**Project 1 - Deploying a Multi-Tier Website using EC2**

**Description**:

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

**Problem Statement:**

**Company ABC wants to move their product to AWS. They have the following things set up right now:**

**1. MySQL DB**

**2. Website (PHP) The company wants high availability on this product, therefore wants Auto Scaling to be enabled on this website.**

Hints:

*a. Database name: intel*

*b. Table name: data*

*c. Database password: intel123*

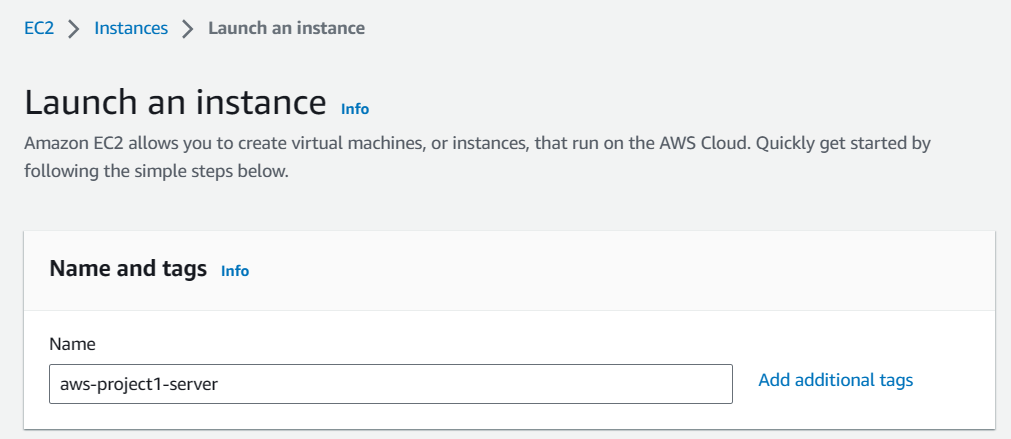
*d. Change hostname in website*

*e. Allow traffic from EC2 to RDS instance*

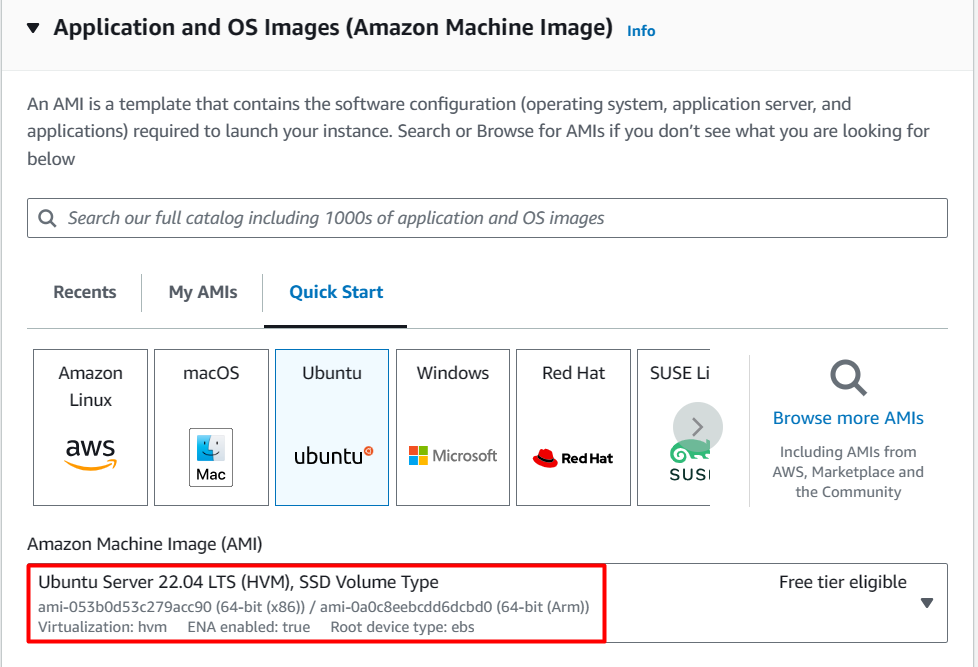
*f. Allow all-traffic to EC2 instance*

**Solution:**

To start with, lets create an **EC2 Instance** and configure an **Apache** web server on top of the instance.

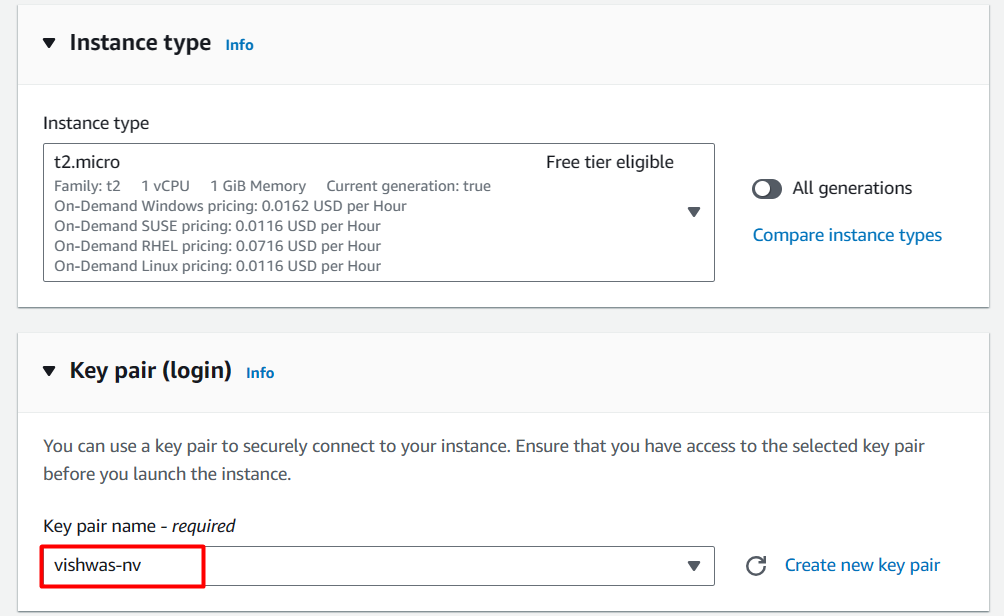
****

Make sure to select the **Ubuntu 22.04 AMI**. As we are going to use dependencies which are supported for this AMI only.

****

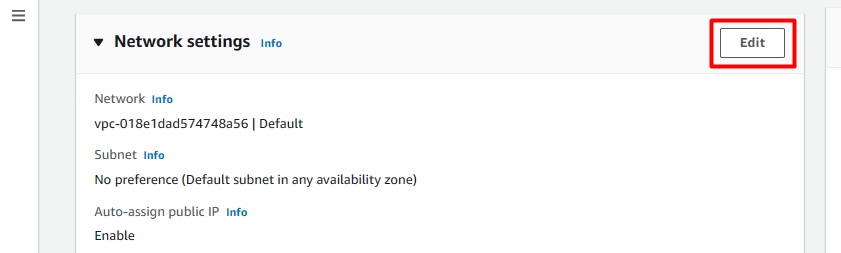
Choose **t2.micro** as **Instance Type**.

Select one of the existing key pairs or create a new one based on your preference.

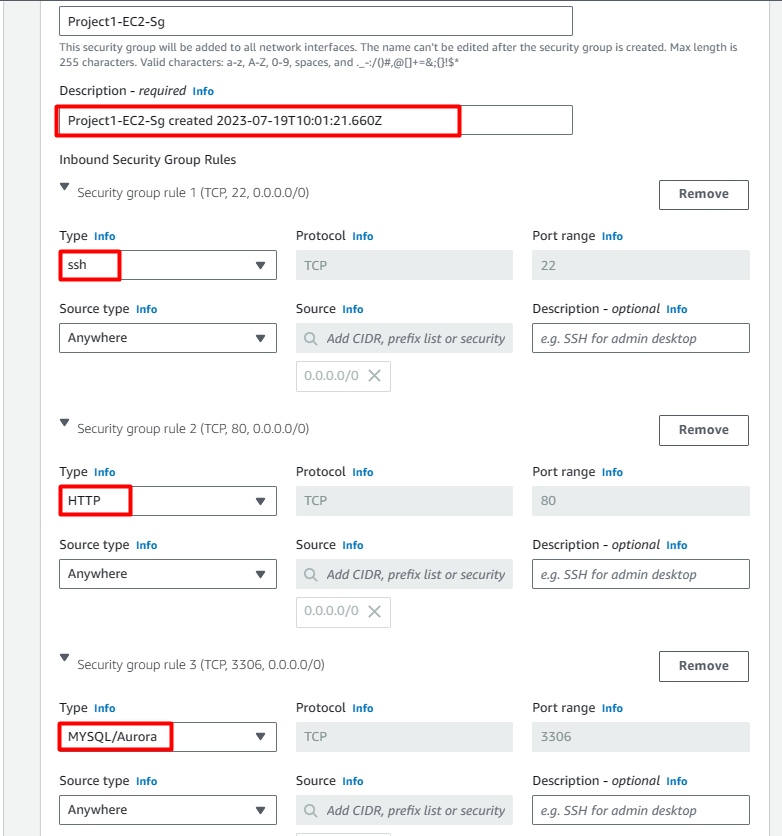
****

Now, In **Network Settings**, Click on **Edit**. Here we will configure rules for our EC2 Server.

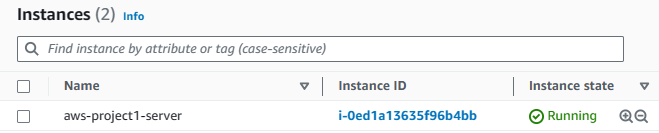
Rules will allow **SSH** & **HTTP** Inbound Traffic from Source **0.0.0.0/0**.

****

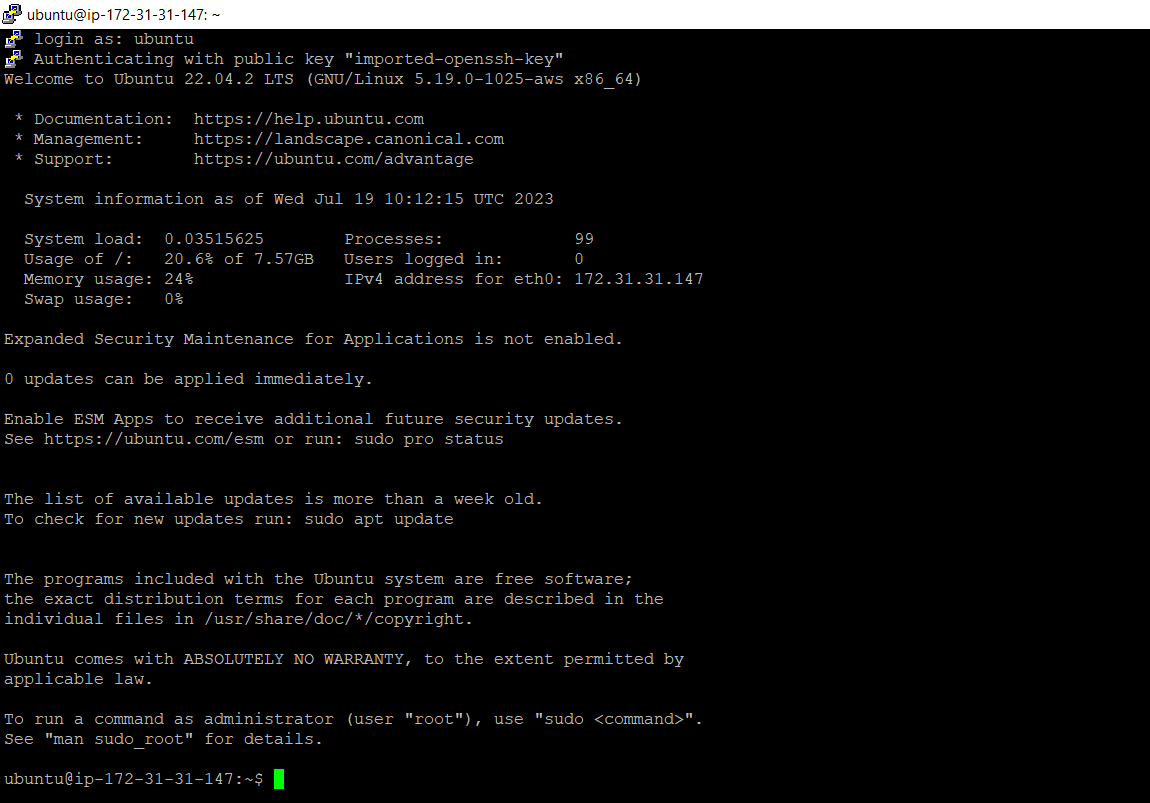
Provide a name to the security group such as **Project1-EC2-Sg** in this case.

