11. Containerization with Docker: Java, Python, and Node.js Applications

PROBLEM STATEMENT:

As a developer at XYZ Solutions, you are tasked with containerizing applications written in Java, Python, and Node.js to streamline the development and deployment process. The company aims to enhance scalability, simplify management, and ensure consistent environments across different application stacks.

USE CASE SCENARIO:

- ➔ **Business Requirement:** XYZ Solutions is looking to modernize its application deployment process by adopting containerization for Java, Python, and Node.js applications.

- ➔ **Technical Challenge:** Develop Dockerfiles for each application to encapsulate them within containers, and create a Docker Compose file to orchestrate the deployment of these containers. The goal is to improve scalability, simplify management, and maintain consistency across diverse application stacks.

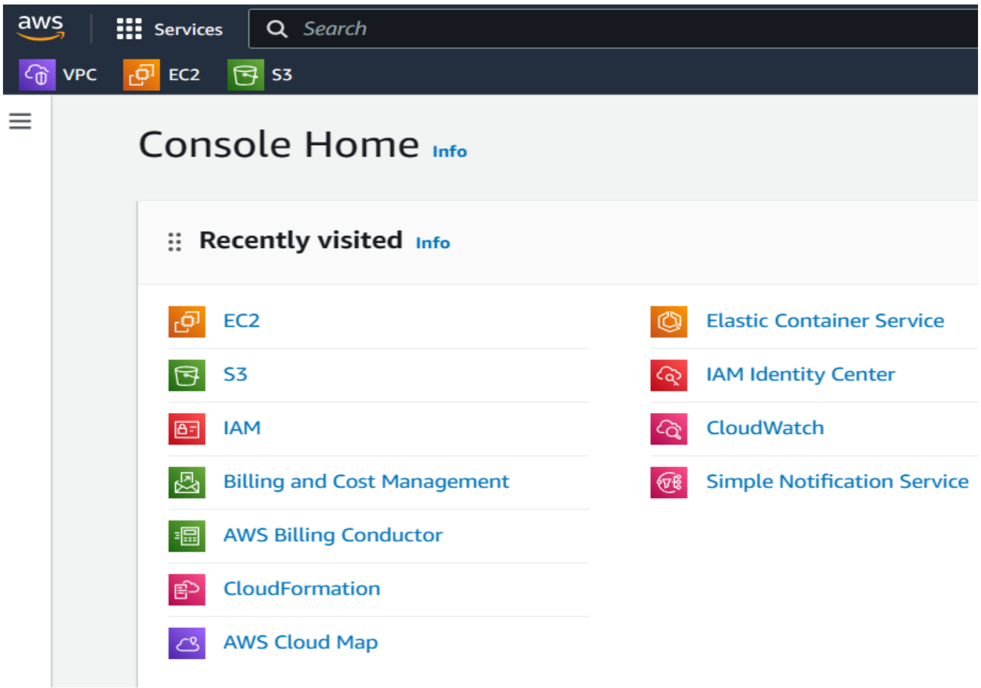
SOLUTION:

Requirements:

- ➔ AWS Cloud
- ➔ GitHub
- ➔ Docker
- ➔ Docker-compose

Step:1 – Create an EC2 instance:

- ➔ First login into your AWS instance:

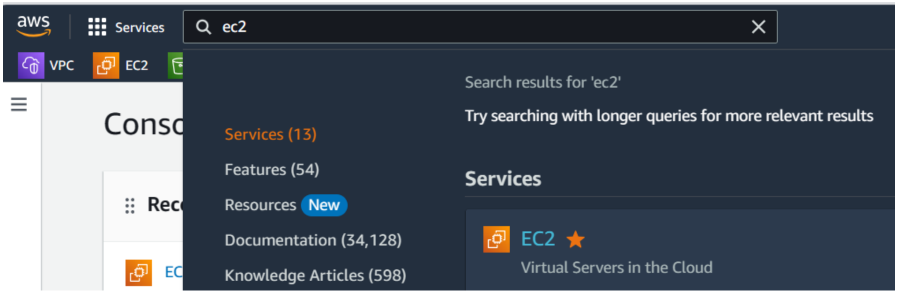


Console Home Info

Recently visited Info

EC2
S3
IAM
Billing and Cost Management
AWS Billing Conductor
CloudFormation
AWS Cloud Map
Elastic Container Service
IAM Identity Center
CloudWatch
Simple Notification Service

➔ Then on service search panel search EC2, click that one:



ec2

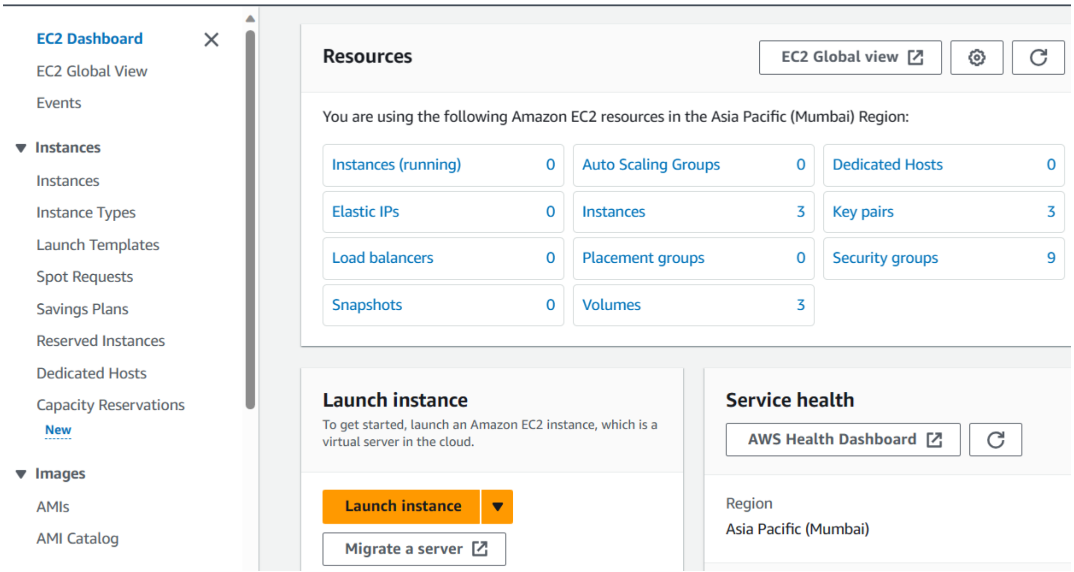
Search results for 'ec2'

Try searching with longer queries for more relevant results

Services

EC2 ★
Virtual Servers in the Cloud

➔ Then click launch instances, for creating an EC2 instance:



Resources

You are using the following Amazon EC2 resources in the Asia Pacific (Mumbai) Region:

Instances (running)	0	Auto Scaling Groups	0	Dedicated Hosts	0
Elastic IPs	0	Instances	3	Key pairs	3
Load balancers	0	Placement groups	0	Security groups	9
Snapshots	0	Volumes	3		

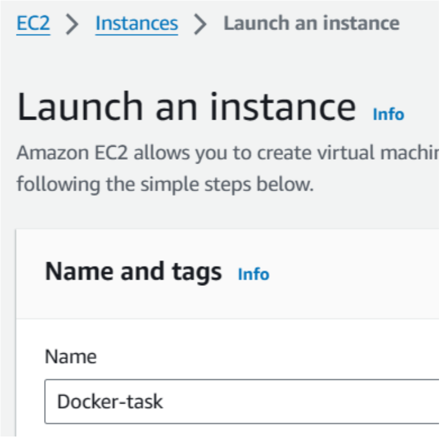
Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Service health

Region
Asia Pacific (Mumbai)

➔ Then name the instance according to your preferences:



EC2 > Instances > Launch an instance

Launch an instance Info

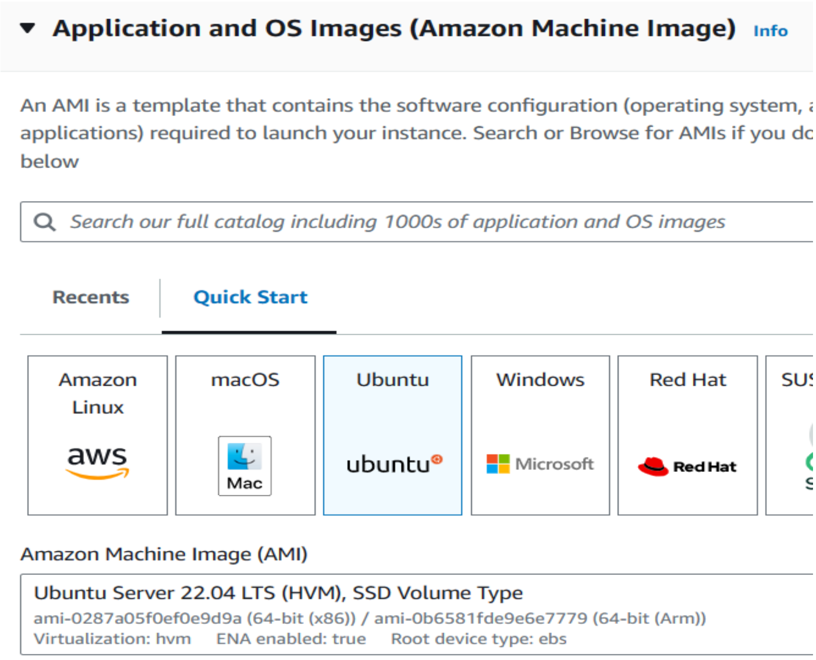
Amazon EC2 allows you to create virtual machir following the simple steps below.

Name and tags Info

Name

Docker-task

➔ Then select the operating system according to your preferences:



▼ Application and OS Images (Amazon Machine Image) Info

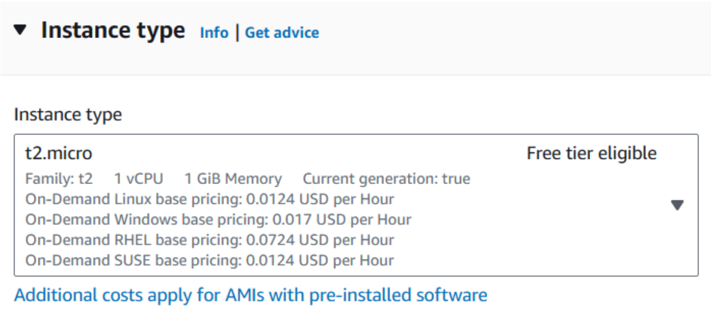
An AMI is a template that contains the software configuration (operating system, a applications) required to launch your instance. Search or Browse for AMIs if you dc below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
ami-0287a05f0ef0e9d9a (64-bit (x86)) / ami-0b6581fde9e6e7779 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

➔ Then select the instance type: according to your preferences, but here I am selecting **t2.micro**



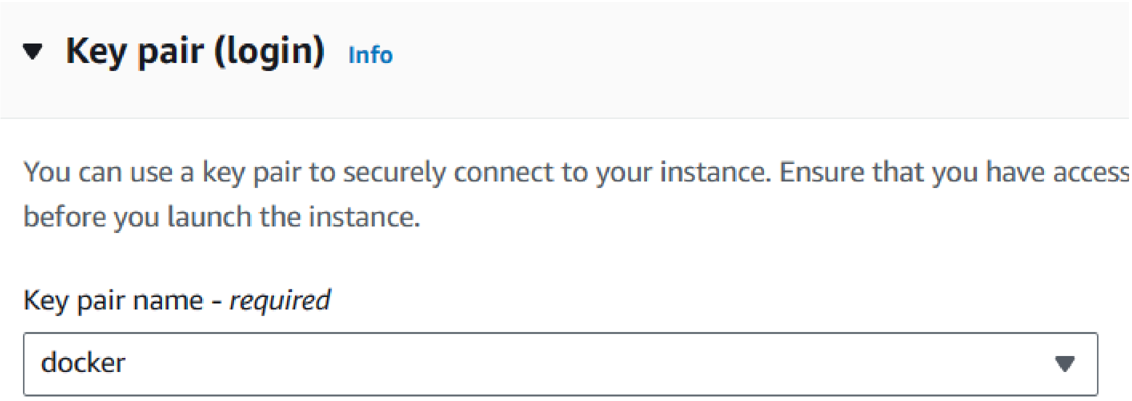
▼ Instance type Info | Get advice

Instance type

t2.micro Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand RHEL base pricing: 0.0724 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour

Additional costs apply for AMIs with pre-installed software

- ➔ Then select the key pair, according to your preferences, but here I am **proceeding with key pair option**, you can go with proceed with **without key pair option:**



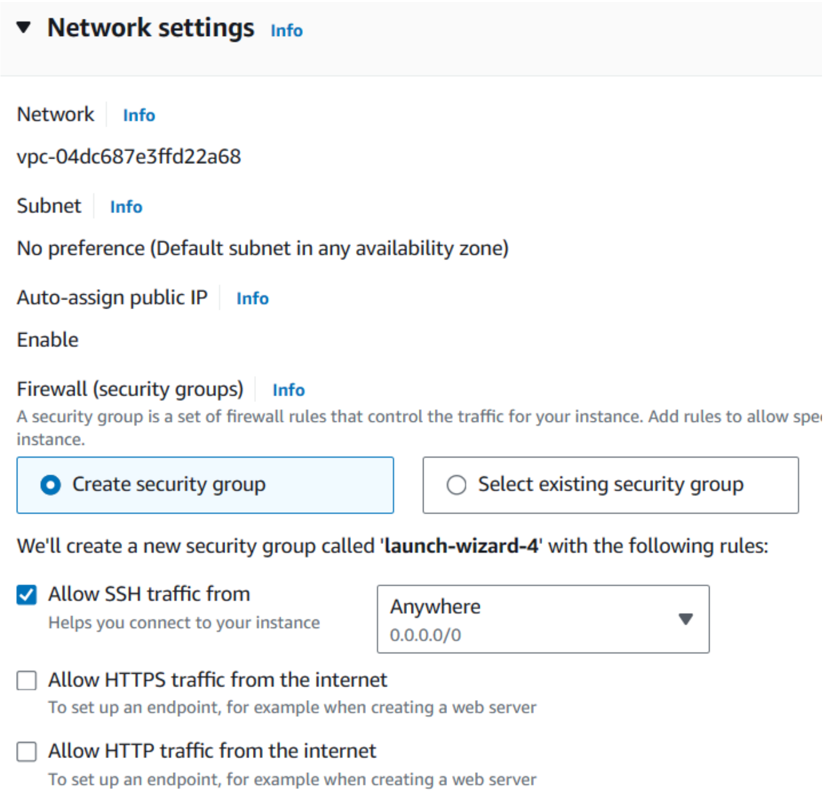
▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access before you launch the instance.

