**AWS 3 Tier Architecture**

How to create and test a web tier, application tier, and database tier for a highly available and scalable architecture? AWS offers comprehensive resources to build and maintain your cloud applications.

These resources can be configured to work together to create highly available, highly reliable cloud structures.

The scenario we will use is that you have been asked to design and create a highly available 3 tier architecture for your company’s new web application.

This Session is quite long, but also thorough, please pay attention in detail.

**What is a 3 Tier Architecture**

A 3-tier architecture consists of a presentation tier, an application tier, and a data tier.

The ***presentation tier***houses the user interface, such as the website that a user or client navigates to. It can also be thought of as the “front end.”

The ***application tier*** is where data is processed and is often called the “back end.”

The ***data tier***is where data is stored and managed.

**Benefits of a 3 Tier Architecture**

* ***Decreased development time***— different teams can work on different tiers simultaneously, resulting in decreased time to deploy.
* ***Increased scalability*** — a tier can have an auto-scaling group independent of other tiers, meaning for each tier, you only use what you need.
* ***Increased reliability***— each tier can have multiple resources in multiple availability zones and the success and availability of one tier is independent of the other tiers.
* ***Increased security*** — each tier can have its own security group, allowing for custom permissions depending on the needs of that tier.

**Steps to be followed:**

Intro

AWS Networking

Three-tier app architecture

Create VPC

Create Subnets

Create route tables

Route table subnet associations

Create internet gateway

Create NAT gateway

Add routes for IGW, NAT

Create a jump server

Create PHP servers

Installation of PHP, Apache

Install phpMyAdmin (Sample app)

Create and configure an application load balancer

Create RDS instance

Configure phpMyAdmin with RDS

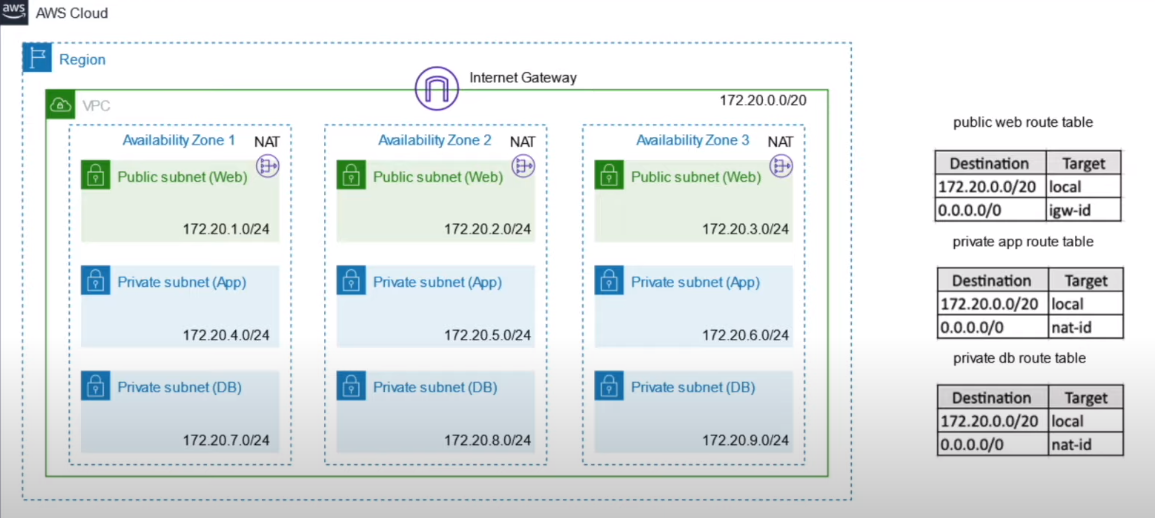
Configure session stickiness

Services recap

**Intro:** This session is on deploying a 3-tier web application on AWS.

**AWS Networking:**

* Before we can actually begin deploying the application components, we will need to create the networking base for the same.

