

Task3

AWS EC2 Deployment

Launch EC2 (cost-optimized: t2.micro / t3.micro)

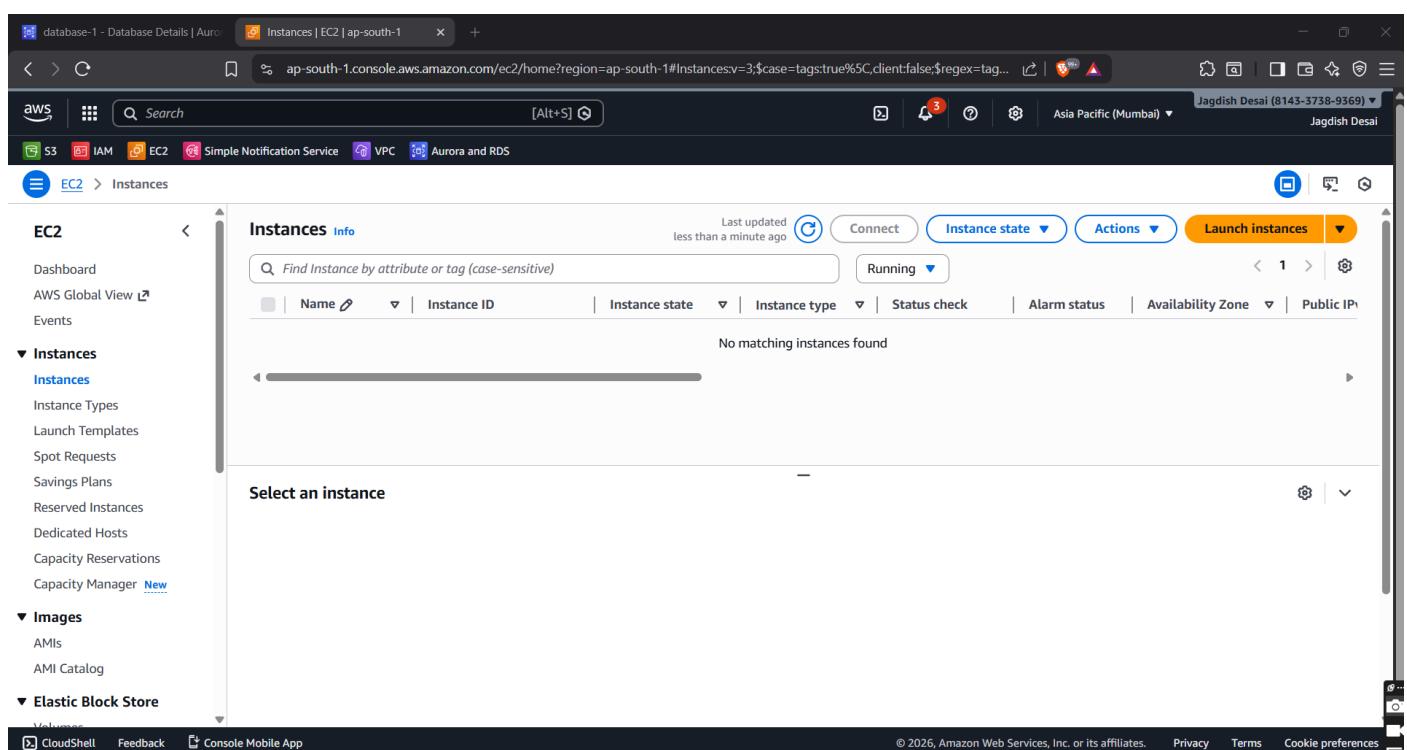
Install Docker

Run containers on EC2

Steps

Step 1: Launch EC2 (t2.micro / t3.micro)

1. Open AWS Console
2. Go to EC2
3. Click on Launch instance



Step 2: Launching instance with task-3

Selecting operating system Amazon Linux

Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags [Info](#)

Name Add additional tags

Application and OS Images (Amazon Machine Image) [Info](#)

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose [Browse more AMIs](#).

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

Recents My AMIs Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux Debian

aws Mac ubuntu Microsoft Red Hat SUSE debian

Browse more AMIs Including AMIs from AWS, Marketplace and the Community

Summary

Number of instances [Info](#) 1

Software Image (AMI) [Amazon Linux 2023 AMI 2023.10....read more](#) ami-0ff50035538b60d5ec

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 of opening an AWS account, you get 760 hours per month of free usage.

Cancel [Launch instance](#) [Preview code](#)

Step 3: Selecting instance type t2.micro

t2.micro is best when we want to learn or run a small app at low cost.

Launch an instance | EC2 | ap-southeast-1 #LaunchInstances:

Instances > Launch an instance

Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand RHEL base pricing: 0.0268 USD per Hour On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0142 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour

All generations

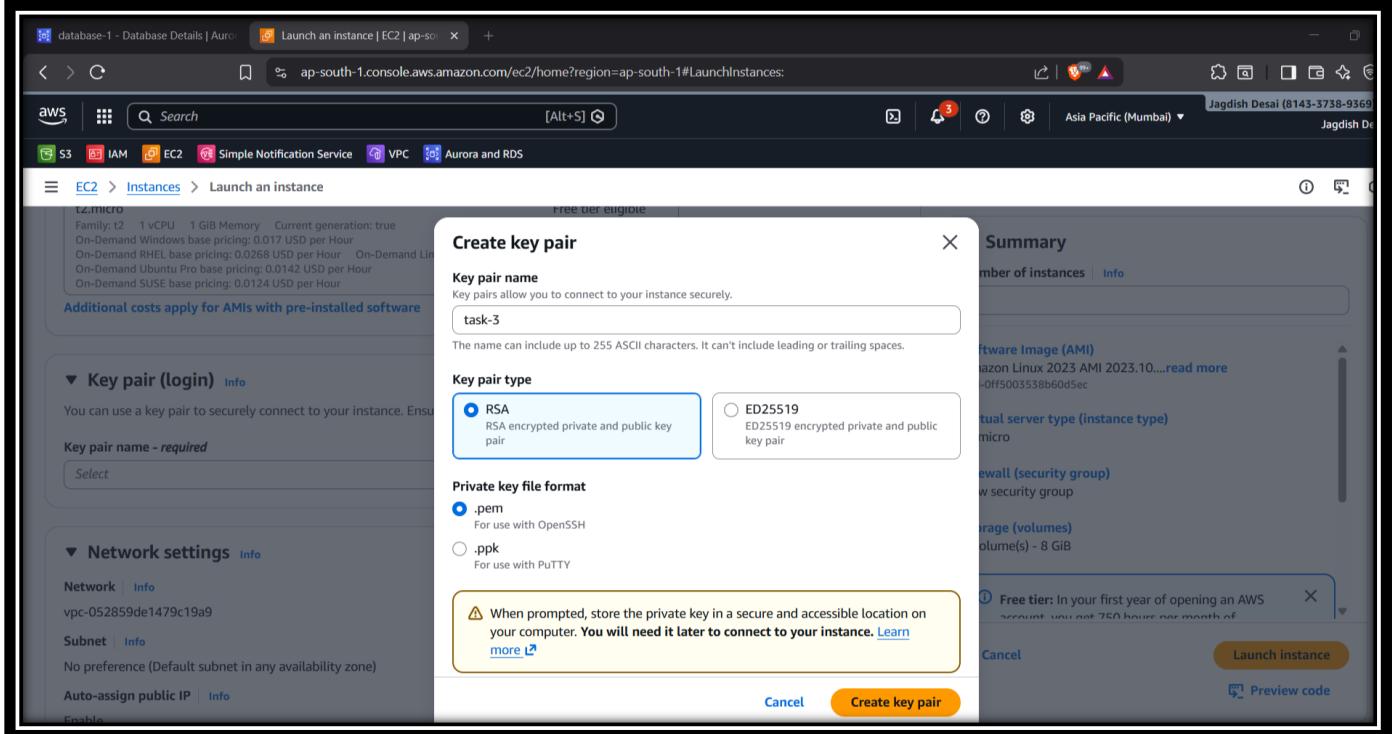
Compare instance types

Additional costs apply for AMIs with pre-installed software

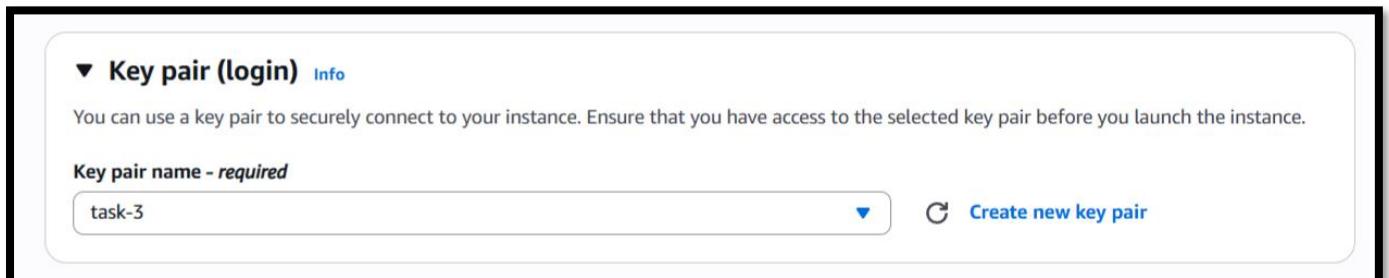
Step 4: Create a key pair or select existing key pair

We can use a key pair to securely connect to our instance.

Select key-pair type and create and download task3.pem file



After Creating key-pair select it:



Step 5: Go to network settings

Select vpc- I choose default vpc // Virtual network

select subnet- I choose subnet of ap-south-1a //Network segment

Auto-assign public IP Info- Enable

Security group- launch-wizard-8 //firewall rules

Allow SSH – 22 (My IP)

Allow HTTP – 80 (Anywhere)

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The 'VPC - required' section is selected, showing a dropdown for 'VPC' set to 'vpc-052859de1479c19a9' (default). Below it, the 'Subnet' dropdown is set to 'subnet-04cff1c02b7d4f454'. The 'Auto-assign public IP' dropdown is set to 'Enable'. The 'Firewall (security groups)' section shows a radio button for 'Create security group' selected. The 'Security group name' field contains 'launch-wizard-8'. The 'Description' field is empty. On the right side, the 'Summary' panel shows 'Number of instances' set to 1. It also lists the 'Software Image (AMI)', 'Virtual server type (instance type)', 'Firewall (security group)', and 'Storage (volumes)'. A tooltip for the 'Free tier' is visible, stating: 'In your first year of opening an AWS account, you get 750 hours per month of free usage.' At the bottom right are 'Cancel' and 'Launch instance' buttons.

Step 6: Configure storage according to requirements I keep default storage.

