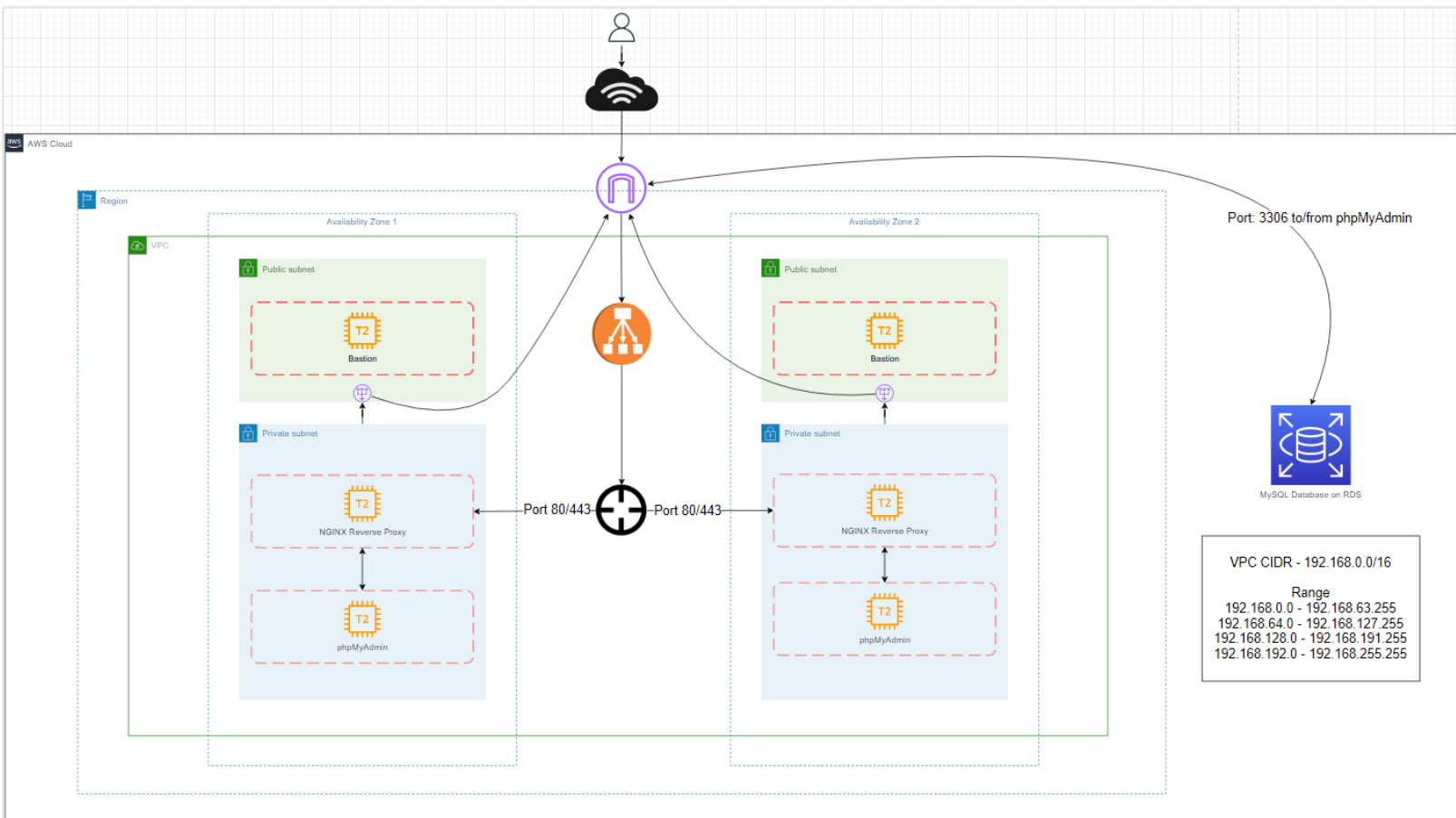




Three Tier Assignment



For this assignment, we automated the process of creating a 3 tier architecture utilizing CloudFormation which resulted in an elastic, reliable, and secure deployment.

A three-tier architecture consists of a presentation layer, an application layer, and a data layer. In this assignment, the presentation layer was an NGINX application that was configured to be a reverse proxy that served traffic on behalf of the backend application. The application layer was a phpMyAdmin application that talked to the RDS database and visualized the data. PhpMyAdmin handles the administration of MySQL databases over the web. This allows us to create, update, drop, alter, delete, import, and export MySQL database tables all through phpMyAdmin. Finally, the data layer is often called the database tier where the information processed by the application is stored and managed.

For this assignment, we created the majority of our AWS architecture using CloudFormation. All EC2 instances, security groups, subnets, and VPC were configured to minimize user error. The only services that need configuration are the load balancer and target group. Once everything was set up, we had to connect to the instances and install the required applications. The bastion EC2 instance is used as a jump host which will allow us to access our private resource only if we have access to the bastion EC2. This will add another layer of security to our architecture. For the database layer, we used MySQL RDS. We then had to configure the phpMyAdmin to connect to the RDS database. Once that was set up, we had to configure the NGINX application to be the reverse proxy. When everything is configured, you can access the application through the load balancer DNS name. Deleting the CloudFormation stack will tear down all the resources for us. We just had to delete the load balancer and target group. With the help of CloudFormation, we created an elastic, reliable, and secure deployment architecture.

Task 1

Create a VPC with a public subnet and a private subnet using CloudFormation

Create a YAML file called base.yaml

Paste the following inside the YAML file and save the file.

```
Description:
  This template deploys a VPC, with a pair of public and private subnets spread
  across two Availability Zones. It deploys an internet gateway, with a default
  route on the public subnets. It deploys a pair of NAT gateways (one in each AZ),
  and default routes for them in the private subnets.

Parameters:
  EnvironmentName:
    Description: An environment name that is prefixed to resource names
    Type: String

  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 192.168.0.0/16

  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the first Availability Zone
    Type: String
    Default: 192.168.0.0/18

  PublicSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the second Availability Zone
    Type: String
    Default: 192.168.64.0/18

  PrivateSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the first Availability Zone
    Type: String
    Default: 192.168.128.0/18

  PrivateSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the second Availability Zone
    Type: String
    Default: 192.168.192.0/18

  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access to the instance
    Type: AWS::EC2::KeyPair::KeyName

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
```

- Key: Name
Value: !Ref EnvironmentName

InternetGateway:
Type: AWS::EC2::InternetGateway
Properties:
Tags:
- Key: Name
Value: !Ref EnvironmentName

InternetGatewayAttachment:
Type: AWS::EC2::VPCGatewayAttachment
Properties:
InternetGatewayId: !Ref InternetGateway
VpcId: !Ref VPC

PublicSubnet1:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [0, !GetAZs ""]
CidrBlock: !Ref PublicSubnet1CIDR
MapPublicIpOnLaunch: true
Tags:
- Key: Name
Value: !Sub \${EnvironmentName} Public Subnet (AZ1)

