Name : Ziad Mohamed Ismail Abdelkhalik.

Gmai : ziadmohamd333@gmail.com.

Phone : 01152259562.

Group_Code : GIZ1_SWD8_M1e.

New horizons cairo.

Scenario:

The university is preparing for the new school year. The admissions department has received complaints that their web application for student records is slow or unavailable during peak admissions periods due to high traffic.

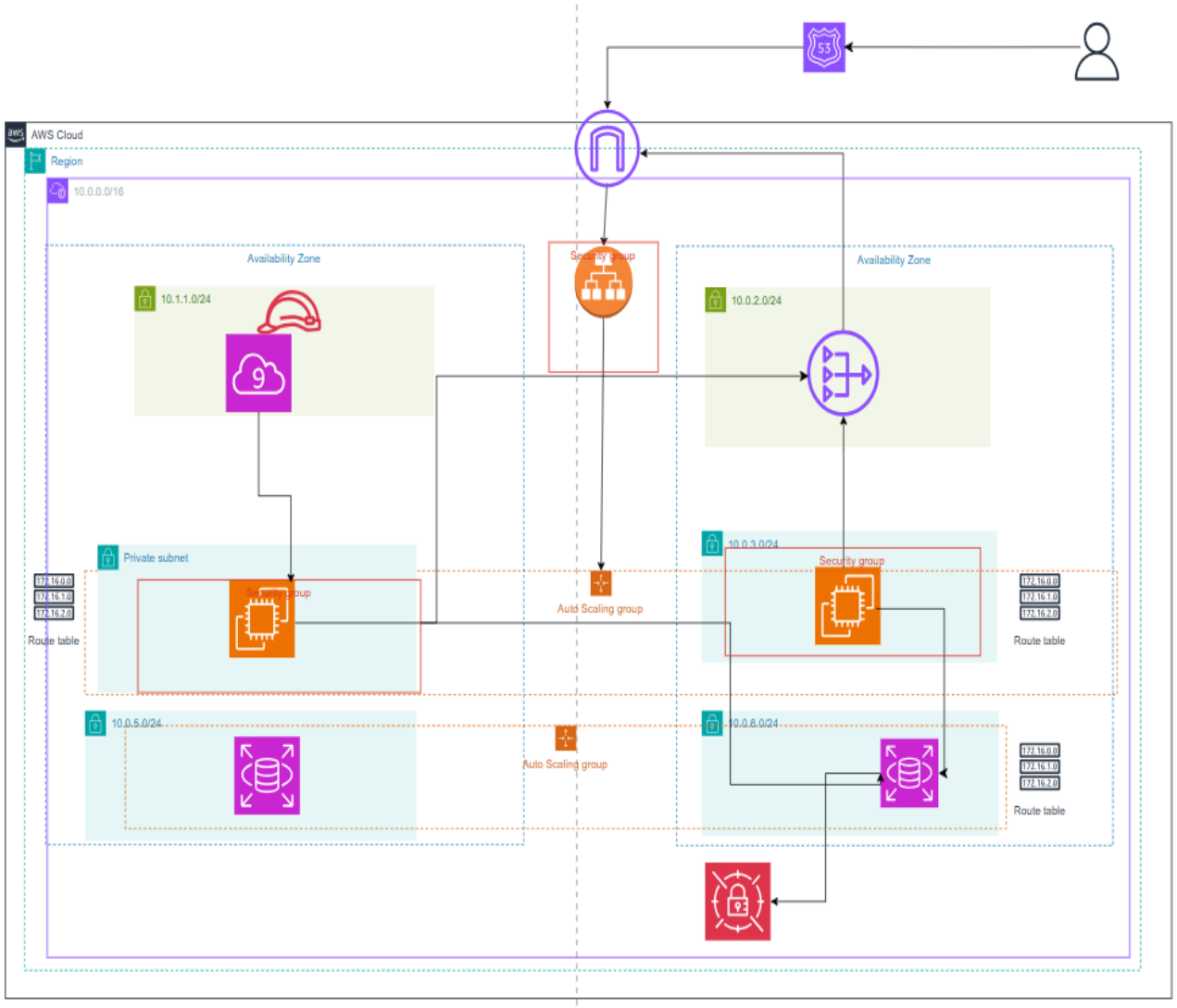
As a cloud engineer, your manager has asked you to create a proof of concept (POC) to host the web application in the AWS Cloud. You are tasked with designing and implementing a new hosting architecture that will improve the user experience for the web application. Your responsibilities include building the infrastructure to host the student records web application in the cloud.

Your challenge is to plan, design, build, and deploy the application to the AWS Cloud in accordance with AWS Well-Architected Framework best practices. During peak admissions periods, the application must support thousands of users while being highly available, scalable, load-balanced, secure, and high-performing.

The following image shows an example of the student records web application. The site lists records of students who have applied for admission to the university. Users can view, add, delete, and modify student records.

To achieve this, I used the following AWS services:

- **EC2 instances** to deploy the application, with an **Application Load Balancer** and **Auto Scaling Group** to ensure high availability and scalability.
- **Security groups** to provide robust security for the application.
- **Amazon RDS** to store student data, as it is a relational database well-suited for this purpose.
- **Cloud9** to develop and implement the code in my environment.
- **Internet Gateway** to ensure the application is accessible over the internet.



