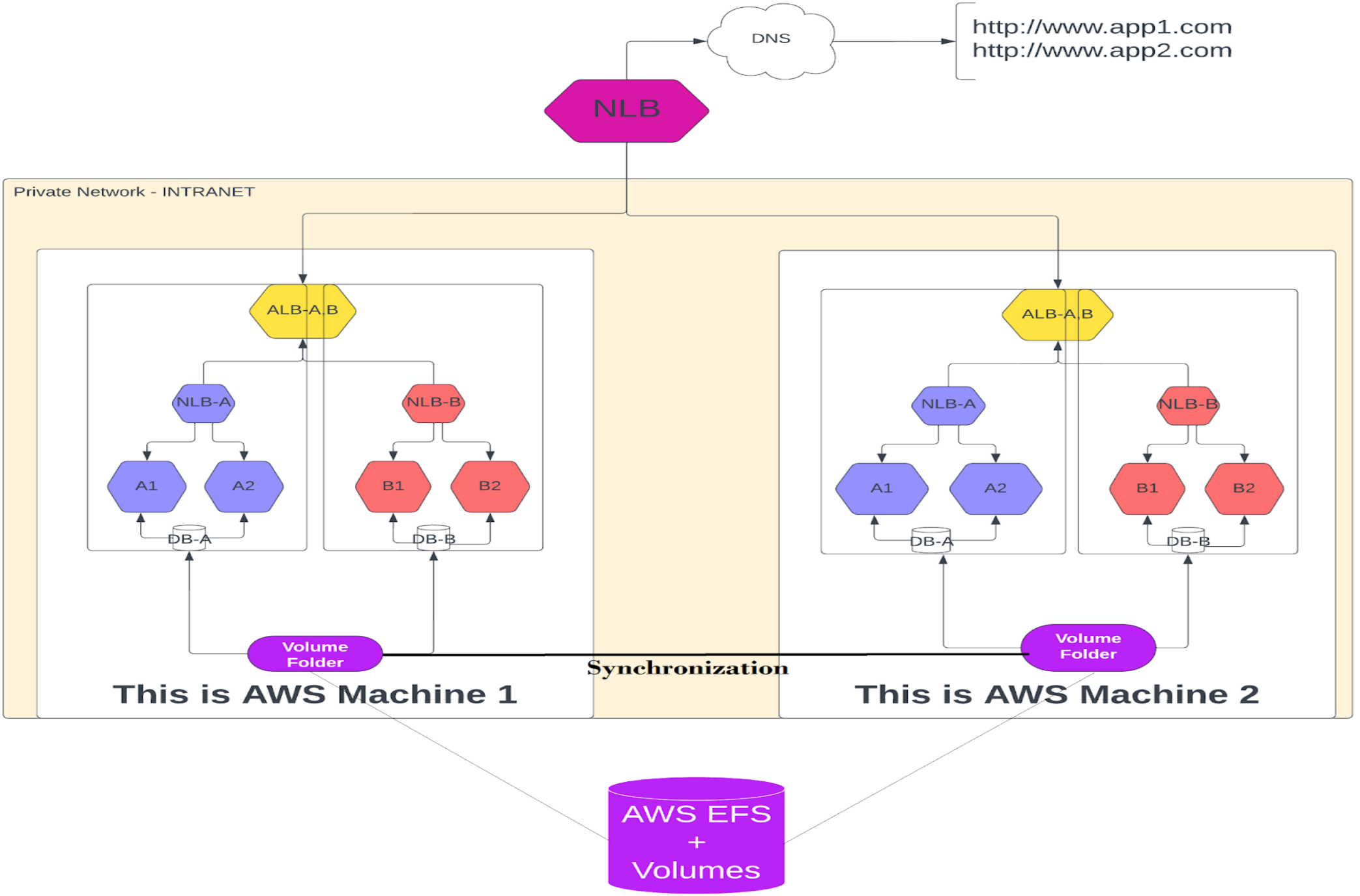
This is Micro-Container Architecture Based

DOCKER PROJECT

Rakesh Kumar Jangid

Date: 20-06-2023


DNS
http://www.app1.com
http://www.app2.com
NLB
Private Network - INTRANET
ALB-A,B
NLB-A
NLB-B
A1
A2
B1
B2
DB-A
DB-B
Volume Folder
Synchronization
This is AWS Machine 1
ALB-A,B
NLB-A
NLB-B
A1
A2
B1
B2
DB-A
DB-B
Volume Folder
This is AWS Machine 2
AWS EFS
+
Volumes

☐ INTRODUCTION

★ We are embarking on a project to create a Decoupled Microservice-based system using Docker containers. This project involves several key components and resources that we will need to set up. Here's an overview of what we'll be working on:

★ Computing Resources: We will require sufficient computing resources, such as servers or virtual machines, to host our Docker containers. These resources should have enough processing power, memory, and storage capacity to handle the workload of our microservices.

★ Load Balancers: To distribute incoming network traffic across multiple containers and ensure high availability and scalability, we will incorporate load balancers into our system. Load balancers help evenly distribute requests and optimise the utilization of resources.

★ External Volume Storage: For persistent data storage, we'll need to set up external volume storage. This allows our containers to store and retrieve data even if they are restarted or moved between different hosts. We'll configure Docker volumes or use external storage solutions like network-attached storage (NAS) or cloud-based storage services.

★ MySQL: As a critical component of many web applications, we'll include a MySQL database in our system. This will provide a reliable and scalable solution for storing and managing our application's data. We'll set up a MySQL container and configure it to ensure proper data persistence and security.

★ WordPress Images for Containers: If our project involves hosting WordPress websites, we'll need Docker images specifically built for WordPress. These images contain all the necessary dependencies and configurations to run WordPress within a containerized environment.

★ Configuration and Settings: Throughout the project, we'll need to define various configurations and settings for our microservices, load balancers, storage, and database. These configurations will include network settings, environment variables, security parameters, and other relevant parameters required for proper operation.

★ Pre-System Setup: Before we can start building our project, we may need to perform some pre-system setup tasks. This could involve installing and configuring Docker on our computing resources, setting up the networking environment, ensuring firewall rules allow necessary communication, and other prerequisites.

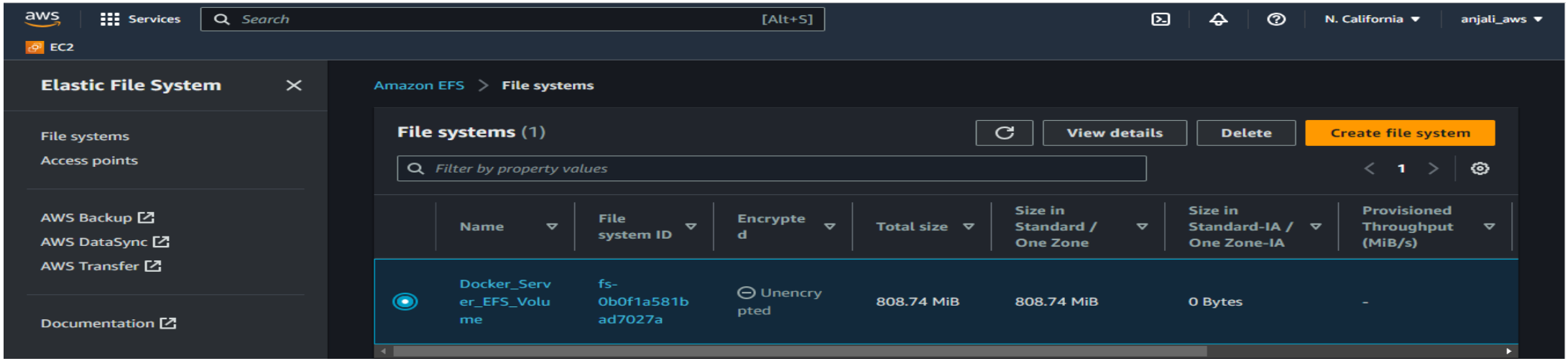
★ We will proceed with this project by following a step-by-step approach. Each step will involve specific tasks, such as setting up computing resources, configuring load balancers, creating Docker volumes, deploying MySQL and WordPress containers, and managing configurations. By completing each step, we'll gradually build our decoupled microservice-based system using Docker containers.

