



Ministry of Communications
and Information Technology



Building a Highly Available, Scalable Web Application Project

Name: Mohamed Fathy soliman

Group: GIZ1_SW D8_M1e

Email: mohammedandfathy23@gmail.com

Business scenario overview:

This project requires an understanding of core AWS services, such as compute, storage, networking, and database services. The project also requires knowledge of architectural best practices, such as high availability, scalability, and security. Students should have completed the AWS Academy Cloud Architecting course to gain this necessary knowledge. Students who have completed the AWS Academy Cloud Foundations course and are enrolled in the AWS Academy Cloud Architecting course can also try to complete this project with the help of course materials, labs from courses, and educator guidance. Knowledge of any programming language, such as Python or JavaScript, is an advantage but isn't mandatory.

The scenario involves planning, designing, building, and deploying a web application to the AWS Cloud, adhering to the principles of the AWS Well-Architected Framework. During peak admissions periods, my design will focus on ensuring that the application can support thousands of users while maintaining high availability, scalability, load balancing, security, and optimal performance.

I will create an architectural diagram to depict various AWS services and their interactions with each other.

I will estimate the cost of using services by using the AWS Pricing Calculator.

Deploy a functional web application that runs on a single virtual machine and is backed by a relational database.

Architect a web application to separate layers of the application, such as the web server and database.

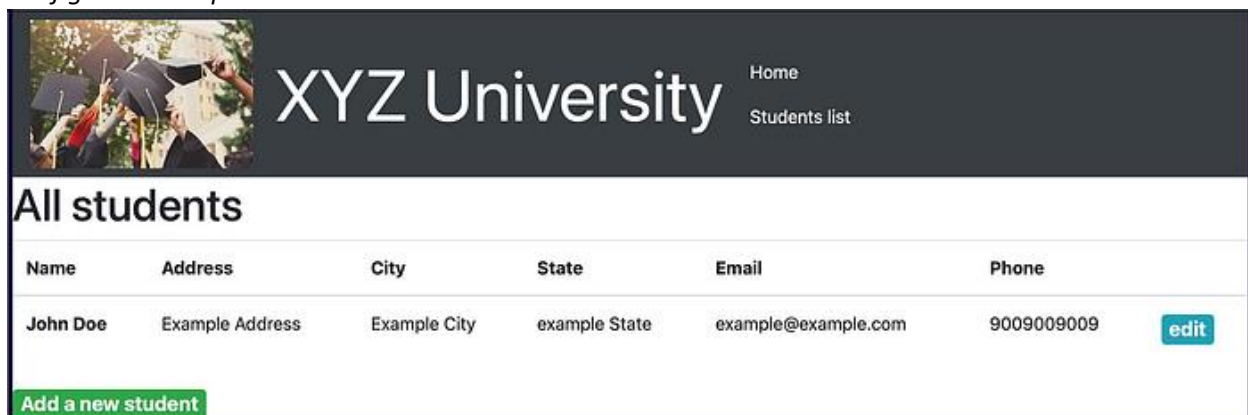
Create a virtual network that is configured appropriately to host a web application that is publicly accessible and secure.

Deploy a web application with the load distributed across multiple web servers.

Configure the appropriate network security settings for the web servers and database.

Implement high availability and scalability in the deployed solution.

Configure access permissions between AWS services



The screenshot shows a web application for XYZ University. The header features the university's name and two navigation links: 'Home' and 'Students list'. Below the header, the page is titled 'All students' and displays a table with student information. The table has columns for Name, Address, City, State, Email, and Phone. A single student, John Doe, is listed with example data. An 'edit' button is next to the student's phone number. At the bottom left, there is a green button labeled 'Add a new student'.

Name	Address	City	State	Email	Phone
John Doe	Example Address	Example City	example State	example@example.com	9009009009

Solution requirements:

- **Functional:** The solution meets the functional requirements, such as the ability to view, **add, delete, or modify** the student records, without any perceivable delay.
- **Load balanced:** The solution can properly balance user traffic to avoid overloaded or underutilized resources.
- **Scalable:** The solution is designed to scale to meet the **demands** that are placed on the application.
- **Highly available:** The solution is designed to have limited downtime when a web server becomes unavailable.
- **Secure:** *The database is secured and can't be accessed directly from public networks. The web servers and database can be accessed only over the appropriate ports. The web application is accessible over the internet. The database credentials aren't hardcoded into the web application.*
- **Cost-optimized:** The solution is designed to keep **costs low**.
- **High performing:** The routine operations (**viewing, adding, deleting, or modifying records**) are performed without a perceivable delay under normal, variable, and peak loads.

Assumptions

This project will be built in a controlled lab environment that has restrictions on services, features, and budget. Consider the following assumptions for the project:

- The application is deployed in one AWS Region (the solution **does not need to be multi-regional**).
- The website **does not need to be** available over HTTPS or a custom domain.
- The solution is deployed on *Ubuntu* machines by using the JavaScript code that is provided.
- Use the JavaScript code as written unless the instructions specifically direct you to change the code.
- The solution uses services and features within the restrictions of the lab environment.
- The **database** is hosted only in a **single Availability Zone**.
- The **website is publicly accessible without authentication**.
- The estimation of cost is approximate.

Disclaimer: A security best practice is to allow access to the website through the university network and authentication. However, because you are building this application as a **POC**.