Key pair name - *required*

docker ▼

- ➔ Then keeping the default options under network settings:



▼ Network settings Info

Network | Info

vpc-04dc687e3ffd22a68

Subnet | Info

No preference (Default subnet in any availability zone)

Auto-assign public IP | Info

Enable

Firewall (security groups) | Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow spe instance.

● Create security group ○ Select existing security group

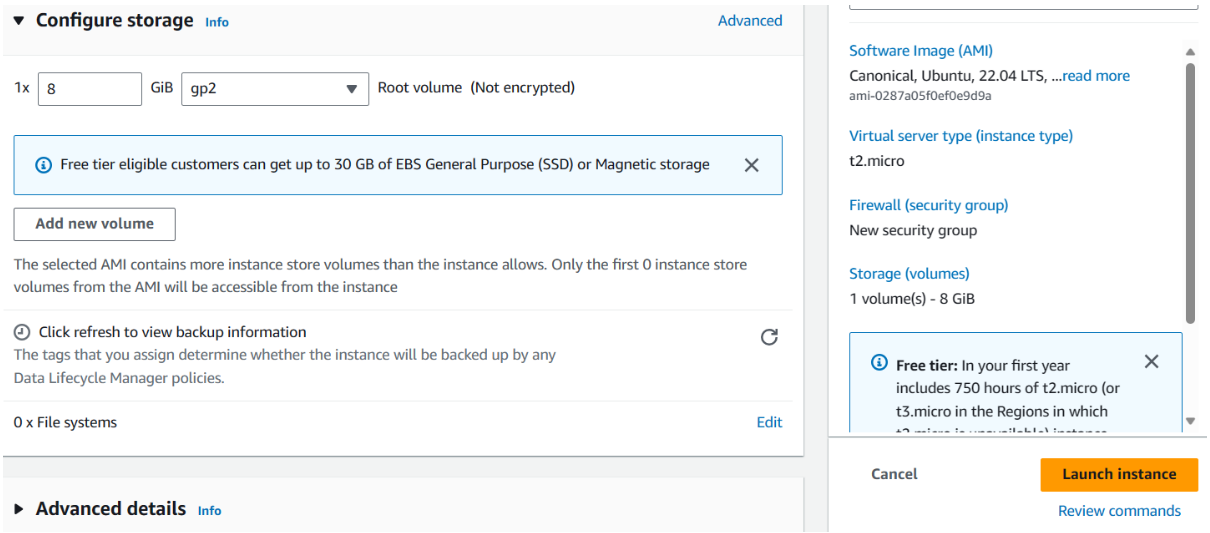
We'll create a new security group called '**launch-wizard-4**' with the following rules:

☑ Allow SSH traffic from
Helps you connect to your instance
Anywhere ▼
0.0.0.0/0

□ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

□ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

- ➔ Then keeping default options for the rest of the settings, click launch instance:



1x 8 GiB gp2 Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

Click refresh to view backup information
The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems Edit

Software Image (AMI)
Canonical, Ubuntu, 22.04 LTS, ...read more
ami-0287a05f0ef0e9d9a

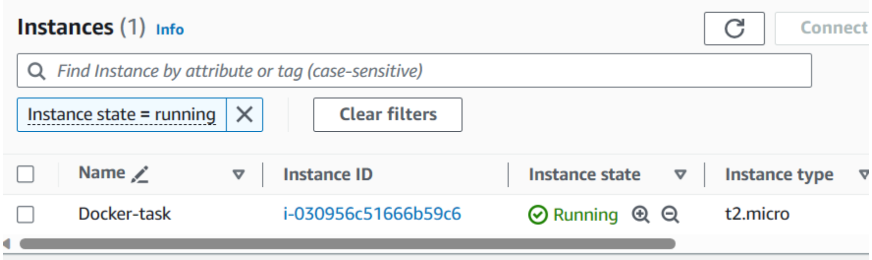
Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

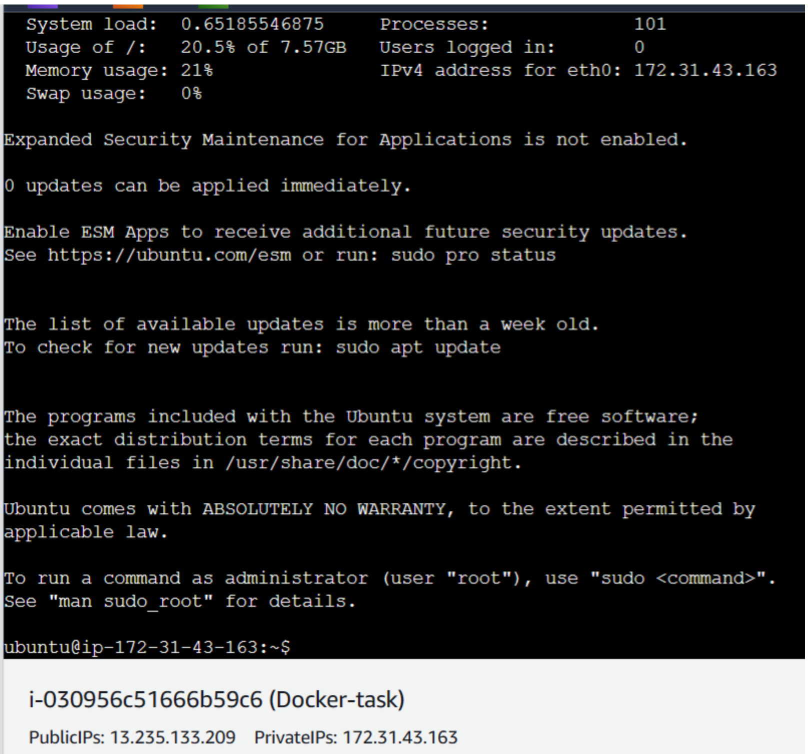
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which

➔ The instance has been launched successfully:



□	Name	Instance ID	Instance state	Instance type
□	Docker-task	i-030956c51666b59c6	✓ Running	t2.micro

➔ Then connect the instance with **putty or with instance connect option:**



```
  System load:  0.65185546875     Processes:             101
  Usage of /:   20.5% of 7.57GB   Users logged in:       0
  Memory usage: 21%               IPv4 address for eth0: 172.31.43.163
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-43-163:~$
```

i-030956c51666b59c6 (Docker-task)

PublicIPs: 13.235.133.209 PrivateIPs: 172.31.43.163

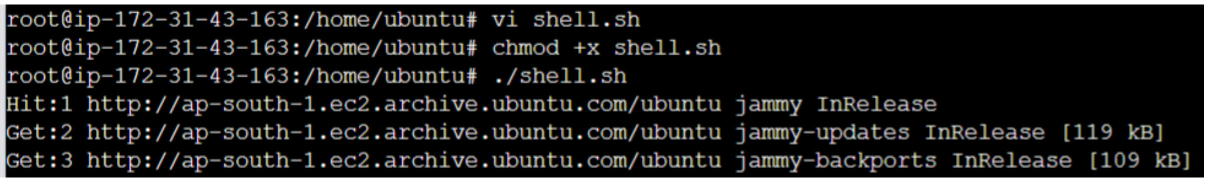
Step:2 – Containerizing the Java application:

➔ Installing necessary services and packages for this task:

- ❖ **Java – 11 or 17 version**
- ❖ **Maven**
- ❖ **Docker**
- ❖ **Docker-compose**

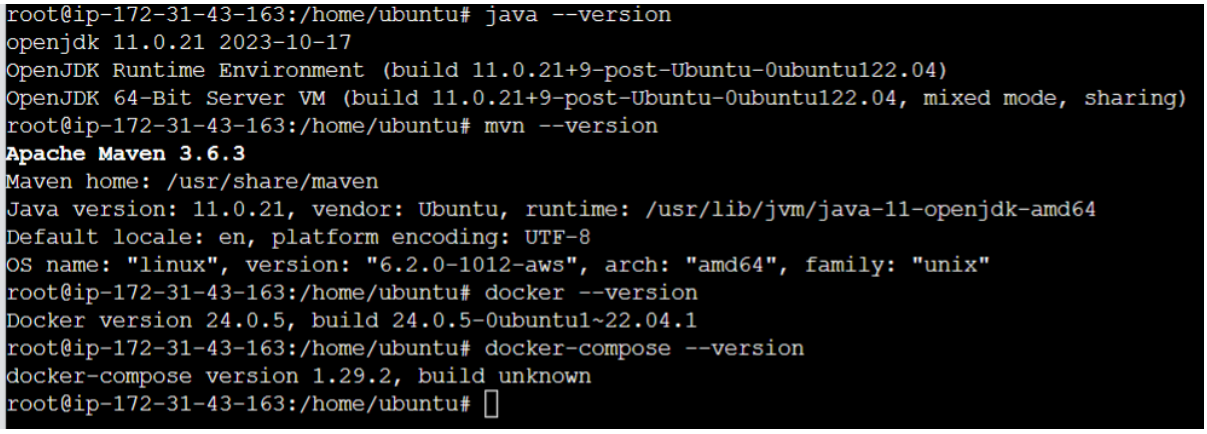
➔ Then I am going to create shell file and include the necessary scripts to enter the above services and packages:

```
#!/bin/bash
apt-get update
apt-get install -y openjdk-11-jre
apt-get install -y maven
apt-get install -y docker.io
apt-get install -y docker-compose
```



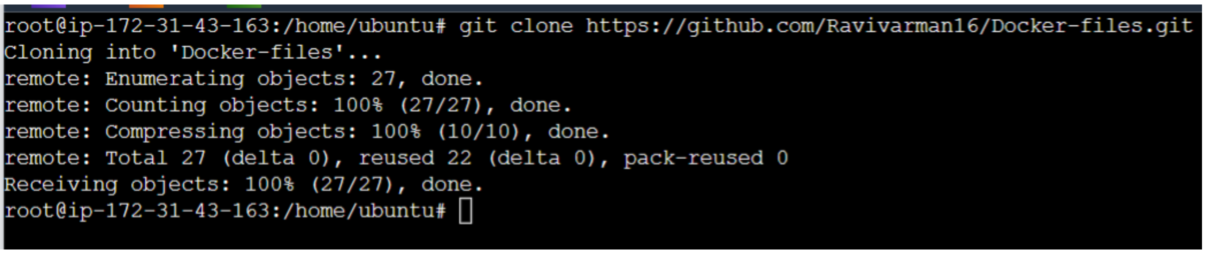
```
root@ip-172-31-43-163:/home/ubuntu# vi shell.sh
root@ip-172-31-43-163:/home/ubuntu# chmod +x shell.sh
root@ip-172-31-43-163:/home/ubuntu# ./shell.sh
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
```

➔ Checking whether the above packages has installed or not:



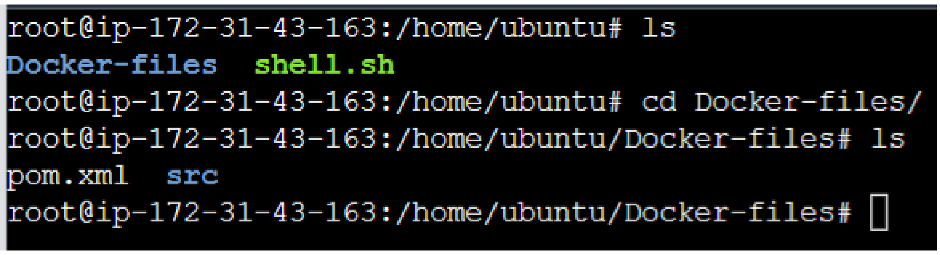
```
root@ip-172-31-43-163:/home/ubuntu# java --version
openjdk 11.0.21 2023-10-17
OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
root@ip-172-31-43-163:/home/ubuntu# mvn --version
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 11.0.21, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.2.0-1012-aws", arch: "amd64", family: "unix"
root@ip-172-31-43-163:/home/ubuntu# docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
root@ip-172-31-43-163:/home/ubuntu# docker-compose --version
docker-compose version 1.29.2, build unknown
root@ip-172-31-43-163:/home/ubuntu# 
```

