

# Building a Highly Available, Scalable Web Application

This document outlines the process of building a highly available, scalable web application using AWS services, as part of the AWS Academy's "Building a Highly Available, Scalable Web Application" lab project. The solution addresses the challenges of handling peak traffic for Example University's student records web application while ensuring minimal downtime and optimized performance. Through the implementation of best practices in AWS architecture, this project demonstrates how to use load balancing, automatic scaling, and AWS security configurations to build a robust cloud-based application.



**by Ahmed Nasr**

# Introduction and Solution Requirements

Author: Ahmad Nasr Ali, AWS Cloud Specialist

Student Number:1110140440

Group Name: CAI1\_SWD8\_M2d

Linkedin Profile: [www.linkedin.com/in/ahmed-nasr-367a21227](https://www.linkedin.com/in/ahmed-nasr-367a21227)



In today's fast-paced digital environment, the need for applications that are highly available, scalable, and secure is paramount. Cloud solutions, specifically those offered by Amazon Web Services (AWS), provide the infrastructure and tools necessary to achieve these goals. This document outlines the process of building a highly available, scalable web application using AWS services, as part of the AWS Academy's "Building a Highly Available, Scalable Web Application" lab project.

This project is a proof of concept (POC) designed to improve the reliability and performance of Example University's student records web application. The solution addresses the challenges of handling peak traffic during admissions while ensuring minimal downtime and optimized performance. Through the implementation of best practices in AWS architecture, this project demonstrates how to use load balancing, automatic scaling, and AWS security configurations to build a robust cloud-based application.

## Solution Requirements

The solution aims to meet the following requirements:

- **Functional:** The web application must allow users to view, add, delete, and modify student records with no perceivable delay.
- **Load Balanced:** Traffic should be evenly distributed across multiple web servers to avoid overloaded resources.
- **Scalable:** The system must be able to scale in response to user demand.
- **Highly Available:** The application should minimize downtime even if one or more servers fail.
- **Secure:** Only the appropriate services can access the database, and sensitive credentials are managed securely.
- **Cost-Optimized:** The solution is designed to be cost-efficient.
- **High Performing:** Regular operations must be fast, even under peak loads.



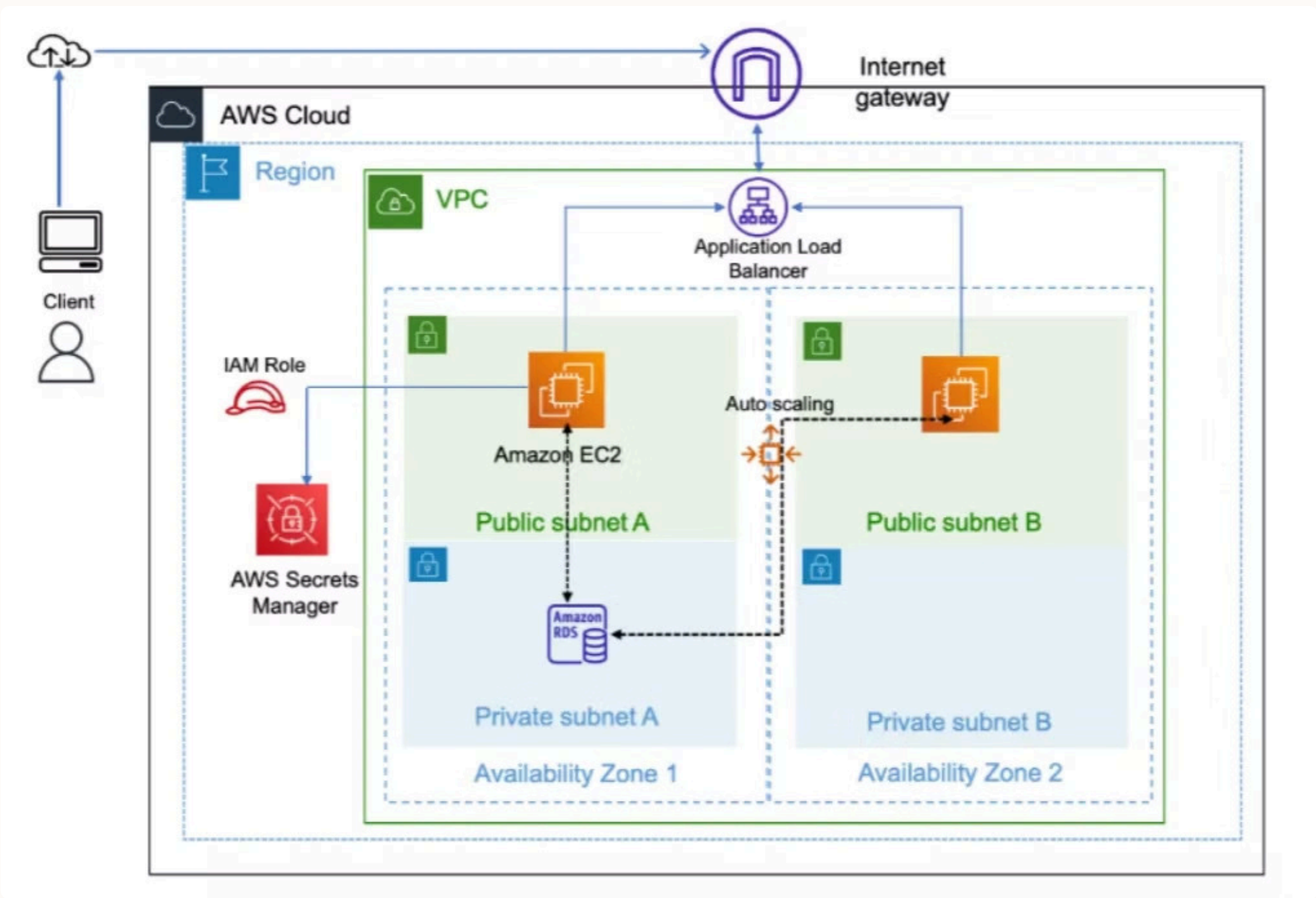
# Planning and Design (Phase 1)

