Deploying Node.js Application with Docker and Eureka Service Registry on AWS EC2

Introduction

This comprehensive guide walks you through deploying a Node.js application using Docker containers and registering it with a Eureka service registry on an AWS EC2 instance. We'll delve into the advantages of Docker, Docker Compose (both for future reference and manual deployment understanding), AWS EC2, and Eureka. Detailed instructions with code snippets will equip you to replicate this setup.

Table of Contents

1. Why Docker?

- Docker excels at packaging your application and its dependencies into a standalone unit called a container.
- These containers ensure consistent application behavior across various environments (development, testing, production), eliminating the "it works on my machine" problem.
- Docker simplifies deployment and enhances scalability as containers can be easily scaled up or down based on demand.

2. Why Docker Compose?

- While not used in this manual deployment, Docker Compose simplifies managing multi-container applications.
- It allows defining the services, networks, and volumes required for your application in a single YAML file (docker-compose.yml).

- With a single command (docker-compose up -d), you can spin up all services defined in the Compose file, streamlining deployments for complex microservices applications.
- Consider using Docker Compose for future deployments to automate service management.

3. Why AWS EC2?

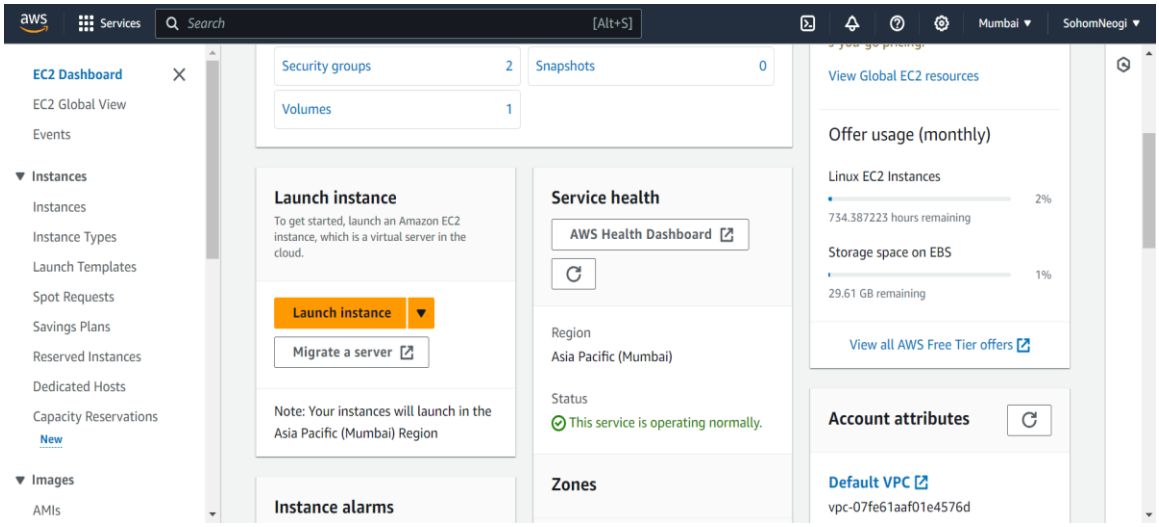
- AWS EC2 (Elastic Compute Cloud) provides scalable compute capacity in the cloud. We'll use EC2 instances to deploy Docker containers and run our Node.js applications. EC2 instances are secure, highly customizable, and scalable, making them ideal for hosting applications in the cloud.