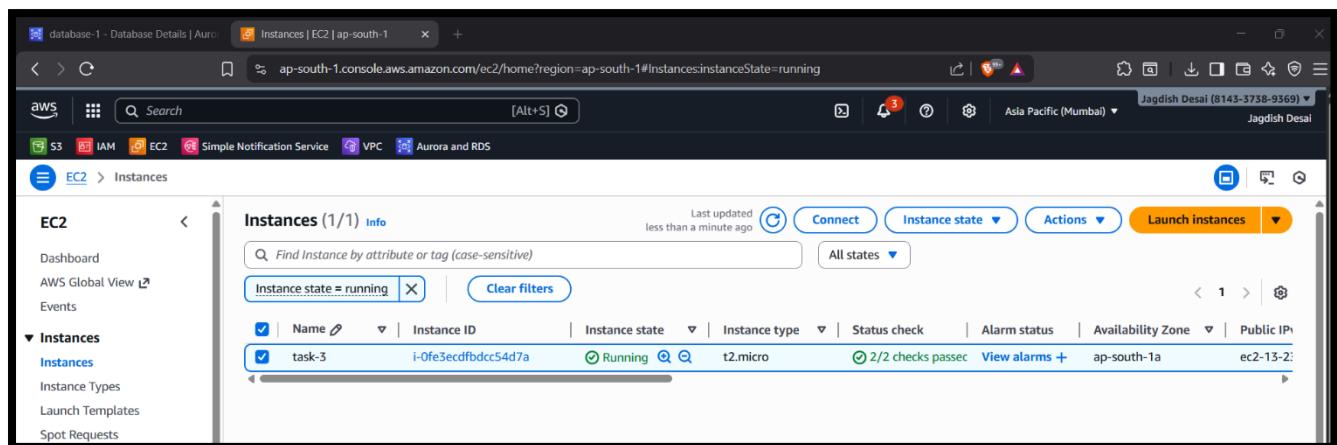
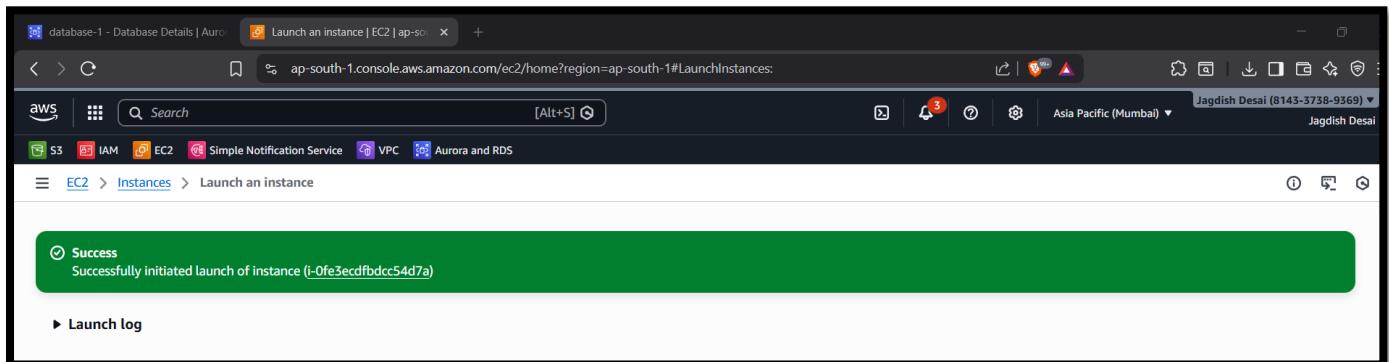
The screenshot shows the 'Configure storage' section of an AWS instance configuration. It displays a single root volume set to 8 GiB with the gp3 type and 3000 IOPS. The volume is noted as 'Not encrypted'. A tooltip indicates that free-tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. There is also a note to click refresh to view backup information. The 'Edit' button is visible at the bottom right.

Step 7: See the overview of our task 3 instance

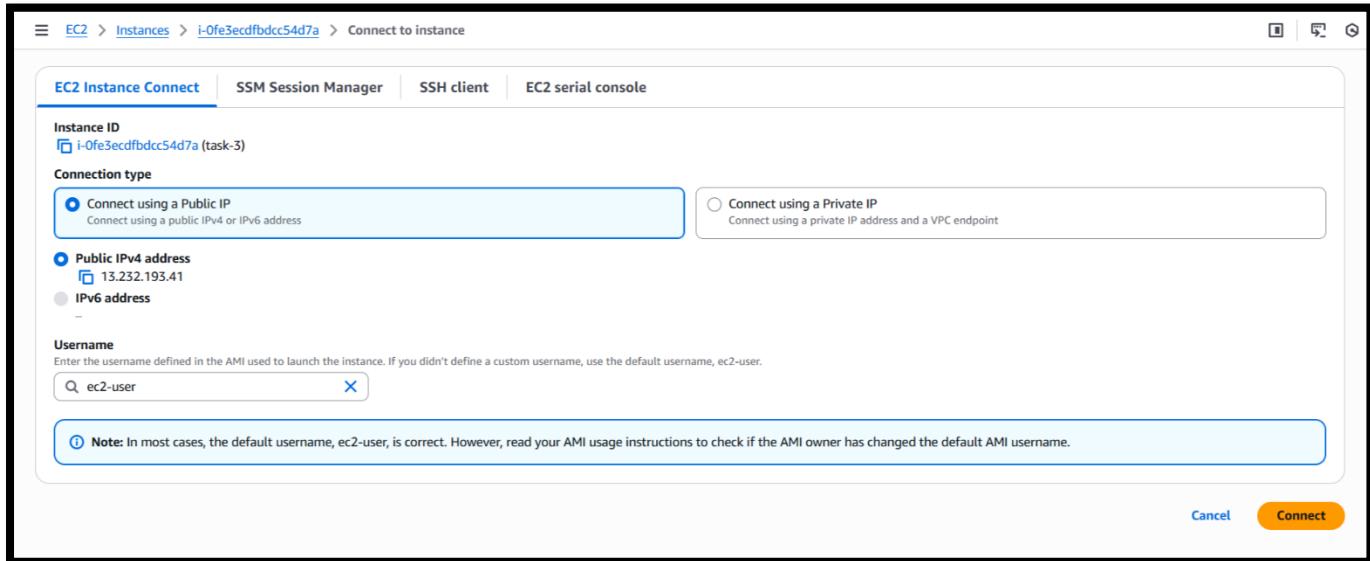
The screenshot shows the 'Summary' section of the instance configuration. It lists the following details: 1 instance, Amazon Linux 2023 AMI 2023.10... (with a 'read more' link), t2.micro instance type, New security group, and 1 volume(s) - 8 GiB. A tooltip provides information about the free tier. At the bottom, there are 'Cancel', 'Launch instance' (in an orange button), and 'Preview code' buttons.

