PROJECT 2 — EC2 + NGINX + Docker + Domain Mapping + SSL

EC2 Instance with Domain Mapping and NGINX

a. Infrastructure Requirements:

i. EC2 Instance:

1. Deploy 2 **EC2 Instance** in a **Private Subnet**.
2. Attach an **Elastic IP** to the instance to ensure it has a static public IP address for domain mapping.

ii. Domain Mapping:

1. Associate with a domain name.
   "**ec2-docker1.<domain-name>**" and
   "**ec2-instance1.<domain-name>**"
   "**ec2-docker2.<domain-name>**" and
   "**ec2-instance2.<domain-name>**"

iii. Application Load Balancer (ALB) & Domain Mapping

1. Set up an **Application Load Balancer (ALB)** in **Public Subnets** to handle incoming HTTP/HTTPS traffic.
2. Set up SSL certificate. The website should not open HTTP.
3. HTTP traffic should be redirected to HTTPS.
4. Configure the ALB to associate with a domain name.
   "**ec2-alb-docker.<domain-name>**"
   "**ec2-alb-instance.<domain-name>**"

iv. Docker

1. Install **Docker**
2. Run a Docker container that responds with "**Namaste from Container**" on an **internal port** (e.g., **8080**)

v. NGINX Configuration:

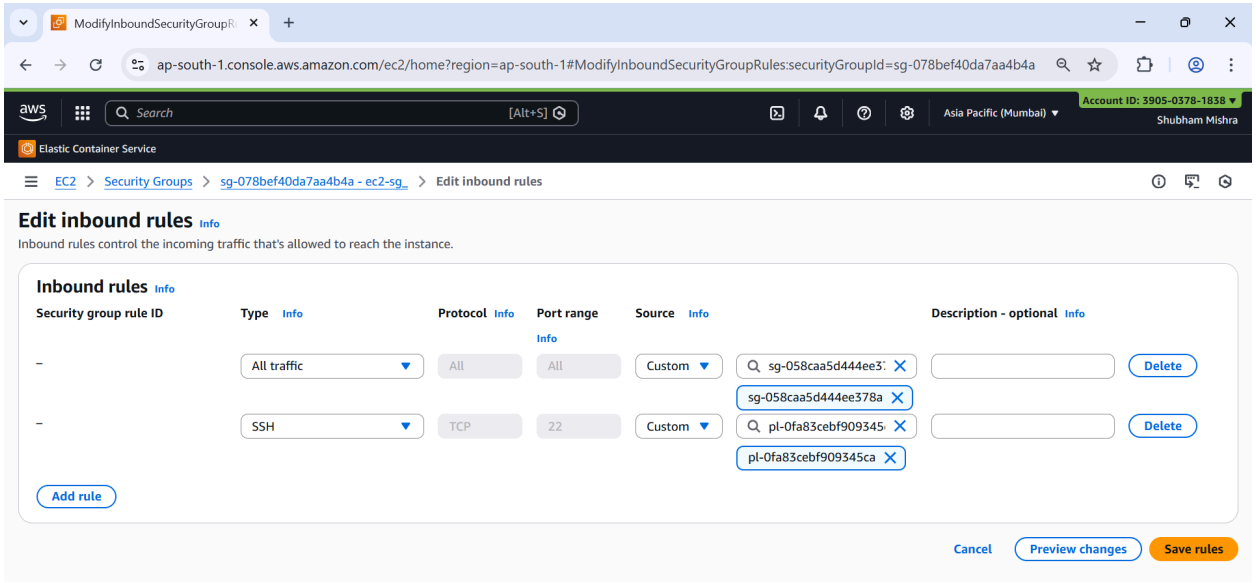
1. Install **NGINX**
2. Domain-based Content Serving:
   a. "**ec2-instance.<domain-name>**" - Configure NGINX to serve the text "**Hello from Instance**"
   b. "**ec2-docker.<domain-name>**" - Set up NGINX to forward requests to a Docker container running on the same instance, which serves the text "**Namaste from Container**".

vi. SSL/TLS with Let's Encrypt:

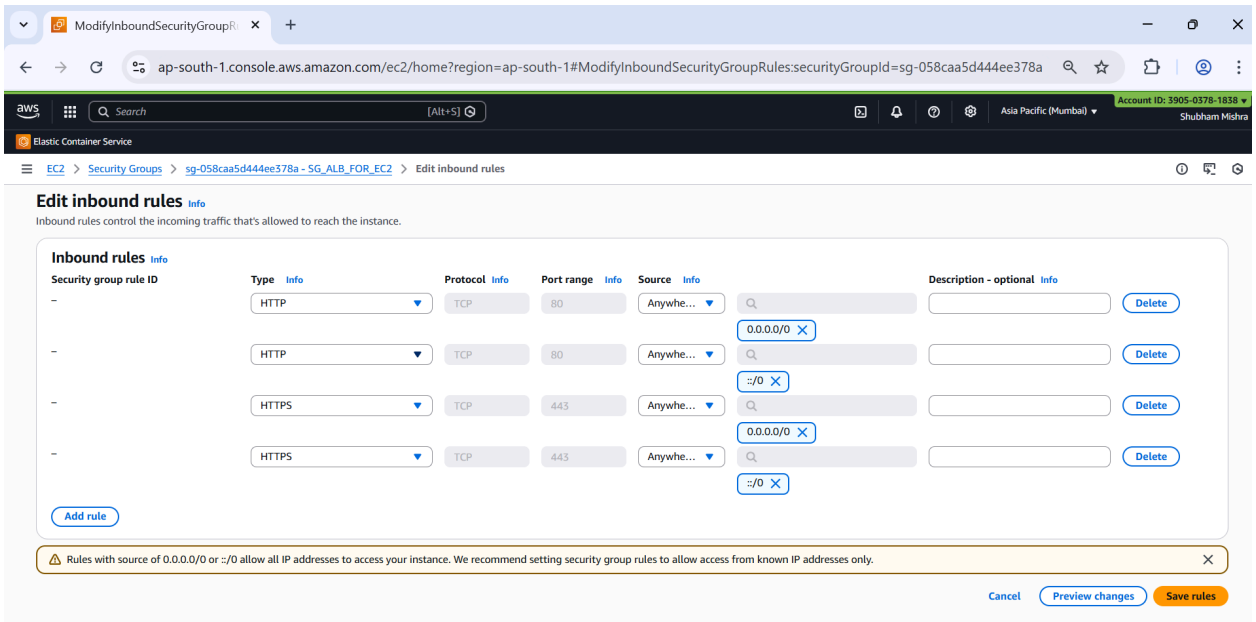
1. Set up **Let's Encrypt** to obtain the subdomain's **SSL certificate**.
2. Configure **NGINX** to use this certificate for **HTTPS** access. Ensure that all **HTTP traffic is redirected to HTTPS** for the subdomain.

We will be using Same vpc and same subnets and same alb for this assignment so skipping vpc subnet steps

