

2. Docker

- Create Docker file(s)
- Run app using Docker containers
- Expose required ports
- Ensure containers auto-start on reboot

Steps:

Step 1: Launching a ec2 instance. Task-2

The screenshot shows the AWS EC2 Instances page. A green success message at the top left says "Successfully initiated stopping of i-08f48e80bd1d38d2a". Below it, the "Instances (1/1) Info" section displays a table with one row. The row for "task-2" is selected, showing details like Instance ID (i-0977102f84ad5cc28), Instance state (Running), Instance type (t2.micro), Status check (2/2 checks passed), and Public IP (ec2-13-203-74-159.ap-south-1.compute.amazonaws.com). The "Details" tab is selected in the instance summary panel.

Creating Database using RDS:

The screenshot shows the "Create database" page under the "Aurora and RDS" section. The "Choose a database creation method" section contains two options: "Full configuration" (selected) and "Easy create". The "Engine options" section lists various database engines with their icons: Aurora (MySQL Compatible), Aurora (PostgreSQL Compatible), MySQL, PostgreSQL, MariaDB (selected), Oracle, Microsoft SQL Server, IBM Db2, and IBM Db2. The "MariaDB" option is highlighted with a blue border.

Step 2:Installing Docker on task-2 instance

```
sudo yum install docker -y
```

```
[ec2-user@ip-172-31-1-172 ~]$ sudo yum install docker -y
Last metadata expiration check: 4:16:16 ago on Sat Feb 7 07:28:10 2026.
Dependencies resolved.

=====
| Package           | Architecture | Version      | Repository | Size |
|=====             |             |             |            |       |
| Installing:      |             |             |            |       |
| docker            | x86_64      | 25.0.14-1.amzn2023.0.1 | amazonlinux | 46 M  |
|=====             |             |             |            |       |
| Installing dependencies: |             |             |            |       |
| container-selinux | noarch      | 4:2.242.0-1.amzn2023 | amazonlinux | 58 k  |
| containerd        | x86_64      | 2.1.5-1.amzn2023.0.4 | amazonlinux | 23 M  |
| iptables-libs    | x86_64      | 1.8.8-3.amzn2023.0.2 | amazonlinux | 401 k |
| iptables-nft     | x86_64      | 1.8.8-3.amzn2023.0.2 | amazonlinux | 183 k |
| libcgroup         | x86_64      | 3.0-1.amzn2023.0.1   | amazonlinux | 75 k  |
| libnetfilter_conntrack | x86_64 | 1.0.8-2.amzn2023.0.2 | amazonlinux | 58 k  |
| libnftnl          | x86_64      | 1.0.1-19.amzn2023.0.2 | amazonlinux | 30 k  |
| libnftnl          | x86_64      | 1.2.2-2.amzn2023.0.2 | amazonlinux | 84 k  |
| pigz              | x86_64      | 2.5-1.amzn2023.0.3   | amazonlinux | 83 k  |
| runc              | x86_64      | 1.3.4-1.amzn2023.0.1 | amazonlinux | 3.9 M |

Transaction Summary

Install 11 Packages

Total download size: 74 M
Installed size: 281 M
Downloading Packages:
(1/11): container-selinux-2.242.0-1.amzn2023.noarch.rpm 1.1 MB/s | 58 kB 00:00
(2/11): iptables-libs-1.8.8-3.amzn2023.0.2.x86_64.rpm 11 MB/s | 401 kB 00:00
(3/11): iptables-nft-1.8.8-3.amzn2023.0.2.x86_64.rpm 4.5 MB/s | 183 kB 00:00
(4/11): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm 2.3 MB/s | 75 kB 00:00
(5/11): libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64.rpm 1.9 MB/s | 58 kB 00:00
(6/11): libnftnl-1.0.1-19.amzn2023.0.2.x86_64.rpm 935 kB/s | 30 kB 00:00

[ec2-user@ip-172-31-1-172 ~]$
```

After installing docker start docker using command:

```
sudo systemctl start docker
```

After starting docker enable docker using command:

```
sudo systemctl enable docker
```

After that verify docker:

```
docker --version
```

```
[ec2-user@ip-172-31-1-172 ~]$ docker --version
Docker version 25.0.14, build 0bab007
[ec2-user@ip-172-31-1-172 ~]$
```

Step 4 : Then starting creating our project by creating app directory.

Create Project Directory

```
mkdir app
```

```
cd app
```

Create package.json

```
npm init -y
```

Install Required Packages

```
npm install express mysql2 body-parser
```

```
Aurora and RDS | ap-sout Instances | EC2 | ap-south EC2 Instance Connect + - □ ×
< > C 🌐 ap-south-1.console.aws.amazon.com 🔍 | ⚡ | ⚡ Jagdish Desai (8143-3738-... Jagdish Desai
aws | 📂 | 🔎 | 🗃 | 3 | ⓘ | ⚙️ | Asia Pacific (Mumbai) ▾
S3 IAM EC2 Simple Notification Service VPC Aurora and RDS

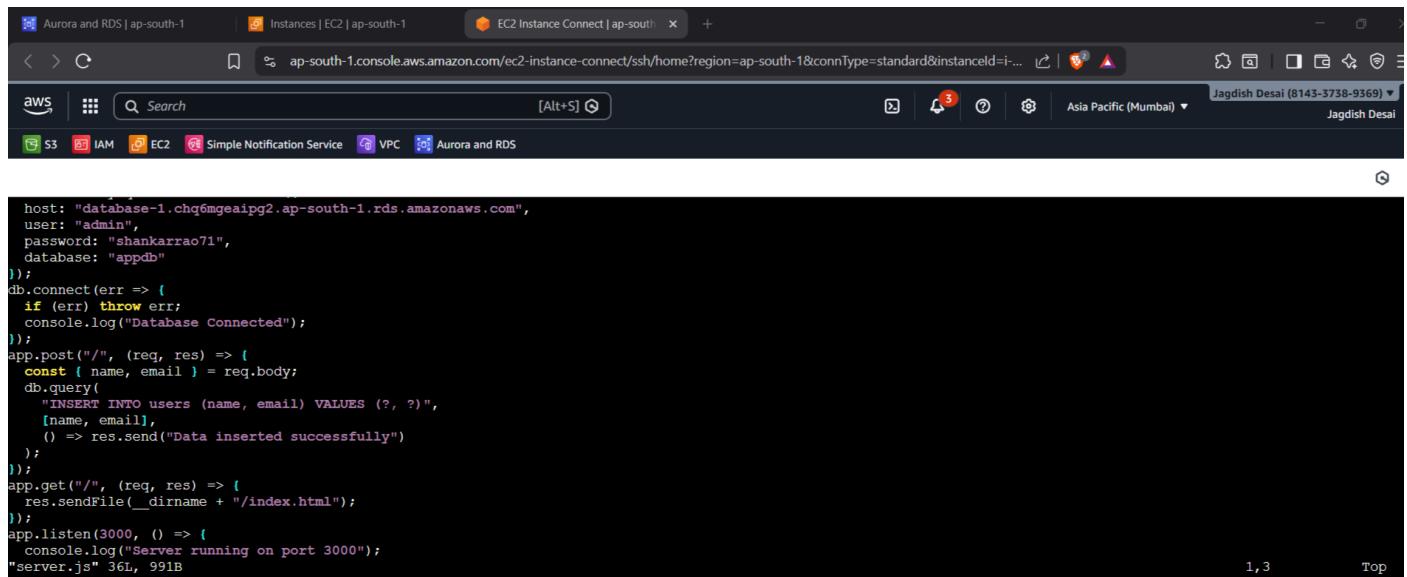
[ec2-user@ip-172-31-1-172 ~]$ docker --version
Docker version 25.0.14, build 0bab007
[ec2-user@ip-172-31-1-172 ~]$ ls
[ec2-user@ip-172-31-1-172 ~]$ mkdir app
[ec2-user@ip-172-31-1-172 ~]$ cd app
[ec2-user@ip-172-31-1-172 app]$ npm init -y
Wrote to /home/ec2-user/app/package.json:

{
  "name": "app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

[ec2-user@ip-172-31-1-172 app]$ npm install express mysql2 body-parser
```