Here we can create number instances according to our requirements.

Step 8: Launch instance



Step 9: Connect Ec2 instance



```
'      #
`~\ _ #####_      Amazon Linux 2023
`~ \#####\
`~ \###|
`~ \#/
`~ V~' '-'>
`~~ /'
`~~ .-./'
`~~ /'-'/
`~~ /m/'  
ec2-user@ip-172-31-40-21 ~]$
```

Installing docker –

Step 1: Install Docker on EC2

```
sudo yum install docker -y  
sudo systemctl start docker
```

```
[ec2-user@ip-172-31-40-21 ~]$ sudo yum install docker -y  
Amazon Linux 2023 Kernel Livepatch repository  
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  
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  
libnftnealink   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  
-----
```

Status: Running

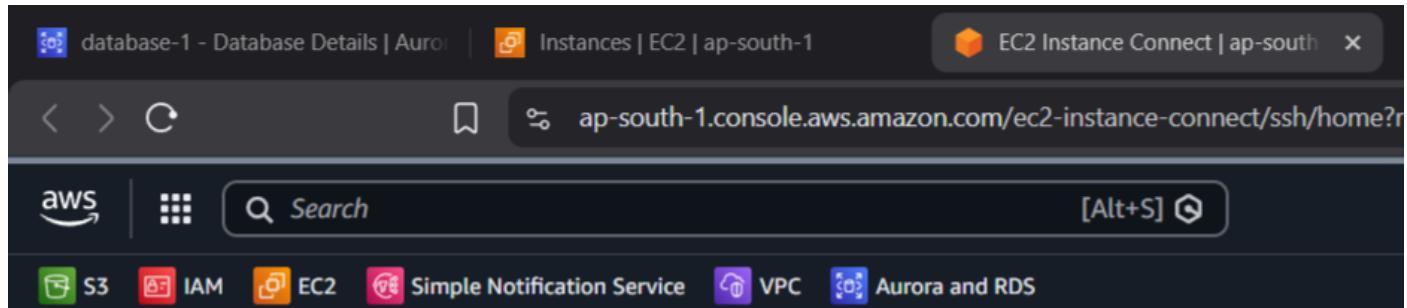
```
[ec2-user@ip-172-31-40-21 ~]$ sudo systemctl start docker  
[ec2-user@ip-172-31-40-21 ~]$ sudo systemctl enable docker  
[ec2-user@ip-172-31-40-21 ~]$ sudo systemctl status docker  
● docker.service - Docker Application Container Engine  
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)  
  Active: active (running) since Sat 2026-02-07 17:49:34 UTC; 1min 52s ago  
TriggeredBy: ● docker.socket  
  Docs: http://dockerd.docker.com  
 Main PID: 30525 (dockerd)  
   Tasks: 7  
    Memory: 28.5M  
     CPU: 328ms  
    CGroup: /system.slice/docker.service  
           └─30525 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536  
Feb 07 17:49:33 ip-172-31-40-21.ap-south-1.compute.internal systemd[1]: Starting docker.service - Docker Application Container Engine...  
Feb 07 17:49:33 ip-172-31-40-21.ap-south-1.compute.internal dockerd[30525]: time="2026-02-07T17:49:33.6243442302" level=info msg="Starting up"  
Feb 07 17:49:33 ip-172-31-40-21.ap-south-1.compute.internal dockerd[30525]: time="2026-02-07T17:49:33.6875882452" level=info msg="Loading containers: start."  
Feb 07 17:49:34 ip-172-31-40-21.ap-south-1.compute.internal dockerd[30525]: time="2026-02-07T17:49:34.1258793132" level=info msg="Loading containers: done."  
Feb 07 17:49:34 ip-172-31-40-21.ap-south-1.compute.internal dockerd[30525]: time="2026-02-07T17:49:34.1513021892" level=info msg="Docker daemon" commit=d334795 contain...  
Feb 07 17:49:34 ip-172-31-40-21.ap-south-1.compute.internal dockerd[30525]: time="2026-02-07T17:49:34.1516739857" level=info msg="Daemon has completed initialization"  
Feb 07 17:49:34 ip-172-31-40-21.ap-south-1.compute.internal dockerd[30525]: time="2026-02-07T17:49:34.1883214372" level=info msg="API listen on /run/docker.sock"  
Feb 07 17:49:34 ip-172-31-40-21.ap-south-1.compute.internal systemd[1]: Started docker.service - Docker Application Container Engine.  
lines 1-20/20 (END)
```