****

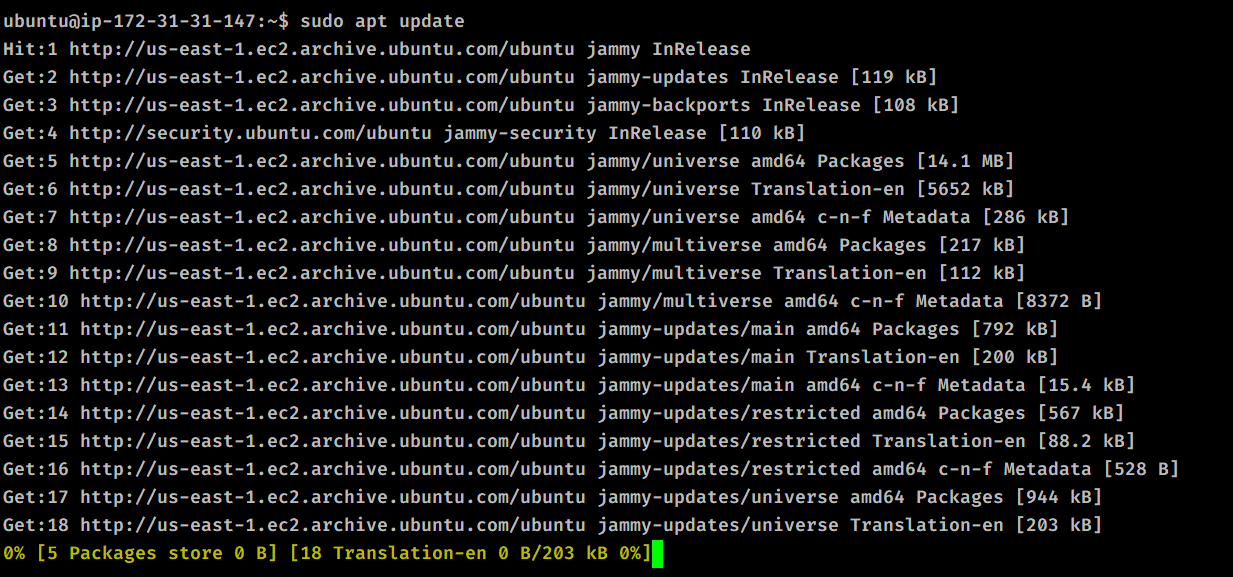
Select the default **8GiB gp2** root volume and launch the instance.

****

Now, let's connect to the EC2 instance.

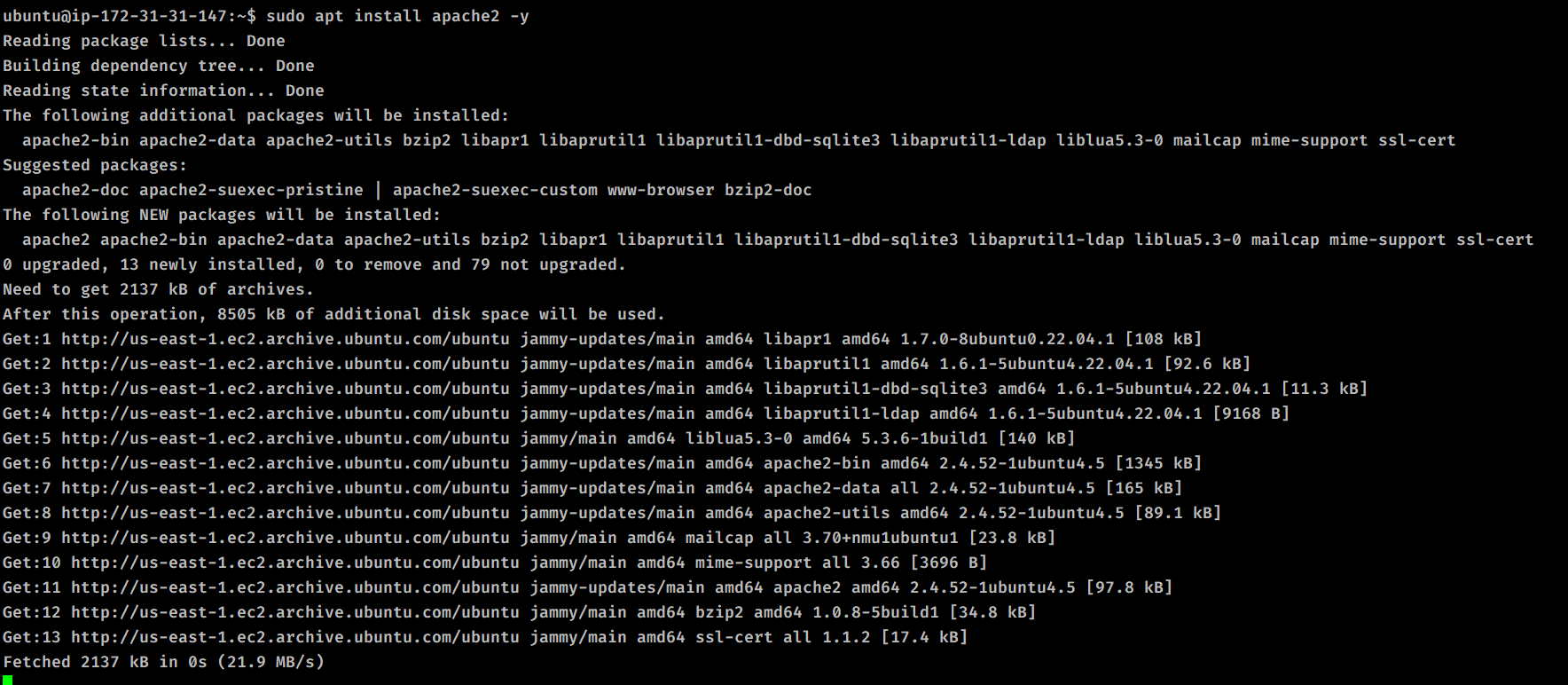
****

To configure Apache, firstly lets update the machine and follow it up by installing Apache web server.

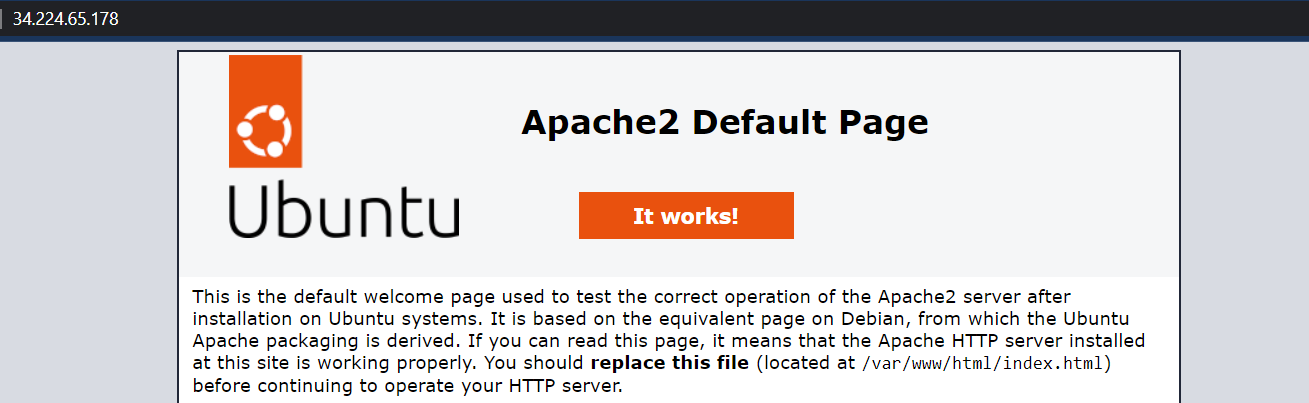
****

Now, To install Apache use the following command:

*sudo apt install apache2 -y*

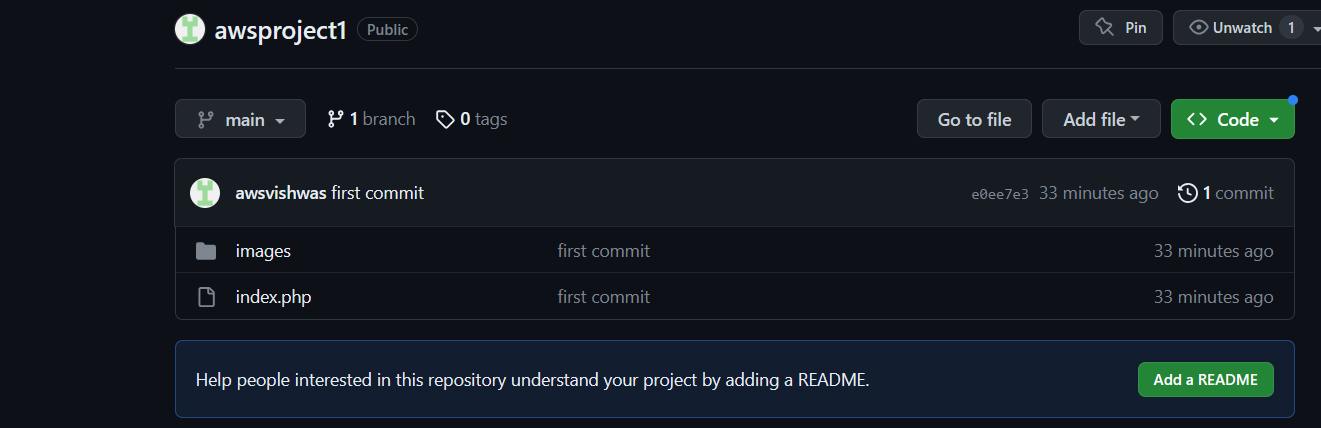
****

Copy and Paste the Public IP on the browser and you should be able to see the default Apache page.

****

Now, we will replace the default Apache web page by content of our website. You can refer to the Github url below for website code.

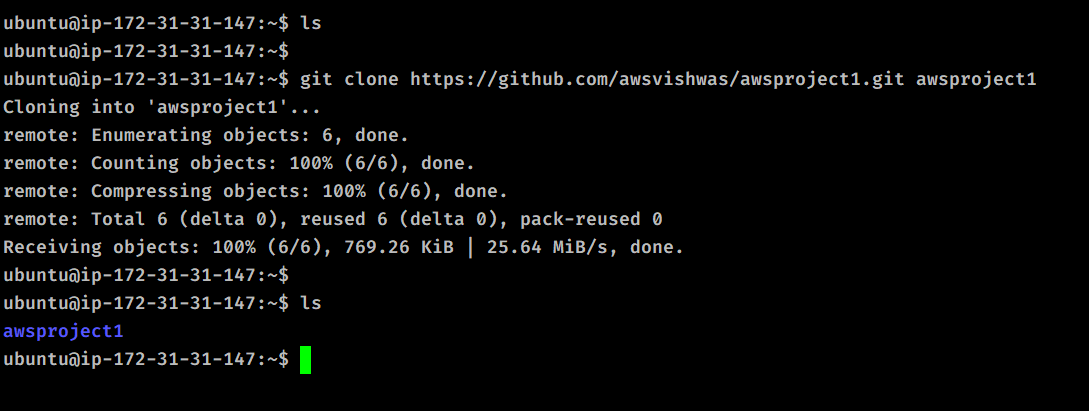
<https://github.com/awsvishwas/awsproject1>

****

Clone the Github repo using the command below:

*git clone https://github.com/awsvishwas/awsproject1.git awsproject1*

Now, you should be able to see the content in the local directory **awsproject1.**

****

Now, copy the content inside **awsproject1** directory to **/var/www/html** directory where our Apache is configured.

Use the command below:

*sudo cp -R awsproject1/\* var/www/html*

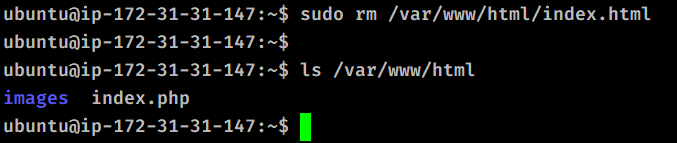
Now, when you list the items inside the **/var/www/html** directory, you can notice **index.html** and **index.php** web pages. As i**ndex.html** is the default Apache page, we don’t need it any longer as our main website is contained inside **index.php**.