4. Why Eureka Service Registry?

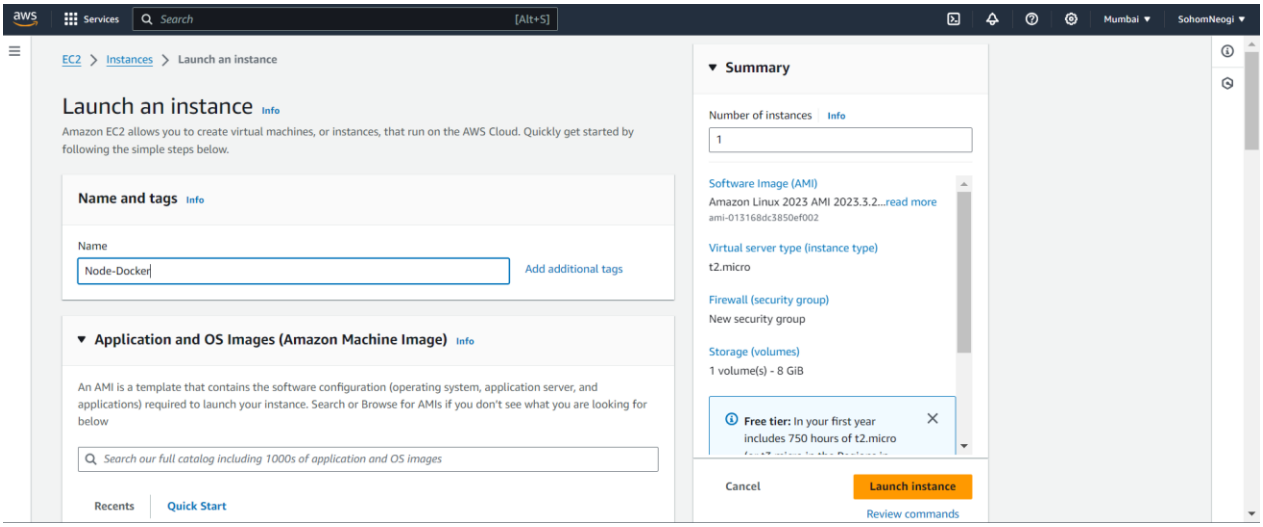
- Eureka, a service registry tool developed by Netflix, empowers microservices to register themselves and discover other services within the ecosystem. By registering services with Eureka, we achieve service discovery and load balancing. This ensures seamless communication between services, even as they scale up or down.

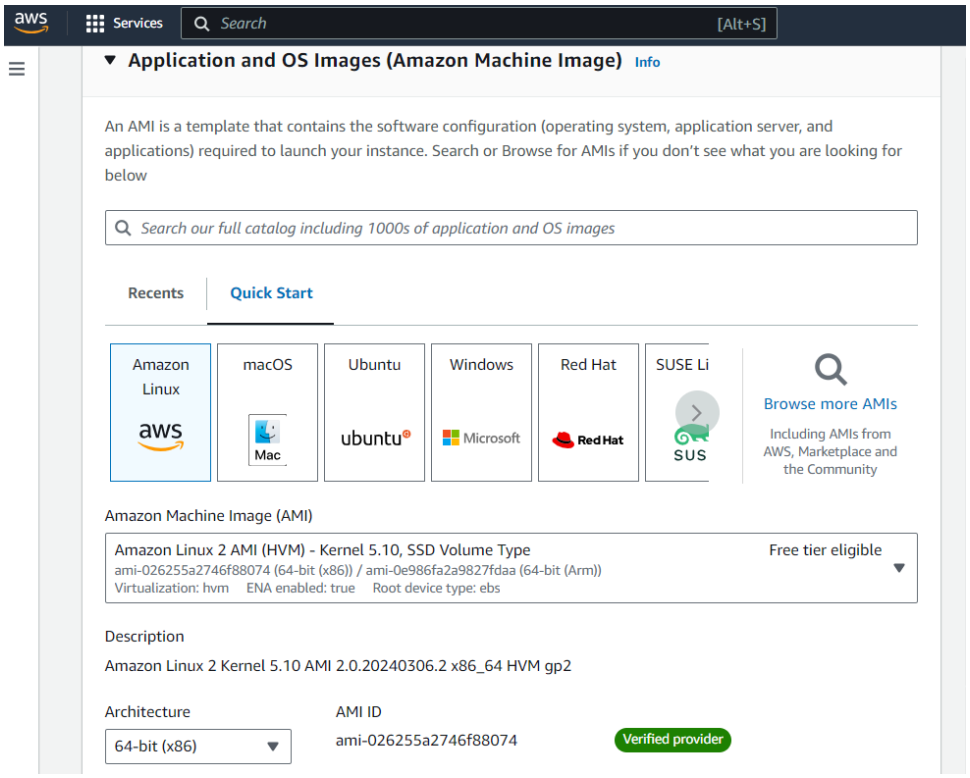
5. Step-by-Step Deployment:

- **Login to the AWS Management Console and navigate to EC2.**

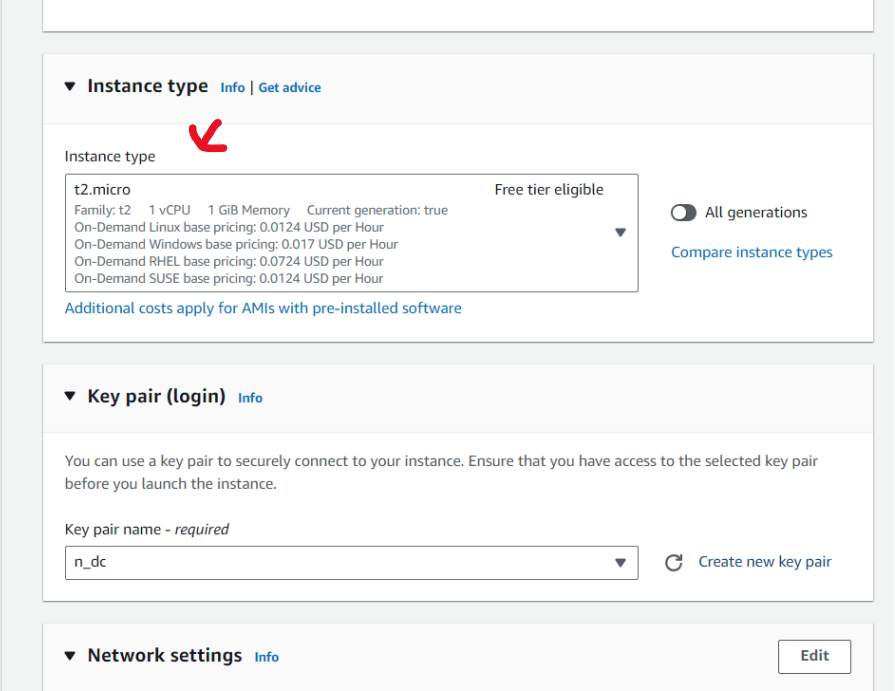


- **Launch a new instance, providing a name and selecting an appropriate Application/OS image (e.g., Ubuntu, Amazon Linux 2).**