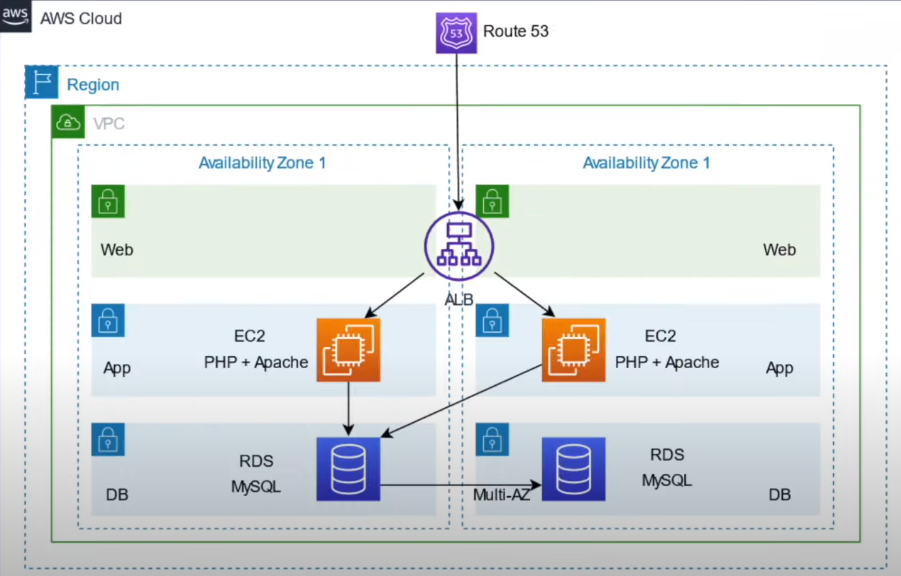


* In this case, we will create a ***VPC, 3 Public Subnets .ie., 1 subnet in each Availability Zone.*** These subnets will be having our internet facing load-balancers, web servers and jump servers (Bastion hosts).
* We will then create ***Private App Subnet*** in each availability zone for application deployment.
* And finally, we will create ***Private Subnets*** for DB instances.
* Once the VPC and Subnets are done, we will require ***Gateways***to configure incoming and outgoing traffic.
* Firstly ***Internet Gateway*** needs to be configured with the ***VPC*** and then and a ***NAT Gateway*** to be created in any of the ***Public Subnet.***
* As a best practice, it is recommended to create multiple NAT Gateways for redundancy. But for this demo, we will stick with single NAT Gateway.

People often used to make this mistake of configuring ***NAT Gateway***  in ***Private Subnet***, although the NAT Gateway is used facilitated to outgoing internet traffic for ***Private Subnet,*** it needs to reside in the ***Public Subnet.***

* After the Gateways are created, we need to create corresponding ***Route-Tables*** for Web, App & DB subnets.

**Three-tier app architecture:**

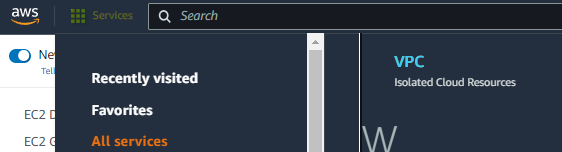


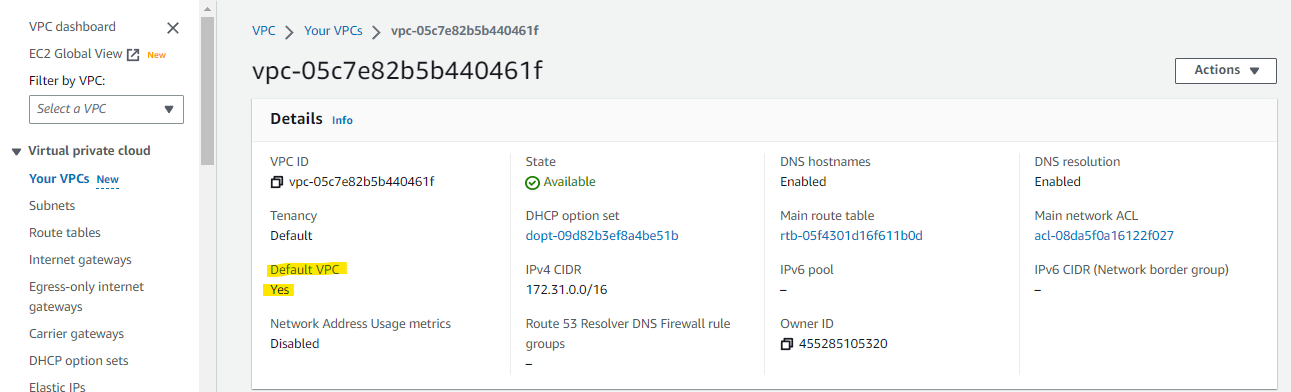
* Once the underlying networking infrastructure to host the 3-tier application is created, we can start with the application components.
* For this use case, Route 53 is optional as we can directly access the application via ***Application Load Balancer.***
* Then, we will create two EC2 instances on both Availability Zones and we will install PHP & Apache on it.
* And finally, we will create the RDS instances. Here the Multi-AZ configuration is optional.

Now, let’s jump into the actual implementation.

**Create VPC:**

* Under ***Services***  section, lets go to VPC and click on VPC id, you will notice that the default VPC is Yes . We are not going to use this VPC. Instead we will create our own VPC.



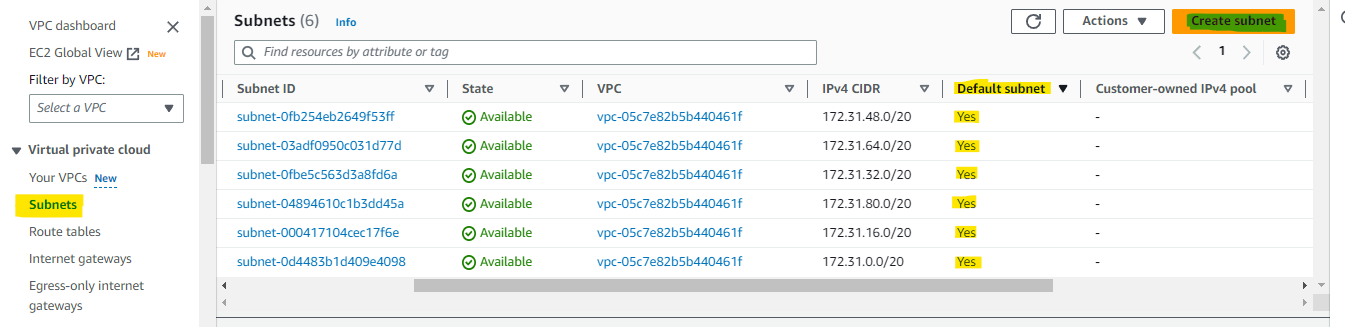


* Create VPC
  + Name the VPC as ***my-vpc***
  + The CIDR would be ***172.20.0.0/20***
  + Keep ***IPV6 CIDR block***
  + Keep the ***Tenancy***  as *Default.*
  + Click on ***Create VPC***

We will quickly verify ***VPC***  is created as expected, then we will move onto the ***Subnets.***

**Create Subnets:**

* Create Subnets
  + VPC --> Subnets
* Here also we notice that the subnets are created. But we will not use that as the subnets are Default.