To delete the **index.html** page, use the command below:

*sudo rm /var/www/html/index.html*

****

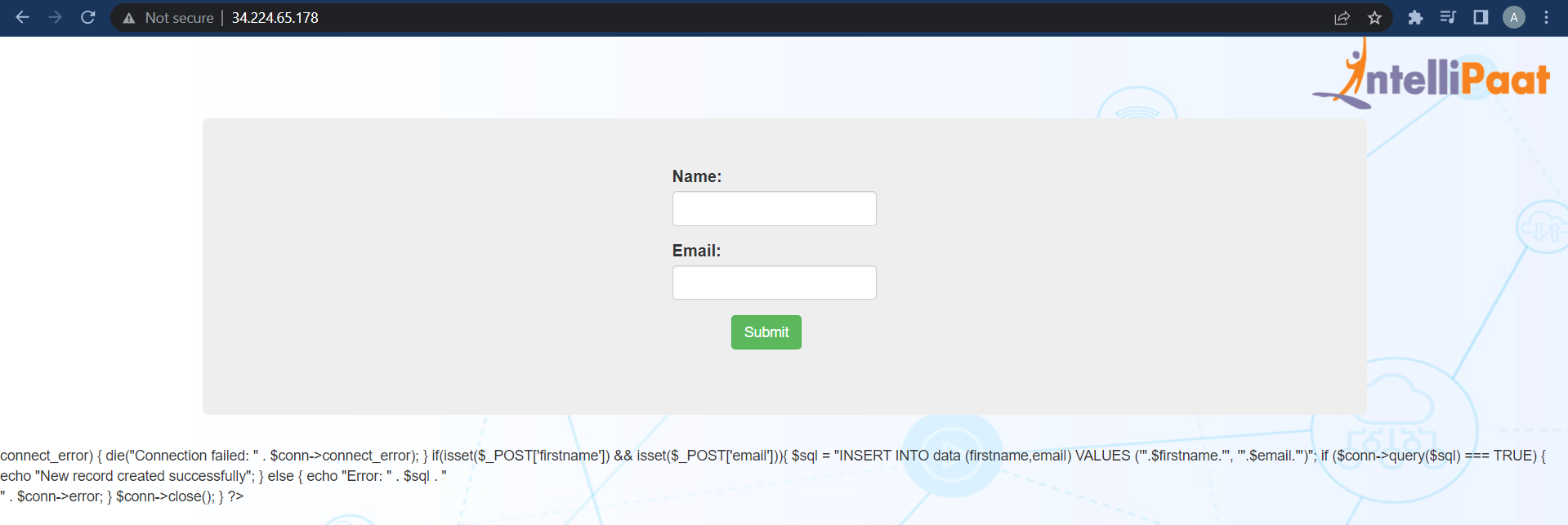
Index.html file is removed and now if you refresh the webpage in the browser, you should be able to see the Php application.

****

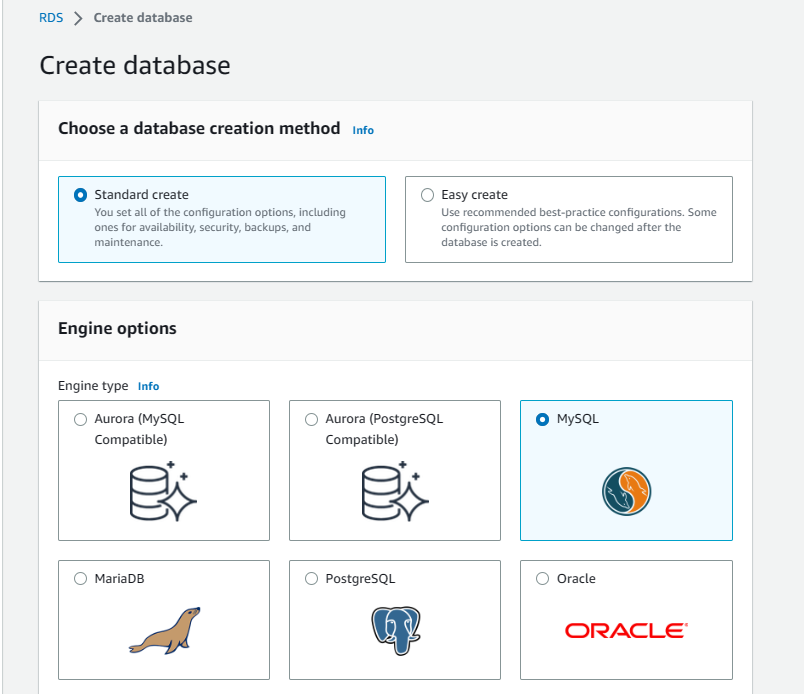
As you can see, our Php application is up and running.

In the console output, it gives **Connection failed** issue as we haven’t set up a database for this website yet.

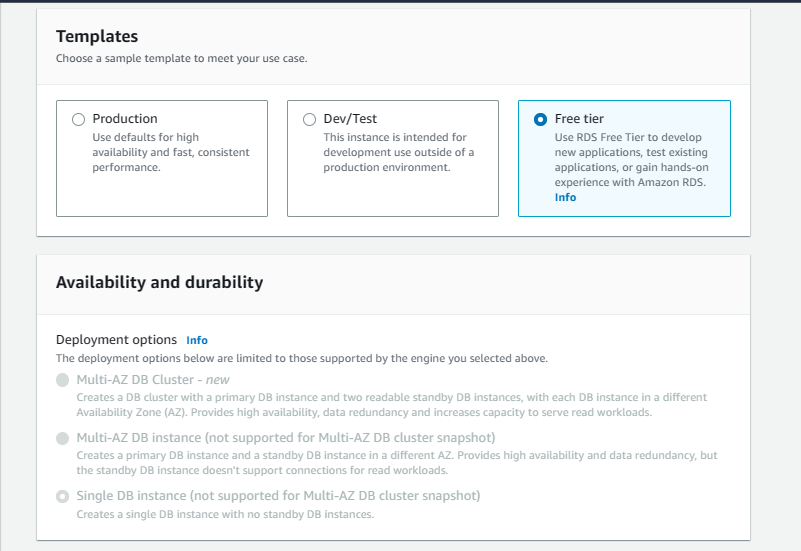
In the upcoming steps, we will configure an RDS Instance to fulfill the requirements.

****

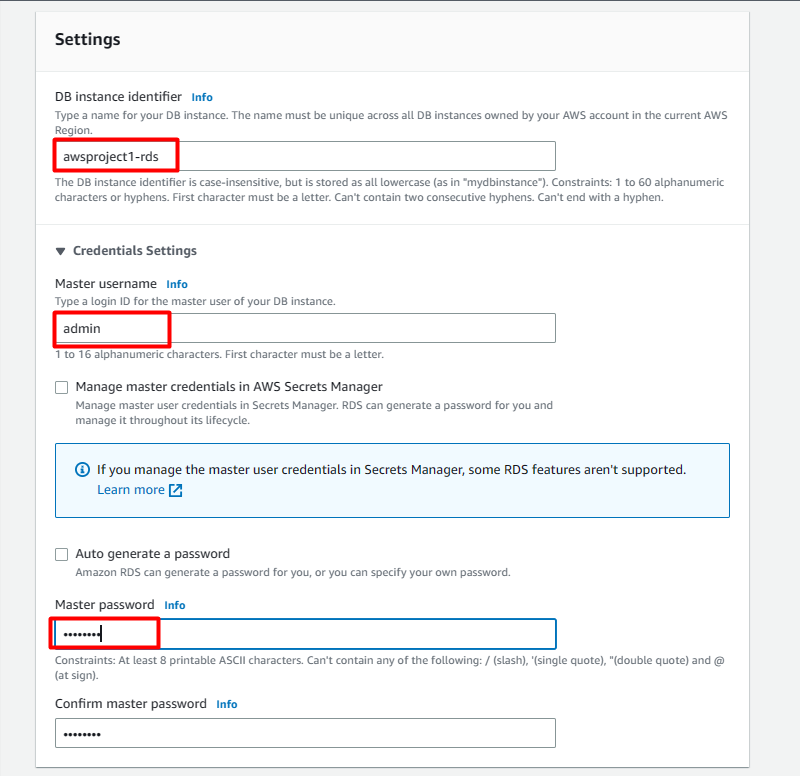
Lets create a **RDS MySQL** database now. Choose the options as shown below.

****

In **Templates**, Choose **Free Tier**.

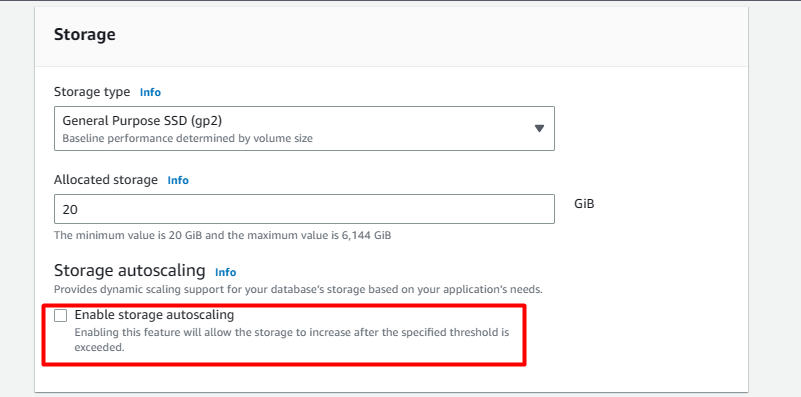
****

Now, provide the credentials as shown below and make sure to keep note of **master** **username** and **password**.

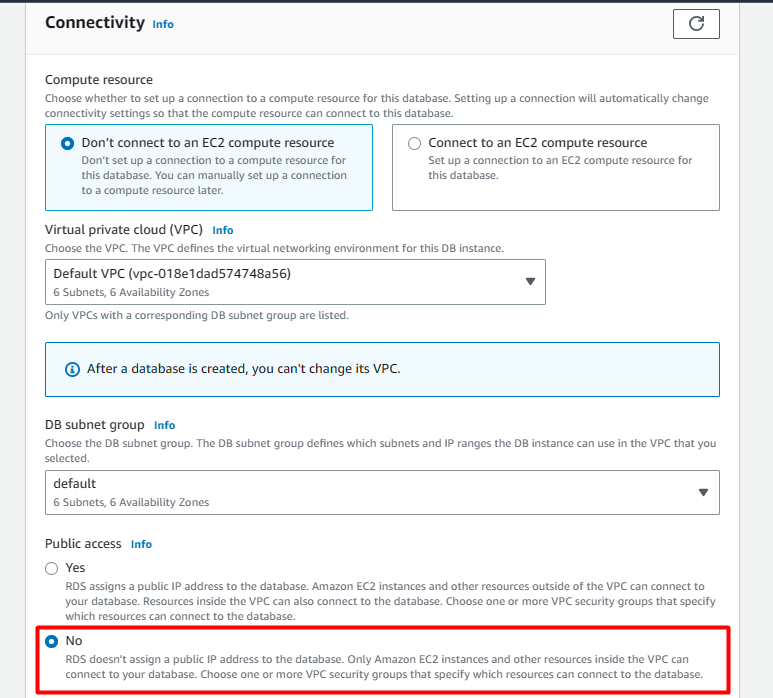
****

Choose the **minimum storage allocation**.