Step 1) Now we will create Security group for EC2 and ALB_EC2

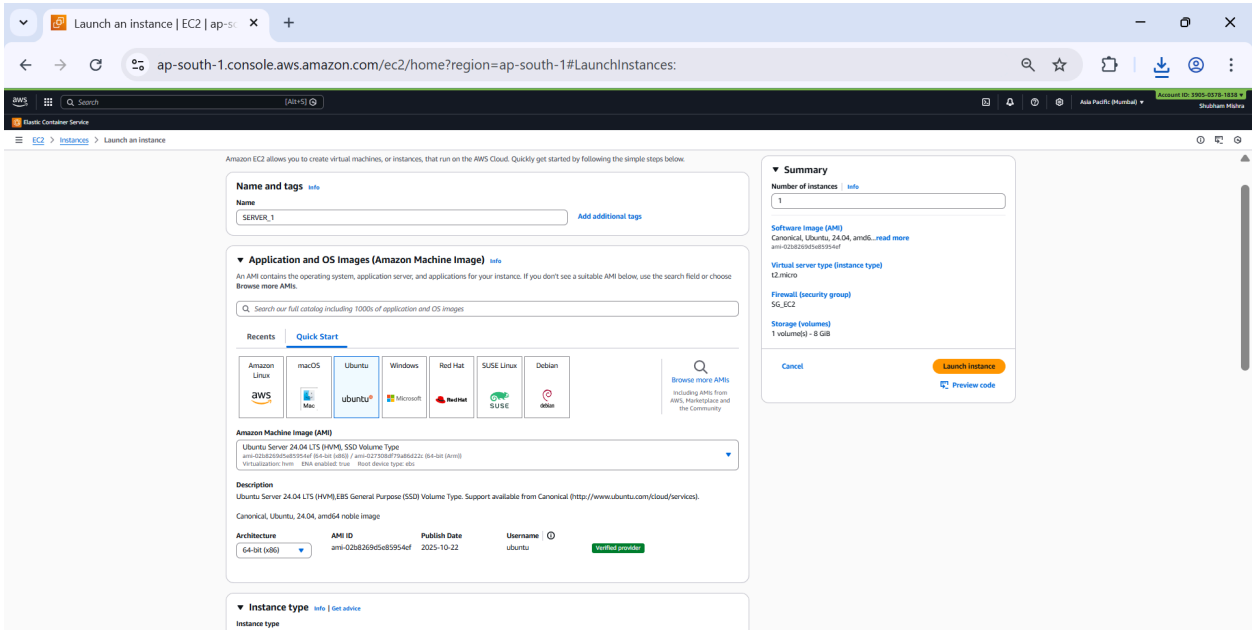


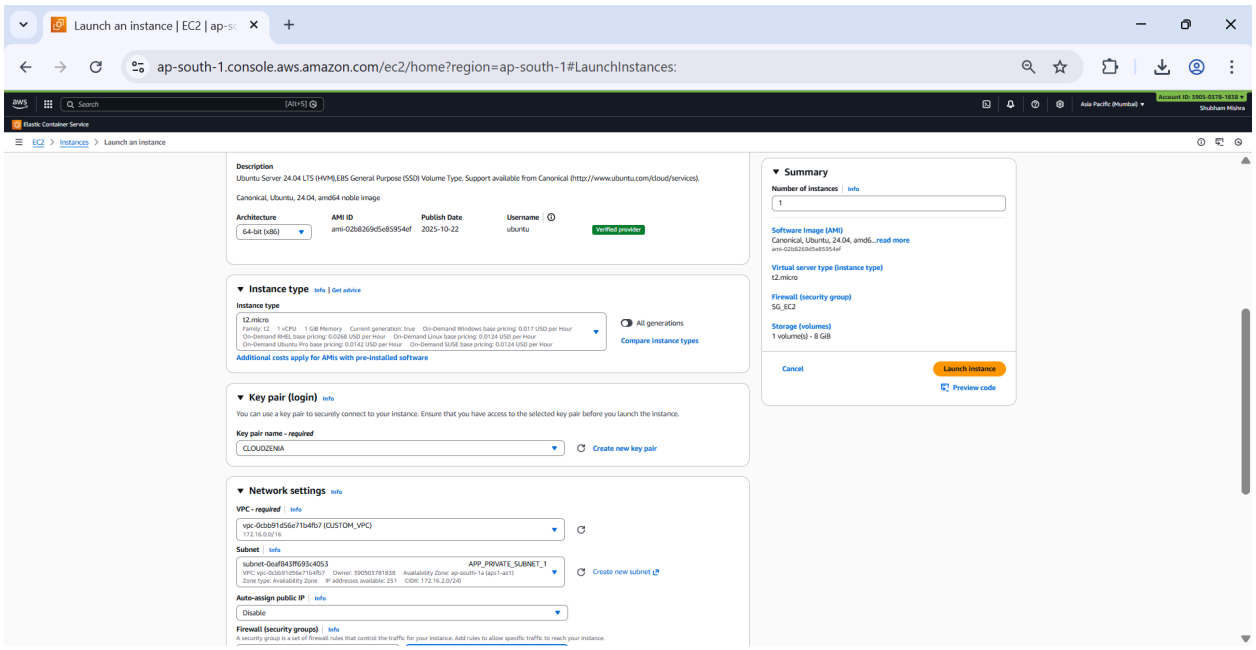
SG for EC2

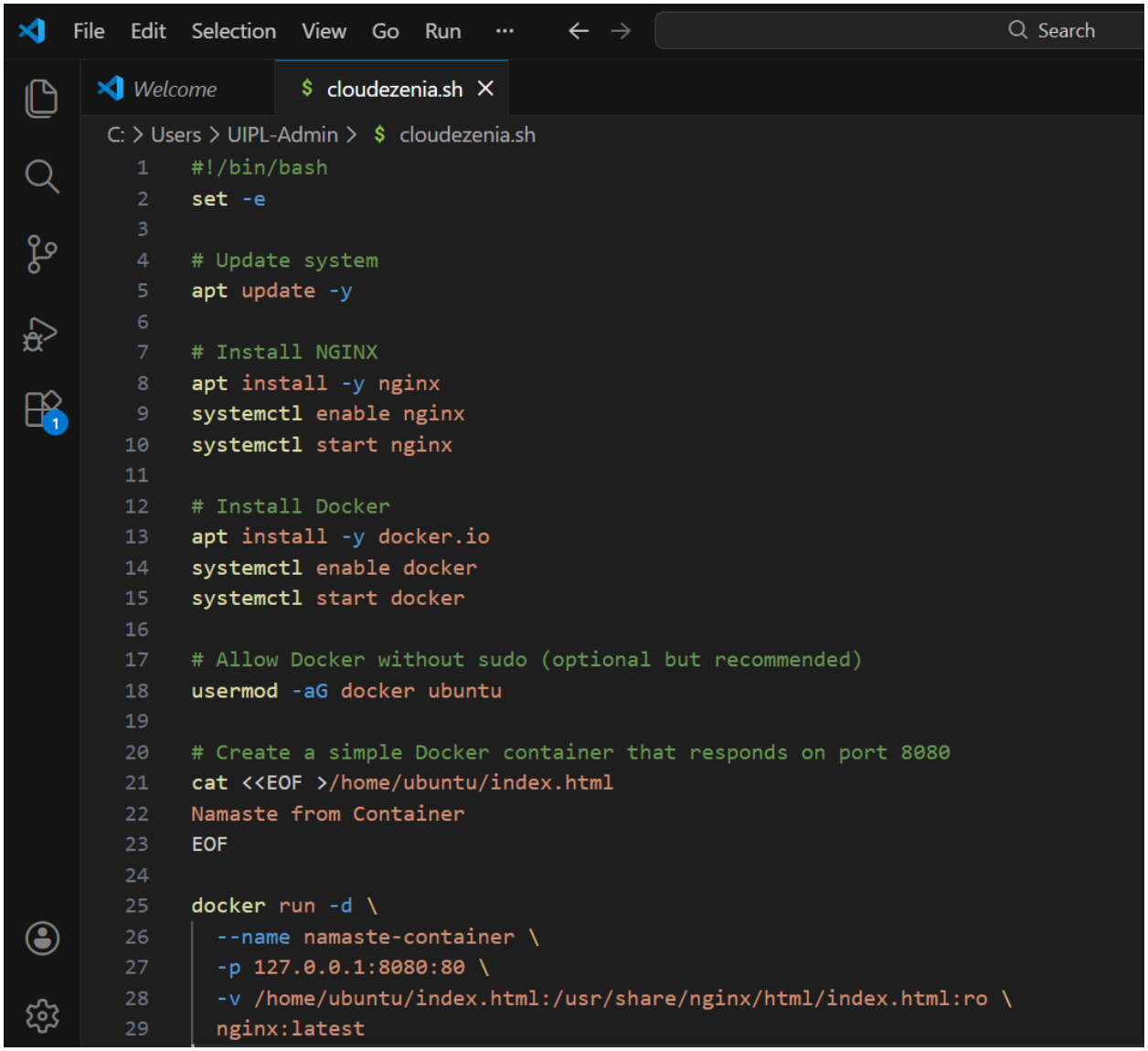


SG for ALB

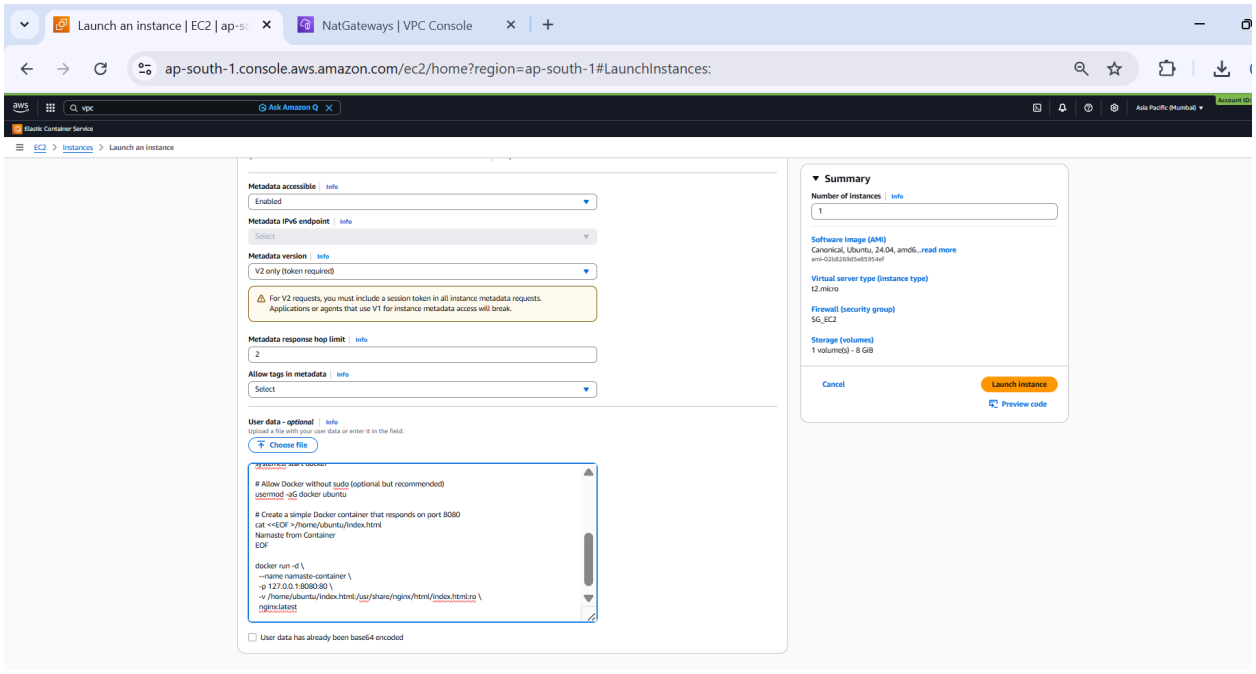
Step 2) We will launch two ec2 instances in private subnet and install nginx and docker on both the servers.

