

3-Tier Application Architecture

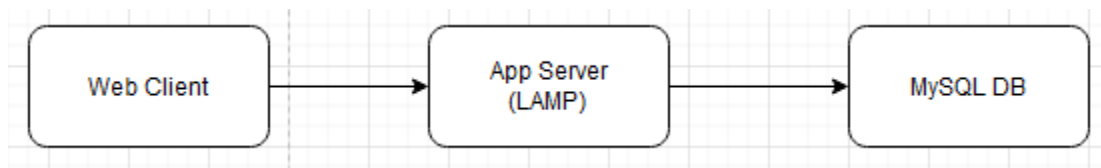
Kenneth Tan

Goal: Create a 3 Tier Architecture to securely serve PHPMyAdmin connected to AWS RDS

A 3 Tier architecture consists of a client, a server, and a database. In the architecture I created for this deployment, the client is the web client that the user accesses the app from, the server is the application server, in this deployment I create a LAMP stack, and the database is MySQL in AWS RDS.

LAMP = Linux Apache MySQL PHP

RDS = Relational Database Service



This is a basic overview of how the application will run. However, this architecture alone is not secure as traffic from the web directly comes into contact with our application's source code. This can lead to security compromises and unwarranted changes to our code. This can also lead to attackers gaining access to our database which may have sensitive information.

To secure this architecture, I created this architecture in AWS and added in a reverse proxy for an added layer of security. This allowed me to port access to the sources of traffic I know are safe. For example, only the reverse proxy will be able to access the application server. No one else that goes through port 80 will be able to access it.

I also created a MySQL server on AWS RDS to allow for higher availability and fault tolerance. In a traditional LAMP stack, it becomes a single point of failure so if the single EC2 instance goes down, everything including the database also goes down. Separating the components allows for fault tolerance and high availability.

To ensure that there is high availability in the architecture, I utilized an application load balancer to have the traffic to route to the application and database available in different regions if needed.

Ultimately, I created an architecture that is more secure and more resilient to potential hardware failure.

Reverse Proxy

A reverse proxy hides the ip of our application server by interfacing with the internet traffic and redirects it. If someone were to try and access our application they would see the ip of the proxy and not the actual server.

I used the following bash script to create the nginx server and to set it up as a reverse proxy.

```
#bin/bash

sudo apt update
sudo apt upgrade -y
sudo apt install nginx -y
echo 'server {
    listen 80;
    location / {
        proxy_pass http://<priate ip of app server>;
    }
}' >> reverse_proxy.conf

sudo mv reverse_proxy.conf /etc/nginx/sites-available/
sudo unlink /etc/nginx/sites-enabled/default
sudo ln -s /etc/nginx/sites-available/reverse_proxy.conf
/etc/nginx/sites-enabled/reverse_proxy.conf
```

This script will update the EC2, install Nginx and create the configuration file for the reverse proxy i called reverse_proxy.conf. I unlink the default file from the symbolic link from the site-enabled folder to allow for the reverse proxy to be linked instead, otherwise it will be a webserver.

This reverse proxy will take incoming traffic from port 80 and forward it to the EC2 where the application lives via the IP address.

LAMP Stack

To set up the app server or the LAMP stack in my case, I used the following scripts and instructions.

```
#!/bin/bash

sudo apt update
sudo apt upgrade -y
```

```
sudo apt install apache2 -y
```

```
sudo ufw enable -y
```

```
sudo ufw allow in "Apache"
```

```
sudo ufw allow in "OpenSSH"
```

```
sudo apt install mysql-server -y
```

```
sudo mysql_secure_installation -y
```

Create a password to finish the installation. Default for the other questions(just hit enter).

Then,

```
sudo apt install phpmyadmin php-mbstring php-zip php-gd php-json php-curl -y
```

This installation will fail because the password is not in the format that php will be able to recognize.