- Functional: The solution meets the functional requirements, such as the ability to view, add, delete, or modify the student records, without any perceivable delay.

Application load balancer: The solution can properly balance user traffic to avoid overloaded or underutilized resources. We use **Application Load Balancing** to distribute incoming application traffic across multiple targets, such as EC2 instances, in various Availability Zones. This increases the availability of your application.

You add one or more listeners to your load balancer. A listener checks for connection requests from clients, using the protocol and port that you configure. The rules you define for a listener determine how the load balancer routes requests to its registered targets. Each rule consists of a priority, one or more actions, and one or more conditions. When the conditions for a rule are met, the corresponding actions are performed.

You must define a default rule for each listener, and you can optionally define additional rules.

Scalable: We use an **Auto Scaling group** to achieve high scalability in our web servers. An Auto Scaling group contains a collection of EC2 instances that are treated as a logical grouping for automatic scaling and management purposes. It also allows the use of Amazon EC2 Auto Scaling features, such as health check replacements and scaling policies.

Both maintaining the number of instances in the Auto Scaling group and automatic scaling are core functionalities of the Amazon EC2 Auto Scaling service. The size of an Auto Scaling group depends on the number of instances you set as the desired capacity. You can adjust this capacity to meet demand, either manually or through automatic scaling.

The Auto Scaling group starts by launching enough instances to meet the desired capacity and maintains this number of instances by performing periodic health checks on the group's instances.

Highly available: We use **Multi-Availability Zones (AZs)** to achieve high availability. AZs are isolated locations within a region. Deploying your

infrastructure across multiple AZs helps ensure that your application remains operational if one AZ experiences an outage.

We use an **Auto Scaling group** to achieve high scalability for our web servers, and **Application Load Balancing** to distribute incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones.

We use **RDS Multi-AZ** to ensure that your database has a standby replica in a different AZ. In case of a failure, automatic failover occurs to the standby instance. For read-heavy applications, you can create read replicas of your database in multiple AZs or even across regions.

Security : The database is secured and cannot be accessed directly from public networks. I placed the **RDS DB** in a private subnet and configured the security group's inbound rules to accept **traffic only from the security group of the EC2 instance**. I use **security groups** to ensure high security for my instance.

AWS IAM roles allow you to manage permissions securely by controlling access to AWS resources. We also apply data encryption to encrypt data at rest. Encryption is enabled for databases and storage volumes, and **AWS Key Management Service (KMS)** provides centralized key management.

We use **VPC** to isolate our network and resources from the public internet. **Secrets Manager** secures sensitive data such as passwords, phone numbers, usernames, database credentials, and more.

High performing: You need to focus on optimizing various aspects of your infrastructure, application design, and resource management. AWS offers a variety of services and tools to help you build scalable, responsive, and high-performing applications. Here's a comprehensive guide on how to optimize for high performance:

- **Choose the right EC2 instance types** for your workload.
- I manage **EC2 instance types, AMIs, storage, networking, security groups**, and **key pairs** to ensure optimal performance.

Cost optimized : choose the right size EC2 instance to meet my requirements, selecting the best fit for the workload (compute-optimized, memory-optimized, or storage-optimized). Use **AWS Cost Explorer's Resource**

Optimization to identify underutilized instances that can be downsized or terminated.

- **Auto Scaling**: Use Auto Scaling Groups to automatically adjust the number of EC2 instances in response to traffic or load changes. This ensures that you're only paying for the resources you need.
- **Reserved Instances**: I can use reserved instances to save up to 80% of the total cost, which can be 70-90% cheaper than On-Demand instances.
- Ensure your load balancers are efficiently sized and not over-provisioned.
- Use **On-Demand Instances** only when workloads are unpredictable or need short-term scaling.
- Use managed services like **Amazon RDS** that automatically handle operational overhead (patching, scaling, backups), allowing you to focus on cost-efficient use cases.
- Choose the right **database engine** and instance type (On-Demand or Reserved). Scale read-heavy workloads with **RDS Read Replicas** instead of launching larger instances.
- Use **RDS Auto Scaling** to dynamically scale up and down based on demand.
- Continuously monitor resource usage using **Cost Explorer** to track idle or underutilized resources, and set up custom cost and usage alerts.

AWS VPC (Virtual Private Cloud) is a foundational networking service within AWS that allows you to create and manage isolated, secure cloud networks for your AWS resources. It provides full control over your virtual networking environment, including the selection of your own IP address ranges, creation of subnets, routing configurations, and security settings

- **Subnets**:

  - A VPC can be divided into subnets (public and private), each residing within an Availability Zone.
  - **Public subnets** allow resources (like EC2 instances) to be accessible from the internet, whereas **private subnets** restrict access and are typically used for internal applications or databases.

- **Route Tables**:

- Define how traffic flows within the VPC. You can create custom routes to direct traffic between subnets or to external destinations like the internet or other VPCs.

- **Internet Gateway (IGW)**:
  - A VPC component that allows communication between instances in the VPC and the internet. It is typically associated with public subnets.