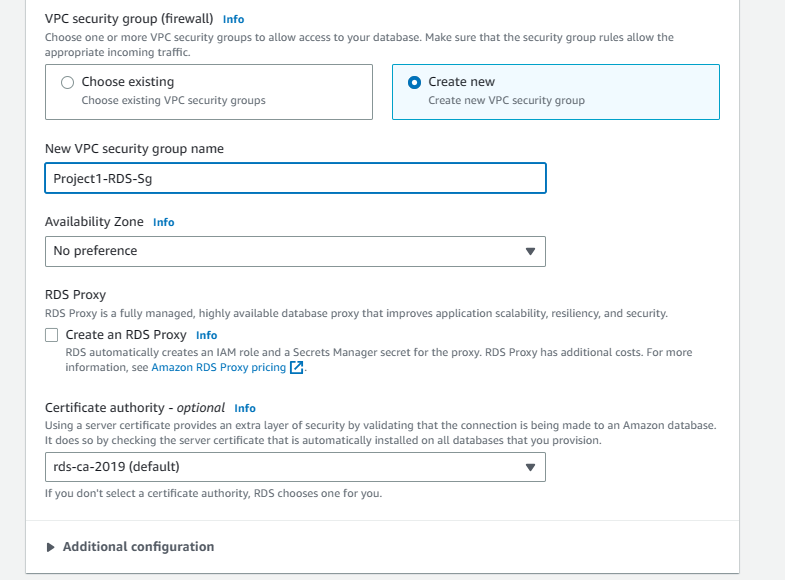
In **Storage Autoscaling,** You can either go with **enabled** or **disabled** based on your preference.

****

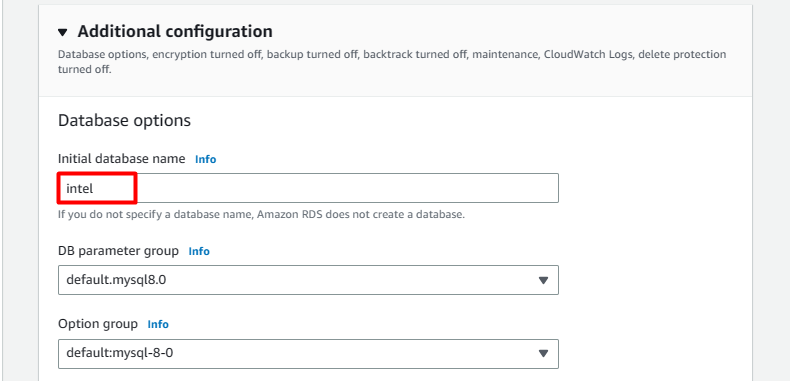
In **Connectivity**, Choose the options as shown below. Make sure you choose **Public Access** as **No** as the database is meant to be private.

****

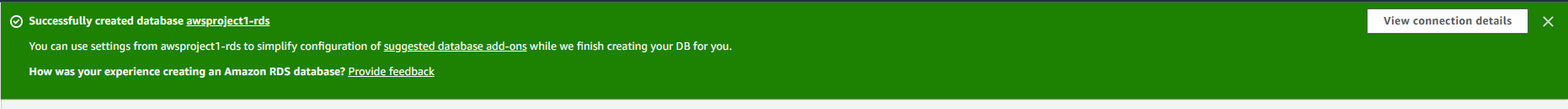
Create a **new security group** and provide a name, it will create a rule allowing necessary inbound traffic to this database through port **3306 ( MySQL/Aurora)**.

****

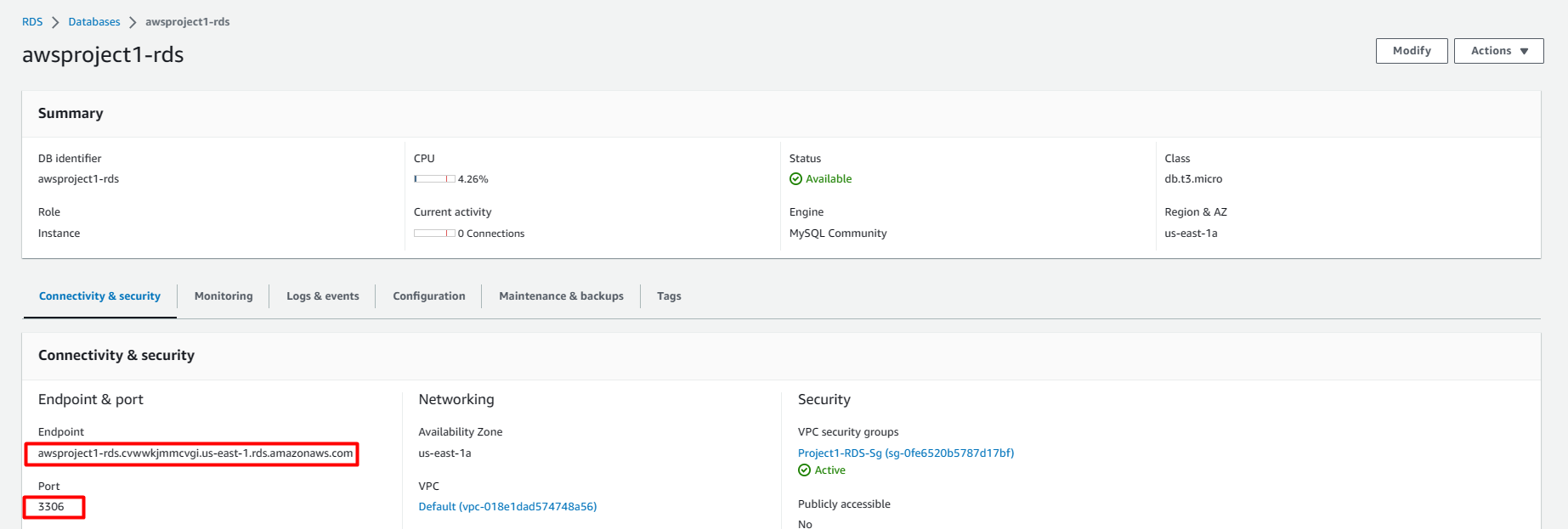
In **Additional Configuration**, provide the **initial** **database name** as per the problem statement.

****

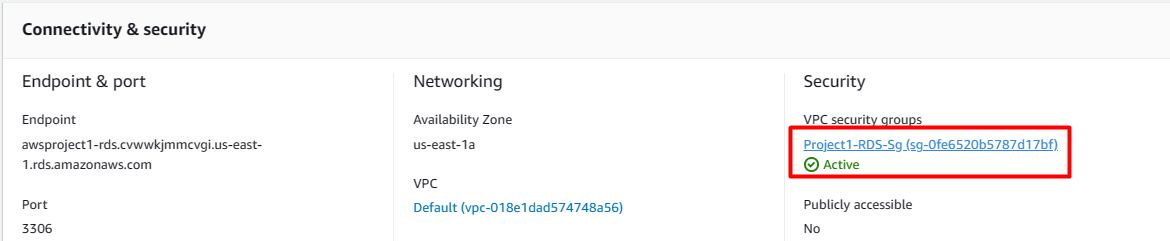
Wait for a few minutes until RDS Instance is created.

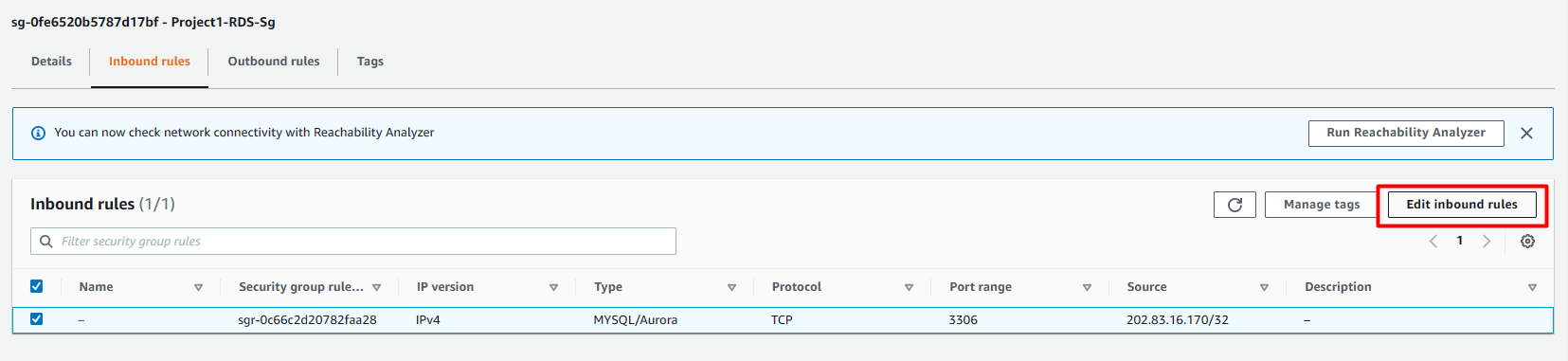
****

Note down the **RDS Endpoint** and **port**, as we will need it later on while connecting our EC2 server with this database.

****

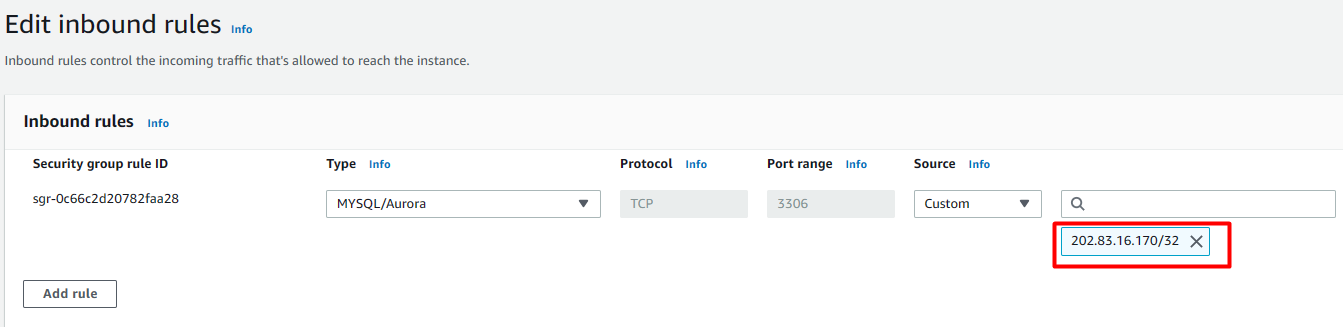
Now, Click on **Project1-RDS-Sg**, as we will need to modify its inbound rules to allow necessary traffic.

****

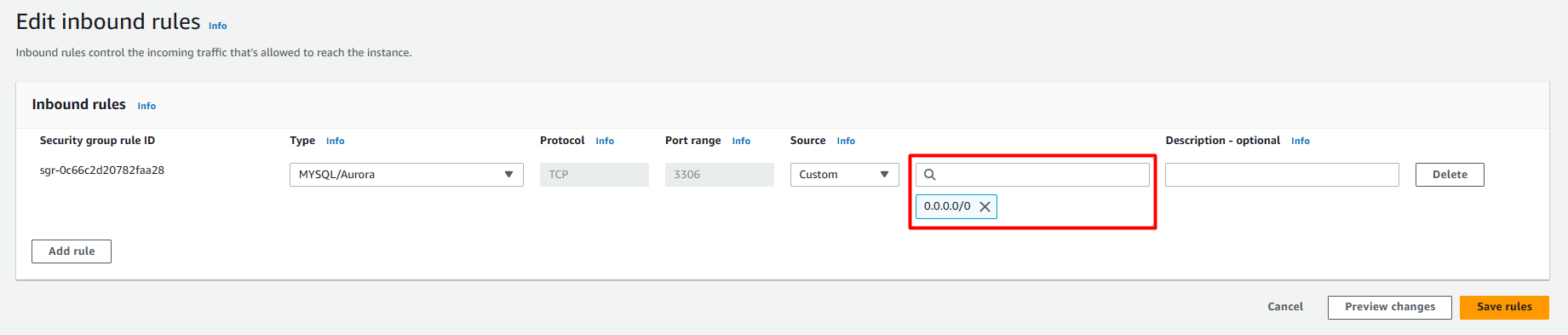
****

Here, you can see the connections are allowed from a **certain IP**. This can become a bottleneck later on if we have multiple servers with the same application that share the same database instance.

We will change this rule to allow **all possible IP addresses**.

****

Allow the traffic from source **0.0.0.0/0**.