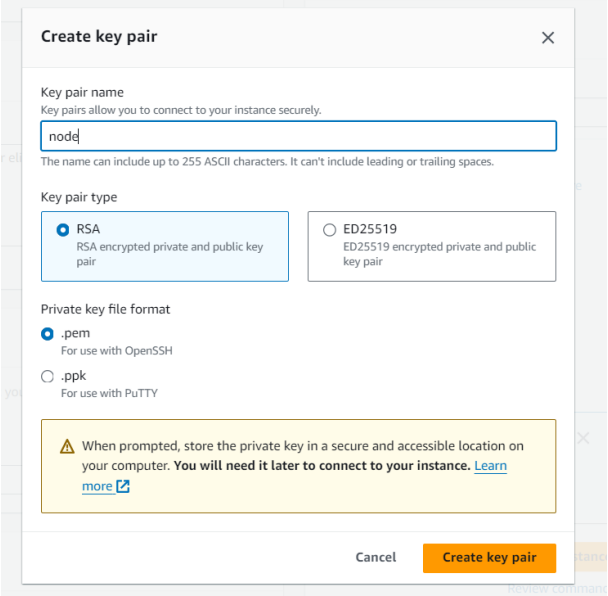




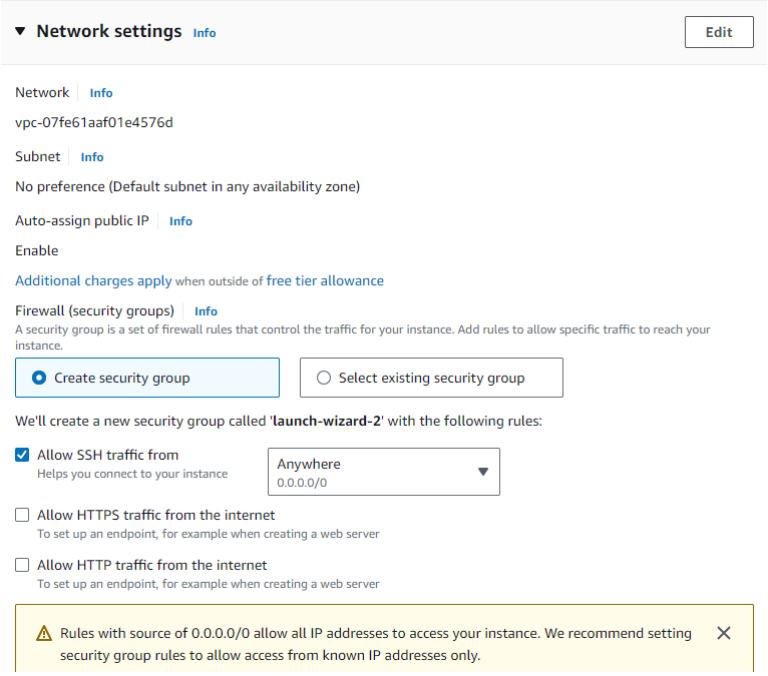
- **Choose an instance type (e.g., "t2.micro") based on your application's resource requirements.**



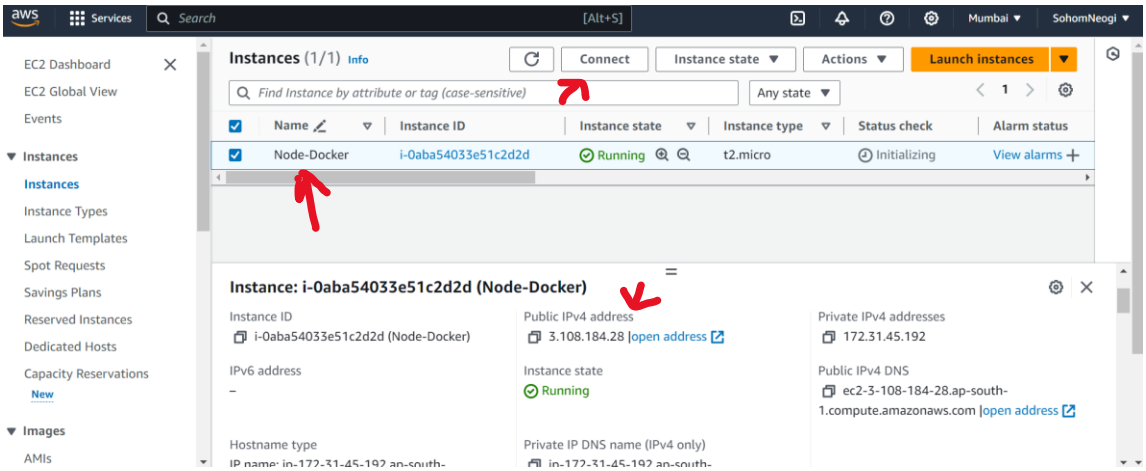
- **Create a key pair and download the private key (.pem) file, storing it securely in your application directory.**



