**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?
2. Why Docker Compose? (Optional)
3. Why AWS EC2?
4. Why Eureka Service Registry?
5. Step-by-Step Deployment:
   1. Setting Up AWS EC2 Instance
   2. Building Docker Images and Pushing to Docker Hub (Done Locally)
   3. Running Docker Containers on EC2
   4. Registering Services with Eureka
6. Conclusion

**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?**

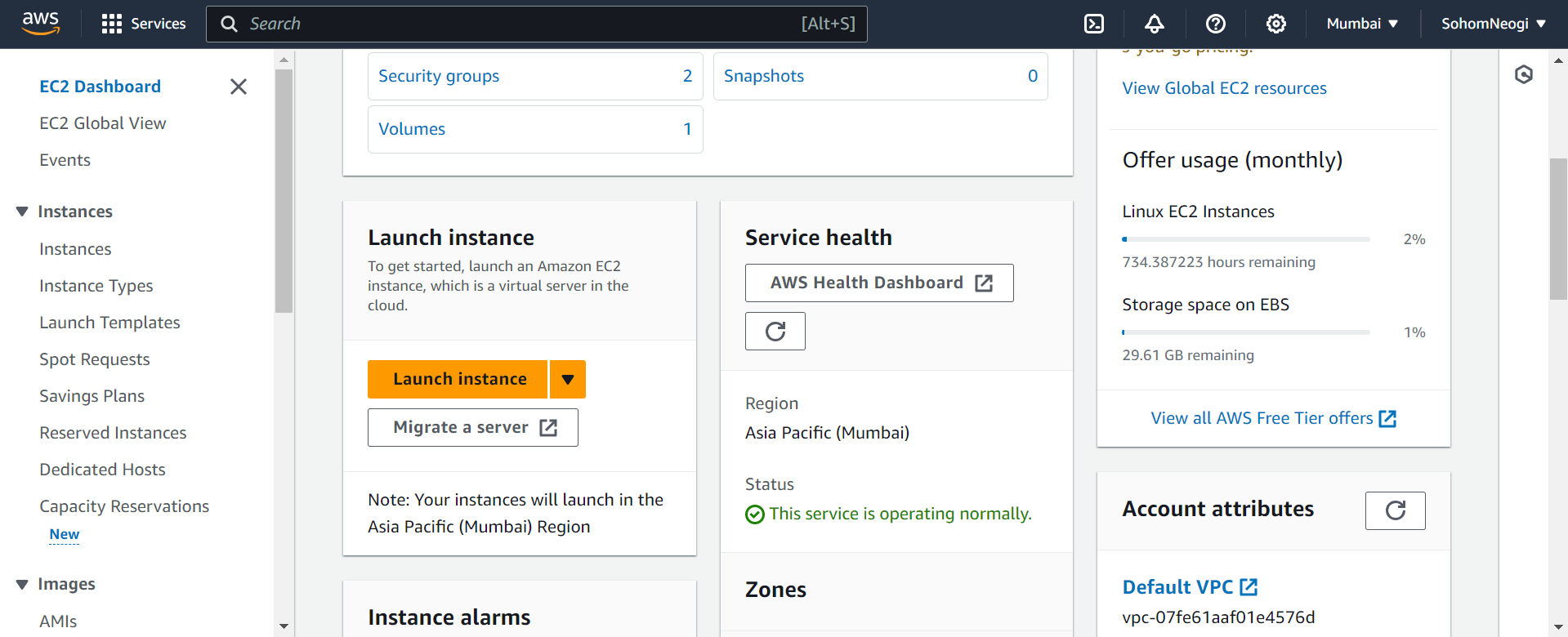
* AWS EC2 (Elastic Compute Cloud) provides scalable compute capacity in the cloud. We'll use EC2 instances to deploy Docker containers and run our Node.js applications. EC2 instances are secure, highly customizable, and scalable, making them ideal for hosting applications in the cloud.

