

## **PROJECT EXPLANATION: DEPLOYMENT OF A WEB APPLICATION USING AWS AND DOCKER**

In this project, a simple web application is deployed on the cloud using Amazon Web Services (AWS) and Docker. The main idea of this project is to understand how a website can be hosted online using cloud servers instead of a personal computer or local system.

First, a VPC (Virtual Private Cloud) was created in AWS. This acts like a private network where all the project resources are placed securely. Inside the VPC, a public subnet was created so that the server could communicate with the internet. An Internet Gateway was attached to the VPC to allow internet access. After that, a route table was configured and connected to the public subnet so that incoming and outgoing traffic could move properly.

Next, an EC2 instance was launched inside the public subnet. This instance works as the main server for the project. A security group was set up to allow web traffic on port 80 and SSH access on port 22. Using a private key file, the EC2 instance was accessed remotely through SSH.

After connecting to the server, Docker was installed on the EC2 instance. Docker is used to run applications inside containers, which makes deployment easier and more reliable. A project folder was then created, and a simple index.html file was written as the web page. A Dockerfile was created to define how the web application should run using the NGINX web server.

The Docker image was built using the Dockerfile, and then a Docker container was started from that image. The container was run in the background and connected to port 80 so the web page could be accessed through a browser. The container was also set to restart automatically in case of any issue.

Finally, the public IP address of the EC2 instance was opened in a web browser. The web page loaded successfully, showing the project message. This confirmed that the application was deployed correctly and is accessible from anywhere.

Overall, this project helped in understanding cloud networking, server setup, and container-based application deployment using AWS and Docker. It shows how modern web applications are deployed in real-world environments

# Deployment of a Web Application on AWS Using Docker

The screenshot shows the AWS VPC dashboard under the 'Your VPCs' section. A new VPC named 'project-VPC' has been created and is listed in the table. The table includes columns for Name, VPC ID, State, Encryption controls, Block Public Access, and IPv4 CIDR. The 'Actions' button is highlighted in orange.

Name	VPC ID	State	Encryption c...	Block Public...	IPv4 CIDR
vpc-0dc3c126ee125269c	Available	-	-	Off	172.31.0.0/20
project-VPC	vpc-0ec6d83187e17f62f	Available	-	Off	10.20.0.0/16

First, the AWS Management Console was opened and the VPC service was selected. A new Virtual Private Cloud (VPC) was created by giving it a name (project-VPC) and assigning an IPv4 CIDR block (10.20.0.0/16).

Default settings like DNS resolution and main route table were enabled, and after creating it, the VPC became available to host project resources securely.

The screenshot shows the AWS VPC Subnets dashboard under the 'Subnets' section. A new subnet named 'public-subnet' has been created and is listed in the table. The table includes columns for Name, Subnet ID, State, VPC, Block Public Access, and IPv4 CIDR. The 'Actions' button is highlighted in orange.

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
subnet-049088c8893f37502	Available	vpc-0dc3c126ee125269c	Off	172.31.16.0/2	
subnet-06cbfb786e5870fc8	Available	vpc-0dc3c126ee125269c	Off	172.31.0.0/20	
subnet-01525ea863d3595b0	Available	vpc-0dc3c126ee125269c	Off	172.31.32.0/2	
public-subnet	subnet-0110b02e37044d149	Available	vpc-0ec6d83187e17f62f   project-VPC	Off	10.20.0.0/20

After creating the project-VPC, the Subnets section in the AWS VPC Dashboard was opened.

A new subnet named **public-subnet** was created by selecting the VPC and choosing an **Availability Zone (ap-south-1a)**.

An **IPv4 CIDR block (10.20.0.0/20)** was assigned to divide the VPC network into a smaller range. The subnet was created successfully and is now **available** to launch EC2 instances and other resources.