PublicSubnet2:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [1, !GetAZs ""]
CidrBlock: !Ref PublicSubnet2CIDR
MapPublicIpOnLaunch: true
Tags:
- Key: Name
Value: !Sub \${EnvironmentName} Public Subnet (AZ2)

PrivateSubnet1:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [0, !GetAZs ""]
CidrBlock: !Ref PrivateSubnet1CIDR
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Sub \${EnvironmentName} Private Subnet (AZ1)

PrivateSubnet2:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [1, !GetAZs ""]
CidrBlock: !Ref PrivateSubnet2CIDR
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Sub \${EnvironmentName} Private Subnet (AZ2)

NatGateway1EIP:
Type: AWS::EC2::EIP
DependsOn: InternetGatewayAttachment
Properties:
Domain: vpc

NatGateway2EIP:
Type: AWS::EC2::EIP
DependsOn: InternetGatewayAttachment
Properties:
Domain: vpc

NatGateway1:
Type: AWS::EC2::NatGateway
Properties:
AllocationId: !GetAtt NatGateway1EIP.AllocationId
SubnetId: !Ref PublicSubnet1

NatGateway2:
Type: AWS::EC2::NatGateway
Properties:
AllocationId: !GetAtt NatGateway2EIP.AllocationId
SubnetId: !Ref PublicSubnet2

PublicRouteTable:
Type: AWS::EC2::RouteTable
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Sub \${EnvironmentName} Public Routes

DefaultPublicRoute:
Type: AWS::EC2::Route
DependsOn: InternetGatewayAttachment
Properties:
RouteTableId: !Ref PublicRouteTable
DestinationCidrBlock: 0.0.0.0/0

GatewayId: !Ref InternetGateway

PublicSubnet1RouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PublicRouteTable

SubnetId: !Ref PublicSubnet1

PublicSubnet2RouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PublicRouteTable

SubnetId: !Ref PublicSubnet2

PrivateRouteTable1:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

- Key: Name

Value: !Sub \${EnvironmentName} Private Routes (AZ1)

DefaultPrivateRoute1:

Type: AWS::EC2::Route

Properties:

RouteTableId: !Ref PrivateRouteTable1

DestinationCidrBlock: 0.0.0.0/0

NatGatewayId: !Ref NatGateway1

PrivateSubnet1RouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PrivateRouteTable1

SubnetId: !Ref PrivateSubnet1

PrivateRouteTable2:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

- Key: Name

Value: !Sub \${EnvironmentName} Private Routes (AZ2)

DefaultPrivateRoute2:

Type: AWS::EC2::Route

Properties:

RouteTableId: !Ref PrivateRouteTable2

DestinationCidrBlock: 0.0.0.0/0

NatGatewayId: !Ref NatGateway2

PrivateSubnet2RouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PrivateRouteTable2

SubnetId: !Ref PrivateSubnet2

NoIngressSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupName: "no-ingress-sg"

GroupDescription: "Security group with no ingress rule"

VpcId: !Ref VPC

BastionSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: "Security group that allows SSH from anywhere"

GroupName: "Bastion"

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 22

ToPort: 22

CidrIp: 0.0.0.0/0

VpcId: !Ref VPC

BastionEC2Instance:

Type: AWS::EC2::Instance

Properties:

ImageId: ami-09e67e426f25ce0d7

InstanceType: t2.micro

SubnetId: !Ref PublicSubnet1

KeyName: !Ref KeyName

SecurityGroupIds:

- !Ref BastionSecurityGroup

Tags:

- Key: "Name"

Value: "Bastion"

NginxSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: "Security group that allows SSH from bastion host only and allows client access on HTTP/HTTPS"

GroupName: "Nginx"

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 22

ToPort: 22

SourceSecurityGroupId:
Fn::GetAtt:
- BastionSecurityGroup
- GroupId
- IpProtocol: tcp
FromPort: 80
ToPort: 80
CidrIp: 0.0.0.0/0
- IpProtocol: tcp
FromPort: 443
ToPort: 443
CidrIp: 0.0.0.0/0
VpcId: !Ref VPC

NginxEC2Instance:
Type: AWS::EC2::Instance
Properties:
ImageId: ami-09e67e426f25ce0d7
InstanceType: t2.micro
SubnetId: !Ref PrivateSubnet1
KeyName: !Ref KeyName
SecurityGroupIds:
- !Ref NginxSecurityGroup
Tags:
- Key: "Name"
Value: "Nginx"

phpMyAdminSecurityGroup:
Type: AWS::EC2::SecurityGroup
Properties:
GroupDescription: "Security group that allows SSH from the bastion host only"
GroupName: "phpMyAdmin"
SecurityGroupIngress:
- IpProtocol: tcp
FromPort: 22
ToPort: 22
SourceSecurityGroupId:
Fn::GetAtt:
- BastionSecurityGroup
- GroupId
- IpProtocol: tcp
FromPort: 80
ToPort: 80
SourceSecurityGroupId:
Fn::GetAtt:
- NginxSecurityGroup
- GroupId
VpcId: !Ref VPC

phpMyAdminEC2Instance:
Type: AWS::EC2::Instance
Properties:
ImageId: ami-09e67e426f25ce0d7
InstanceType: t2.micro
SubnetId: !Ref PrivateSubnet1
KeyName: !Ref KeyName
SecurityGroupIds:
- !Ref phpMyAdminSecurityGroup
Tags:
- Key: "Name"
Value: "phpMyAdmin"

ThreeTierSecurityGroup:
Type: AWS::EC2::SecurityGroup
Properties:
GroupDescription: "Security group that allows client access on HTTP/HTTPS for the Load Balancer"
GroupName: "ThreeTier"
SecurityGroupIngress:
- IpProtocol: tcp
FromPort: 80
ToPort: 80
CidrIp: 0.0.0.0/0
- IpProtocol: tcp
FromPort: 443
ToPort: 443
CidrIp: 0.0.0.0/0
VpcId: !Ref VPC

ThreeTierDBSecurityGroup:
Type: AWS::EC2::SecurityGroup
Properties:
GroupDescription: "Security group for the RDS MySQL database that allows access from phpMyAdmin SG only"
GroupName: "ThreeTierDB"
SecurityGroupIngress:
- IpProtocol: tcp
FromPort: 3306
ToPort: 3306
SourceSecurityGroupId:
Fn::GetAtt:
- phpMyAdminSecurityGroup
- GroupId
VpcId: !Ref VPC

Outputs:
VPC:
Description: A reference to the created VPC
Value: !Ref VPC

PublicSubnets:
Description: A list of the public subnets
Value: !Join [",", [!Ref PublicSubnet1, !Ref PublicSubnet2]]

PrivateSubnets:
Description: A list of the private subnets
Value: !Join [",", [!Ref PrivateSubnet1, !Ref PrivateSubnet2]]