☐ CREATE CLOUD INFRASTRUCTURE RESOURCES INCLUDING :- EC2 INSTANCES, EFS VOLUME, ELASTIC-IP, NLB

❖ PRE- SYSTEM CONFIGURATION

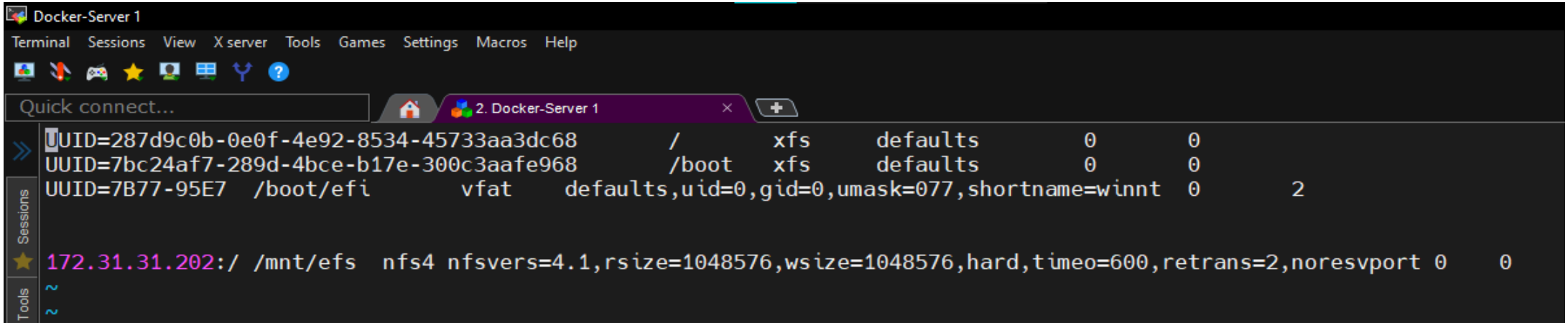
Name	Configuration & Setup
Docker-Server1 & Docker-Server2	edit hosts file with all connected instances ip setup repository on /etc/yum.repos.d/online.repo setup EPEL repository (if system os is fedora based) set hostname

❖ Mount EFS volumes between Docker-Server1 and Docker-Server2 EC2 instances :-




```
# mkdir -p  /mnt/efs
# yum install nfs-utils
# systemctl enable nfs-client.target
# systemctl is-enabled nfs-client.target
# mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,
  noresvport 172.31.31.202:/     /mnt/efs
# vi /etc/fstab
  172.31.31.202:/     /mnt/efs   nfs4 nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,
  retrans=2,noresvport
# mount -a
# df -hT
# mount  | grep /mnt/efs
```

——----------Note:- Ping both server instances and make ssh connection with passwordless login—----------



❖ Work on Docker-Server1 & 2

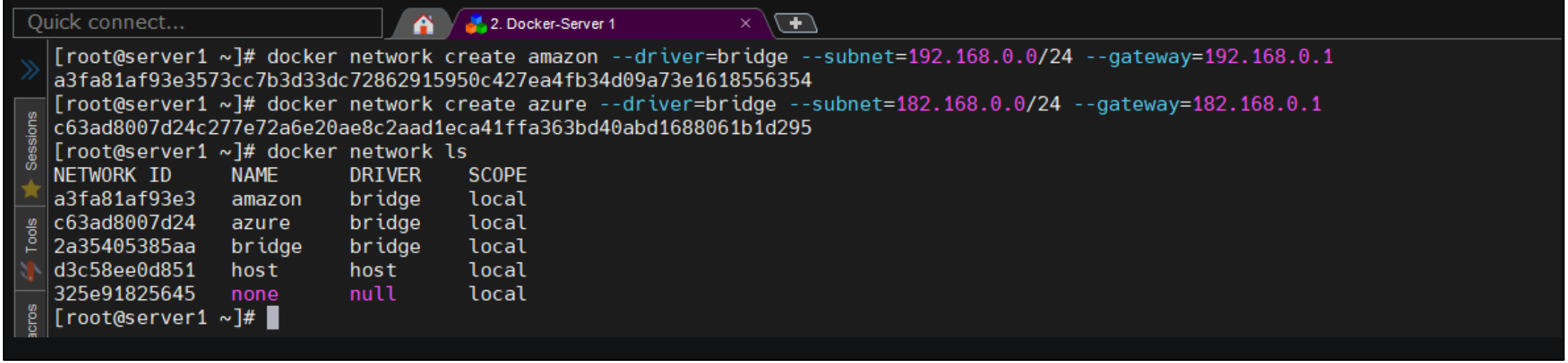
➢ Docker Setup on EC2-Instances

```
# yum -y install git
# git –version
#-------> For fedora &  Debian Based Linux Version:-
# git clone https://github.com/rakesh08061994/docker.git
# cd docker
# chmod a+x Docker_Setup_On_Debian.sh
# chmod a+x Docker_Setup_on_fedora.sh
# ./Docker_Setup_on_fedora.sh
# ./Docker_Setup_On_Debian.sh
# systemctl restart docker
# systemctl enable docker
```

```
# systemctl is-enabled docker
# docker info
# docker –version
```

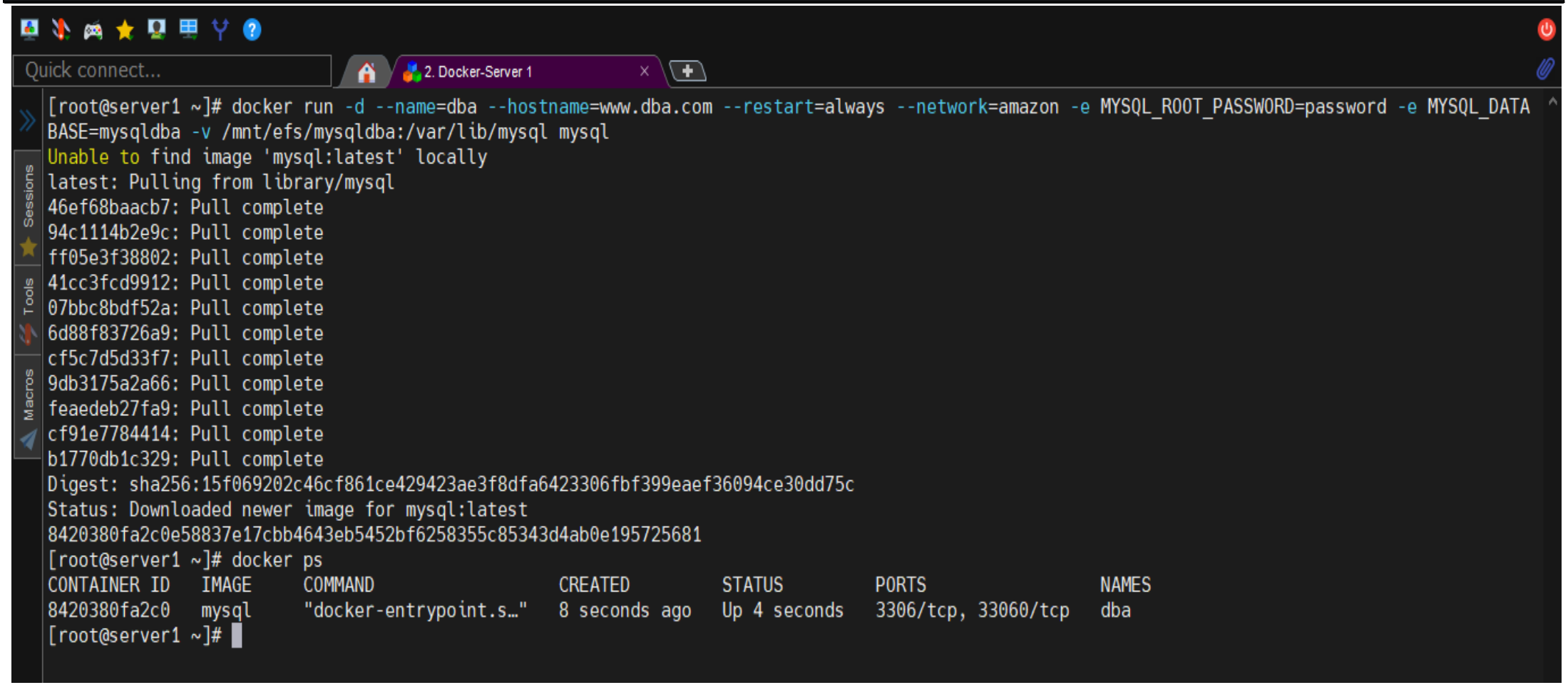
➢ Create Network Bridge for application isolation (Server1 & 2)