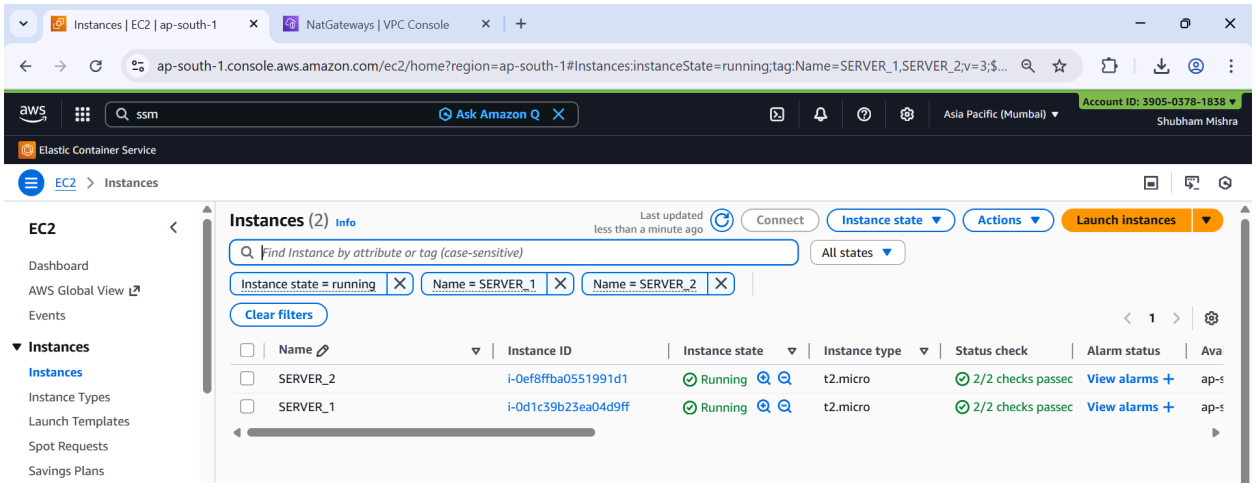




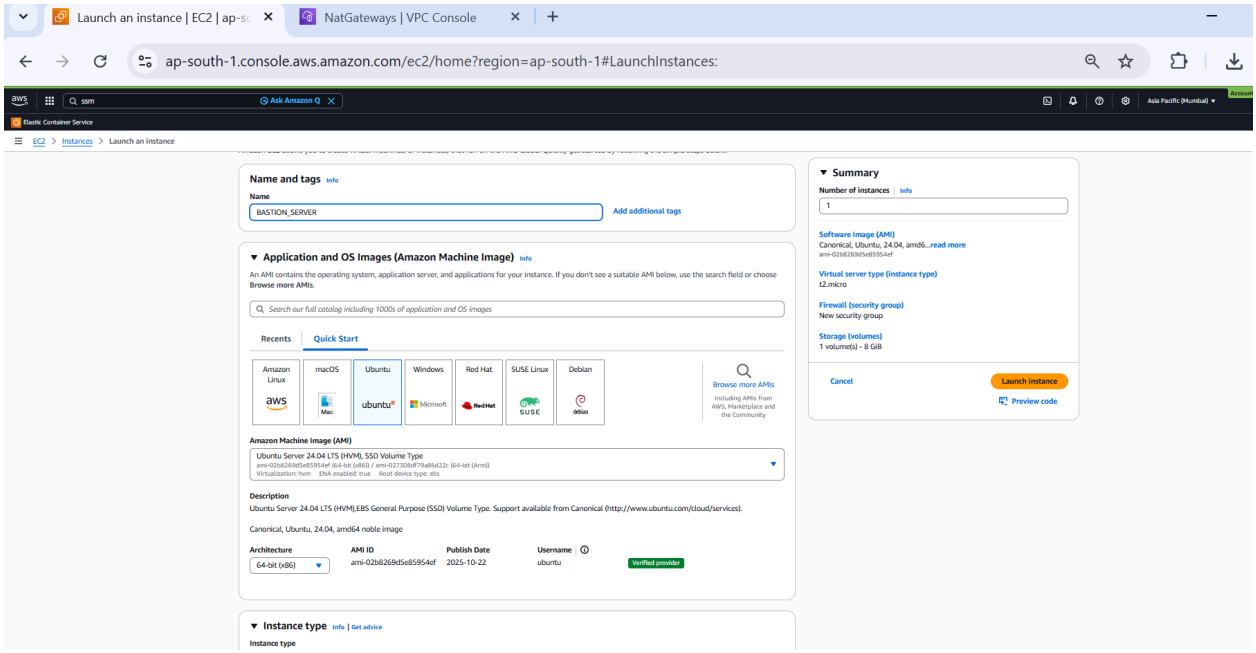


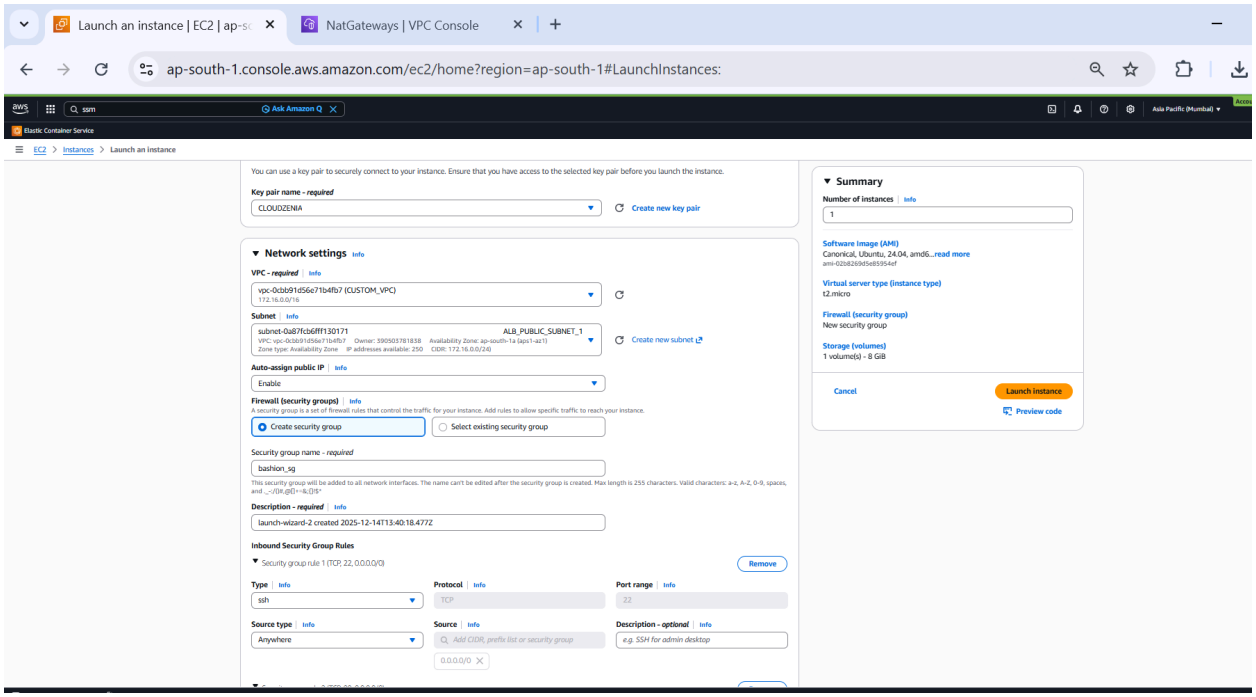
Paste this user data into servers to install during bootstrapping.

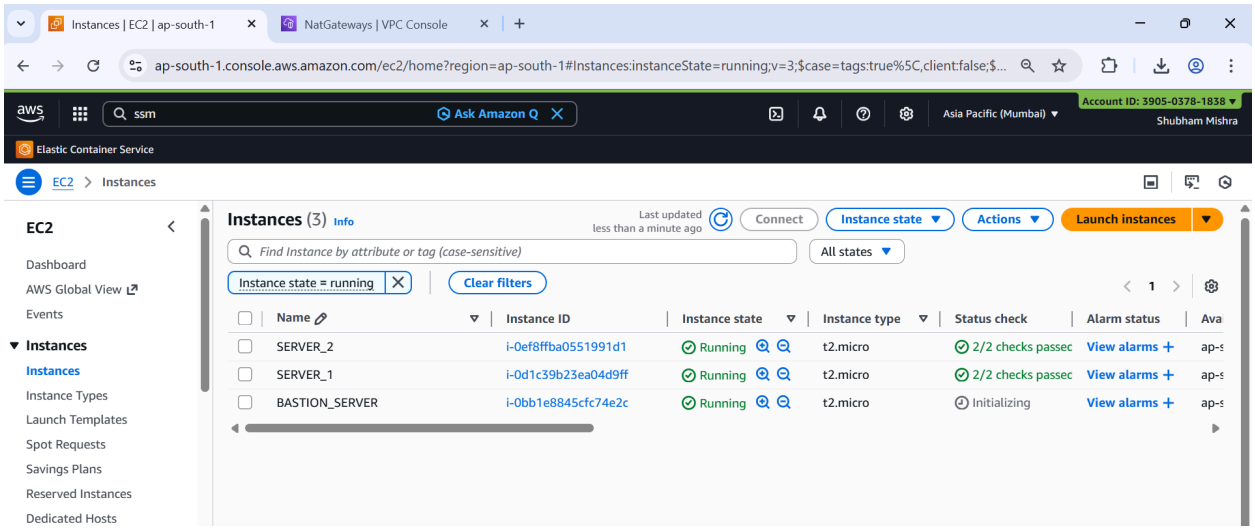




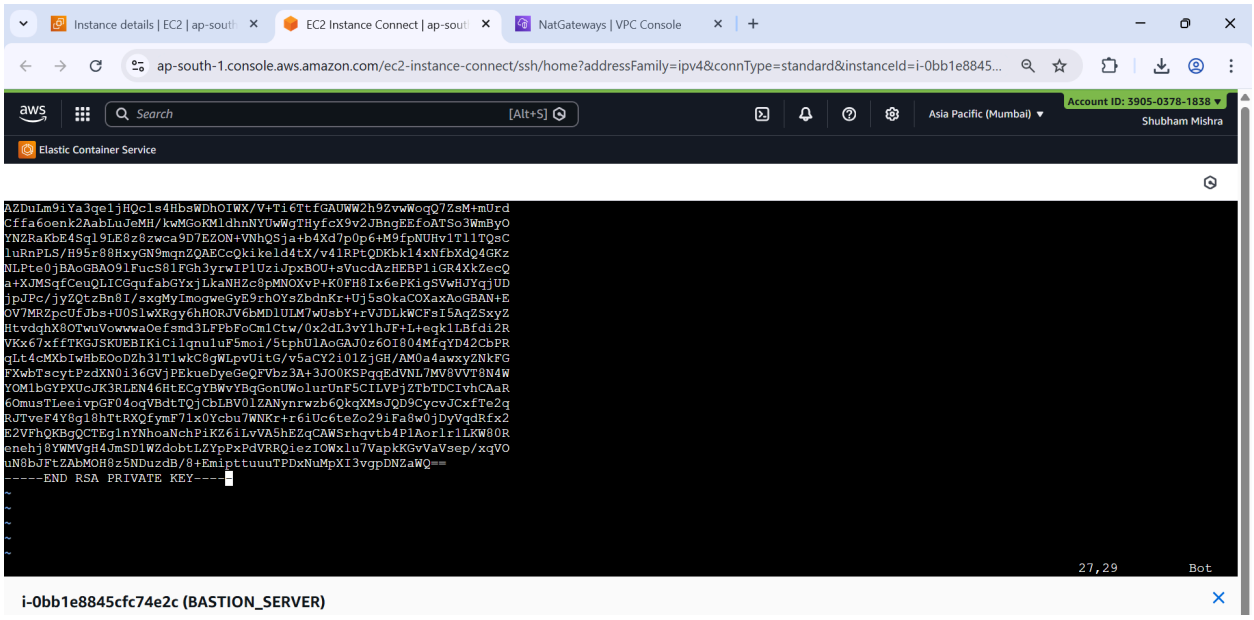
Step 3) Now we will launch bastion server to configure nginx on both servers in private subnets.