Internet gateways (1/2) [Info](#)

Name	Internet gateway ID	State	VPC ID	Owner
-	igw-0539fe5f99b21031b	Attached	vpc-0dc3c126ee125269c	463816980004
<input checked="" type="checkbox"/> IGW	igw-06f07ee545c737001	Attached	vpc-0ec6d83187e17f62f   project-VPC	463816980004

igw-06f07ee545c737001 / IGW

[Details](#) [Tags](#)

**Details**

Internet gateway ID <a href="#">igw-06f07ee545c737001</a>	State <a href="#">Attached</a>	VPC ID <a href="#">vpc-0ec6d83187e17f62f   project-VPC</a>	Owner <a href="#">463816980004</a>
--	-----------------------------------	---	---------------------------------------

From the AWS VPC Dashboard, the **Internet Gateway** option was selected and a new gateway named **IGW** was created.

After creation, the Internet Gateway was **attached to the project-VPC**. This allows resources inside the VPC, such as public subnets, to **connect to the internet**.

aws | Search [Alt+S] | Asia Pacific (Mumbai) | Gauri valdyo

VPC > Route tables > rtb-02045dcec2a288835 > Edit subnet associations

**Edit subnet associations**  
Change which subnets are associated with this route table.

**Available subnets (1/1)**

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR	Route table ID
<input checked="" type="checkbox"/> public-subnet	subnet-0110b02e37044d149	10.20.0.0/20	-	rtb-02045dcec2a288835 / Route-Table

**Selected subnets**

subnet-0110b02e37044d149 / public-subnet <a href="#">X</a>
--

[Cancel](#) [Save associations](#)

The **Route Tables** section was opened in the AWS VPC Dashboard and an existing route table was selected.

The **public-subnet** was associated with this route table by editing subnet associations. After saving, this route table controls network traffic for the public subnet, enabling proper routing.

The **route table** was opened and the **Edit routes** option was selected. A new route with destination **0.0.0.0/0** was added and the target was set to the **Internet Gateway (IGW)**.

After saving the changes, the public subnet was able to **send and receive internet traffic** through the IGW.

An **EC2 instance** named **project-instance** was launched from the AWS EC2 Dashboard. The instance was placed in the **public subnet** of the project VPC and assigned a **public IPv4 address**.

A **t3. micro instance type** was selected, and after successful launch, the instance entered the **running state** and became ready for use.

**Inbound rules**

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0b4c1ec58e92e08a8	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-070df588399a2b208	SSH	TCP	22	Custom	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

A **Security Group** was opened and its **inbound rules** were edited. **HTTP (port 80)** was allowed to enable web access, and **SSH (port 22)** was allowed for remote login.

The source was set to **0.0.0.0/0**, allowing access from any IP address to the EC2 instance.

```
shrey@LAPTOP-KNBCS8I9 MINGW64 ~
$ cd Downloads/

shrey@LAPTOP-KNBCS8I9 MINGW64 ~/Downloads
$ ssh -i "shreyash.pem" ec2-user@13.201.50.201
ssh: connect to host 13.201.50.201 port 22: Connection timed out

shrey@LAPTOP-KNBCS8I9 MINGW64 ~/Downloads
$ ssh -i "shreyash.pem" ec2-user@13.201.50.201
The authenticity of host '13.201.50.201 (13.201.50.201)' can't be established.
ED25519 key fingerprint is SHA256:xbgM3J+h7JTRBZK9Dm0KjM9VRuXMnckWUXcJNEM6VHg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '13.201.50.201' (ED25519) to the list of known hosts.

      #_
     ~\_ #####_      Amazon Linux 2023
     ~~\#####\
     ~~ \|##|
     ~~  \#/   _--> https://aws.amazon.com/linux/amazon-linux-2023
     ~~~
     ~~ .-. / \
     _/m/ , -/ \

```

The EC2 instance was accessed remotely using **SSH** from a local system with a **private key**. After accepting the host authenticity, a successful connection was established to the **Amazon Linux 2023** server.

```

[ec2-user@ip-10-20-8-11 ~]$ sudo yum install docker -y
install: invalid option -- 'y'
Try 'install -h' for more information.
[ec2-user@ip-10-20-8-11 ~]$ sudo yum install docker -y
Error: This command has to be run with superuser privileges (under the root user on most systems).
[ec2-user@ip-10-20-8-11 ~]$ sudo -i
[root@ip-10-20-8-11 ~]# yum install docker -y
Amazon Linux 2023 Kernel Livepatch repository
Dependencies resolved.
=====
Transaction Summary
=====
Install  11 Packages

```

Docker was installed on the EC2 instance using the package manager with root permissions.