➔ Then I am going to clone code from GitHub repository for java application:



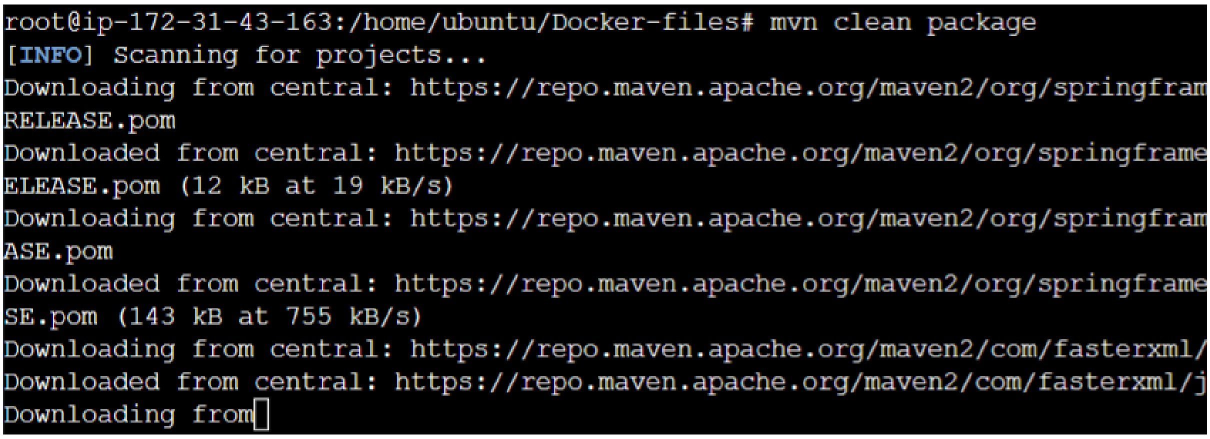
```
root@ip-172-31-43-163:/home/ubuntu# git clone https://github.com/Ravivarman16/Docker-files.git
Cloning into 'Docker-files'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 27 (delta 0), reused 22 (delta 0), pack-reused 0
Receiving objects: 100% (27/27), done.
root@ip-172-31-43-163:/home/ubuntu# 
```

➔ Then going inside the cloned directory:



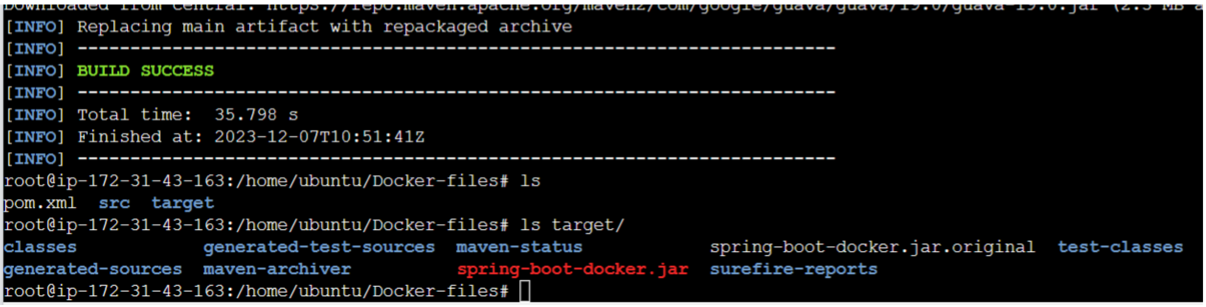
```
root@ip-172-31-43-163:/home/ubuntu# ls
Docker-files  shell.sh
root@ip-172-31-43-163:/home/ubuntu# cd Docker-files/
root@ip-172-31-43-163:/home/ubuntu/Docker-files# ls
pom.xml  src
root@ip-172-31-43-163:/home/ubuntu/Docker-files# 
```

➔ Then compiling and packaging java application with the help of maven with the command: **mvn clean package**



```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# mvn clean package
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/springfram
RELEASE.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframe
ELEASE.pom (12 kB at 19 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/springfram
ASE.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframe
SE.pom (143 kB at 755 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/com/fasterxml/
Downloaded from central: https://repo.maven.apache.org/maven2/com/fasterxml/j
Downloading from
```

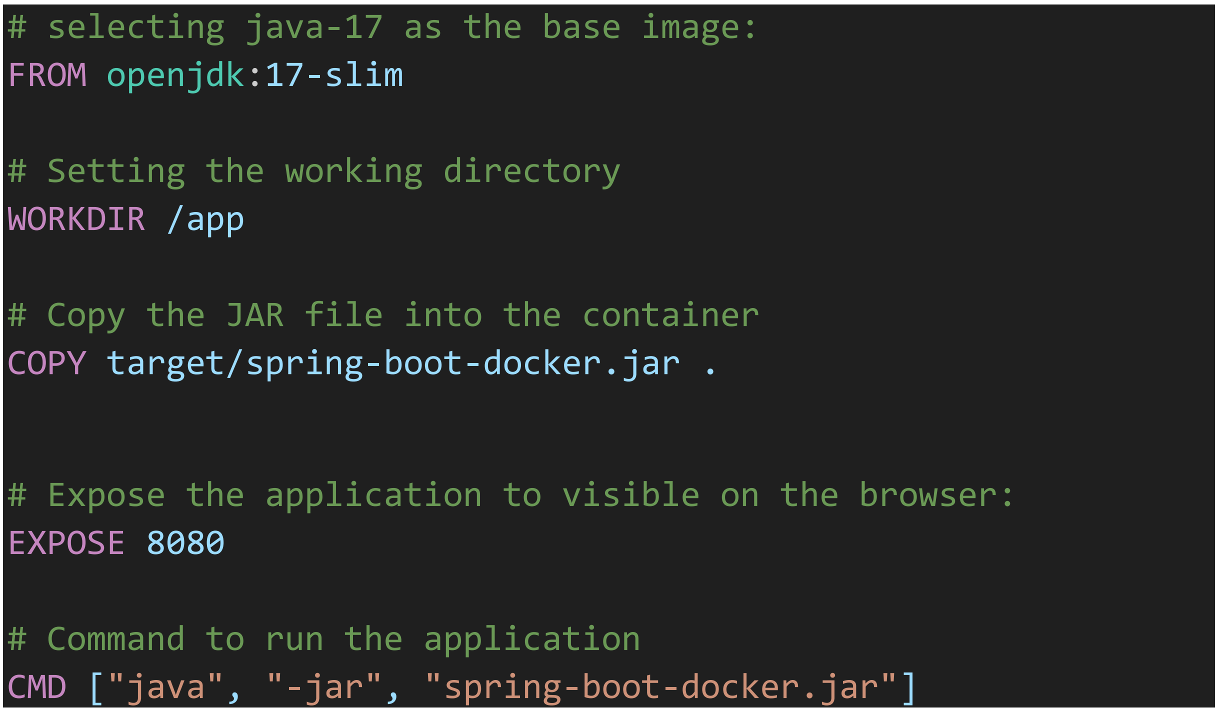
➔ Then we can able to see the build is success and we can able to see jar file is created successfully:



```
Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/19.0/guava-19.0.jar (2.3 MB a
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  35.798 s
[INFO] Finished at: 2023-12-07T10:51:41Z
[INFO] ------------------------------------------------------------------------
root@ip-172-31-43-163:/home/ubuntu/Docker-files# ls
pom.xml  src  target
root@ip-172-31-43-163:/home/ubuntu/Docker-files# ls target/
classes            generated-test-sources  maven-status            spring-boot-docker.jar.original  test-classes
generated-sources  maven-archiver          spring-boot-docker.jar  surefire-reports
root@ip-172-31-43-163:/home/ubuntu/Docker-files# 
```

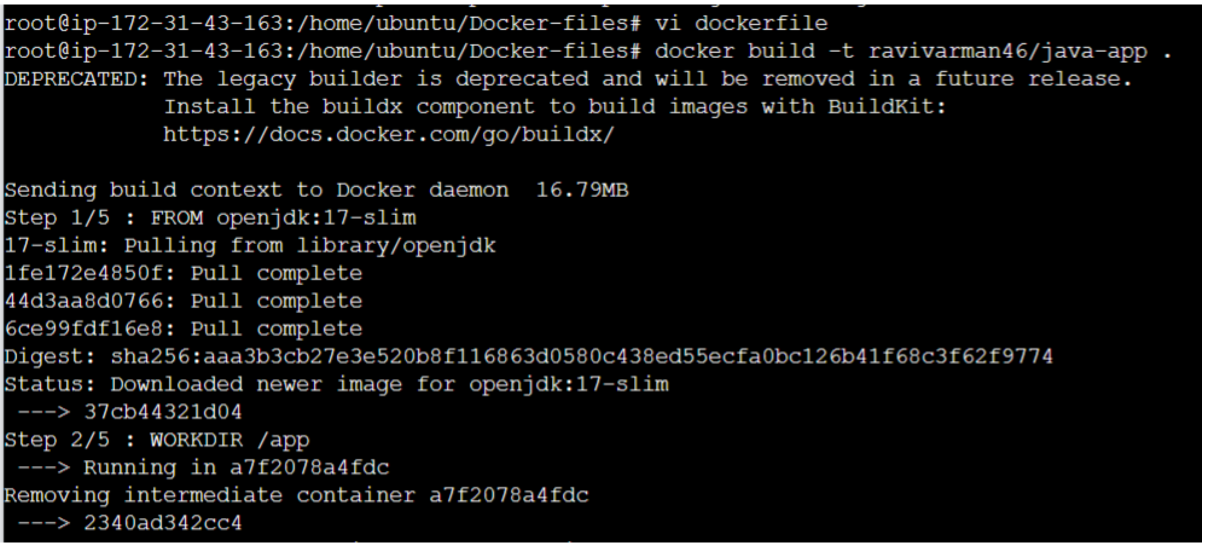
➔ Now we need to create Dockerfile for above application:

Dockerfile:



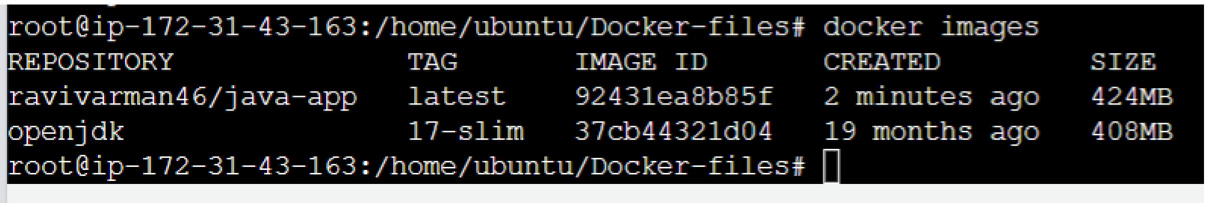
```
# selecting java-17 as the base image:
FROM openjdk:17-slim

# Setting the working directory
WORKDIR /app

# Copy the JAR file into the container
COPY target/spring-boot-docker.jar .


# Expose the application to visible on the browser:
EXPOSE 8080

# Command to run the application
CMD ["java", "-jar", "spring-boot-docker.jar"]
```

➔ Then creating docker image from the above dockerfile:



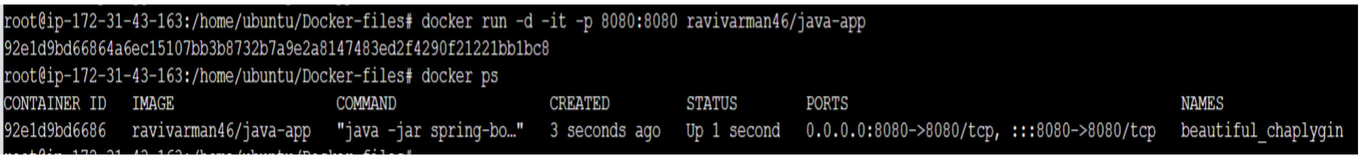
```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# vi dockerfile
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker build -t ravivarman46/java-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  16.79MB
Step 1/5 : FROM openjdk:17-slim
17-slim: Pulling from library/openjdk
1fe172e4850f: Pull complete
44d3aa8d0766: Pull complete
6ce99fdf16e8: Pull complete
Digest: sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc126b41f68c3f62f9774
Status: Downloaded newer image for openjdk:17-slim
 ---> 37cb44321d04
Step 2/5 : WORKDIR /app
 ---> Running in a7f2078a4fdc
Removing intermediate container a7f2078a4fdc
 ---> 2340ad342cc4
```

