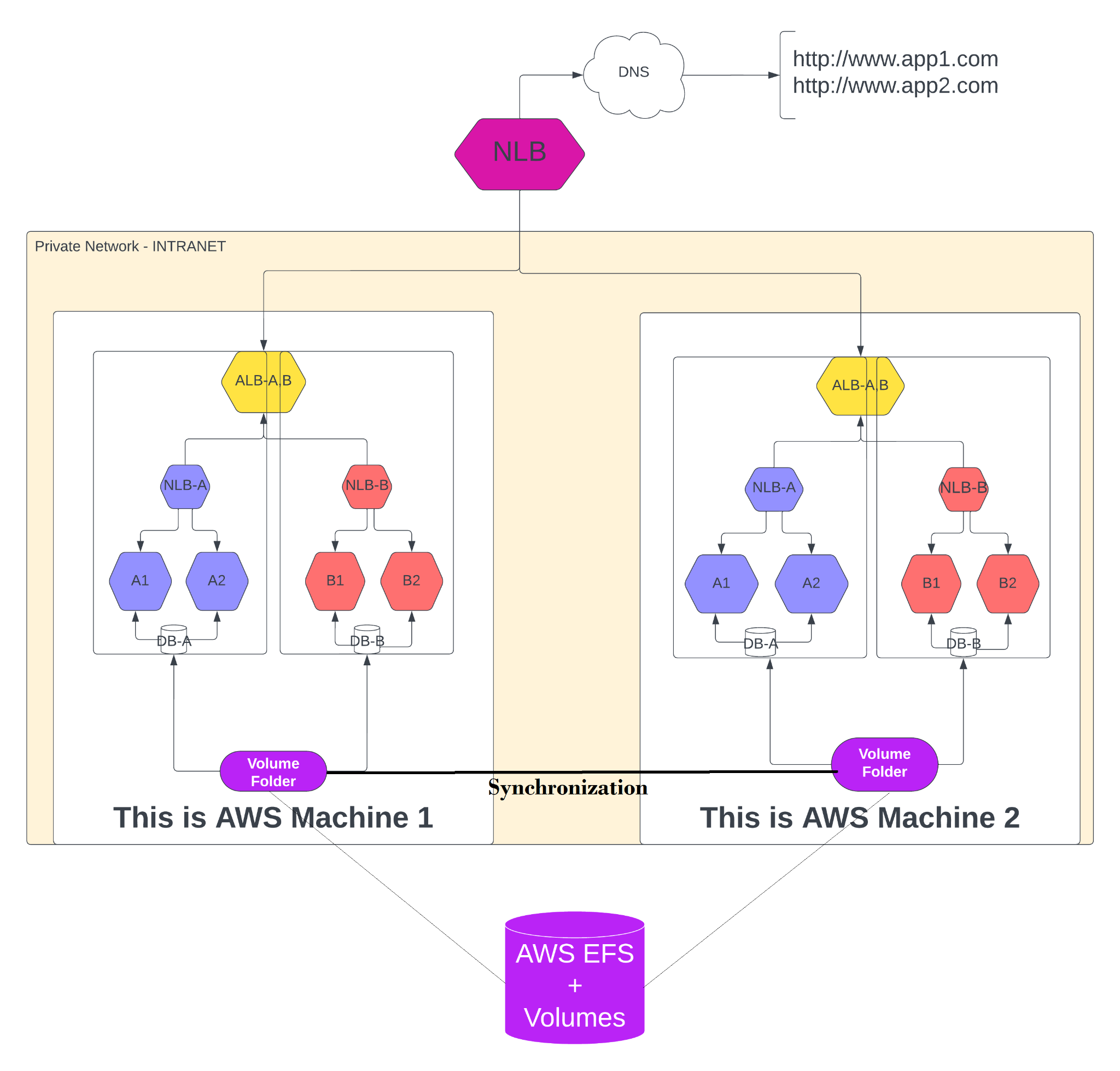
This is Micro-Container Architecture Based   
DOCKER PROJECT

Rakesh Kumar Jangid

Date: 20-06-2023

# 

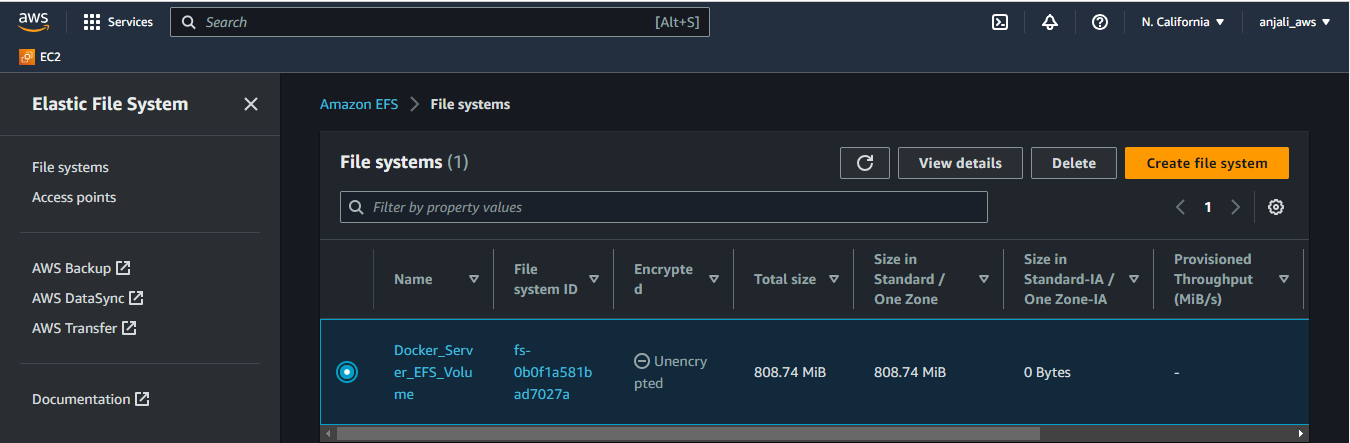


# INTRODUCTION

* 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,

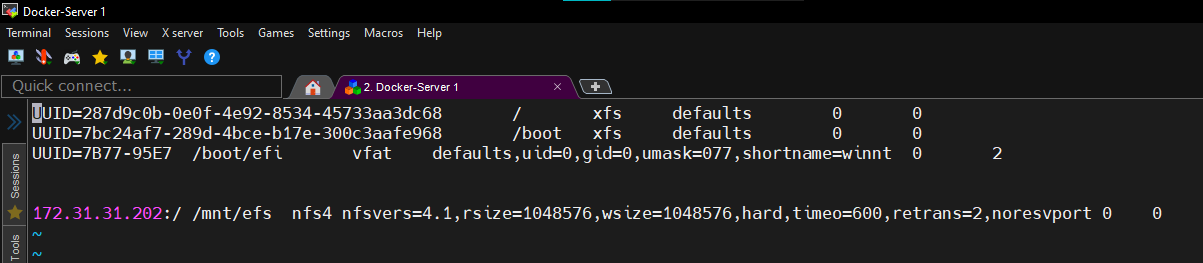
retrans=2,noresvport

# 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](https://github.com/rakesh08061994/docker/blob/main/Docker_Setup_On_Debian.sh)