PublicSubnet1:
Description: A reference to the public subnet in the 1st Availability Zone
Value: !Ref PublicSubnet1

PublicSubnet2:
Description: A reference to the public subnet in the 2nd Availability Zone
Value: !Ref PublicSubnet2

PrivateSubnet1:
Description: A reference to the private subnet in the 1st Availability Zone
Value: !Ref PrivateSubnet1

PrivateSubnet2:
Description: A reference to the private subnet in the 2nd Availability Zone
Value: !Ref PrivateSubnet2

NoIngressSecurityGroup:
Description: Security group with no ingress rule
Value: !Ref NoIngressSecurityGroup

BastionSecurityGroup:
Description: Security group with SSH from anywhere ingress rule
Value: !Ref BastionSecurityGroup

NginxSecurityGroup:
Description: Security group with SSH from anywhere ingress rule
Value: !Ref NginxSecurityGroup

NginxSecurityGroup:
Description: Security group that allows SSH from bastion host only and allows client access on HTTP/HTTPS
Value: !Ref NginxSecurityGroup

phpMyAdminSecurityGroup:
Description: Security group with SSH from only the bastion SG ingress rule
Value: !Ref phpMyAdminSecurityGroup

ThreeTierSecurityGroup:
Description: Security group that allows client access on HTTP/HTTPS for the Load Balancer
Value: !Ref ThreeTierSecurityGroup

ThreeTierDBSecurityGroup:
Description: Security group for the RDS MySQL database that allows access from phpMyAdmin SG only
Value: !Ref ThreeTierDBSecurityGroup

Once you have created that file. Save it and go To AWS CloudFormation

Select Create a Stack

Create a CloudFormation stack

Use your own template or a sample template to quickly get started.

Create stack

Upload your YAML file that has your CloudFormation template.

Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Template is ready

☐ Use a sample template

☐ Create template in Designer

Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

☐ Amazon S3 URL

☒ Upload a template file

Upload a template file

Choose file

base.yaml

JSON or YAML formatted file

S3 URL: https://s3-external-1.amazonaws.com/cf-templates-1fxqp5dq624ei-us-east-1/2021313sHI-base.yaml

View in Designer

Cancel

Next

Select a Stack Name

Stack name

Stack name

ThreeTier

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).


Select your Key Pair that you will use to SSH into the ec2 instances.

KeyName
Name of an existing EC2 KeyPair to enable SSH access to the instance

rixardo

rixardo

Once you have done that, just proceed with default Stack configuration.

A rectangular button with an orange background and the text "Create stack" in white.

**We can SSH into the Bastion EC2 instance to make sure everything is working
(The publicIPv4 is from the Bastion EC2)**

```
ssh -i key.pem ubuntu@publicIPv4  
sudo apt-get update && sudo apt-get upgrade -y
```

**Once inside that instance, you will need to create a private key that has your key pair that
will allow you to SSH into the other instance.**

```
nano key.pem
```

**Paste the key value inside the file and save it. We will then need to change the
permissions of the file**

```
chmod 400 key.pem
```

**We can SSH into the NGINX EC2 instance to make sure everything is working
(The privateIPv4 is from the NGINX EC2)**

```
ssh -i key.pem ubuntu@privateIPv4  
sudo apt-get update && sudo apt-get upgrade -y
```

Once this EC2 has updated, we can exit it and SSH into the next EC2

```
exit
```

**We can SSH into the phpMyAdmin EC2 instance to make sure everything is working
(The privateIPv4 is from the phpMyAdmin EC2)**

```
ssh -i key.pem ubuntu@privateIPv4  
sudo apt-get update && sudo apt-get upgrade -y
```


Task 2

Create an AWS Application Load Balancer to connect to your reverse proxy.

First, we will need to create a Target Group,

▼ Load Balancing

Load Balancers

Target Groups New

Create a target group

Create target group

For configuration, we will need choose the Instance for target type

Basic configuration

Settings in this section cannot be changed after the target group is created.

Choose a target type



Instances

- Supports load balancing to instances within a specific VPC.

We then need to name the target group and choose HTTP protocol

Target group name

ThreeTierTG

A maximum of 32 alphanumeric characters including hyphens

Protocol

Port

HTTP ▼

:

80

When we select the VPC, make sure you select the one we created (starts in 192)

VPC

Select the VPC with the instances that you want to include in the target group.

-

vpc-0238db5085275ac32
IPv4: 192.168.0.0/16

▼

Protocol version

- ☒ **HTTP1**
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.
- ☐ **HTTP2**
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.
- ☐ **gRPC**
Send requests to targets using gRPC. Supported when the request protocol is gRPC.

Leave the defaults and go to the next step.

In the next step, we will need to select the EC2 instance we want to target. This will be the NGINX EC2 that will be our reverse proxy. Once selected, choose include as pending below.

Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic t

Available instances (1/3)

Filter resources by property or value

	Instance ID	Name	State	Security groups
<input type="checkbox"/>	i-0cb61ed73dc49ff9a	phpMyAdmin	running	phpMyAdmin
<input checked="" type="checkbox"/>	i-0a05cc9a1dd1b009a	Nginx	running	Nginx
<input type="checkbox"/>	i-069337359a90ef080	Bastion	running	Bastion

1 selected

Ports for the selected instances
Ports for routing traffic to the selected instances.
80
1-65535 (separate multiple ports with commas)

Include as pending below

We can then create the target group

Create target group

We will now need to configure a Load Balancer. Go to Load Balancing

▼ Load Balancing

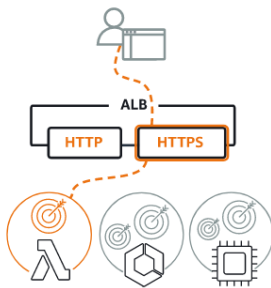
Load Balancers

Create a Load Balancer

Create Load Balancer

Select an Application Load Balancer

Application Load Balancer [Info](#)



Name the Load balancer, select Internet facing, and IPv4 as address type.

Basic configuration

Load balancer name

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

ThreeTier

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

Scheme [Info](#)

Scheme cannot 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](#)

☐ Internal

An internal load balancer routes requests from clients to targets using private IP addresses.

IP address type [Info](#)

Select the type of IP addresses that your subnets use.

☒ IPv4

Recommended for internal load balancers.

☐ Dualstack

Includes IPv4 and IPv6 addresses.

For Network mapping, select the VPC that we created (the IPv4 should start with 192)

VPC [Info](#)

Select the virtual private cloud (VPC) for your targets. Only VPCs with an internet gateway are enabled for selection. The selected VPC cannot be changed after confirm the VPC for your targets, view your [target groups](#).

-

vpc-0238db5085275ac32
IPv4: 192.168.0.0/16

▼

↺

We then need to select two public subnets for the Mapping...

Mappings [Info](#)

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in the balancer or the VPC are not available for selection. Subnets cannot be removed after the load balancer is created.