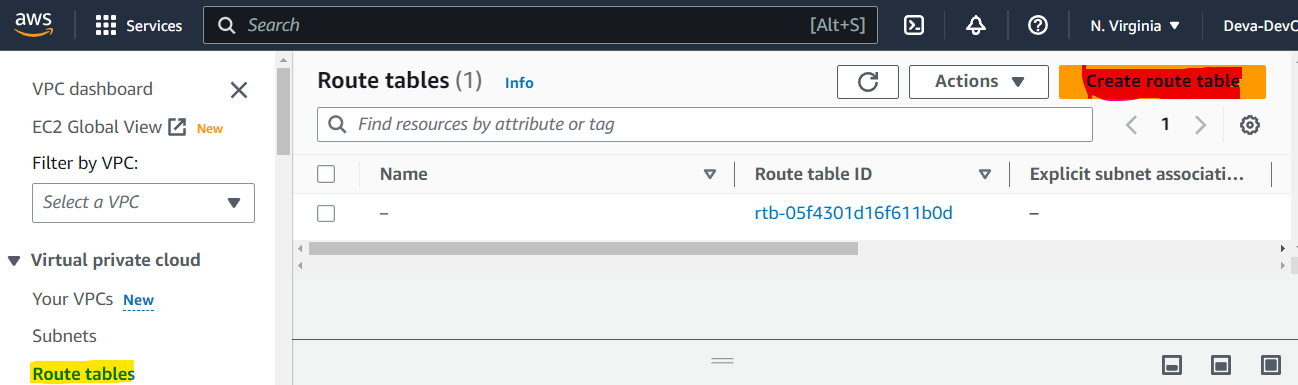
* So we do ***Subnets*** creation.
  + Create Subnet
  + Give the name ***my-public-web-subnet-1***
  + Select the **A*vailability Zone*** as ***us-east-2a***
  + Put the ***IPV4 CIDR block*** as ***172.20.1.0/24***
  + Create Second subnet by clicking on ***Add new subnet***
  + Give the name ***my-public-web-subnet-2***
  + Select the **A*vailability Zone*** as ***us-east-2b***
  + Put the ***IPV4 CIDR block*** as ***172.20.2.0/24***
  + Create Third subnet by clicking on ***Add new subnet***
  + Give the name ***my-public-web-subnet-3***
  + Select the **A*vailability Zone*** as ***us-east-2c***
  + Put the ***IPV4 CIDR block*** as ***172.20.3.0/24***
* Quickly verify if the subnets as as expected and once that is done, we will move with creation of App Subnet.
  + Create Fourth subnet by clicking on ***Add new subnet***
  + Give the name ***my-private-app-subnet-1***
  + Select the **A*vailability Zone*** as ***us-east-2a***
  + Put the ***IPV4 CIDR block*** as ***172.20.4.0/24***
  + Create Fifth subnet by clicking on ***Add new subnet***
  + Give the name ***my-private-app-subnet-2***
  + Select the **A*vailability Zone*** as ***us-east-2b***
  + Put the ***IPV4 CIDR block*** as ***172.20.5.0/24***
  + Create Sixth subnet by clicking on ***Add new subnet***
  + Give the name ***my-private-app-subnet-3***
  + Select the **A*vailability Zone*** as ***us-east-2c***
  + Put the ***IPV4 CIDR block*** as ***172.20.6.0/24***
* Now let’s move the last tier i.e; App Tier.
  + Create Seventh subnet by clicking on ***Add new subnet***
  + Give the name ***my-private-db-subnet-1***
  + Select the **A*vailability Zone*** as ***us-east-2a***
  + Put the ***IPV4 CIDR block*** as ***172.20.7.0/24***
  + Create Eight subnet by clicking on ***Add new subnet***
  + Give the name ***my-private-db-subnet-2***
  + Select the **A*vailability Zone*** as ***us-east-2b***
  + Put the ***IPV4 CIDR block*** as ***172.20.8.0/24***
  + Create 9th (last )subnet by clicking on ***Add new subnet***
  + Give the name ***my-private-db-subnet-3***
  + Select the **A*vailability Zone*** as ***us-east-2c***
  + Put the ***IPV4 CIDR block*** as ***172.20.9.0/24***

Now, quickly verify if all the subnets are defined/created as expected. And click on ***Create Subnet*** option.

Verify if all the defined subnets are created as expected or not.

**Create route tables:**

Once the subnets are created, lets move to create ***Route Table***



* Create route tables:
  + VPC --> Route Tables
  + The first ***Route Table*** would be ***my-public-web-route-table***
  + Select the VPC (***my-vpc***) that we have created.
  + Click on ***Create route table***
  + The second ***Route Table*** would be ***my-private-app-route-table***
  + Select the VPC (***my-vpc***) that we have created.
  + Click on ***Create route table***
  + The last ***Route Table*** would be ***my-private-db-route-table***
  + Select the VPC (***my-vpc***) that we have created.
  + Click on ***Create route table***

Quickly verify if the route tables are created.