The system automatically downloaded Docker and its required dependencies.

After installation, Docker became available to run and manage containers on the server.

```

complete!
[root@ip-10-20-8-11 ~]# docker --version
Docker version 25.0.14, build Obab007
[root@ip-10-20-8-11 ~]# systemctl docker status
Unknown command verb docker.
[root@ip-10-20-8-11 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
     Active: inactive (dead)
       Docs: https://docs.docker.com
[root@ip-10-20-8-11 ~]# systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-10-20-8-11 ~]# systemctl start docker
[root@ip-10-20-8-11 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since sun 2026-02-08 06:34:30 UTC; 3s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Process: 27199 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
   Process: 27200 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
 Main PID: 27201 (dockerd)
    Tasks: 8
   Memory: 30.5M
      CPU: 317ms
     CGroup: /system.slice/docker.service
             └─27201 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

Feb 08 06:34:29 ip-10-20-8-11.ap-south-1.compute.internal systemd[1]: Starting docker.service - Docker Application Container Engine...

```

The installed Docker version was verified to confirm a successful setup.

The Docker service was then **enabled and started** using systemctl.

Finally, the service status showed Docker running actively, confirming it is ready to manage containers.

```
[root@ip-10-20-8-11 ~]# mkdir project
[root@ip-10-20-8-11 ~]# cd project/
[root@ip-10-20-8-11 project]# touch index.html
[root@ip-10-20-8-11 project]# vim index.html
[root@ip-10-20-8-11 project]# ls
index.html
[root@ip-10-20-8-11 project]# cat index.html
This is my project
based in docker and AWS
[root@ip-10-20-8-11 project]# touch docker
[root@ip-10-20-8-11 project]# rm -f docker
[root@ip-10-20-8-11 project]# ls
index.html
[root@ip-10-20-8-11 project]# touch dockerfile
[root@ip-10-20-8-11 project]# ls
dockerfile index.html
[root@ip-10-20-8-11 project]# vim dockerfile
[root@ip-10-20-8-11 project]# cat dockerfile
```

A project directory was created on the EC2 server and a basic **index.html** file was added as the web page content.

A **Dockerfile** was then created in the same folder to define how the application will be containerized.