☒ **us-east-1a**

Subnet

subnet-0966e1f84da610bf4Public Subnet (AZ1) ▼

IPv4 settings

Assigned by AWS

☒ **us-east-1b**

Subnet

subnet-01561230ea4c9ed71Public Subnet (AZ2) ▼

IPv4 settings

Assigned by AWS

We will need to select the security group that allows HTTP and HTTPS . This was created during the CloudFormation stack creation. It's called "ThreeTier"

Security groups [Info](#)

A security group is a set of firewall rules that control the traffic to your load balancer.

Security groups

Select security groups

Create new security group [↗](#)

ThreeTier sg-0385ab798c35a71a2 [✕](#)
VPC: vpc-0293cc05635099b8f

For Listeners and routing, we need to Forward traffic to our Target Group

▼ Listener HTTP:80

Protocol

HTTP ▼

:

Port

80

1-65535

Default action [Info](#)

Forward to

ThreeTierTG
Target type: Instance, IPv4

HTTP ▼

[Create target group](#) [↗](#)

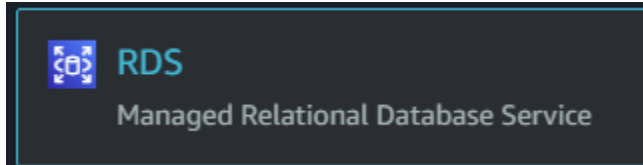
We can then create out load balancer

Create load balancer

Task 3

We will now need to create a MYSQL database

Go to AWS RDS



We will need to create a subnet group

Select Subnet Groups in the left

Subnet groups

Create a DB Subnet Group

Create DB Subnet Group

When configuring the Subnet Group, make sure to select the correct VPC that we created. We also need to put a brief name and description

Subnet group details

Name

You won't be able to modify the name after your subnet group has been created.

phpMyAdmin

Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and p

Description

ThreeTier Assignment

VPC

Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You cannot change the VPC identifier after your subnet group has been created.

(vpc-0238db5085275ac32)

For the next step, we will need to add subnets. Select 2 availability zones us-east-1a, and us-east-1b.

Availability Zones

Choose the Availability Zones that include the su

Choose an availability zone

us-east-1a ✕

us-east-1b ✕

We then need to select two PRIVATE Subnets (You can find the private subnet IP ranges inside AWS VPC service -> Subnet Association)

Subnets

Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.

Select subnets

subnet-016c383b6c88ccd76 (192.168.128.0/18) ✕

subnet-09248983277f6fb91 (192.168.192.0/18) ✕

Once we configured the Subnet Group, we can create it.

Create

We can now create our database. Go back to the Dashboard

Amazon RDS

Dashboard

Create a database

Create database

The creation method should be standard

Choose a database creation method [Info](#)

- ☒ **Standard create**
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

We will select MySQL as the Engine type

Engine options

Engine type [Info](#)

- ☐ Amazon Aurora



- ☒ MySQL



It is important to select Free Tier

Templates

Choose a sample template to meet your use case.

- ☐ **Production**
Use defaults for high availability and fast, consistent performance.

- ☐ **Dev/Test**
This instance is intended for development use outside of a production environment.

- ☒ **Free tier**
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. [Info](#)

When configuring the database, it is important to write down the password. We can name it and leave the username as admin.

Settings

DB instance identifier [Info](#)

Type a name for your DB instance. The name is unique in the Region.

The DB instance identifier is case-insensitive, but cannot contain characters or hyphens. First character must be a letter.

▼ Credentials Settings

Master username [Info](#)

Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter.

☐ Auto generate a password
Amazon RDS can generate a password for you.

Master password [Info](#)

Constraints: At least 8 printable ASCII characters (including uppercase and lowercase letters, numbers, and symbols).

Confirm password [Info](#)

Password: KuraLabs123\$

Scroll down to Connectivity and select the VPC that we created

Virtual private cloud (VPC) [Info](#)

VPC that defines the virtual networking environment for the database instance.

Select the Subnet Group that we created

Subnet group [Info](#)

DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

phpmyadmin ▼

Public access [Info](#)

☐ Yes

Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more \ specify which EC2 instances and devices inside the VPC can connect to the database.

☒ No

RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside t your database.

We will need to select the Security Group for the database that was created in CloudFormation.

Existing VPC security groups

Choose VPC security groups ▼

ThreeTierDB ✕

We then need to select an Availability Zone

Availability Zone [Info](#)

us-east-1a

We can leave everything else default and create the database

Create database

This will take a couple of minutes to create, you can move on to the next task

Task 4

We will need to set up phpMyAdmin on our EC2 and connect to our MySQL database

Connect to your phpMyAdmin EC2 instance
ssh into the bastion EC2 -> ssh into the phpMyAdmin EC2

Run the following command

```
sudo apt-get update && sudo apt-get upgrade -y
```

Download apache2

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

Install PHP and module that will have php connect with apache and php connect to mysql server.

```
sudo apt install php libapache2-mod-php php-mysql -y
```

We will need to check that our PHP is working

We need to change directory to where apache host web pages
`cd /var/www/html`

Create a PHP file

```
sudo nano test.php
```

Paste the following into the file and save it

```
<?php phpinfo();
```

Install MySQL server

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

Run the basic MYSQL installation

```
sudo mysql_secure_installation
```

```
Y
```

```
1
```

```
Password for root user mysql: same as rds database KuraLabs123$
```

```
Y
```

```
<ENTERKEY>
```

```
<ENTERKEY>
```

```
<ENTERKEY>
```

```
<ENTERKEY>
```

Enter into the interactive shell of mysql to check if installation was successful.