Pricing : **this services that I use and the price**

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
35.92 USD	**1,305.34 USD**	**15,700.00 USD** Includes upfront cost

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
AWS Web Application Firewall (WAF) **Status**: - **Description**: **Config summary**: Number of Web Access Control Lists (Web ACLs) utilized (2 per month)	No group applied	US East (N. Virginia)	0.00 USD	10.00 USD
Amazon Route 53 **Status**: - **Description**: **Config summary**: Hosted Zones (2), Number of Elastic Network Interfaces ()	No group applied	US East (N. Virginia)	0.00 USD	1.00 USD
Amazon EC2 **Status**: - **Description**: **Config summary**: Tenancy (Shared Instances), Operating system (Windows Server), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t3.nano), Pricing strategy (Compute Savings Plans 1yr Partial 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)	No group applied	US East (N. Virginia)	35.92 USD	2.99 USD

Amazon RDS Custom for Oracle	No group applied	US East (N. Virginia)	0.00 USD	419.57 USD

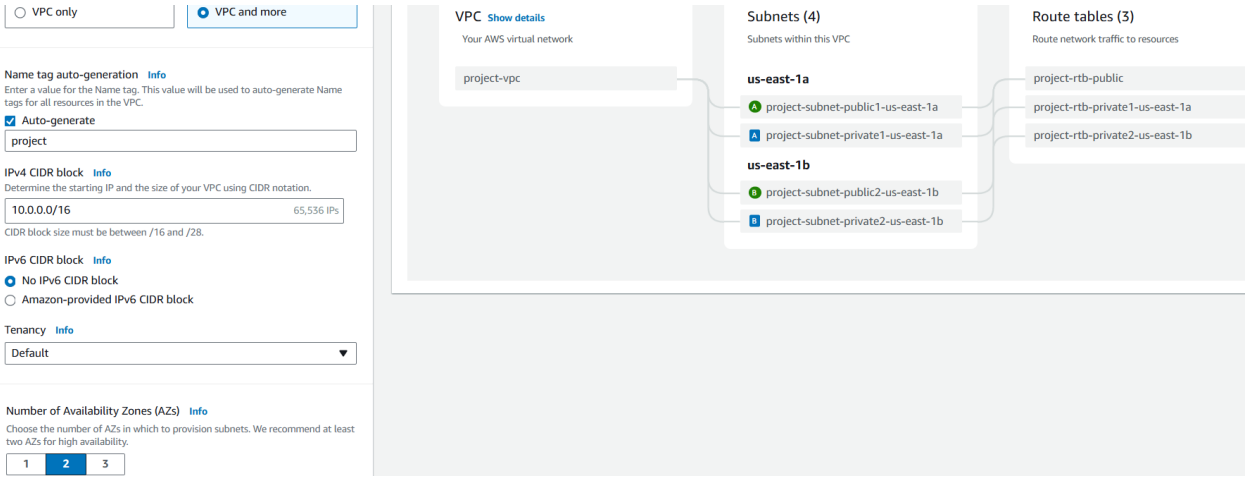
Status: -
Description:
Config summary: Storage for each RDS Custom instance (General Purpose SSD (gp2)), Storage amount (100 GB), Instance type (db.r5.xlarge), Number of RDS Custom for Oracle instances (1), Utilization (On-Demand only) (100 %Utilized/Month), License (Customer-provided), Database edition (Enterprise), Deployment option (Single-AZ)

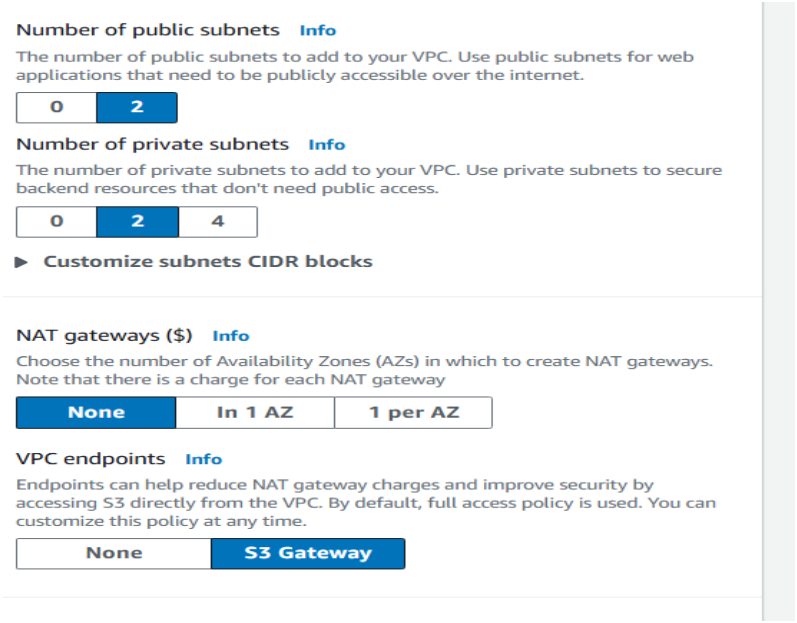
Amazon Virtual Private Cloud (VPC)	No group applied	US East (N. Virginia)	0.00 USD	871.78 USD