**Route table subnet associations:**

Once the ***Route Tables*** are created, we need to associate them with appropriate ***Subnets.***

* + VPC --> Route tables --> Subnet associations --> Edit subnet associations
  + Select the web route tables with web subnets.
  + Click on ***Save associations***

Do the same for App route tables as well

* + VPC --> Route tables --> Subnet associations --> Edit subnet associations
  + Select the app route tables with app subnets.
  + Click on ***Save associations***

And finally the Db route table

* + VPC --> Route tables --> Subnet associations --> Edit subnet associations
  + Select the db route tables with db subnets.
  + Click on ***Save associations***

**Create internet gateway:**

Once the ***route tables***  are created, we need to establish the connectivity, for that we need ***Internet gateway*** for ***Public web subnet***

* Create internet gateway
  + VPC --> Internet Gateways --> Create Internet Gateway
  + Name the ***Internet Gateway*** as**my-internet-gateway**
  + Click on ***Create Internet Gateway***
* Once the Internet Gateway is created, we have to attach it to the VPC. Let’s do that.
  + VPC --> Internet gateways --> ***my-internet-gateway***
  + Available VPCs --> ***my-vpc***

Once the ***Internet Gateway*** is created, we have to move to ***NAT Gateway***

**Create NAT gateway:**

* Lets create the ***NAT Gateway***
  + VPC --> NAT gateways --> Create NAT gateway
  + Name it as ***my-nat-gateway-1***
  + Select the public subnet 1 (***my-public-web-subnet-1***)
* For this demo, we are going to create the NAT gateway only in one subnet, but as a best practice, you should create multiple NAT Gateways to ensure redundancy.
  + Also allocate an elastic ip to NAT Gateway by clicking on ***Allocate Elastic IP***
  + Click on ***Create NAT gateway***
  + This creation will take some time,

Once the ***Internet Gateway & NAT gateway***  are created, we have to make changes in the ***Route tables*** so that the connectivity to the internet will be established from your ***Internet Gateway*** to your ***Public Subnet*** and ***NAT Gateway*** for ***Private Subnet.***

**Add routes for IGW, NAT:**

* We will add a route in the Public Route Table (***my-public-web-route-table***) for ***0.0.0.0/0*** and we will point it to Internet Gateway (**my-internet-gateway**).
  + VPC --> Route tables --> ***my-public-web-route-table*** --> Routes --> Edit Routes --> Add route
  + Destination --> 0.0.0.0/0 Target ---> ***my-internet-gateway***
  + Click on ***Save Changes***
* Similarly we will make the changes to app route table (***my-private-app-route-table***)
  + VPC --> Route tables --> ***my-private-app-route-table*** --> Routes --> Edit Routes --> Add route
  + Destination --> 0.0.0.0/0 Target ---> ***my-nat-gateway-1***
  + Click on ***Save Changes***
* And finally, we will do for DB route table as well.
  + VPC --> Route tables --> ***my-private-db-route-table*** --> Routes --> Edit Routes --> Add route
  + Destination --> 0.0.0.0/0 Target ---> ***my-nat-gateway-1***
  + Click on ***Save Changes***

This completes our networking configurations. We have created VPC, Subnets, Route tables, Internet Gateway, NAT gateway. We have associated them.

Now let’s start deploying the application components.

**Create a jump server:**

* EC2 --> Instances (running) --> launch Instances
* Select Amazon Linux AMI
* Select t2.micro
* Select the vpc that we created (***my-vpc***)
* Select the public subnet ***my-public-web-subnet-1*** because this would be our Jump server.
* Select ***Enable***  for Auto-assign Public IP.
* Tags --> ***Name:my-jump-server***
* Create a new security group ***my-jump-server-sg***
* Create a new key pair ***mykeypair.pem***
* Click on ***Launch Instances.***

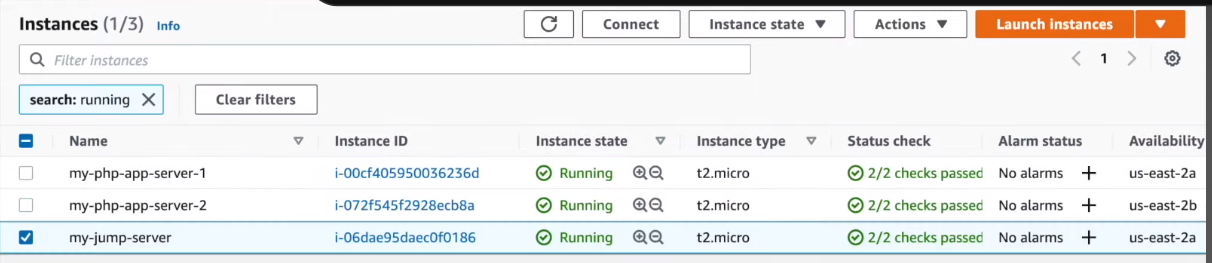
Please ensure that you have downloaded the ***.pem*** key. Now let’s create the PHP instances.

**Create PHP servers**

* EC2 --> Instances (running) --> launch Instances
* Select Amazon Linux AMI
* Select t2.micro
* Select the vpc that we created (***my-vpc***)
* Select the app subnet ***my-private-app-subnet-1***
* Tags --> ***Name:my-php-app-server-1***
* Create a new security group for app layer ***my-php-app-server-sg***
* We will allow SSH connection to this app server only from the jump server. Source --> Custom --> ***my-jump-server***
* Choose an existing key pair ***mykeypair.pem***
* Click on ***Launch Instances.***

Create one more PHP server with same configuration.