➔ Checking the docker image:



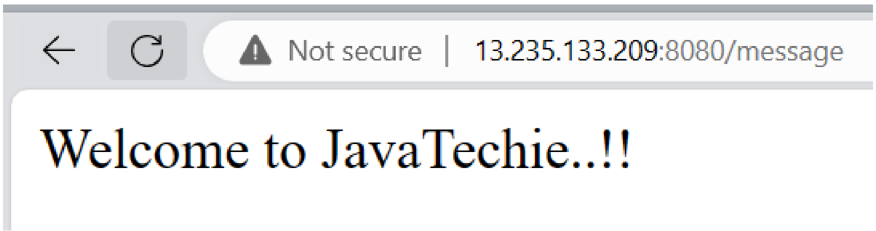
```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker images
REPOSITORY              TAG        IMAGE ID       CREATED         SIZE
ravivarman46/java-app   latest     92431ea8b85f   2 minutes ago   424MB
openjdk                 17-slim    37cb44321d04   19 months ago   408MB
root@ip-172-31-43-163:/home/ubuntu/Docker-files# 
```

- ➔ Then running the container from the above created image with the command: **docker run -d -it -p 8080:8080 <image-name>**



```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker run -d -it -p 8080:8080 ravivarman46/java-app
92e1d9bd66864a6ec15107bb3b8732b7a9e2a8147483ed2f4290f21221bb1bc8
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED         STATUS         PORTS                                       NAMES
92e1d9bd6686   ravivarman46/java-app   "java -jar spring-bo…"   3 seconds ago   Up 1 second    0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   beautiful_chaplygin
```

The browser output:



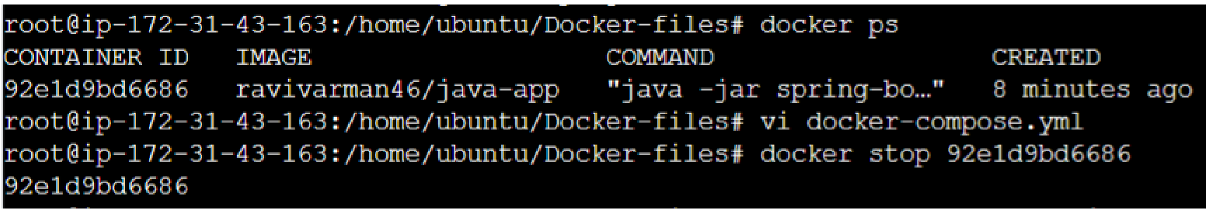
Welcome to JavaTechie..!!

- ➔ Deploying the application through docker-compose:

Docker-compose.yml file contains:

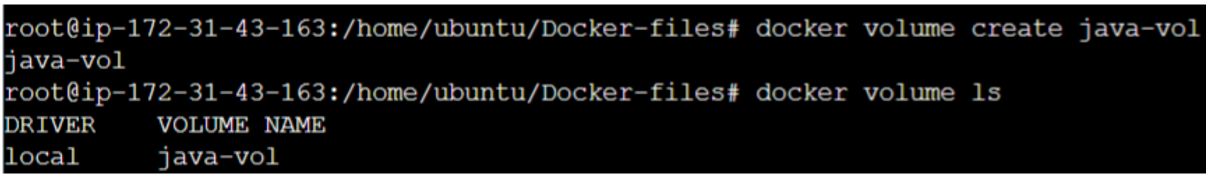
```
# docker-compose file for java:
version: '3'
services:
  java-app: # service name & you can give any name:
    image: ravivarman46/java-app #yours image name:
    container_name: java-app
    ports:
      - 8080:8080 # port mapping
    volumes:
      - java-vol:/app/

volumes:
  java-vol:
    external: true
```

- ➔ Stopping the existing running container:



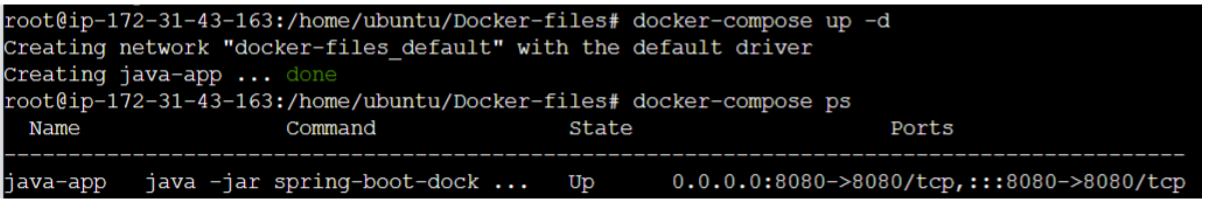
```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED
92e1d9bd6686   ravivarman46/java-app      "java -jar spring-bo…"   8 minutes ago
root@ip-172-31-43-163:/home/ubuntu/Docker-files# vi docker-compose.yml
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker stop 92e1d9bd6686
92e1d9bd6686
```

- ➔ Creating a docker volume with the command: **docker volume create <volume_name>**



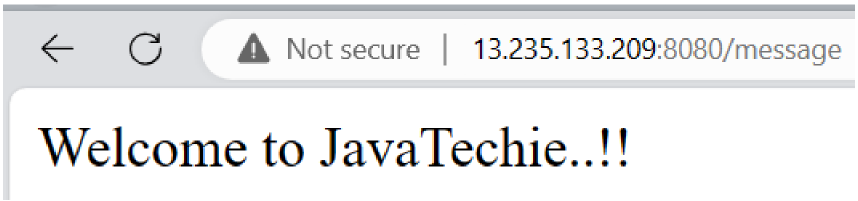
```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker volume create java-vol
java-vol
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker volume ls
DRIVER     VOLUME NAME
local      java-vol
```

- ➔ Then deploying the application: **docker-compose up -d**



```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker-compose up -d
Creating network "docker-files_default" with the default driver
Creating java-app ... done
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker-compose ps
  Name                  Command                State                     Ports
-------------------------------------------------------------------------------------------
java-app   java -jar spring-boot-dock ...   Up      0.0.0.0:8080->8080/tcp,:::8080->8080/tcp
```

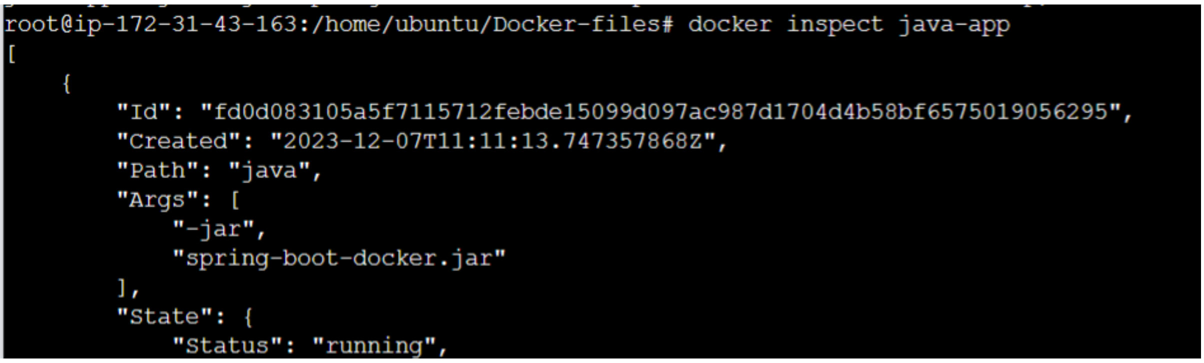
Browser output:



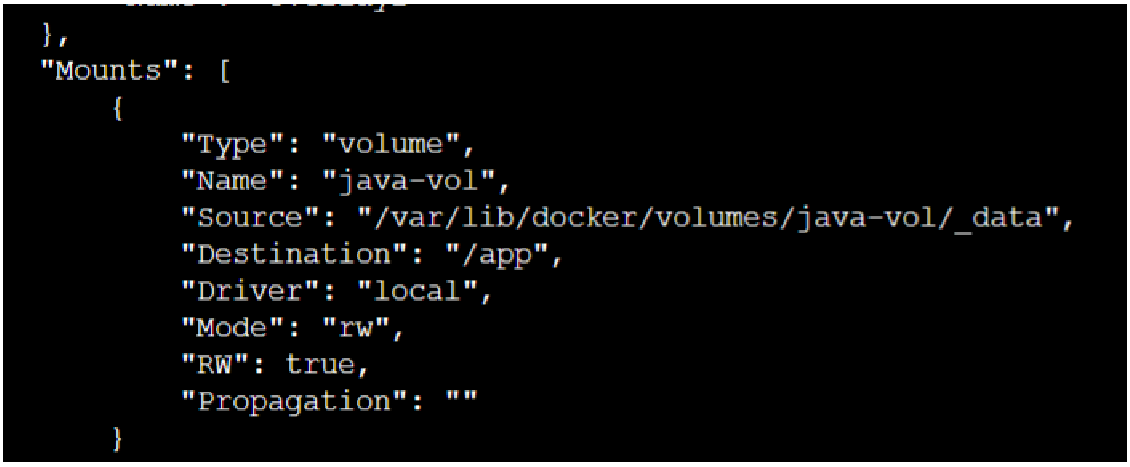
Not secure | 13.235.133.209:8080/message

Welcome to JavaTechie..!!

- ➔ Checking the volume is attached the container or not by using: **docker inspect <container name or container id>**



```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker inspect java-app
[
    {
        "Id": "fd0d083105a5f7115712febde15099d097ac987d1704d4b58bf6575019056295",
        "Created": "2023-12-07T11:11:13.747357868Z",
        "Path": "java",
        "Args": [
            "-jar",
            "spring-boot-docker.jar"
        ],
        "State": {
            "Status": "running",
```

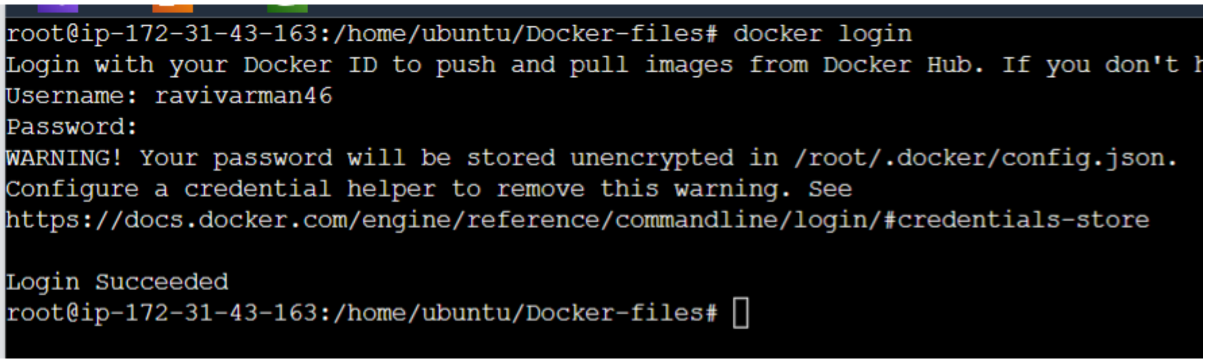


```
    },
    "Mounts": [
        {
            "Type": "volume",
            "Name": "java-vol",
            "Source": "/var/lib/docker/volumes/java-vol/_data",
            "Destination": "/app",
            "Driver": "local",
            "Mode": "rw",
            "RW": true,
            "Propagation": ""
        }
```

The volume is attached perfectly with the container:

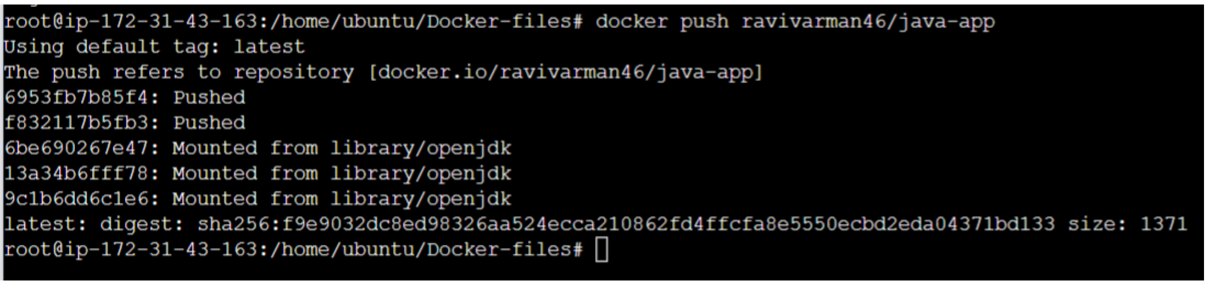
➔ Then pushing the created image to docker-hub, for that we need to login: **docker login**

under username you must give your Docker hub username and for password you must give your Docker hub password:



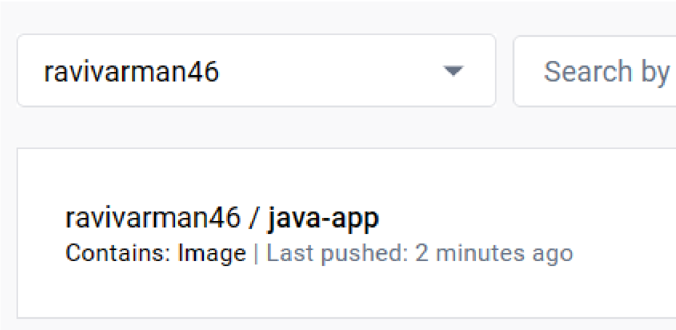
```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't h
Username: ravivarman46
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@ip-172-31-43-163:/home/ubuntu/Docker-files# 
```

➔ Then push the docker image to docker hub: **docker push <image name>**



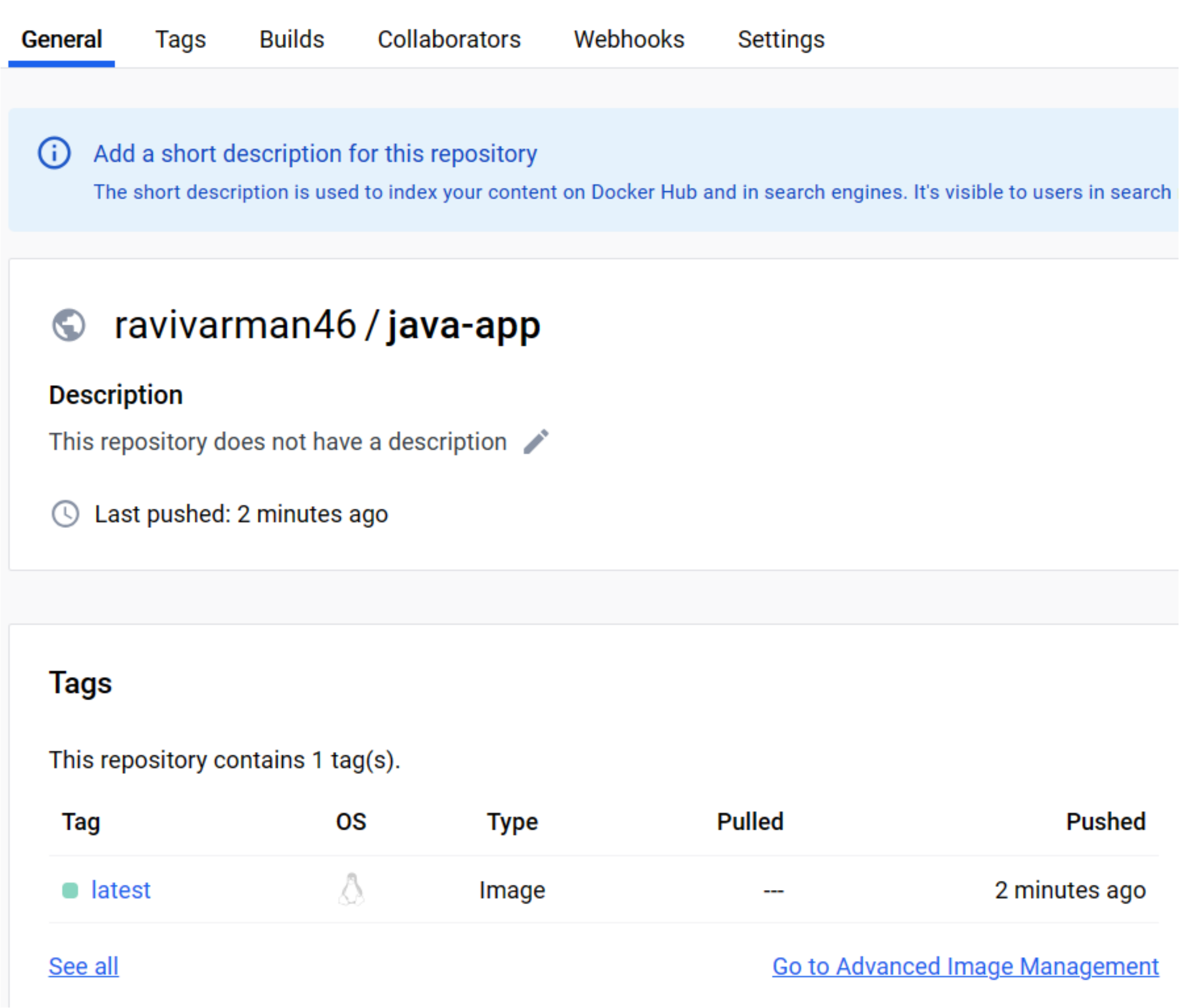
```
root@ip-172-31-43-163:/home/ubuntu/Docker-files# docker push ravivarman46/java-app
Using default tag: latest
The push refers to repository [docker.io/ravivarman46/java-app]
6953fb7b85f4: Pushed
f832117b5fb3: Pushed
6be690267e47: Mounted from library/openjdk
13a34b6fff78: Mounted from library/openjdk
9c1b6dd6c1e6: Mounted from library/openjdk
latest: digest: sha256:f9e9032dc8ed98326aa524ecca210862fd4ffcfa8e5550ecbd2eda04371bd133 size: 1371
root@ip-172-31-43-163:/home/ubuntu/Docker-files# 
```

Docker hub output:



ravivarman46 / java-app

Contains: Image | Last pushed: 2 minutes ago



Add a short description for this repository

The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search

ravivarman46 / java-app

Description

This repository does not have a description

Last pushed: 2 minutes ago

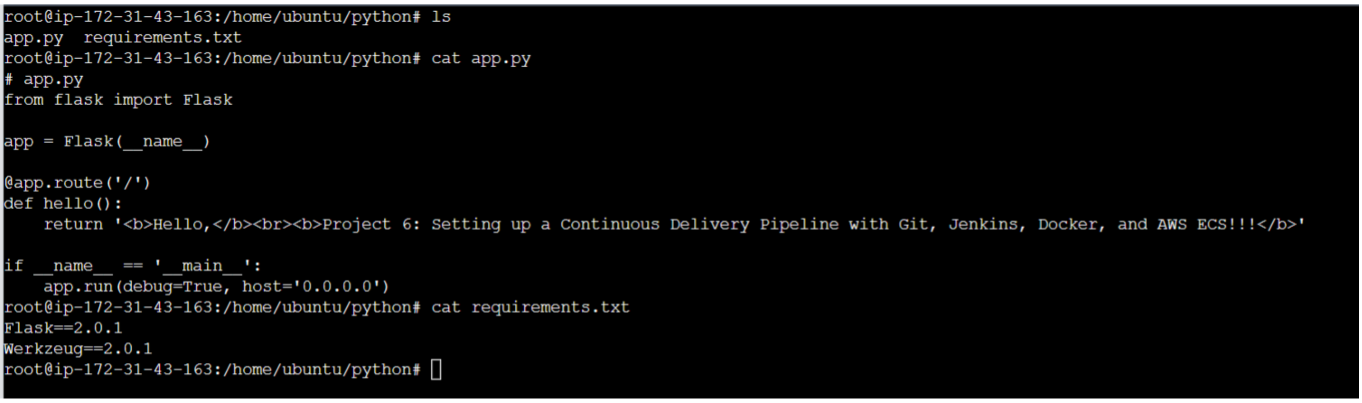
Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	2 minutes ago

Step:3 – Containerizing Python application:

➔ Assume you are having python application code like this:



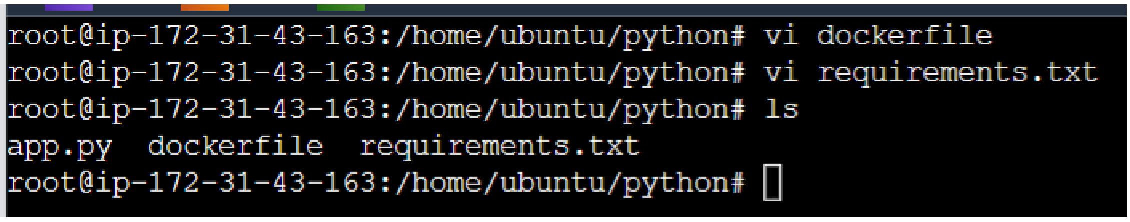
```
root@ip-172-31-43-163:/home/ubuntu/python# ls
app.py  requirements.txt
root@ip-172-31-43-163:/home/ubuntu/python# cat app.py
# app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return '<b>Hello,</b><br><b>Project 6: Setting up a Continuous Delivery Pipeline with Git, Jenkins, Docker, and AWS ECS!!!</b>'

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
root@ip-172-31-43-163:/home/ubuntu/python# cat requirements.txt
Flask==2.0.1
Werkzeug==2.0.1
root@ip-172-31-43-163:/home/ubuntu/python# 
```

➔ Creating dockerfile for above python application:

Dockerfile contains:

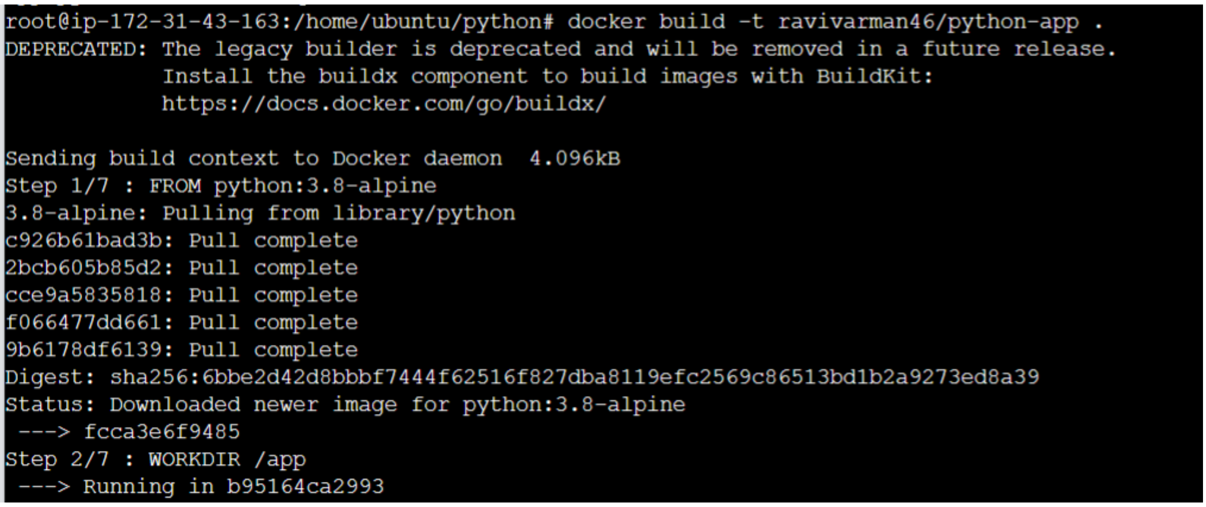
```
#choosing the base image:
FROM python:3.8-alpine

#choosing working directory for the application:
WORKDIR /app

#copying the requirements.txt file to app directory and
installing packages:
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

#copying the rest of application code to the working
directory:
COPY . .

#exposing the application:
EXPOSE 5000

#Executing the application after creating image:
CMD ["python", "app.py"]
```

➔ The dockerfile has been created successfully:



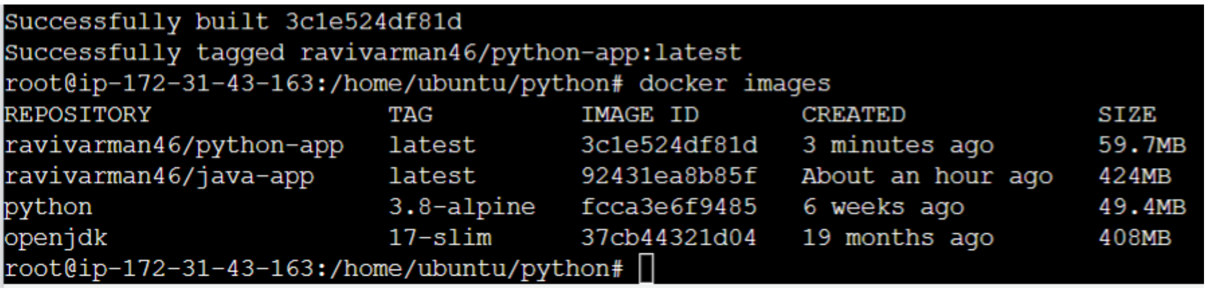
```
root@ip-172-31-43-163:/home/ubuntu/python# vi dockerfile
root@ip-172-31-43-163:/home/ubuntu/python# vi requirements.txt
root@ip-172-31-43-163:/home/ubuntu/python# ls
app.py  dockerfile  requirements.txt
root@ip-172-31-43-163:/home/ubuntu/python# 
```

- ➔ Creating a docker image from the dockerfile: **docker build -t <image_name> .**



