*



```
rikeshkarma@pop-os: ~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-rikeshkarma/Qno5
rikeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-rikeshkarma/Qno5$ sudo docker-compose push
rikeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-rikeshkarma/Qno5$ sudo docker stack deploy --compose-file docker-compose.yml qno5
Creating network qno5_stack
Creating service qno5_nodejs
Creating service qno5_reactjs
Creating service qno5_nginx
rikeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-rikeshkarma/Qno5$
```

◆ We can now check the details of the stack we just deployed using the following command:

- *sudo docker stack ps qno5*



```
rikeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-rikeshkarma/Qno5$ sudo docker stack ps qno5
ID            NAME                IMAGE                                NODE    DESIRED STATE    CURRENT STATE    ERROR    PORTS
kujbp638bcex  qno5_nginx.1       rikeshkarma/qno3_nginx:latest      pop-os  Running          Running 3 minutes ago
ln2hxse2a9pl  qno5_nodejs.1      rikeshkarma/qno3_nodejs:latest     pop-os  Running          Running 3 minutes ago
g0pawy5cj0sl  qno5_reactjs.1     rikeshkarma/qno3_reactjs:latest    VM      Running          Preparing 3 minutes ago
rikeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-rikeshkarma/Qno5$
```

- We can observe that the qno5_reactjs container has been deployed to the worker node i.e in the VM.

◆ We can now check on the VM as well for the containers, it must show the reactjs container created.

- We can use the following command to verify:
 - *sudo docker container ls*

◆ If we need we can also scale containers using the service from the manager node to the worker node. For this we need to invoke this command:

- *sudo docker service scale <network-name>=num.*
- In our case we will use this command:

- `sudo docker service scale qno5_nginx=2`

```

riqeshkarma@pop-os: ~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-riqeshkarma/Qno5
riqeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-riqeshkarma/Qno5$ sudo docker service scale qno5_nginx=2
qno5_nginx scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service converged
riqeshkarma@pop-os:~/Desktop/LF-DevOps-Intern/Assignments/4_4_server-docker-krishna-riqeshkarma/Qno5$

```

- ◆ Let's check on the VM i.e worker-node for the running containers again using this command:

- `sudo docker container ls`

```

kangVM:~$ sudo docker container ls
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
6d51be40ccf0   riqeshkarma/qno5_nginx:latest       "/docker-entrypoint..." About a minute ago Up 57 seconds 80/tcp, 5000/tcp                   qno5_nginx-2.6s9hbyv8Buacg02f9e7soc17v
1e11cb73914c   nginx:latest                         "/docker-entrypoint..." 9 minutes ago  Up 8 minutes  80/tcp                             nginx-service.rqvs9phs6hj79rknhv8ues4x.cc6599p1enc303z0b73vs04kr

```

- Here the container with my react js is not showing because just before this my VM crashed while taking the screenshot for the previous ls command to show reactjs running on the worker node and I tried to review the previous processes but due to some issue it's not showing, my apologies.

6. Install the portainer and manage the docker from GUI. Practice the features and attach the snapshots.

→ Portainer is a Web User Interface (WUI/GUI) dashboard tool, which can be used to monitor and centrally manage a docker platform. With Portainer, We can add/login to a registry, Manage multiple hosts, Manage hosts on different networks, Build a Container (Docker) Image, pull and push an image from/to container registry, Manage running/deployed containers, deploy a stack/container, Manage accounts, and many more. Other than the standalone docker platform, Portainer also supports Docker Swarm and Kubernetes cluster. Above that, All of them can be managed from a single centrally managed dashboard UI.