* EC2 --> Instances (running) --> launch Instances
* Select Amazon Linux AMI
* Select t2.micro
* Select the vpc that we created (***my-vpc***)
* Select the other app subnet ***my-private-app-subnet-2***
* Tags --> ***Name:my-php-app-server-2***
* Select an existing security group for app layer ***my-php-app-server-sg***
* We will allow SSH connection to this app server only from the jump server. Source --> Custom --> ***my-jump-server***
* Choose an existing key pair ***mykeypair.pem***
* Click on ***Launch Instances.***



Once all the servers are in running state, try to connect to the jump server from you local laptop.

You can get the connectivity details from Connect option in EC2 instance.

* We have already downloaded the key-pair (***mykeypair.pem***), we will give appropriate permissions to that key pair ***mykeypair.pem***
  + chmod 400 ***mykeypair.pem***
* Now, we will connect to the server using the ***mykeypair.pem***
  + ssh -i “***mykeypair.pem***” ec2-user@server-public-ip
* We have logged into the Jump server.
* Now lets try to copy our SSH key, give appropriate permission to ssh key
  + chmod 400 mykeypair.pem
* Now we have to connect to PHP app server
  + Click on my-php-app-server-1 --> Connect --> SSH Client --> Example connectivity ***ssh -i “mykeypair,pem” ec2-user@public ip of server***
* Paste that in the jump server.
* Now we should be logged into our php server, and now we have to install PHP & Apache on that app server

**Installation of PHP, Apache**

[Follow the documentation from AWS Install LAMP on Amazon Linux 2023](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-lamp-amazon-linux-2023.html)

$ sudo dnf update -y

$ sudo dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel

We do not require MariaDB because we are going to use RDS instance

$ sudo systemctl start httpd

$ sudo systemctl enable httpd

$ sudo systemctl is-enabled httpd

Check if the httpd service is enabled and it is working

$ curl <http://localhost>

Give permissions to ec2-user

$ sudo usermod -a -G apache ec2-user

Log out and then log back in again to pick up the new group, and then verify your membership.

Log out (use the exit command or close the terminal window): $ **exit**

To verify your membership in the apache group, reconnect to your instance, and then run the following command: $ **groups**

$ sudo chown -R ec2-user:apache /var/www

$ sudo chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;

$ find /var/www -type f -exec sudo chmod 0664 {} \;

**Install phpMyAdmin (Sample app)**

$ sudo dnf install php-mbstring php-xml -y

$ sudo systemctl restart httpd

$ sudo systemctl restart php-fpm

$ cd /var/www/html

$ wget <https://www.phpmyadmin.net/downloads/phpMyAdmin-latest-all-languages.tar.gz>

$ mkdir phpMyAdmin && tar -xvzf phpMyAdmin-latest-all-languages.tar.gz -C phpMyAdmin --strip-components 1

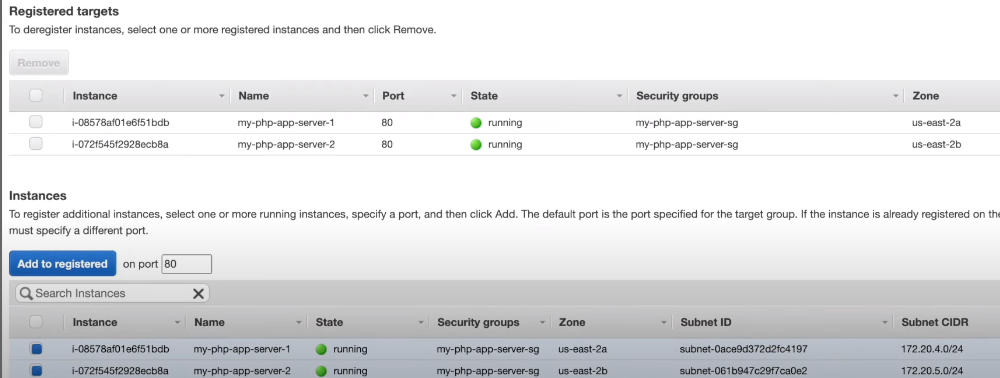
$ rm phpMyAdmin-latest-all-languages.tar.gz

For now, we are done with the configuration of phpMyAdmin. At a later stage when RDS is created, we will make certain changes to the config file.

Once the App instances are created, we will move towards creation of Load Balancer

**Create and configure an application load Balancer**

* EC2 --> Load Balancing --> Load Balancers --> Application Load Balancer
* Name it as **my-alb**
* Scheme should be ***Internet-facing*** because we want to expose the Load Balancer to the outside world.
* Availability Zones ---> VPC --> ***my-vpc***
* Select the availability zones as public subnets, **my-public-web-subnet-1** & **my-public-web-subnet-2**
* Under Security Group settings --> Create a new security group --> name it as ***my-alb-sg*** and source 0.0.0.0/0 would be fine.
* Create a new target group --> name as ***my-alb-tg*** Target *type : Instances* Port : *80*
* Register the targets, we have already created the PHP servers, select those 2 servers and click on ***Add to registered***  button.



* Click on ***Create***
* Once the Load Balancer is successfully created, check the ***Target Groups*** and check whether the app ec2 instances are in healthy state.
* Allow the Load Balancer security group in App servers security group. This will establish connection between your ALB and the instances.
* Go back to load balancer section, and copy the DNS name and paste the DNS name in the browser and see if the test page is displayed.

