**Cloud Web Application Builder Project**

**Project Overview**

**Objective**

The goal of this project was to develop a cloud-based web application for managing student records at Example University. This initiative was prompted by performance issues experienced during peak admissions periods, necessitating a robust solution capable of handling high traffic volumes efficiently.

**Challenges**

As a cloud engineer, key challenges included:

* Designing an architecture that supports high availability and scalability.
* Ensuring secure access to sensitive data.
* Implementing load balancing to manage user traffic effectively.
* Maintaining optimal performance under varying loads.

**Solution Architecture**

**Architectural Diagram**

The following diagram illustrates the architecture of the proposed solution:

A screenshot of a computer screen

Description automatically generated  
*Figure 1: Architectural Diagram of AWS Cloud Infrastructure*

**Key Components**

1. **VPC (Virtual Private Cloud)**: A logically isolated section of AWS where resources are deployed.
2. **Subnets**:
   * **Public Subnets**: For resources accessible from the internet (e.g., web servers).
   * **Private Subnets**: For resources that should not be directly accessible from the internet (e.g., databases).
3. **EC2 Instances**: Virtual servers hosting the web application, using **Ubuntu** as the AMI.
4. **RDS (Relational Database Service)**: Managed database service for storing student records, configured with a **db.t3. micro** instance type.
5. **Load Balancer**: Distributes incoming application traffic across multiple EC2 instances.
6. **Auto Scaling Group**: Automatically adjusts the number of running EC2 instances based on demand.
7. **AWS Secrets Manager**: Used to securely manage database credentials.

**Implementation Phases**

**Phase 1: Design and Cost Estimation**

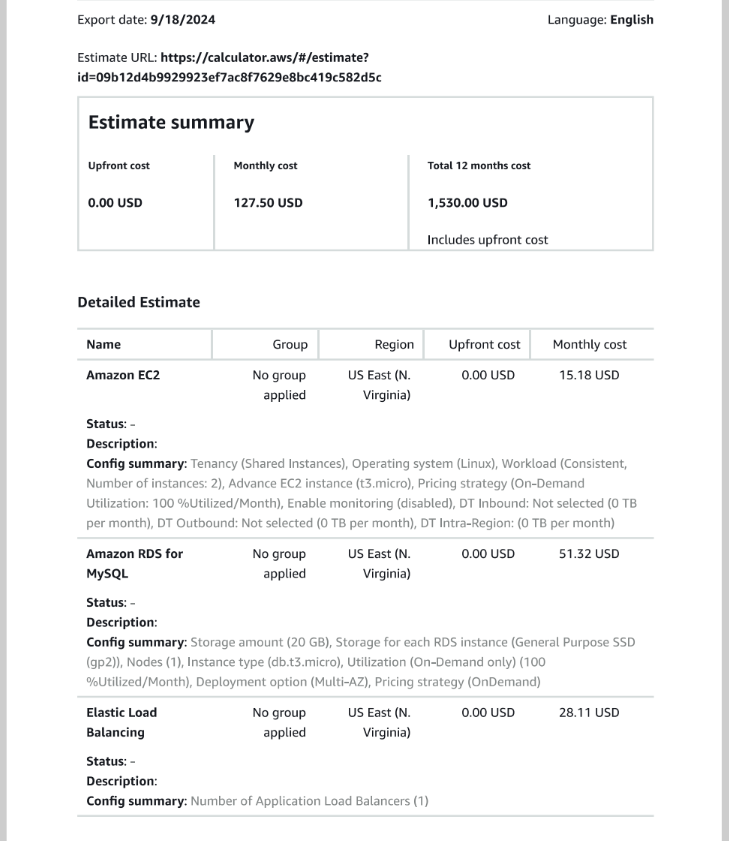
**Task 1: Architectural Design**

* **Create a VPC**: Set up a VPC with an IPv4 CIDR block of 10.0.0.0/16.
* **Define Subnets**:
  + **Public Subnets**: Two public subnets (10.0.1.0/24 and 10.0.2.0/24) for the web server testing.
  + **Private Subnets**: Two private subnets (10.0.3.0/24 and 10.0.4.0/24) for RDS and web servers.

**Task 2: Cost Estimation**

* **AWS Pricing Calculator**: Used to estimate costs for the following services:
  + **EC2 Instances**: Based on instance types and expected usage.
  + **RDS**: Configured with db.t3.micro, estimating costs based on the instance type and storage.
  + **Elastic Load Balancer**: Costs associated with traffic distribution.
  + **NAT Gateway**: Pricing for NAT usage to allow private subnet instances to access the internet.

A screenshot of a computer

Description automatically generated

**Phase 2: Basic Web Application Setup**

**Task 1: Creating the VPC**

1. **Navigate to the VPC Dashboard** in the AWS Management Console.
2. **Create a New VPC** with the CIDR block 10.0.0.0/16.
3. **Create Subnets**: Define two public and two private subnets as specified.

**Task 2: Launching EC2 Instances**

1. **Select an AMI**: Choose the latest **Ubuntu** AMI.
2. **Configure Instance Details**:
   * Select the VPC and appropriate subnet.
   * Enable Auto-assign Public IP for instances in public subnets.
   * Add the user data for the application.
3. **Configure Security Groups**:
   * A screenshot of a computer

     Description automatically generatedCreate a security group allowing HTTP (port 80) and SSH (port 22) access for the web server.

**Task 3: Testing the Deployment**

* Access the public IP address of the EC2 instance to ensure the web application is accessible.

A screen shot of a computer

Description automatically generated

**Phase 3: Database and Environment Configuration**

**Task 1: Configuring Amazon RDS**

1. **Create an RDS Instance**:
   * Choose MySQL as the database engine.
   * Set the instance type to **db.t3.micro** and configure storage options.
   * Place the database in a private subnet for security.
2. **Configure Security Groups**: Ensure the security group for the RDS allows inbound traffic on port 3306 from the web application security group.

A screenshot of a computer

Description automatically generated

**Task 2: Setting Up the Development Environment**

* **Provision AWS Cloud9**:
  + Create a Cloud9 environment using an EC2 instance.
  + Install necessary tools (e.g., AWS CLI, MySQL client).

**Phase 4: High Availability and Scalability**

**Task 1: Setting Up Load Balancer**

1. **Create an Application Load Balancer**:
   * Choose the VPC and subnets for the load balancer.
   * Configure listeners to route traffic to the EC2 instances.
   * Associate the security group that allows HTTP traffic from everywhere.

**Task 2: Configuring Auto Scaling**

1. **Create an Auto Scaling Group**:
   * Use a launch template based on the EC2 instance configuration.
   * Specify the IAM role **LabRole** for the instances to access AWS resources securely.
   * Add the user data for the application
   * Set minimum and maximum instance counts based on expected traffic.
   * Enable scaling policies to adjust the number of instances automatically.

A screen shot of a computer

Description automatically generated

**Task 4: Load Testing the Application**

* Use a tool like **load test** to simulate traffic and evaluate performance:
  + Execute the command to generate load: loadtest --rps 1000 -c 500 <ELB URL>
  + Monitor application behavior and response times.
  + A screenshot of a computer

    Description automatically generatedYou will find other instances initialized

**Conclusion**

The project successfully established a cloud-based web application using AWS services. The architecture ensures high availability and scalability while maintaining security and performance.

The integration of services like VPC, EC2, RDS, load balancers, and AWS Secrets Manager creates a resilient environment capable of handling peak traffic efficiently. This foundational work sets the stage for future enhancements and projects within AWS.