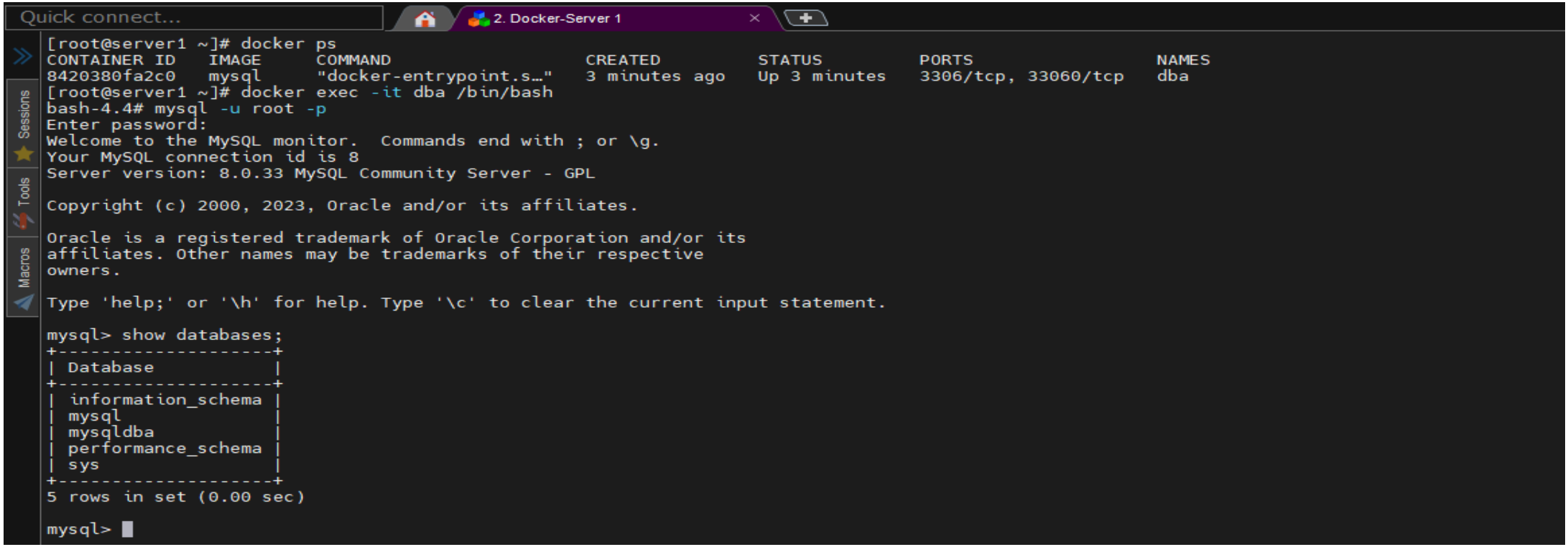
```
# docker network ls
# docker network create --help
# docker network create amazon --driver=bridge --subnet=192.168.0.0/24 --gateway=192.168.0.1
# docker network create azure --driver=bridge --subnet=182.168.0.0/24 --gateway=182.168.0.1
# docker network ls
# docker network inspect amazon
# docker network inspect azure
```



➢ Create MYSQL Database Container (Server1 & 2)

```
# docker run -d --name=dba --hostname=www.dba.com --restart=always   --network=amazon -e
   MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=mysqldba -v
   /mnt/efs/mysqldba1:/var/lib/mysql mysql
docker run -d --name=dba --hostname=www.dba.com --restart=always   --network=amazon -e
   MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=mysqldba -v
   /mnt/efs/mysqldba2:/var/lib/mysql mysql
```





➢ SYNCHRONISATION REAL TIME MYSQL DATABASE BETWEEN TWO SYSTEMS.

To achieve real-time synchronization of MySQL and WordPress data between two systems, you can consider implementing database replication with the following steps:

- Configure MySQL replication: MySQL provides built-in replication features that allow you to replicate data between multiple database servers. You'll need to set up one server as the master and the other as the replica.
  a. On the master machine, enable binary logging in the MySQL configuration file (my.cnf) by adding the "log_bin = mysql-bin" line.
- b. Create a replication user on the master machine with appropriate permissions for replication.
  c. On the replica machine, configure the MySQL replication settings by editing the my.cnf file. Set the replica's unique server ID and configure the replication connection details, such as the master host, user, and password.
  d. Start the MySQL replication process on the replica machine, connecting it to the master server. The replica will start syncing data from the master in real-time.

- ❖ Update WordPress configuration: In the WordPress installation on both machines, you'll need to update the wp-config.php file to use the appropriate database connection settings for each server.
  a. Locate the wp-config.php file in the WordPress installation directory.
  b. Update the database connection details (host, database name, username, password) to match the respective MySQL server on each machine.
- ❖ Configure the EFS for file synchronization: Since you have already mounted an EFS on both machines, you can use it to synchronize the WordPress files between the two systems.
  a. Ensure that the WordPress files and directories are stored on the EFS mount point.
  b. Modify the wp-config.php file on both machines to use the EFS path for the WP_CONTENT_DIR and WP_CONTENT_URL constants.
  c. This setup will ensure that both instances share the same WordPress files stored on the EFS, allowing changes made on one system to be immediately reflected on the other.
- ❖ With these steps, you will have MySQL database replication set up between the two machines, ensuring real-time synchronization of data. Additionally, the shared EFS will enable both systems to use the same WordPress files, eliminating discrepancies between the instances.
- ❖ Remember to thoroughly test the setup and ensure that both machines are properly configured before deploying it to a production environment.

Certainly! Here's a step-by-step guide to implementing real-time synchronization of MySQL and WordPress data between two machines using database replication and an EFS:

Prerequisites:

- Two EC2 instances (Machine A and Machine B) with WordPress and MySQL installed.
- An EFS file system mounted on both machines.

To synchronize the MySQL databases and WordPress data between two Linux servers for running two web applications, you can follow the detailed steps outlined below:

Set up the target server: Ensure that the target server has a web server (such as Apache or Nginx) installed, along with PHP and MySQL. Install WordPress on the target server but do not configure it yet.Configure the source server: Make sure that the source

server is properly configured and running the WordPress site with the desired data. Verify that the source server has a working MySQL database containing the necessary tables and records.Install and configure MySQL replication: MySQL replication is a mechanism that allows you to synchronize data between two servers. Set up MySQL replication between the source and target servers by following these steps:
a. On the source server, open the MySQL configuration file **(my.cnf or my.ini)** and enable the replication settings by adding the following lines:

```
# docker ps
# docker cp dba:/etc/my.cnf .
# vi my.cnf
[mysqld]
server-id = 1
log_bin = /var/log/mysql/mysql-bin.log
binlog_do_db = mysqldba
```

b. Restart the MySQL service on the source server for the changes to take effect.

Log in to the MySQL shell and create a replication user with the necessary privileges. Replace <replication_user> and <password> with your desired values:

```
# docker exec -it dba /bin/bash
bash-4.4# mysql -u root -p
Password
mysql> CREATE USER 'repl'@'%' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%';
```

```
mysql> FLUSH PRIVILEGES;
```

In the MySQL shell, run the following command and note down the values of Master_Log_File and Exec_Master_Log_Pos:

```
mysql> SHOW MASTER STATUS;
```

c. On the target server, open the MySQL configuration file and add the following lines:

```
server-id = 2
replicate-do-db = mysqldba
# docker cp ./my.cnf dba:/etc/my.cnf
```

Restart the MySQL service to apply the changes.

Log in to the MySQL shell and run the following command, replacing the placeholders with the values from the master server:

(On server2)

```
# docker exec -it dba /bin/bash
bash-4.4# mysql -u root -p
PasswordCHANGE
MASTER TO MASTER_HOST='<master_server_ip>', MASTER_USER='<replication_user>',
MASTER_PASSWORD='<password>', MASTER_LOG_FILE='<log_file>', MASTER_LOG_POS=<log_pos>;
```

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+-------------
-----+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gt
_Set |
+------------------+----------+--------------+------------------+-------------
-----+
| mysql-bin.000001 |      157 | mysqldba     |                  |
     |
+------------------+----------+--------------+------------------+-------------
```

d. Restart the MySQL service on the target server.
e. Log in to the MySQL server on the target server and execute the following command to set up replication:

CHANGE MASTER TO MASTER_HOST='[source_server_ip]', MASTER_USER='[replication_user]', MASTER_PASSWORD='[replication_password]', MASTER_LOG_FILE='[log_file_name]', MASTER_LOG_POS=[log_position];

Replace [source_server_ip] with the IP address of the source server, [replication_user] and [replication_password] with the MySQL replication user credentials, and [log_file_name] and [log_position] with the details from the source server's binary log.

f. Start the replication on the target server with the following command:

```
START SLAVE;
```

g. Verify that replication is working by checking the replication status on the target server:

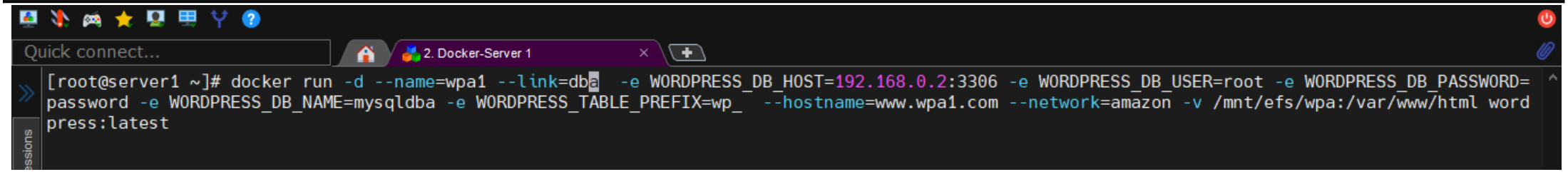
```
SHOW SLAVE STATUS\G
```

Ensure that the Slave_IO_Running and Slave_SQL_Running values are both set to Yes.