# chmod a+x [Docker\_Setup\_on\_fedora.sh](https://github.com/rakesh08061994/docker/blob/main/Docker_Setup_on_fedora.sh)

# ./[Docker\_Setup\_on\_fedora.sh](https://github.com/rakesh08061994/docker/blob/main/Docker_Setup_on_fedora.sh)

# ./[Docker\_Setup\_On\_Debian.sh](https://github.com/rakesh08061994/docker/blob/main/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

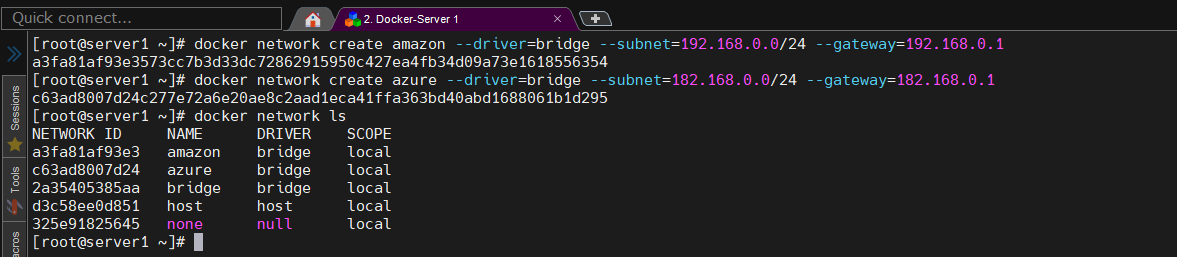
# docker network create amazon --driver=bridge --subnet=192.168.0.0/24 --gateway=192.168.0.1

# 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

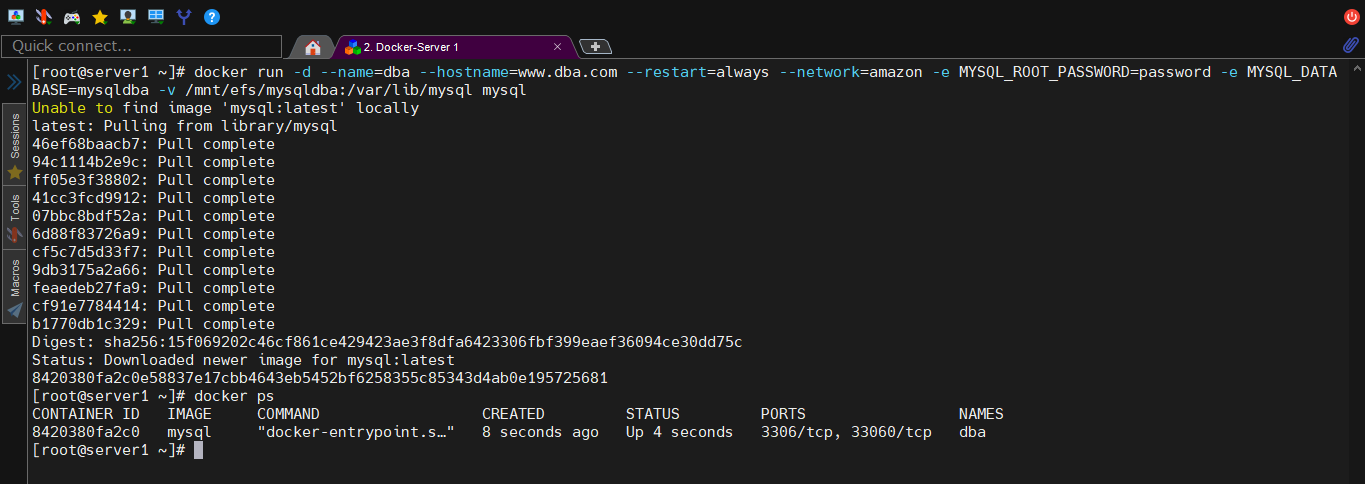
MYSQL\_ROOT\_PASSWORD=password -e MYSQL\_DATABASE=mysqldba -v

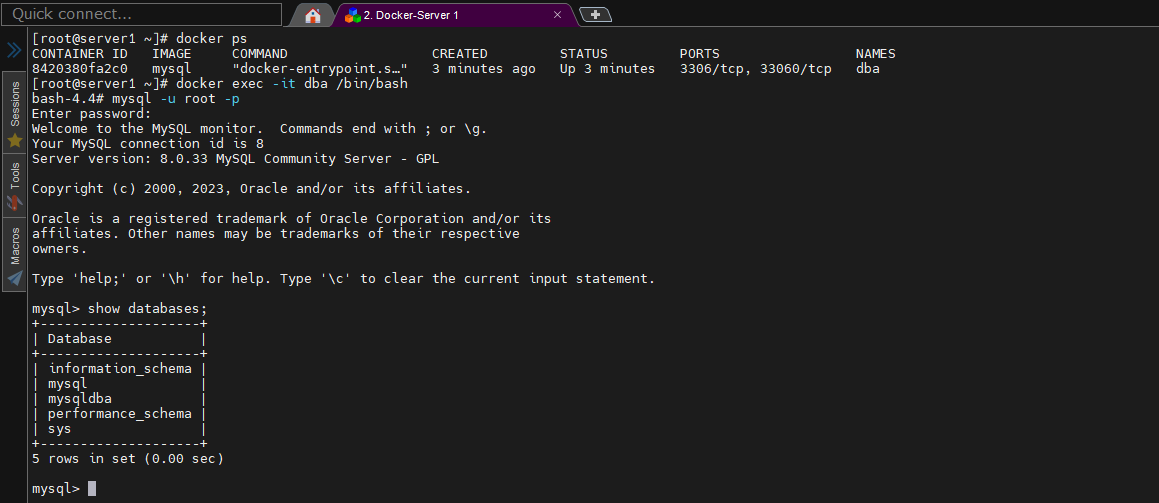
/mnt/efs/mysqldba1:/var/lib/mysql mysql

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:

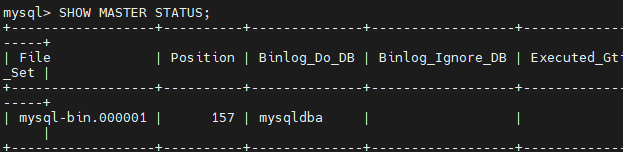
(On server2)

# 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>;



* 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

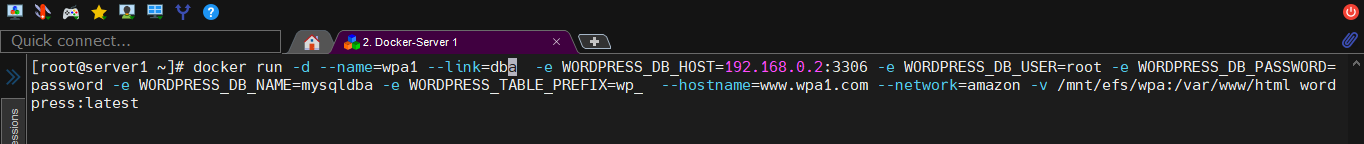
# docker exec -it dba /bin/bash

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

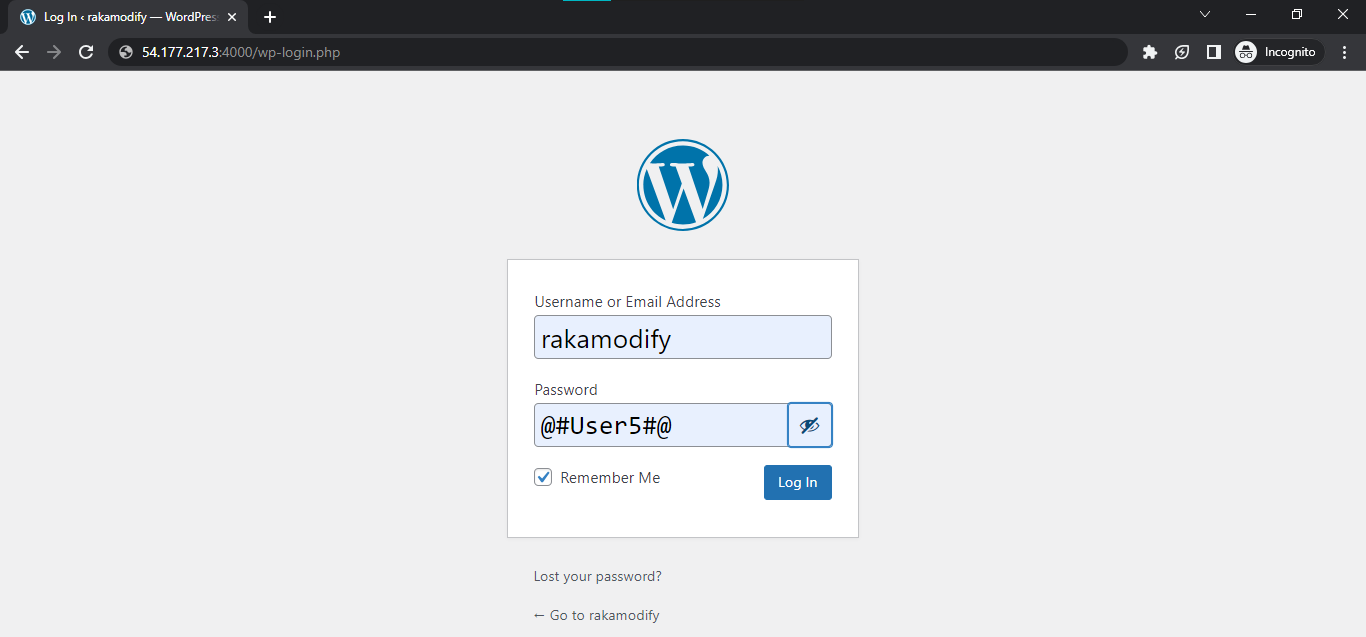
then replace the $web1\_container\_IP from its actual ip of wpa1 & wpb1

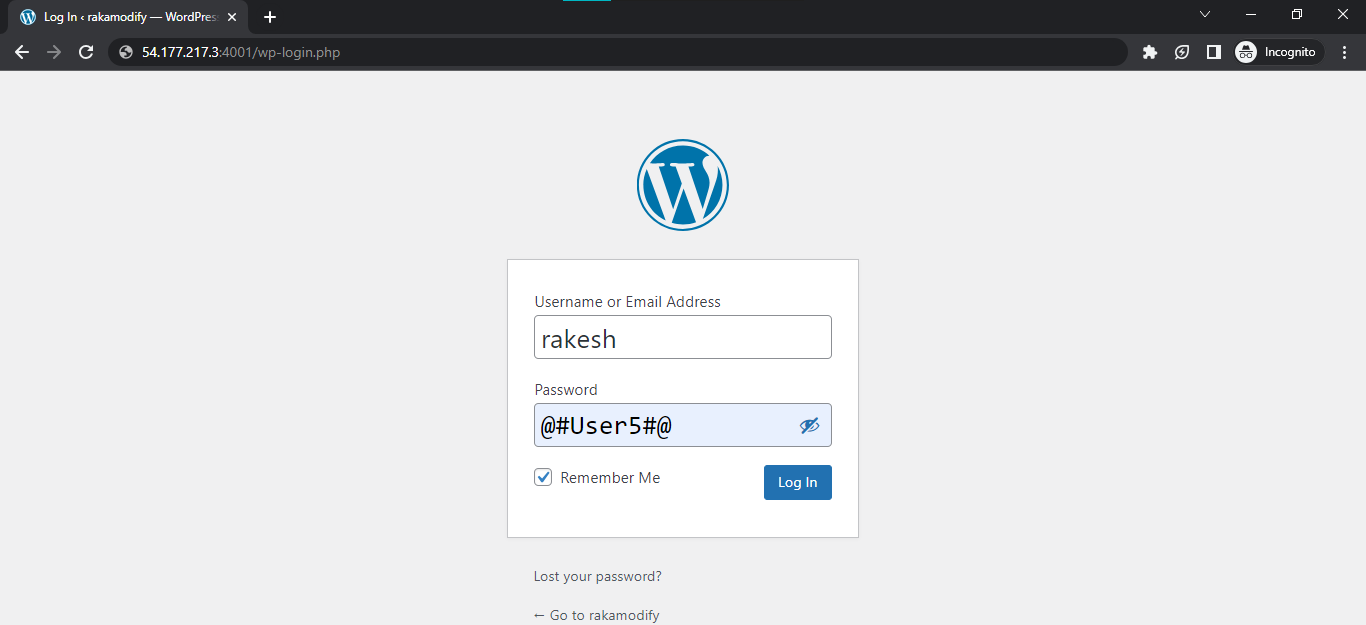
and replace the $web2\_container\_IP from its actual ip of wpa1 & wpb1