## Architectural Diagram

To ensure a well-structured solution, the architecture follows a multi-tier model separating the web server and database layers. The design includes:

- Amazon EC2 instances to host the web application.
- Amazon RDS for a managed relational database.
- Amazon Application Load Balancer (ALB) to distribute incoming traffic across multiple instances.
- AWS Auto Scaling to automatically adjust the number of instances based on demand.
- Amazon VPC (Virtual Private Cloud) with both public and private subnets for optimal security.

Figure 1: Architectural Diagram (Include a diagram showing components such as VPC, public/private subnets, EC2 instances, RDS, ALB, etc.)



## Cost Estimation

Using the AWS Pricing Calculator, we estimated the cost for running this application in the us-east-1 region for 12 months. The estimate includes EC2 instances, RDS, load balancer, and other services used. Based on the calculator, the total cost for the solution is approximately \$14,153.54 USD per year. (Note, the cost include real life scenario cost estimation)

10/17/24, 12:14 AM

My Estimate - AWS Pricing Calculator

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

Export date: 10/17/2024

Language: English

Estimate URL: <https://calculator.aws/#/estimate?id=a860323bb70440c33698dde8a5afe64f4bac574f>

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
196.22 USD	1,163.11 USD	14,153.54 USD
Includes upfront cost		

### Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon EC2	No group applied	US East (N. Virginia)	196.22 USD	85.05 USD

Status: -

Description:

**Config summary:** Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 4), Advance EC2 instance (t3a.small), Pricing strategy ( 1yr Partial Upfront), Enable monitoring (enabled), EBS Storage amount (40 GB), DT Inbound: Internet (50 GB per month), DT Outbound: Internet (50 GB per month), DT Intra-Region: (0 TB per month)

Elastic Load Balancing	No group applied	US East (N. Virginia)	0.00 USD	40.85 USD
------------------------	------------------	-----------------------	----------	-----------

Status: -

Description:

**Config summary:** Number of Application Load Balancers (2)

Amazon RDS Custom for SQL Server	No group applied	US East (N. Virginia)	0.00 USD	700.41 USD
----------------------------------	------------------	-----------------------	----------	------------

Status: -

Description:

**Config summary:** Storage for each RDS Custom for SQL Server instance (General Purpose SSD (gp2)), Storage amount (1000 GB), Instance type (db.m5.xlarge), Number of RDS Custom for SQL Server instances (1), Utilization (On-Demand only) (730 Hours/Month), Database edition (Web), Deployment option (Single-AZ), License (AWS-provided), Pricing strategy (Reserved 1yr No Upfront), Additional backup storage (1000 GB)

Amazon Virtual Private Cloud (VPC)	No group applied	US East (N. Virginia)	0.00 USD	332.30 USD
------------------------------------	------------------	-----------------------	----------	------------

Status: -

Description:

**Config summary:** Working days per month (22) Number of NAT Gateways (1) Number of In-use public IPv4 addresses (2), Number of Idle public IPv4 addresses (1)

Amazon Simple Queue Service (SQS)	No group applied	US East (N. Virginia)	0.00 USD	4.50 USD
-----------------------------------	------------------	-----------------------	----------	----------

Status: -

Description:

**Config summary:** DT Inbound: Internet (50 GB per month), DT Outbound: Internet (50 GB per month), Standard queue requests (1 million per month), Data transfer cost (4.5)



# Creating the Basic Web Application (Phase 2)

## Virtual Network Setup

The first step in building the solution is creating the necessary networking components. The web application requires a Virtual Private Cloud (VPC) with public and private subnets. The VPC is configured to allow internet access through an Internet Gateway for the web servers and restrict direct access to the database.

VPC > Your VPCs > Create VPC

Create VPCInfo

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

VPC settings

Resources to createInfo

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.

myvpc

IPv4 CIDR blockInfo

☒ IPv4 CIDR manual input

☐ IPAM-allocated IPv4 CIDR block

IPv4 CIDR

10.0.0.0/16

CIDR block size must be between /16 and /28.

IPv6 CIDR blockInfo

☒ No IPv6 CIDR block

☐ IPAM-allocated IPv6 CIDR block

☐ Amazon-provided IPv6 CIDR block

☐ IPv6 CIDR owned by me

TenancyInfo

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

QName

X

Value - optional

Qmyvpc

X

Remove tag

Add tag

You can add 49 more tags

Cancel

Create VPC

## Deploying the Web Server

Using Amazon EC2, we launch a virtual machine to host the web application. The instance runs the latest Ubuntu AMI, and the provided JavaScript code is deployed to serve as the frontend and backend of the student records application.