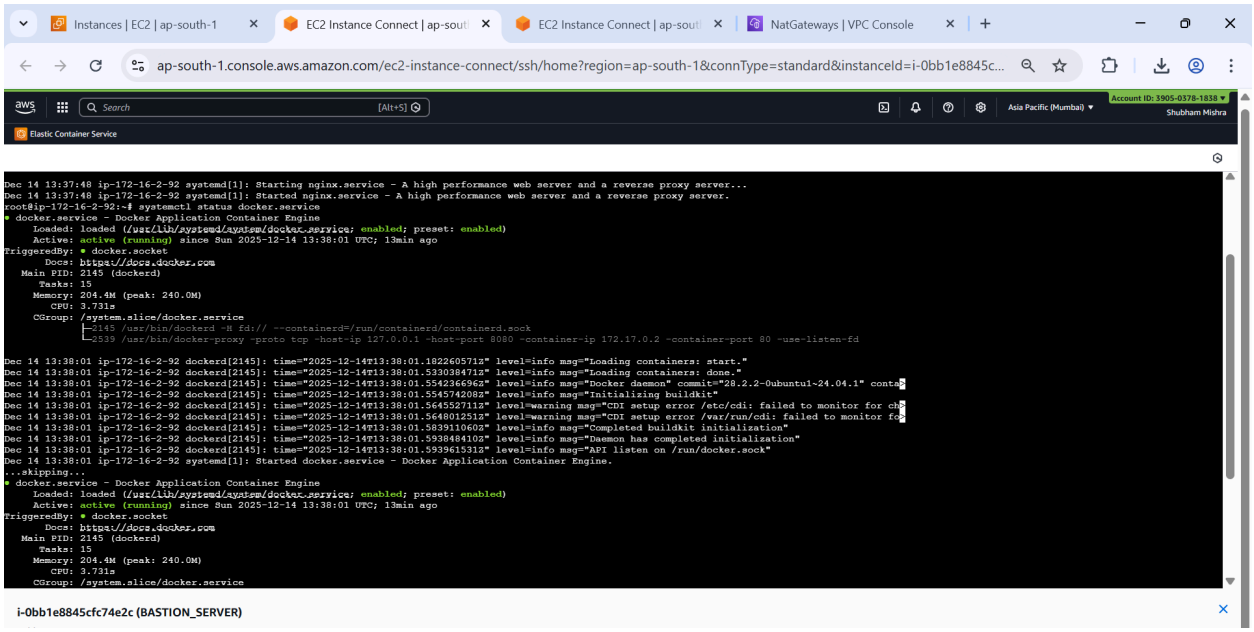

Launch an instance | EC2 | ap-s
NatGateways | VPC Console
ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LaunchInstances:
Name and tags Info
Name
BASTION_SERVER
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI contains the operating system, application server, 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
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Linux
Debian
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Ubuntu Server 24.04 LTS (HVM), SSD Volume Type
Description
Ubuntu Server 24.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).
Canonical, Ubuntu, 24.04, amd64 noble image
Architecture
64-bit (x86)
AMI ID
ami-02b8269d5e85954ef
Publish Date
2025-10-22
Username
ubuntu
Verified provider
Instance type Info | Get advice
Instance type
Summary
Number of instances Info
1
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-02b8269d5e85954ef
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Cancel
Launch instance
Preview code


Launch an instance | EC2 | ap-s
NatGateways | VPC Console
ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LaunchInstances:
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
CLOUDZENIA
Create new key pair
Network settings Info
VPC - required Info
vpc-0cbb91d56e71b4fb7 (CUSTOM_VPC)
172.16.0.0/16
Subnet Info
subnet-0a87fcb6fff130171
ALB_PUBLIC_SUBNET_1
Create new subnet
Auto-assign public IP Info
Enable
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
Security group name - required
bashion_sg
Description - required Info
launch-wizard-2 created 2025-12-14T13:40:18.477Z
Inbound Security Group Rules
Security group rule 1 (TCP, 22, 0.0.0.0/0)
Remove
Type Info
ssh
Protocol Info
TCP
Port range Info
22
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security group
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Summary
Number of instances Info
1
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-02b8269d5e85954ef
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Cancel
Launch instance
Preview code

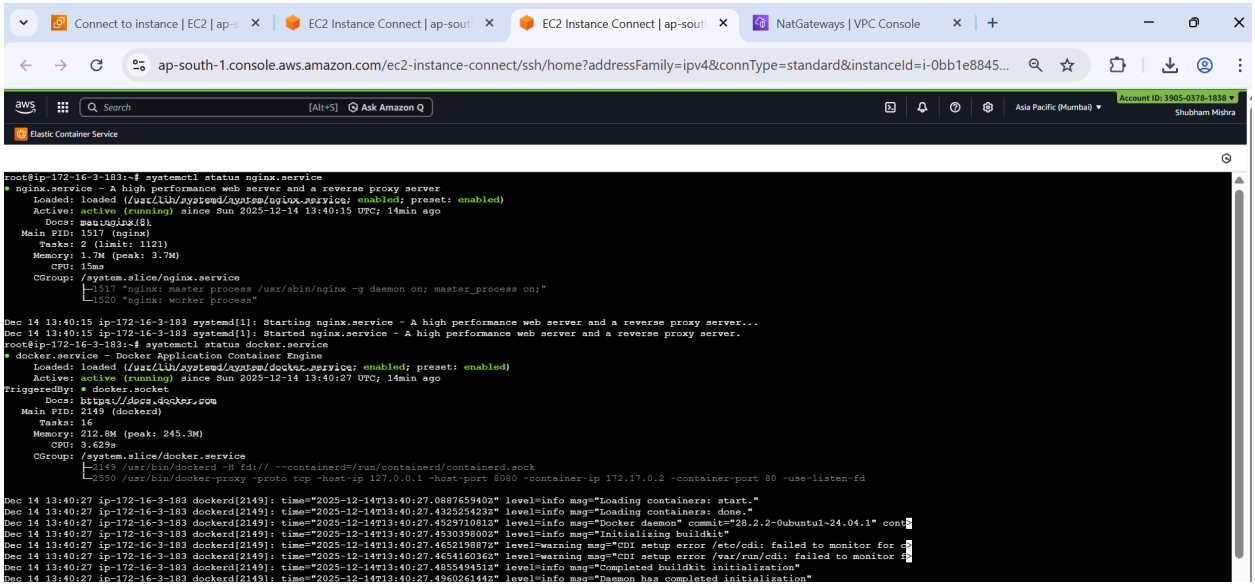


Step 4) Now we will ssh into bashion server and copy pem file to ssh into private servers to configure nginx





Successfull ssh into server 1 and we can see nginx and docker installed successfully.



Successfull ssh into server 2 and we can see nginx and docker installed successfully.

Step 5) Now we will remove default file at location etc/nginx/sites-available and create two files on both private ec2 servers

a) **ec2-instance.conf**

```nginx
server {
    listen 80;
    server_name ec2-instance1.shubham-mishra.online;

    location / {
        return 200 "Hello from Instance\n";
    }
}
```

b) **ec2-docker.conf**

```
server {
    listen 80;
    server_name ec2-docker1.shubham-mishra.online;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

```
root@ip-172-16-2-92:/etc/nginx/sites-available# ls
ec2-docker.conf  ec2-instance.conf
root@ip-172-16-2-92:/etc/nginx/sites-available#
```

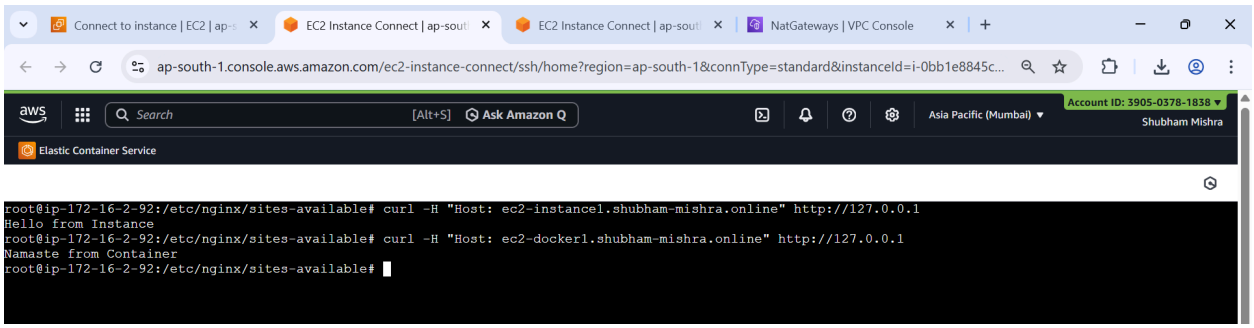
c) **Then run**