**4. Why Eureka Service Registry?**

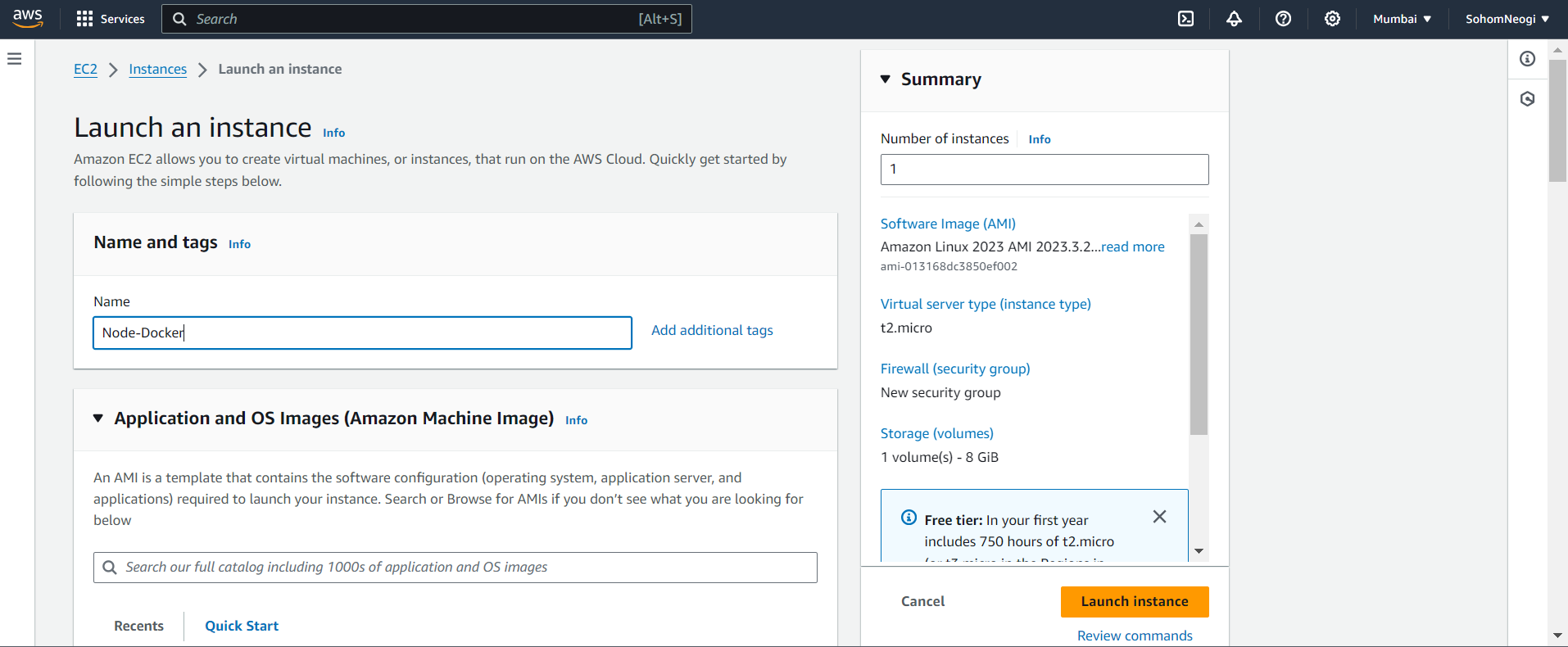
* Eureka, a service registry tool developed by Netflix, empowers microservices to register themselves and discover other services within the ecosystem. By registering services with Eureka, we achieve service discovery and load balancing. This ensures seamless communication between services, even as they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