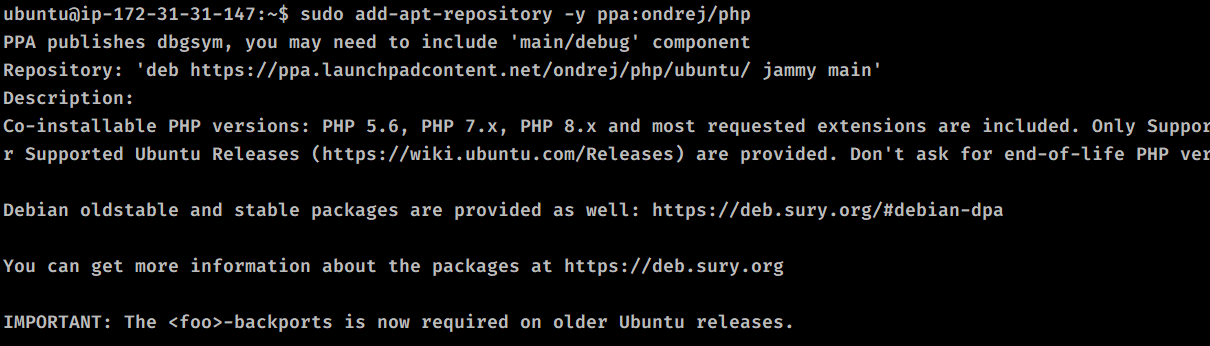
****

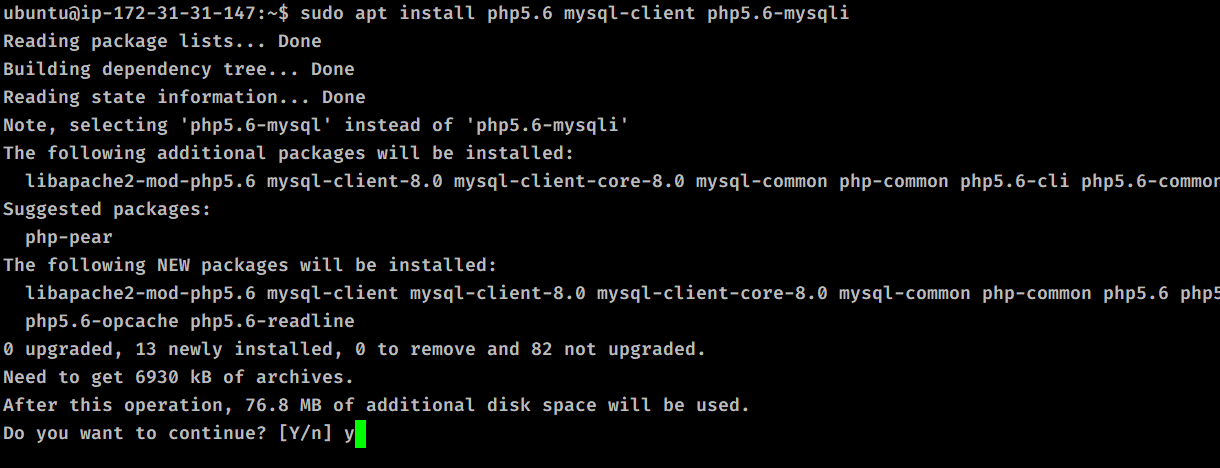
Now, let's head back to the EC2-server and install **Mysql** Packages that are compatible with our PHP application.

Import the repository and install the packages using the commands below:

*sudo add-apt-repository -y ppa:ondrej/php*

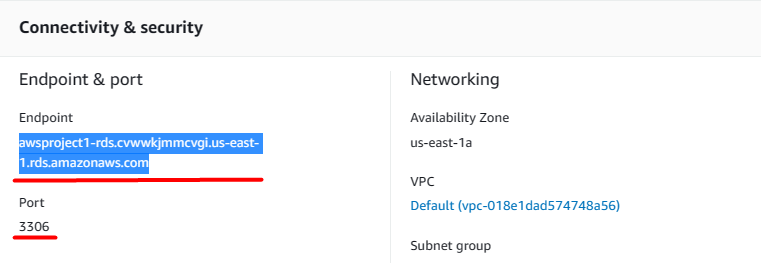
*sudo apt install php5.6 mysql-client php5.6-mysqli*

****

****

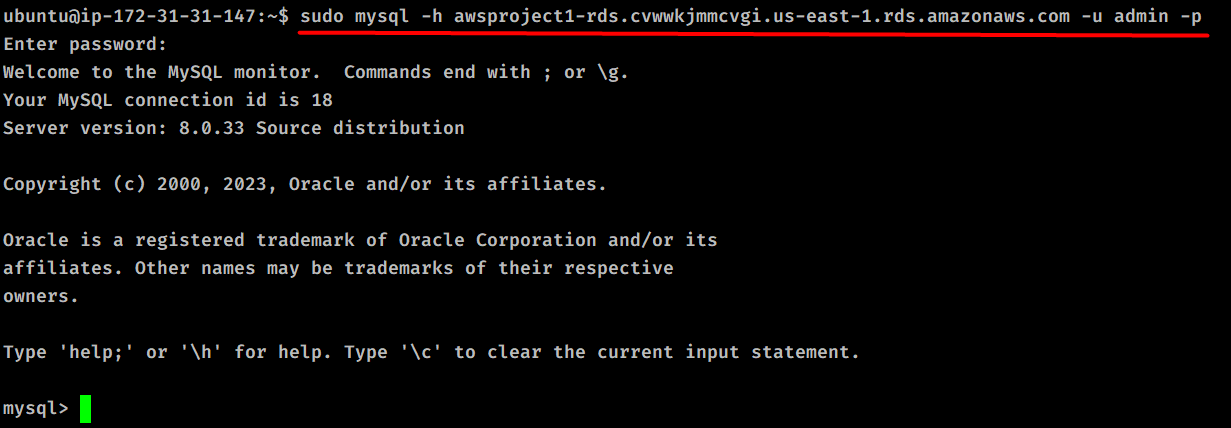
Now, we will connect with our RDS Instance and create a table in the database to store the user data coming through the application.

Copy the **RDS endpoint** and head back to the EC2 server.

****

Use the command as shown below.

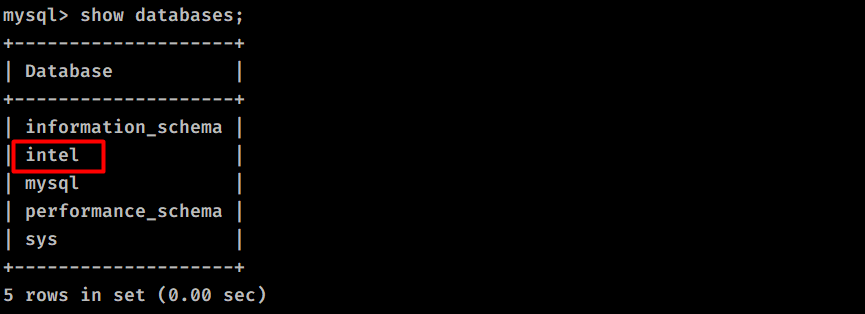
Replace **RDS endpoint**, **username** credentials according to your RDS Instance.

****

After connecting to the database, use the command below to list available databases:

*show databases;*

You can see the initial database available here:

****

To change the database use the command below:

*use intel;*

To create table use the command below:

*create table data(firstname varchar(21),email varchar(21));*

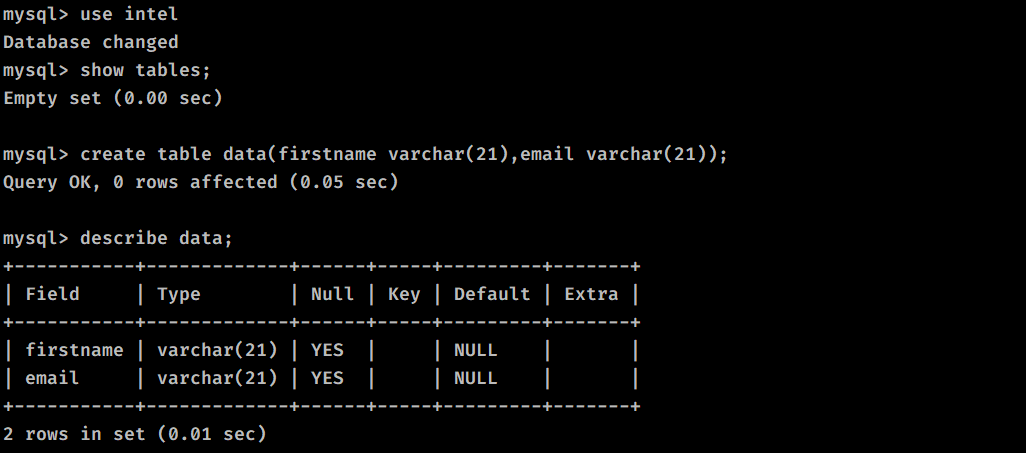
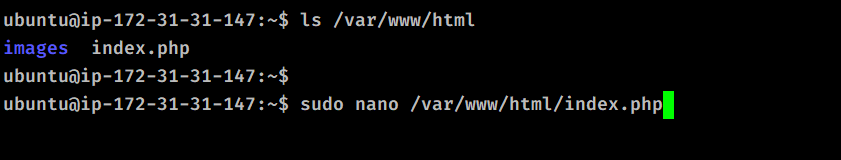
****

Table is created. Use **exit** to quit the Mysql shell.

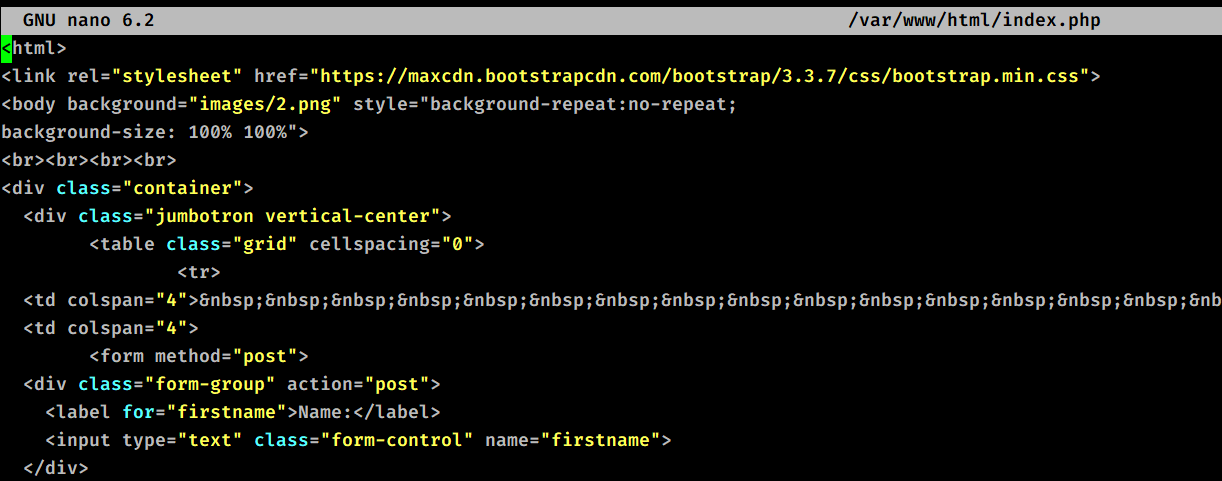
****

Now, it's time we integrate this database with our website.

Open the **index.php** file in the nano editor.

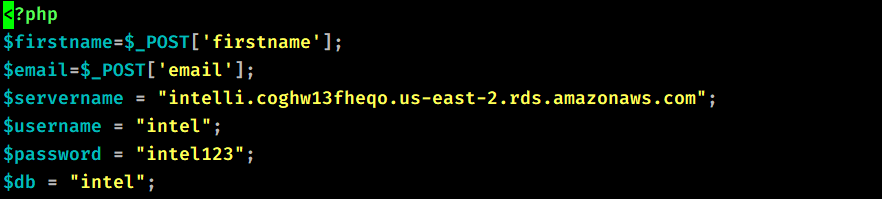
****

This is how our code looks.

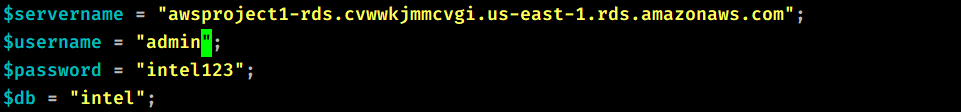
****

Scroll down to this line, where we need to update the database credentials.