- Create server.js:

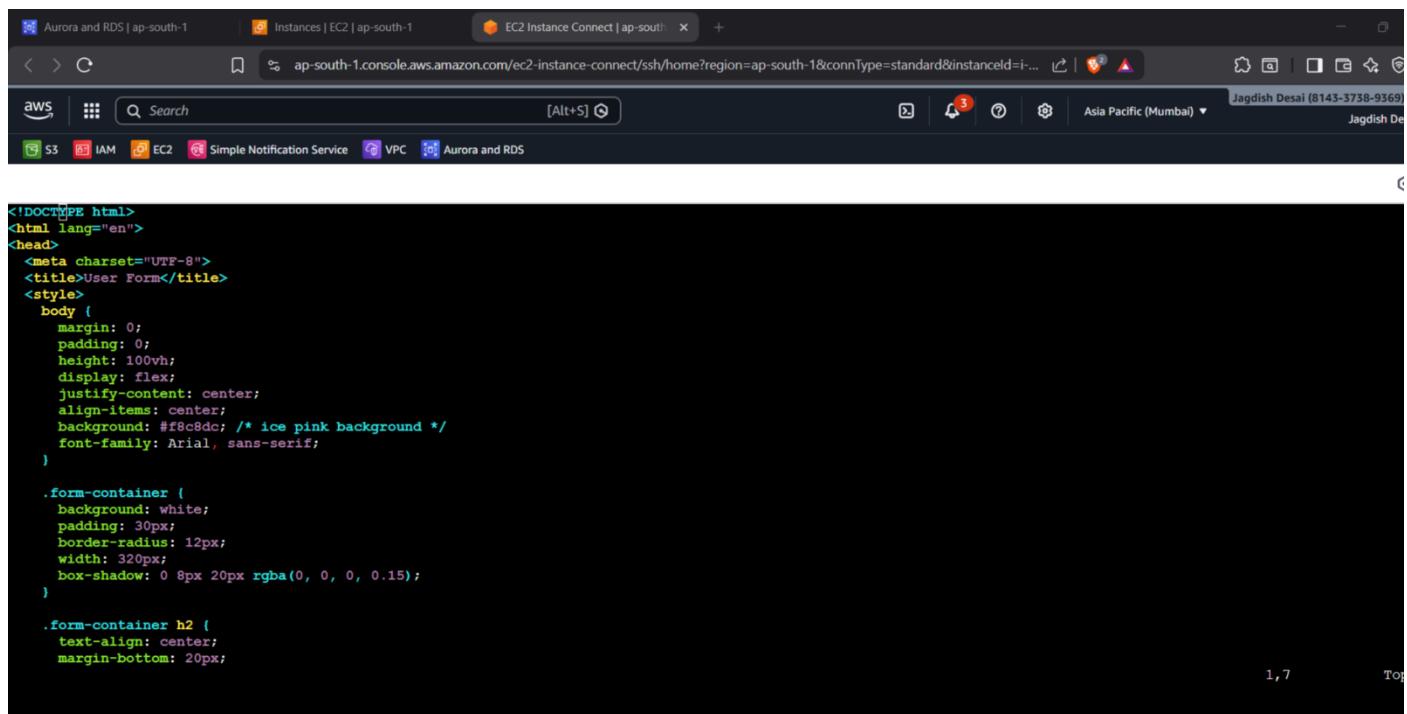
```
vi server.js
```



```
host: "database-1.chq6mgeaipg2.ap-south-1.rds.amazonaws.com",
user: "admin",
password: "shankarrao71",
database: "appdb"
});
db.connect(err => {
if (err) throw err;
console.log("Database Connected");
});
app.post("/", (req, res) => {
const { name, email } = req.body;
db.query(
"INSERT INTO users (name, email) VALUES (?, ?)",
[name, email],
() => res.send("Data inserted successfully")
);
});
app.get("/", (req, res) => {
res.sendFile(__dirname + "/index.html");
});
app.listen(3000, () => {
console.log("Server running on port 3000");
"server.js" 36L, 991B
1, 3 Top
```

