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.

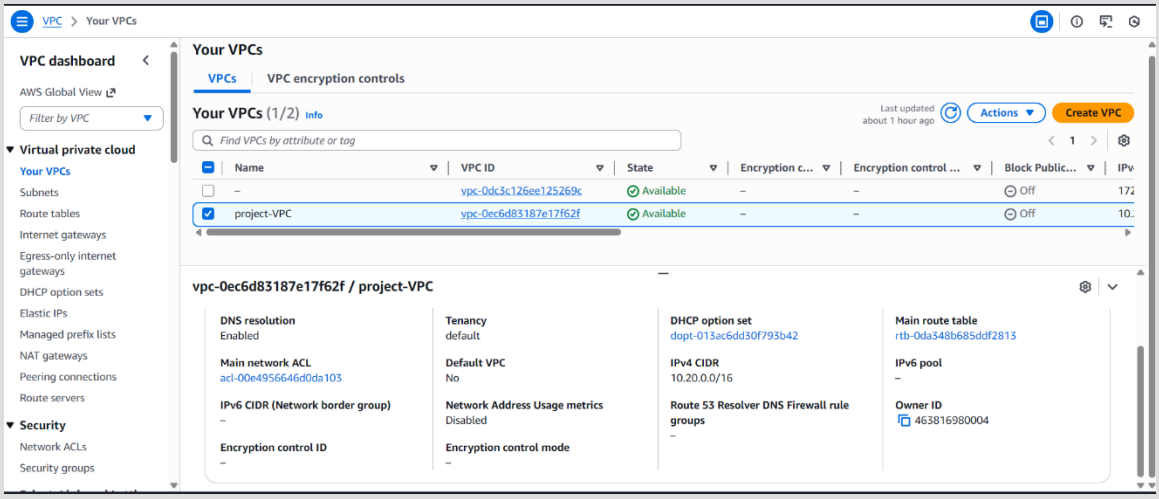
After connecting to the server, Docker was installed on the EC2 instance. Docker is used to run applications inside containers, which makes deployment easier and more reliable. A project folder was then created, and a simple index.html file was written as the web page. A Dockerfile was created to define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

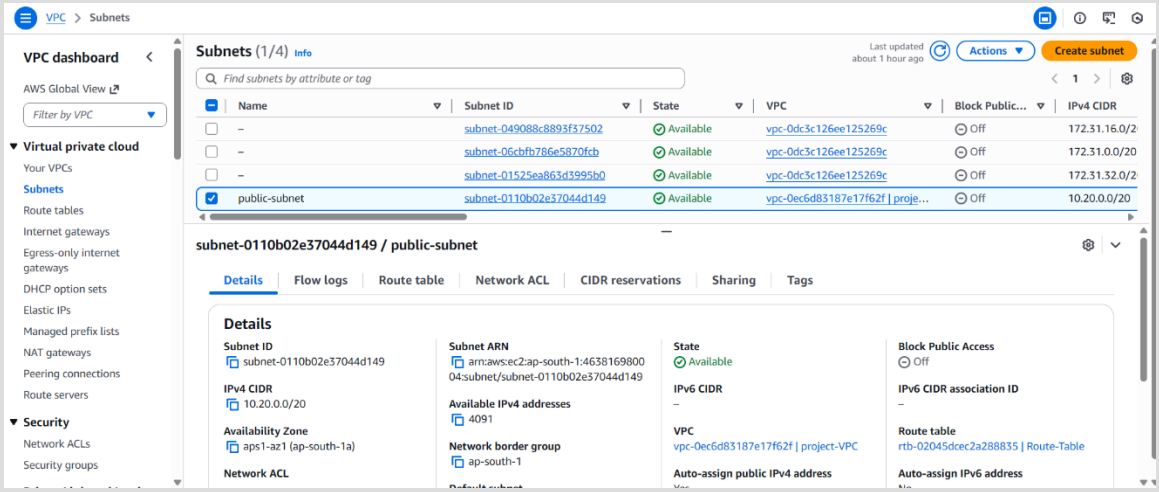
Finally, the public IP address of the EC2 instance was opened in a web browser. The web page loaded successfully, showing the project message. This confirmed that the application was deployed correctly and is accessible from anywhere.