`sudo mysql`

Enter the following command inside the interactive shell

`show databases;`

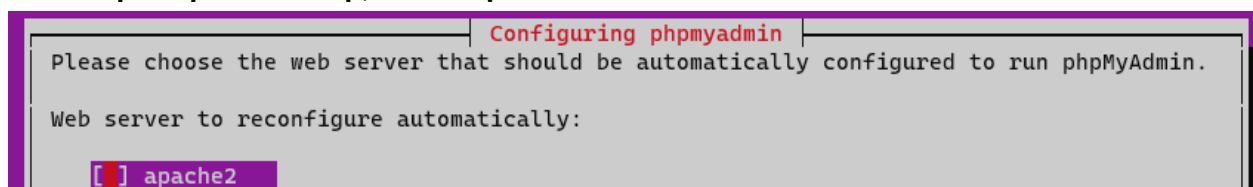
Exit the interactive shell

`exit`

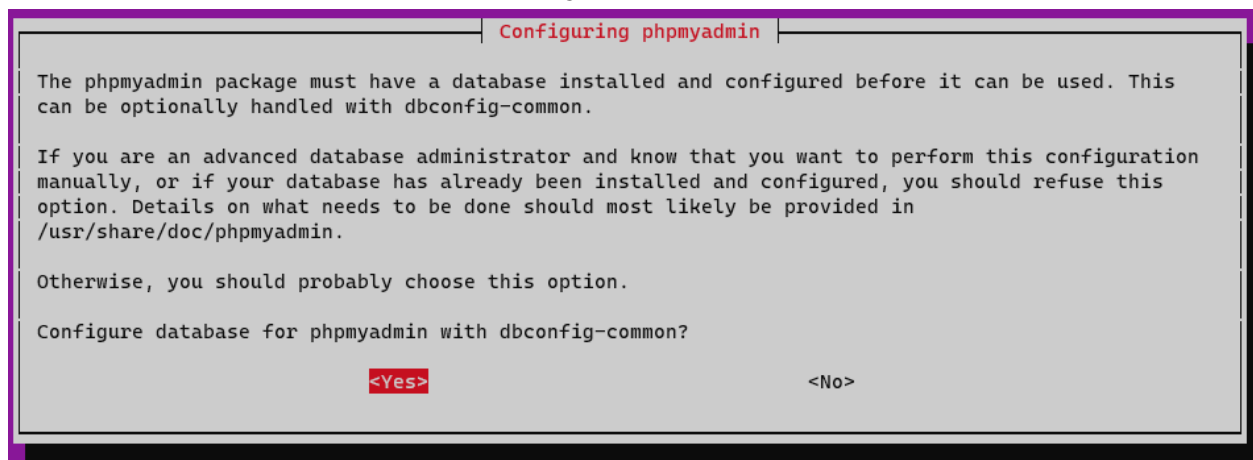
We will need to download some other necessary packages

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

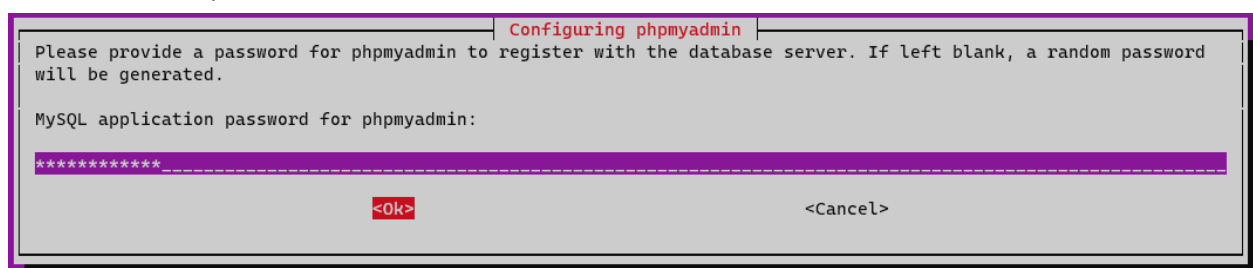
When a prompt comes up, select apache2



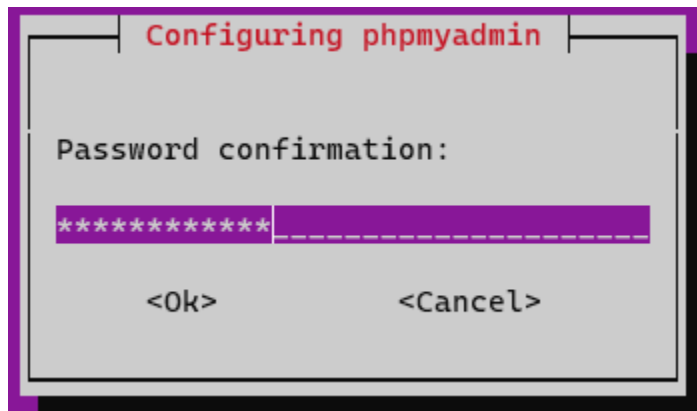
When the next prompt comes up, select yes



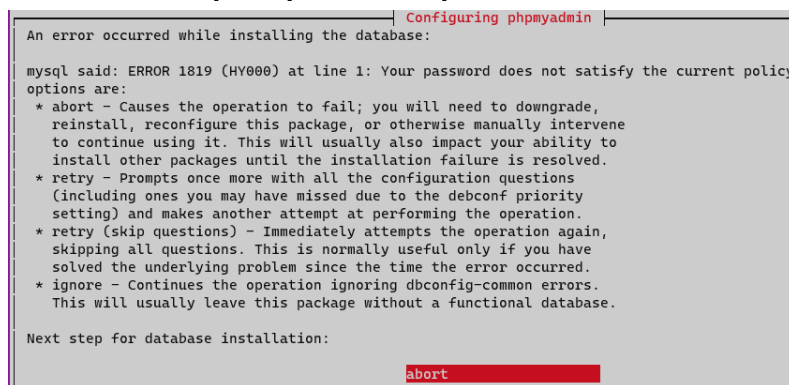
When the next prompt comes, we will have to enter a password. We can use KuraLabs123\$ from the database we created



Re-enter the same password



When the next prompt comes up select ok then choose abort.



Log back into mysql

`sudo mysql`

Paste the following inside the MySQL interactive shell

`SELECT user,authentication_string,plugin,host FROM mysql.user;`

Inside the table, the root should be empty



Run the following command inside the MySQL interactive shell

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

We can then run the next command inside the MySQL interactive shell and exit

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

`exit`

Install the following packages

sudo phpenmod mbstring

Go back into the MySQL

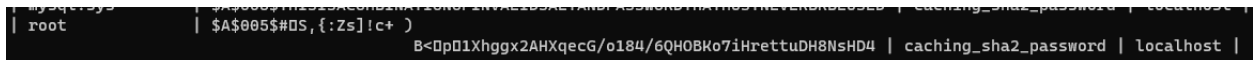
sudo mysql

We can use the following command which will use a hashing algorithm to encrypt our password and store it into the root localhost field.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH caching_sha2_password BY 'KuraLabs123$';
```

We can check if the changes were made using the following command

```
SELECT user,authentication_string,plugin,host FROM mysql.user;
```



```
| mysql:sys | $A$005$#05,{:Zs]!c+ ) | B<0p01Xhggx2AHXqecG/o184/6QH0Bko7iHrettuDH8NsHD4 | caching_sha2_password | localhost |
```

Exit the MySQL interactive shell

exit

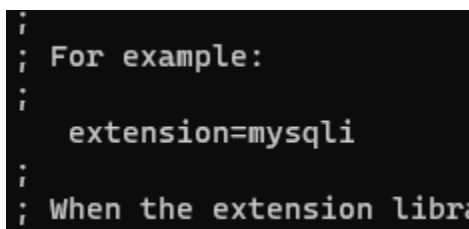
Change directory to the following

cd /etc/php/7.4/apache2/

We will need to edit a file...

sudo nano php.ini

Inside Nano select ALT + G. This will allow us to go to a line. Go to line 895 and remove the semicolon ;