Planning the design and estimating the cost:

I've planned out some architecture as well as a pricing calculator below:

AWS Icons



EC2 Instance



Application Load Balance



Nat Gateway



Internet Gateway



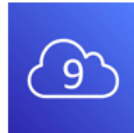
Secret role



Secret Manager

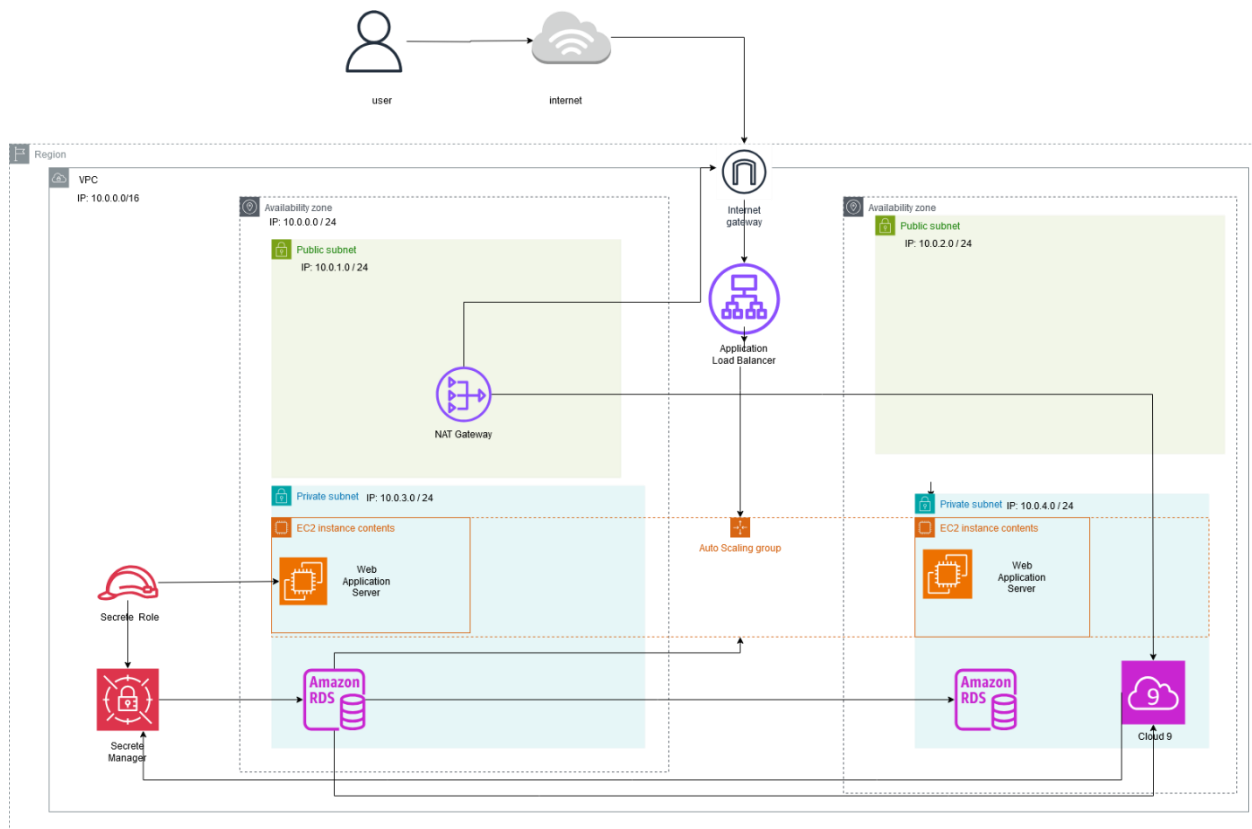


AWS RDS



AWS Cloud9

Architecture diagram of the solution:



Pricing calculator:



Contact your AWS representative: [Contact Sales](#)

Export date: 10/18/2024

Language: English

Estimate URL: <https://calculator.aws/#/estimate?id=dd71affb2c713c2304dedc03074bcb783551d76d>

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	128.80 USD	1,545.60 USD
		Includes upfront cost

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon Virtual Private Cloud (VPC)	No group applied	US East (N. Virginia)	0.00 USD	33.75 USD
Status: - Description: vpc network Config summary: Number of NAT Gateways (1)				
Elastic Load Balancing	No group applied	US East (N. Virginia)	0.00 USD	28.11 USD
Status: - Description: Config summary: Number of Application Load Balancers (1)				
Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	7.08 USD
Status: - Description: Config summary: Tenancy (Shared Instances), Operating system (Ubuntu Pro), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t2.micro), Pricing strategy (1yr No Upfront), Enable monitoring (disabled), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB				
Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	7.08 USD
Status: - Description: development server using Cloud9 Config summary: Tenancy (Shared Instances), Operating system (Ubuntu Pro), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t2.micro), Pricing strategy (Compute Savings Plans 1yr No Upfront), Enable monitoring (disabled), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)				
Amazon RDS for MySQL	No group applied	US East (N. Virginia)	0.00 USD	52.78 USD
Status: - Description: Config summary: Storage amount (20 GB), Storage for each RDS instance (General Purpose SSD (gp2)), Nodes (1), Instance type (db.m1.small), Utilization (On-Demand only) (60 %Utilized/Month), Deployment option (Multi-AZ), Pricing strategy (OnDemand)				

Solution Steps:

First, we will create **VPC**: A **Virtual Private Cloud (VPC)** is a virtual network on the cloud that enables you to securely connect and manage resources within your cloud environment. It provides isolation and control over your network infrastructure, allowing you to define and configure network settings according to your specific requirements.

At the top of the AWS Management Console, in the search bar, search for and choose **VPC**

Create VPC, and configure the following:

Resources to create: **Choose VPC only**

Name tag: Enter the name you want to define

IPv4 CIDR: Enter **10.0.0.0/16** Note: The CIDR range that is provided for the VPC configuration is only an example. You can use a different range as allowed by the lab environment.

Choose **Create VPC**.

Create VPC Info

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances.

VPC settings

Resources to create Info
Create only the VPC resource or the VPC and other networking resources.

☒ VPC only ☐ VPC and more

Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify.

VPC-WebApp

IPv4 CIDR block Info
☒ IPv4 CIDR manual input
☐ IPAM-allocated IPv4 CIDR block

IPv4 CIDR
10.0.0.0/24
CIDR block size must be between /16 and /28.