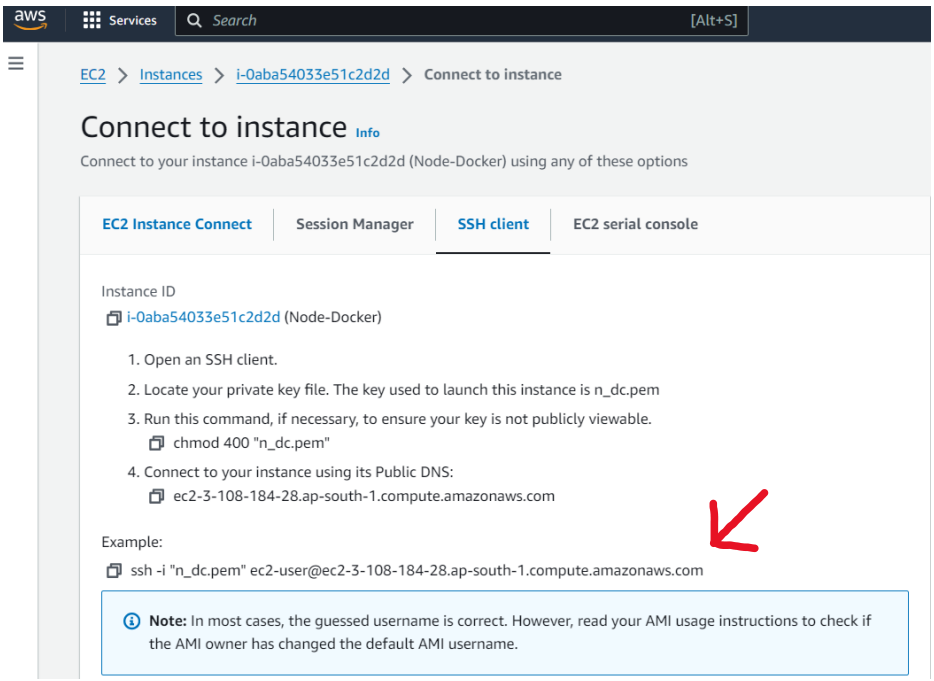
- **Configure security groups to allow inbound traffic for ports used by your services (typically HTTP/HTTPS ports for Node.js applications).**



- **After launching the instance, it would appear like this under the instances tab, wait for it until it' in the running state**



- **Connect to the instance using SSH Client**



- **Use your SSH client (e.g., Git Bash) and the downloaded private key file to connect to the instance using the following command, replacing placeholders with your actual details:**

- **Open git bash, redirect to your application directory**

```
ec2-user@ip-172-31-45-192:~

sohom.neogi@LT-1134-HO MINGW64 ~
$ cd "C:\Node-apis\NodeDev"
```

- **Copy the ssh key in your git bash (e.g - "ssh –i "n_dc.pem...."") for making connection to your instance**

```
sohom.neogi@LT-1134-HO MINGW64 /c/Node-apis/NodeDev (dev)
$ ssh -i "n_dc.pem" ec2-user@ec2-3-108-184-28.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-3-108-184-28.ap-south-1.compute.amazonaws.com (3.1
08.184.28)' can't be established.
ED25519 key fingerprint is SHA256:YII6OS5neEjMqAOiW8LYJiV4MnTmTrNC5PhcJlV3DBc.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:8: ec2-43-204-22-12.ap-south-1.compute.amazonaws.com
    ~/.ssh/known_hosts:10: ec2-52-66-49-230.ap-south-1.compute.amazonaws.com
    ~/.ssh/known_hosts:11: ec2-52-66-130-250.ap-south-1.compute.amazonaws.com
    ~/.ssh/known_hosts:12: ec2-65-2-148-212.ap-south-1.compute.amazonaws.com
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-108-184-28.ap-south-1.compute.amazonaws.com' (
ED25519) to the list of known hosts.
Last login: Fri Mar 15 04:18:31 2024 from 59.160.211.138
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/
```

- **Install Docker and on the EC2 instance and start the server with admin permission**

  - Install docker on EC2 - sudo yum install docker

- Start docker service - sudo service docker start

- Give admin permission to user - sudo usermod -aG ec2 -user

- Reinstall docker again - sudo yum install docker

- Reboot the system - sudo reboot

- Use this again to reconnect to the instance - ssh -i "n_dc.pem" ec2-user@ec2-52-66-130-250.ap-south-1.compute.amazonaws.com

- This hostname is subject to change on restarting the instance everytime (update it accordingly) -52-66-130-250

- Restart the docker service - sudo service docker start

- To see docker container/image information or to check whether docker has installed properly or not – docker info

- Build docker image of your applications in your local system

    - First create a DockerFile in your app directory

```
FROM node:16-alpine
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 8000
ENV NODE_ENV=development
CMD ["npm", "run", "dev"]
```

- Build an image of the docker file in your local machine terminal – docker image build -t <dockerHub_username>/<app-name>:<version_number> -f <file_name> .