```
;
; For example:
;
; extension=mysql
;
; When the extension libra
```

Save the file and exit it

CTRL + O

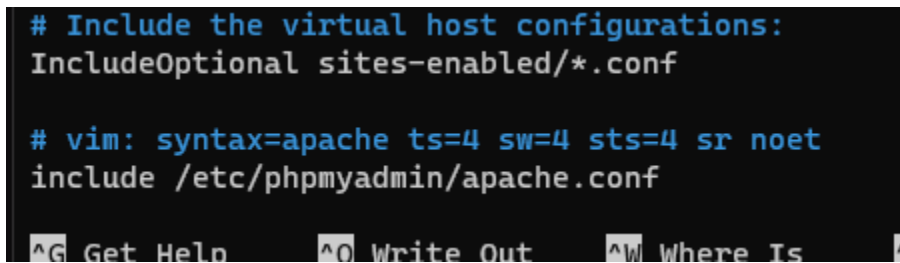
CTRL + X

We will edit the apache2 config file

sudo nano /etc/apache2/apache2.conf

Scroll all the way to the bottom and paste the following

```
include /etc/phpmyadmin/apache.conf
```



```
# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
include /etc/phpmyadmin/apache.conf

^G Get Help  ^O Write Out  ^W Where Is  ^A
```

Save the file and exit it

CTRL + O

CTRL + X

We will need to restart apache

```
sudo systemctl restart apache2
```

The PHP application is the PHMyAdmin that lets us interact with the database.

We will need to connect our MySQL database hosted on AWS to our phpmyadmin

Edit the config file

```
sudo nano /etc/phpmyadmin/config.inc.php
```

Inside nano select ALT + G. This will allow us to go to a line. Go to line 102 and paste the following below

```
$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__';
```

We will have to enter our information in the lines that read __FILL_IN_DETAILS__

Host is the endpoint URL found on the AWS RDS database we created

Enter the username and password in the user and password line of the code

```

/* Advance to next server for rest of config */
$i++;
}
$i++;
$cfg['Servers'][$i]['host']      = 'threetierdb.██████████.us-east-1.rds.amazonaws.com';
$cfg['Servers'][$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']      = 'admin';
$cfg['Servers'][$i]['password']  = 'KuraLabs123$';

/* Authentication type */

```

^{^G} Get Help ^{^O} Write Out ^{^W} Where Is ^{^K} Cut Text ^{^J} Justify ^{^C} Cur Pos ^{M-U} Undo

We can now test our connection to the new database. We should be able to log into your AWS RDS

curl localhost:80/phpmyadmin/

Restart nginx

sudo systemctl restart apache2

We can now configure the NGINX Proxy. Exit the phpMyAdmin EC2

exit

Connect to your NGINX EC2 instance

ssh into the bastion EC2 -> ssh into the NGINX EC2

Run the following command

sudo apt-get update && sudo apt-get upgrade -y

Install NGINX

sudo apt-get install nginx -y

Change directories to Sites available

sites-available are conf files that tell NGINX where to look for.

cd /etc/nginx/sites-available/

We need to unlink the default sites-enabled file

sudo unlink /etc/nginx/sites-enabled/default

sudo unlink /etc/nginx/sites-enabled/reverse-proxy.conf

Unlinking the reverse-proxy.conf will say there is no file. We need to create a configuration file for the reverse proxy

sudo nano reverse-proxy.conf

Paste the following inside the reverse-proxy configuration file (The proxy_pass IP is the phpMyAdmin private IPv4)

```
server {  
    listen 80;  
    location / {  
        proxy_pass http://192.168.163.173;  
    }  
}
```

Save the file and exit it

CTRL + O

CTRL + X

Check if the following directory is empty

ls /etc/nginx/sites-enabled/

We will link reverse-proxy to sites enabled so that apache can read it and use it. (ONE LINE COMMAND)

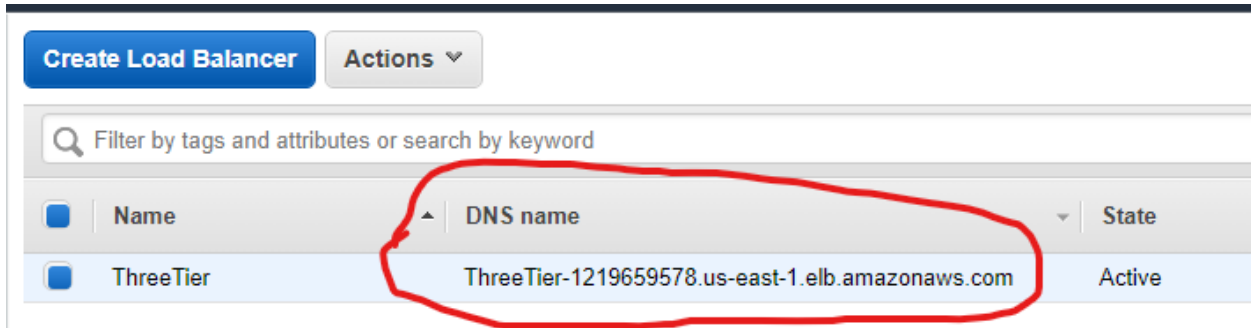
```
sudo ln -s /etc/nginx/sites-available/reverse-proxy.conf  
/etc/nginx/sites-enabled/reverse-proxy.conf
```

We will need to restart NGINX