```
# docker restart dba
# docker exec -it dba /bin/bash
Mysql> cat /etc/my.cnf
```

➢ **Create WP Container (On server1 & 2 with attaching dba) & do same with dbb and there wp containers on both server1 & 2**

```
# docker run -d --name=wpa1 --link=dba  -e WORDPRESS_DB_HOST=192.168.0.2:3306 -e
WORDPRESS_DB_USER=root -e WORDPRESS_DB_PASSWORD=password -e
WORDPRESS_DB_NAME=mysqldba -e WORDPRESS_TABLE_PREFIX=wp_
--hostname=www.wpa1.com --network=amazon -v /mnt/efs/wpa:/var/www/html wordpress:latest
# docker run -d --name=wpa2 --link=dba  -e WORDPRESS_DB_HOST=192.168.0.2:3306 -e
WORDPRESS_DB_USER=root -e WORDPRESS_DB_PASSWORD=password -e
WORDPRESS_DB_NAME=mysqldba -e WORDPRESS_TABLE_PREFIX=wp_
--hostname=www.wpa2.com --network=amazon -v /mnt/efs/wpa:/var/www/html wordpress:latest
```



For wordpress content we will use shared EFS attached volume folders for both servers

➢ **Create Network Load Balancer (NLB) and attach Target IP over wp containers**

```
# docker run -d --name=nlba -v /mnt/efs/nlba:/usr/local/etc/haproxy -p 4000:80 --network=amazon -u root
haproxy
# docker run -d --name=nlbb -v /mnt/efs/nlbb:/usr/local/etc/haproxy -p 4001:80 --network=azure -u root
haproxy
```