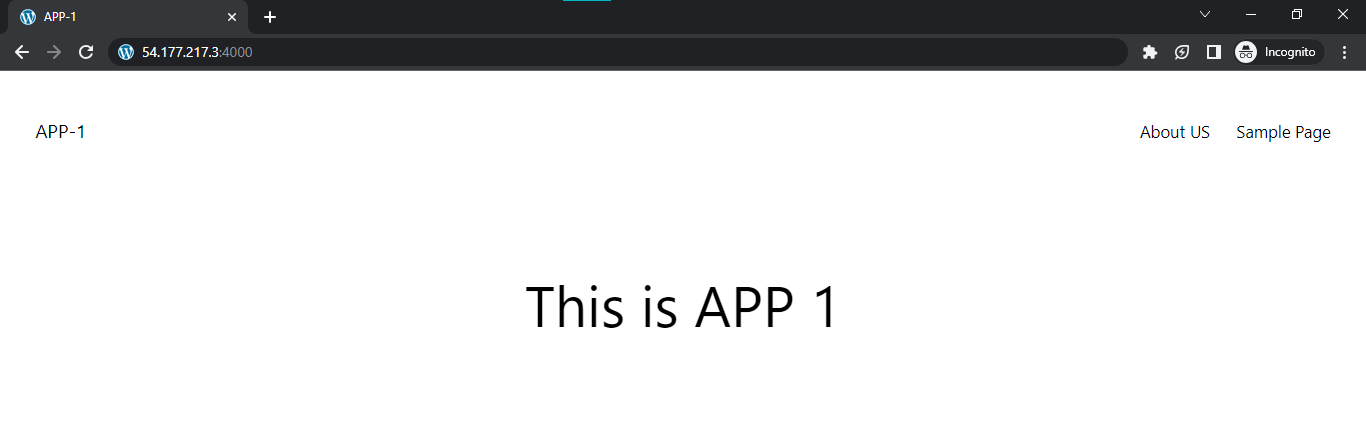
---imp note end -----------------

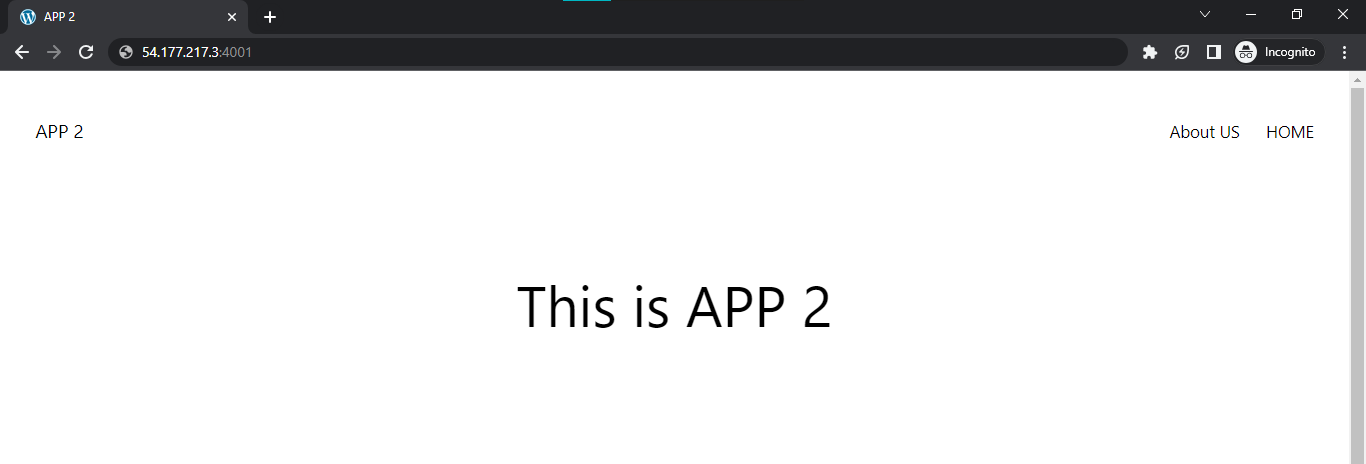
* ACCESS & CONFIGURE WPA1 & WPA2 CONTAINER COMBINED WITH NLBA (NETWORK LOAD BALANCER) WITH HOST’s SERVER PUBLIC ADDRESS (Login on )