→ Now let's start by installing portainer first:

- ◆ The installation requires root access for the proper functioning of the dashboard so let's invoke this command first:
 - `sudo su`
- ◆ Let's create a docker volume for portainer data by using this command:
 - `docker volume create portainer_data`
- ◆ Now let's deploy portainer server on our local machine using this command:

- `docker run --detach --name portainer --publish 9443:9443 --volume /var/run/docker.sock:/var/run/docker.sock --volume portainer_data:/data --restart=always portainer/portainer-ce:2.9.0-alpine`

```

rtkeshkarma@pop-os:~$ sudo su
[sudo] password for rtkeshkarma:
root@pop-os:~# docker volume create portainer_data
portainer_data
root@pop-os:~# docker run --detach --name portainer --publish 9443:9443 --volume /var/run/docker.sock:/var/run/docker.sock --volume
portainer_data:/data --restart=always portainer/portainer-ce:2.9.0-alpine
Unable to find image 'portainer/portainer-ce:2.9.0-alpine' locally
2.9.0-alpine: Pulling from portainer/portainer-ce
a0d0a0d46f8b: Pull complete
86253b00ae0d: Pull complete
Digest: sha256:ae5fd2d4992bc9d56932b7c54ee4fbcfdd56762d2af2b936e1f81795976ddf
Status: Downloaded newer image for portainer/portainer-ce:2.9.0-alpine
c6116c05ae3df2bdc5afb37af4c5cf3b0ba0fa2273135a94660ac81e092fb77a
root@pop-os:~#

```

- ◆ Now Port 9443 can be used for web UI access. Docker socket is required to be mounted on Portainer container to allow it to access docker daemon.
- ◆ We can check if portainer is up and running or not by invoking the following command:
 - `docker ps`

```

rtkeshkarma@pop-os:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
c6116c05ae3d   portainer/portainer-ce:2.9.0-alpine "/portainer"            2 minutes ago Up 2 minutes 8080/tcp, 9080/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp portainer
63a8ff93bee3   qno3/nginx                        "docker-entrypoint.s..." 2 hours ago   Up 2 hours    80/tcp, 0.0.0.0:5080->5080/tcp, :::5080->5080/tcp nginx_app
ba9fa38e5a4b   qno3/reactjs                       "docker-entrypoint.s..." 2 hours ago   Up 2 hours    0.0.0.0:3000->3000/tcp, :::3000->3000/tcp reactjs_app
6761fd89a9eb   qno3/nodejs                        "docker-entrypoint.s..." 2 hours ago   Up 2 hours    0.0.0.0:3306->3306/tcp, :::3306->3306/tcp nodejs_app
1420af7ab04f   mysql:8                           "docker-entrypoint.s..." 22 hours ago   Up 3 hours    0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp mysql_container_db_1

```

- We can observe a lot that portainer server is up and running with CONTAINER ID c6116c05ae3d.
- ◆ Now let's check on our browser at <https://localhost:9443>.

Portainer

https://localhost:9443/#/init/admin

portainer.io

New Portainer installation

Please create the initial administrator user.

Username: admin

Password:

Confirm password:

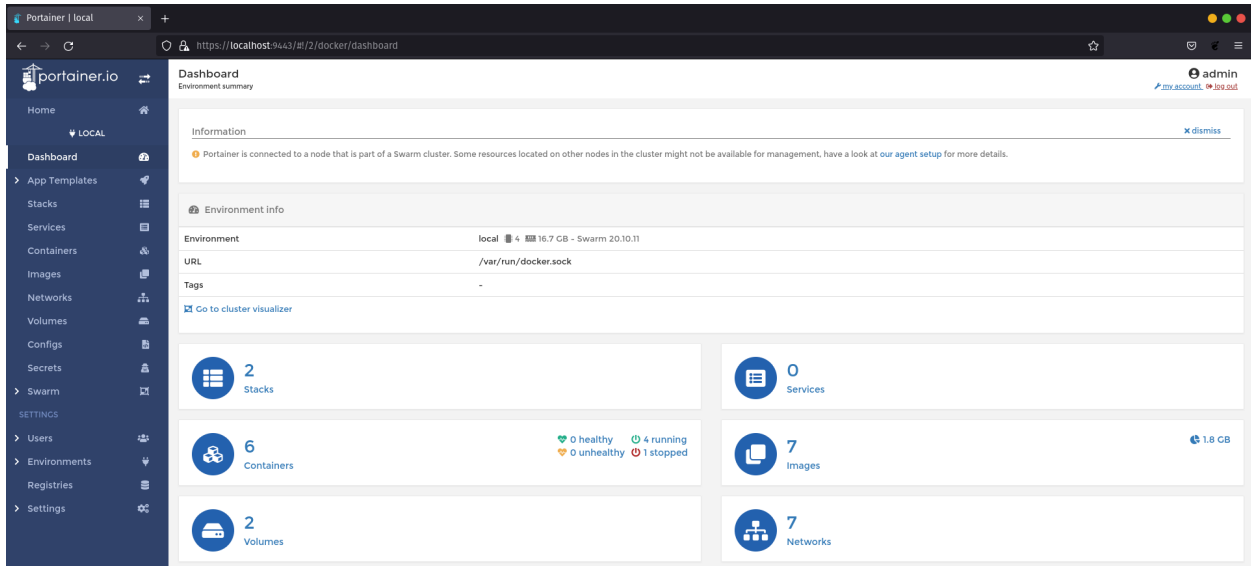
✓ The password must be at least 8 characters long