Status: -
Description:
Config summary: Working days per month (22) Number of NAT Gateways (1) Number of Availability Zones that Gateway Load Balancer is deployed to (1), Number of Gateway Load Balancer Endpoints (1), Total processed bytes (100 GB per hour)

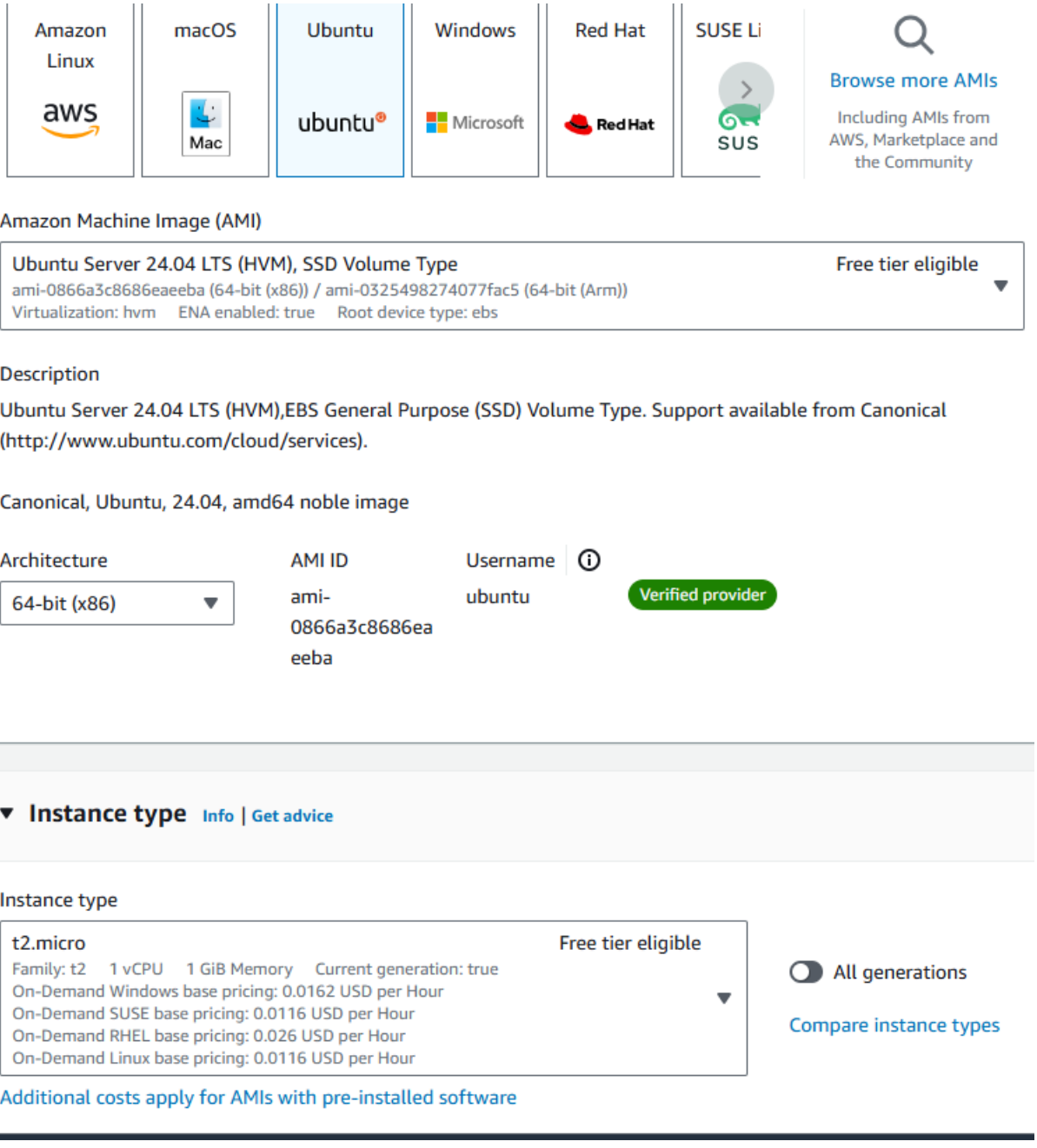
VPC :

Is a foundational networking service within AWS that allows you to create and manage isolated, secure cloud networks for your AWS resources.