Give permission:

```
sudo usermod -aG docker ec2-user
```

allows ec2-user to run Docker commands without sudo

Step 2: Check Docker version



```
[ec2-user@ip-172-31-40-21 ~]$ docker --version
Docker version 25.0.14, build 0bab007
[ec2-user@ip-172-31-40-21 ~]$ 
```

Step 3: What is happening?

we:

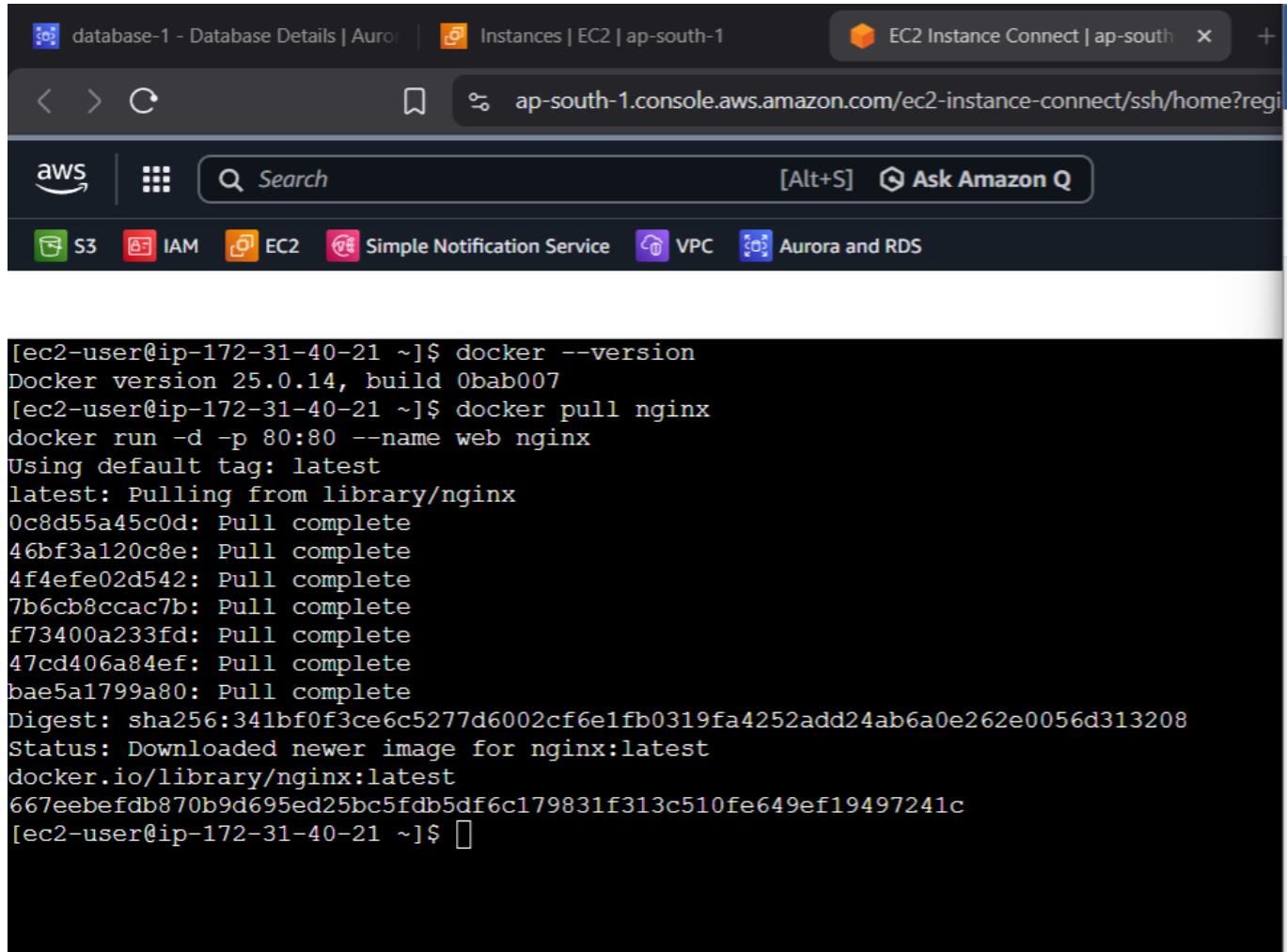
- Using **EC2**
- Running **Docker**
- Pulling **Nginx** from **Docker Hub**
- Hosting a **web server**
- Accessing it using **EC2 Public IP**
-

Step 4: Pull Docker Image

```
docker pull nginx
```

Explanation:

- Downloads the **nginx image** from **Docker Hub**
- Docker Hub = online image store
- Nginx = web server



The screenshot shows a terminal window within the AWS Management Console. The title bar includes tabs for 'Database Details | Aurora', 'Instances | EC2 | ap-south-1', and 'EC2 Instance Connect | ap-south-1'. The main area displays a command-line session:

```
[ec2-user@ip-172-31-40-21 ~]$ docker --version
Docker version 25.0.14, build 0bab007
[ec2-user@ip-172-31-40-21 ~]$ docker pull nginx
docker run -d -p 80:80 --name web nginx
Using default tag: latest
latest: Pulling from library/nginx
0c8d55a45c0d: Pull complete
46bf3a120c8e: Pull complete
4f4efe02d542: Pull complete
7b6cb8ccac7b: Pull complete
f73400a233fd: Pull complete
47cd406a84ef: Pull complete
bae5a1799a80: Pull complete
Digest: sha256:341bf0f3ce6c5277d6002cf6e1fb0319fa4252add24ab6a0e262e0056d313208
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
667eebefdb870b9d695ed25bc5fdb5df6c179831f313c510fe649ef19497241c
[ec2-user@ip-172-31-40-21 ~]$ 
```