```
# vi haproxy.cfg
#vim /haproxy/haproxy.cfg
global
      daemon
      maxconn 1024
       pidfile /var/run/haproxy.pid
       user root
       group root
defaults
       balance roundrobin
       timeout client 60s
       timeout connect 60s
       timeout server 60s
frontend haproxy_server
       bind *:80
       default_backend web_server
backend web_server
      balance roundrobin
      server server1 $web1_container_ip:80
      server server2 $web2_container_ip:80
---imp note for this file--------
find web1 and web2 container ip from docker
```

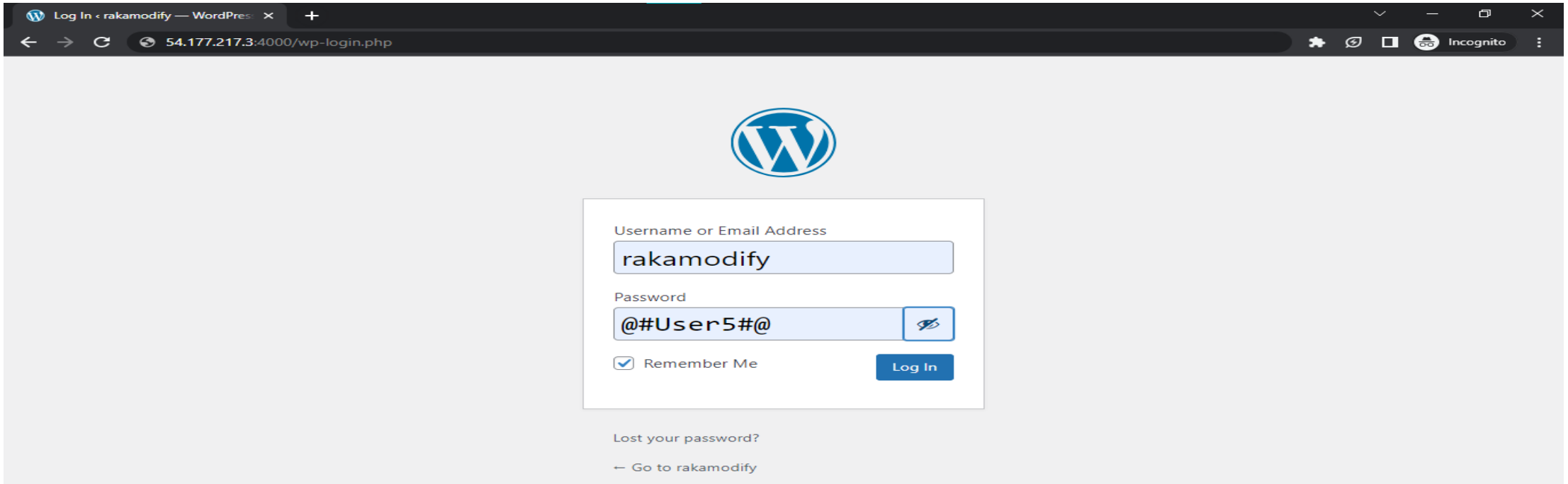
then replace the $web1_container_IP from its actual ip of wpa1 & wpb1

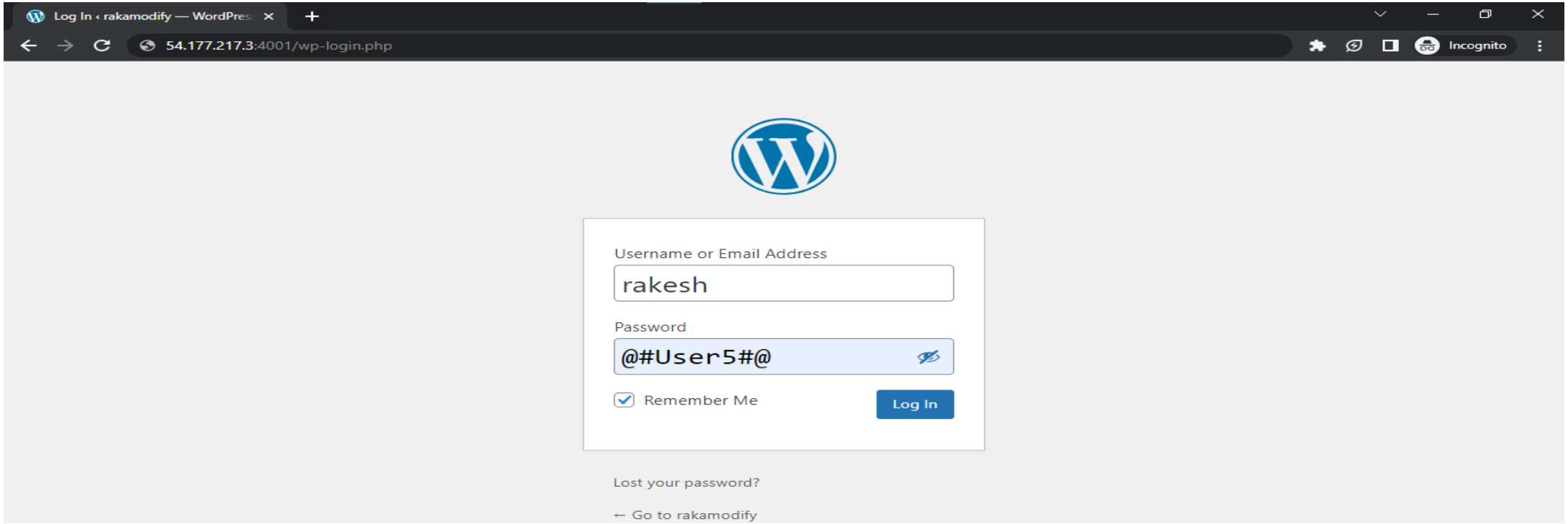
and replace the $web2_container_IP from its actual ip of wpa1 & wpb1

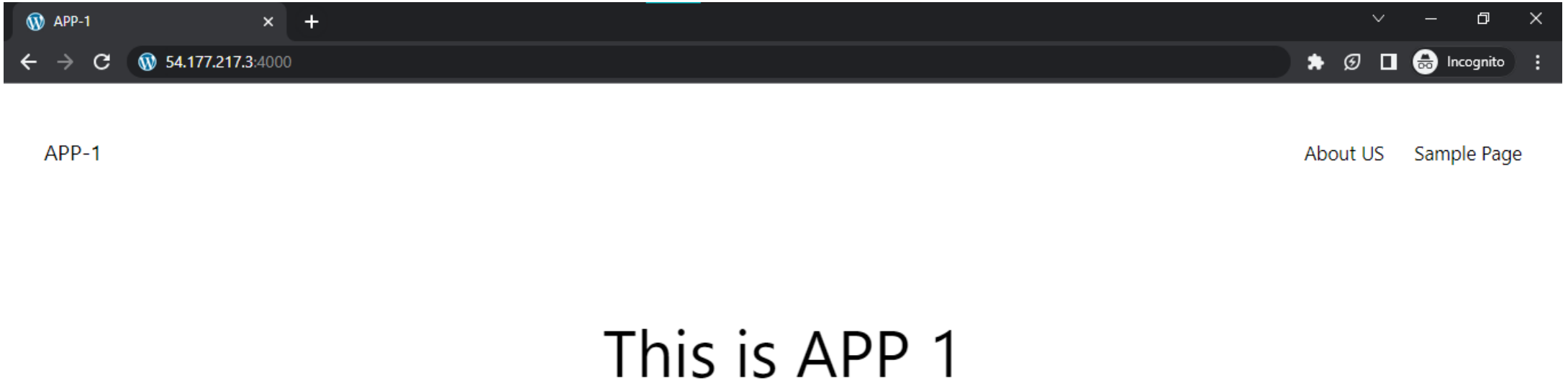
---imp note end ----------