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

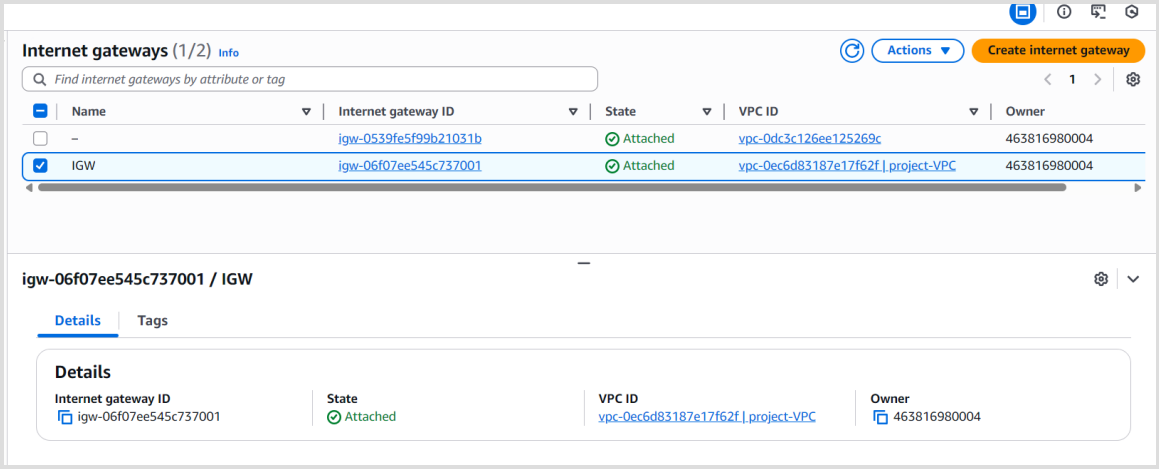
Deployment of a Web Application on AWS Using Docker



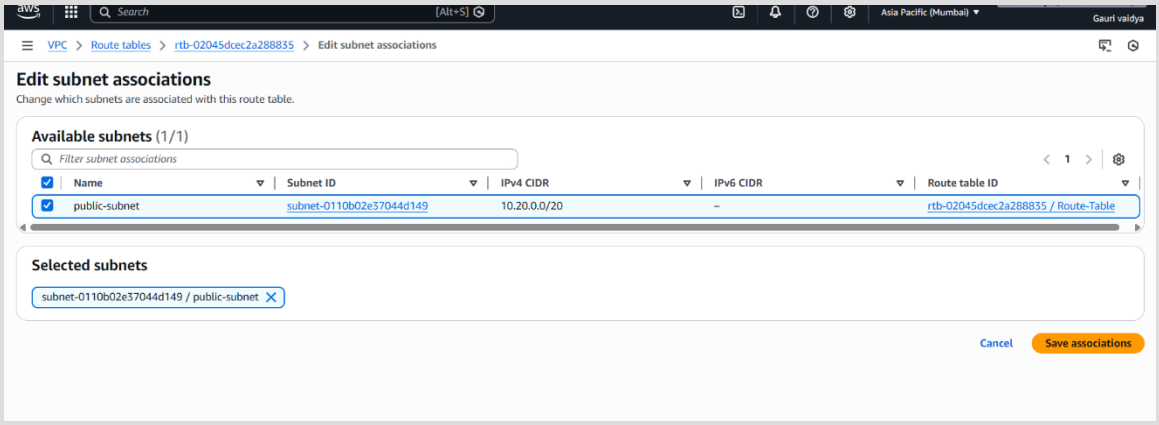
First, the AWS Management Console was opened and the VPC service was selected. A new Virtual Private Cloud (VPC) was created by giving it a name (project-VPC) and assigning an IPv4 CIDR block (10.20.0.0/16).
Default settings like DNS resolution and main route table were enabled, and after creating it, the VPC became available to host project resources securely.