EC2 > ... > Launch an instance

Launch an instanceInfo

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tagsInfo

Name

Public-Web

Add additional tags

Application and OS Images (Amazon Machine Image)Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

QSearch our full catalog including 1000s of application and OS images

Recents

My AMIs

Quick Start

Amazon Linux

aws

macOS

Mac

Ubuntu

ubuntu

Windows

Microsoft

Red Hat

Red Hat

SUSE Linux

SUSE Linux

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

## Testing Deployment

Once the instance is running, the web application is accessible via its public IPv4 address. Basic operations such as viewing, adding, deleting, and modifying student records are tested to ensure functionality.

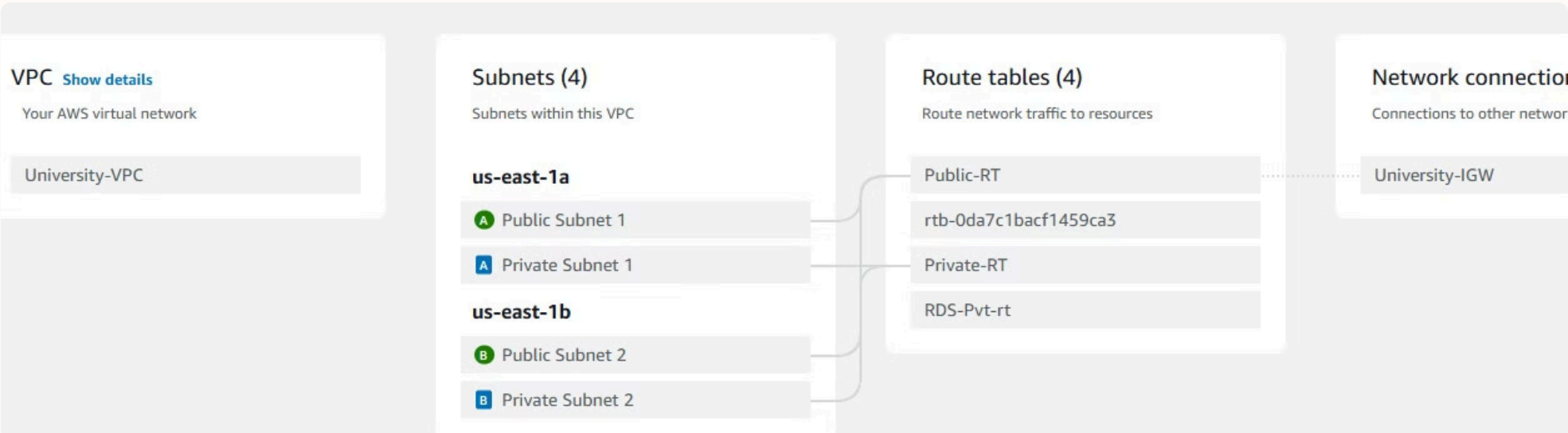
Made with Gamma



# Decoupling Components (Phase 3)

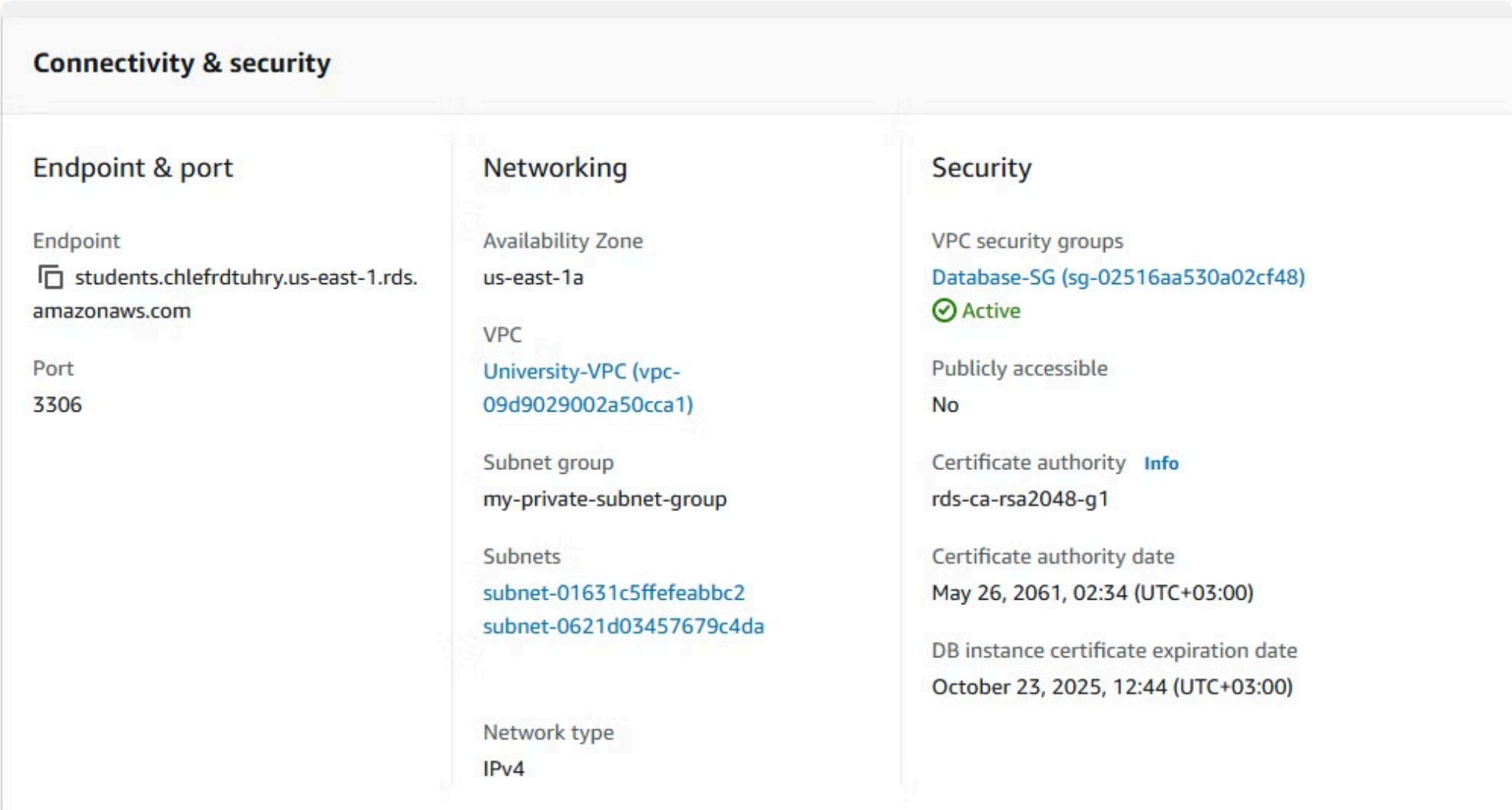
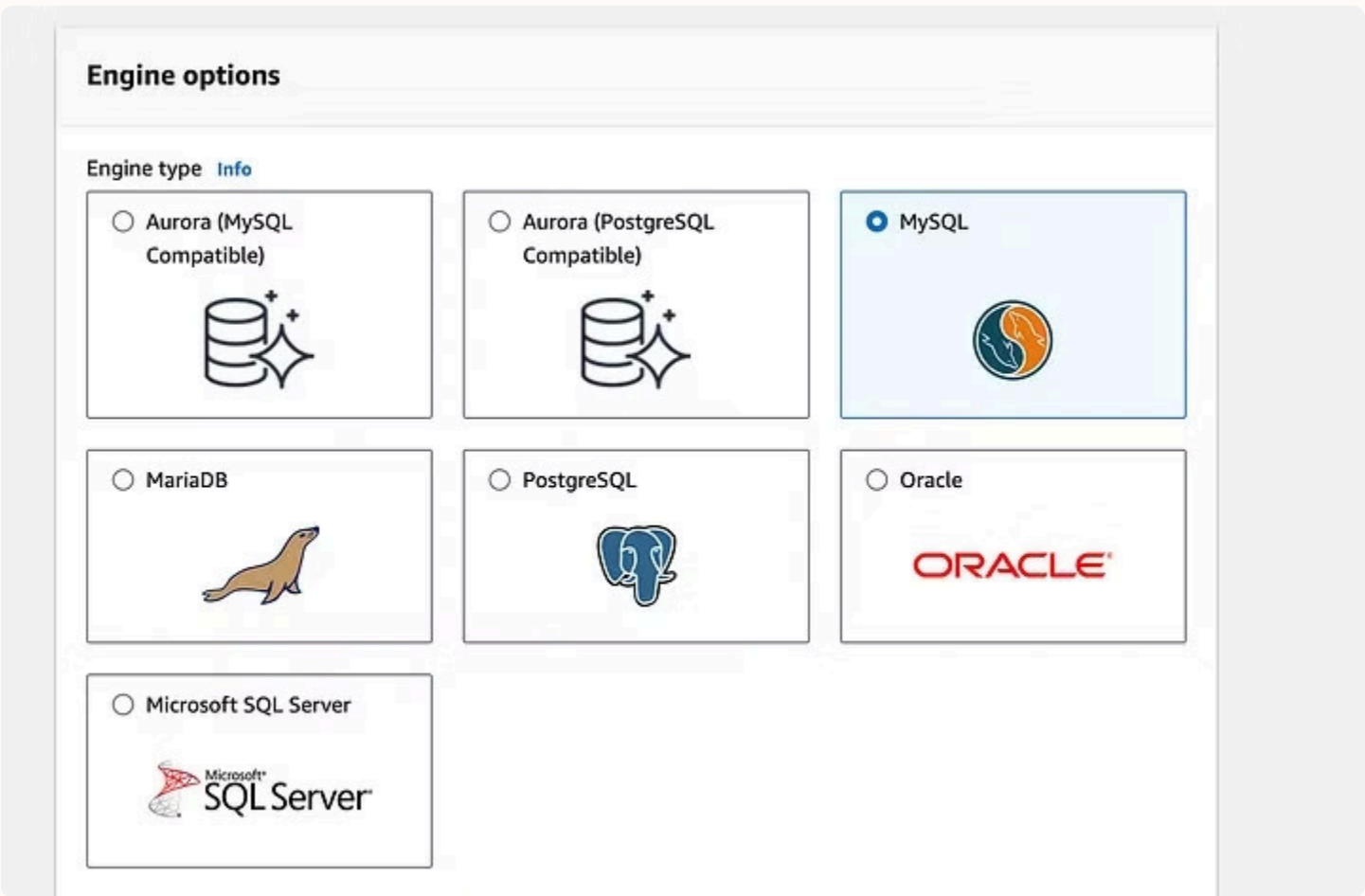
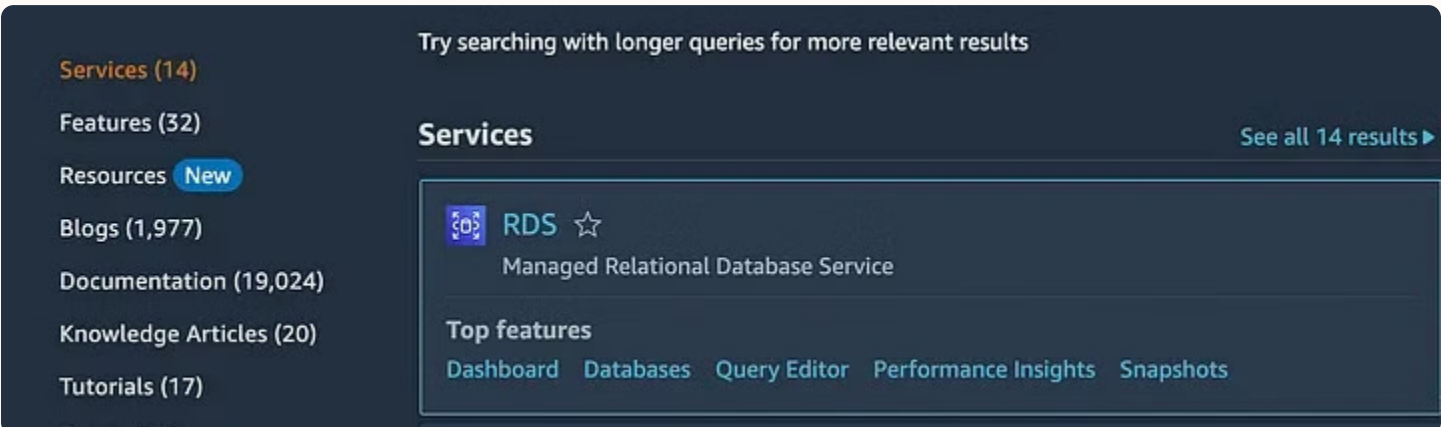
## Virtual Network Reconfiguration