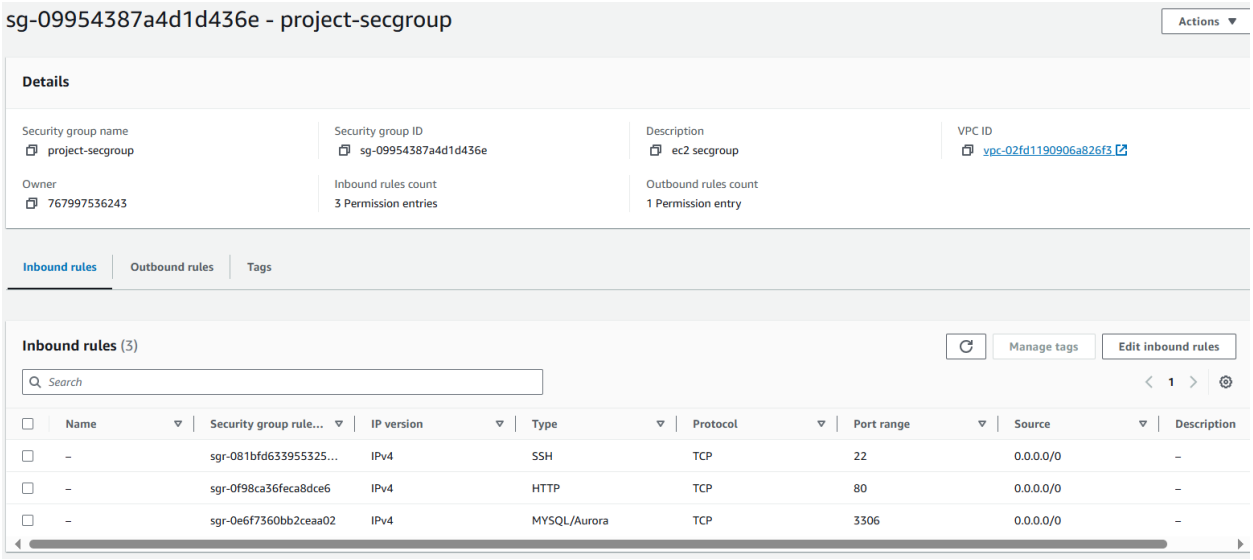




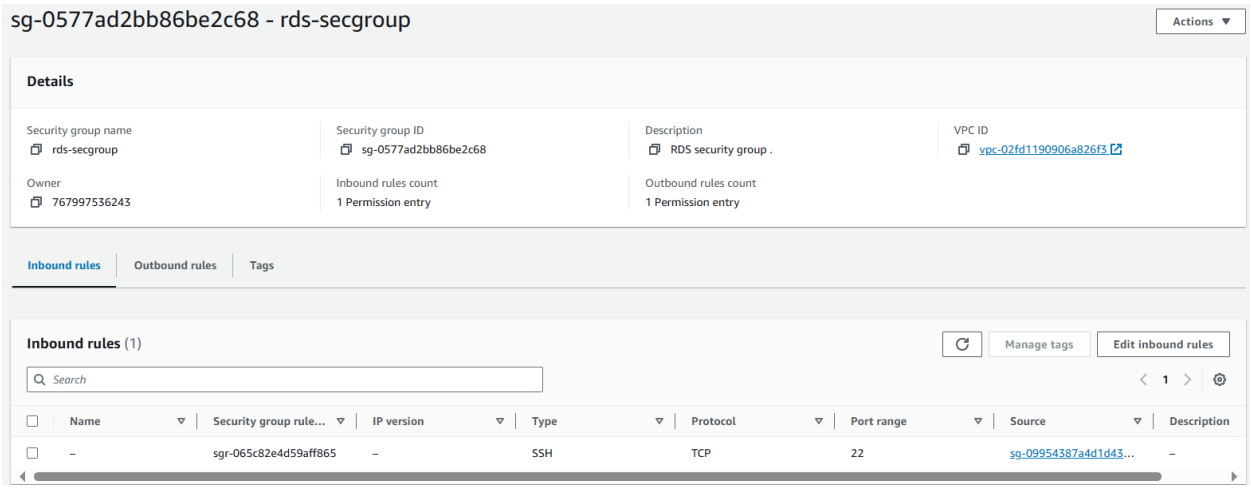
Ec2 Instance : is a web service that provides resizable compute capacity in the cloud. It allows you to run virtual servers, known as EC2 instances



Security group of Instance : I use **security groups** to ensure high security for my instance , in this case I use **SSH** to SSH is commonly used to securely connect to and manage your EC2 instances. By allowing inbound traffic on port 22, you enable secure remote access to your Instances , I use **MySQL/Aurora** is a popular relational database management system, and it typically uses port 3306 for communication.