- Push the latest changes into dockerHub - docker push <dockerHub_username>/<app-name>:<version_number>

- Since my application is NodeJS application I have pulled an already set up image of an Eureka Server - docker pull steeltoeoss/eureka-server

- Run the pulled Eureka Server image in EC2 instance - docker run -d -p 8761:8761 steeltoeoss/eureka-server

- Pull your latest updated service/services from DockerHub that you've pushed from your local machine into the EC2 instance – docker pull <dockerHub_username>/<app-name>:<version_number>

- Run the service/services – docker run –d –p PORT: PORT --name <any-name> <dockerHub-username>/<app-name>

- To check for running containers (Eureka Server & Service 1) – docker ps

```
[ec2-user@ip-172-31-45-192 ~]$ docker run -d -p 8761:8761 --name e1 steeltoeoss/
eureka-server
51f0d84b1da61065bd9b5de8f27d7edc4203fae09fdb7b437b25973198304125
[ec2-user@ip-172-31-45-192 ~]$ docker run -d -p 8000:8000 --name s1 sohom179/dem
o-app:1.0.0
795649e17f27c9daa7c021a31c0127239444fab44871e3ed46ac9caa14040073
[ec2-user@ip-172-31-45-192 ~]$ docker run -d -p 5001:5000 --name s2 sohom179/dem
o-service2:1.0.0
5dfa15ffe28293104ef9c1d6cba1d5d41149af6cc334a23b12c7843aaae9a968
[ec2-user@ip-172-31-45-192 ~]$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED
         STATUS          PORTS                                       NAMES
5dfa15ffe282   sohom179/demo-service2:1.0.0   "docker-entrypoint.s…"   4 seconds
 ago    Up 3 seconds    0.0.0.0:5001->5000/tcp, :::5001->5000/tcp   s2
795649e17f27   sohom179/demo-app:1.0.0        "docker-entrypoint.s…"   10 second
s ago   Up 9 seconds    0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   s1
51f0d84b1da6   steeltoeoss/eureka-server      "java -Djava.securit…"   18 second
s ago   Up 17 seconds   0.0.0.0:8761->8761/tcp, :::8761->8761/tcp   e1
[ec2-user@ip-172-31-45-192 ~]$ |
```

- Now, install a npm package "npm i eureka-js-client" into your node application and create a file "eureka-client.js" in your current Node application to make connection to the Eureka Server possible

```js
const Eureka = require("eureka-js-client").Eureka;

const client = new Eureka({
  // application instance information
  instance: {
    app: "nodedev",
    hostName: "3.108.184.28",
    ipAddr: "3.108.184.28",
    port: {
      $: 8000,
      "@enabled": "true",
    },
    vipAddress: "nodedev",
    statusPageUrl: "http://3.108.184.28:8000",
    healthCheckUrl: "http://3.108.184.28:8000/health",
    dataCenterInfo: {
      "@class": "com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo",
      name: "MyOwn",
    },
  },
  eureka: {
    // eureka server host / port / serviceUrls
    host: "3.108.184.28",
    port: 8761,
    servicePath: "/eureka/apps/",
  },
});

client.start();

client.on("error", (error) => {
  console.error("Error with Eureka client", error);
});
```

- Stop the current service – docker container stop <container_id>/<name>

```
[ec2-user@ip-172-31-45-192 ~]$ docker stop e1
e1
[ec2-user@ip-172-31-45-192 ~]$ docker stop s1
s1
[ec2-user@ip-172-31-45-192 ~]$ docker stop s2
s2
[ec2-user@ip-172-31-45-192 ~]$ |
```