Now we have seen the Apache test page is getting displayed, we have to check whether the load balancing is happening as expected or not.

* + We have to create an index.html file in the directory /var/www/html folder in both the app servers to validate whether the requests are going to both the app servers.
  + We will put a sample text in the ***index.html*** page, as PHP server 1 and PHP server 2.
  + Once the files are saved in two servers, we will navigate to index.html the browser to check whether the requests are flowing to both the app servers.

So now, we have created the two layers of the 3-Tier architecture the web layer and app layer, lets move to the third layer that is DB layer.

**Create RDS instance**

* Navigate to RDS
* Click on the ***Subnet groups --> Create DB Subnet Group.***
* As we created separate subnets for DB instances, give the name as ***my-db-subnet-group.***
* Choose the VPC ***my-vpc***
* Choose the availability zones 2a & 2b
* Select the subnets that we have created for DB
* Click on ***Create***

Once our ***Subnet groups*** are created, we will move to creation of actual DB instance.

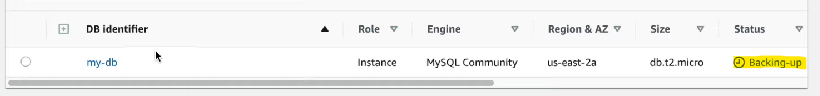
* RDS --> Dashboard --> Create database
* Select Database creation method as ***Standard create***
* Select ***MySQL***  under Engine options.
* Under Templates, click ***Free tier***
* Under Settings --> DB instance identifier, give the name as ***my-db***
* Under Credentials Settings, keep the Master username as ***admin*** as it is.
* Put the password, confirm the password
* Keep the DB Instance class as ***db.t2.micro***
* Disable the Storage Auto-scaling
* In Connectivity, select the VPC ***my-vpc*** and the db Subnet group is selected automatically.
* Under VPC security group, create a new Security Group with name as ***my-db-sg***
* Click on ***Create database.***

Once our database is created, we will need to establish the connectivity between your app server and your db server.

**Configure phpMyAdmin with RDS**

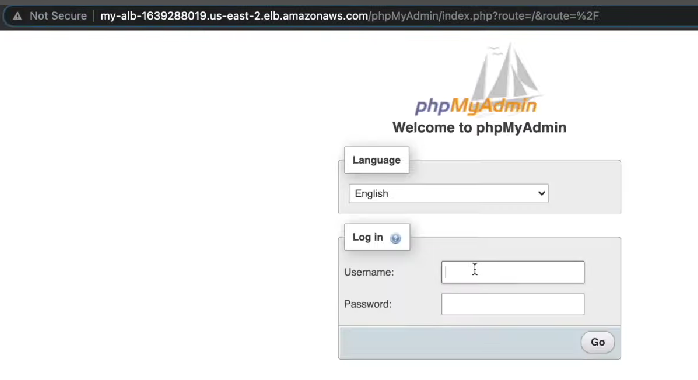
We have to go to security group of DB and we have to add entry of PHP app servers in inbound rules.

* Go to RDS --> Databases --> db --> Connectivity & security --> Security --> VPC security groups --> Edit inbound rules
* Type: Custom TCP Port range: 3306 Source: Custom : app server security group ***my-php-app-server-sg***
* Delete the default rule that is created.
* Click on ***Save rules***



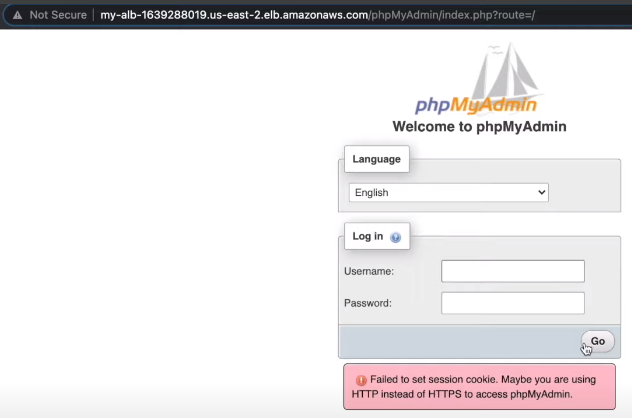
By the time it is in Backing up state, lets copy the endpoint from Connectivity & security --> Endpoint & port --> Endpoint

* Go to PHP server, navigate to ***phpMyAdmin***  folder under /var/www/html
* $ cd /var/www/html/phpMyAdmin
* $ mv config.sample.inc.php config.inc.php
* $ vi config.inc.php
* Search for the word **host** in the file and replace the **localhost**  with the copied hostname of RDS instance that we have created.
* Save the file.
* Go to the browser, and navigate to phpMyAdmin and check whether the application is running or not.



**Configure session stickiness**

* By the time once we punch in the credentials, you might see the error “Failed to set session cookie.”