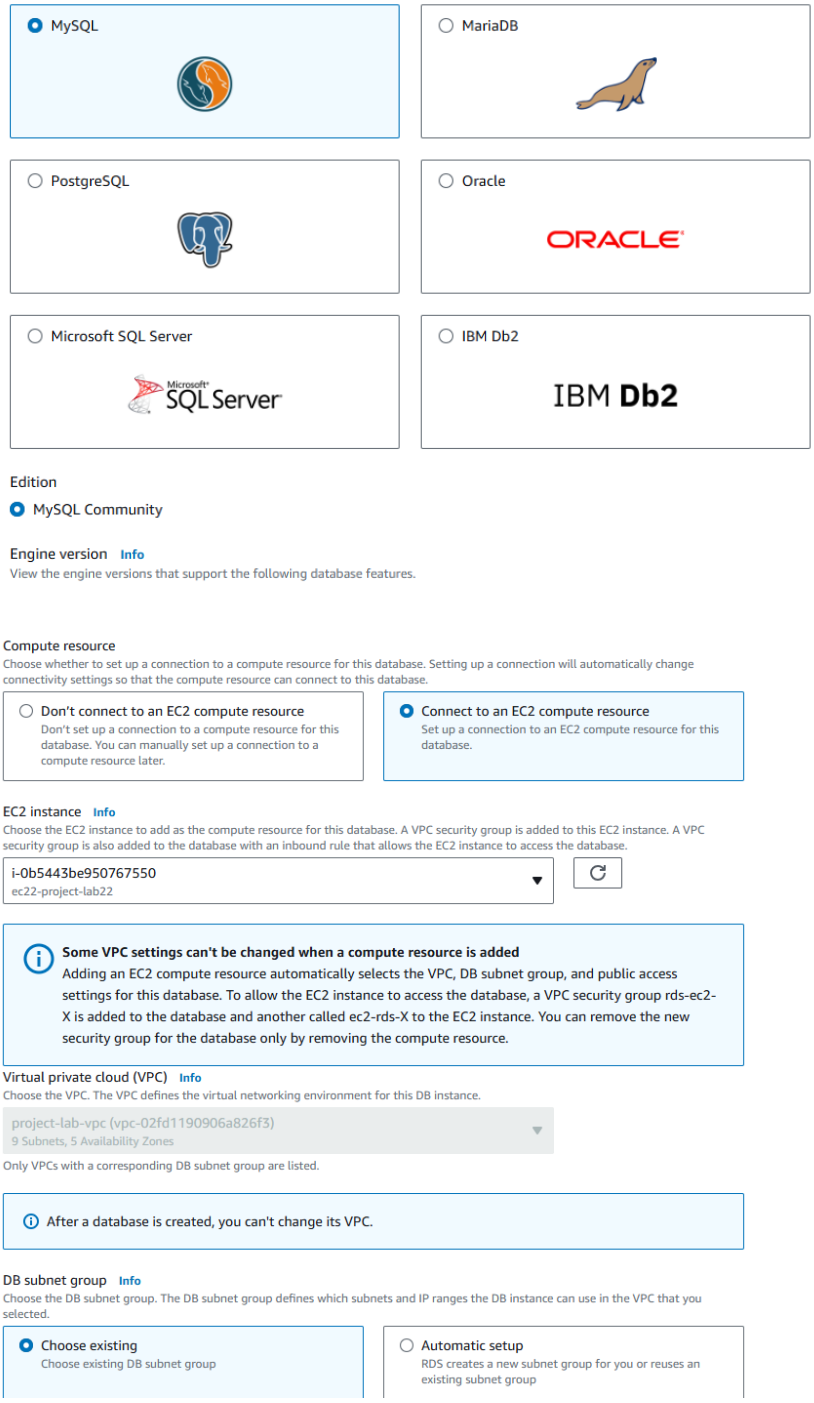


Security group of RDS : **in the inbound I identify the source is the security group of the ec2 instance to insure no request connect RDS unless the ec2 instance**



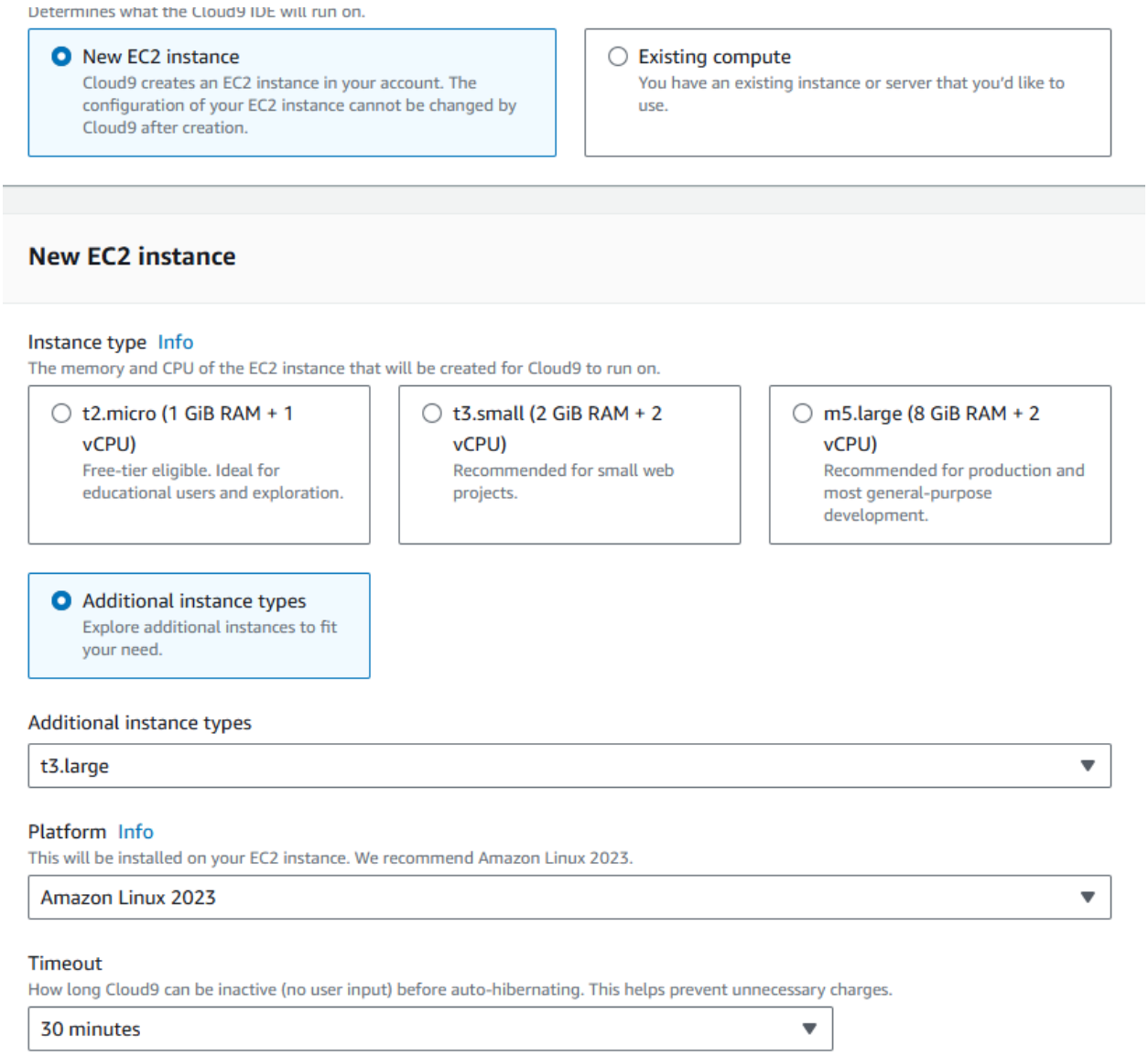
RDS :