- Creating index.html:

```
vi index.html
```



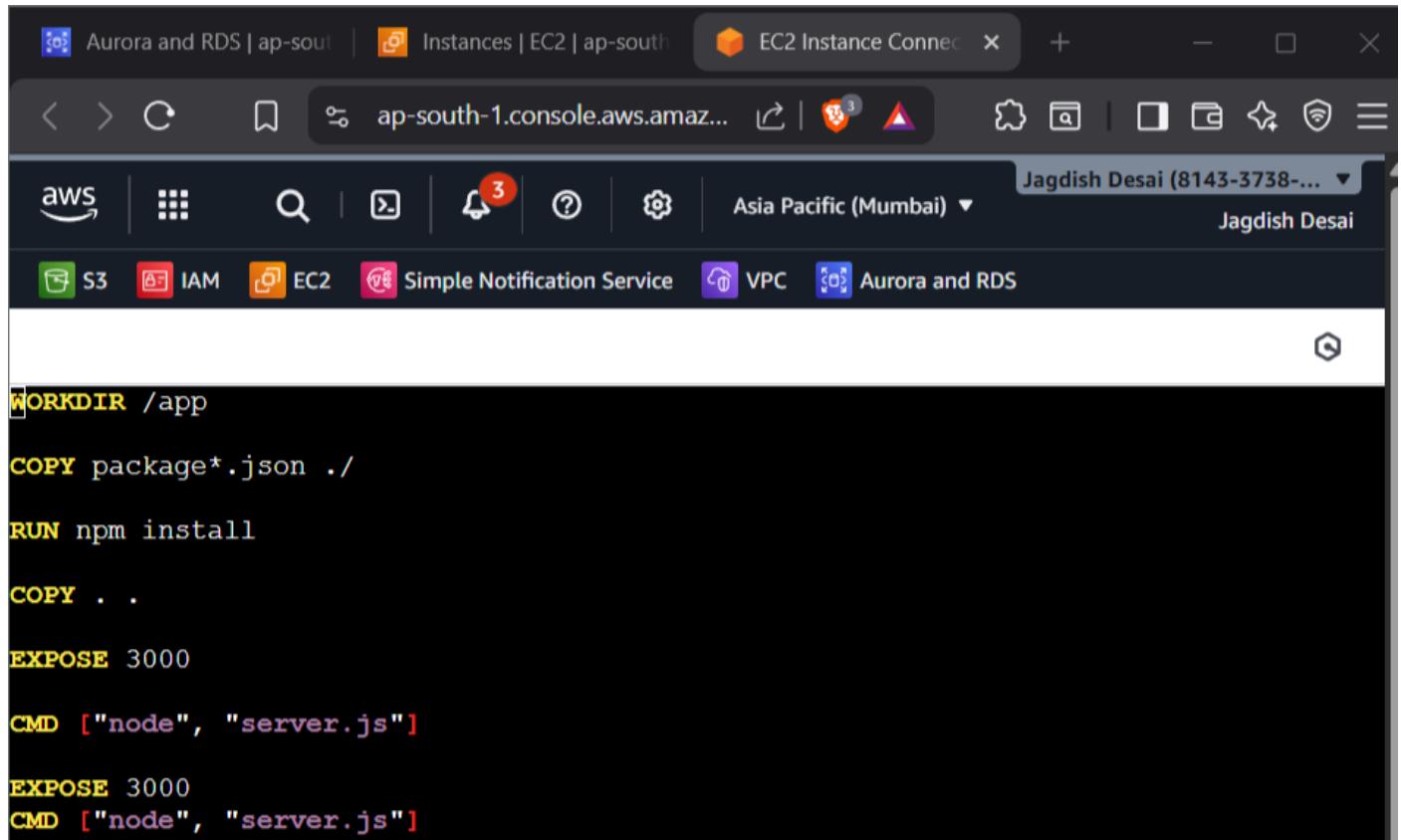
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>User Form</title>
<style>
body {
margin: 0;
padding: 0;
height: 100vh;
display: flex;
justify-content: center;
align-items: center;
background: #f8c8dc; /* ice pink background */
font-family: Arial, sans-serif;
}

.form-container {
background: white;
padding: 30px;
border-radius: 12px;
width: 320px;
box-shadow: 0 8px 20px rgba(0, 0, 0, 0.15);
}

.form-container h2 {
text-align: center;
margin-bottom: 20px;
}
1, 7 Top
```

- **Creating Dockerfile**

```
vi dockerfile
```



The screenshot shows a browser window with the AWS Lambda console. The URL is ap-south-1.console.aws.amazon.com. The top navigation bar includes tabs for Aurora and RDS, Instances | EC2 | ap-south, and EC2 Instance Connect. Below the tabs are standard browser controls like back, forward, and search. The main content area displays a terminal-like interface with a Dockerfile:

```
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
EXPOSE 3000
CMD ["node", "server.js"]
```

Dockerfile does (simple):

- Uses Node.js image
- Copies application files
- Installs dependencies
- Exposes port 3000
- Starts Node.js server

Step 5: Building Docker Image