```
sudo nginx -t
sudo systemctl reload nginx
```

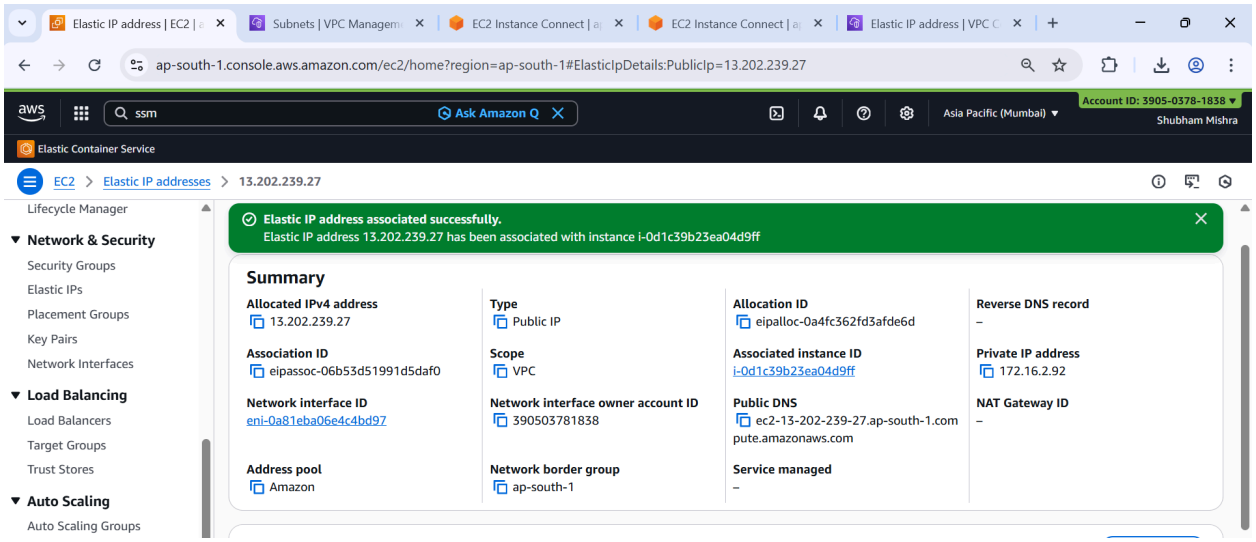
d) **Then**

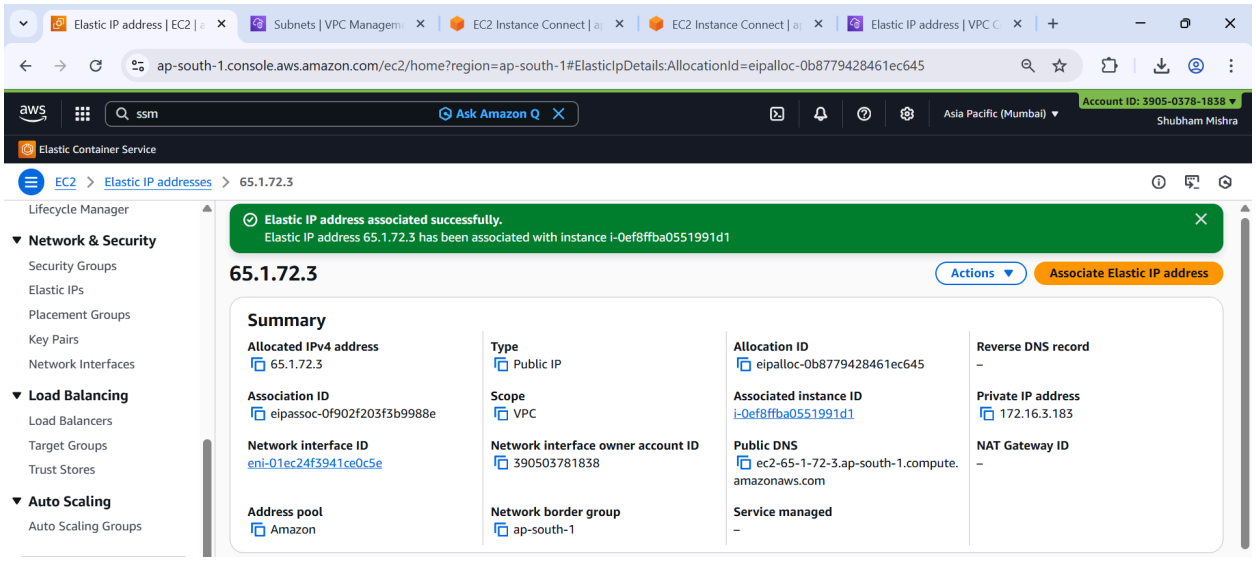
```
curl -H "Host: ec2-instance1.shubham-mishra.online" http://127.0.0.1
curl -H "Host: ec2-docker1.shubham-mishra.online" http://127.0.0.1
```

e) **Got output**

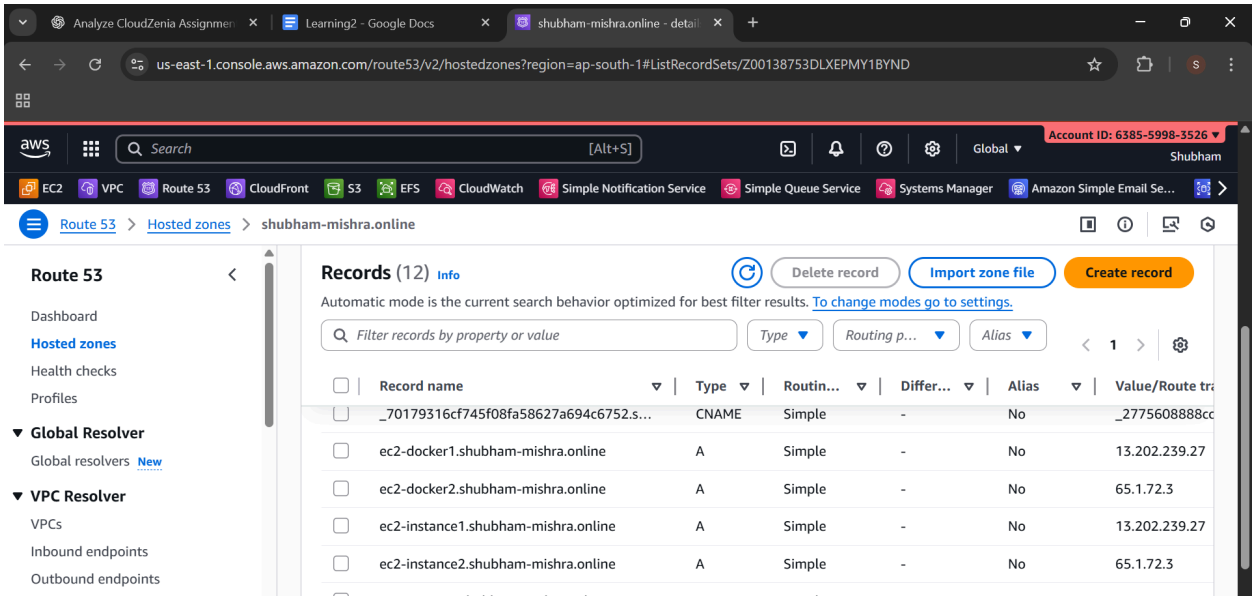


Step 6) Now we will associate elastic ip as per assignment requirement to both private servers

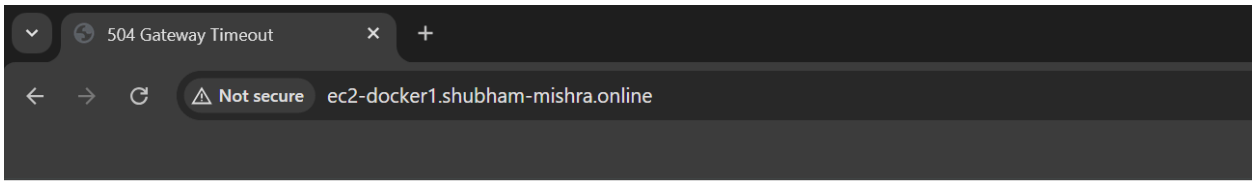




Step 7) Now we will create records in route53 for the servers for domain mapping to elastic ip



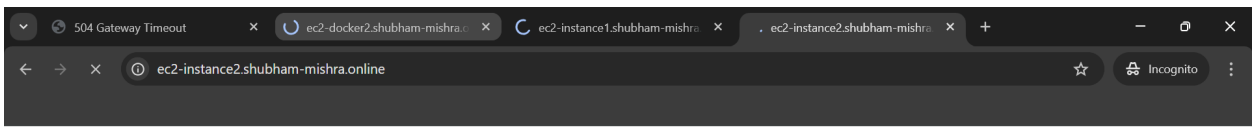
Step 8) Now we will test the record



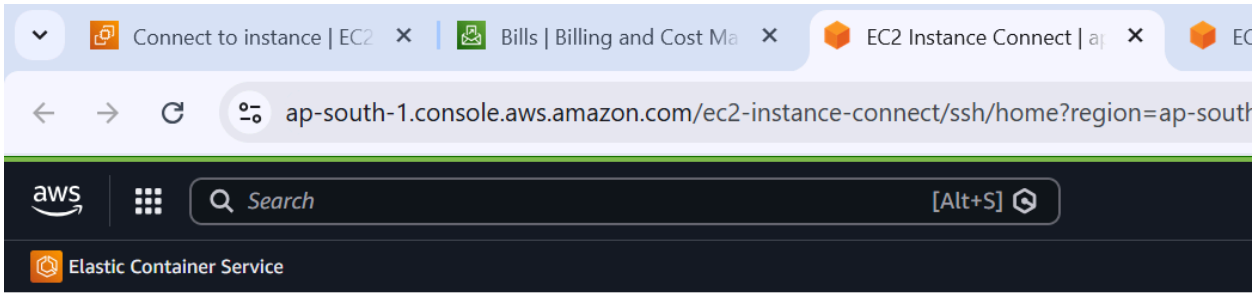
Gateway Timeout

Server error - server 13.202.239.27 is unreachable at this moment.

Please retry the request or contact your administrator.



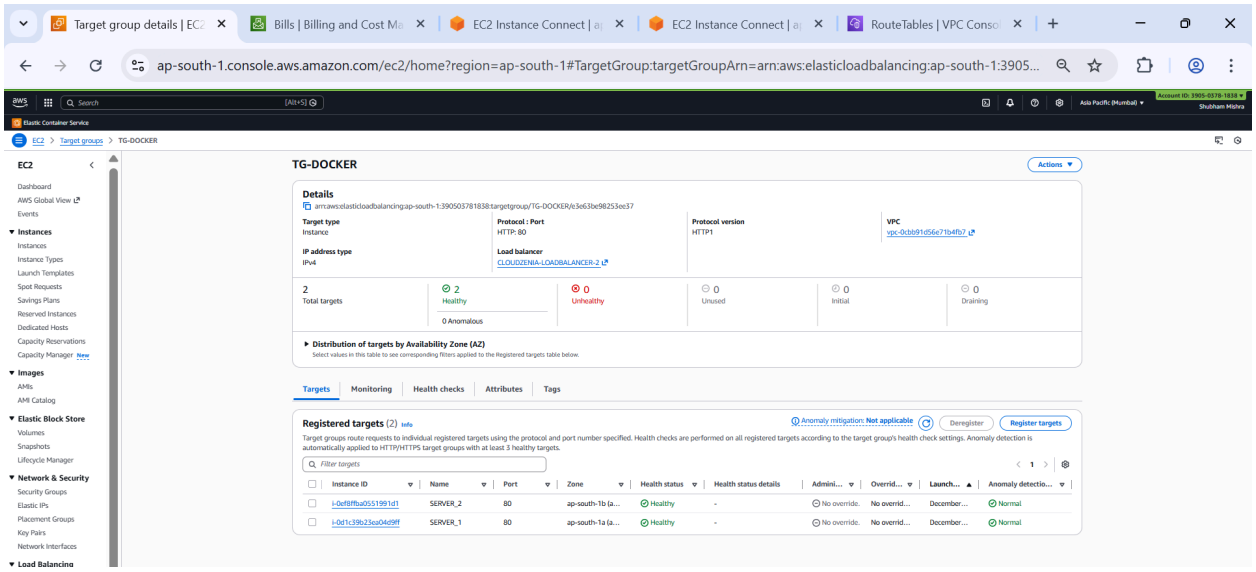
Timeout error as elastic ip needs Internet gateway not NAT gateway so fix is either use Internet gateway or move instances in public subnet
Also modify subnet setting to allow auto assign public ip address
Then working now

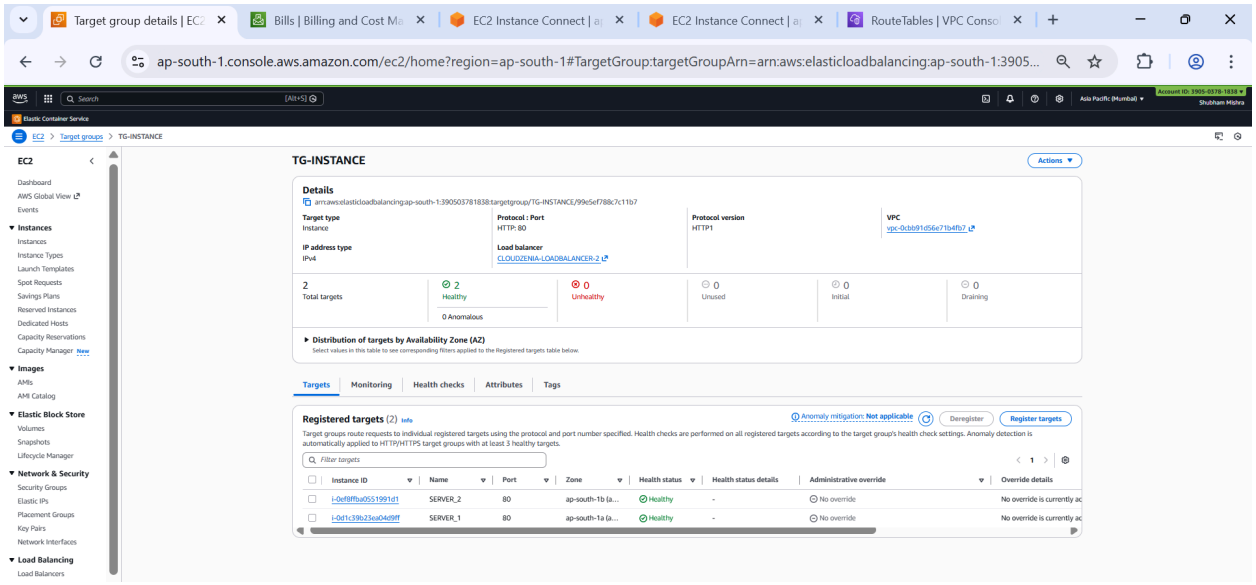