IPv6 CIDR block Info
☒ No IPv6 CIDR block
☐ IPAM-allocated IPv6 CIDR block
☐ Amazon-provided IPv6 CIDR block
☐ IPv6 CIDR owned by me

Tenancy Info
Default

Tags
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key: Name Value - optional: VPC-WebApp

You can add 40 more tags

VPC CIDR block in AWS is a crucial element in defining the IP address range for your VPC. It determines the number of IP addresses available for your resources and subnets. When choosing a CIDR block, you should consider the size, potential overlap, scaling needs, and subnetting requirements of your VPC.

The notation “10.0.0.0/16” represents a block of IP addresses ranging from 10.0.0.0 to 10.0.255.255. The “/16” indicates the subnet mask, which means that the first 16 bits of the IP address are fixed and the remaining 16 bits can be used to assign specific addresses within that range. This allows for a total of 65,536 possible IP addresses within the 10.0.0.0/16 subnet.

On the other hand, “10.0.0.0/24” represents a smaller block of IP addresses ranging from 10.0.0.0 to

10.0.0.255. The “/24” subnet mask means that the first 24 bits are fixed, leaving only 8 bits available for addressing. This results in a total of 256 possible IP addresses within the 10.0.0.0/24 subnet.

VPC reserve 5 IP address so you always subtract the IP address that you will receive for example 256 IP address subtract 5, and the available IP address is 251 addresses.

*Note: Add more **IP address for private resources** than public IP resources.*

IPv4 is always enabled default, you can enable ipv6 but you can't remove IPv4

Update the settings for the VPC:

Choose Actions > **Edit VPC settings.**

In the DNS settings section, **select Enable DNS hostnames.**

Choose **Save.**

The screenshot shows the AWS VPC dashboard. At the top, a green banner states: "You have successfully created 1 subnet: subnet-0b5faaf7d104e9983". Below this, the "Subnets (1/8) Info" section displays a table of subnets. The table has columns for Name, Subnet ID, State, VPC, and IPv4 CIDR. The subnets listed are:

Name	Subnet ID	State	VPC	IPv4 CIDR
-	subnet-0c8d1c058a24fa40e	Available	vpc-0766706d416368324	172.31.64.0/20
-	subnet-0c673c5a808244491	Available	vpc-0766706d416368324	172.31.48.0/20
-	subnet-04c79868f7c3a39af2	Available	vpc-0766706d416368324	172.31.0.0/20
-	subnet-0e834edc5f0e458c0	Available	vpc-0766706d416368324	172.31.16.0/20
-	subnet-0072a958048231767	Available	vpc-0766706d416368324	172.31.32.0/20
PublicSubnet1	subnet-01bc5e649b56577c0	Available	vpc-01ab6f19660584137 VPC...	10.0.1.0/24
PublicSubnet2	subnet-0b5faaf7d104e9983	Available	vpc-01ab6f19660584137 VPC...	10.0.2.0/24

Below the table, the details for "subnet-01bc5e649b56577c0 / PublicSubnet1" are shown. The details include:

- Subnet ID: subnet-01bc5e649b56577c0
- Subnet ARN: arnaws:ec2:us-east-1:629746028105:subnet/subnet-01bc5e649b56577c0
- State: Available
- IPv4 CIDR: 10.0.1.0/24
- Available IPv4 addresses: 251
- IPv6 CIDR association ID: -
- Availability Zone: us-east-1a

We will create Components of VPC, A VPC consists of the following components:

- Subnets:** Subnets are subdivisions of a VPC's IP address range. They allow you to logically isolate resources within your VPC and control access to them. You can configure routing tables at the subnet level to control traffic flow.
- Internet Gateway:** An Internet Gateway (IGW) is a horizontally scalable, redundant component that provides a connection between your VPC and the Internet. It allows resources in your VPC to communicate with the internet and vice versa.
- Route Tables:** Route tables define the rules for routing traffic within your VPC. Each subnet is associated with a route table, which determines how traffic is directed between subnets, the Internet, and other connected networks.

In the navigation pane, choose **Internet gateways**, and configure the following:

Choose: **Create internet gateway**

Name tag: **AccessToInternetGW**

Choose **Create internet gateway**

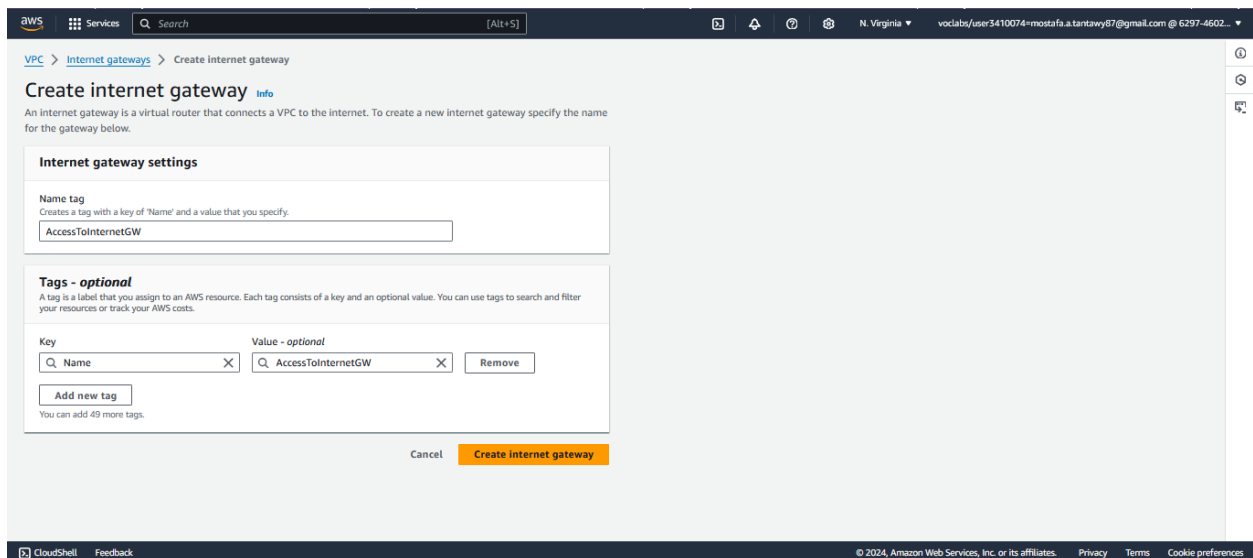
Internet Gateway: An Internet Gateway (IGW) is a horizontally scalable, redundant component that provides a connection between your VPC and the Internet. It allows resources in your VPC to **communicate** with the **internet** and vice versa.