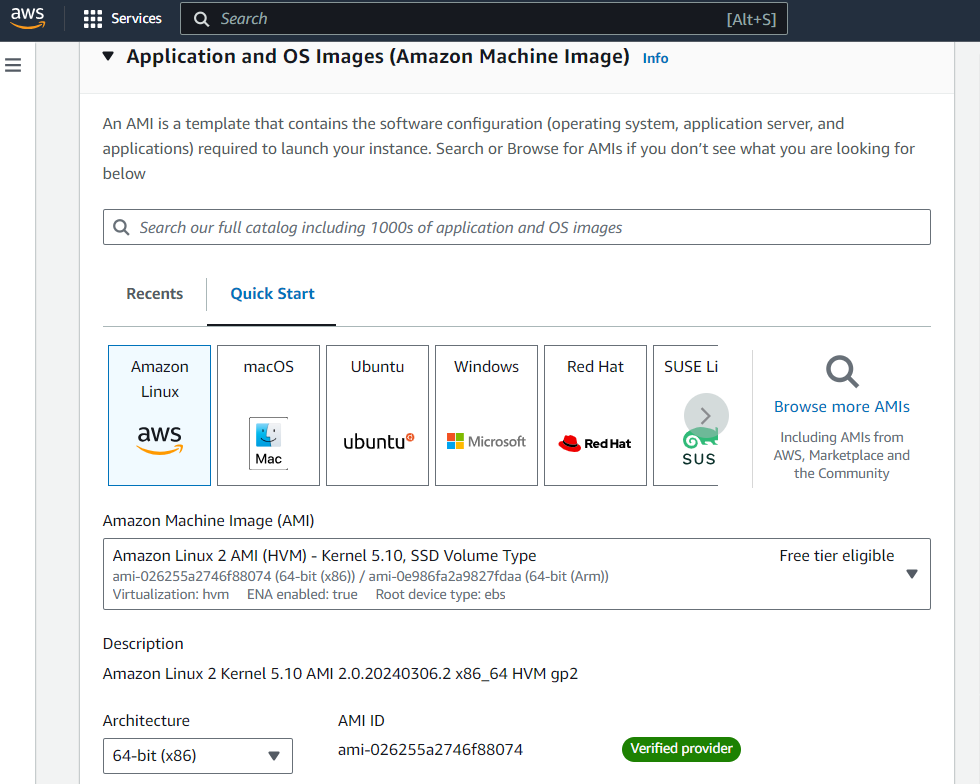
**5. Step-by-Step Deployment:**

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

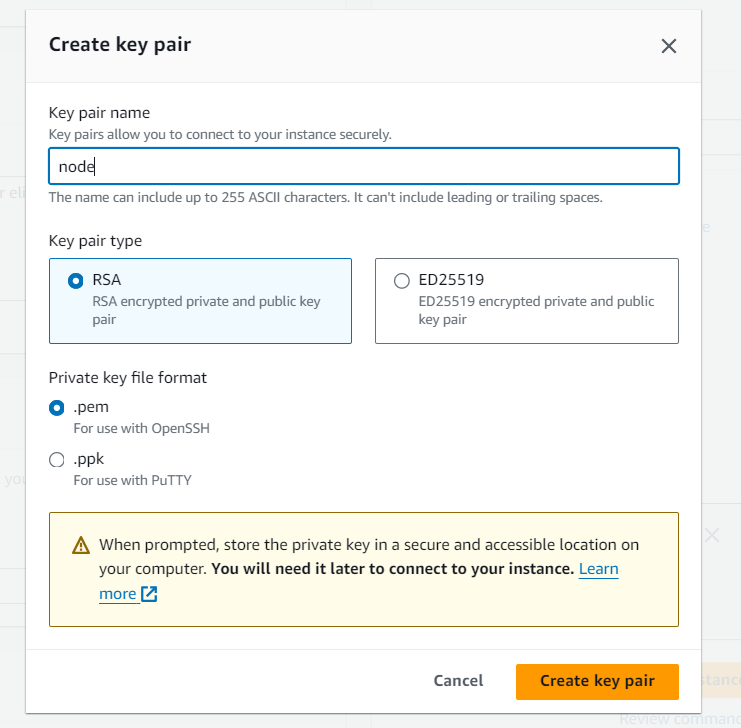


* **Launch a new instance, providing a name and selecting an appropriate Application/OS image (e.g., Ubuntu, Amazon Linux 2).**