```
root@ip-172-16-0-228:~# curl http://ec2-instance1.shubham-mishra.online/
curl http://ec2-instance2.shubham-mishra.online/
curl http://ec2-docker1.shubham-mishra.online/
curl http://ec2-docker2.shubham-mishra.online/
Hello from Instance
Hello from Instance
Namaste from Container
Namaste from Container
root@ip-172-16-0-228:~#
```

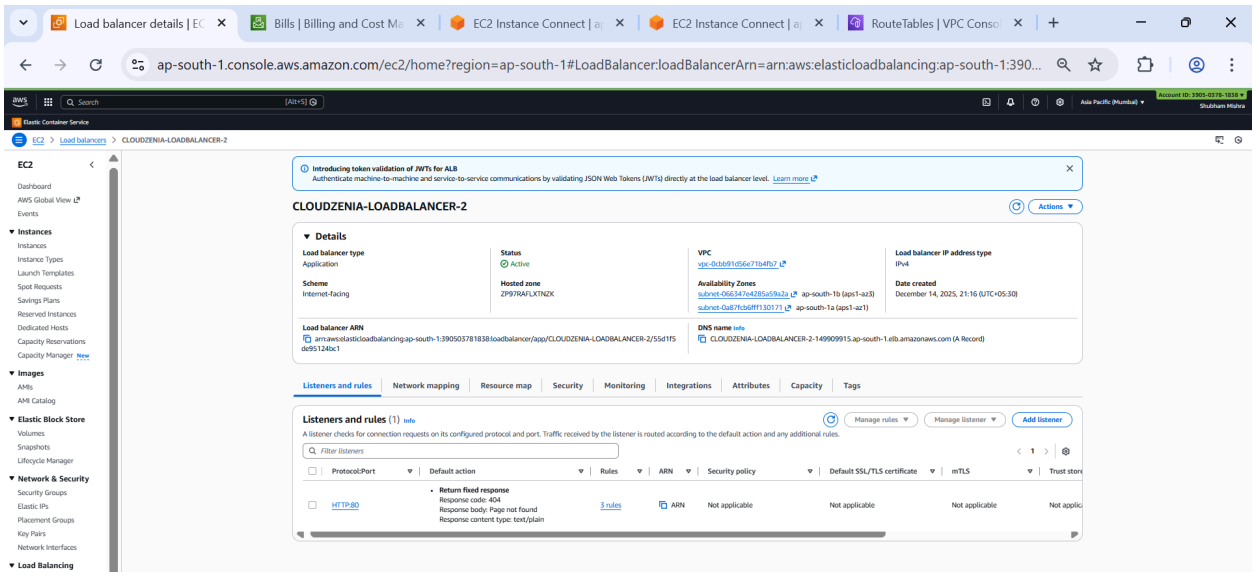
Step 9) Now we will create TG_DOCKER and TG_INSTANCE FOR new ALB

CLOUDZENIA-LOADBALANCER-2

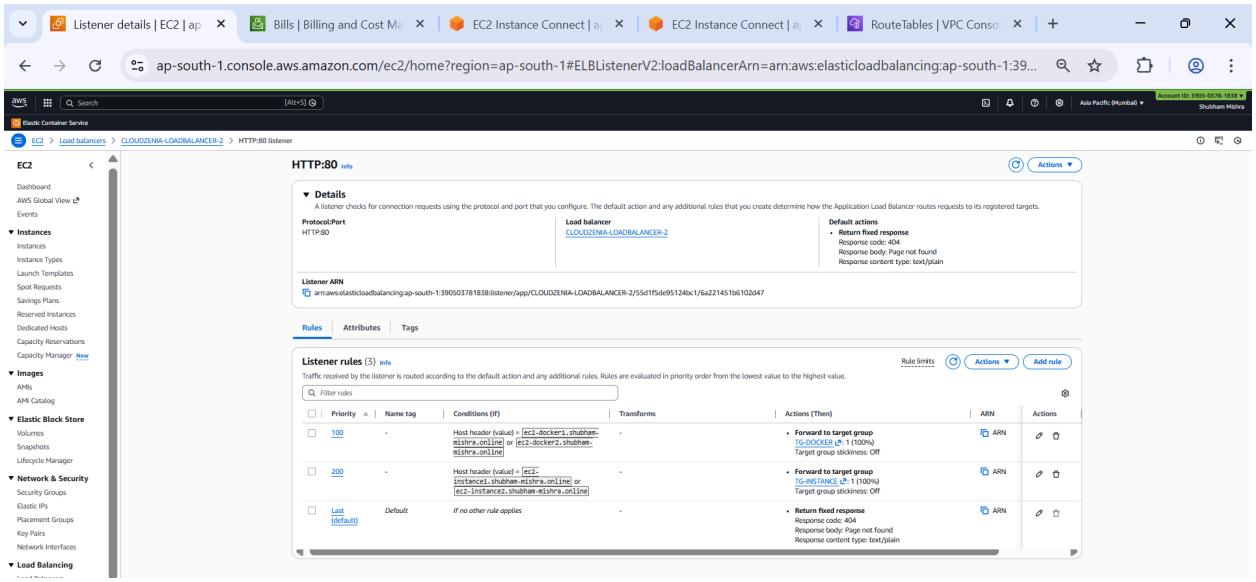




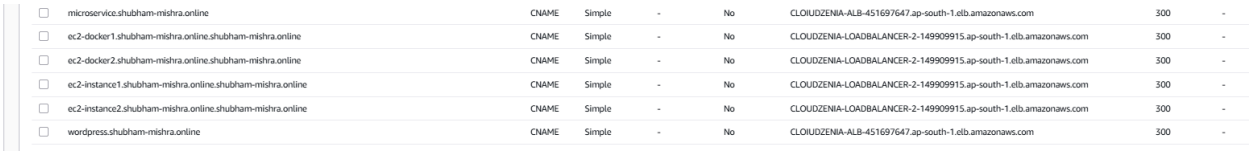
Step 10) Now we will create new ALB



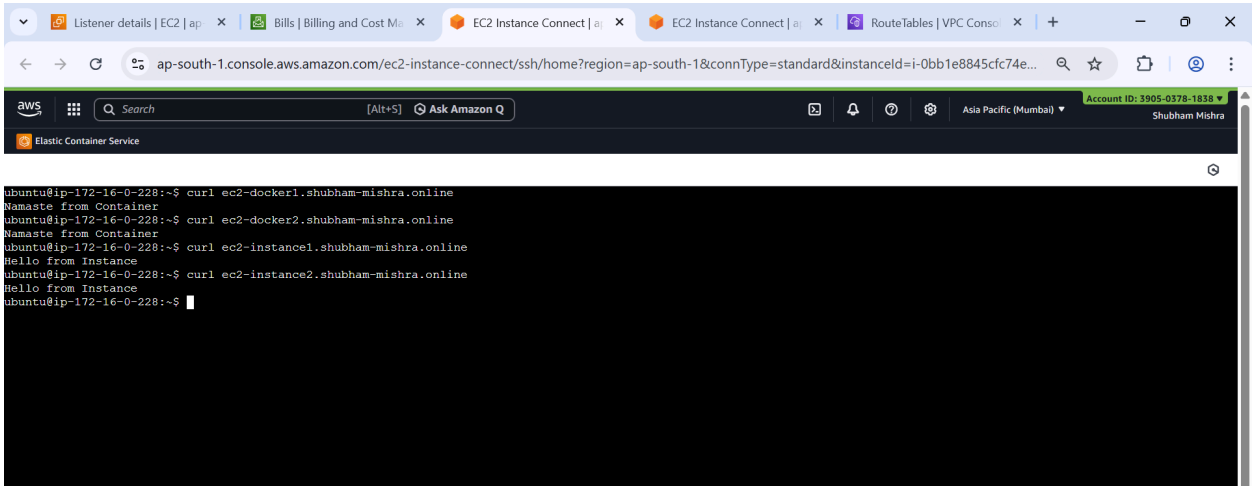
Step 11) Now we will create rules for both TG based on header