Attach the internet gateway to the VPC:

Choose Actions > **Attach to VPC**.

Available VPCs: **Choose VPC That We Created**

Choose **Attach internet gateway**



The screenshot shows the AWS Management Console interface for creating an internet gateway. The breadcrumb trail is 'VPC > Internet gateways > Create internet gateway'. The main heading is 'Create internet gateway' with an 'Info' link. A descriptive text states: 'An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.' The 'Internet gateway settings' section includes a 'Name tag' field with the value 'AccessToInternetGW'. Below this is the 'Tags - optional' section, which contains a table with one tag: 'Name' as the key and 'AccessToInternetGW' as the value. At the bottom of the form are 'Cancel' and 'Create internet gateway' buttons.

In the navigation pane, choose Subnets, and configure the following: Choose Create subnet. •

VPC ID: Choose **VPC That We Have Created**

Subnet name: Enter **Public Subnet 1**

Availability Zone: **Choose the first Availability Zone from the dropdown list**

IPv4 VPC CIDR block: Enter **10.0.0.0/16**

IPv4 subnet CIDR block: Enter **10.0.1.0/24**

Choose **Create subnet**

Subnets: Subnets are subdivisions of a VPC's IP address range. They allow you to logically isolate resources within your VPC and control access to them. You can configure routing tables at the subnet level to control traffic flow.

create one more time but make sure the Ipv4 subnet CIDR Block is different and does not overlap:

*For example, use Public subnet 2 CIDR **10.0.2.0/24** to **put AZ in a different Availability zone** to make it highly available, add another subnet to specific another AZ because you can only **associate to 1 AZ only**.*

We need to **auto-assign public IPv4**, public IP is **not enabled by default**, you need to **Enable** auto-assign public IP.

Make sure to Enable the **Auto-assign IP** setting, it allows for the automatic allocation of IPv4 addresses to instances launched within the **public subnet**. This means that whenever you launch a new instance in the subnet, it will **automatically** be assigned a **unique public IPv4 address**. *This eliminates the need for manual configuration and ensures that each instance has a distinct address for communication purposes.*

Fun fact: When you stop your instance and start it, the public IP changes however the private IP address stays the same.

Route Table: Give your **route table the name** *PublicRouteTable* and select the **VPC we created**:

Click on the **“Create”** button to complete the process.

A public route table in AWS is a network resource that controls the **traffic** flow between subnets within a virtual private cloud (**VPC**) and the **internet**. It contains a set of rules, known as **routes**, that determine how traffic is directed. In this section, we will explore the key features and components of an AWS public route table.

Route → Edit routes → (destination **0.0.0.0/0** means **Anywhere**) (target Internet gateway)

When it comes to routing traffic, AWS **public route tables** function similarly to traditional routing tables. They use a destination-based routing model, where each route in the table specifies a destination CIDR block and a target. The target can be an **internet gateway**, a **subnet has auto-assigned public IPv4**.

Now we need to edit explicit **subnet associations** (*attach subnet to our route table*)

Explicit subnet associations

Edit subnet associations.

Select and **Add** a Public subnet to attach to the route table.

Edit subnet associations
Change which subnets are associated with this route table.

Available subnets (2/2)

<input checked="" type="checkbox"/>	Name	Subnet ID	IPv4 CIDR	IPv6 CIDR	Route table ID
<input checked="" type="checkbox"/>	publicsubnet	subnet-069976d27a8d169b5	10.0.1.0/24	–	Main (rtb-0532d9f100b469423)
<input checked="" type="checkbox"/>	publicsubnet2	subnet-04f369654b256df55	10.0.2.0/24	–	Main (rtb-0532d9f100b469423)

Selected subnets

subnet-04f369654b256df55 / publicsubnet2 X subnet-069976d27a8d169b5 / publicsubnet X

Once you have created a public route table, you can configure its routes and associate it with subnets within your VPC. By associating a subnet with a route table, you enable the subnets to use the routes defined in the table for traffic routing.

It's important to note that a **VPC** can have **multiple route tables**, but each **subnet** can only be associated with **one route table** at a time. This allows you to have different routing configurations for different subnets within your VPC.

*for example, we will **create** now a **private subnet associated with a private route table**.*

Create the same procedure as creating a public subnet but for the private subnet.

Make sure CIDR block different as well as availability zone. example:

Private subnet 10.0.3.0/24. AZ (1a).

Private subnet2 10.0.4.0/24. AZ (1b).

Important note: no need to assign (auto-assign public IPv4), because it's private no need to have a public IPv4.

Create a Private Route Table, In the Amazon VPC dashboard, click on Route Tables in the left navigation pane.

2. Click on the **Create Route Table** button

3. In the **Name tag** field, enter a descriptive name for your route table (e.g., "**Private Route Table**").

4. From the **VPC** dropdown menu, select the **VPC** that we have created.

5. Click on the **Create** button to create the route table.

Step 4: Associate Subnets with the Route Table

1. In the **Route Tables** dashboard, locate the newly created private route table.

2. Click on the **Actions** button and select **Edit Subnet Associations**.

3. In the **Subnet Associations** section, click on the **Add Subnet** button.

4. **Select the private subnets** that you want to associate with the private route table.