Create user

✓ Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#).

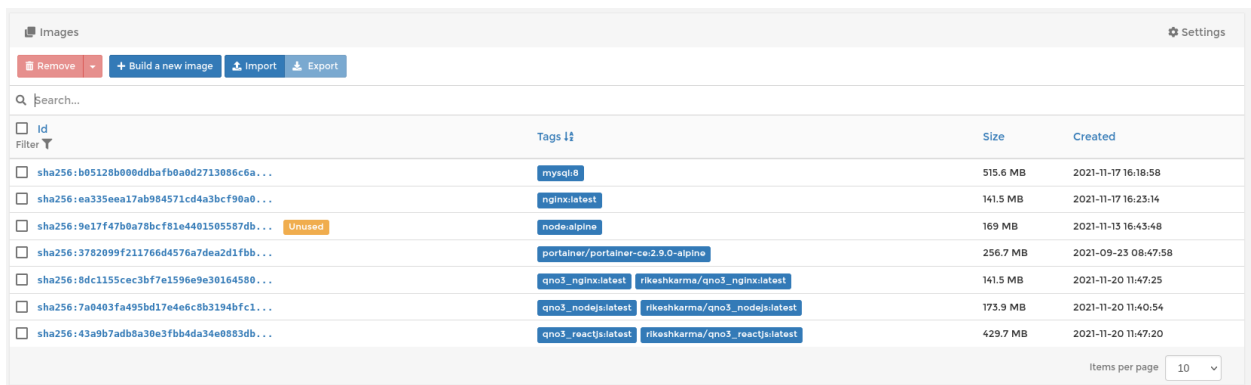
Restore Portainer from backup

- ◆ Let's setup the admin credentials first. Then we can see our dashboard with all our local services, containers, images, volumes, networks and others.

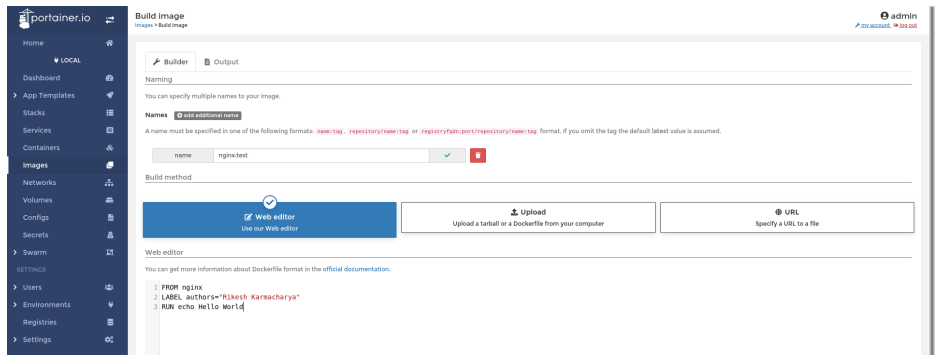


- ◆ We can find a lot of features here. Let's practice some of the features:

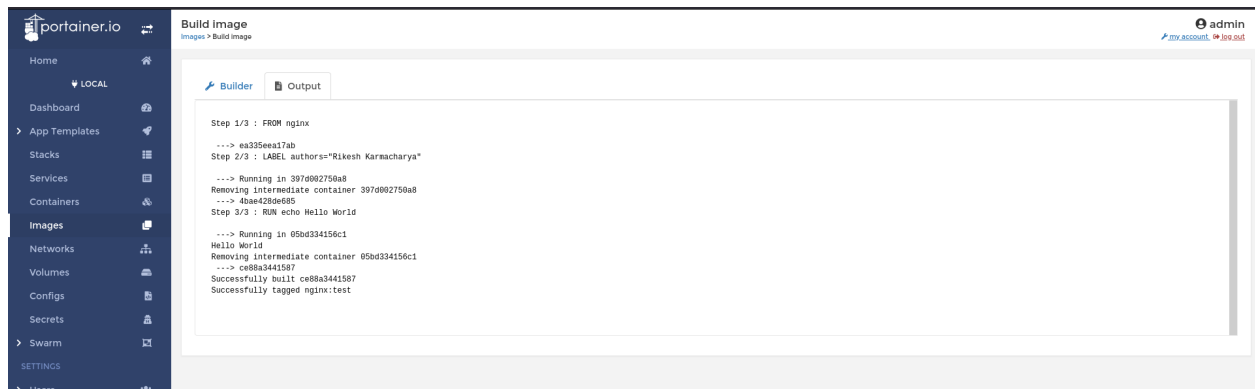
- Build a container/ docker image
 - Image -> Build a new image



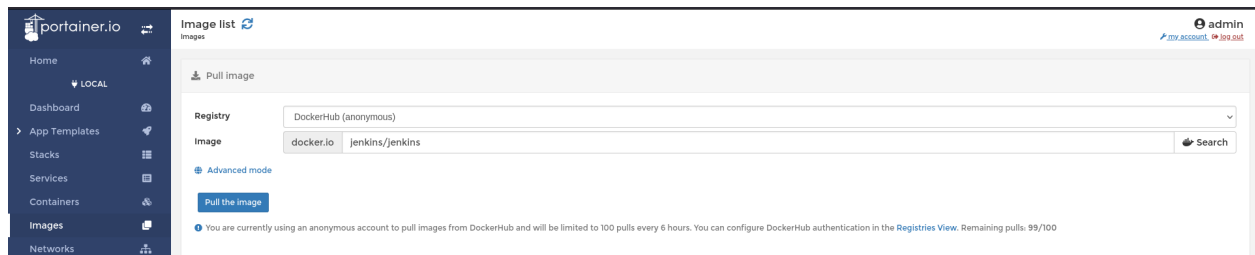
- In image section, we can see all the list of the pulled images in the environment. We can also see the **Build a new image** button. We can click on it and we can see that we can write a Dockerfile in the web editor or upload a tarball or use an URL to build an image. Let's build a sample Nginx:test image,



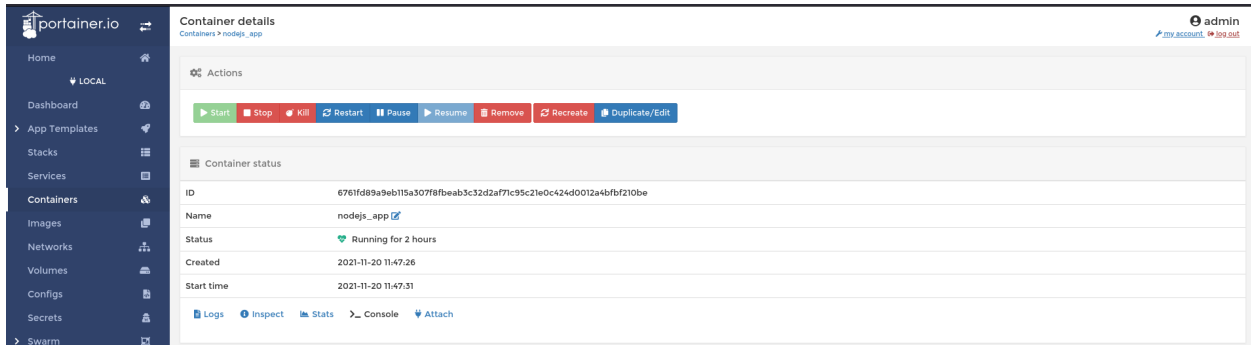
- We can see this working in the snapshot below:



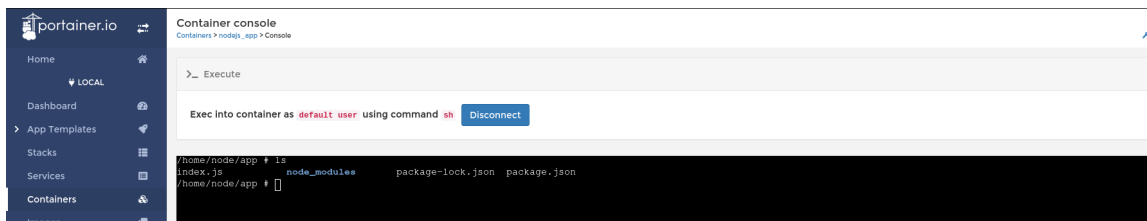
- Pull an image from a registry
 - Let's click on images from the left panel and choose registry from the drop-down list. We need to add image name with tag and then pull the image:



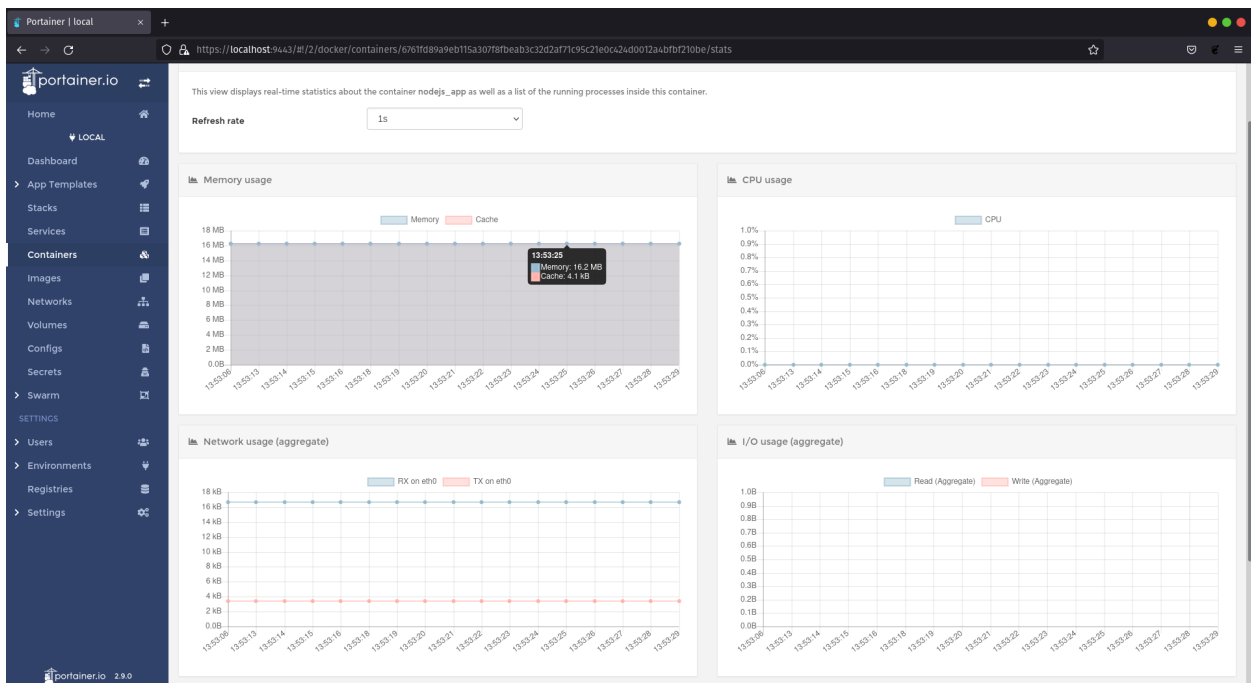
- Run commands inside the container (Exec inside the container)
 - Now let's select containers from the left panel and choose any running container. Then we can click on the console as in the snapshot below:



- After that, we can give it with a custom command or use shell binary present in the container.