Step 12) Now we will create new records in Route 53

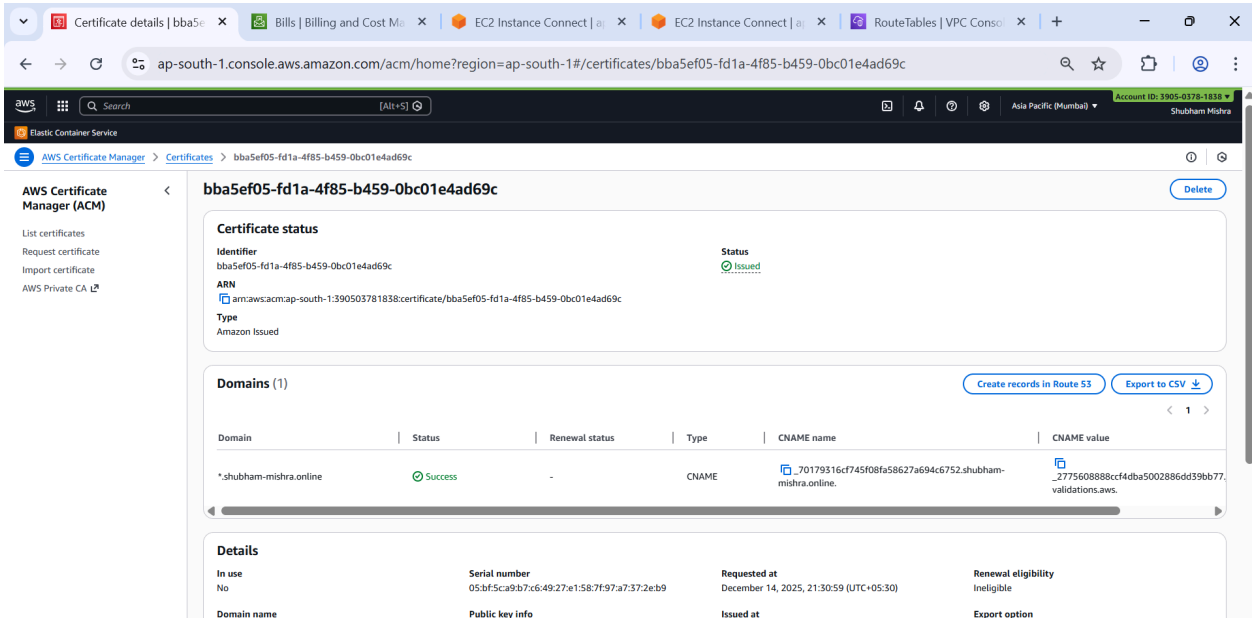


Step 13) Now we will verify new records are working or not

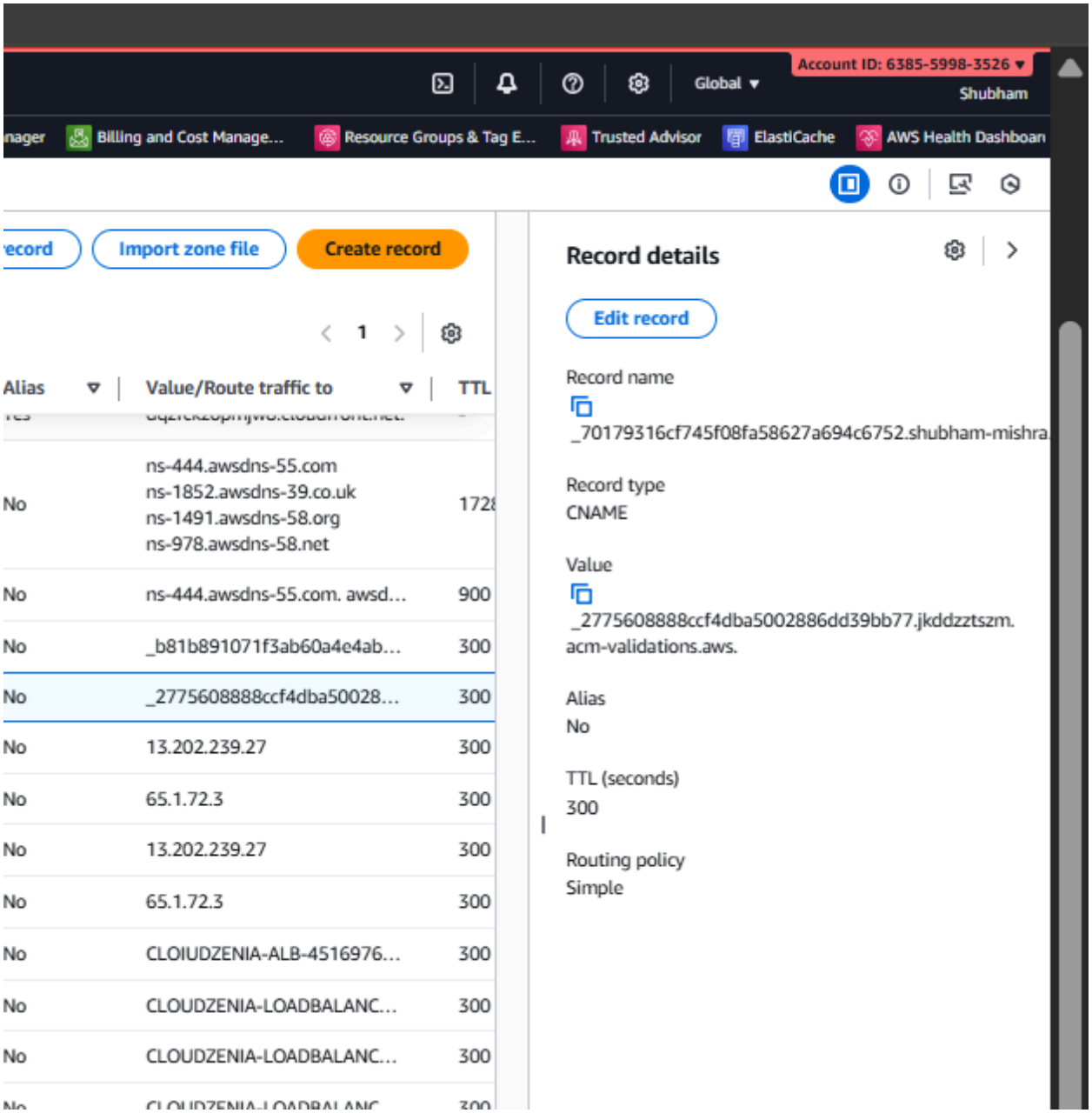


This confirms its working correctly

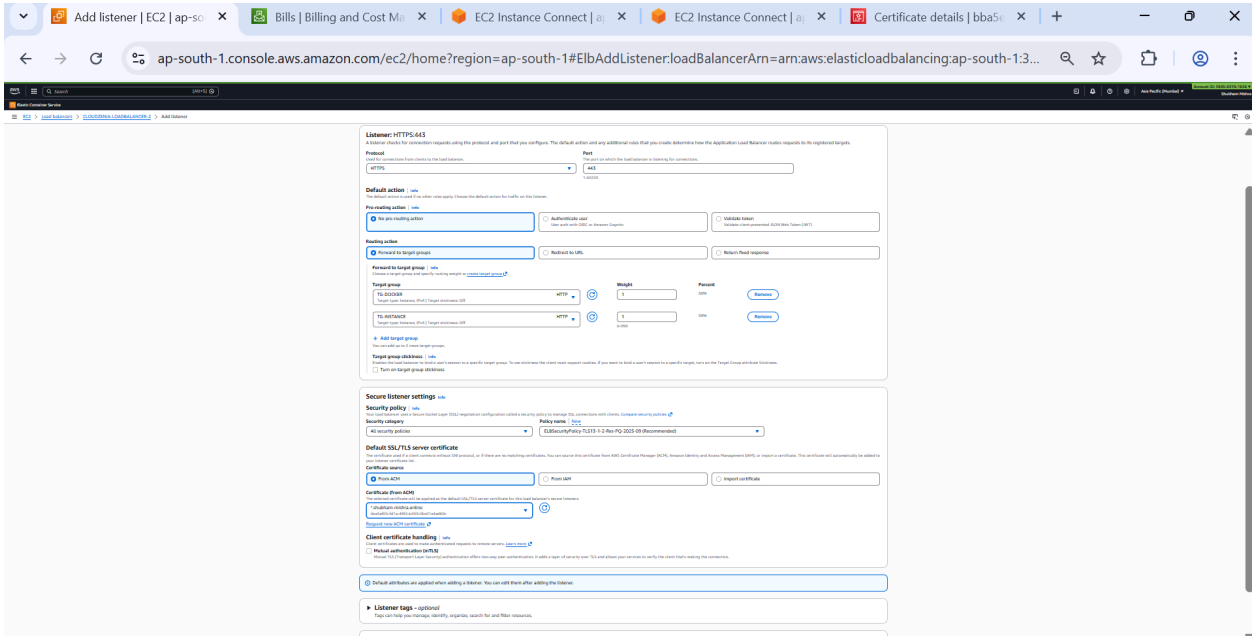
Step 14) Now we will add SSL certificate for this new ALB



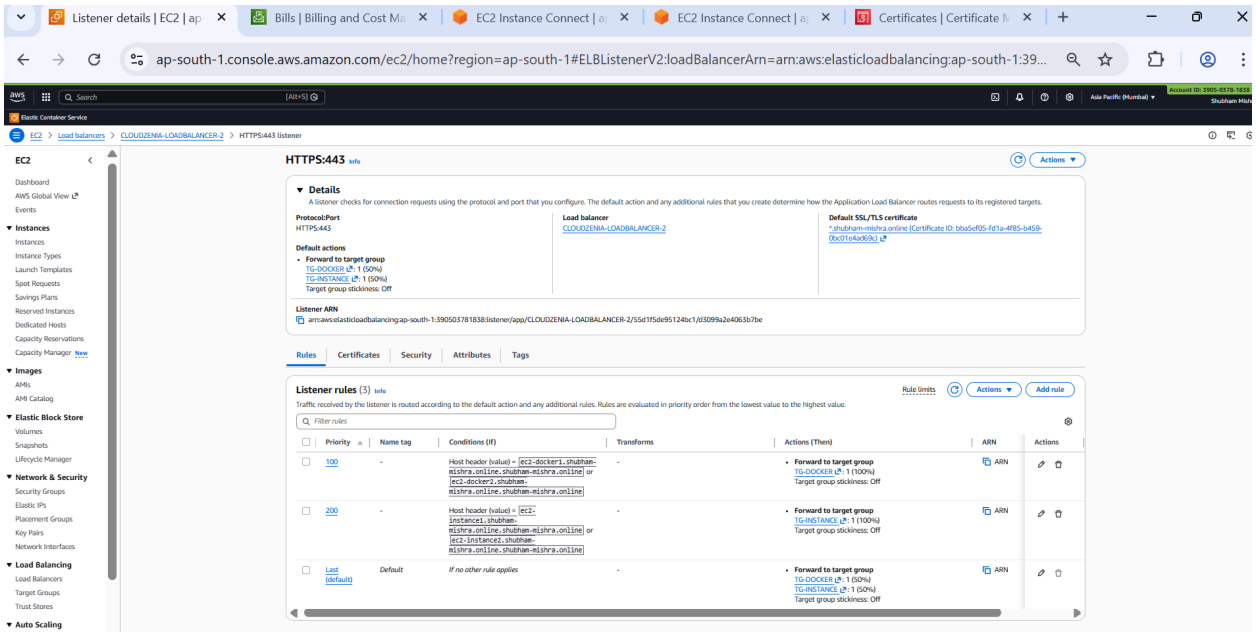
Step 15) New SSL aaded in route53



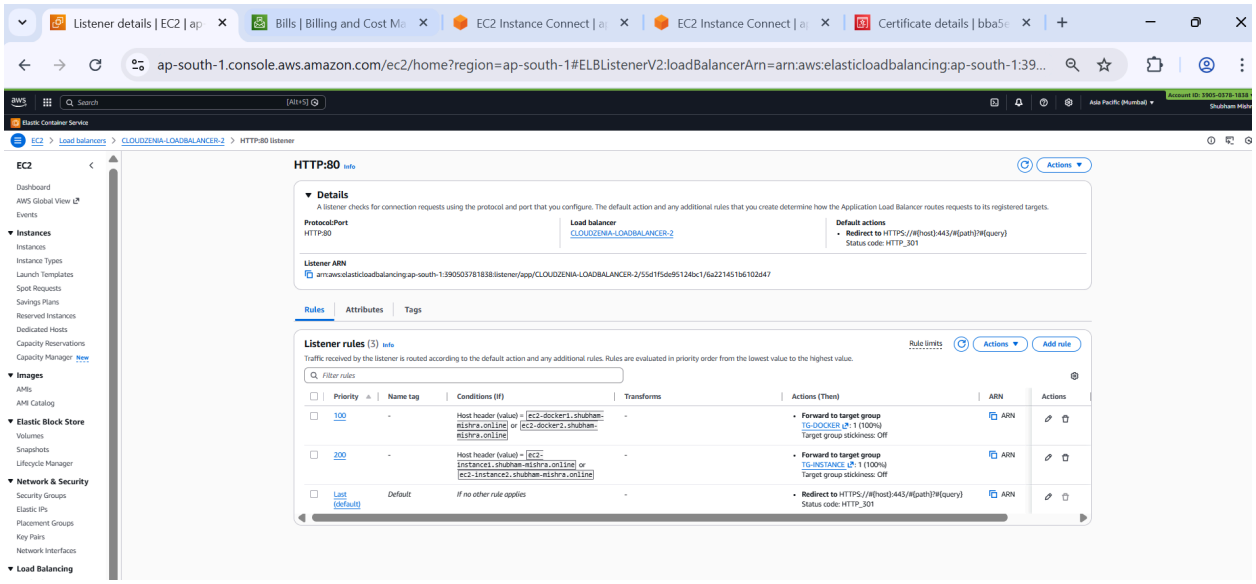
Step 16) Now we will attach this SSL to new ALB



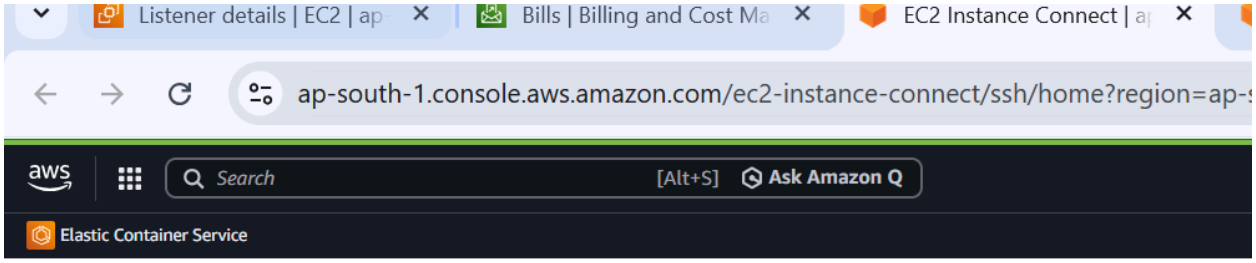
Step 17) Now we will add https rule



Step 18) Now we will redirect all traffic on http to https

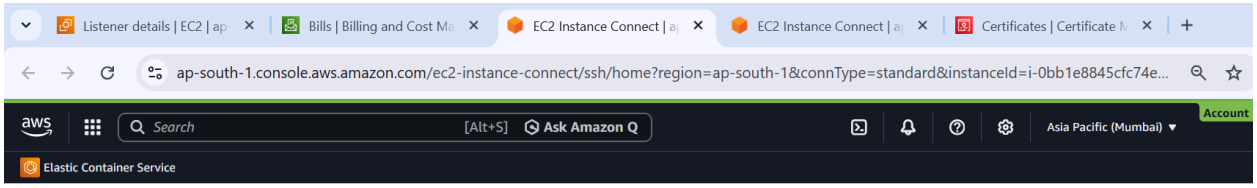


Step 19) This confirms http redirected to https