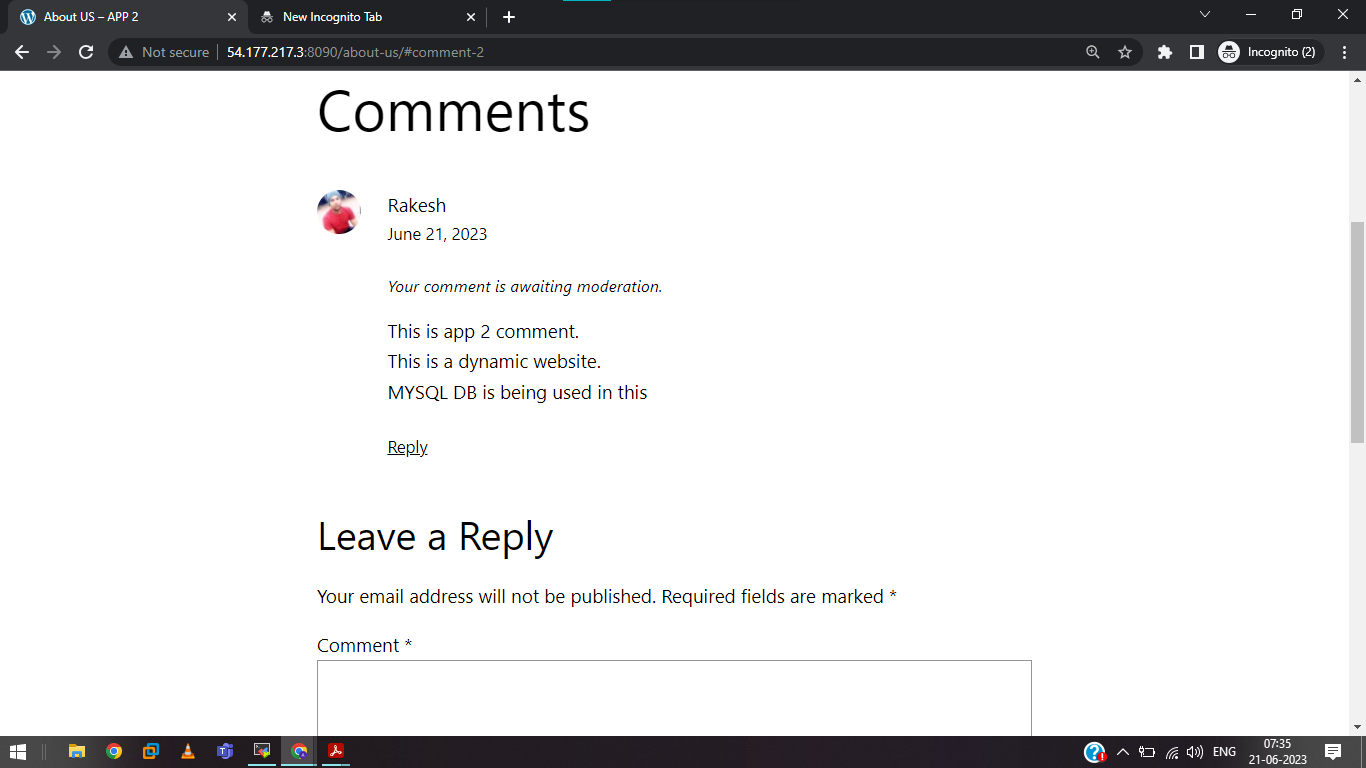
| ***PUBLIC ADDRESS:NLB PORT*** | ***USERNAME*** | ***PASSWORD*** | ***OPEN LINK*** |
| --- | --- | --- | --- |
| ***http://54.177.217.3:4000*** | ***rakamodify*** | ***@#User5#@*** | [***http://54.177.217.3:4000/wp-login.php***](http://54.177.217.3:4000/wp-login.php) |
| ***http://54.177.217.3:4001***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

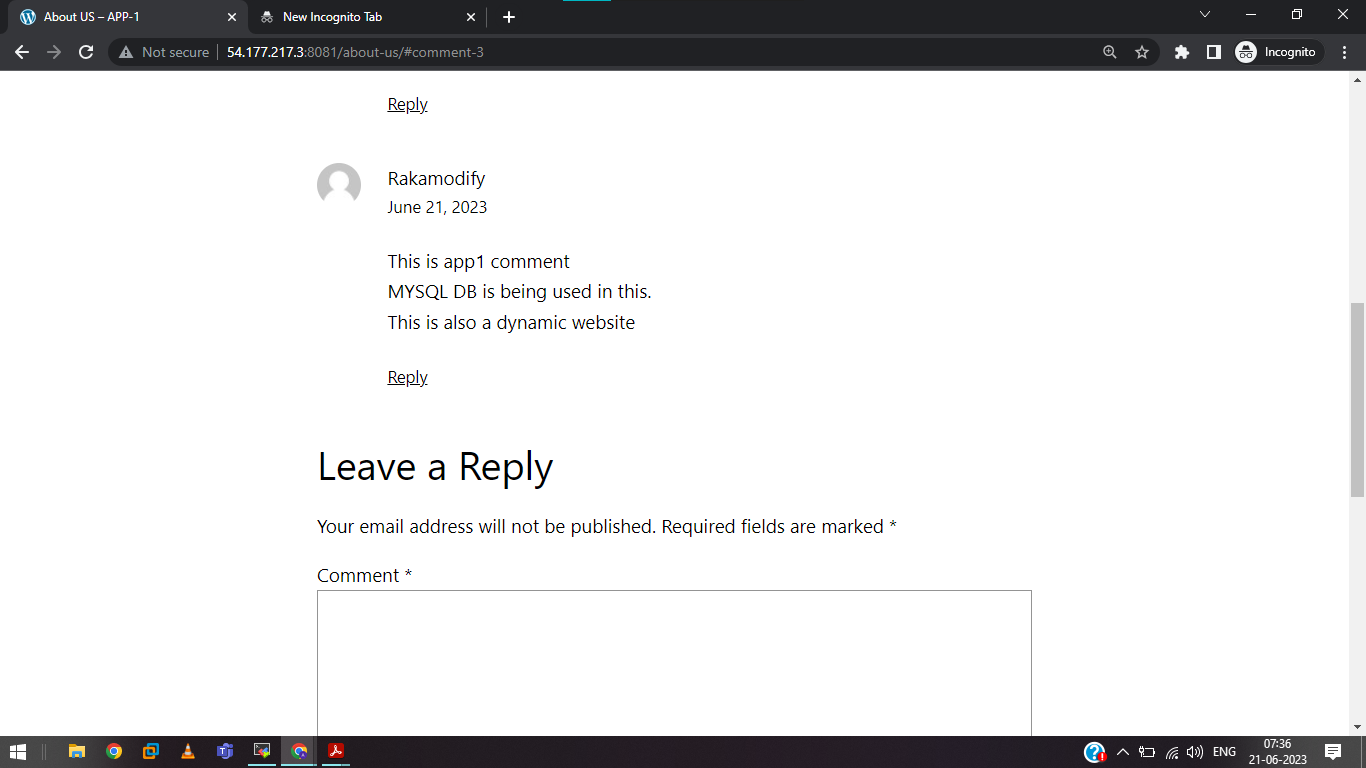
****











* + 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;