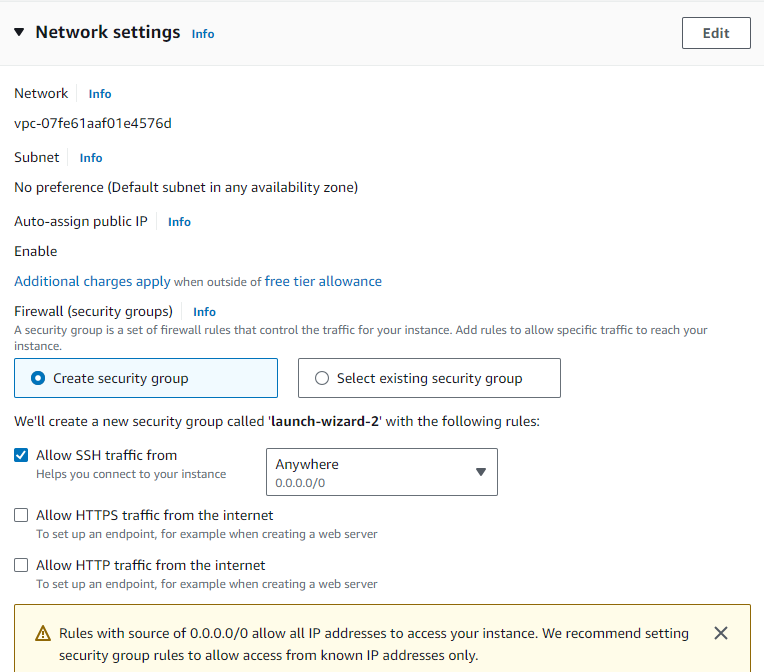




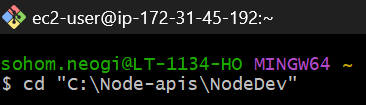
* **Choose an instance type (e.g., "t****2.micro") based on your application's resource requirements.**
* **Create a key pair and download the private key (.pem) file, storing it securely in your application directory.**



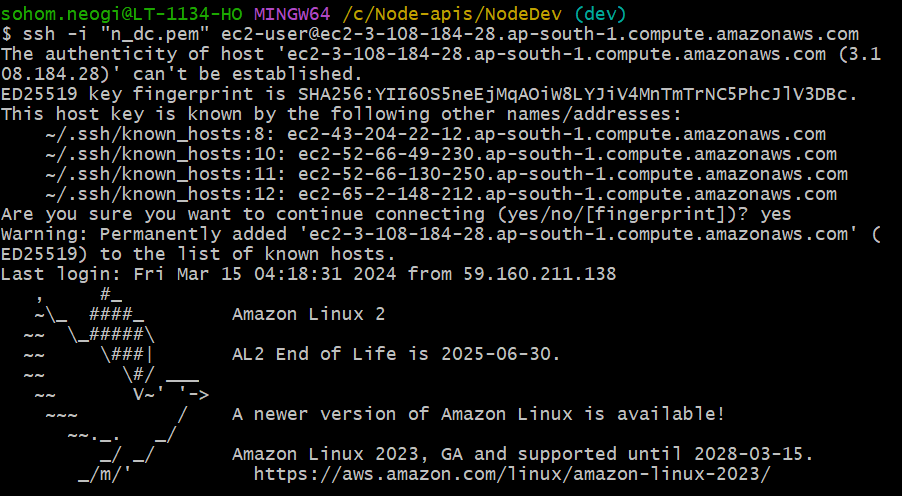
* **Configure security groups to allow inbound traffic for ports used by your services (typically HTTP/HTTPS ports for Node.js applications).**