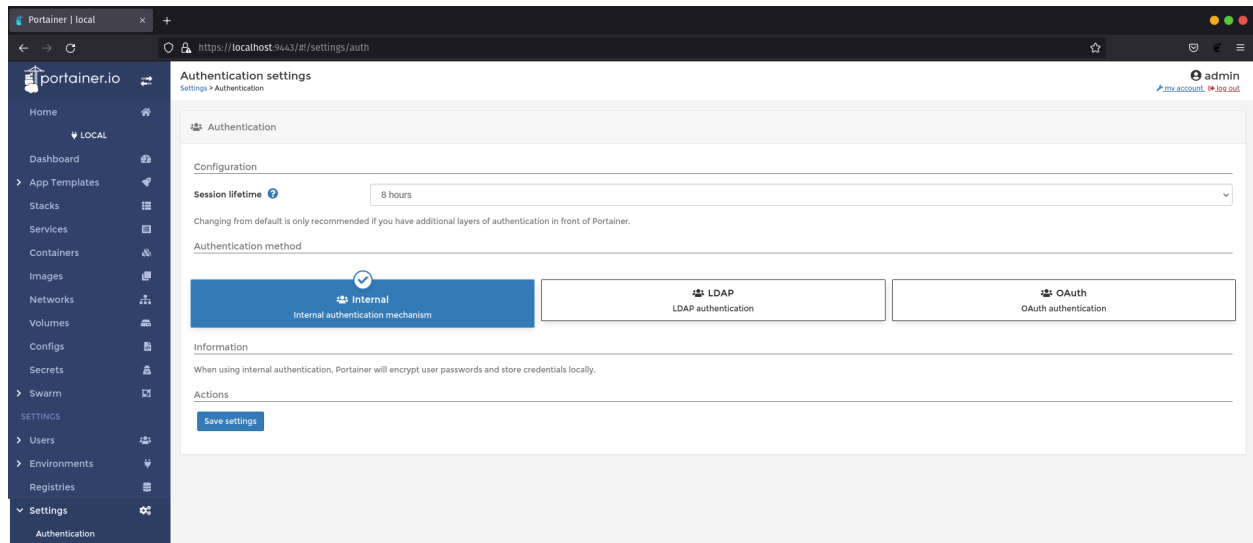


- Monitor resource usage of a container
 - Like before opening a container from a running container's list.
 - Then click on stats to monitor resource usages of the particular container.



- Manage accounts

- We can also set up custom authentication methods under settings and then click on authentication as on the snapshot below:
- We can create and manage Roles, Teams, and users from Servings and then the Users Section. This can be used to grant access to other developers and cut off their duties.



Findings:

- While creating a container from the docker-compose.yml file, I learned that Docker Compose provides us with a lot of flexibility and configurations that if we want to switch our configuration, in our case let's say database then we can do it in about a minute or even in seconds if we want, just by changing some lines in the *docker-compose.yml* file. And we just need to do is restart the Docker Compose.
- Found out that there is .dockerignore similar to .gitignore which allows us to mention a list of files and/or directories that we want to ignore while building the image. This also then helps us to reduce the size of the image and also helps to speed up the docker build process.
- While using the version of the docker-compose file, we should always update to the latest version whenever possible because it will likely contain bug fixes.
- Portainer creates its self-signed SSL certificate for secure communication between Portainer Server and User's browser and between Portainer Server and other Portainer agents. If you want to add your certificates, that can be done from

web UI or It can also be added during installation using sslcert and sslkey arguments passed as below:

- `--sslcert /path/to/certs/cername.crt --sslkey /path/to/certs/keyname.key`