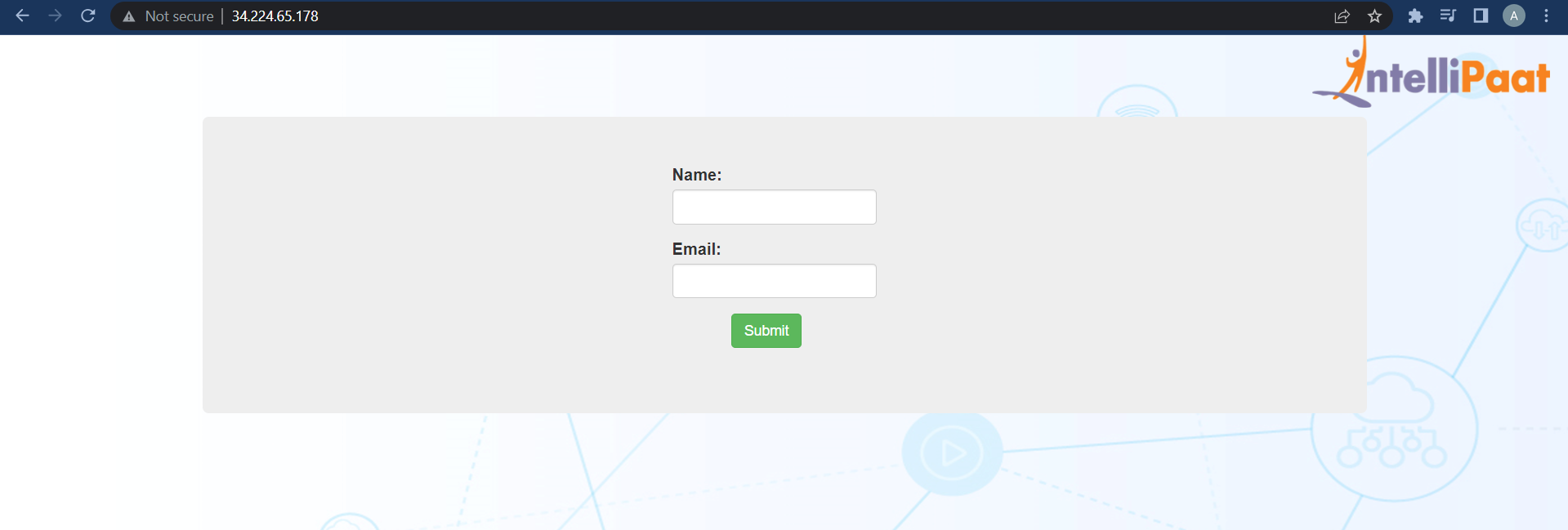
Update **servername**, **username**, **password** and **database name**.

****

Now, credentials are updated and it is now integrated with our RDS database.

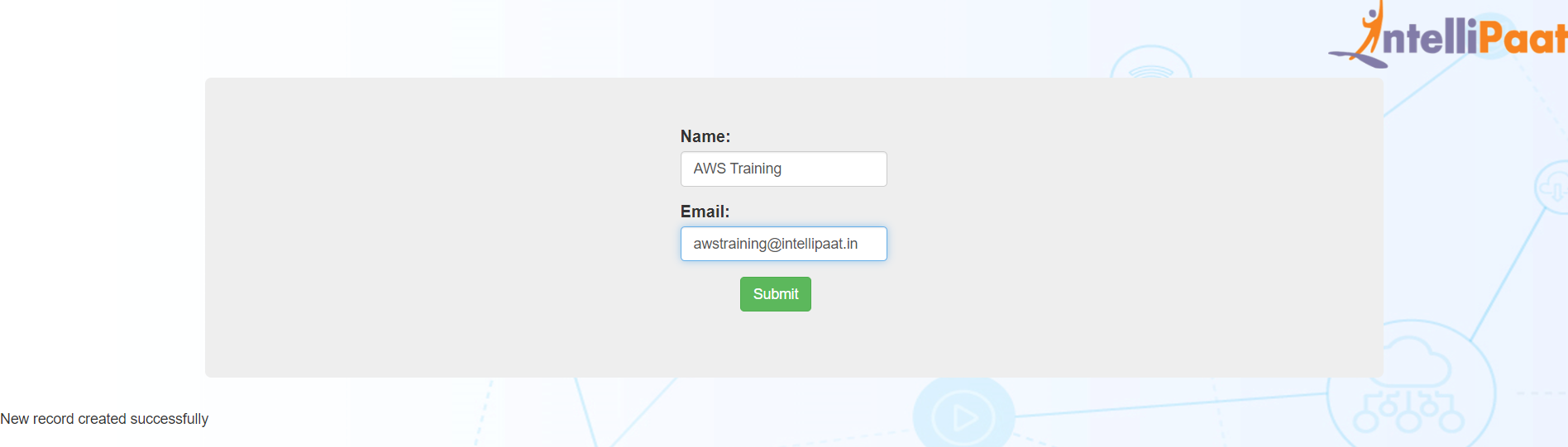
****

Now, refresh the page and you should be able to see that the console output which was showing an error is now rectified and now we are ready to use this application.

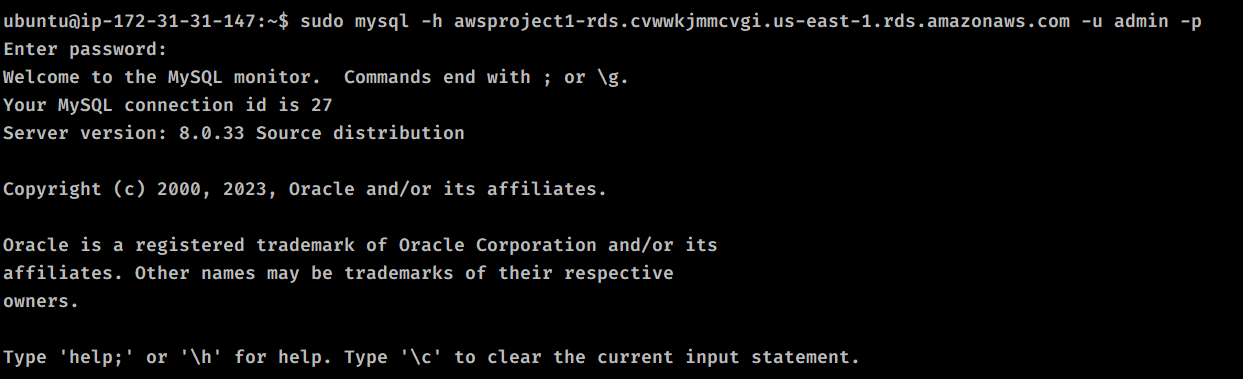
****

Provide the **Name** and **Email** and click on **Submit**.

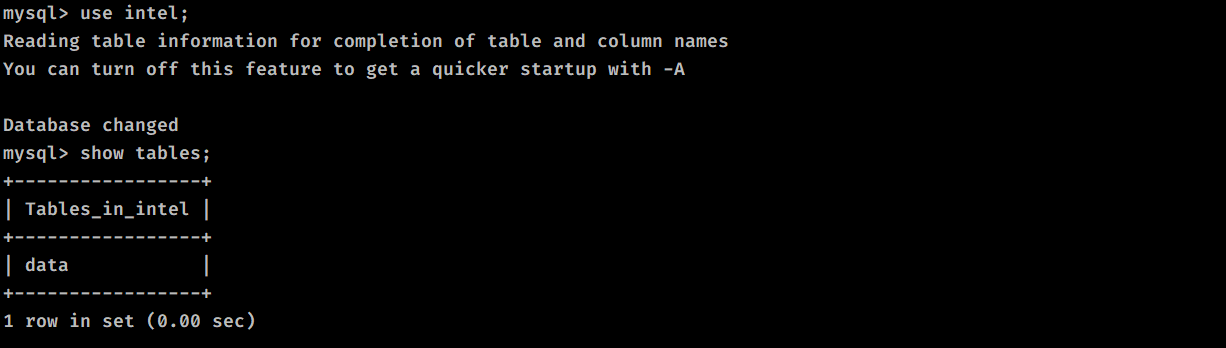
You will see a console output suggesting “ **New record updated successfully**”.

****

Now, lets connect to our MySQL database again and verify whether the database has new entries from the application.

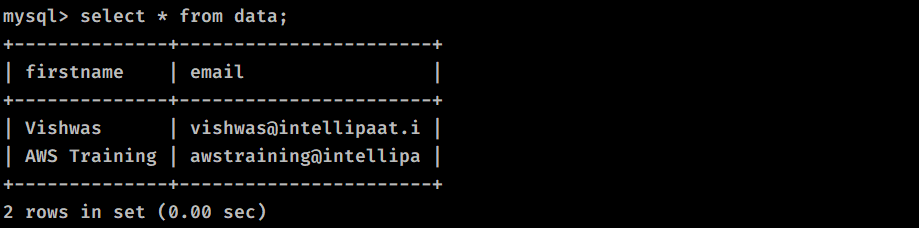
****

Change the database to **intel**.

****

Use the command below to list all the new row entries in the database.

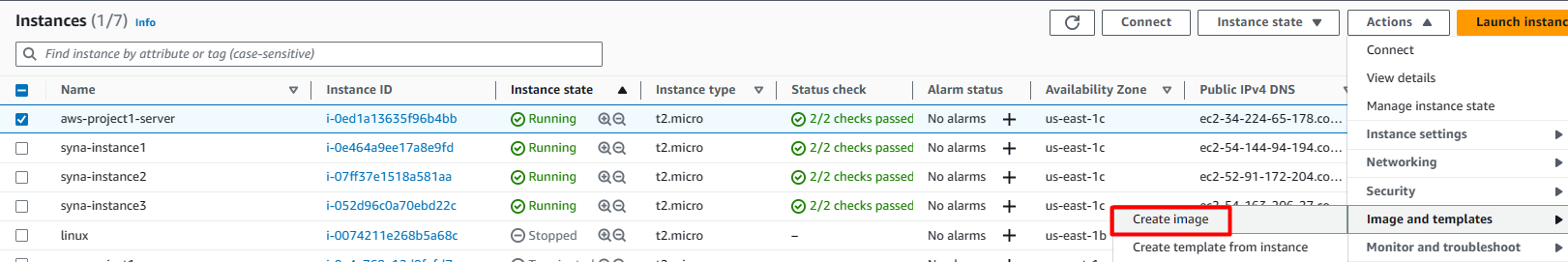
*select \* from data;*

****

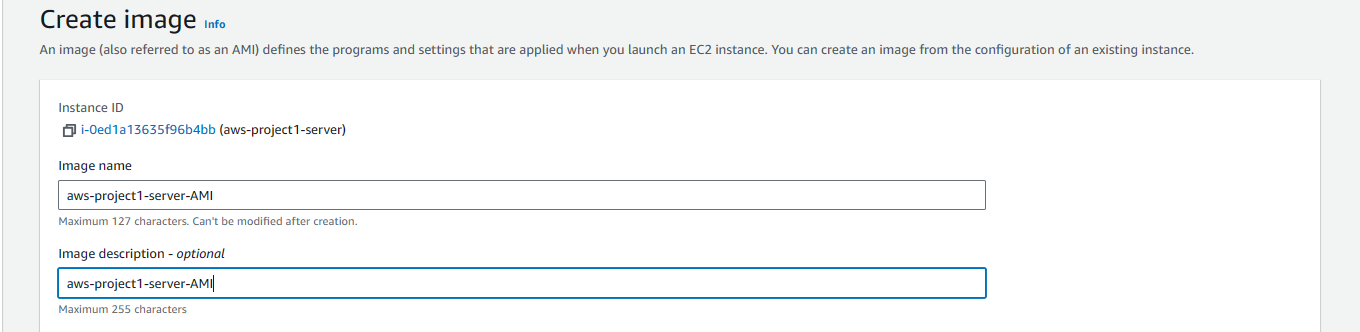
As you can see, the application is running successfully and we are able to store the data from the application.

Now, let's implement the Autoscaling for this application.

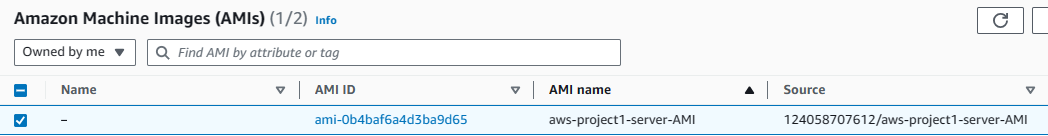
Choose the Instance and Click on **Create Image**.

****

Provide a Name to the image and click on **Create**.