5. Click on the **Save** button to associate the subnets with the route table.

Important note for private subnet:

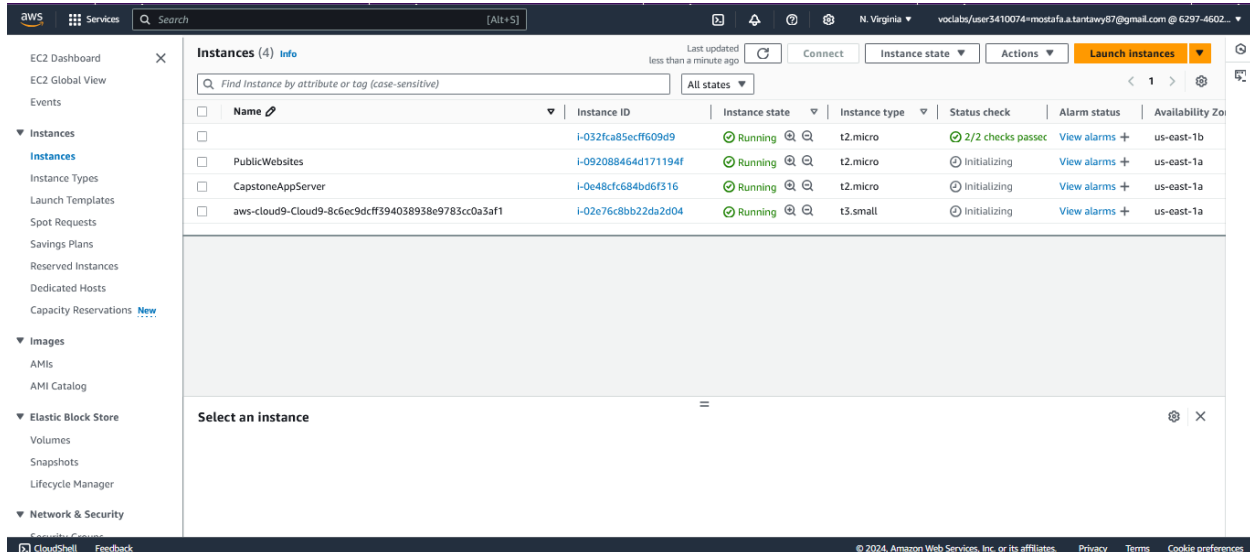
*Private subnet **doesn't have auto-assign public IPv4**.*

*Route table **doesn't route to internet gateway** which access to the internet, only local traffic in VPC network which is logically isolated.*

Destination field, enter the CIDR block for the network traffic you want to route (e.g., 10.0.0.0/16).
Network will move internally only at VPC Network CIDR Block.

Creating EC2 for our web application

Click on “Launch Instance” to start the instance creation process.



3. In the Application and OS Images section, under Quick Start, choose **Ubuntu**

4. Select the desired instance type and click on “Next: Configure Instance Details”. **t2.micro** for **free tier** (just for POC, if for production website will increase the instance type based on the specification business need).

In the key pair section, for Key pair name, choose **vockey**

In the Network settings section, configure the following:

Choose **Edit**.

VPC: Choose **VPC That We Have Created**.

Auto-assign public IP: **Choose Enable**.

Note: This allows the instance to have a public IP address automatically assigned to it. This will enable the instance to communicate over the internet.

Firewall (security groups): Choose **Create security group**.

Security group name: **Enter SG NAME YOU WANT**

Choose **Add** security group rule.

Keep the existing SSH rule, and add two new rules with the following settings:

■New rule 1: For Type, choose **HTTP**. For Source type, choose **Anywhere**. Note: This rule allows traffic from a web browser.

■New rule 2: For Type, choose **MYSQL/Aurora**. For Source, enter **10.0.0.0/16** Note: This rule allows data to be exported from the database in a later task.

9. In the “**Advanced details**” page, you can also specify an **IAM instance profile** if you want the instance to communicate with AWS Secrets Manager or any other AWS services. This allows the instance to have the necessary permissions to access secrets securely.

10. Lastly, you can provide user data on the “Configure Instance Details” page. **User data** is a script that runs on the instance during the boot process. You can specify the **necessary steps to be performed after the instance is launched**.

```
#!/bin/bash -xe
apt update -y
apt install nodejs unzip wget npm mysql-client -y
#wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCAP1-1-DEV/code.zip -P
/home/ubuntu
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCAP1-1-79581/1-lab-
capstone-project-1/code.zip -P /home/ubuntu
cd /home/ubuntu
unzip code.zip -x "resources/codebase_partner/node_modules/*"
cd resources/codebase_partner
npm install aws aws-sdk
export APP_PORT=80
npm start &
echo '#!/bin/bash -xe
cd /home/ubuntu/resources/codebase_partner
export APP_PORT=80
npm start' > /etc/rc.local
chmod +x /etc/rc.local
```

11. Review all the configurations and click on “**Launch**” to create the EC2 instance.

Important: Check the formatting of the script after copying it into the user data field. If the code lines appear to be broken.

Note: This script installs Node.js, the student records application (website, JavaScript, CSS, and other files), and the MySQL database on the EC2 instance.

Important: Before Status check column moving to the next task, confirm that the instance is in the Running state and that the says “2/2 checks passed.” This will take a few minutes.

To test the web application, access it from the internet by using the Public IPv4 address or Public IPv4 DNS of the instance.

Note: Ensure to use http (instead of https) while accessing the web application from the browser.

Perform a few tasks, such as adding new student records, editing records, and deleting records. Keep so that you have data to migrate to a new database in a later task. now have a functional website that is running on an EC2 instance.

Components exist on one virtual machine, is not flexible and is difficult to scale. In the next phase, students will separate the various layers. at least one record in the database The architecture that we have created so far was quick to build, with few components and a low cost. This approach would be suitable for a proof of concept (POC).