```
docker build -t node-rds-app .
```

```
aurora and RDS | ap-south-1 Instances | EC2 | ap-south-1 EC2 Instance Connect | ap-south-1 ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=ap-south-1&connType=standard&instanceId=i-... + Jagdish Desai (8143-3758-9569) Asia Pacific (Mumbai) Jagdish Desai aws Search [Alt+S] Ask Amazon Q S3 IAM EC2 Simple Notification Service VPC Aurora and RDS 5.38 => => extracting sha256:3e6b9d1a95114e19f12262a4e8a59ad1d1a10ca7b82108adc0f0605a200294964 5.5s => => sha256:cda7ff44fb2bddcc4bb7514474024b3f3705de00db6355a33beac7808e5b7125 3.32kB / 3.32kB docker:default => => sha256:c6b30c3f16966552af10ac00521f60355b1fcfd46ac1c20b1038587e20583ce7 45.68MB / 45.68MB [+] Building 24.4s (4/9) docker:default => [internal] load build definition from dockerfile 0.0s => transferring dockerfile: 255B 0.0s => [internal] load metadata for docker.io/library/node:18 0.0s => [internal] load .dockerignore 0.0s => transferring context: 2B 0.0s => [1/5] FROM docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 21.6s => => resolve docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 0.0s => => sha256:37927ed901b1b2608b72796c6881bf645480268eca4ac9a37b9219e050bd4b4d8 24.02MB / 24.02MB 1.95 => => sha256:79bf74ad443652b95cc0195ede249df976310efdbbe159f29638783778co 64.40MB / 64.40MB 2.45 => => sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 6.41kB / 6.41kB 0.05 => => sha256:eb29363371lee2259fad6a3c5af80d4abc6ff7d399adbb13b7ed3c1f1l026bd05e0016ba4f09a505f7045ee563 0.05 => => sha256:b50082bc0670d0396bd904e0b5b102655ba5d0e016ba4f09a505f7045ee563 6.39kB / 6.39kB 0.05 => => sha256:3e6b9d1a95114e19f12262a4e8a59ad1d1a10ca7b82108adc0f0605a200294964 48.49MB / 48.49MB 1.48 => => sha256:e2f0999115f5cd2d01418004d3684d34ab 211.36MB / 211.36MB 5.58 => => extracting sha256:3e6b9d1a95114e19f12262a4e8a59ad1d1a10ca7b82108adc0f0605a200294964 5.58 => => sha256:cda7ff44fb2bddcc4bb7514474024b3f3705de00db6355a33beac7808e5b7125 3.32kB / 3.32kB 4.85 => => sha256:c6b30c3f16966552af10ac00521f60355b1fcfd46ac1c20b1038587e20583ce7 45.68MB / 45.68MB [+] Building 24.5s (4/9) docker:default => [internal] load build definition from dockerfile 0.05 => transferring dockerfile: 255B 0.08 => [internal] load metadata for docker.io/library/node:18 2.75 => [internal] load .dockerignore 0.05 => transferring context: 2B 0.05 => [1/5] FROM docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 21.7s => => resolve docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 0.05 => => sha256:37927ed901b1b2608b72796c6881bf645480268eca4ac9a37b9219e050bd4b4d8 24.02MB / 24.02MB 1.95 => => sha256:79bf74ad443652b95cc0195ede249df976310efdbbe159f29638783778co 64.40MB / 64.40MB 2.45
```

Check image:

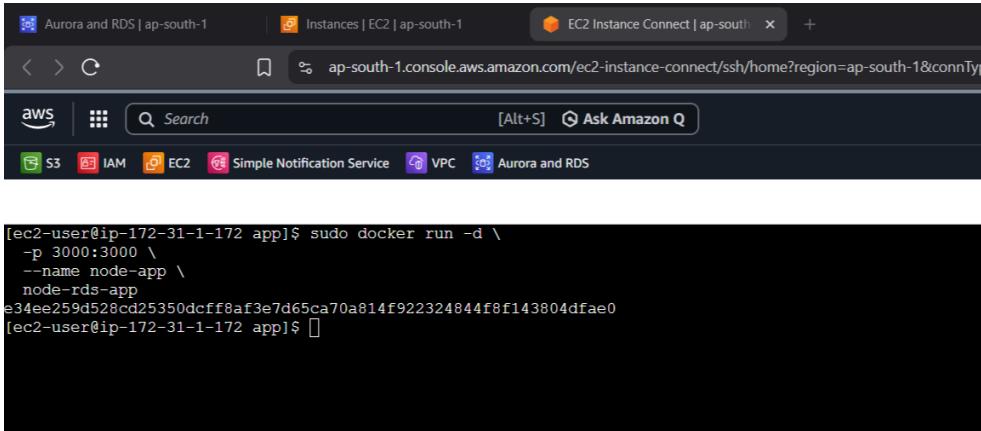
docker images

The screenshot shows the AWS Lambda function configuration page. The top navigation bar includes links for Aurora and RDS, Instances | EC2, and EC2 Instance Connect. The main content area displays the function details:

- Function name:** HelloWorld
- Description:** A simple Lambda function that prints "Hello World" to the CloudWatch logs.
- Runtime:** Python 3.8
- Memory:** 128 MB
- Timeout:** 3 seconds
- Role:** Lambda execution role - `lambda-role`
- Code:** A Python file named `lambda_function.py` containing the following code:

```
[ec2-user@ip-172-31-1-172 app]$ sudo docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
node-rds-app    latest   234cc5e9e12c  3 minutes ago  1.1GB
[ec2-user@ip-172-31-1-172 app]$
```

• Running Application Using Docker Container



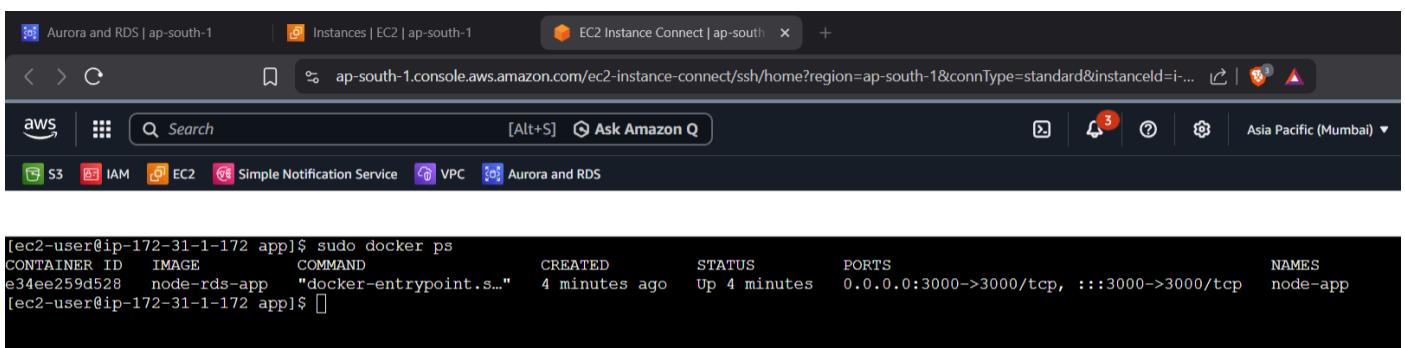
A screenshot of the AWS EC2 Instance Connect interface. The top navigation bar shows 'Aurora and RDS | ap-south-1', 'Instances | EC2 | ap-south-1', and 'EC2 Instance Connect | ap-south-1'. The main area is a terminal window with the following command and output:

```
[ec2-user@ip-172-31-1-172 app]$ sudo docker run -d \
-p 3000:3000 \
--name node-app \
node-rds-app
e34ee259d528cd25350dcff8af3e7d65ca70a814f922324844f8f143804dfaef
[ec2-user@ip-172-31-1-172 app]$
```

- -d → run in background
- -p 3000:3000 → expose port
- --name → container name

• Verifying Container is Running

docker ps

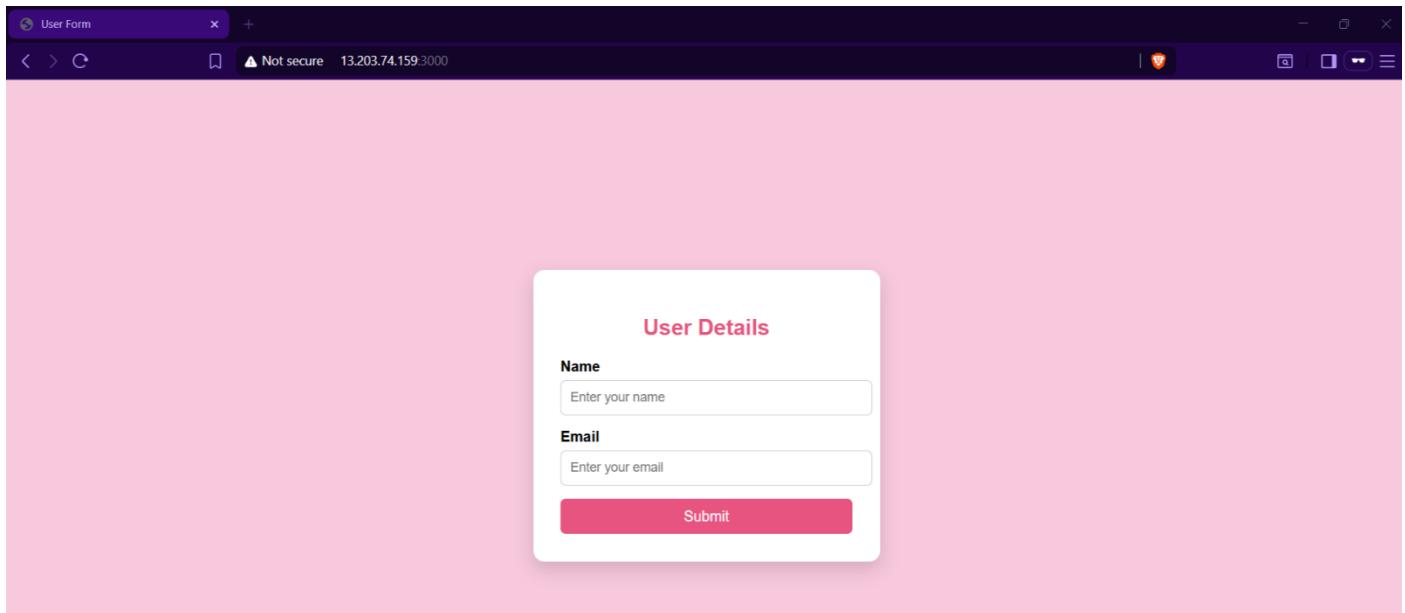


A screenshot of the AWS EC2 Instance Connect interface. The top navigation bar shows 'Aurora and RDS | ap-south-1', 'Instances | EC2 | ap-south-1', and 'EC2 Instance Connect | ap-south-1'. The main area is a terminal window with the following command and output:

```
[ec2-user@ip-172-31-1-172 app]$ sudo docker ps
CONTAINER ID   IMAGE       COMMAND          CREATED        STATUS         PORTS          NAMES
e34ee259d528   node-rds-app "docker-entrypoint.s..."  4 minutes ago   Up 4 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node-app
[ec2-user@ip-172-31-1-172 app]$
```

Step 6: Access Application using public ip

<http://13.203.74.159:3000/>



Step 7: Start Database on task2 instance

Allow port 3306 mysql port in security groups inbound rules.

systemctl start mariadb.service

sudo systemctl enable mariadb.service

sudo systemctl status mariadb.service

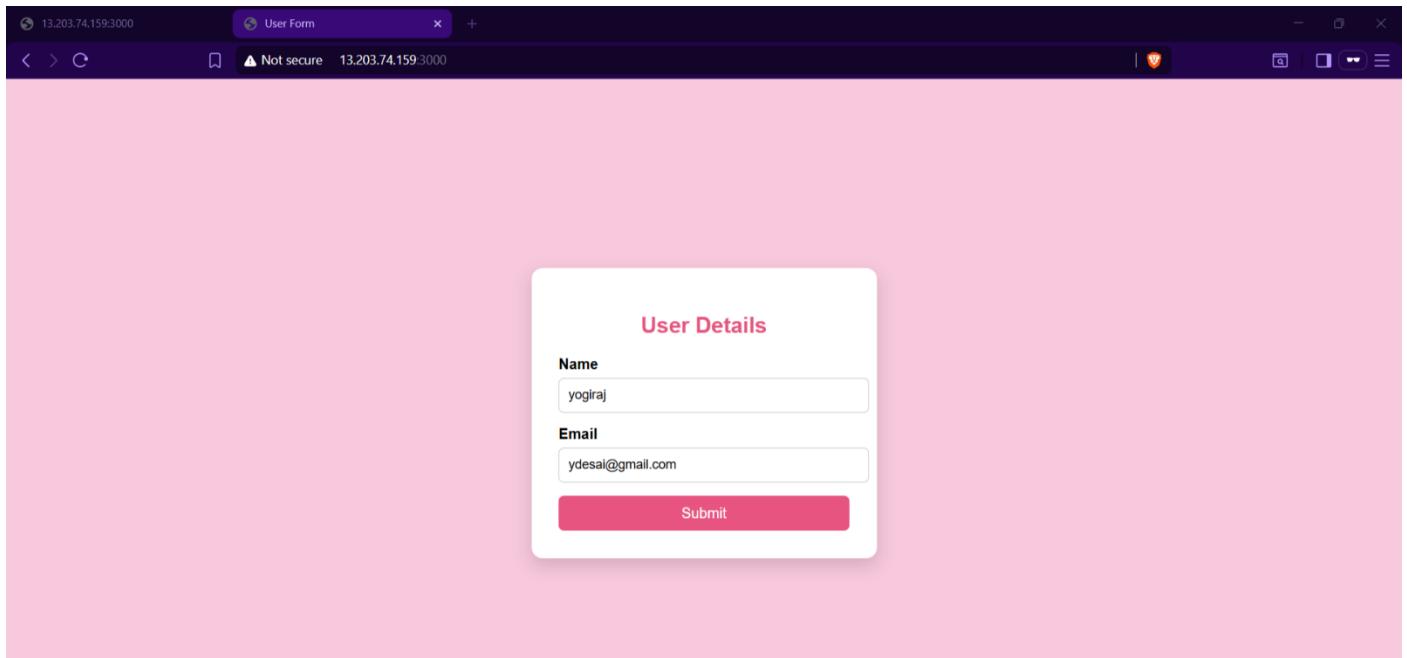
```
[ec2-user@ip-172-31-1-172 app]$ systemctl start mariadb.service
Failed to start mariadb.service: Access denied
See system logs and 'systemctl status mariadb.service' for details.
[ec2-user@ip-172-31-1-172 app]$ sudo !!
sudo systemctl start mariadb.service
[ec2-user@ip-172-31-1-172 app]$ sudo systemctl enable mariadb.service
[ec2-user@ip-172-31-1-172 app]$ sudo systemctl status mariadb.service
● mariadb.service - MariaDB 10.5 database server
  Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; preset: disabled)
  Active: active (running) since Sat 2026-02-07 08:30:53 UTC; 4h 27min ago
    Docs: man:mariadb(8)
          https://mariadb.com/kb/en/library/systemd/
   Main PID: 2060 (mariadb)
     Status: "Taking your SQL requests now..."
      Tasks: 16 (limit: 1120)
    Memory: 65.5M
       CPU: 1.839s
      CGroup: /system.slice/mariadb.service
              └─2060 /usr/libexec/mariadb --basedir=/usr