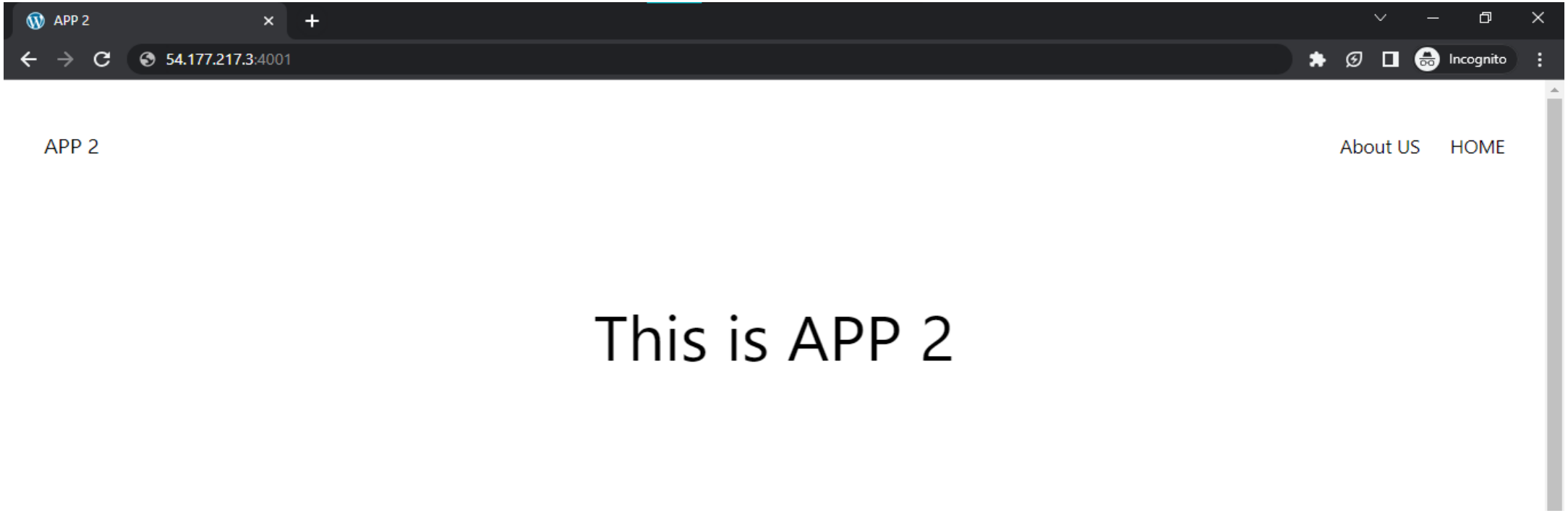
☐ ACCESS & CONFIGURE WPA1 & WPA2 CONTAINER COMBINED WITH NLBA (NETWORK LOAD BALANCER) WITH HOST's SERVER PUBLIC ADDRESS (Login on)

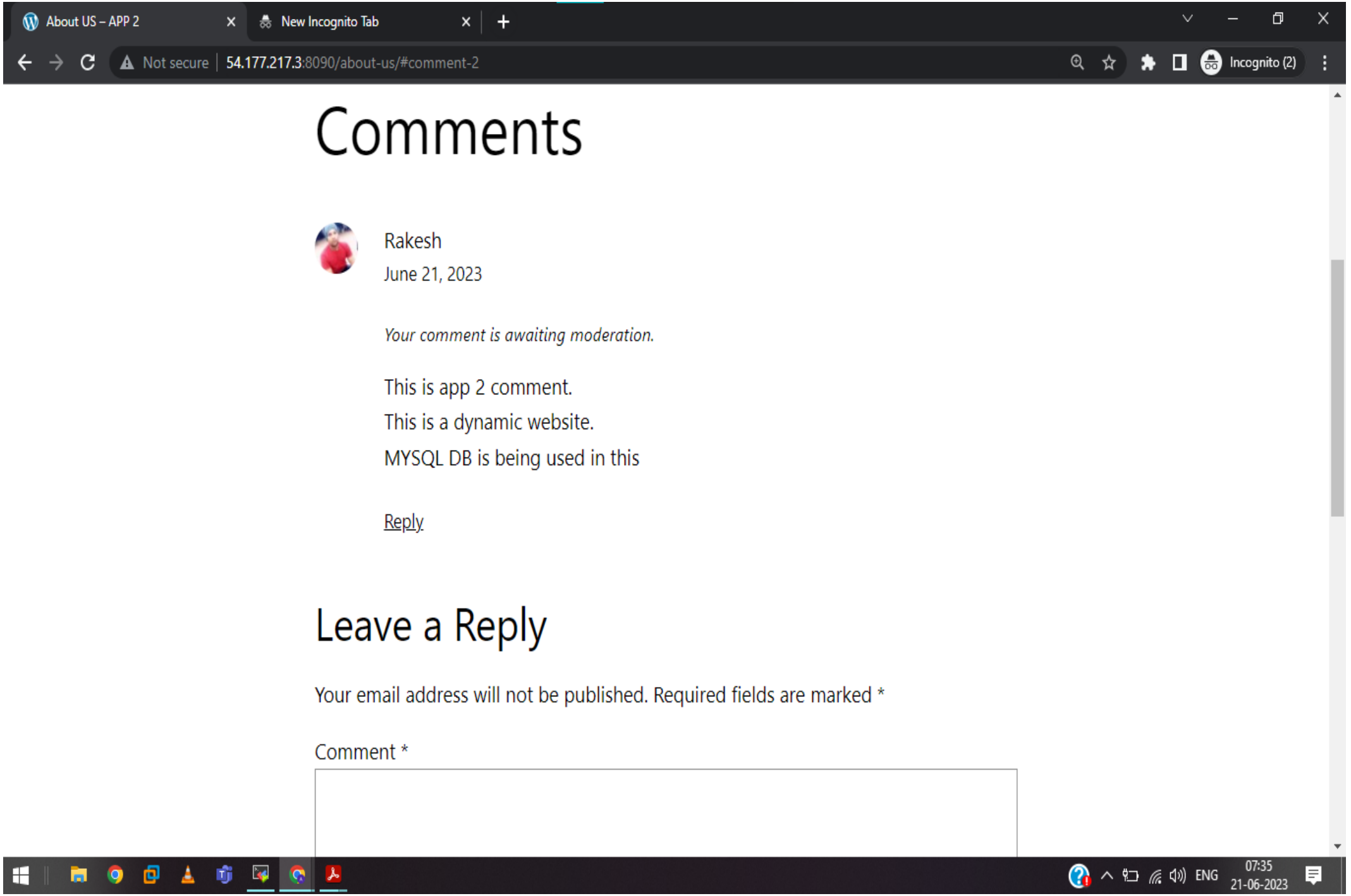
PUBLIC ADDRESS:NLB PORT	USERNAME	PASSWORD	OPEN LINK
http://54.177.217.3:4000	rakamodify	@#User5#@	http://54.177.217.3:4000/wp-login.php
http://54.177.217.3:4001	rakesh	@#User5#@	http://54.177.217.3:4001/wp-login.php

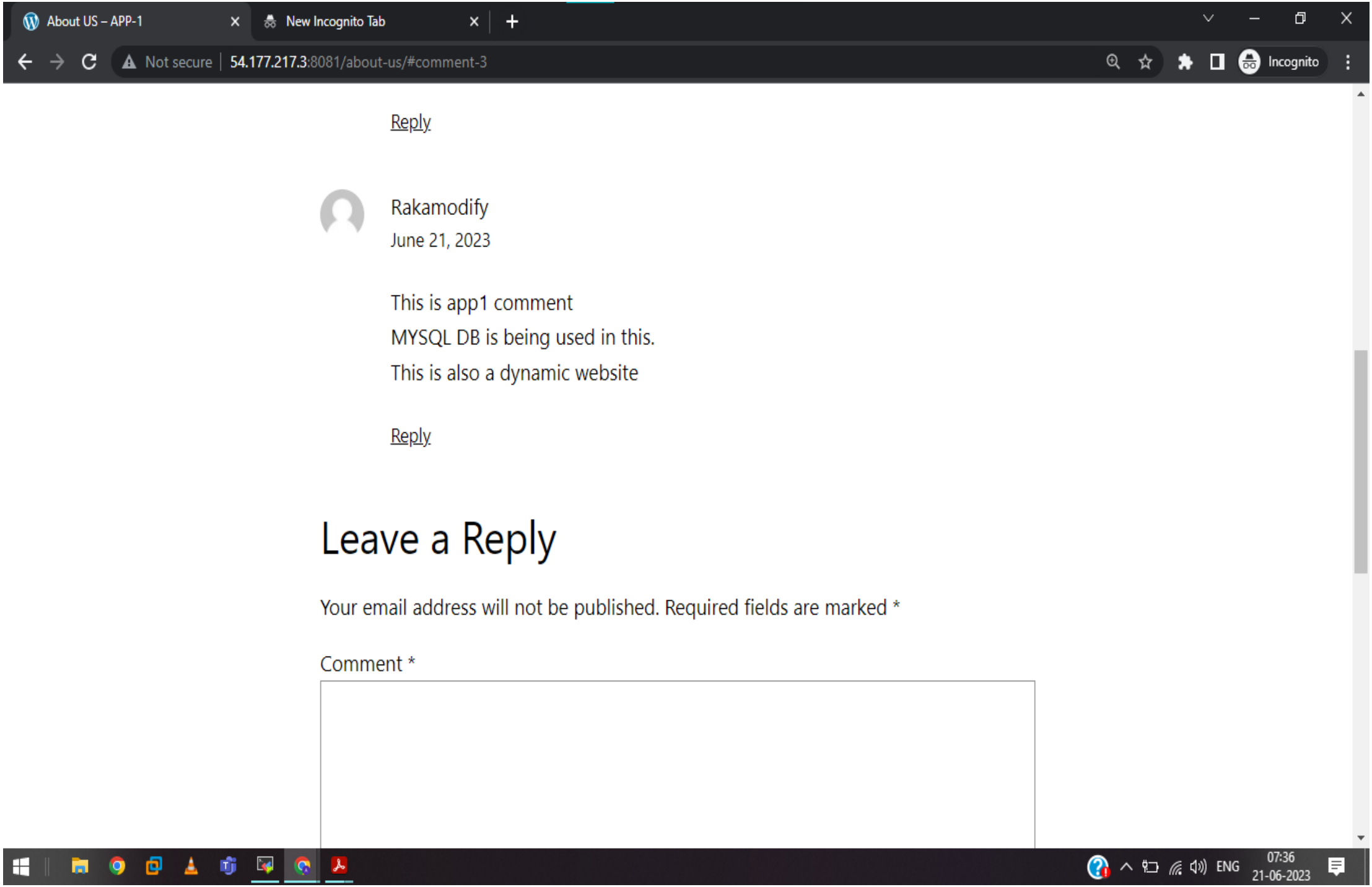



Log In ‹ rakamodify — WordPres
54.177.217.3:4001/wp-login.php
Incognito
Username or Email Address
rakesh
Password
@#User5#@
Remember Me
Log In
Lost your password?
← Go to rakamodify


APP-1
54.177.217.3:4000
Incognito
APP-1
About US
Sample Page
This is APP 1


APP 2
54.177.217.3:4001
Incognito
APP 2
About US
HOME
This is APP 2


About US – APP 2