To enhance the architecture, the database is separated from the web server. The VPC is updated with private subnets in at least two Availability Zones, ensuring high availability for the database.



## Creating the Amazon RDS Database

A MySQL database is provisioned using Amazon RDS. Security groups are configured to allow access to the database only from the web servers. For database credentials, AWS Secrets Manager is used to securely store and access these secrets.



## Setting Up Cloud9 Development Environment

An AWS Cloud9 environment is provisioned to run AWS CLI commands and manage the deployment. Using AWS Cloud9, we run scripts to store database credentials in Secrets Manager and connect the web application to the new RDS instance.

## Migrating the Database

The student records data is migrated from the EC2 instance to the new RDS database using AWS CLI commands and a provided migration script.

First Script about creating SecretsManager:

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

```
{
  "ARN": "arn:aws:secretsmanager:us-east-1:626828110804:secret:Mydbnewsecret-8YhdQ0",
  "Name": "Mydbnewsecret",
  "VersionId": "8ad0cca9-75f0-47f2-97cc-0d920c1efb1f"
}
```

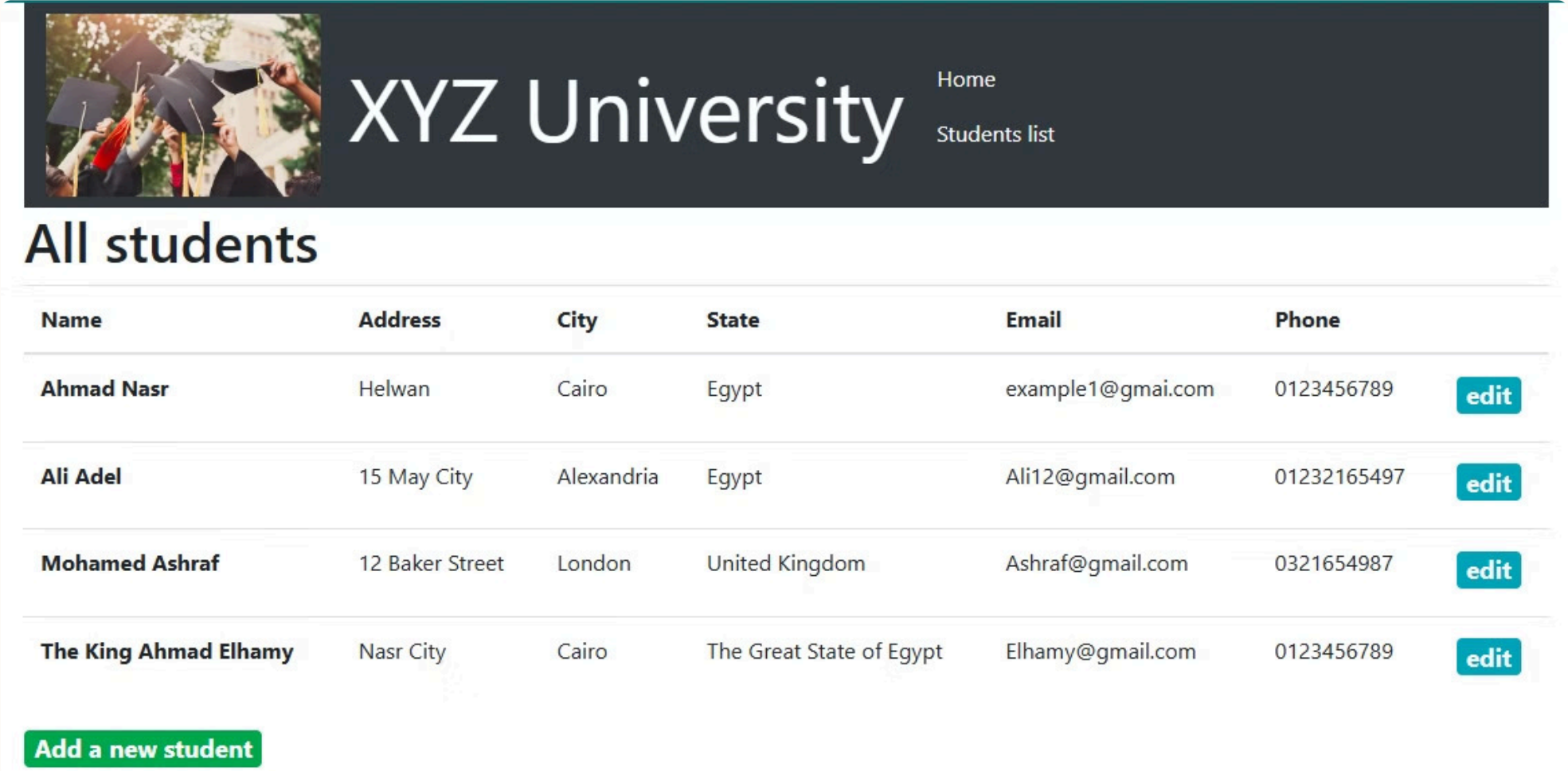
Migrate the Database using these two Scripts in Cloud9:

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

mysql -h -u nodeapp -p STUDENTS < data.sql
```

## Testing the Application

After reconfiguring the application to use the RDS database, the web application is tested to ensure it can still perform all operations.

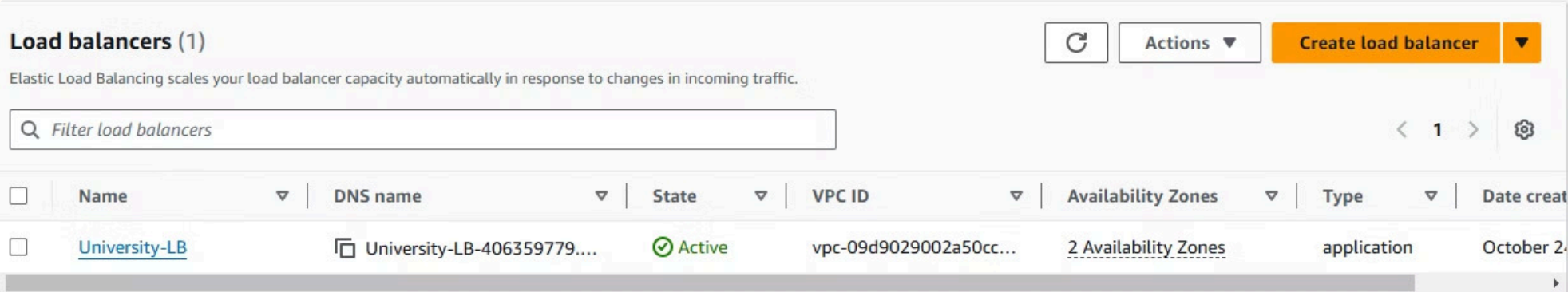




# High Availability and Scalability (Phase 4)

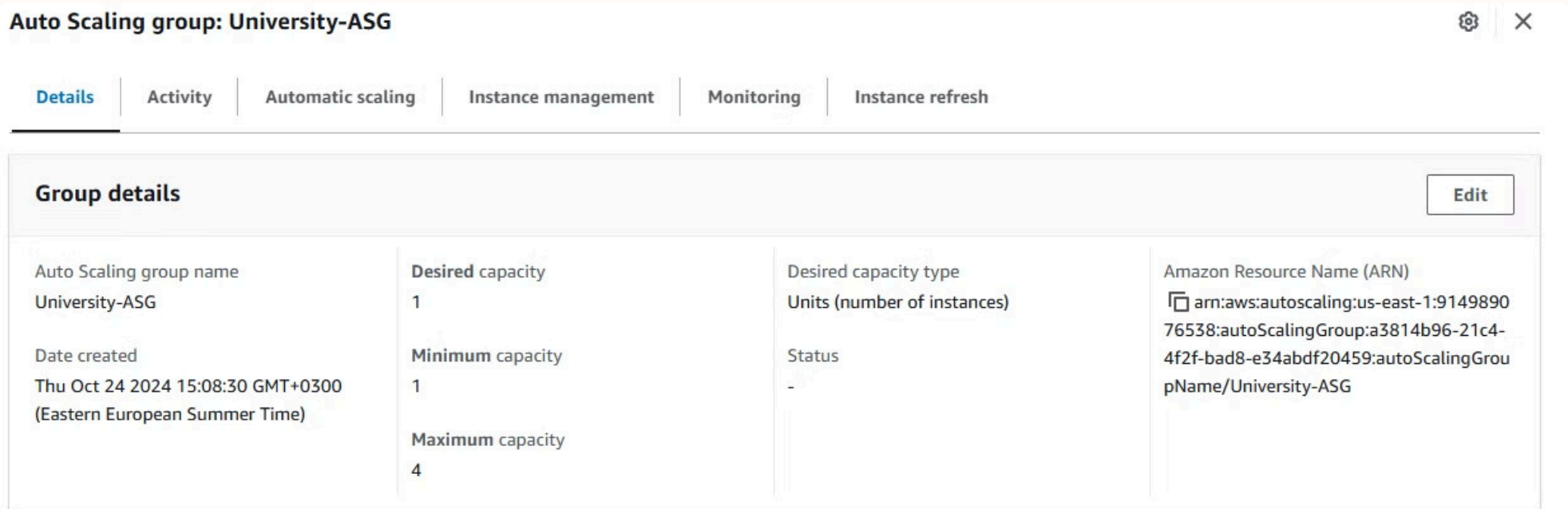
## Application Load Balancer

An Application Load Balancer (ALB) is created to distribute incoming traffic across multiple EC2 instances, ensuring that no single instance is overwhelmed during peak loads.



## Auto Scaling Group

To enable automatic scaling, an Auto Scaling group is set up with a launch template based on the web server configuration. The Auto Scaling group dynamically adjusts the number of instances in response to CPU utilization.



## Testing and Load Balancing

The application is accessed via the load balancer URL, and basic operations are performed to ensure proper functionality. Using AWS Cloud9, a load test is performed to simulate peak traffic conditions, and the scaling behavior is monitored.