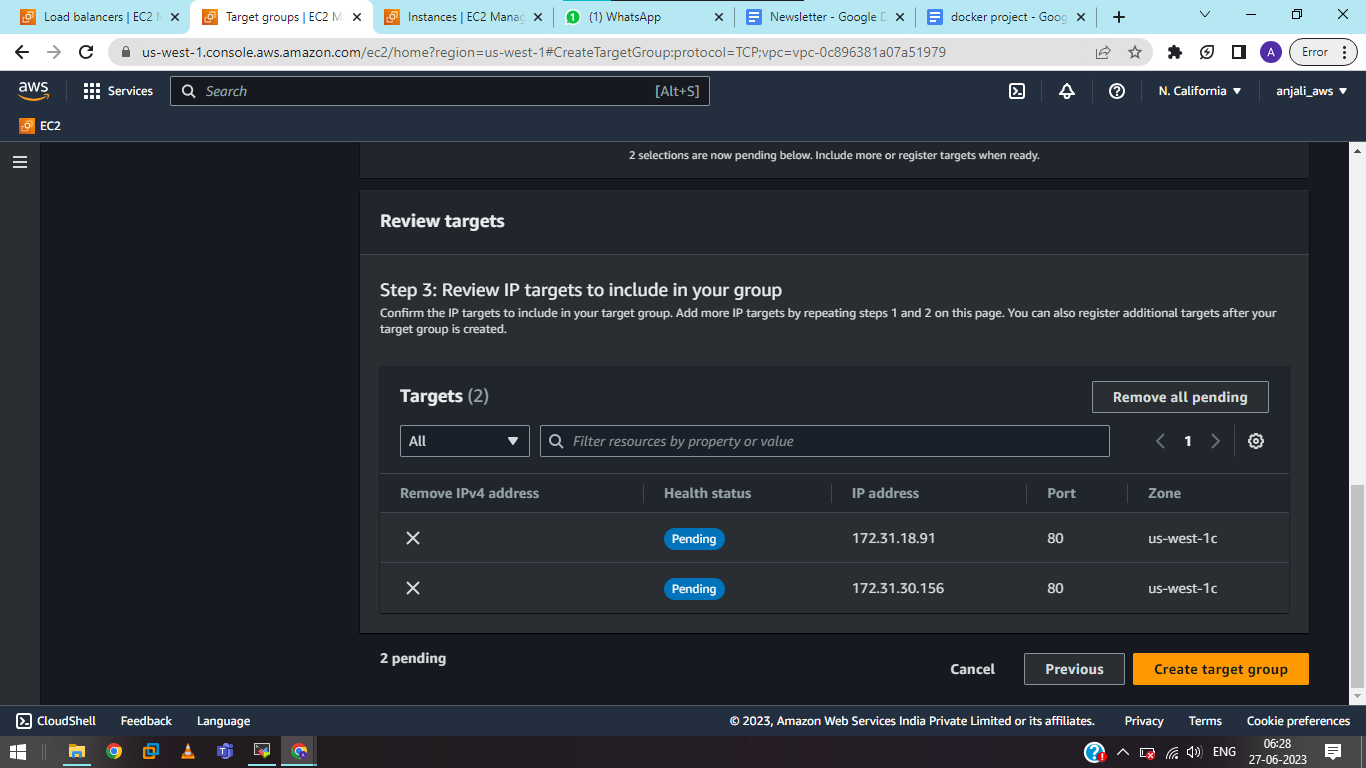
# Additional proxy settings if needed

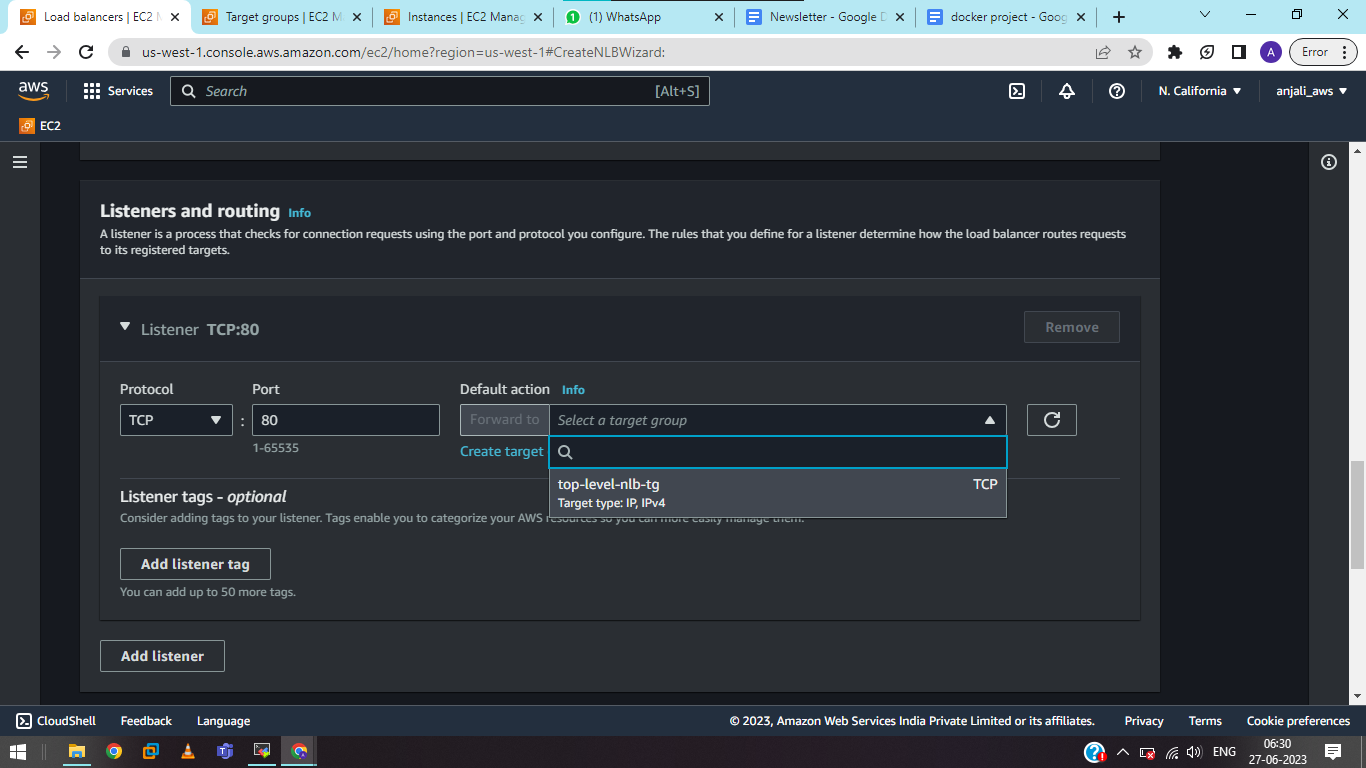