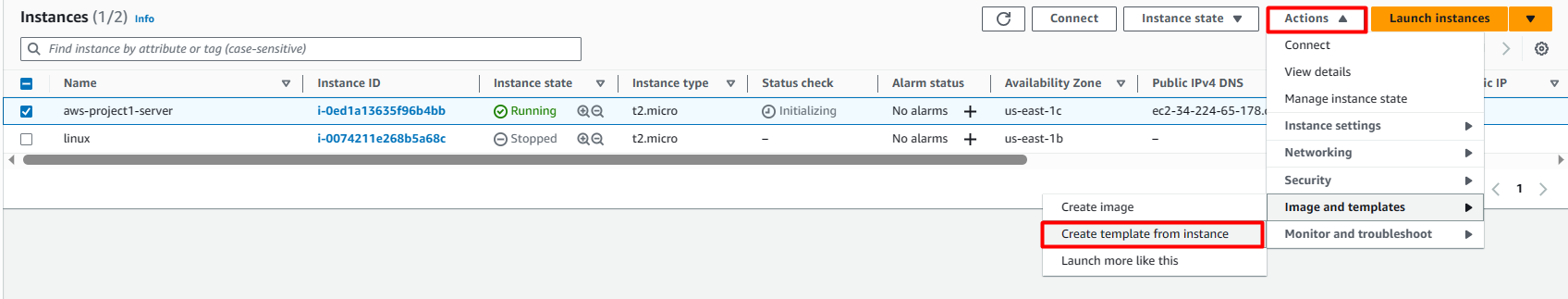
****

AMI is created.

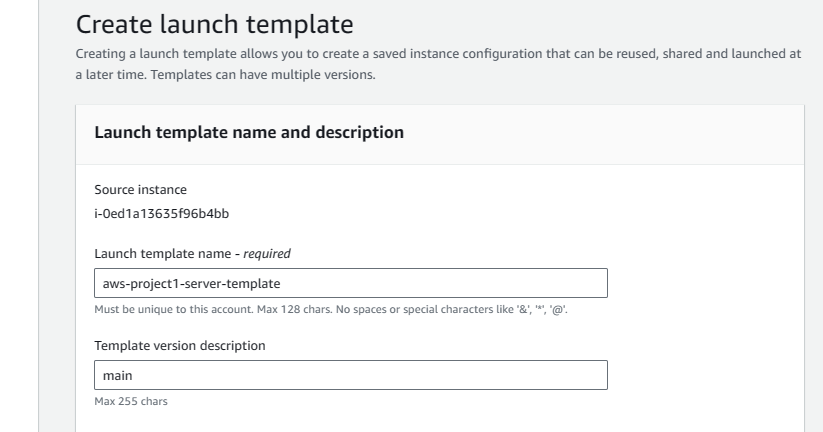
****

Now, we will use this AMI to create a Launch Template.

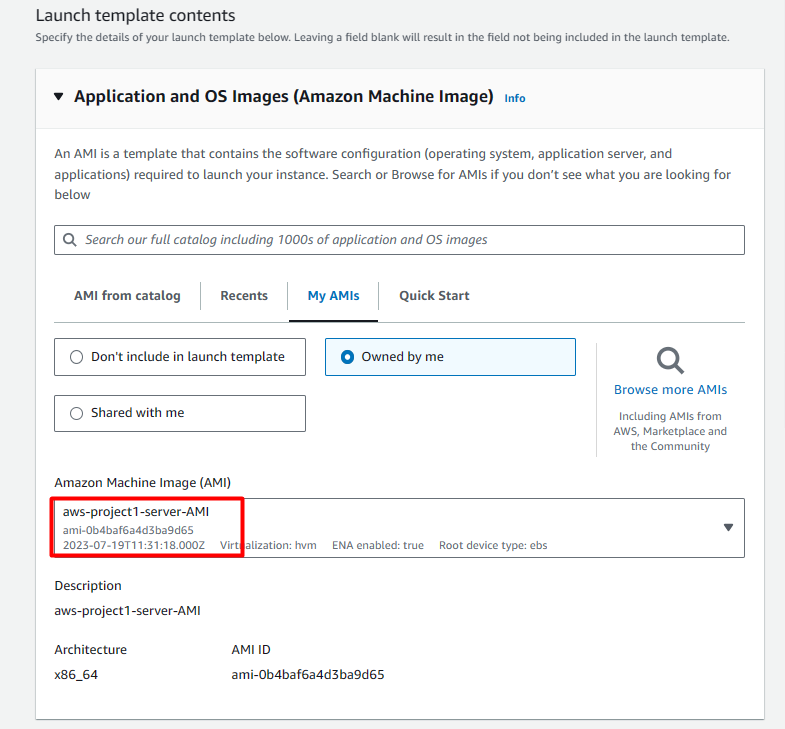
Select the EC2 Instance, Go to Actions, Images and Templates. Select Create Template from Instance.

****

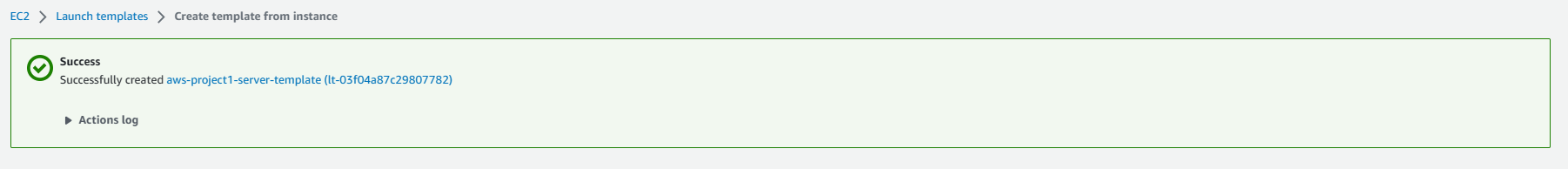
Provide a Name to the template and in version description, you can provide **main**.

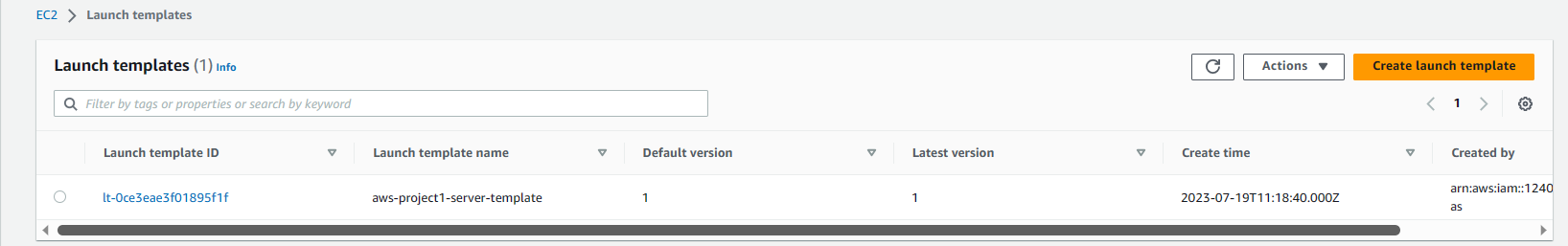
****

Now, When selecting AMI, go to **My AMI’s** and select the AMI that we created.

****

**Launch configuration** is created successfully.

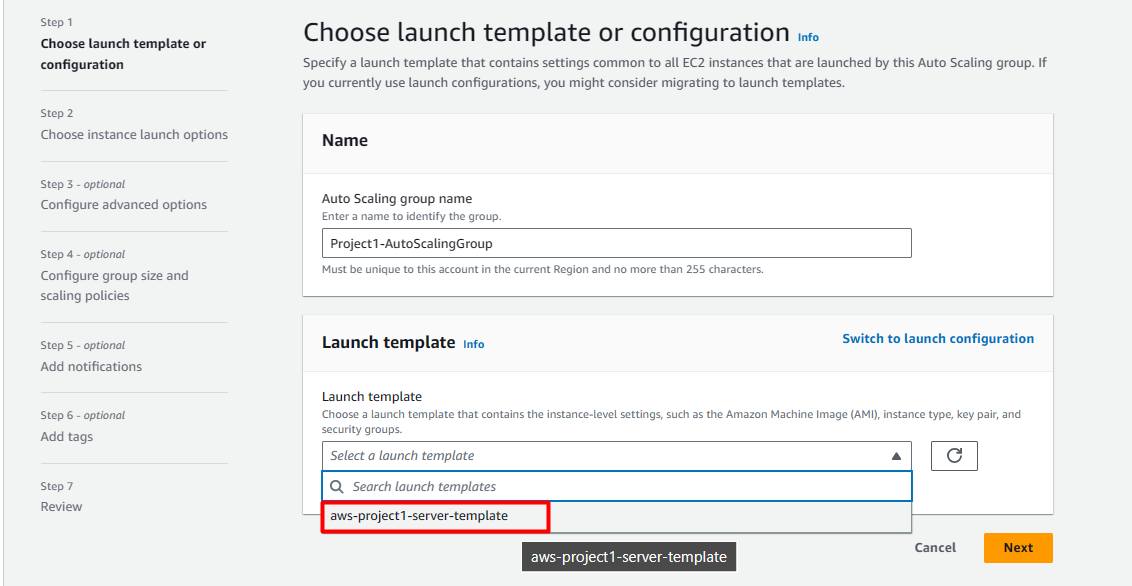
****

****

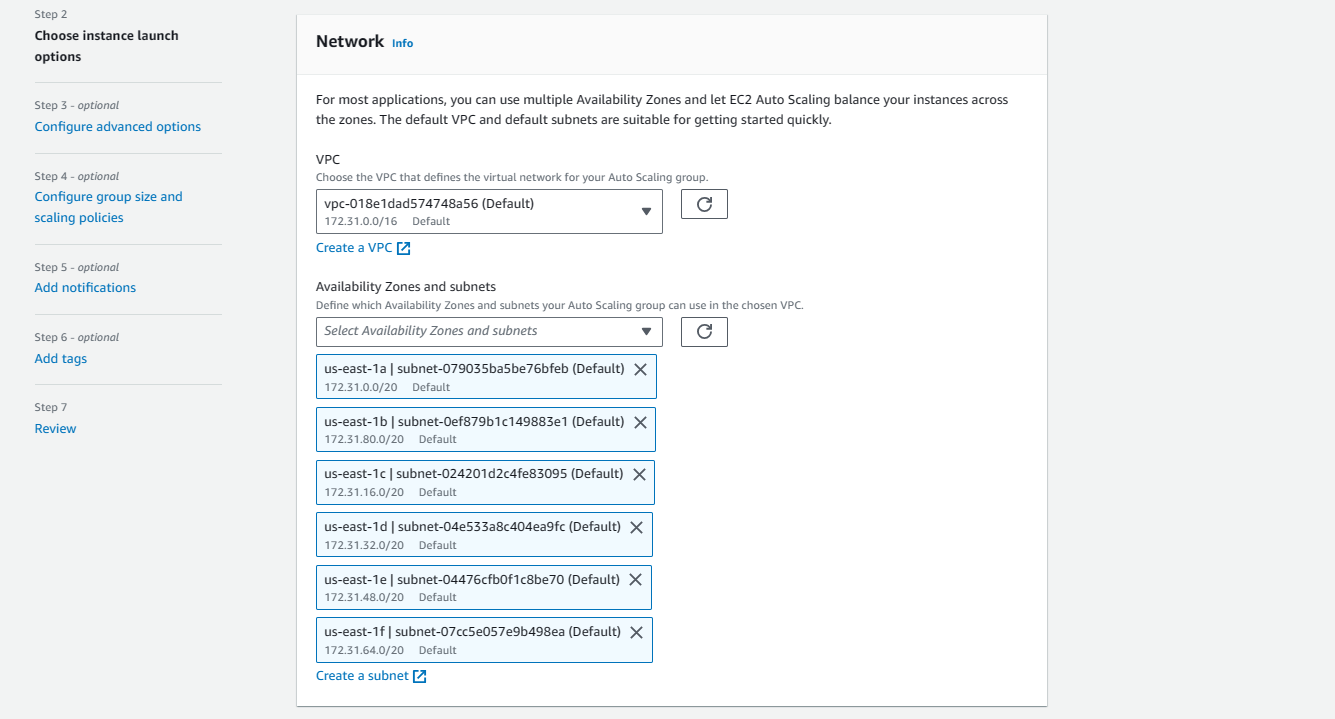
Now, let's create an Autoscaling Group using this Launch Template.

Provide a Name and select the Launch Template for Autoscaling group as shown below.