Is a managed database service provided by AWS that simplifies the setup, operation, and scaling of relational databases in the cloud. RDS supports several database engines,



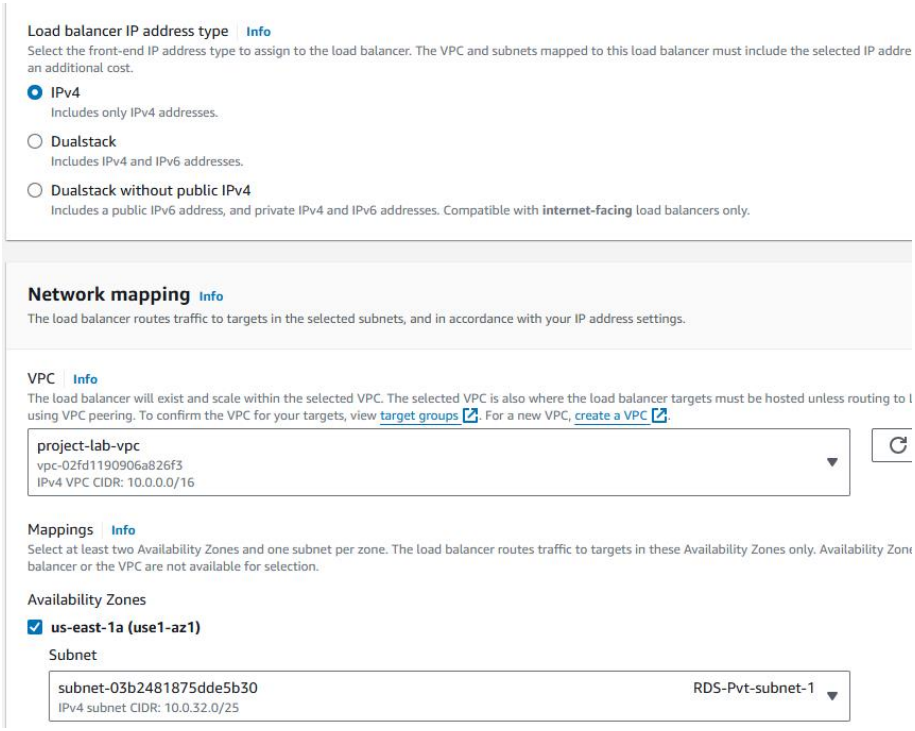
Cloud9 :

Is a cloud-based integrated development environment (IDE) that allows you to write, run, and debug your code using just a web browser. It provides a rich set of tools and features to streamline the development process, making it easier for developers to collaborate and manage their applications in the cloud. Here's a detailed overview of AWS Cloud9, including its features, benefits, and use cases.



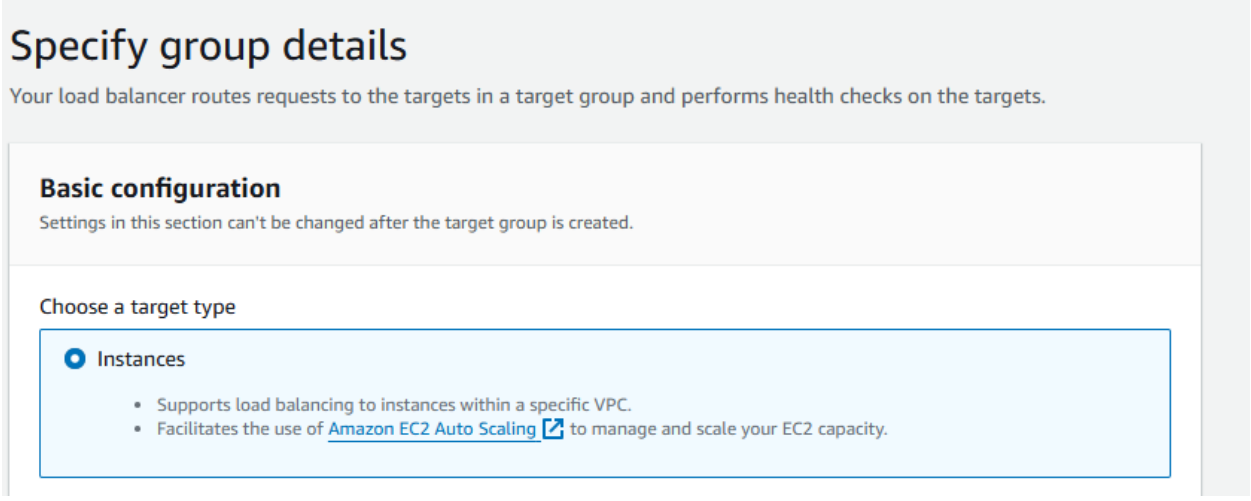
Application load balancer :