}

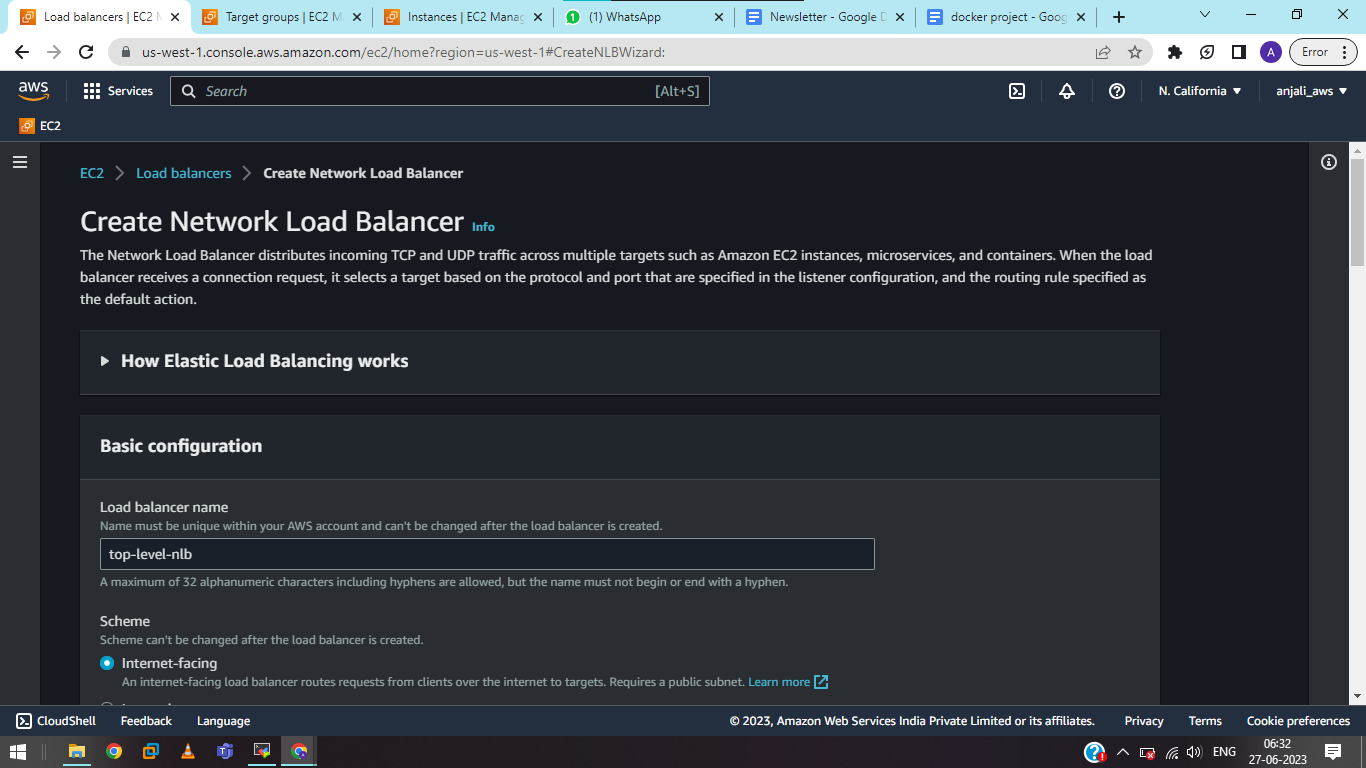
# Additional server configurations if needed