* **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**



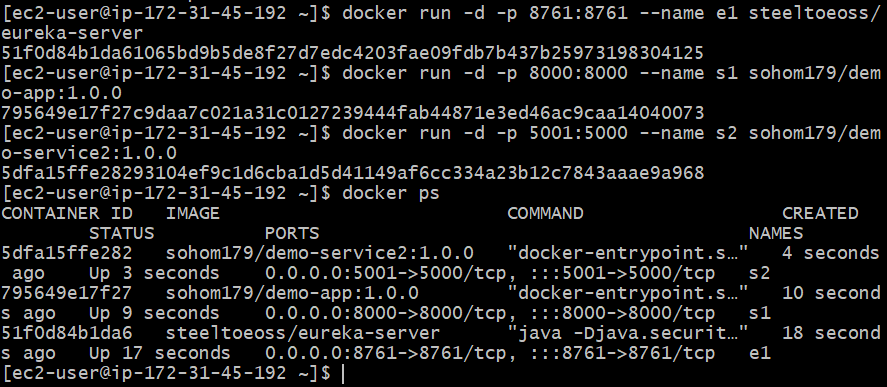
* **Copy the ssh key in your git bash (e.g - “ssh –i “n\_dc.pem....”) for making connection to your instance**



* **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



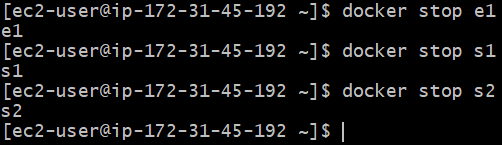
* + - Build an image of the docker file in your local machine terminal – docker image build -t <dockerHub\_username>/<app-name>:<version\_number> -f <file\_name> .
    - Push the latest changes into dockerHub - docker push <dockerHub\_username>/<app-name>:<version\_number>
  + Since my application is 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



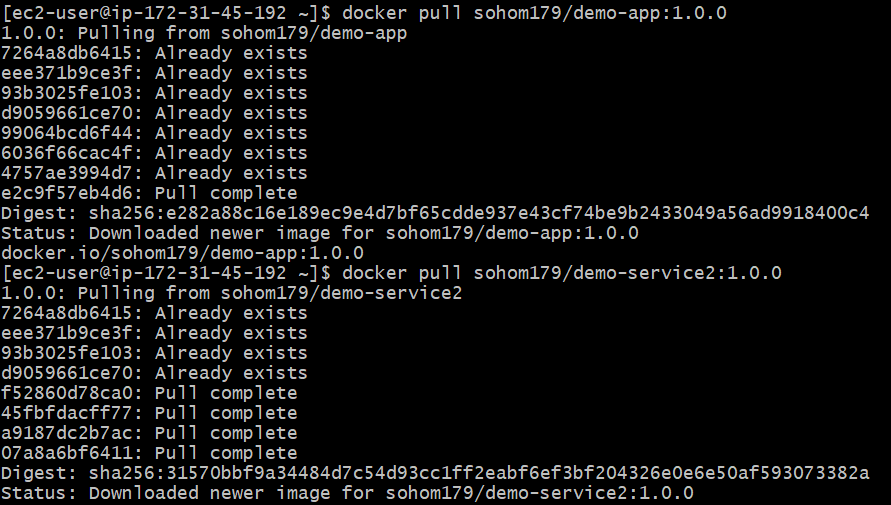
* + Now, install a npm package “npm i eureka-js-client” into your node application and create a file “eureka-client.js” in your current Node application to make connection to the Eureka Server possible