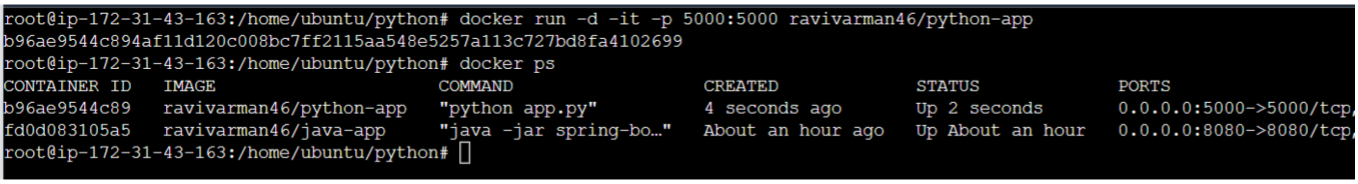
```
root@ip-172-31-43-163:/home/ubuntu/python# docker build -t ravivarman46/python-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM python:3.8-alpine
3.8-alpine: Pulling from library/python
c926b61bad3b: Pull complete
2bcb605b85d2: Pull complete
cce9a5835818: Pull complete
f066477dd661: Pull complete
9b6178df6139: Pull complete
Digest: sha256:6bbe2d42d8bbbf7444f62516f827dba8119efc2569c86513bd1b2a9273ed8a39
Status: Downloaded newer image for python:3.8-alpine
 ---> fcca3e6f9485
Step 2/7 : WORKDIR /app
 ---> Running in b95164ca2993
```

- ➔ Checking the created docker image: **docker images**



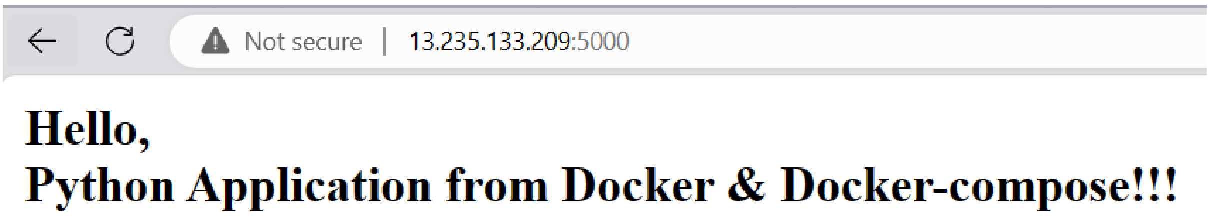
```
Successfully built 3c1e524df81d
Successfully tagged ravivarman46/python-app:latest
root@ip-172-31-43-163:/home/ubuntu/python# docker images
REPOSITORY                TAG          IMAGE ID       CREATED          SIZE
ravivarman46/python-app   latest       3c1e524df81d   3 minutes ago    59.7MB
ravivarman46/java-app     latest       92431ea8b85f   About an hour ago   424MB
python                    3.8-alpine   fcca3e6f9485   6 weeks ago      49.4MB
openjdk                   17-slim      37cb44321d04   19 months ago    408MB
root@ip-172-31-43-163:/home/ubuntu/python# 
```

- ➔ Creating a container from the above image: **docker run -d -it -p 5000:5000 ravivarman46/python-app**



```
root@ip-172-31-43-163:/home/ubuntu/python# docker run -d -it -p 5000:5000 ravivarman46/python-app
b96ae9544c894af11d120c008bc7ff2115aa548e5257a113c727bd8fa4102699
root@ip-172-31-43-163:/home/ubuntu/python# docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED             STATUS              PORTS
b96ae9544c89   ravivarman46/python-app   "python app.py"          4 seconds ago       Up 2 seconds        0.0.0.0:5000->5000/tcp,
fd0d083105a5   ravivarman46/java-app     "java -jar spring-bo…"   About an hour ago   Up About an hour    0.0.0.0:8080->8080/tcp,
root@ip-172-31-43-163:/home/ubuntu/python# 
```

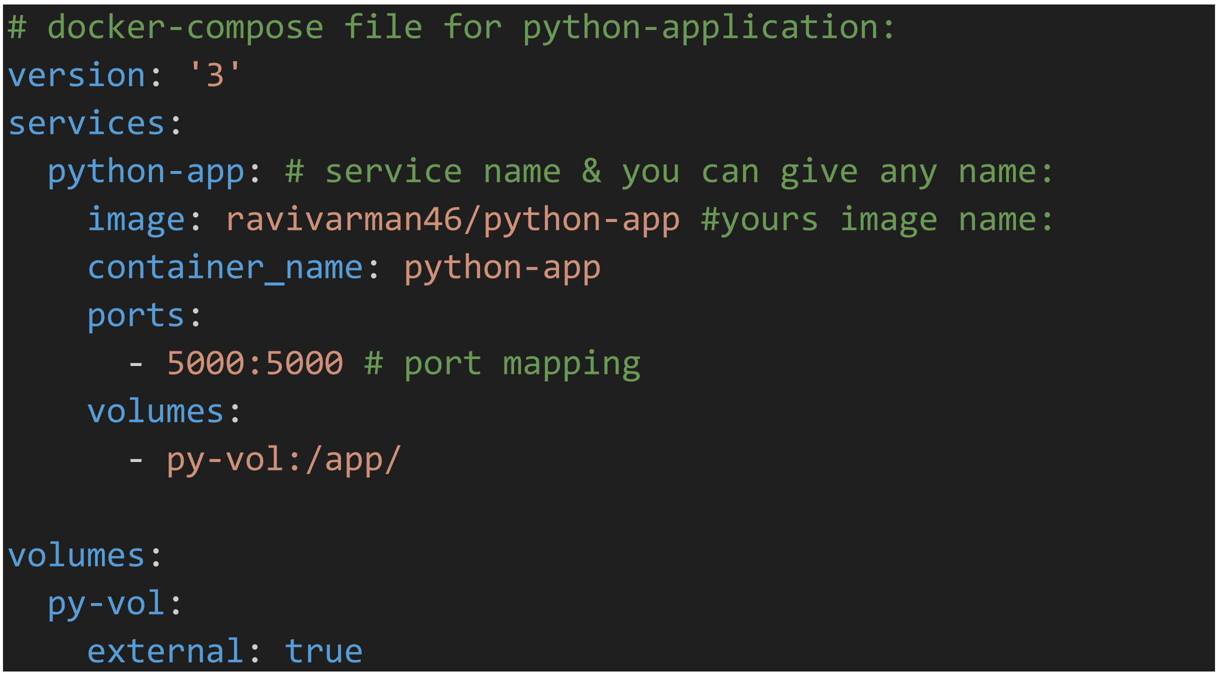
Browser output:



Hello,
Python Application from Docker & Docker-compose!!!

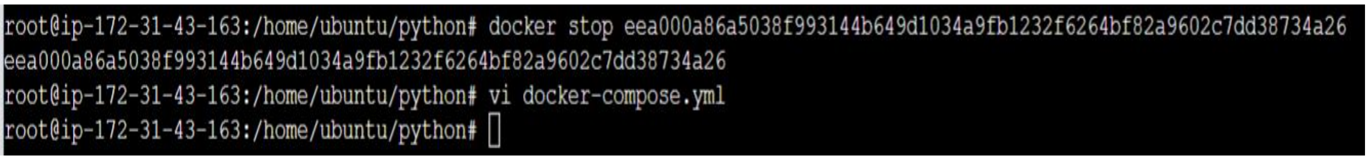
- ➔ Deploying the python application through docker-compose:

Docker-compose.yml file contains:



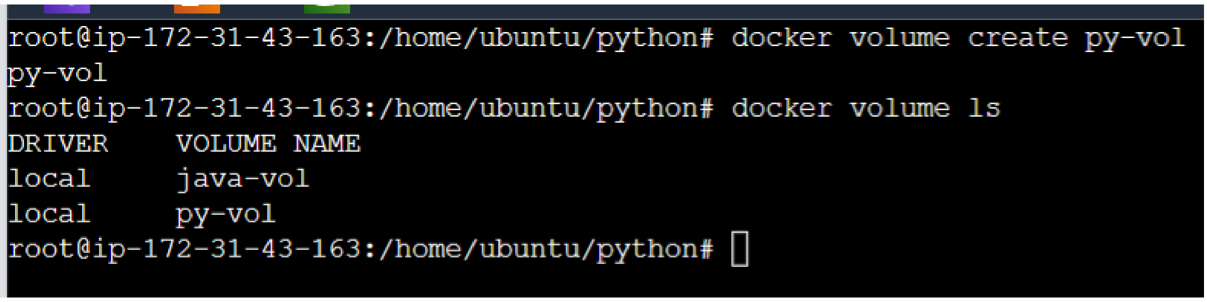
```
# docker-compose file for python-application:
version: '3'
services:
  python-app: # service name & you can give any name:
    image: ravivarman46/python-app #yours image name:
    container_name: python-app
    ports:
      - 5000:5000 # port mapping
    volumes:
      - py-vol:/app/

volumes:
  py-vol:
    external: true
```

- ➔ Stopping the existing container: **docker stop <container_id or container name>**
- ➔ Creating docker-compose.yml file:



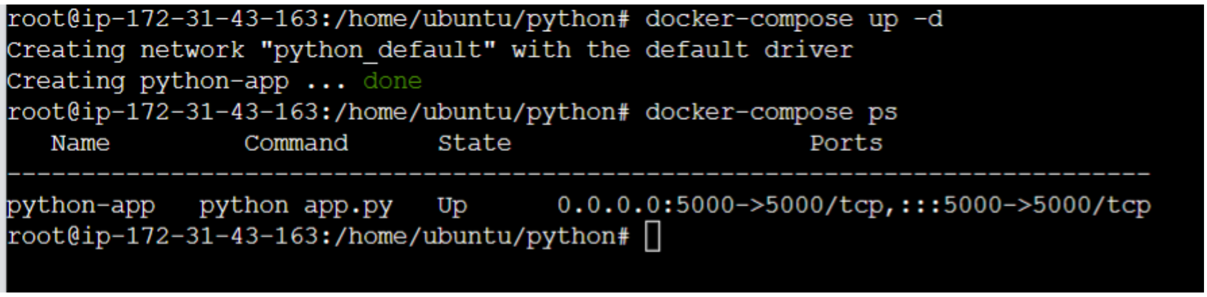
```
root@ip-172-31-43-163:/home/ubuntu/python# docker stop eea000a86a5038f993144b649d1034a9fb1232f6264bf82a9602c7dd38734a26
eea000a86a5038f993144b649d1034a9fb1232f6264bf82a9602c7dd38734a26
root@ip-172-31-43-163:/home/ubuntu/python# vi docker-compose.yml
root@ip-172-31-43-163:/home/ubuntu/python# 
```

- ➔ Creating a docker-volume**: docker volume create <volume_name> and docker volume ls**



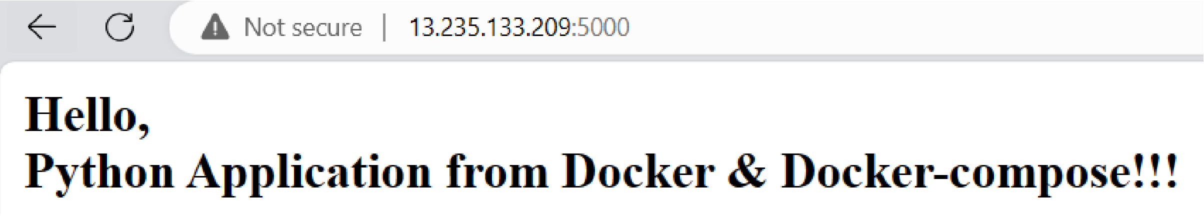
```
root@ip-172-31-43-163:/home/ubuntu/python# docker volume create py-vol
py-vol
root@ip-172-31-43-163:/home/ubuntu/python# docker volume ls
DRIVER    VOLUME NAME
local     java-vol
local     py-vol
root@ip-172-31-43-163:/home/ubuntu/python# 
```

➔ Deploying it through docker-compose: **docker-compose up -d**



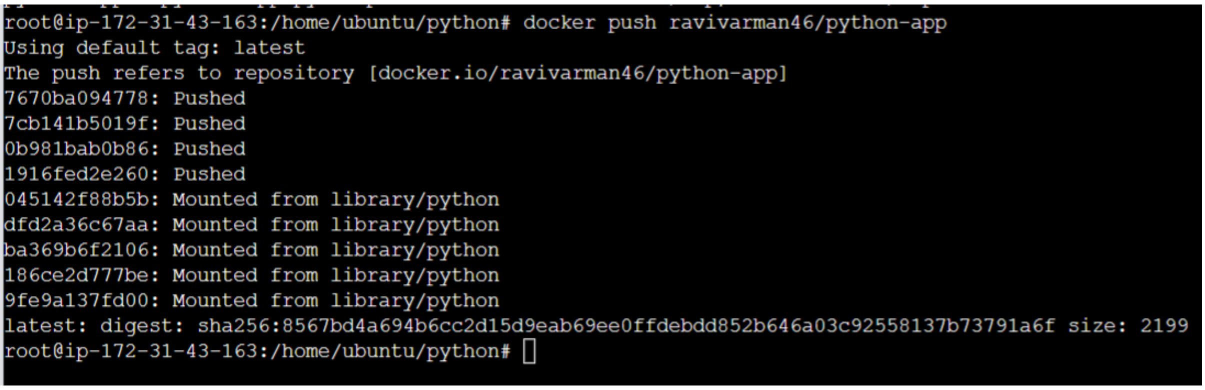
```
root@ip-172-31-43-163:/home/ubuntu/python# docker-compose up -d
Creating network "python_default" with the default driver
Creating python-app ... done
root@ip-172-31-43-163:/home/ubuntu/python# docker-compose ps
   Name         Command      State                   Ports
-----------------------------------------------------------------------------
python-app   python app.py   Up      0.0.0.0:5000->5000/tcp,:::5000->5000/tcp
root@ip-172-31-43-163:/home/ubuntu/python# 
```

Browser output:



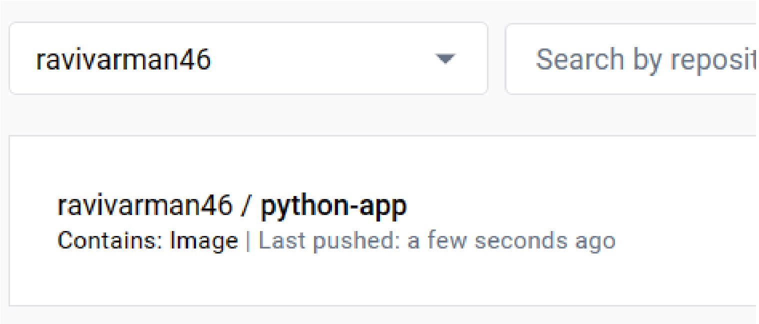
Hello,
Python Application from Docker & Docker-compose!!!