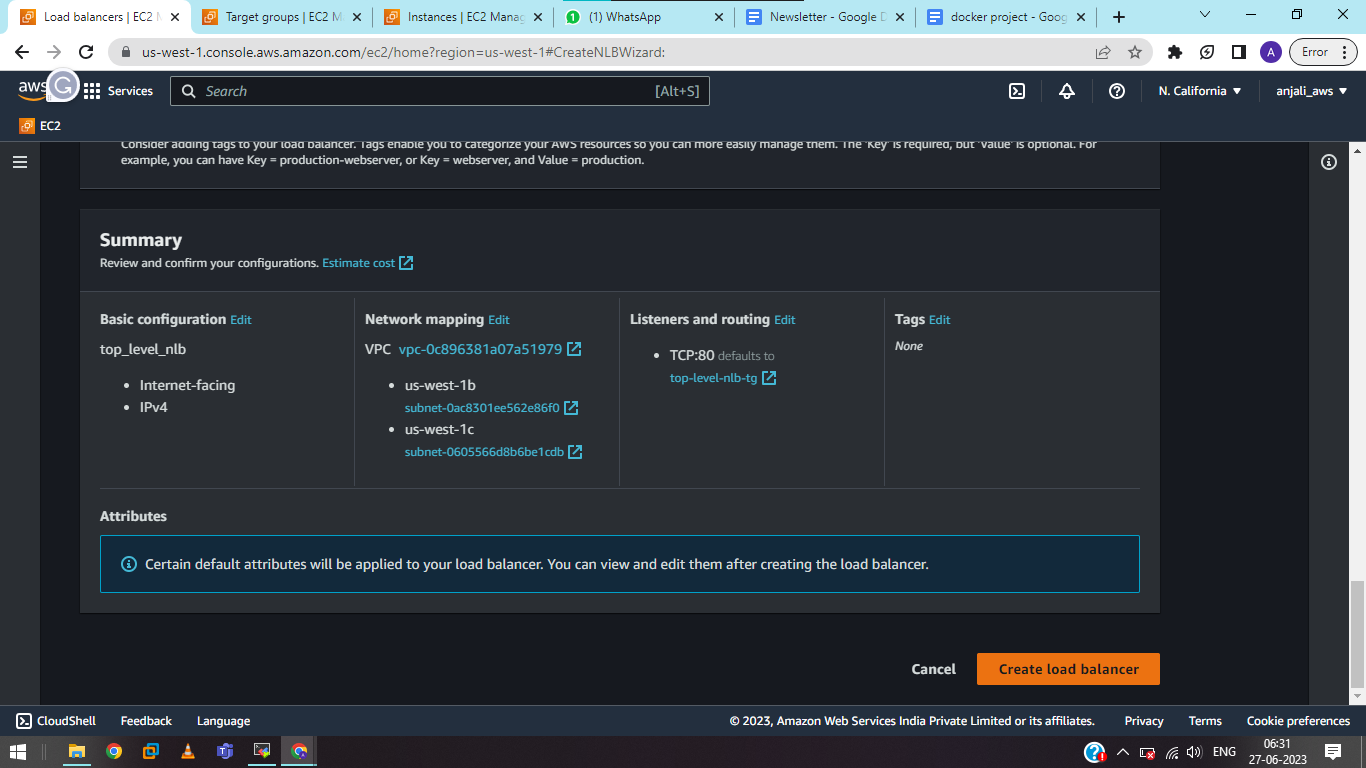
}

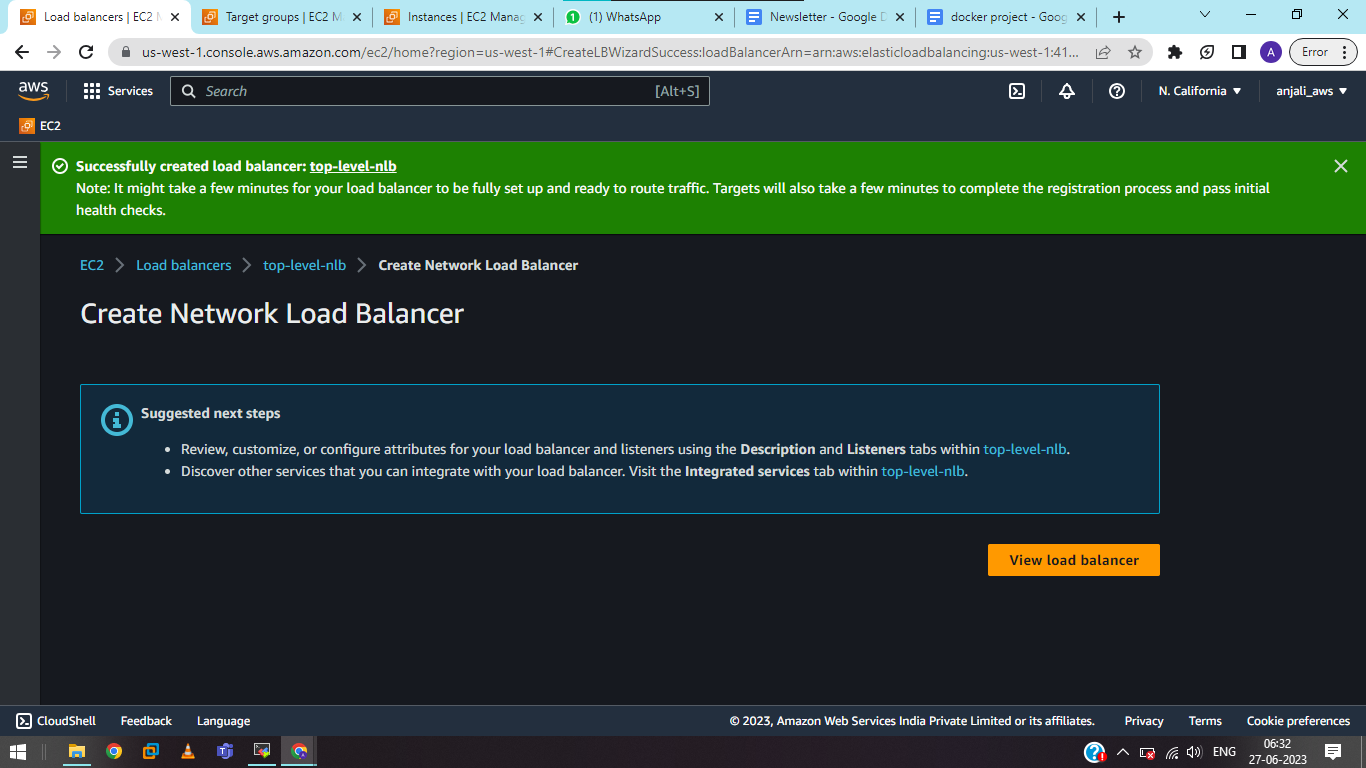
}

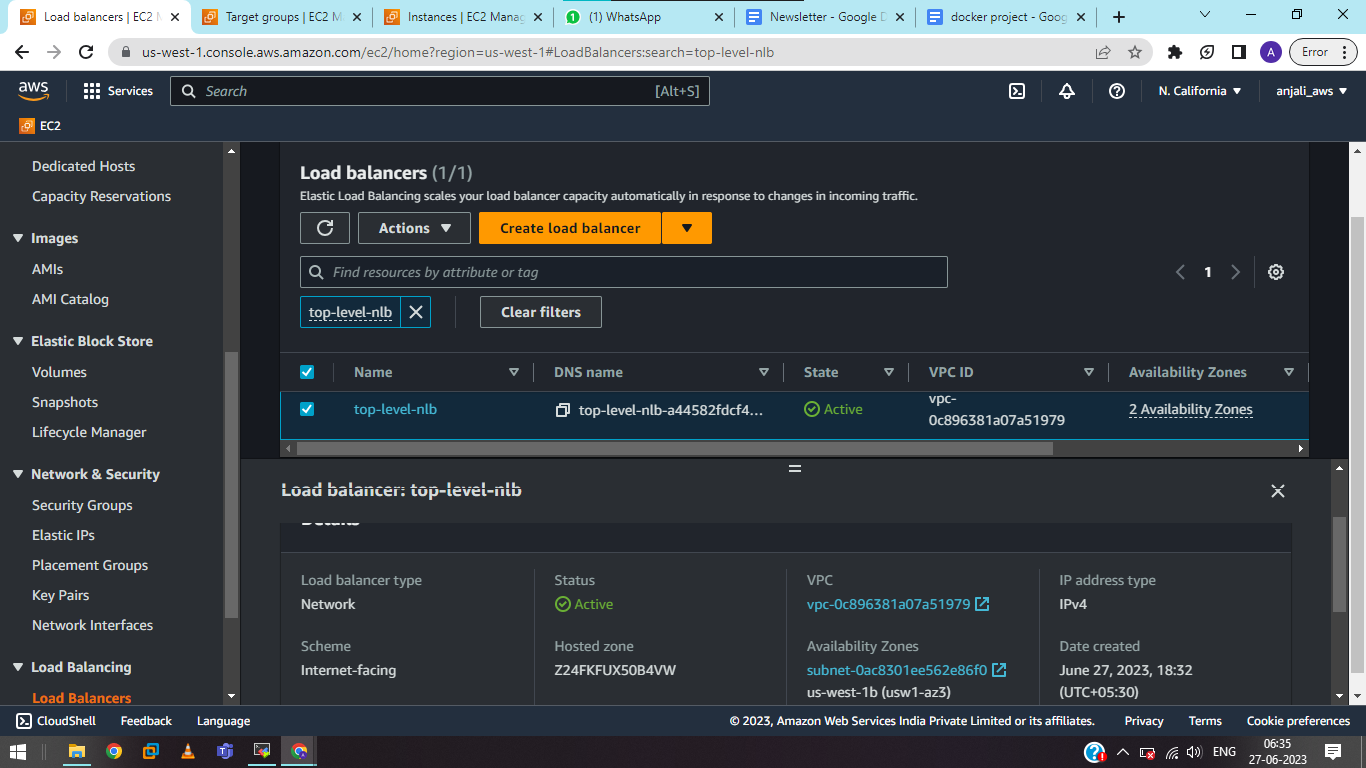












Access App1 & App2 over NLB DNS address:

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>