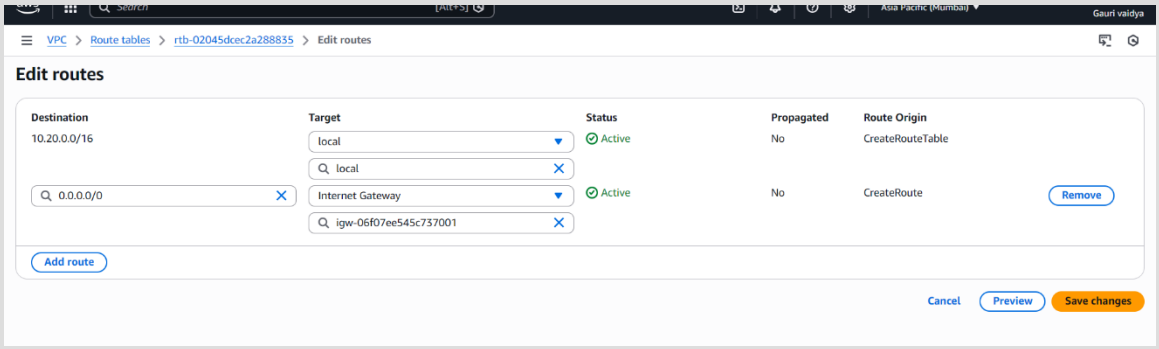
After creating the **project-VPC**, the **Subnets** section in the AWS VPC Dashboard was opened.
A new subnet named **public-subnet** was created by selecting the VPC and choosing an **Availability Zone (ap-south-1a)**.
An **IPv4 CIDR block (10.20.0.0/20)** was assigned to divide the VPC network into a smaller range. The subnet was created successfully and is now **available** to launch EC2 instances and other resources.



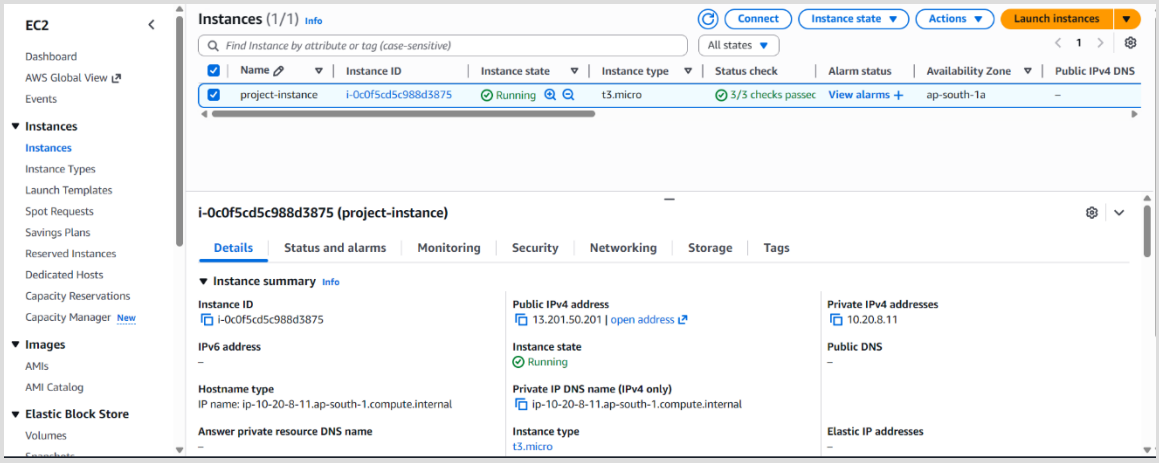
From the AWS VPC Dashboard, the **Internet Gateway** option was selected and a new gateway named **IGW** was created.
After creation, the Internet Gateway was **attached to the project-VPC**.
This allows resources inside the VPC, such as public subnets, to **connect to the internet**.