Step 3: Run Docker Container

`docker run -d -p 80:80 --name web nginx`

`docker run`

Start a container

`-d`

Run in background (detached mode)

`-p 80:80`

Connect ports

- Left 80 → EC2 port
- Right 80 → container port
- Means: browser traffic goes to container

`--name web`

Container name = web

`nginx`

👉 Image name

The screenshot shows a Lambda function named "HelloWorld" with a successful deployment. The code is a simple "Hello World" function in Python. The deployment ID is "2023-06-14T104544Z-12345678901234567890123456789012". The status bar at the bottom indicates "Success" and "1 second ago".

```
[ec2-user@ip-172-31-40-21 ~]$ docker run -d -p 80:80 --name web nginx  
7cc5c7d0aea2343ad5ca2a59a8454927acd2109697bdc661dd85799f1de2f2f7  
[ec2-user@ip-172-31-40-21 ~]$
```

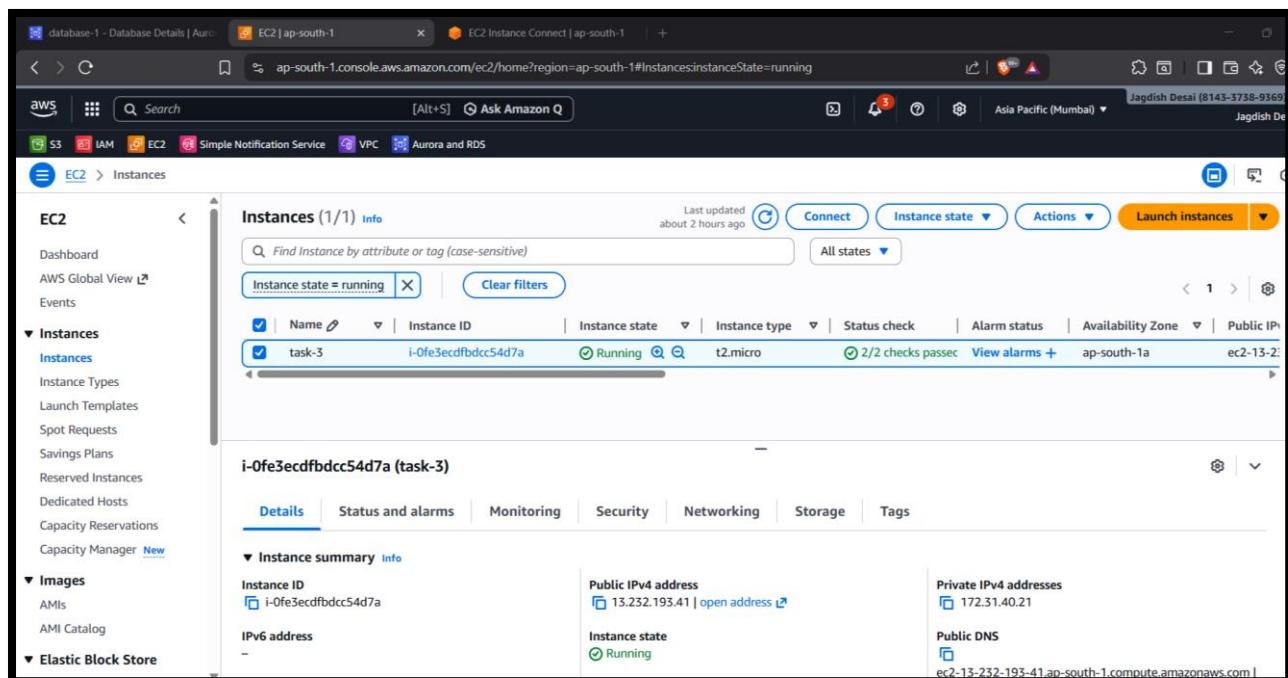
EC2 → Security Group → Inbound Rules

Add:

- Type: **HTTP**
- Port: **80**
- Source: **0.0.0.0/0**

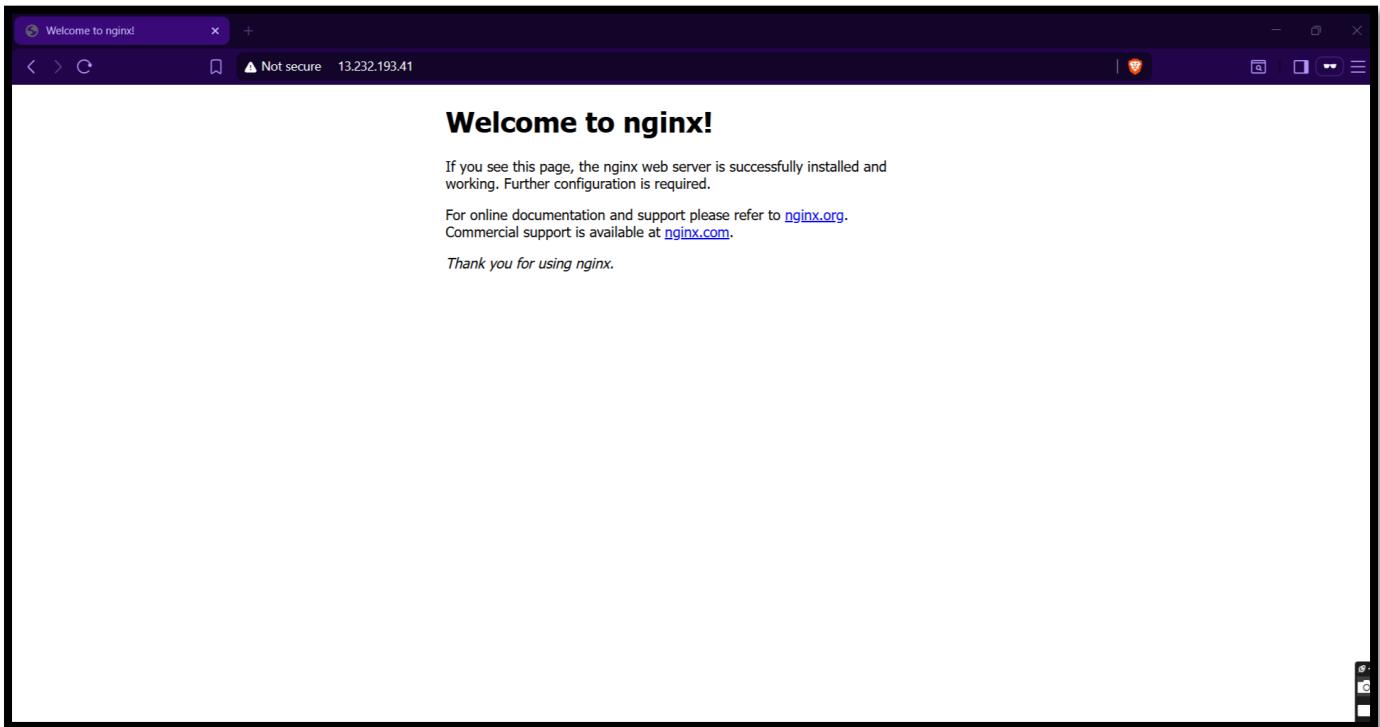
This allows internet users to access your web server

Step 4: Copy Public IP address.



Step 5: Going to browser and paste public ip.

http://13.232.193.41/



Output This confirms:

- EC2 is running
- Docker is working
- Nginx container is live

Creating Own Image

1. Connect to EC2:

Create directory – mkdir app

Create- index.html

Create- server.js

Create package.json - npm init -y

2. Install Required Packages:

npm install express mysql2 body-parser

3. Install Docker

```
sudo yum install docker -y  
sudo systemctl start docker  
sudo systemctl enable docker  
sudo usermod -aG docker ec2-user  
exit
```

4. Login again to EC2

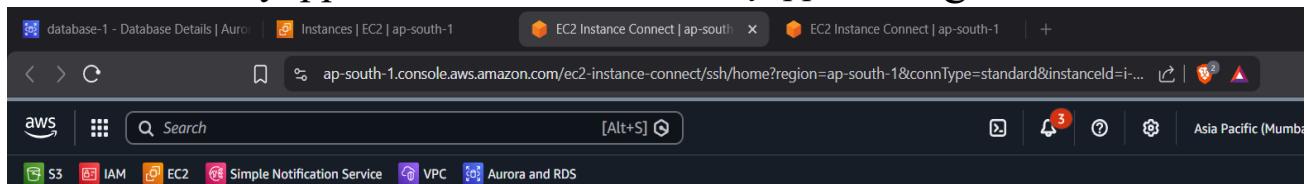
ssh -i yourkey.pem ec2-user@EC2-Public-IP

5. Going to app folder

cd app

6. Build Docker image

docker build -t myapp . //myapp = image name



```
[ec2-user@ip-172-31-1-172 ~]$ cd app  
[ec2-user@ip-172-31-1-172 app]$ docker build -t myapp .  
ERROR: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://<2Fvar%2Frur%2Fdoc  
var/run/docker.sock": connect: permission denied  
[ec2-user@ip-172-31-1-172 app]$ sudo docker build -t myapp .  
[+] Building 1.7s (10/10) FINISHED docker:default  
=> [internal] load build definition from dockerfile 0.0s  
=> => transferring dockerfile: 255B 0.0s  
=> [internal] load metadata for docker.io/library/node:18 1.5s  
=> [internal] load .dockerrcignore 0.0s  
=> => transferring context: 2B 0.0s  
=> [1/5] FROM docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6 0.0s  
=> [internal] load build context 0.2s  
=> => transferring context: 109.28kB 0.1s  
=> CACHED [2/5] WORKDIR /app 0.0s  
=> CACHED [3/5] COPY package*.json ./ 0.0s  
=> CACHED [4/5] RUN npm install 0.0s  
=> CACHED [5/5] COPY . . 0.0s  
=> exporting to image 0.0s  
=> => exporting layers 0.05s  
=> initlayer sha256:2244a55c0a12a05740add1d015d1e0b02f0a217a5af1 0.0s
```

7. Run Docker container

```
docker run -d -p 3000:3000 --restart always myapp
```

```
[ec2-user@ip-172-31-1-172 app]$ sudo docker run -d -p 3000:3000 --restart always myapp
1b2fcf642c14c7a4617e86582e76a15f122b1934d5b399e0ed1270960808cb83
[ec2-user@ip-172-31-1-172 app]$
```