Feb 07 08:30:52 ip-172-31-1-172.ap-south-1.compute.internal systemd[1]: Starting mariadb.service - MariaDB 10.5 database server...
Feb 07 08:30:52 ip-172-31-1-172.ap-south-1.compute.internal mariadb-prepare-db-dir[2025]: Database MariaDB is probably initialized in /var/lib/mysql already, nothing >
Feb 07 08:30:52 ip-172-31-1-172.ap-south-1.compute.internal mariadb-prepare-db-dir[2025]: If this is not the case, make sure the /var/lib/mysql is empty before running >
Feb 07 08:30:53 ip-172-31-1-172.ap-south-1.compute.internal systemd[1]: Started mariadb.service - MariaDB 10.5 database server.
lines 1-17/17 (END)
```

create table users for storing user details:

```
CREATE TABLE users ( id INT AUTO_INCREMENT PRIMARY KEY,
                     name VARCHAR(100),
                     email VARCHAR(100));
```

Step 8: Submit the above user form



After inserting user details : data goes to our existing MySQL RDS appdb



Using command:

```
mysql -h database-1.chq6mgeaipg2.ap-south-1.rds.amazonaws.com -u admin -p  
Select*from users;
```

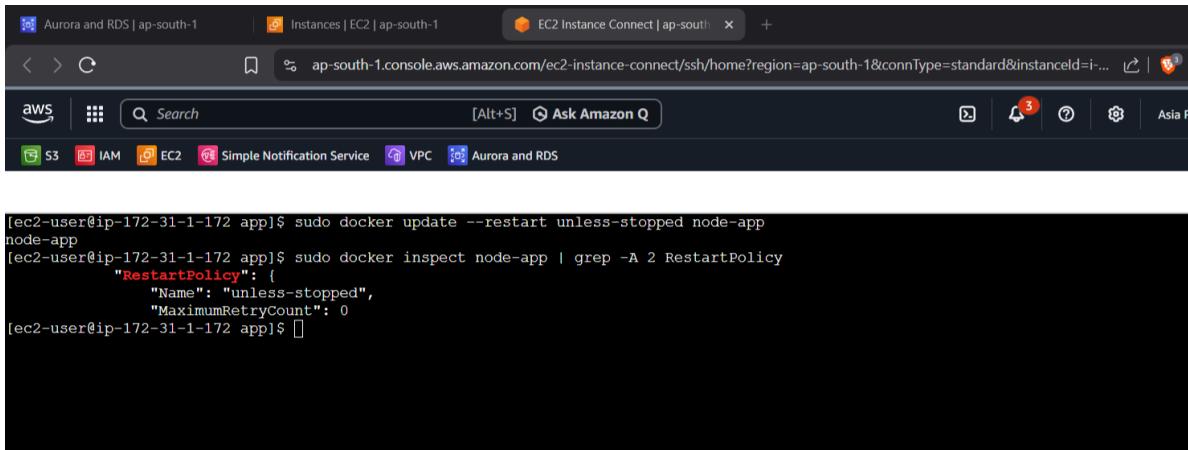
A screenshot of the AWS Management Console. The top navigation bar includes links for Aurora and RDS, Instances, EC2, and EC2 Instance Connect. The search bar contains "Search" and "Ask Amazon Q". The bottom navigation bar has icons for S3, IAM, EC2, Simple Notification Service, VPC, and Aurora and RDS. The main content area is a terminal window showing the output of a MySQL command:

```
MySQL [appdb]> select*from users;  
+----+----+  
| id | name | email |  
+----+----+  
| 1 | yogiraj | ydesai@gmail.com |  
+----+----+  
1 row in set (0.001 sec)
```

Step 9: Ensuring Container Auto-Starts on Reboot

docker update --restart unless-stopped node-app

It tells Docker to automatically restart your container (node-app) whenever the EC2 instance reboots or the container stops unexpectedly, unless you explicitly stop it.



