Lab : Deploy Docker on Amazon Linux 2 EC2 and Run a Web App in a Docker Container

Objective

By the end of this lab, you will:

• Launch an Amazon Linux EC2 instance.

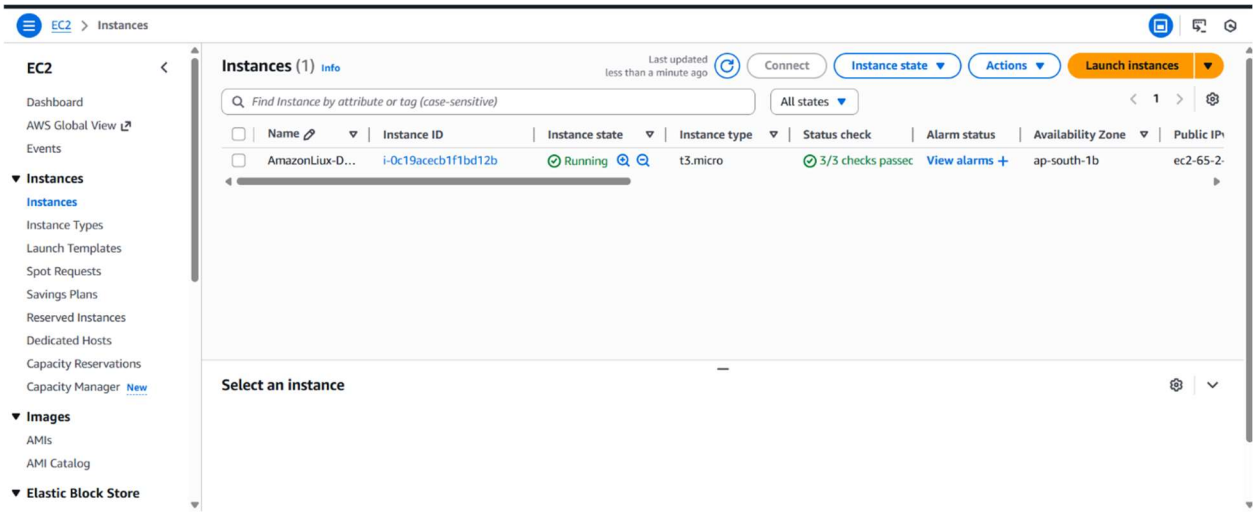
• Install Docker on it.

• Run an Apache (httpd) container.

• Serve a custom webpage.

Step 1: Launch EC2 Instance

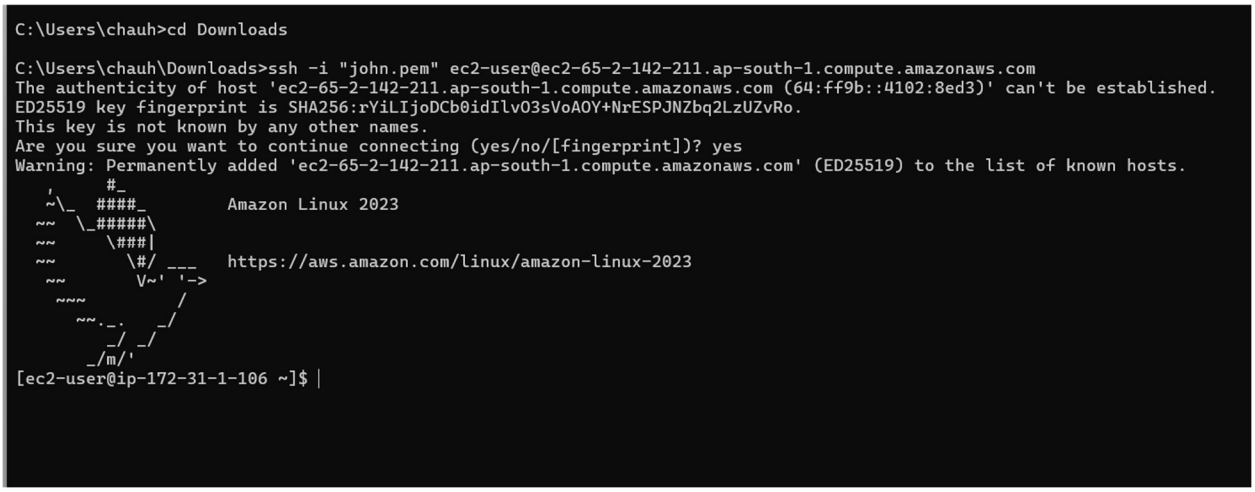
- Name: AmazonLinux-Docker-Lab
- AMI: **Amazon Linux**
- Instance Type: t2.micro
- Key Pair + Security Group (allow SSH 22 and HTTP 80).
- Launch and copy Public IPv4.



Step 2: Connect to EC2 via SSH

From your terminal:

ssh -i "john.pem" ec2-user@ec2-65-2-142-211.ap-south-1.compute.amazonaws.com



Step 3: Install Docker on Amazon Linux

1 Update OS

sudo yum update -y

sudo yum upgrade -y

2 Install Docker package

sudo yum install -y docker

3 Start Docker service

sudo service docker start

3.1.Check it's active:

sudo service docker status

3.2.Enable Docker to start automatically at boot:

sudo chkconfig docker on

4️⃣ Verify version

docker --version

5️⃣ Allow ec2-user to use Docker without sudo (optional)

sudo usermod -aG docker ec2-user

exit

Reconnect: through SSH

ssh -i "john.pem" ec2-user@ec2-65-2-142-211.ap-south-1.compute.amazonaws.com

Step 4: Run Apache (httpd) in Docker

Pull the official image:

docker pull httpd:latest

Run container:

docker run -d -p 80:80 --name my-apache-app httpd:latest

Check container status:

docker ps -a

If it's stopped:

docker start my-apache-app

Step 5: Customize the Web Page

Enter container shell:

docker exec -it my-apache-app /bin/bash

Go to Apache web root:

cd /usr/local/apache2/htdocs

Create your custom index:

echo "MyWebsite - Running on Apache inside Docker" > index.html

cat index.html

exit

```
a6c97c1311d6: Pull complete
4f4fb700ef54: Pull complete
37be031b3615: Pull complete
359a248d4bde: Pull complete
72ba0317f875: Pull complete
Digest: sha256:d3b88ca0822f91e2dec6eb58a2ac7cfade27880926467fc63dcdbc857010b
083
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
[ec2-user@ip-172-31-1-106 ~]$ docker run -d -p 80:80 --name my-apache-app ht
tpd:latest
cf5aaf1b001b4c36802a462559d746e4a458883737296c994dec8a8d95bae12f
[ec2-user@ip-172-31-1-106 ~]$ docker ps -a
CONTAINER ID   IMAGE          COMMAND              CREATED          STATUS
        PORTS                                  NAMES
cf5aaf1b001b   httpd:latest   "httpd-foreground"   29 seconds ago   Up 28 se
conds   0.0.0.0:80->80/tcp, :::80->80/tcp   my-apache-app
[ec2-user@ip-172-31-1-106 ~]$ client_loop: send disconnect: Connection reset

C:\Users\chauh\Downloads>ssh -i "john.pem" ec2-user@ec2-65-2-142-211.ap-sout
h-1.compute.amazonaws.com
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Thu Oct 30 03:07:50 2025 from 152.59.8.94
[ec2-user@ip-172-31-1-106 ~]$ docker exec -it my-apache-app /bin/bash
root@cf5aaf1b001b:/usr/local/apache2# cd /usr/local/apache2/htdocs
root@cf5aaf1b001b:/usr/local/apache2/htdocs# echo "MyWebsite - Running on Ap
ache inside Docker" > index.html
root@cf5aaf1b001b:/usr/local/apache2/htdocs# cat index.html
MyWebsite - Running on Apache inside Docker
root@cf5aaf1b001b:/usr/local/apache2/htdocs# exit
exit
[ec2-user@ip-172-31-1-106 ~]$
```

Step 6: Test the Web App

On your local computer(cmd)

curl http://<EC2-Public-IP> (example : curl http://65.2.142.211)

result:

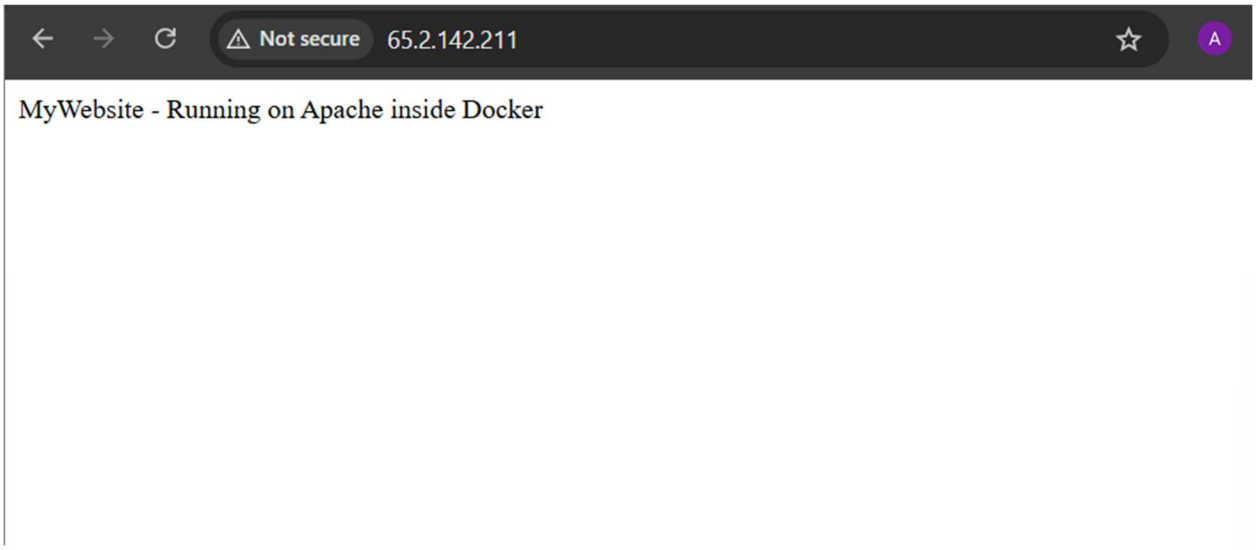
```
[ec2-user@ip-172-31-1-106 ~]$ curl http://65.2.142.211
MyWebsite - Running on Apache inside Docker
[ec2-user@ip-172-31-1-106 ~]$
```

Or open in your browser:

Go to

👉 http://65.2.142.211

You should see:



Step 7: Manage and Monitor Containers

1. Inspect container — **docker inspect my-apache-app**

What it does: shows low-level JSON details about the container (network, mounts, ports, created timestamp, state, config, environment, PID, etc.).

2. Live resource monitoring — **docker stats my-apache-app**

What it does: shows real-time CPU, memory, network I/O, block I/O, and PIDs for containers.

3. Stop a container **— docker stop my-apache-app**

What it does: sends SIGTERM to PID 1 inside the container, waits (default 10s) for graceful shutdown, then SIGKILL if not stopped.

4. Start a container — **docker start my-apache-app**

What it does: restarts a stopped container using its previous configuration.

5. Remove a container — **docker rm my-apache-app**

What it does: deletes the stopped container (its writable layer). Data inside the container that is not in volumes/bind mounts will be lost.