8. Allow port 3000 in EC2 Security Group

The screenshot shows the 'Edit inbound rules' section of the AWS EC2 Security Groups configuration. It lists several existing rules and allows for adding new ones. A warning message at the bottom advises against using 0.0.0.0/0 or ::/0 source ranges.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-01c71eaf5b270a223	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-017579a0e5829fc	SSH	TCP	22	Custom	0.0.0.0/0
sgr-0e438dbf23d6acde9	MYSQL/Aurora	TCP	3306	Custom	0.0.0.0/0
-	Custom TCP	TCP	3000	Anyw...	0.0.0.0/0

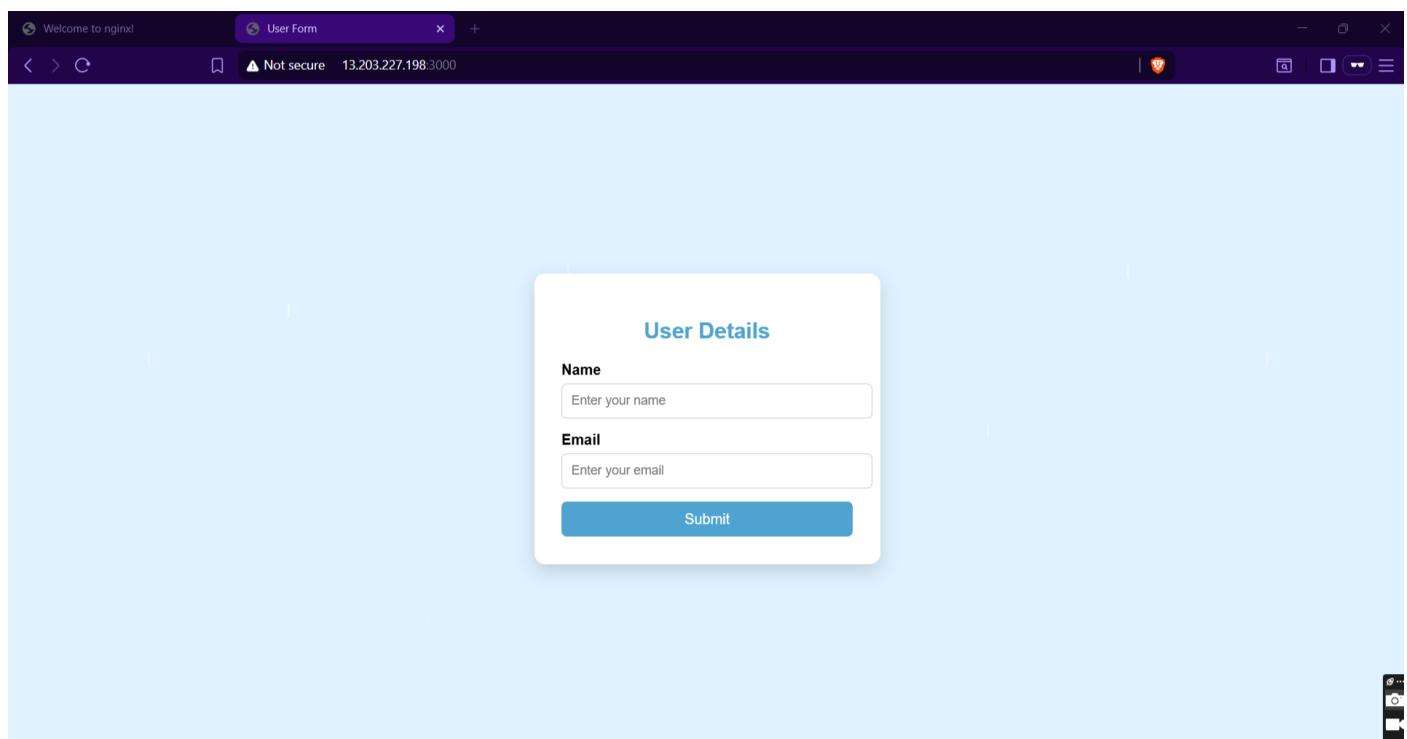
Add rule

⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Preview changes Save rules

Step 9: Using Public IP accessing app in browser

<http://13.203.227.198:3000/>



Final Output:

- EC2 running
- Docker installed
- Container running
- App accessible via **EC2 Public IP**

Index.html code

```
<!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;  
    color: #e75480;  
}  
  
label {
```

```
    font-weight: bold;  
    display: block;  
    margin-bottom: 5px;  
}
```

```
input {  
    width: 100%;  
    padding: 10px;  
    margin-bottom: 15px;  
    border-radius: 6px;  
    border: 1px solid #ccc;  
    font-size: 14px;  
}
```

```
input:focus {  
    outline: none;  
    border-color: #e75480;  
}
```

```
button {  
    width: 100%;  
    padding: 10px;
```

```
background: #e75480;  
border: none;  
color: white;  
font-size: 16px;  
border-radius: 6px;  
cursor: pointer;  
}  
  
button:hover {  
background: #d64570;  
}  
</style>  
</head>  
<body>  
<div class="form-container">  
  <h2>User Details</h2>  
  <form method="POST">  
    <label>Name</label>  
    <input type="text" name="name" placeholder="Enter your name" required>  
    <label>Email</label>  
    <input type="email" name="email" placeholder="Enter your email" required>  
    <button type="submit">Submit</button>  
  </form>  
</div>  
</body>  
</html>
```

Server.js

```
const express = require("express");
const mysql = require("mysql2");
const bodyParser = require("body-parser");
const app = express();
app.use(bodyParser.urlencoded({ extended: true }));
const db = mysql.createConnection({
  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");
});

db.connect((err) => {
    if (err) {
        console.error("DB connection failed:", err.message);
        process.exit(1);
    }
    console.log("Database Connected");
});
```