```
[ec2-user@ip-172-31-1-172 app]$ sudo docker update --restart unless-stopped node-app
[ec2-user@ip-172-31-1-172 app]$ sudo docker inspect node-app | grep -A 2 RestartPolicy
  "RestartPolicy": {
    "Name": "unless-stopped",
    "MaximumRetryCount": 0
[ec2-user@ip-172-31-1-172 app]$ 
```

Then;

sudo docker inspect node-app | grep -A 2 RestartPolicy

It lets you **check the restart policy** of your container to confirm whether it will automatically restart after a reboot or on failure.

Container **automatically starts** when:

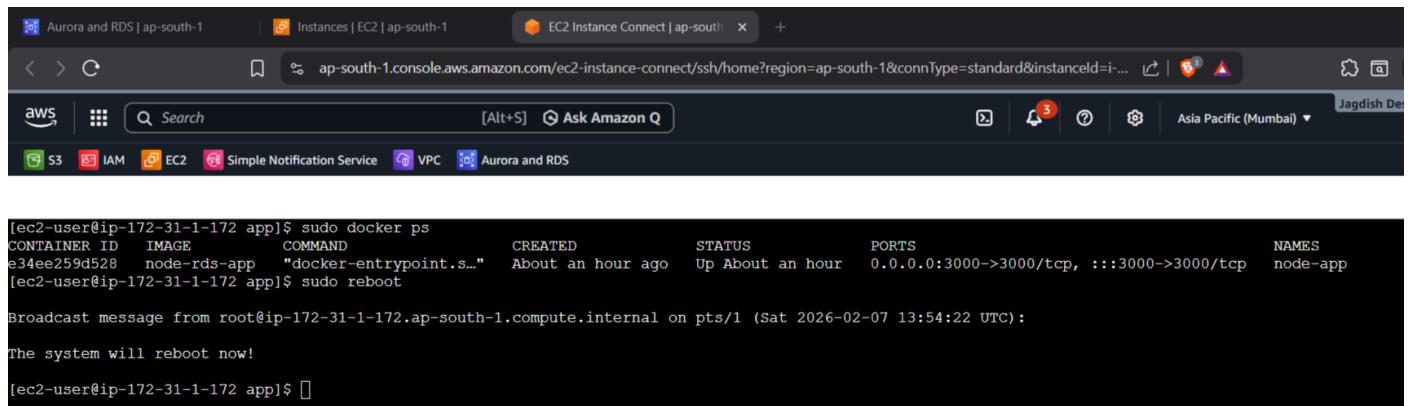
- EC2 reboots
- Docker service restarts

Container will NOT restart only if you manually stop it

Step 10 :Test it by Restarting EC2

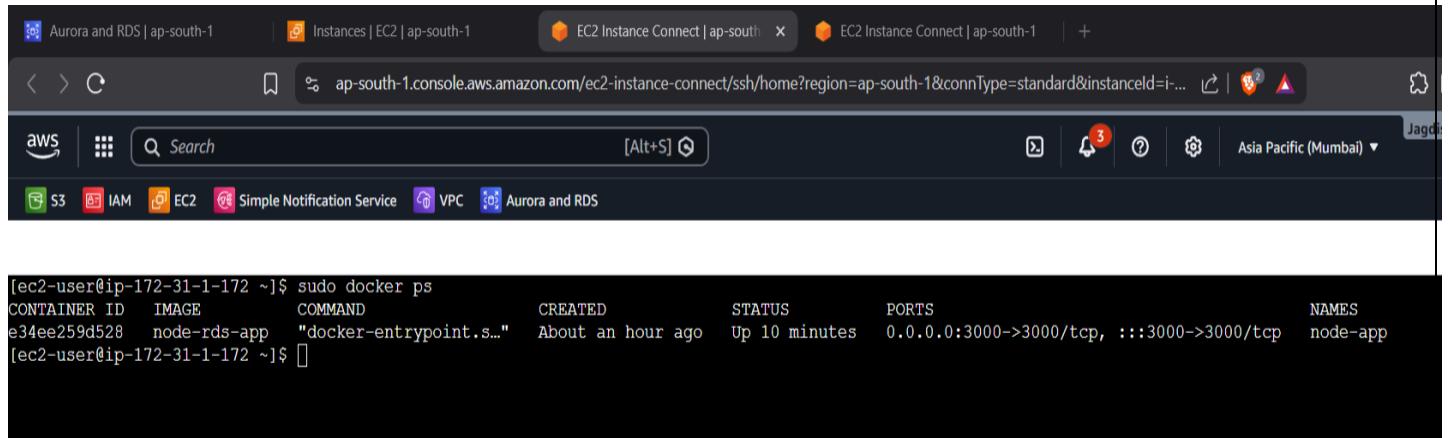
Sudo docker ps

sudo reboot