```
sudo systemctl restart nginx
```

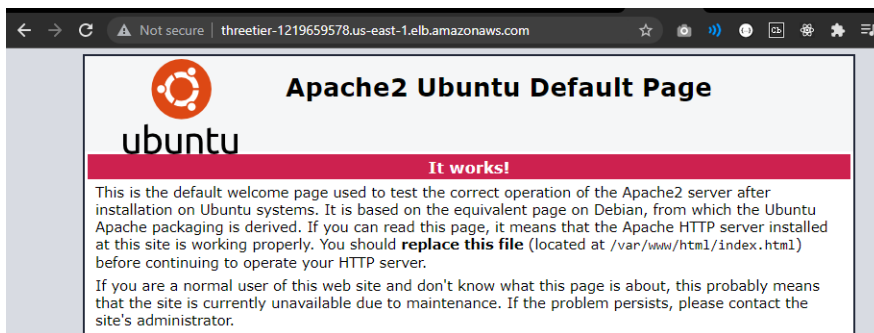
Task 5

Access your application. Go back to the Load balancer on AWS

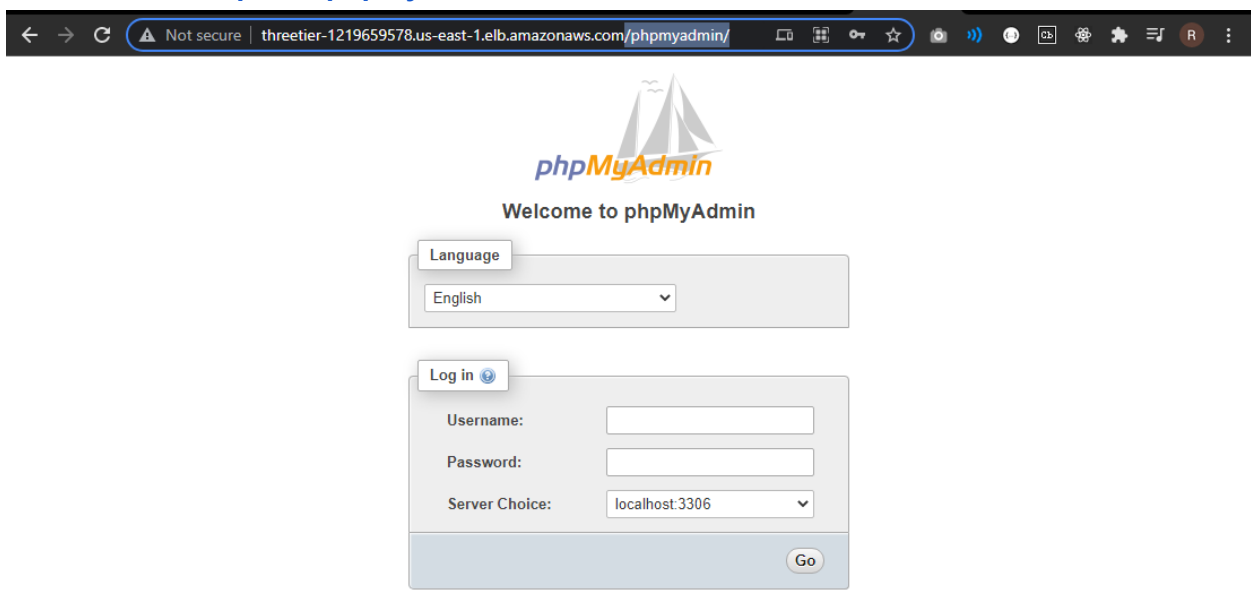


Select the DNS name and paste it into your browser.

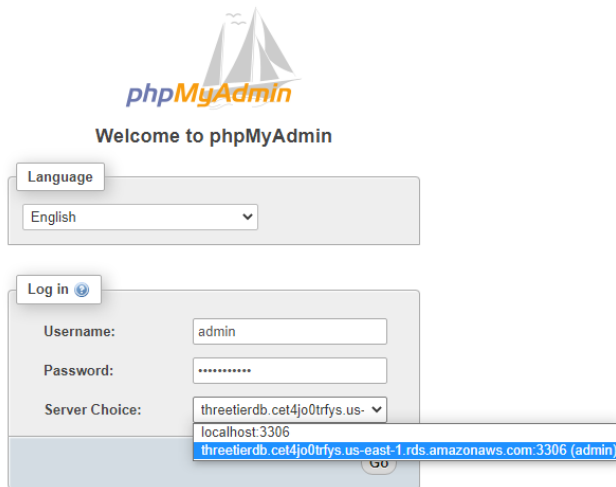
You should see an apache2 default page



To access our phpMyAdmin application, we will have to put a route in the URL. The format will be <http://url/phpmyadmin/>.



You will see the RDS has been configured in the server choice.

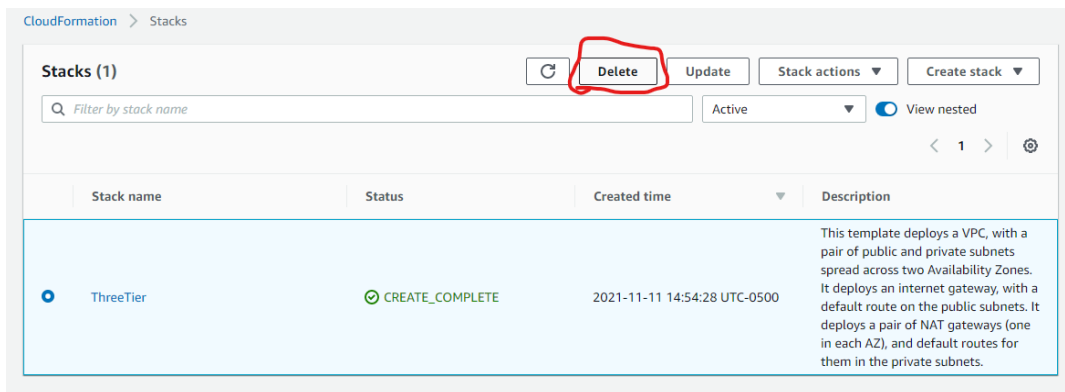


The image shows the phpMyAdmin login page. At the top is the phpMyAdmin logo and the text "Welcome to phpMyAdmin". Below this is a "Language" dropdown menu set to "English". Underneath is a "Log in" button with a help icon. The login form contains three fields: "Username:" with the value "admin", "Password:" with masked characters, and "Server Choice:" with a dropdown menu. The dropdown menu is open, showing three options: "threetierdb.cet4jo0trfys.us-", "localhost:3306", and "threetierdb.cet4jo0trfys.us-east-1.rds.amazonaws.com:3306 (admin)". The third option is highlighted in blue.

Once you logged into the database and can access it. You have completed the assignment!

Tear Down Time!

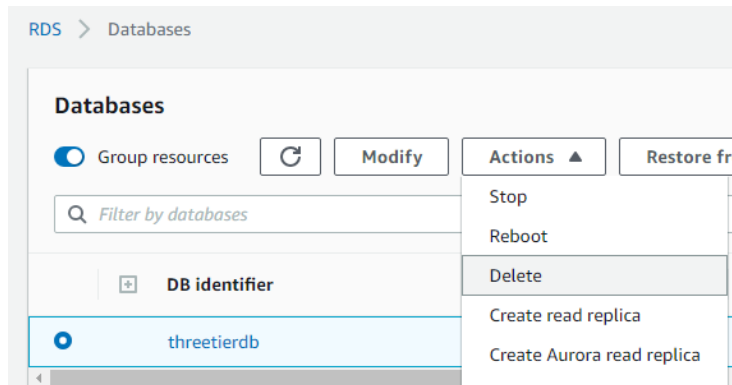
Go back into AWS CloudFormation. Select your Stack and delete it



The image shows the AWS CloudFormation "Stacks" page. At the top, there's a "Stacks (1)" header with a search bar "Filter by stack name", a "Delete" button (circled in red), an "Update" button, a "Stack actions" dropdown, and a "Create stack" dropdown. Below the header is a table with columns: "Stack name", "Status", "Created time", and "Description". There is one stack listed: "ThreeTier" with status "CREATE_COMPLETE" and created time "2021-11-11 14:54:28 UTC-0500". The description for the stack is: "This template deploys a VPC, with a pair of public and private subnets spread across two Availability Zones. It deploys an internet gateway, with a default route on the public subnets. It deploys a pair of NAT gateways (one in each AZ), and default routes for them in the private subnets."