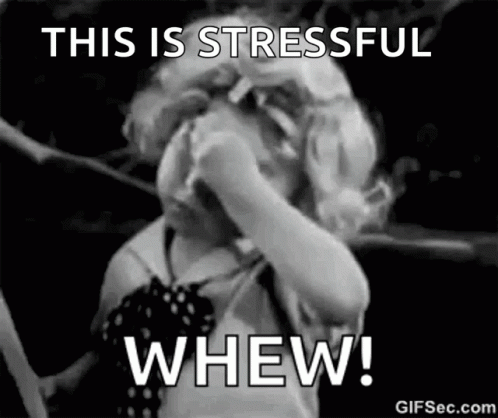


* This happens because phpMyAdmin is a stateful application and the session is stored on individual servers.
* Since the Load Balancer is configured as round robin, requests are going to both the servers

As a work-around, we can what we can do is, go to Target Groups --> Select the target group --> Attributes --> Edit --> Enable Stickiness option --> Load balancer generated cookie --> Save changes.

This way your request flows through only one server and in case of failure it will be redirected to another server.

Once the changes are made, open the browser in incognito and check phpMyAdmin. You should be able to login to the application.



**Services recap**

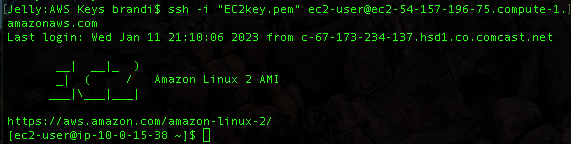
1. Created VPC
2. Corresponding Subnets
3. Route tables
4. Internet gateway
5. Attached Internet Gateway to VPC
6. Created NAT Gateway in public subnet.
7. Created Jump server and php app servers
8. Load balancer, Target Group
9. RDS instance.

You did it! You created a three tier architecture. That was a lot of work. But we’re not done. Now that the architecture is built, let’s test it.

**Testing Connectivity**

**SSH to Tier 1 Instance**

Let’s make sure we can connect to our web server instance via SSH:



If we navigate to our tier 1 public IPv4 address, we’re greeted with this:

Perfect! Tier 1 is working.

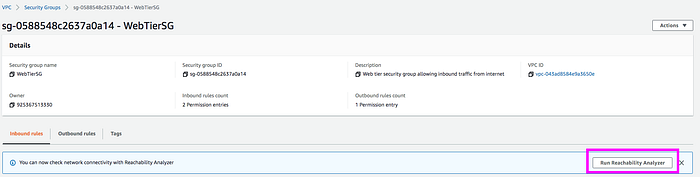
**Ping Tier 2 from Tier 1**

Now test the application layer from tier 1. You are already logged into the tier 1 instance, so try to ping the tier 2 private IPv4 address.

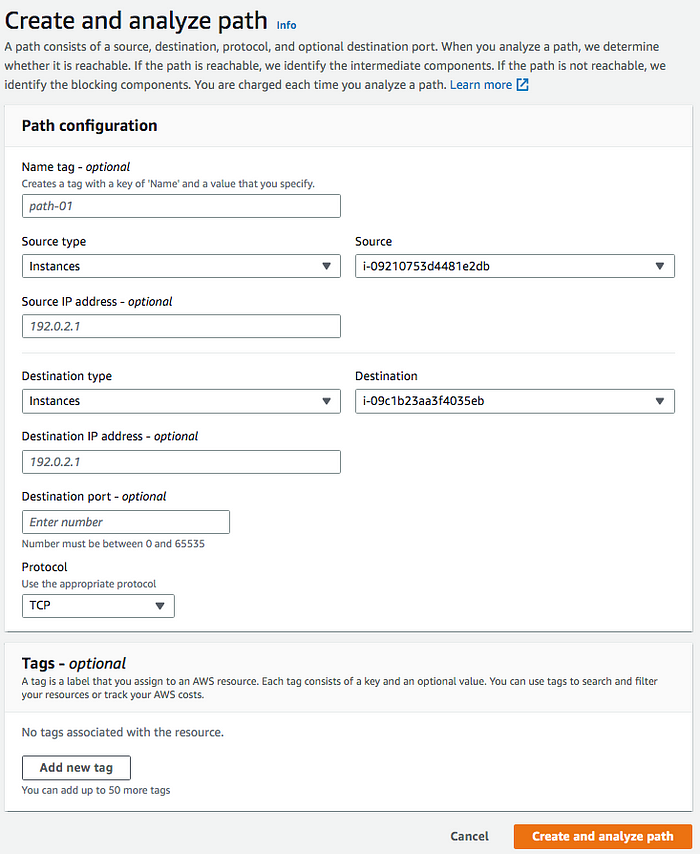
That works also, so we know our tier 1 and tier 2 are able to communicate.

**Analyzing Reachability**

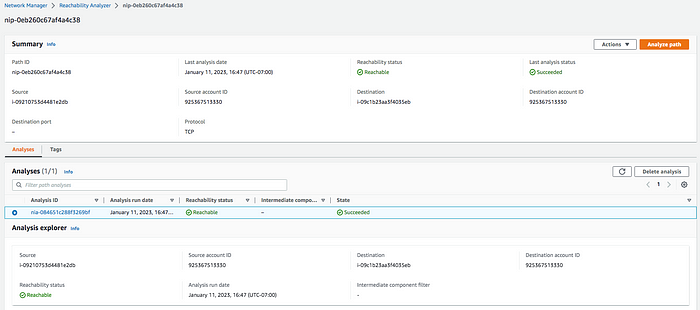
Navigate to the VPC console and click on Security groups. Open your web tier security group and click “Run Reachability Analyzer.”



In the analyzer, you create a path to determine if it is reachable. Our first path will test the reachability of the tier 1 instance to the tier 2 instance (similar to what we did in the CLI). Under Source type select “Instances” and under source select the web server tier 1 instance. Under Destination type select “Instances” and for Destination choose the application tier 2 instance. Then click “Create and analyze path.”