- Rebuild the image, push the latest build from your local machine and pull the updated service into your EC2 instance and re-run it with the above commands

```
[ec2-user@ip-172-31-45-192 ~]$ docker pull sohom179/demo-app:1.0.0
1.0.0: Pulling from sohom179/demo-app
7264a8db6415: Already exists
eee371b9ce3f: Already exists
93b3025fe103: Already exists
d9059661ce70: Already exists
99064bcd6f44: Already exists
6036f66cac4f: Already exists
4757ae3994d7: Already exists
e2c9f57eb4d6: Pull complete
Digest: sha256:e282a88c16e189ec9e4d7bf65cdde937e43cf74be9b2433049a56ad9918400c4
Status: Downloaded newer image for sohom179/demo-app:1.0.0
docker.io/sohom179/demo-app:1.0.0
[ec2-user@ip-172-31-45-192 ~]$ docker pull sohom179/demo-service2:1.0.0
1.0.0: Pulling from sohom179/demo-service2
7264a8db6415: Already exists
eee371b9ce3f: Already exists
93b3025fe103: Already exists
d9059661ce70: Already exists
f52860d78ca0: Pull complete
45fbfdacff77: Pull complete
a9187dc2b7ac: Pull complete
07a8a6bf6411: Pull complete
Digest: sha256:31570bbf9a34484d7c54d93cc1ff2eabf6ef3bf204326e0e6e50af593073382a
Status: Downloaded newer image for sohom179/demo-service2:1.0.0
```

```
[ec2-user@ip-172-31-45-192 ~]$ docker run -d -p 8761:8761 --name e1 steeltoeoss/
eureka-server
d9c6fc26558731b574fbe57a75c3f286a1e060b2d890d6abe9aa6abd326b8004
[ec2-user@ip-172-31-45-192 ~]$ docker run -d -p 8000:8000 --name s1 sohom179/dem
o-app:1.0.0
01379c90bd36a96818b477d2a3811faf59d5aca9c79882d139fb7c2109c7fefa
[ec2-user@ip-172-31-45-192 ~]$ docker run -d -p 5001:5000 --name s2 sohom179/dem
o-service2:1.0.0
da1645b4b61b658d4b901ddd3139652f6fcfedf818367c017652e6c9a8aa193c
[ec2-user@ip-172-31-45-192 ~]$ docker logs e1
10:57:07.505 [main] INFO io.steeltoe.docker.eurekaserver.EurekaServer -- Spring
Cloud Netflix Eureka Server 4.1.0 by Pivotal Software, Inc.

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)
```

- Remove any stopped/unused containers or dangling images (images with version number as <none> which gets created while rebuilding an application) so that no name clash or port mismatch occurs – docker system prune