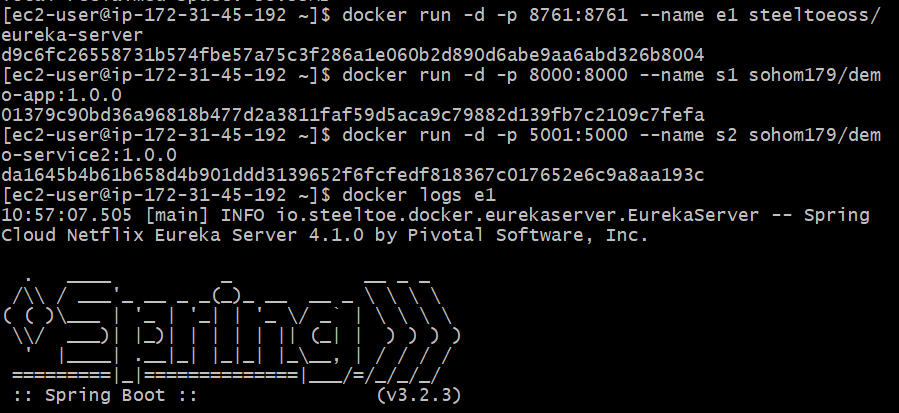


* + Stop the current service – docker container stop <container\_id>/<name>

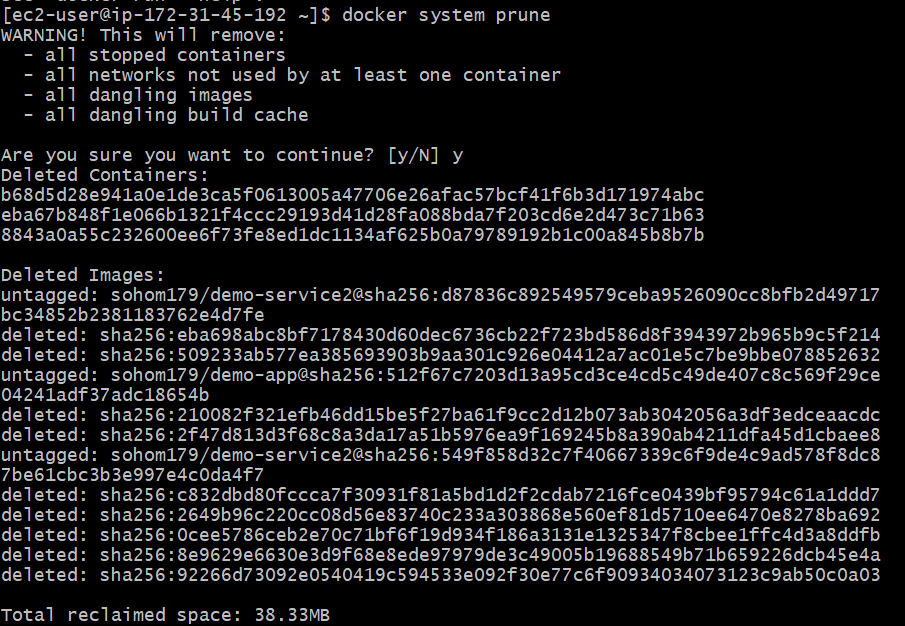


* + Rebuild the image, push the latest build from your local machine and pull the updated service into your EC2 instance and re-run it with the above commands