New Incognito Tab
Not secure
54.177.217.3:8090/about-us/#comment-2
Incognito (2)
Comments
Rakesh
June 21, 2023
Your comment is awaiting moderation.
This is app 2 comment.
This is a dynamic website.
MYSQL DB is being used in this
Reply
Leave a Reply
Your email address will not be published. Required fields are marked *
Comment *
ENG
07:35
21-06-2023

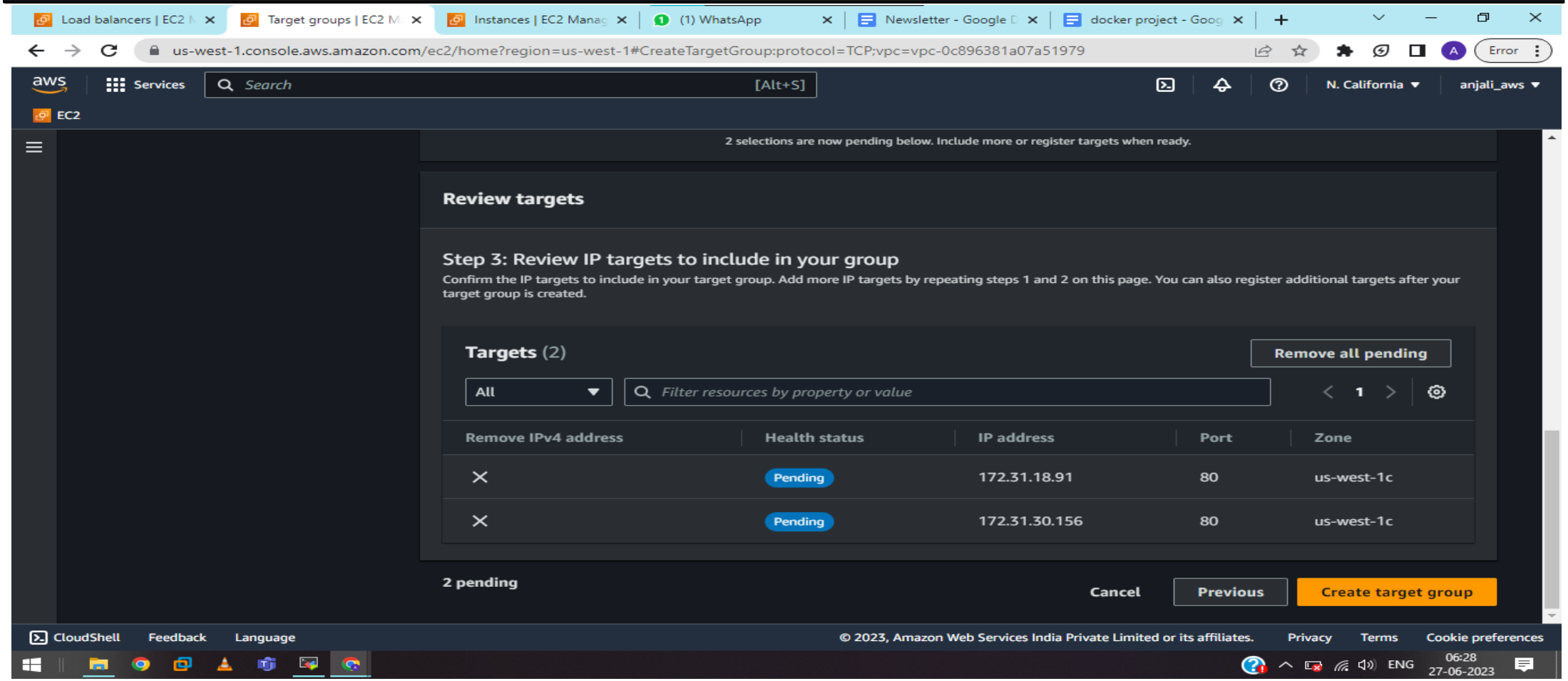

About US – APP-1
New Incognito Tab
Not secure
54.177.217.3:8081/about-us/#comment-3
Incognito
Reply
Rakamodify
June 21, 2023
This is app1 comment
MYSQL DB is being used in this.
This is also a dynamic website
Reply
Leave a Reply
Your email address will not be published. Required fields are marked *
Comment *
ENG
07:36
21-06-2023

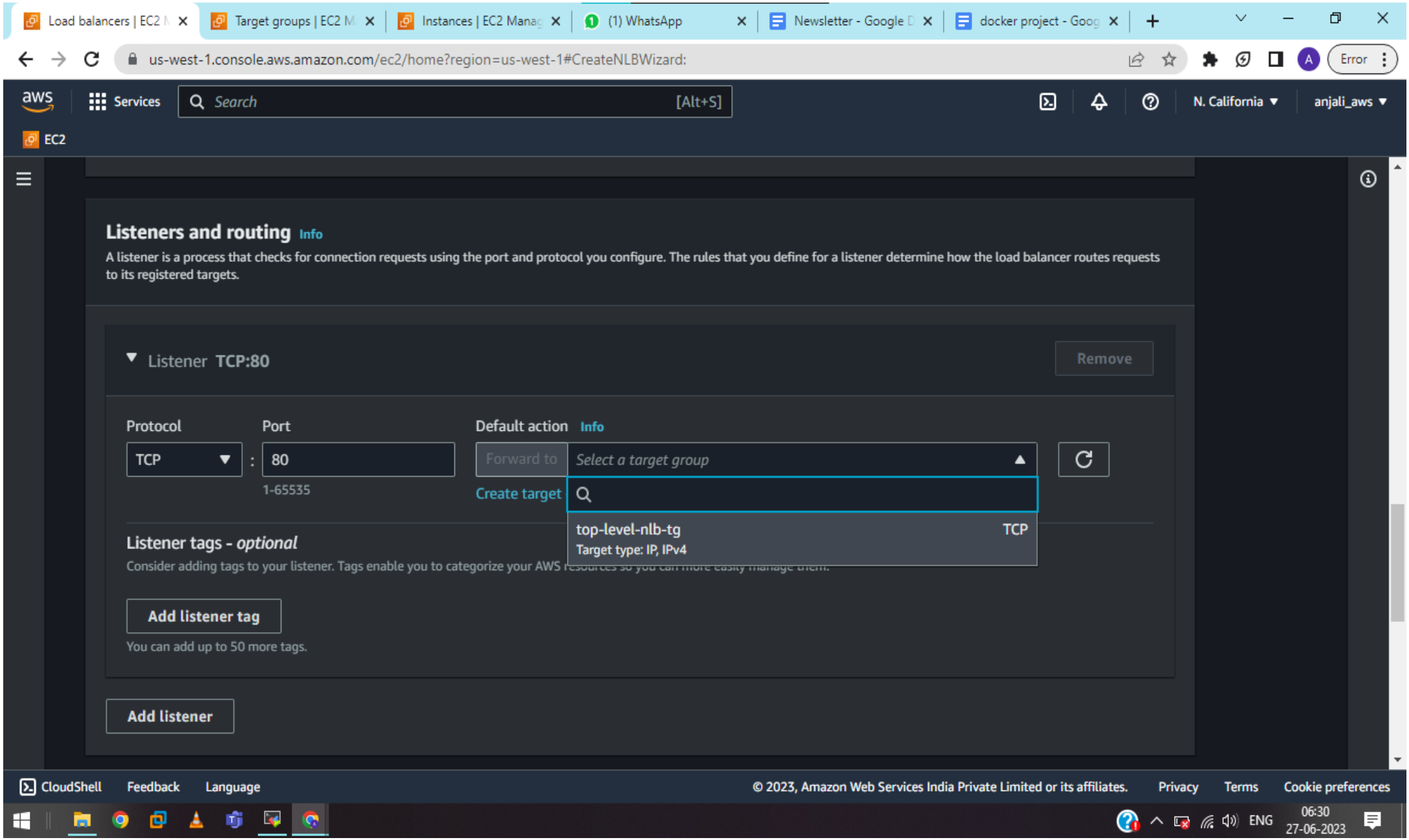
➢ Create Application Load Balancer and Configurations over attaching NLB