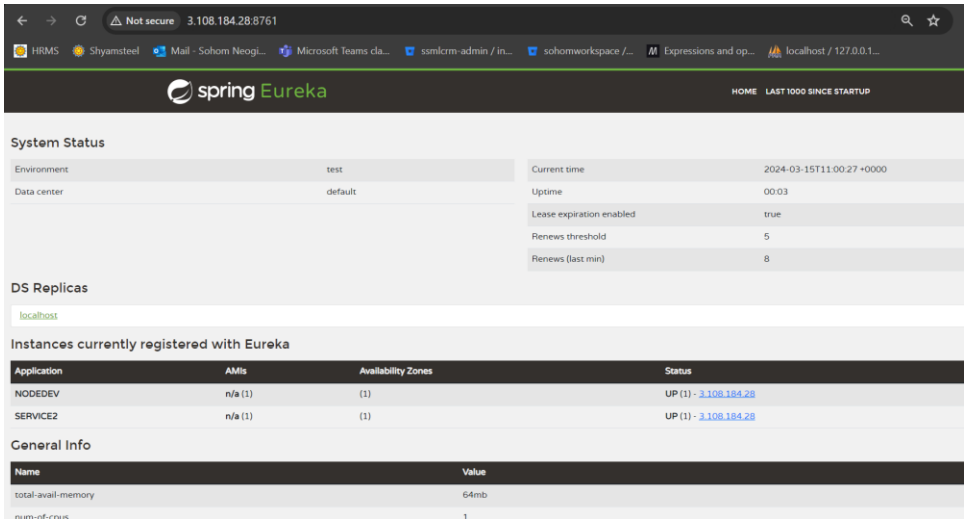
```
[ec2-user@ip-172-31-45-192 ~]$ docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
b68d5d28e941a0e1de3ca5f0613005a47706e26afac57bcf41f6b3d171974abc
eba67b848f1e066b1321f4ccc29193d41d28fa088bda7f203cd6e2d473c71b63
8843a0a55c232600ee6f73fe8ed1dc1134af625b0a79789192b1c00a845b8b7b

Deleted Images:
untagged: sohom179/demo-service2@sha256:d87836c892549579ceba9526090cc8bfb2d49717
bc34852b2381183762e4d7fe
deleted: sha256:eba698abc8bf7178430d60dec6736cb22f723bd586d8f3943972b965b9c5f214
deleted: sha256:509233ab577ea385693903b9aa301c926e04412a7ac01e5c7be9bbe078852632
untagged: sohom179/demo-app@sha256:512f67c7203d13a95cd3ce4cd5c49de407c8c569f29ce
04241adf37adc18654b
deleted: sha256:210082f321efb46dd15be5f27ba61f9cc2d12b073ab3042056a3df3edceaacdc
deleted: sha256:2f47d813d3f68c8a3da17a51b5976ea9f169245b8a390ab4211dfa45d1cbaee8
untagged: sohom179/demo-service2@sha256:549f858d32c7f40667339c6f9de4c9ad578f8dc8
7be61cbc3b3e997e4c0da4f7
deleted: sha256:c832dbd80fccca7f30931f81a5bd1d2f2cdab7216fce0439bf95794c61a1ddd7
deleted: sha256:2649b96c220cc08d56e83740c233a303868e560ef81d5710ee6470e8278ba692
deleted: sha256:0cee5786ceb2e70c71bf6f19d934f186a3131e1325347f8cbee1ffc4d3a8ddfb
deleted: sha256:8e9629e6630e3d9f68e8ede97979de3c49005b19688549b71b659226dcb45e4a
deleted: sha256:92266d73092e0540419c594533e092f30e77c6f90934034073123c9ab50c0a03

Total reclaimed space: 38.33MB
```

- Check in logs – docker logs <container_id>/<name> whether your services and Eureka Server are running properly or not as your services are

```
[ec2-user@ip-172-31-45-192 ~]$ docker logs s1

> nodedev@1.0.0 dev
> nodemon -L server.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server connected in development mode on port 8000
DB Connected!
registered with eureka:  nodedev/3.108.184.28
[ec2-user@ip-172-31-45-192 ~]$ docker logs s2

> service2@1.0.0 dev
> nodemon -L server.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server connected in development mode on port 5000
registered with eureka:  service2/3.108.184.28
[ec2-user@ip-172-31-45-192 ~]$
```

- After service 1 registers with Eureka, you should be able to see it listed in the Eureka server UI. Access the Eureka server UI by navigating to http://<EC2_instance_public_DNS>:8761/ in your browser.

- You should see your service listed under "Applications."



- From there you can redirect to your service url to check whether your services are working properly or not and whether they are getting registered/communicating

Conclusion

This documentation has guided you through deploying a Node.js application using Docker containers on AWS EC2 and registering it with a Eureka service registry. By leveraging Docker, AWS EC2, and Eureka, we've achieved a scalable and resilient architecture for your microservices-based application. This setup ensures seamless communication between services and facilitates easy deployment and management.

Referenced resources (YouTube)

- Docker - Docker Compose **- YT**
- AWS EC2 - Image Deployment **- YT**
- Deploy Node.js Server to AWS EC2 with Docker **- Medium**
- Eureka Service Registry **- YT**
- Setting up Eureka for NodeJS **- Medium**