➔ Pushing it image to Docker hub: **docker push <image_name>**



```
root@ip-172-31-43-163:/home/ubuntu/python# docker push ravivarman46/python-app
Using default tag: latest
The push refers to repository [docker.io/ravivarman46/python-app]
7670ba094778: Pushed
7cb141b5019f: Pushed
0b981bab0b86: Pushed
1916fed2e260: Pushed
045142f88b5b: Mounted from library/python
dfd2a36c67aa: Mounted from library/python
ba369b6f2106: Mounted from library/python
186ce2d777be: Mounted from library/python
9fe9a137fd00: Mounted from library/python
latest: digest: sha256:8567bd4a694b6cc2d15d9eab69ee0ffdebdd852b646a03c92558137b73791a6f size: 2199
root@ip-172-31-43-163:/home/ubuntu/python# 
```

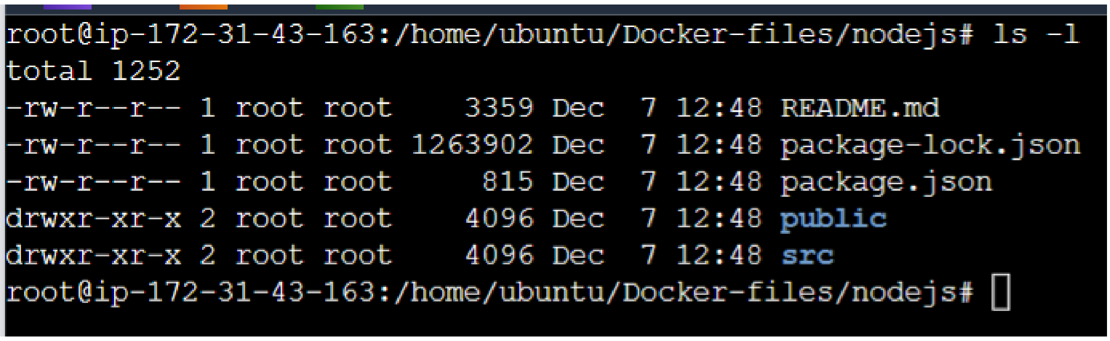
Docker hub output:



ravivarman46 / python-app
Contains: Image | Last pushed: a few seconds ago

Step:4 – Containerizing the Nodejs Application:

➔ Assume that you have nodejs application like this:



```
root@ip-172-31-43-163:/home/ubuntu/Docker-files/nodejs# ls -l
total 1252
-rw-r--r-- 1 root root    3359 Dec  7 12:48 README.md
-rw-r--r-- 1 root root 1263902 Dec  7 12:48 package-lock.json
-rw-r--r-- 1 root root     815 Dec  7 12:48 package.json
drwxr-xr-x 2 root root    4096 Dec  7 12:48 public
drwxr-xr-x 2 root root    4096 Dec  7 12:48 src
root@ip-172-31-43-163:/home/ubuntu/Docker-files/nodejs# 
```

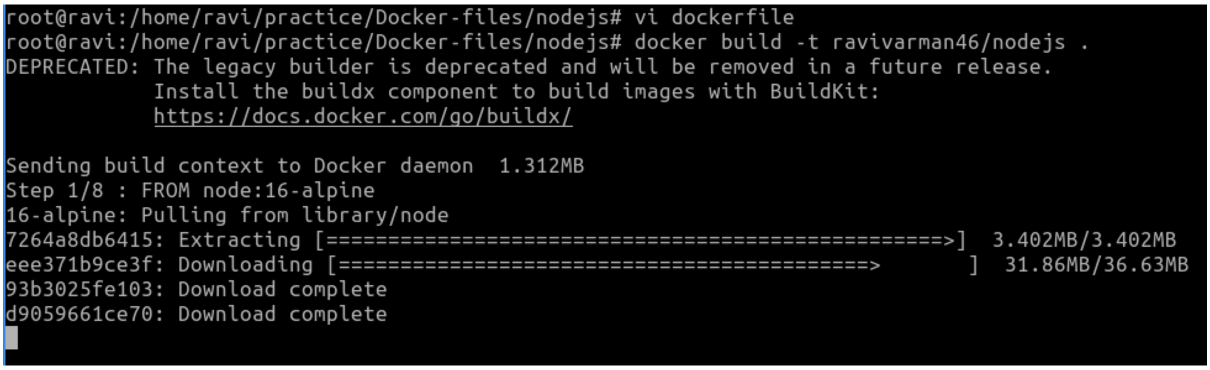
➔ Creating a dockerfile for above nodejs application:

Dockerfile contains:

```
#choosing the base image:
FROM node:16-alpine

#choosing working directory for the application:
WORKDIR /app

#copying the package.json file to app directory and
installing packages:
COPY package.json .
RUN npm install

#copying the rest of application code to the working
directory:
COPY . .
```

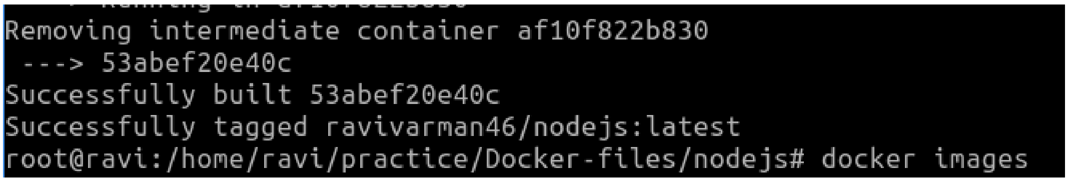
```
#building the application:
RUN npm run build

#exposing the application:
EXPOSE 3000

#Executing the application after creating image:
CMD ["npm", "start"]
```

➔ Then creating a docker image from above dockerfile:

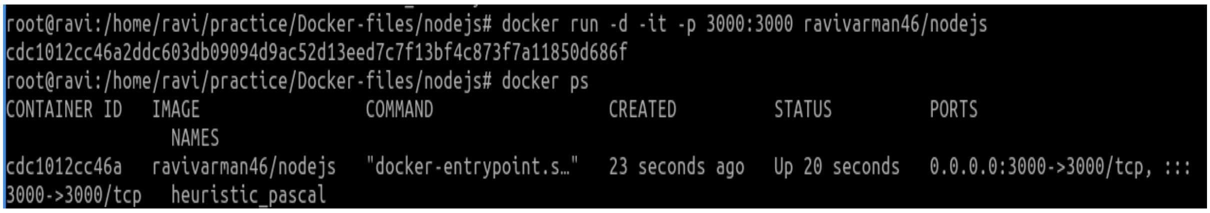


```
root@ravi:/home/ravi/practice/Docker-files/nodejs# vi dockerfile
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker build -t ravivarman46/nodejs .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  1.312MB
Step 1/8 : FROM node:16-alpine
16-alpine: Pulling from library/node
7264a8db6415: Extracting [==================================================>]  3.402MB/3.402MB
eee371b9ce3f: Downloading [===========================================>       ]  31.86MB/36.63MB
93b3025fe103: Download complete
d9059661ce70: Download complete
```

➔ Checking the image & running the container from the above image:

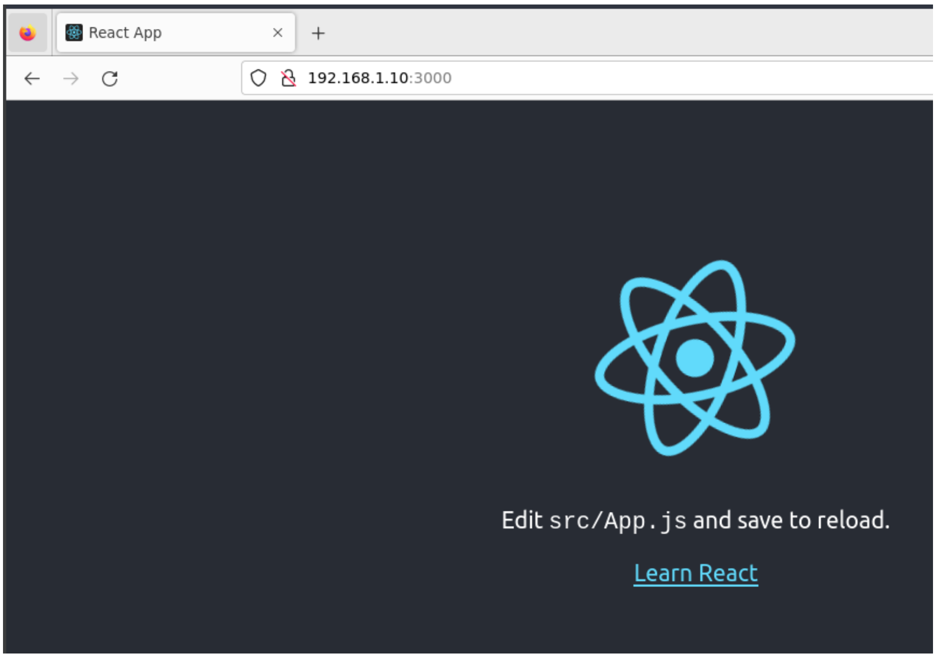


```
Removing intermediate container af10f822b830
 ---> 53abef20e40c
Successfully built 53abef20e40c
Successfully tagged ravivarman46/nodejs:latest
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker images
```



```
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker run -d -it -p 3000:3000 ravivarman46/nodejs
cdc1012cc46a2ddc603db09094d9ac52d13eed7c7f13bf4c873f7a11850d686f
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS
                 NAMES
cdc1012cc46a   ravivarman46/nodejs   "docker-entrypoint.s…"   23 seconds ago   Up 20 seconds   0.0.0.0:3000->3000/tcp, :::
3000->3000/tcp   heuristic_pascal
```

Browser output:

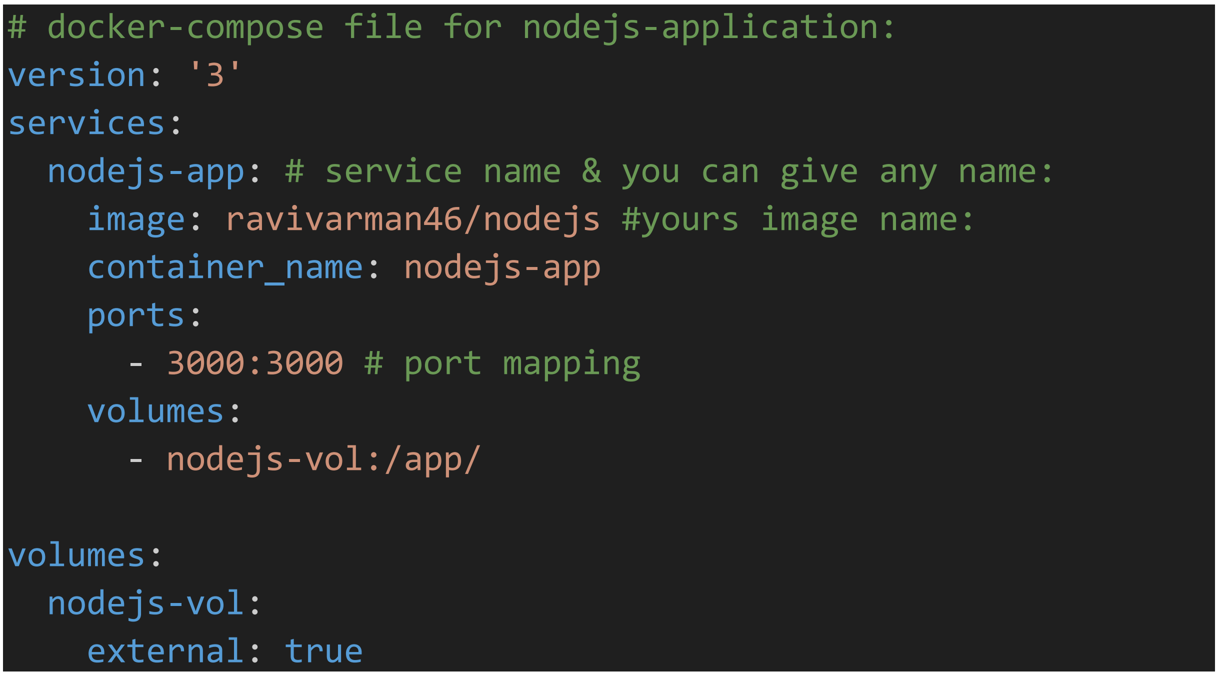


Edit src/App.js and save to reload.