```
voclabs:~/environment $ loadtest --rps 1000 -c 500 -k http://University-LB-406359779.us-east-1.elb.amazonaws.com
Requests: 4998, requests per second: 1000, mean latency: 9.4 ms

Target URL:      http://University-LB-406359779.us-east-1.elb.amazonaws.com
Max time (s):    10
Target rps:      1000
Concurrent clients: 44
Agent:           keepalive

Completed requests: 9999
Total errors:      0
Total time:        10.001 s
Mean latency:      6.4 ms
Effective rps:     1000

Percentage of requests served within a certain time
50%      2 ms
90%      17 ms
95%      30 ms
99%      49 ms
100%     307 ms (longest request)
voclabs:~/environment $
```

This output is a summary of the load test results. Here's what each part means:

- Requests: I sent 4,998 requests during the test period.
- Requests per second: My test aimed to generate 1,000 requests per second (rps), and that's the rate at which they were sent.
- Mean latency: The average time taken to receive a response for a request was 9.4 milliseconds (ms).

### Breakdown of key sections:

- Target URL: The URL I tested, <http://University-LB-406359779.us-east-1.elb.amazonaws.com>.
- Max time (s): The load test ran for 10 seconds.
- Target rps: I aimed for 1,000 requests per second (as specified).
- Concurrent clients: The test used 44 concurrent clients (simultaneous connections).
- Agent: keepalive means that connections to the target server are reused rather than reopened for each request.

### Summary of the Test:

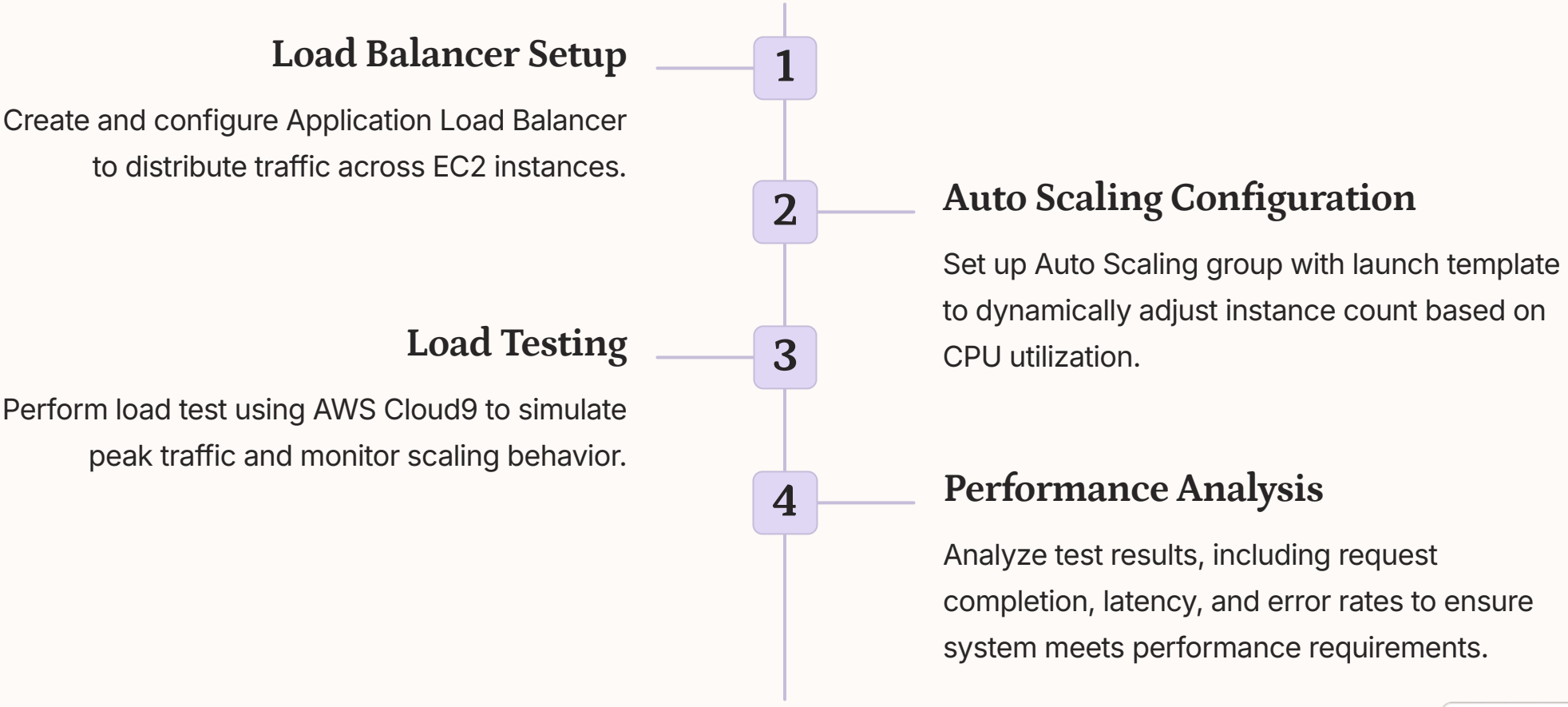
- Completed requests: The total number of requests completed was 9,999.
- Total errors: No errors occurred (total errors: 0).
- Total time: The entire test ran for approximately 10.001 seconds.
- Mean latency: The average latency (or delay) for requests was 6.4 ms.
- Effective rps: I maintained an effective rate of 1,000 requests per second, as intended.

### Latency Distribution:

- 50% of requests: 50% of the requests were served within 2 milliseconds.
- 90% of requests: 90% were served within 17 milliseconds.
- 95% of requests: 95% were served within 30 milliseconds.
- 99% of requests: 99% were served within 49 milliseconds.
- 100% of requests: The longest request took 307 milliseconds (meaning that the slowest request out of all took 307 ms).

### Overall:

The test was successful, as all requests were served without errors. The latency was low, with most requests being served in under 30 milliseconds, indicating good performance. The longest request (307 ms) was an outlier but still within acceptable limits for many applications.



# Results


By following the outlined approach, the student records web application is successfully deployed on AWS with the following outcomes:

- High Availability: The application remains available even if one or more servers fail.
- Scalability: The infrastructure automatically scales based on traffic, maintaining performance during peak periods.
- Cost Optimization: Resources are dynamically allocated and deallocated based on demand, preventing unnecessary costs.