This may take several minutes to complete, but if it succeeds, you will get a green “Reachable” and “Succeeded” status.



**Analyze Tier 2 and Tier 3 Reachability**

In tier 3 we created a DB instance, which is an isolated database environment running in the cloud. Don’t confuse a DB instance with an EC2 instance. We can’t directly SSH to the database, but we can use an SSH forwarding agent to achieve this.

**Accessing Private Subnet From a Public Subnet**You need to add your access key pair file to your keychain. To do this, first make sure you are in your local host (use the command *exit* to get out of any EC2 instance you’re connected to). Then use the following command:

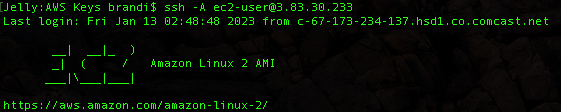
ssh-add -K <keypair.pem>



The output should be “Identity added.” Now that your key pair file is added to your keychain, SSH agent will scan through all of the keys associated with the keychain and find your matching key.

Now we can reconnect to our public web tier 1 EC2 instance via SSH, but this time use -A to specify you want to use the SSH agent.

ssh -A ec2-user@<public-ip-address>



Now that you are logged back into your public instance, check to make sure that SSH agent forwarded the private key to the instance. Use the following command to do this. Note that the output says error fetching identities, but then the private key is listed in the following line (I’ve grayed out my key for security purposes):

ssh-add -l



Ok so our key pair has been forwarded to our public instance, so let’s see if we can now connect to our private instance. Grab your application tier 2 private IPv4 address and input it into the following command:

ssh -A ec2-user@<private-ip-address>

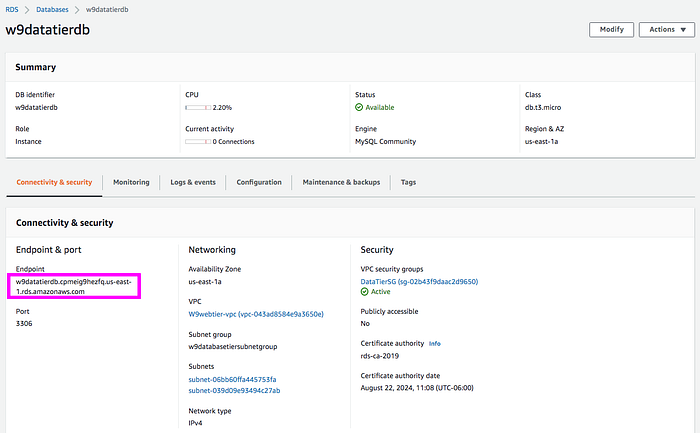


You have now SSH’d from your public tier 1 web instance into your private tier 2 application instance!

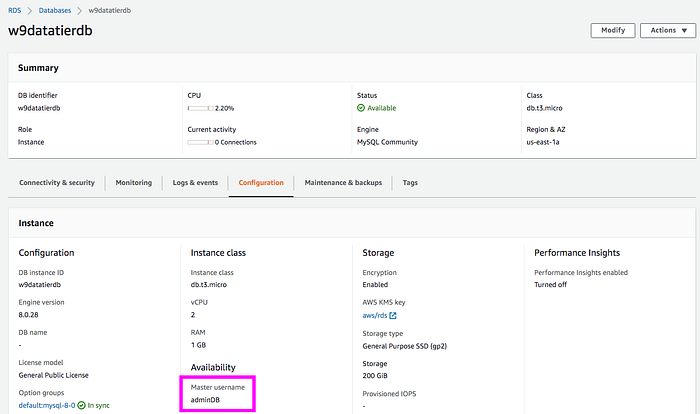
**Testing Connectivity to the Database Tier**There are a few ways you can connect to your RDS database from your application tier. One way is to install MySQL on your private tier 2 instance to access your database. Make sure you are still logged into your tier 2 application instance, and use the following command:

sudo yum install mysql -y

This actually installs the MariDB package, which is used to read MySQL. Once this is installed, you should be able to use the following command to log into your RDS MySQL database. Before typing the command, you will need your RDS database endpoint and the username and password you created when you created the RDS database. To find your RDS database endpoint, navigate to the database you created and find the endpoint under Connectivity & Security.

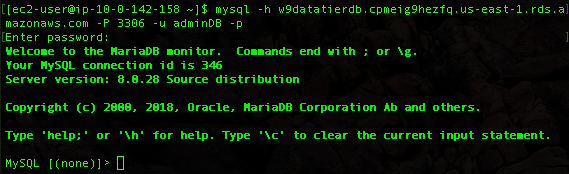


If you forgot your username, you can find it by going to the Configuration tab in the RDS database.



Good luck if you forgot your password. Seriously, I don’t know how you can recover it. Ok now we are ready to connect to our database from the CLI. Use the following command:

mysql -h <rds-database-endpoint> -P 3306 -u <username> -p



You are now connected to your MySQL database from your application tier. All of your tiers have connectivity!

This project was a labor of love and not without its challenges, particularly testing the connectivity. I hope it helps someone, and if you have questions about a particular step, please comment below and I will try to help out.

***\*REMEMBER\* When done with this session, make sure to delete your resources that cost money, including your NAT gateway, elastic IP, and ALBs. Stop your instances to prevent them from running up unnecessary costs.***