```
# docker run -d --name=alb1 -p 80:80 -v /mnt/efs/alb1:/etc/nginx/ --network=amazon nginx:latest
# vi nginx.conf
# cp    ./nginx.conf       /mnt/efs/alb1/
# cat    /mnt/efs/alb1/nginx.conf
# docker exec -it alb1 /bin/bash
```

```
events {
    worker_connections 1024;
}
http {
    upstream app1_backend {
        server 192.168.0.5:80;   # NLB1 IP and port
    }
    upstream app2_backend {
        server 12.168.0.5:80;   # NLB1 IP and port
    }
    server {
        listen 80;
        server_name your_domain.com;  # Replace with your actual domain name
        location /app1 {
            proxy_pass http://app1_backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            # Additional proxy settings if needed
```

```
    }
    location /app2 {
        proxy_pass http://app2_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        # Additional proxy settings if needed
    }
    # Additional server configurations if needed
  }
}
```




Load balancers | EC2
Target groups | EC2
Instances | EC2 Manag
(1) WhatsApp
Newsletter - Google
docker project - Goog
us-west-1.console.aws.amazon.com/ec2/home?region=us-west-1#CreateNLBWizard:
Error
aws
Services
Search
[Alt+S]
N. California
anjali_aws
EC2
Listeners and routing Info
A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.
Listener TCP:80
Remove
Protocol
Port
Default action Info
TCP
80
1-65535
Forward to
Select a target group
Create target
top-level-nlb-tg
TCP
Target type: IP, IPv4
Listener tags - optional
Consider adding tags to your listener. Tags enable you to categorize your AWS
Add listener tag
You can add up to 50 more tags.
Add listener
CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences
ENG
06:30
27-06-2023

Load balancers | EC2 | Target groups | EC2 | Instances | EC2 Manag | (1) WhatsApp | Newsletter - Google | docker project - Goog

us-west-1.console.aws.amazon.com/ec2/home?region=us-west-1#CreateNLBWizard:

Error

aws | Services | Search [Alt+S] | N. California | anjali_aws

EC2

EC2 > Load balancers > Create Network Load Balancer

Create Network Load Balancer Info

The Network Load Balancer distributes incoming TCP and UDP traffic across multiple targets such as Amazon EC2 instances, microservices, and containers. When the load balancer receives a connection request, it selects a target based on the protocol and port that are specified in the listener configuration, and the routing rule specified as the default action.

How Elastic Load Balancing works

Basic configuration

Load balancer name

Name must be unique within your AWS account and can't be changed after the load balancer is created.

top-level-nlb

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Scheme

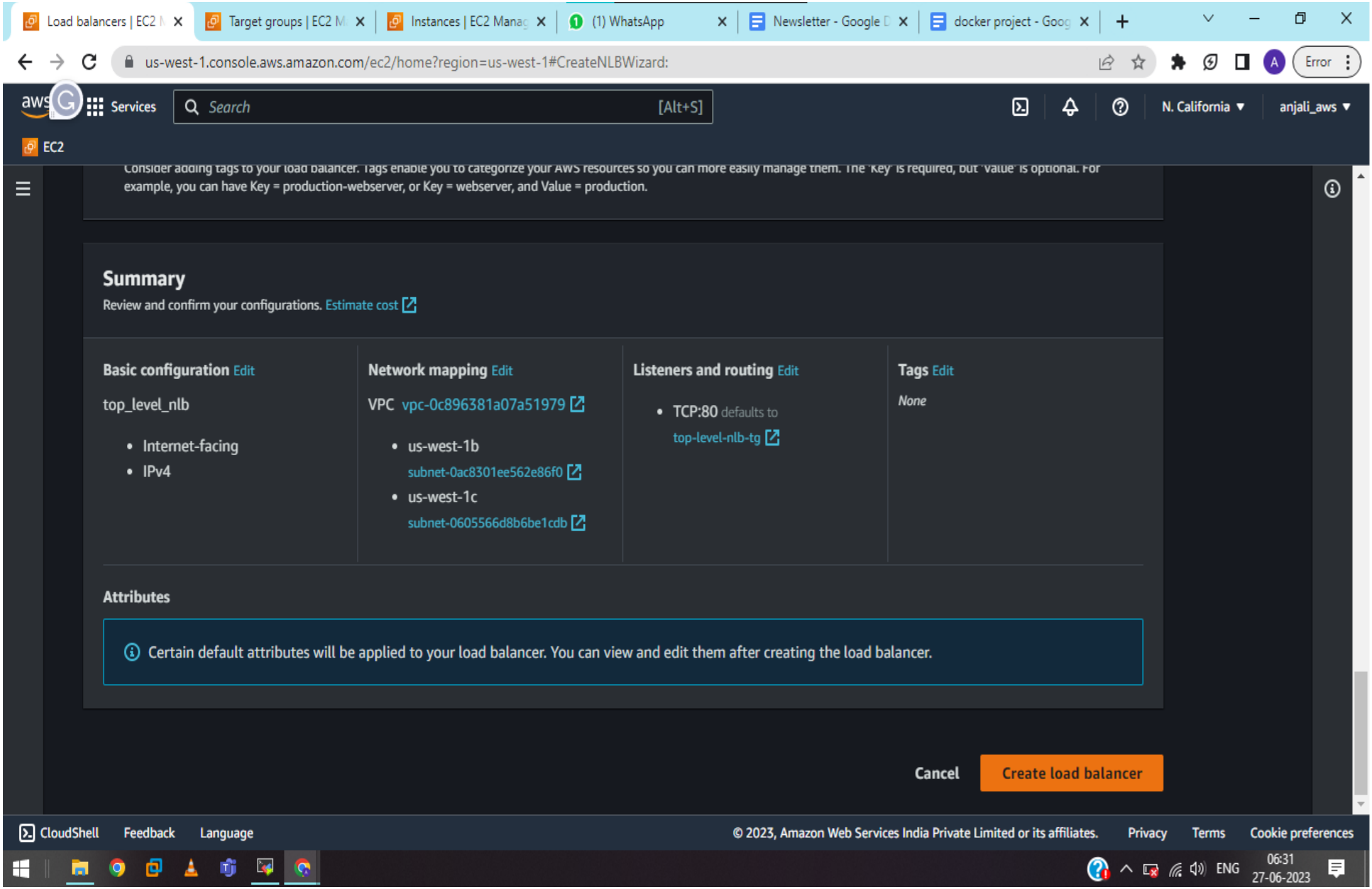
Scheme can't be changed after the load balancer is created.

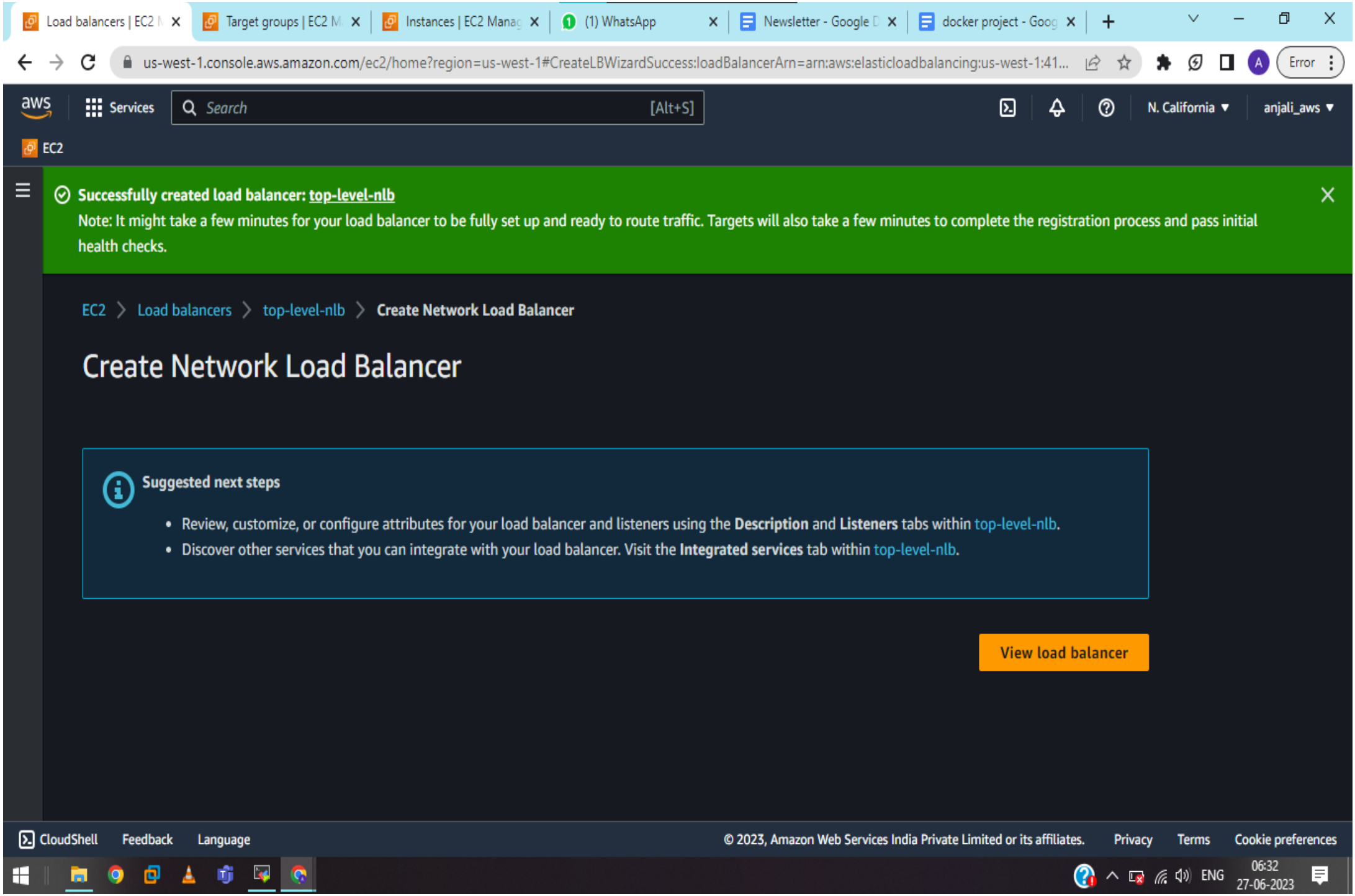
- Internet-facing
  An internet-facing load balancer routes requests from clients over the internet to targets. Requires a public subnet. Learn more

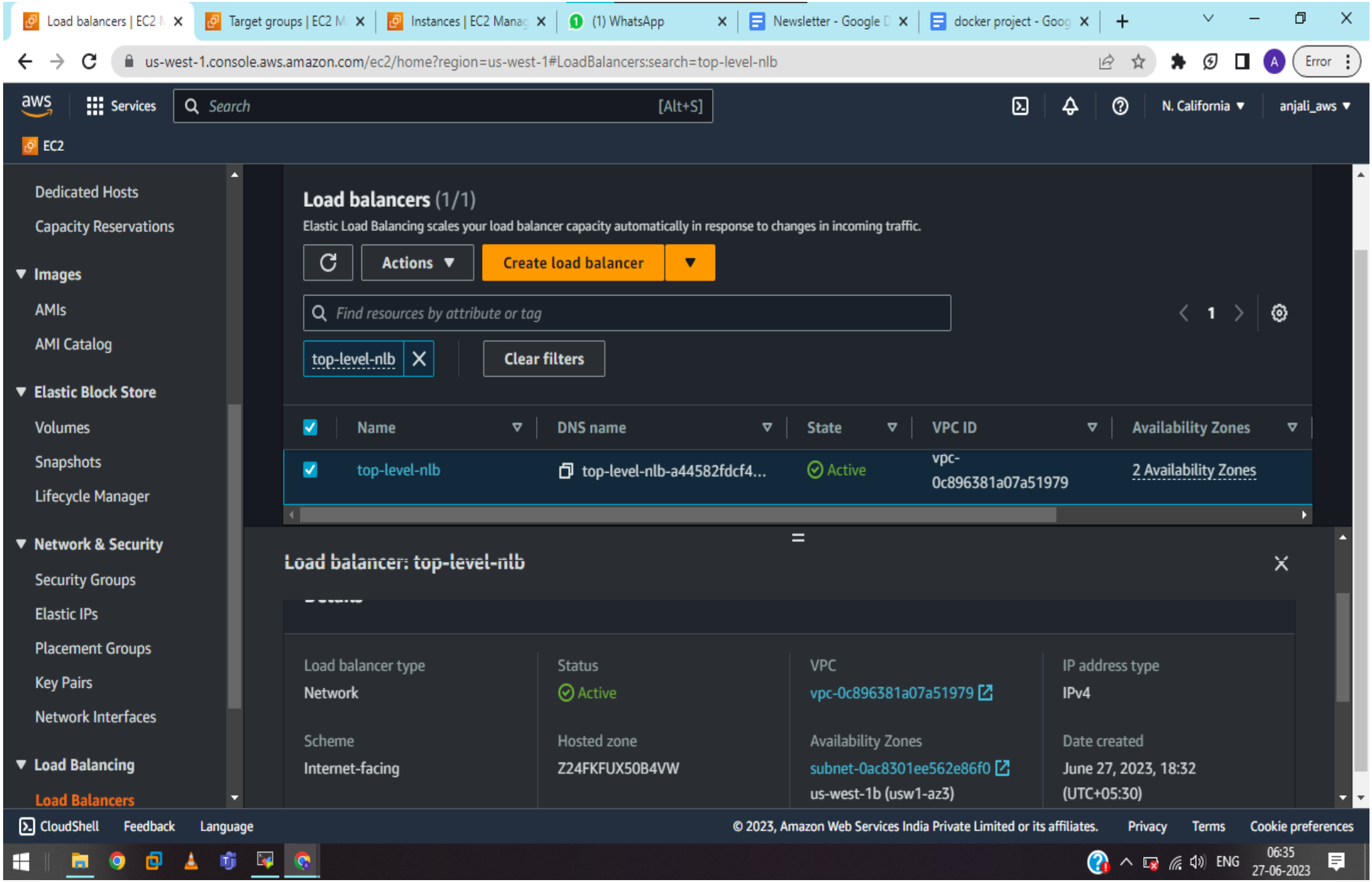
CloudShell | Feedback | Language

 Privacy | Terms | Cookie preferences

ENG 06:32 27-06-2023


Load balancers | EC2
Target groups | EC2
Instances | EC2 Manag
(1) WhatsApp
Newsletter - Google
docker project - Goog
us-west-1.console.aws.amazon.com/ec2/home?region=us-west-1#CreateNLBWizard:
Error
Services
Search
[Alt+S]
N. California
anjali_aws
EC2
Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them. The 'Key' is required, but 'Value' is optional. For example, you can have Key = production-webserver, or Key = webserver, and Value = production.
Summary
Review and confirm your configurations. Estimate cost
Basic configuration Edit
top_level_nlb
Internet-facing
IPv4
Network mapping Edit
VPC vpc-0c896381a07a51979
us-west-1b
subnet-0ac8301ee562e86f0
us-west-1c
subnet-0605566d8b6be1cdb
Listeners and routing Edit
TCP:80 defaults to
top-level-nlb-tg
Tags Edit
None
Attributes
Certain default attributes will be applied to your load balancer. You can view and edit them after creating the load balancer.
Cancel
Create load balancer
CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences
ENG
06:31
27-06-2023


Load balancers | EC2
Target groups | EC2
Instances | EC2 Manag
(1) WhatsApp
Newsletter - Google
docker project - Goog
us-west-1.console.aws.amazon.com/ec2/home?region=us-west-1#CreateLBWizardSuccess:loadBalancerArn=arn:aws:elasticloadbalancing:us-west-1:41...
Error
aws
Services
Search
[Alt+S]
N. California
anjali_aws
EC2
Successfully created load balancer: top-level-nlb
Note: It might take a few minutes for your load balancer to be fully set up and ready to route traffic. Targets will also take a few minutes to complete the registration process and pass initial health checks.
EC2 > Load balancers > top-level-nlb > Create Network Load Balancer
Create Network Load Balancer
Suggested next steps
Review, customize, or configure attributes for your load balancer and listeners using the Description and Listeners tabs within top-level-nlb.
Discover other services that you can integrate with your load balancer. Visit the Integrated services tab within top-level-nlb.
View load balancer
CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences
ENG
06:32
27-06-2023


Load balancers | EC2
Target groups | EC2
Instances | EC2 Manag
(1) WhatsApp
Newsletter - Google
docker project - Goog
us-west-1.console.aws.amazon.com/ec2/home?region=us-west-1#LoadBalancers:search=top-level-nlb
Error
aws
Services
Search
[Alt+S]
N. California
anjali_aws
EC2
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Load balancers (1/1)
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.
Actions
Create load balancer
Find resources by attribute or tag
1
top-level-nlb
Clear filters
Name
DNS name
State
VPC ID
Availability Zones
top-level-nlb
top-level-nlb-a44582fdcf4...
Active
vpc-0c896381a07a51979
2 Availability Zones
Load balancer: top-level-nlb
Load balancer type
Network
Status
Active
VPC
vpc-0c896381a07a51979
IP address type
IPv4
Scheme
Internet-facing
Hosted zone
Z24FKFUX50B4VW
Availability Zones
subnet-0ac8301ee562e86f0
us-west-1b (usw1-az3)
Date created
June 27, 2023, 18:32
(UTC+05:30)
CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences
ENG
06:35
27-06-2023

Access App1 & App2 over NLB DNS address:

On browser

http://us-west-1.console.aws.amazon.com/console/home?region=us-west-1/app1

http://us-west-1.console.aws.amazon.com/console/home?region=us-west-1/app1