- Security: The database is secured in private subnets, and secrets management is used for database credentials.

Final View Of The Functional Webapp. Using the Load Balancer DNS To Access it.

URL: University-LB-406359779.us-east-1.elb.amazonaws.com

Not secure university-lb-406359779.us-east-1.elb.amazonaws.com/students



# XYZ University

[Home](#)[Students list](#)

## All students

Name	Address	City	State	Email	Phone	
Ahmad Nasr	Helwan	Cairo	Egypt	example1@gmai.com	0123456789	<a href="#">edit</a>
Ali Adel	15 May City	Alexandria	Egypt	Ali12@gmail.com	01232165497	<a href="#">edit</a>
Mohamed Ashraf	12 Baker Street	London	United Kingdom	Ashraf@gmail.com	0321654987	<a href="#">edit</a>
The King Ahmad Elhamy	Nasr City	Cairo	The Great State of Egypt	Elhamy@gmail.com	0123456789	<a href="#">edit</a>

Add a new student

## High Availability

The application remains available even if one or more servers fail, ensuring continuous service for users.

## Scalability

Infrastructure automatically scales based on traffic, maintaining performance during peak periods like admissions season.

## Cost Optimization

Resources are dynamically allocated and deallocated based on demand, preventing unnecessary costs during low-traffic periods.

## Security

The database is secured in private subnets, and secrets management is used for database credentials, enhancing overall system security.

# Conclusion

This project demonstrates the power of AWS in building highly available, scalable, and secure applications. By implementing AWS services such as EC2, RDS, ALB, Auto Scaling, and VPC, we have transformed a single-server web application into a robust, cloud-based solution capable of handling thousands of users during peak periods. The solution also adheres to the best practices outlined in the AWS Well-Architected Framework, ensuring a high level of reliability, performance, and cost-efficiency.

## Special Thanks

I would like to extend my heartfelt gratitude to my professor, Ahmad Elhamy, whose guidance and support have been instrumental in the successful completion of this project. His dedication to teaching and commitment to helping his students excel has been a source of inspiration throughout my journey.

Thank you, Professor Elhamy, for your invaluable insights, patience, and encouragement, which have helped me grow both academically and professionally. I am truly grateful for all the knowledge and wisdom you have shared with me.

Ahmad Nasr Ali, AWS Cloud Specialist



### Cloud Expertise

Gained practical experience in designing and implementing cloud-based solutions using AWS services.



### Scalability Skills

Learned to create systems that can automatically adjust to varying workloads, ensuring consistent performance.



### Security Knowledge

Developed understanding of best practices in cloud security, including network isolation and secrets management.



### Performance Optimization

Acquired skills in load testing and performance tuning for high-traffic web applications.