CREATING RDS (Relational Database)

The first step is to create a security group that will be associated to the RDS to open the ports needed for connection to the EC2 web server.

Create a security group for the database:

At the top of the AWS Management Console, in the search bar, search for and choose **VPC**.

In the navigation pane, choose **Security Groups**.

Choose **create Security Group**, and configure the following:

Security group name: **Enter Your name DBSG**.

Description: Enter **Security group** for database

VPC: Start to enter **VPC** that we had created and **choose** it when it appears.

In the **Inbound rules** section, choose **Add rule** and configure the following:

Type: Choose **MYSQL/Aurora**. Source: Enter **10.0.0.0/16** in the field to the right of Custom.

Choose **Create Security Group**.

1. Login to your AWS account and navigate to the **Amazon RDS** console.
2. Click on **"Create database"** to start the RDS creation process.
3. Choose the standard creation method and select the **MySQL engine** and keep the **engine version default**.
4. In the **"Templates"** section, choose the **"Free Tier"** option if you want to use the free tier benefits. This will ensure that you stay within the **free tier** limits and avoid any charges.
5. Assign a **name** for your **DB instance identifier**, this name will be unique across all DB instances owned by your AWS Accounts.
6. You have two options to choose either using **master password** or **enable manage master credentials in AWS Secrets Manager**.

Master username: **nodeapp**

To securely store the **master password**.

OR

Enable Manage Master credentials in **AWS Secretes Managers**.

You can write the **master password** and then use the **Cloud9 IDE** in the next step to create the secret in the **secrets manager**.

Type: Choose **MYSQL/Aurora**. Source: Enter **10.0.0.0/16** in the field to the right of Custom.

Choose **Create security group**.

7. Choose the instance type that best suits your needs. For example, you can select the **"t3.micro"** instance type if you are just starting out and don't require a lot of computational power.
8. Set the storage size for your RDS instance. If you want to use the same storage as the instance type, **select the default** option.

9. Very important step:

Connect to EC2 Compute Resources (it's like attaching EC2 to RDS) no point in connecting RDS and EC2 if your connectivity section is not established.

Ensure that they are in the **same VPC** (Virtual Private Cloud) or have proper network connectivity between them. This can be done by checking the VPC settings and security groups associated with the EC2 instance and the RDS database. **Make sure that the security groups allow incoming and outgoing traffic between the two instances, for RDS we will create the SG in the below guide.**

EC2 instance: choose the public website that we have created.

Optional: If you launched your RDS already and you want to connect to your EC2.

Management console:

Click RDS

Choose your database

Connectivity and security sections

Connected compute resources sections.

Action and select the instances

10. choose the existing subnet, and VPC security group select our DB-SG just now created.

Database authentication: **Default (Password authentication).**

Monitoring: **Disable** Enable Enhanced monitoring.

CLICK **Additional configurations**

Specify a **name** for your **Initial database**. This is the name of the database that will be created once the **RDS** is created. You will use this database in the RDS instance for your web application hosted on the **EC2**.

Initial database name: STUDENTS

(The initial database is important that you write the name correctly and in uppercase, because name will be used to connect to your RDS instance).

Disable automated backups

Disable Encryptions

Disable Log exports

Disable Maintenance

No preference Maintenance window

Disable Deletion protection

LAUNCH RDS DATABASE

Migrate the data from the original database, which is on an **EC2** instance, to the new Amazon **RDS** database.

```
mysqldump -h 10.0.1.152 -u nodeapp -p --databases STUDENTS > data.sql
```

```
bash - "ip-10-0-1-70.ec2 x" Immediate (Javascript) x
version: 0.10.0.1.70-ec2 x
}
voclabs:~/environment $ mysqldump -h 10.0.1.152 -u nodeapp -p --databases STUDENTS > data.sql
Enter password:
voclabs:~/environment $ ls
data.sql README.md
voclabs:~/environment $
```

```
mysql -h students.c3mfcmmx9cjj.us-east-1.rds.amazonaws.com -u nodeapp -p STUDENTS < data.sql
```

```
voclabs:~/environment $ mysql -h students.c3mfcmmx9cjj.us-east-1.rds.amazonaws.com -u nodeapp -p STUDENTS < data.sql
Enter password:
voclabs:~/environment $ ls
data.sql README.md
voclabs:~/environment $
```

Cloud9 is an integrated development environment (**IDE**) provided by Amazon Web Services (AWS). It allows developers to write, run, and debug their code in the cloud. With **Cloud9**, you can access your development environment from anywhere using just a web browser.

AWS **Cloud9** Scripts file to migrate the original data into the Amazon **RDS** database.

Create a New instance, AWS recommends using **t3.micro**.

With **Secure Shell (SSH)**, you can securely access your AWS **Cloud9** environment or any other remote server using encryption. This ensures that your communication and data transfer are protected from eavesdropping and tampering. Make sure it's in a **public subnet** and **within your vpc**.

Template script:

```
aws secretsmanager create-secret \  
  --name Mydbsecret \  
  --description "Database secret for web app" \  
  --secret-string "{\"user\":\"<username>\",\"password\":\"<password>\",\"host\":\"<RDS  
Endpoint>\",\"db\":\"<dbname>\"}"
```

Go to **Configuration tab** in RDS Section

<DB name>: **STUDENTS**

<username>: **nodeapp**

<password>: **Go to AWS Secret Manager, *rds!db-autogenerate!D***

Note: If you created it using **Cloud9** it will be named **Mydbsecret**.