```
ubuntu@ip-172-16-0-228:~$ curl ec2-docker1.shubham-mishra.online.shubham-mishra.online
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
</body>
</html>
ubuntu@ip-172-16-0-228:~$ curl ec2-docker2.shubham-mishra.online.shubham-mishra.online
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
</body>
</html>
ubuntu@ip-172-16-0-228:~$ curl ec2-instance2.shubham-mishra.online.shubham-mishra.online
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
</body>
</html>
ubuntu@ip-172-16-0-228:~$ curl ec2-instance1.shubham-mishra.online.shubham-mishra.online
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
</body>
</html>
ubuntu@ip-172-16-0-228:~$
```

Step 20) Confirmation setup is successful




```
ubuntu@ip-172-16-0-228:~$ curl ec2-instance2.shubham-mishra.online.shubham-mishra.online
Namaste from Container
ubuntu@ip-172-16-0-228:~$ curl ec2-instance2.shubham-mishra.online
Hello from Instance
ubuntu@ip-172-16-0-228:~$ curl ec2-instance1.shubham-mishra.online
Hello from Instance
ubuntu@ip-172-16-0-228:~$ curl ec2-docker1.shubham-mishra.online
Namaste from Container
ubuntu@ip-172-16-0-228:~$ curl ec2-docker2.shubham-mishra.online
Namaste from Container
```

Multi-Domain Traffic Routing Using AWS ALB, EC2, Docker, and Route 53

Overview

Designed and implemented a highly available, domain-based routing architecture on AWS that cleanly separates EC2-hosted services and Docker-based microservices using Application Load Balancers and Route 53 DNS.

Key Outcomes Achieved

1. **Domain-Based Traffic Segregation**
    - Successfully routed multiple custom domains to different backend workloads.
    - EC2 instance domains returned **“Hello from Instance”**.
    - Docker container domains returned **“Namaste from Container”**.
    - Ensured deterministic routing using ALB listener rules and host headers.
2. **Load Balancer Integration with Route 53**
    - Integrated Application Load Balancers with Route 53 using **A (Alias) records**.
    - Eliminated direct Elastic IP exposure, enabling AWS-managed, scalable traffic handling.
    - Achieved seamless DNS resolution across multiple subdomains.
3. **Secure HTTPS Enablement**
    - Implemented TLS termination at the ALB using **AWS Certificate Manager (ACM)**.
    - Enabled HTTP-to-HTTPS redirection (301) to enforce secure access.
    - Used wildcard certificates to support multiple subdomains with a single certificate.
4. **Backend Isolation and Clean Architecture**

    - Separated EC2 and Docker workloads using **dedicated target groups** and **separate ALBs** for simplicity.
    - Prevented accidental traffic leakage by explicitly defining routing rules and server blocks.
    - Ensured Docker services were isolated on internal ports and accessed only via NGINX proxying.
5. **High Availability and Observability Readiness**
    - Configured health checks for all target groups to ensure fault-tolerant routing.
    - Designed the setup to support future enhancements such as Auto Scaling, WAF, and CloudFront.
    - Validated end-to-end flow using curl, DNS lookups, and TLS handshake verification.

Technologies Used

- **AWS Services:** EC2, Application Load Balancer (ALB), Route 53, AWS Certificate Manager (ACM)
- **Containers:** Docker, NGINX
- **Networking:** HTTP/HTTPS, TLS, DNS, Host-based routing
- **OS:** Linux (Ubuntu)

Final Result

Delivered a **secure, scalable, and interview-ready AWS architecture** demonstrating real-world troubleshooting, DNS and load-balancer integration, HTTPS configuration, and multi-service routing—aligned with production best practices.