Is a service that automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances

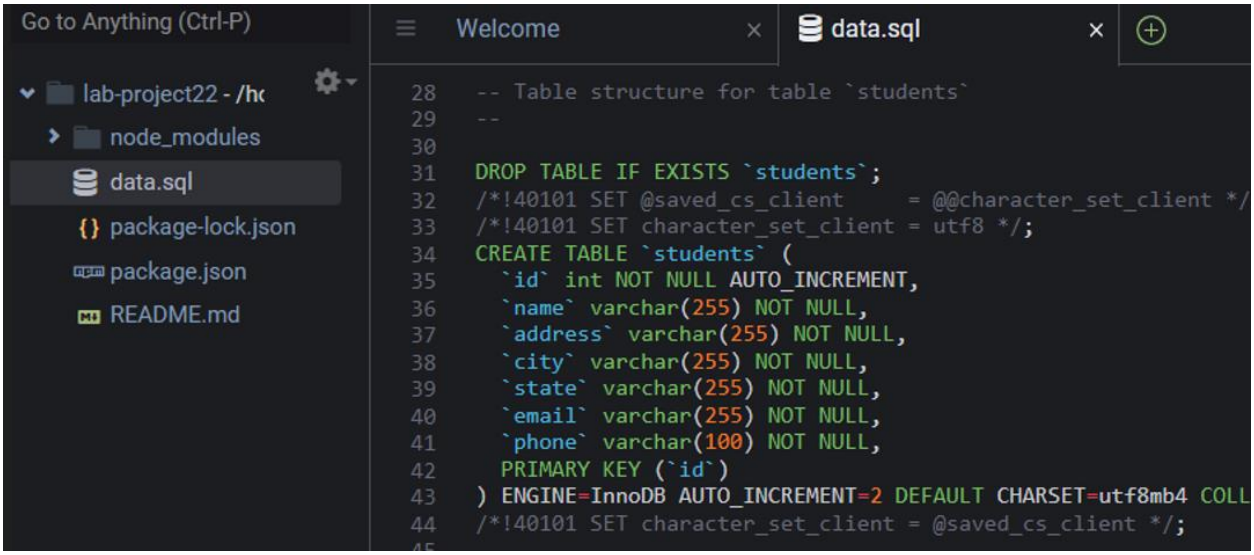


Target Group :

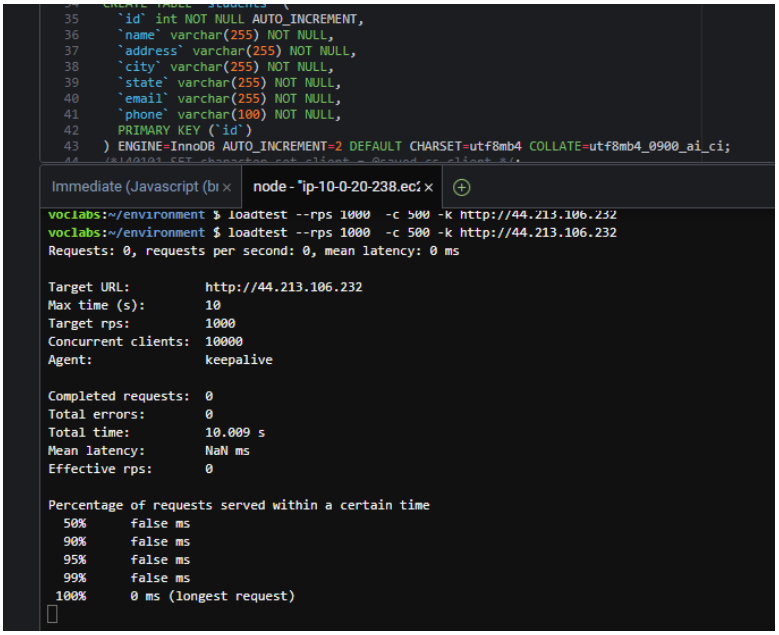
Is a logical grouping of resources (such as EC2 instances, IP addresses, or AWS Lambda functions) that you want the load balancer to route traffic to.



DB authentication with CLOUD9 IDE :



Cloud9 with load test :



Finally, **I tested the output of the application and inserted my data into the application.** :