<RDS Endpoint>: **RDS connectivity & security**

Modified version script:

```
aws secretsmanager create-secret \  
  --name Mydbsecret \  
  --description "Database secret for web app" \  
  --secret-string "{\"user\":\"nodeapp\",\"password\":\"RY$uD:6c.iZ3C4*0W48?!0.HeF-  
p\",\"host\":\"students.cxkdvdkrxasc.us-east-1.rds.amazonaws.com\",\"db\":\"STUDENTS\"}"
```

Now we have finished the Optional section

Note:

- The secret that is created in this step stores database credentials, which the web application will use through an AWS Identity and Access Management (IAM) role named LabRole. This enhances the security posture by not storing credentials in the application or the database.
- LabRole was pre-created in the lab environment. The role facilitates secure interactions between AWS services. The role already includes the appropriate permissions policies.

In this task, We will provision a **new EC2 instance** in the public subnet and install the web application. The solution is as follows:

1. At the top of the AWS Management Console, in the search bar, search for and choose **EC2**
2. Choose Launch instance > **Launch instance**, and then configure the following:

In the Name and tags section, for Name, enter **CapstoneAppServer**

in the Application and OS Images section, under Quick Start, choose **Ubuntu**.

In the Instance type section, for Instance type, choose **t2.micro**.

In the key pair section, for the Key pair name, choose **vockey**.

in the Network settings section, configure the following:

Choose Edit. VPC: Choose **VPCWeHaveCreated**.

Subnet: Choose **Public Subnet 1**.

Auto-assign public IP: Choose **Enable**.

Firewall (security group): Choose Select **existing security group**.

Common security groups: Choose **EC2-SG**.

Expand the Advanced Details section, and configure the following:

IAM instance profile: Choose **LabinstanceProfile**.

User data: Copy and paste the following code:

```
#!/bin/bash -xe
apt update -y
apt install nodejs unzip wget npm mysql-server -y
#wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCAP1-1-DEV/code.zip -P
/home/ubuntu
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCAP1-1-79581/1-lab-
capstone-project-1/code.zip -P /home/ubuntu
cd /home/ubuntu
unzip code.zip -x "resources/codebase_partner/node_modules/*"
cd resources/codebase_partner
npm install aws aws-sdk
mysql -u root -e "CREATE USER 'nodeapp' IDENTIFIED WITH mysql_native_password BY 'student12'";
mysql -u root -e "GRANT all privileges on *.* to 'nodeapp'@'%';"
mysql -u root -e "CREATE DATABASE STUDENTS;"
mysql -u root -e "USE STUDENTS; CREATE TABLE students(
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    city VARCHAR(255) NOT NULL,
    state VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(100) NOT NULL,
    PRIMARY KEY ( id ));"
sed -i 's/.*bind-address.*/bind-address = 0.0.0.0/' /etc/mysql/mysql.conf.d/mysqld.cnf
systemctl enable mysql
service mysql restart
export APP_DB_HOST=$(curl http://169.254.169.254/latest/meta-data/local-ipv4)
export APP_DB_USER=nodeapp
```

```
export APP_DB_PASSWORD=student12
export APP_DB_NAME=STUDENTS
export APP_PORT=80
npm start &
echo '#!/bin/bash -xe
cd /home/ubuntu/resources/codebase_partner
export APP_PORT=80
npm start' > /etc/rc.local
chmod +x /etc/rc.local
```

An important part, we will go to **Cloud9** to migrate **EC2** data to **RDS** and connect it.

Now we will mysqldump EC2 web application to our Cloud9 environment.

```
mysqldump -h <EC2PrivateIP> -u nodeapp -p --databases <initaildatabasename> > data.sql
```

Run the following command, replacing ``<EC2PrivateIP>`` with the **private IP address of your EC2 instance**, ``<initaildatabasename>`` with the **initial name of your database**, and ``data.sql`` with a desired name for your SQL dump file:

Private IP EC2 we have created for our web application, initial database we have created earlier **STUDENTS**.

When you run this command, you will be prompted to enter the password for the ``nodeapp`` user. Enter the password and press Enter.

```
mysqldump -h 10.0.0.254 -u nodeapp -p --databases STUDENTS > data.sql
```

4. When you run this command, you will be prompted to enter the password for the ``nodeapp`` user. Enter the password and press Enter.

Password: **student12**

5. The ``mysqldump`` command will connect to the EC2 instance, dump the specified database, and save the output to the ``data.sql`` file in your Cloud9 environment.

6. Once the command finishes executing, you will have a complete SQL dump of your database in the ``data.sql`` file.

Now we will *interface it with RDS*

```
mysql -h <RDSEnpoint> -u nodeapp -p <initial datsbase name> < data.sql
```

Then go to RDS to **copy** the **RDS endpoint**

password: RY\$uD:6c.iZ3C4*0W48?!0.HeF-p

from secret manger when we created it earlier in RDS

```
mysql -h students.cxkdvdkrxasc.us-east-1.rds.amazonaws.com -u nodeapp -p STUDENTS < data.sql
```

As we set up EC2, now we need to attach ALB (application load balancer) to our EC2 but first, what is ALB in the first place?

A load balancer is a component that evenly distributes incoming network traffic across multiple servers or instances to ensure the high availability and scalability of your application. It helps distribute the workload and prevents any single server from becoming overwhelmed with traffic.