To resolve this, follow the next few steps:

In terminal:

```
Sudo mysql
```

```
UNINSTALL COMPONENT "file://component_validate_password";
```

```
Exit
```

```
Sudo apt install phpmyadmin -y
```

```
Sudo mysql
```

```
INSTALL COMPONENT "file://component_validate_password";
```

```
Exit
```

```
sudo phpenmod mbstring
```

```
Sudo mysql
```

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH caching_sha2_password BY '<new password you set here>';
```

```
Exit
```

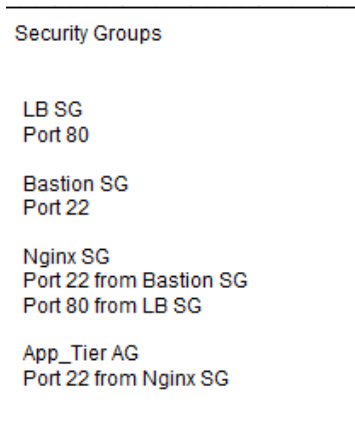
```
Sudo systemctl restart apache2
```

You will now be able to connect to the db via phpmyadmin.

Security Groups

AWS has security groups which act as firewalls between compute instances. I am able to control what traffic is allowed into the compute instance, give an extra layer of protection.

To ensure that the instances only take in traffic from sources we expect it to come from and to block out outsiders. For example, if the app server only accepts http traffic from the reverse proxy server, then other computers will not be able to access it.









Each security group is identified by their id, so only sources with a certain security group id can be allowed to access an EC2. As shown above, I was able to configure the permissions for certain ports for different EC2's.

Database

The AWS RDS service allows for different types of Databases that can be used. In this case, I chose MySQL because that is what my LAMP stack was already using. To include AWS RDS MySQL would be the same as adding in a different server for it to access at port 3306.

Engine options

Engine type [Info](#)

<input type="radio"/> Amazon Aurora 	<input checked="" type="radio"/> MySQL 	<input type="radio"/> MariaDB 
<input type="radio"/> PostgreSQL 	<input type="radio"/> Oracle 	<input type="radio"/> Microsoft SQL Server 

To allow for the connection between the LAMP EC2 and RDS, I created a subnet group. This attaches the Database to a subnet in multiple availability zones, making the data stores in the database available in different regions making sure it is highly available. This also allows for the EC2s in the subnet we select to be able to communicate with it. I had chosen my private subnets in availability zones us-east-1a and us-east-1b.

After that, I moved on to create the database. I selected the free tier for this deployment because I did not need that much functionality or features aside from the basic relational database.

I set a password for the login information. This is the login that will be used to access this DB. Then hit create. It will take a few minutes to make so in parallel, I logged into my LAMP EC2 and I made changes to the config.inc.php file to allow the EC2 to be able to connect.

```
Cd /etc/phpmyadmin  
Sudo nano config.inc.php
```

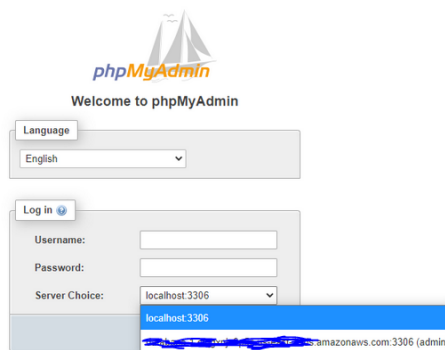
Jumping to line 103 (you can do this by pressing alt + G then typing 1-3 when prompted), I then pasted the below code into the file with information about my AWS RDS database inside.

```
$i++;  
$cfg['Servers'][$i]['host']      = '___FILL_IN_DETAILS___';  
$cfg['Servers'][$i]['port']     = '3306';  
$cfg['Servers'][$i]['socket']   = '';  
$cfg['Servers'][$i]['connect_type'] = 'tcp';  
$cfg['Servers'][$i]['extension'] = 'mysql';
```

```
$cfg['Servers'][$i]['compress'] = FALSE;  
$cfg['Servers'][$i]['auth_type'] = 'config';  
$cfg['Servers'][$i]['user'] = '__FILL_IN_DETAILS__';  
$cfg['Servers'][$i]['password'] = '__FILL_IN_DETAILS__';
```

The 'host' value will be the endpoint you can find in your AWS RDS console. 'User' and 'password' are the credentials set in the RDS creation process.

After those changes are made, save them. I then copy and pasted the dns name of the load balancer with /phpmyadmin to check if the connection is successful. If it is then there should be an option between localhost and the endpoint address on the screen.



The final topology looks like this:

Security Groups	
LB SG	Port 80
Bastion SG	Port 22
Nginx SG	Port 22 from Bastion SG Port 80 from LB SG
App_Tier AG	Port 22 from Nginx SG
IP Address Ranges	
VPC	192.168.100.0/24
AZ1	Public Subnet: 192.168.100.0/26 Private Subnet: 192.168.100.64/26
AZ2	Public Subnet: 192.168.100.128/26 Private Subnet: 192.168.100.192/26