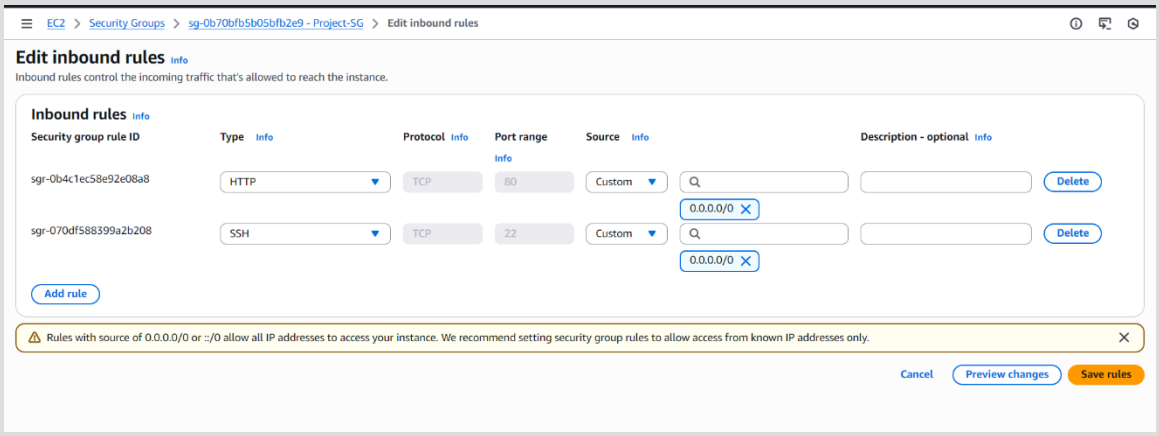
The **Route Tables** section was opened in the AWS VPC Dashboard and an existing route table was selected.
The **public-subnet** was associated with this route table by editing subnet associations.
After saving, this route table controls network traffic for the public subnet, enabling proper routing.



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.



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+h7JTRBZK9DmOKjM9VRuXMnckWUXcJNEM6VHg.
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
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/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 ~]$ install docker -y
install: invalid option -- 'y'
Try 'install --help' for more information.
[ec2-user@ip-10-20-8-11 ~]$ 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                                                    257 kB/s |  31 kB     00:00
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
 libnfnetlink                           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
```

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 0bab007
[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)
TriggeredBy: ○ docker.socket
       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.

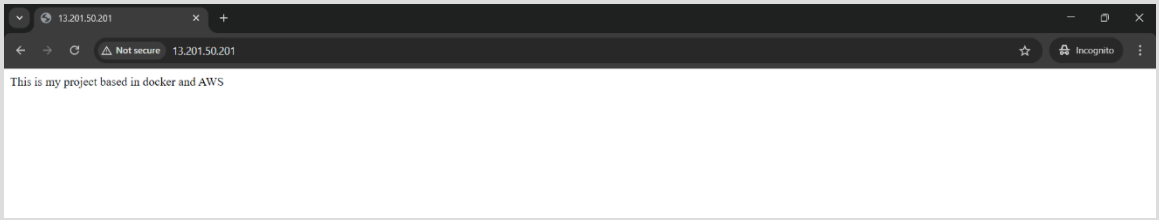
```
[root@ip-10-20-8-11 ~]# mkdir project
[root@ip-10-20-8-11 ~]# cd project/
[root@ip-10-20-8-11 project]# touch index.html
[root@ip-10-20-8-11 project]# vim index.html
[root@ip-10-20-8-11 project]# ls
index.html
[root@ip-10-20-8-11 project]# cat index.html
This is my project
based in docker and AWS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

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
[root@ip-10-20-8-11 project]# AC
```

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**.