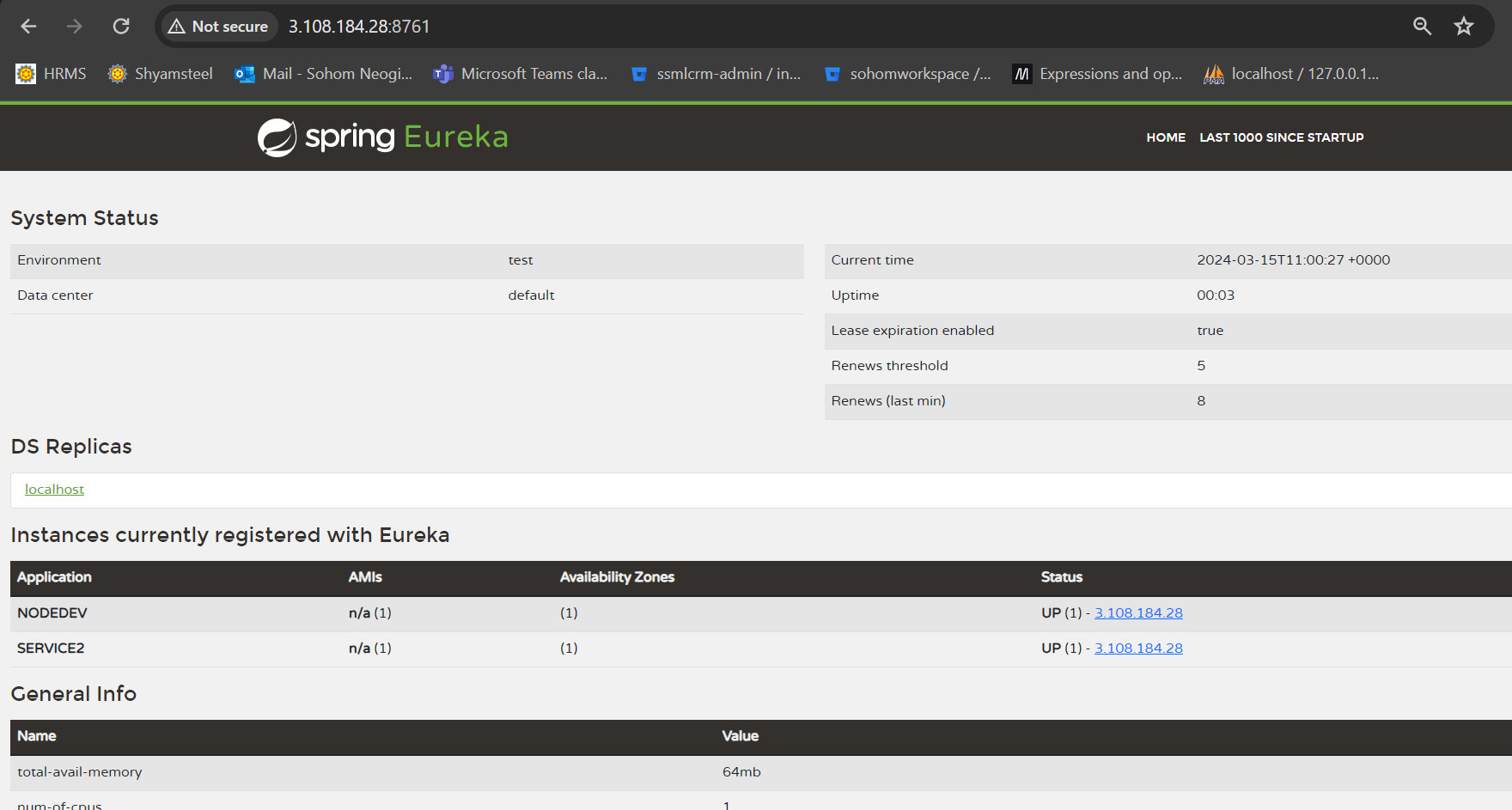
* + Remove any stopped/unused containers or dangling images (images with version number as <none> which gets created while rebuilding an application) so that no name clash or port mismatch occurs – docker system prune



* + Check in logs – docker logs <container\_id>/<name> whether your services and Eureka Server are running properly or not as your services are



* 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**](https://www.youtube.com/watch?v=StXoFPs7Cok) **- YT**
* [**AWS EC2 - Image Deployment**](https://www.youtube.com/watch?v=9TQOWmOniPs&t=41s)  **- YT**
* [**Deploy Node.js Server to AWS EC2 with Docker**](https://aws.plainenglish.io/deploy-node-js-server-to-aws-ec2-with-docker-78687493b53) **- Medium**
* [**Eureka Service Registry**](https://www.youtube.com/watch?v=yKZVdkrTBTg) **- YT**
* [**Setting up Eureka for NodeJS**](https://medium.com/@zilayhuda/register-nodejs-service-with-netflix-eureka-and-use-zuul-for-routing-service-part-1-e50fc49d1219) **- Medium**