Learn React

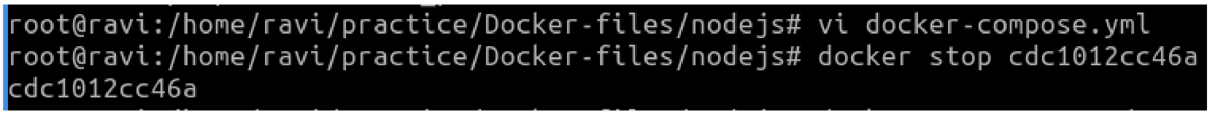
➔ Deploying the nodejs application through docker-compose:

Docker-compose.yml:



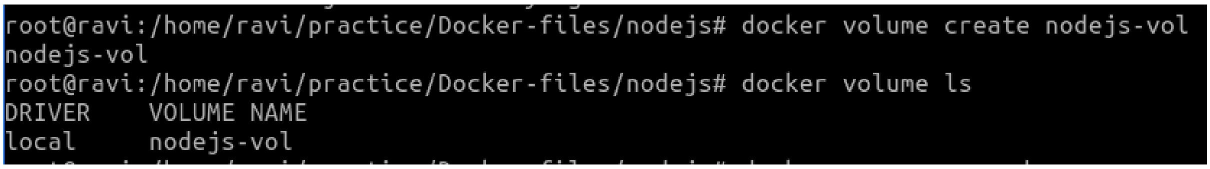
```
# docker-compose file for nodejs-application:
version: '3'
services:
  nodejs-app: # service name & you can give any name:
    image: ravivarman46/nodejs #yours image name:
    container_name: nodejs-app
    ports:
      - 3000:3000 # port mapping
    volumes:
      - nodejs-vol:/app/

volumes:
  nodejs-vol:
    external: true
```

➔ Stopping the already running container:



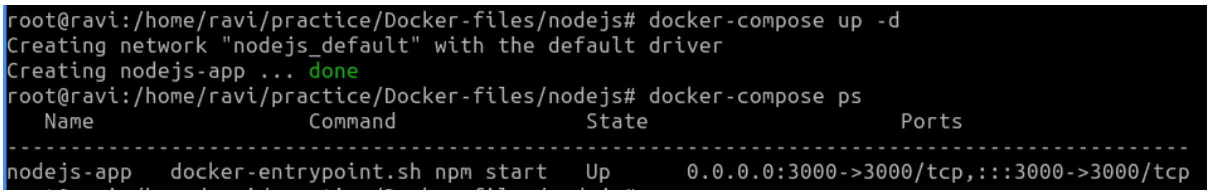
```
root@ravi:/home/ravi/practice/Docker-files/nodejs# vi docker-compose.yml
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker stop cdc1012cc46a
cdc1012cc46a
```

➔ Creating docker volume for nodejs application:



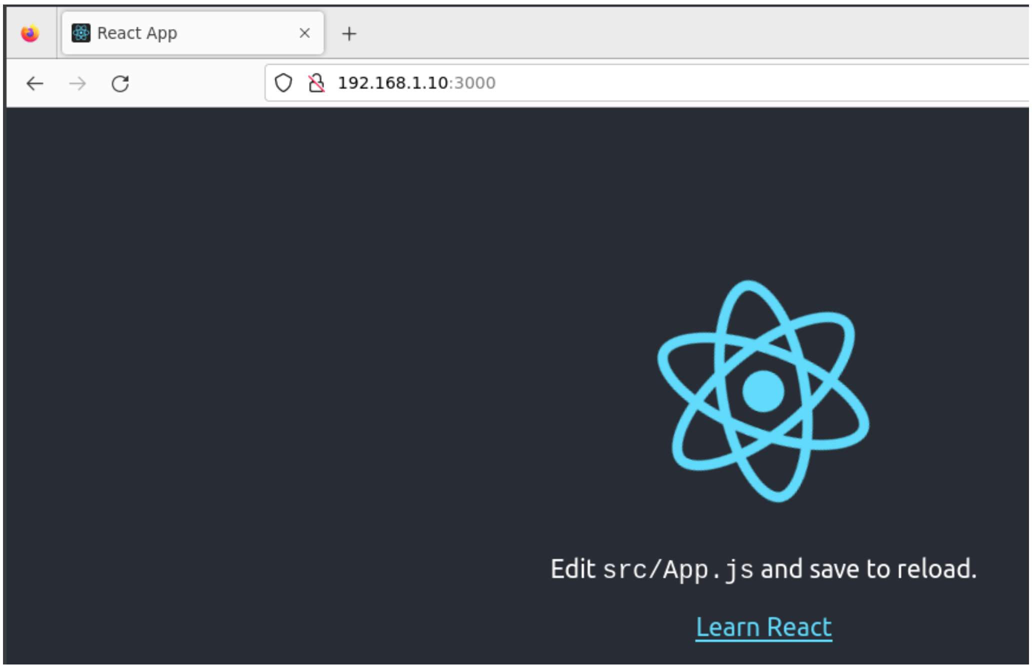
```
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker volume create nodejs-vol
nodejs-vol
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker volume ls
DRIVER    VOLUME NAME
local     nodejs-vol
```

➔ Deploying the application:

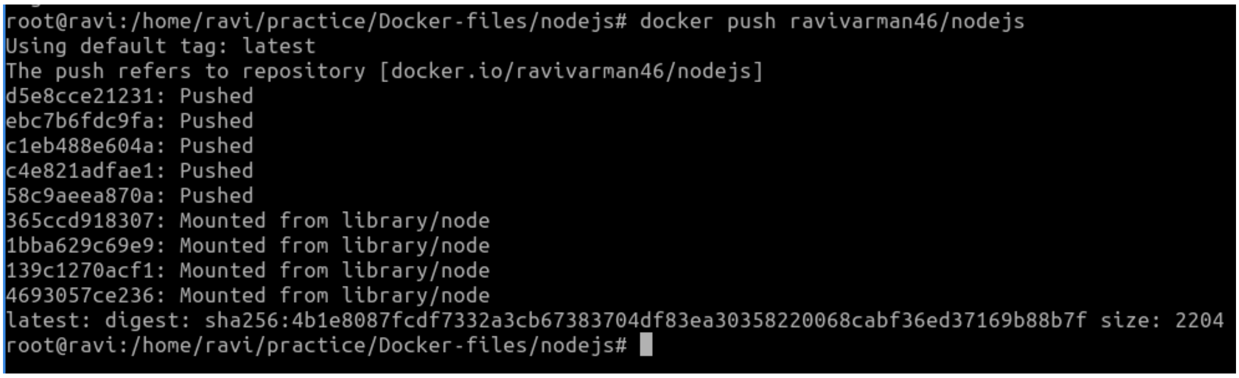


```
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker-compose up -d
Creating network "nodejs_default" with the default driver
Creating nodejs-app ... done
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker-compose ps
   Name               Command             State                  Ports
-------------------------------------------------------------------------------------------
nodejs-app   docker-entrypoint.sh npm start   Up      0.0.0.0:3000->3000/tcp,:::3000->3000/tcp
```

Browser output:

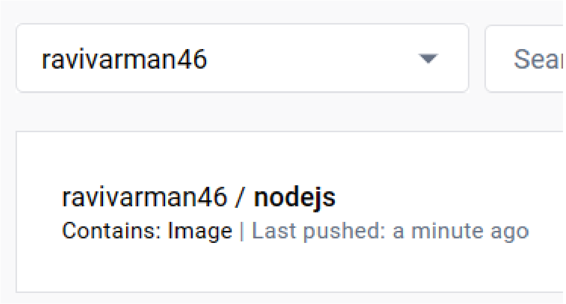


➔ Pushing the above nodejs image to Docker hub:



```
root@ravi:/home/ravi/practice/Docker-files/nodejs# docker push ravivarman46/nodejs
Using default tag: latest
The push refers to repository [docker.io/ravivarman46/nodejs]
d5e8cce21231: Pushed
ebc7b6fdc9fa: Pushed
c1eb488e604a: Pushed
c4e821adfae1: Pushed
58c9aeea870a: Pushed
365ccd918307: Mounted from library/node
1bba629c69e9: Mounted from library/node
139c1270acf1: Mounted from library/node
4693057ce236: Mounted from library/node
latest: digest: sha256:4b1e8087fcdf7332a3cb67383704df83ea30358220068cabf36ed37169b88b7f size: 2204
root@ravi:/home/ravi/practice/Docker-files/nodejs# █
```

Docker hub output:

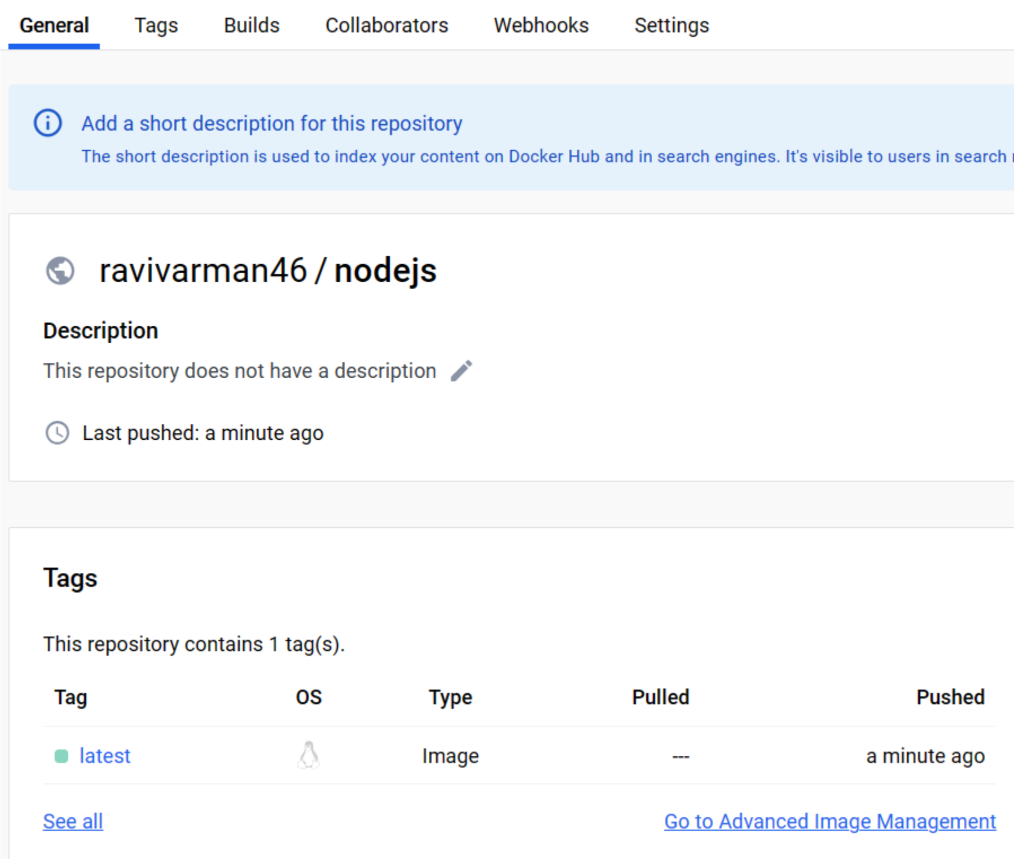


ravivarman46

Sea

ravivarman46 / **nodejs**

Contains: Image | Last pushed: a minute ago



Add a short description for this repository

The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search

ravivarman46 / **nodejs**

Description

This repository does not have a description

Last pushed: a minute ago

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	a minute ago

See all

Go to Advanced Image Management

The image has been pushed to docker hub successfully:

Benefits of above task:

- ➔ **Streamlined Development:** Containerization simplifies the development process by encapsulating each application within a Docker container, ensuring consistent environments, and reducing compatibility issues.

- ➔ **Efficient Deployment:** Docker Compose orchestrates the deployment of multiple containers, providing a straightforward and unified method for deploying Java, Python, and Node.js applications.

- ➔ **Enhanced Scalability:** Containers facilitate easy scaling of applications, allowing developers to efficiently manage workloads and adapt to changing demands. This ensures optimal performance and resource utilization

----

All the files for the given task have been uploaded to the following GitHub repository: https://github.com/Ravivarman16/Docker-files.git