```
[ec2-user@ip-172-31-1-172 app]$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
e34ee259d528        node-rds-app      "docker-entrypoint.s..."   About an hour ago   Up About an hour   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
[ec2-user@ip-172-31-1-172 app]$ sudo reboot
Broadcast message from root@ip-172-31-1-172.ap-south-1.compute.internal on pts/1 (Sat Feb  7 13:54:22 UTC):
The system will reboot now!
[ec2-user@ip-172-31-1-172 app]$ 
```

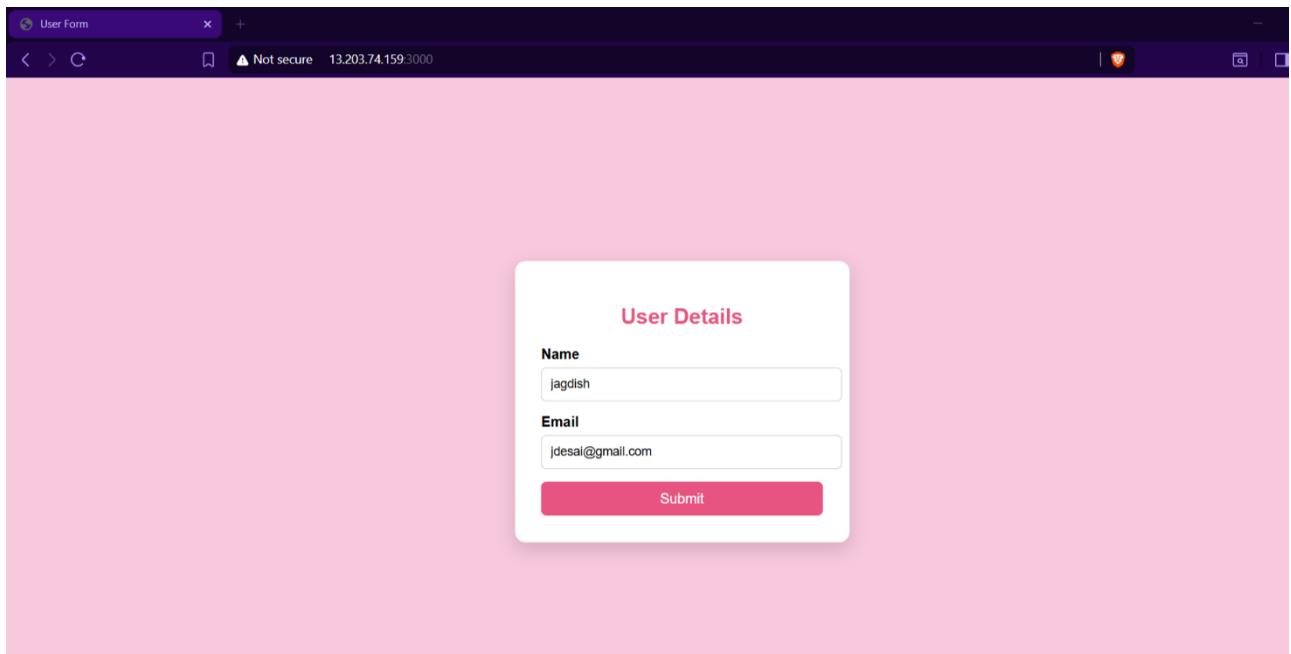
- After Rebooting Reconnect to our EC2 instance:



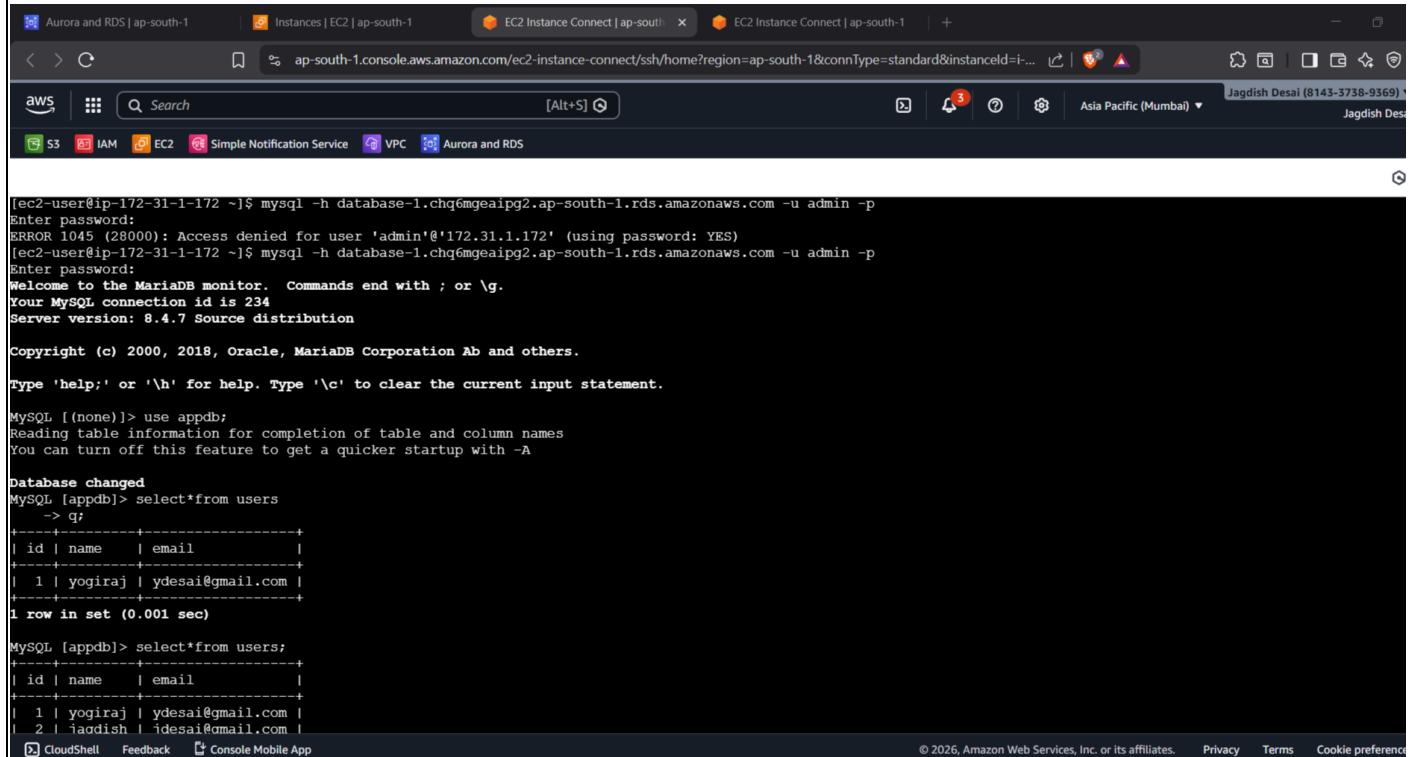
```
[ec2-user@ip-172-31-1-172 ~]$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e34ee259d528 node-rds-app "docker-entrypoint.s..." About an hour ago Up 10 minutes 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp node-app
[ec2-user@ip-172-31-1-172 ~]$
```

see node-app running, just like before, because you set the restart policy to unless-stopped.

- ReTest our app after rebooting:



Then You would see database are also properly working after rebooting.



The screenshot shows a CloudShell terminal window with the following content:

```
[ec2-user@ip-172-31-1-172 ~]$ mysql -h database-1.chq6mgeaipg2.ap-south-1.rds.amazonaws.com -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'172.31.1.172' (using password: YES)
[ec2-user@ip-172-31-1-172 ~]$ mysql -h database-1.chq6mgeaipg2.ap-south-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 234
Server version: 8.4.7 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> use appdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [appdb]> select * from users
-> q;
+----+-----+-----+
| id | name | email |
+----+-----+-----+
| 1  | yogiraj | ydesai@gmail.com |
+----+-----+-----+
1 row in set (0.001 sec)

MySQL [appdb]> select * from users;
+----+-----+-----+
| id | name | email |
+----+-----+-----+
| 1  | yogiraj | ydesai@gmail.com |
| 2  | iaddish | idesai@mail.com |
+----+-----+-----+
```

CloudShell Feedback Console Mobile App © 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

• What should I done in task 2

Docker installed

Dockerfile created

App runs inside container

Port exposed

Auto-start enabled

Summary :

1. Created a Node.js application using nodejs.
2. Created a MySQL database using AWS RDS.
3. Installed Docker on AWS EC2.
4. Prepared the application files for Docker.
5. Created a Dockerfile for the Node.js app.
6. Built a Docker image from the Dockerfile.
7. Ran the application inside a Docker container.
8. Exposed the required port to access the app.
9. Connected the app to MySQL RDS database.
10. Enabled auto-start for the Docker container.
11. Rebooted EC2 and verified the app runs automatically.
12. Accessed the application using EC2 public IP.