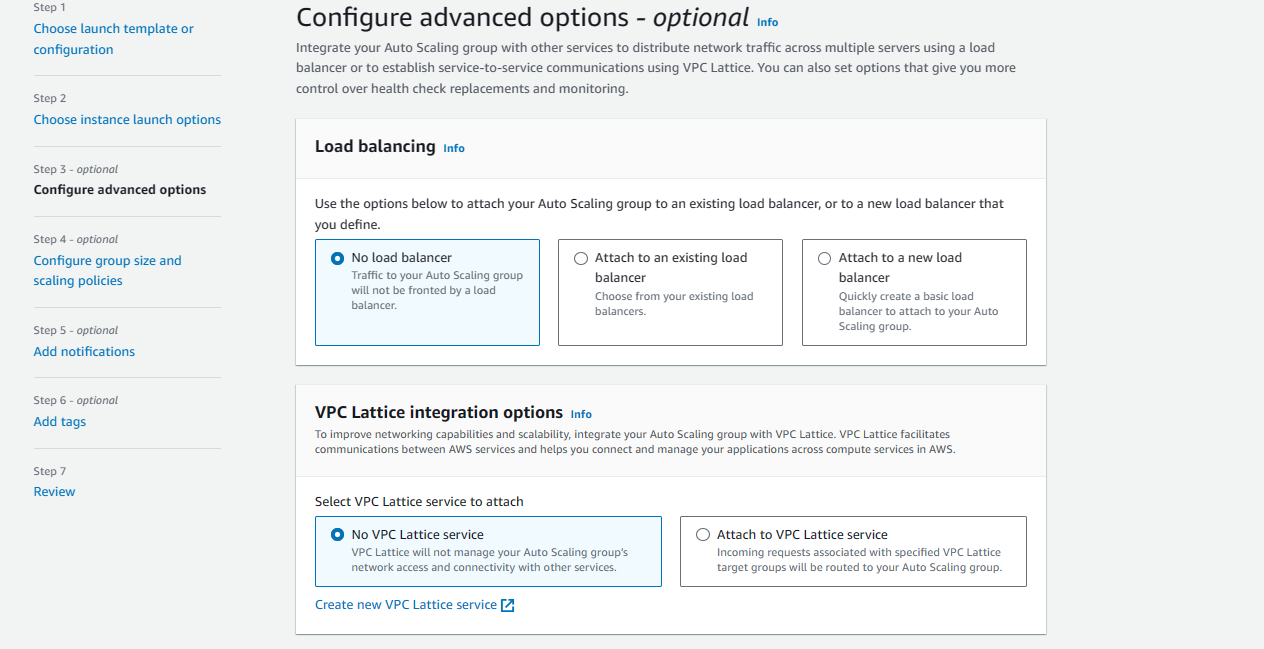
Click on Next.

****

Select the **VPC** and **minimum of two availability zones** for high availability. In this case we have selected all the availability zones available.

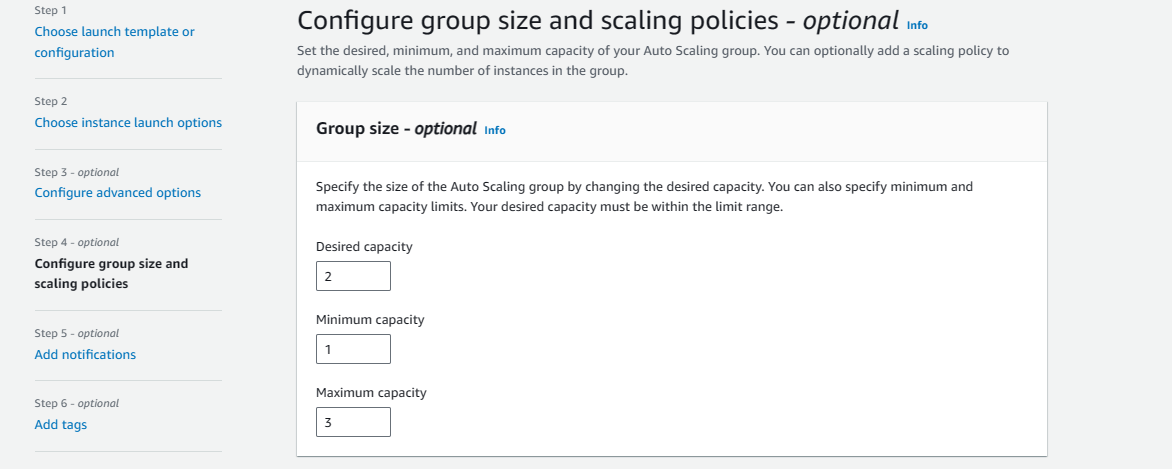
****

In load balancer, **Choose No Load Balancer**. Select the options as shown below.

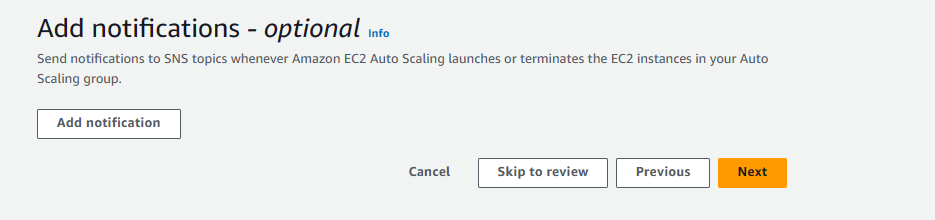
****

Choose the desired capacity in Group Size, in this case we have gone with **2,1 & 3**.

So, This autoscaling group will create minimum of two EC2 Instances to start with.

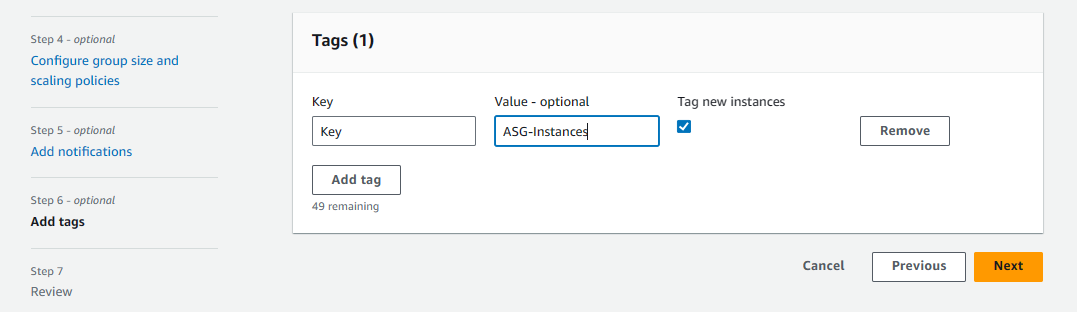
****

Notifications are optional, skip this and click on **Next**.

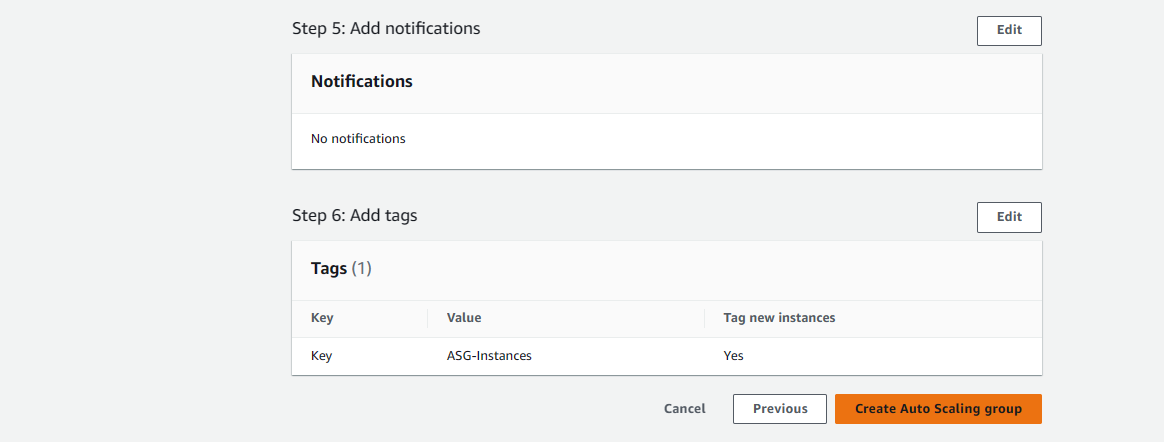
****

Tags are also optional, depending on whether you want to have tags on EC2 Instances created by Auto Scaling group.

Click on **Next**.

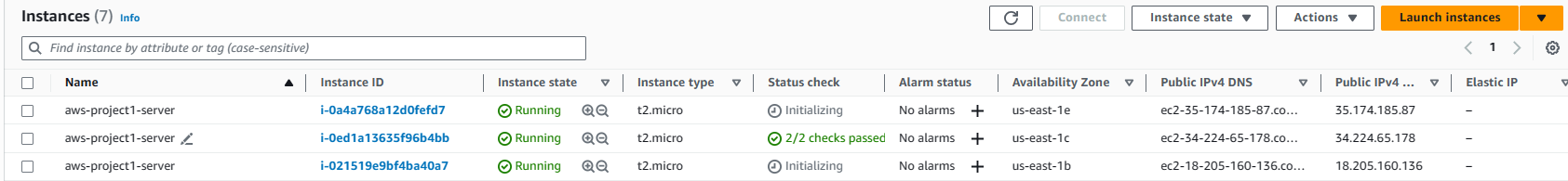
****

Review and Create Auto Scaling Group.

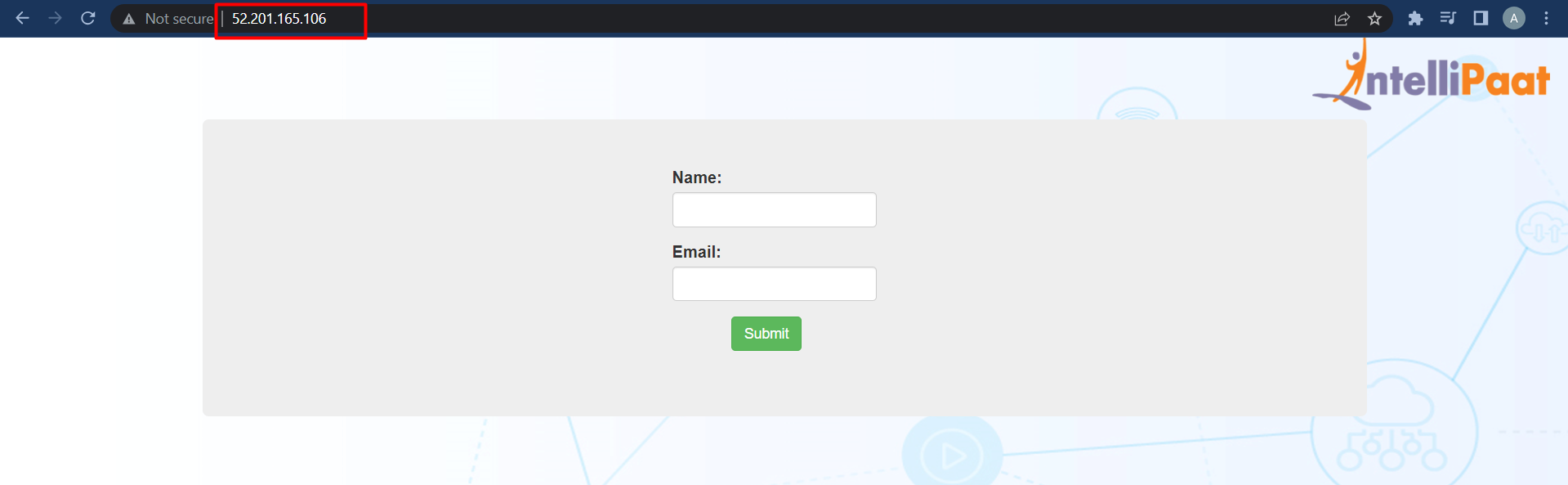
****

After creating Auto Scaling group, Go to Instances.

Here you should be able to see the newly created instances by Autoscaling Group.

****

Copy and paste the public IP of any of the newly created servers in the browser and you should be able to see the application already up and running inside them.

****

*Note: To clean up all the resources provisioned, terminate the RDS Instance first, then Autoscaling Group, Launch Templates and AMI in the order followed up by the Initial EC2 Server we started with.*

This Completes the tutorial for **AWS Project 1 - Deploying a Multi-tier Website using EC2.**

**Thanks.**