Delete your Target Group and Load Balancer

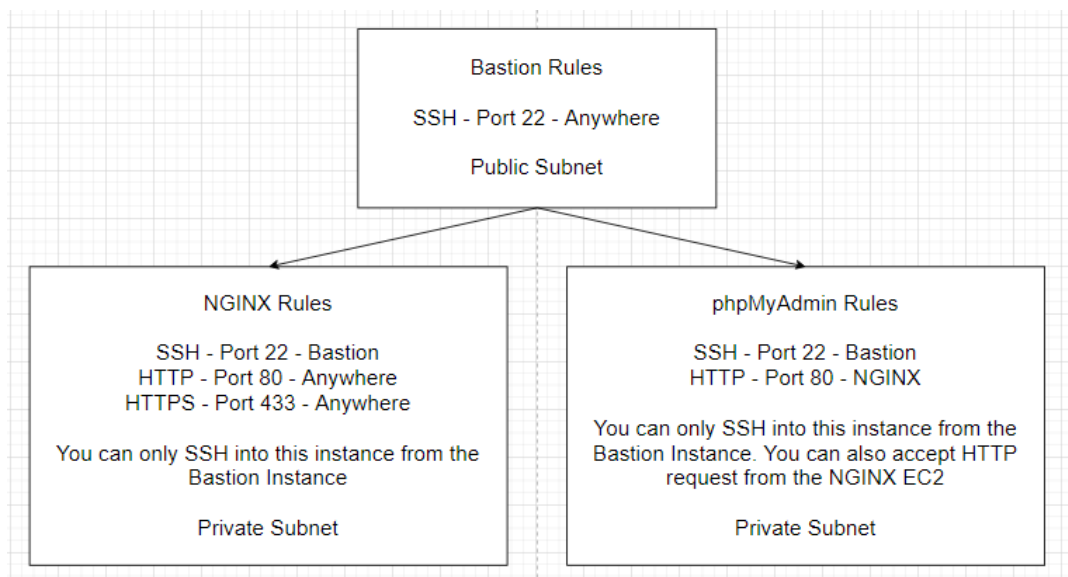
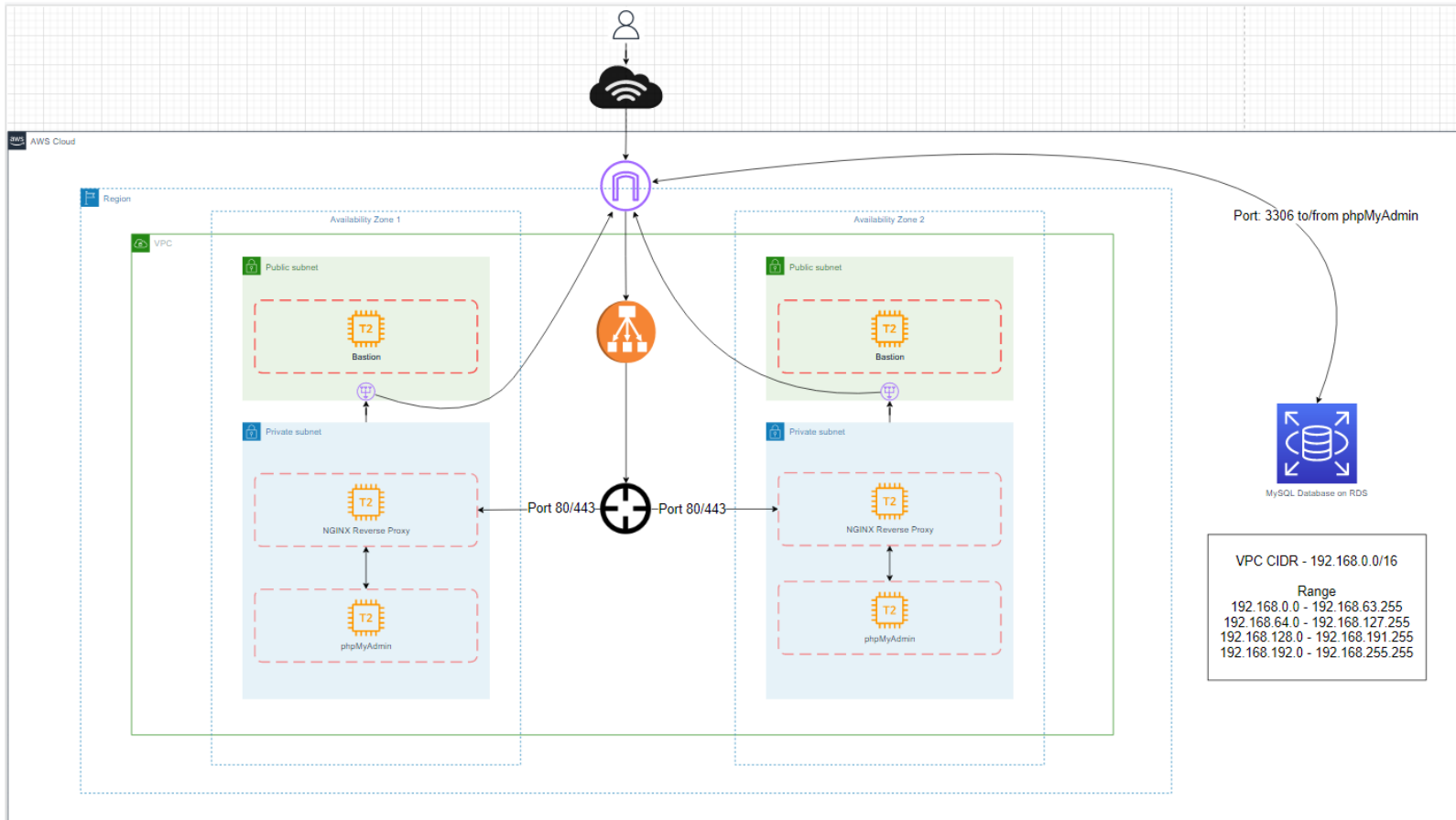
Go into AWS RDS and delete the database that we created (threetierdb)



We can also delete the Subnet Group, phpMyAdmin, that we created once the Database has been deleted.

Make sure to check if the CloudFormation stack was deleted after 10minutes.

Topology



The bastion host allows us to get access to a private network from an external network. Bastion hosts are used to mitigate the risk of allowing SSH connections to our main application. This will minimize the chance of penetrating our main application. We can also use monitoring tools to see who accesses our bastion host.

Traffic is then sent through an internet gateway which is then sent to an application load balancer. The load balancer has a target group that will tell where specifically the traffic should be sent. In this case, traffic is sent to the NGINX ec2 Instance. This instance has a reverse proxy which talks to the phpMyAdmin EC2 which has a phpMyAdmin application. The phpMyAdmin application talks to the RDS MySQL database through port 3306 and connection details that were configured.

The NGINX server and phpMyAdmin server reach out to the internet gateway to get updates such as security and application updates. Our architecture needs to be configured using NAT gateways in the public subnets. NAT Gateways allows our instances to access the internet without exposing the instances to incoming connections. If our applications are not updated, the server will be left vulnerable.

Whenever the private subnet instances need to access the internet, it goes to the NAT Gateway which is inside the public subnet. The request is then sent to the internet gateway which accesses the internet.