This prepares the project files for building a Docker image.

```
[root@ip-10-20-8-11 project]# sudo docker build -t project .
[+] Building 6.4s (7/7) FINISHED
--> [internal] load build definition from dockerfile
--> [internal] load metadata for docker.io/library/nginx:latest
--> [internal] load .dockernignore
--> [internal] transfer context: 2B
--> [internal] load build context
--> [internal] transfer context: 1428
--> [1/2] FROM docker.io/library/nginx:latest@sha256:341bf0f3ce6c5277d6002cf6e1fb0319fa4252add24ab6a0e262e0056d313208
--> => resolve docker.io/library/nginx:latest@sha256:341bf0f3ce6c5277d6002cf6e1fb0319fa4252add24ab6a0e262e0056d313208
--> => sha256:5cddef4ac335f68438701c14c5f17a992f5e3669ceab7309257d169ea7a856b1 9.09kB / 9.09kB
--> => sha256:46bf3a120c8ebc0c207a3a819b44ad7e6bbcd0b27c2c8e743068baeac6c321d2 33.13MB / 33.13MB
--> => sha256:341bf0f3ce6c5277d6002cf6e1fb0319fa4252add24ab6a0e262e0056d313208 10.23kB / 10.23kB
--> => sha256:514a9c2814250e61395ef4d6125ece1a8fb3b09f4aa2ba441e9f7acf0b66b5b 2.29kB / 2.29kB
--> => sha256:0c8d55a450dc58de60579b9cc5b708de9e7957f4391fc7de941b7c7e245da0 29.78MB / 29.78MB
--> => sha256:4f4fe02d542a6a146723a51b42b2e06362f89f58430dd4d9aaad7ac26c905c 625B / 625B
--> => sha256:7b6cb5cac/b09e037d667024c9af9c0429738a2b7b24c2d41f5bce008c139c 955B / 955B
--> => extracting sha256:0c8d55a450dc58de60579b9cc5b708de9e7957f4391fc7de941b67c7e245da0
--> => sha256:f73400a233fdfe4b0e1ebf8a8035353b7808c1fd8a65c9704099d1e8ae/d1 404B / 404B
--> => sha256:47cd406a84ef91b76fa/a653ed5e89663a79f8c2644a0e0387fa0b1395e511a 1.21kB / 1.21kB
--> => sha256:bae5a1799a03b0a3df8179c56dd95f7db6f78c7aa10600d101196c723994f679 1.40kB / 1.40kB
--> => extracting sha256:46bf3a120c8ebc0c207a3a819b44ad7e6bbcd0b27c2c8e743068baeac6c321d2
--> => extracting sha256:46bf3a120c8ebc0c207a3a819b44ad7e6bbcd0b27c2c8e743068baeac6c321d2
--> => extracting sha256:7b6cb5cac/b09e037d667024c9af9c0429738a2b7b24c2d41f5bce008c139c
--> => extracting sha256:f73400a233fdfe4b0e1ebf8a8035349b78d8c1fd83d665c9704099d1e8ae/d1
--> => extracting sha256:47cd406a84ef91b76fa/a653ed5e89663a79f8c2644a0e0387fa0b1395e511a
--> => extracting sha256:bae5a1799a03b0a3df8179c56dd95f7db6f78c7aa10600d101196c723994f679
--> [2/2] COPY index.html /usr/share/nginx/html/index.html
--> exporting to image
--> => exporting layers
--> => writing image sha256:b58abef96aca3f391099f2e296203e876f278567295276d83e0907511c0c52f
--> naming to docker.io/library/project
```

A Docker image was built using the **Dockerfile** with the command `docker build -t project`

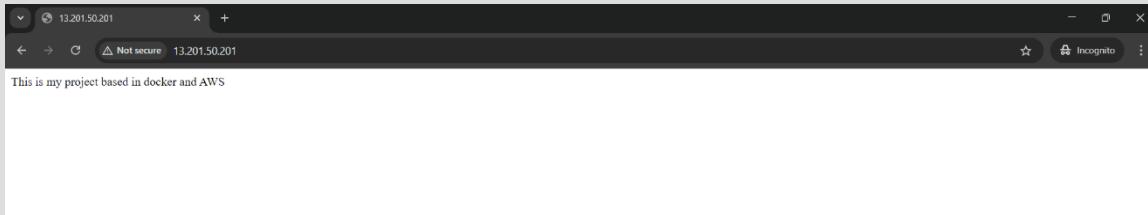
During the build, the **NGINX base image** was pulled and the project's **index.html** file was added to the container.

After completion, a ready-to-use Docker image was created successfully

```
[root@ip-10-20-8-11 project]# sudo docker run -d \
-p 80:80 \
--restart always \
--name project-container \
project
e2ffa2f612eefc9644f53036e48ef8c5147f845e356a9118e4b043652cb79ba2
[root@ip-10-20-8-11 project]# docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED          STATUS          PORTS          NAMES
e2ffa2f612ee  project    "/docker-entrypoint..."   27 seconds ago   Up 26 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   project-container
```

The Docker image was run as a **container** in detached mode using port mapping **80:80**.  
The container was named **project-container** and configured to **restart automatically**.

The running status confirms that the web application is live and accessible.



The web application was successfully accessed using the **public IP address** of the EC2 instance in a browser.

The displayed page confirms that the **Docker container with NGINX** is running correctly.

This verifies the successful deployment of the project on **AWS using Docker**.