Application Load Balancer (ALB)

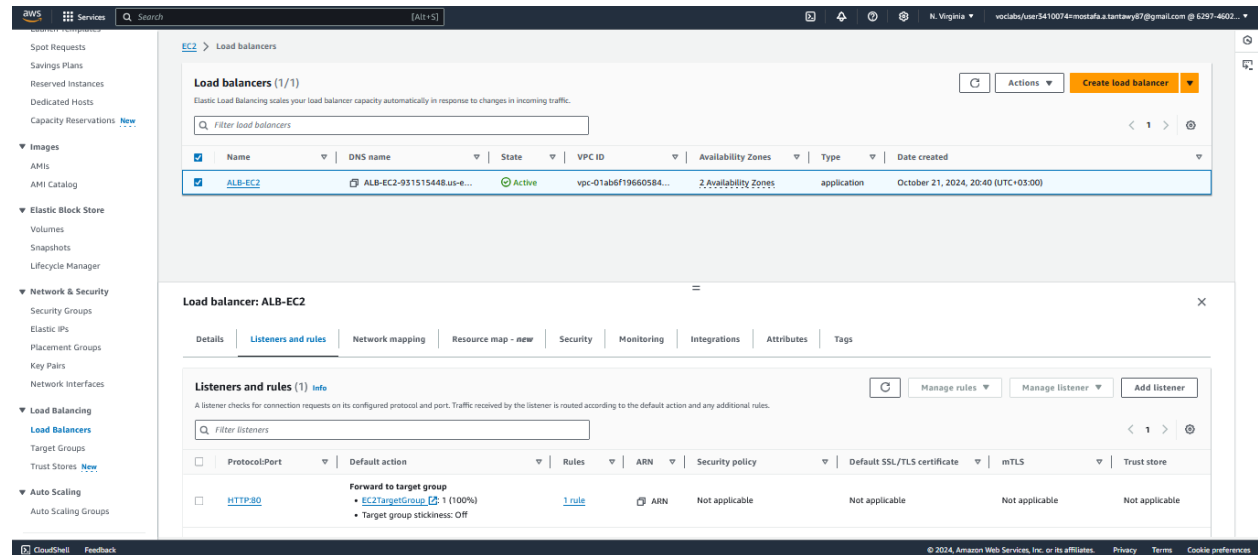
The Application Load Balancer (ALB) is a type of load balancer provided by AWS that operates at the application layer (Layer 7) of the OSI model. It can intelligently route traffic based on various criteria such as URL path, host header, or query strings.

A Target Group acts as a logical group of EC2 instances that receive traffic from the ALB. To create a Target Group, follow these steps:

1. Open the Amazon **EC2 console**.
2. In the navigation pane, under **"LOAD BALANCING"**, choose **"Target Groups"**.
3. Choose **"Create target group"**.
4. Choose target type **instances** (Supports load balancing to instances within a specific VPC) **provide a name** for your Target Group, and select the protocol HTTP for PoC purposes and port **80** for **HTTP**, use the **VPC** that we created, rest keep it default.
5. **Add** the EC2 instances that should be part of this Target Group.
6. Click **Include as pending below**.
7. **Create** a target group.

Step 3: Create an Application Load Balancer

Now that you have a Target Group, you can create an ALB that will distribute traffic to your EC2 instances. Follow these steps to create an ALB:



1. In the Amazon EC2 console, under “LOAD BALANCING”, choose “Load Balancers”.
2. Choose “Create Load Balancer”.
3. Select “Application Load Balancer” and click “Create”.
4. Provide a name for your ALB, Internet-facing (Public ALB) and select the VPC and Public Subnets of each availability zone where your EC2 instances are located.
5. Remove the default SG and configure a new security group for ALB.
6. Configure the listeners by specifying the protocol and port to listen on, and the Target Group you created in the previous step.
7. Review your settings and click “Create” to create the ALB.

Default DNS:

Once your ALB is created, you will be provided with a DNS name for your load balancer. No need to update your DNS settings to point your domain or subdomain to this DNS name, this will ensure that incoming traffic is directed to your ALB us its POC and the requirement section they don’t need the domain name to be listed and be connected alias with ALB.

Check Target group is listening to ELB so ALB send traffic to the EC2.

Select your ALB

Listeners and rules section:

Now we have finished ALB, we will create now Autoscaling Group (ASG)

To copy a new Amazon Machine Image (**AMI**) instance, follow these steps:

1. Open the Amazon EC2 console.
2. In the navigation pane, choose “**Instances**”.
3. Select the **instance** that *you want to copy the AMI*.
4. Right-click on the selected instance and choose “**Create Image**”.
5. In the “**Create Image**” dialogue box, provide a unique name and description for the new AMI.
6. Choose “**Create**” to start the AMI creation process.
7. Wait for the AMI creation process to complete. This might take a few minutes.
8. Once the AMI is created, you can view it in the “AMIs” section of the EC2 console.

You can proceed to create an autoscaling group. In the EC2 dashboard.

Click on the “**Autoscaling Groups**” option in the sidebar menu. Then, click on the “**Create Auto Scaling group**” button.

Step 5: Configure the Autoscaling Group

In the autoscaling group creation wizard, you will need to configure several settings:

- Name: **Provide a name** for your autoscaling group.
- Launch Template or Configuration: Select the instance configuration you want to use for scaling.

Creating EC2 Launch template.

Choose the launch template that we created just now:

Network: Choose the VPC and subnets where your instances will be launched.

Load Balancer: If you have a load balancer set up, you can associate it with your autoscaling group.

Group Size: Specify the desired capacity for your group, which is the number of instances you want to maintain, Scaling Policies: Define the scaling policies that dictate when and how instances are added or removed based on demand.

Load Testing

```
voclabs:~/environment $ loadtest --rps 1000 -c 500 -k http://ALB-EC2-931515448.us-east-1.elb.amazonaws.com
Requests: 4997, requests per second: 1000, mean latency: 5.5 ms

Target URL:          http://ALB-EC2-931515448.us-east-1.elb.amazonaws.com
Max time (s):        10
Target rps:          1000
Concurrent clients:  42
Agent:               keepalive

Completed requests:  9999
Total errors:        0
Total time:          10.001 s
Mean latency:        4.6 ms
Effective rps:       1000

Percentage of requests served within a certain time
 50%    2 ms
 90%   10 ms
 95%   19 ms
 99%   35 ms
100%   88 ms (longest request)
```

Conclusion

Creating an autoscaling group in AWS allows you to automatically manage the number of instances in your infrastructure based on demand. By following the steps outlined in this guide, you can easily create an autoscaling group and ensure that your application can handle varying workloads efficiently.