DALHOUSIE UNIVERSITY

CSCI 5411

Adv. Cloud Architecting

Term Project Report

Name: Jay Sanjaybhai Patel
Banner ID: B00982253

Table of Contents

Project

RoomM8 Link: https://github.com/Jay-Kumar-Patel/MACS/tree/main/ASDC/Project

Why?

Technology Stack

- **Frontend:** Next.js
- **Backend:** Spring Boot
- **Database:** MySQL

About Project

RoomM8 is a web-based application designed to enhance the living experience for roommates by streamlining communication, task management, and grocery planning. The system operates in a client-server environment where the backend is developed using Spring Boot and the frontend is built with Next.js. The primary users of the system are roommates who need to coordinate daily activities, manage shared tasks, and communicate effectively. Building the application with a cloud architecture allows us to ensure scalability, high availability, and cost optimization, providing a reliable and responsive experience as the user base grows and traffic fluctuates.

Chosen AWS Services :

AWS services by category :

Category	**Services**
Compute	EC2, EC2 Auto Scaling, Lambda
Networking and Content Delivery	VPC, API Gateway, Elastic Load Balancing
Database	RDS
Application Integration	SNS
Management and Governance	CloudFormation, CloudWatch
Security, identity, and compliance	Secrets Manager

➢ Amazon EC2 (Elastic Compute Cloud)

Usage: EC2 instances are used to host both the frontend (Next.js) and backend (Spring Boot) of the roomM8 application. The frontend is deployed in a public subnet and the backend is deployed in a private subnet, with instances in multiple availability zones to ensure high availability.

Scalability: EC2 Auto Scaling automatically adjusts the number of backend instances based on traffic. For example, during times of heavy user interaction (e.g., during room searches or profile updates), the system can scale up by adding more instances.

Cost: EC2's pay-as-you-go pricing model allows for flexible cost management. For predictable usage, Reserved Instances can be leveraged to lower costs by up to 72%, which is beneficial for the long-term hosting of the app. Spot Instances could also be used for non-critical workloads, providing additional cost savings.

Best Fit: EC2 is ideal for hosting both the frontend and backend of roomM8, as it allows for full control over the operating system, instance types, and configurations. The ability to scale automatically based on traffic ensures the app remains responsive while minimizing costs. For frontend, a single EC2 instance in us-east-1a provides a stable environment, with instance type t2.medium.

➢ EC2 Auto Scaling

Usage: EC2 Auto Scaling is used to automatically manage the number of backend instances running, ensuring that roomM8 can handle varying levels of user activity. The system scales up or down based on predefined metrics, like CPU usage or request count.

Scalability: Auto Scaling ensures the backend can handle unexpected spikes in demand, such as during high-traffic periods when users are actively searching for rooms or messaging other users.

Cost: By automatically scaling the backend based on demand, Auto Scaling reduces the need for over-provisioning, ensuring cost-effective resource allocation. The ability to automatically scale down during low-demand periods helps to lower EC2 costs by eliminating underutilized instances.

➢ Lambda

Usage: AWS Lambda is used in roomM8 for handling specific serverless functions, such as sending notifications when a new user signs up or when an already existing user logIn to this app. Lambda allows you to run backend code without provisioning or managing servers, making it ideal for lightweight tasks.

Scalability: Lambda automatically scales to handle varying volumes of requests. As traffic increases (e.g., more users sign up or logIn with the app), Lambda can scale seamlessly to process multiple functions concurrently.

Cost: Lambda operates on a pay-as-you-go model based on the number of requests and the execution time of your functions. This makes it cost-effective, as you only pay for the compute time used by your functions and not for idle server capacity.

Best Fit: In roomM8, Lambda functions are triggered by user actions or events, such as sending a welcome email or a notification when a user logged In. These serverless functions ensure that these tasks are handled efficiently without impacting the performance of the main application, and they scale automatically as the user base grows.

➢ VPC

Usage: The roomM8 application is deployed within a VPC to ensure secure, isolated networking between frontend (Next.js), backend (Spring Boot), and database (MySQL RDS). Public and private subnets are configured for optimized security and availability.

- Public Subnet: Hosts the frontend EC2 instances (Next.js), allowing them to be accessible to users and communicate with the backend via API Gateway.
- Private Subnet: Hosts the backend EC2 instances (Spring Boot) and the MySQL RDS database, ensuring these components are not directly accessible from the internet, providing an extra layer of security.

Scalability: The VPC can scale by adding more subnets or resources as roomM8 grows. New services can be added to the VPC without compromising security or performance. It provides flexibility to grow the infrastructure by simply adding more instances to the private or public subnets.

Cost: There are no additional costs for the VPC itself, but the resources within the VPC (e.g., EC2 instances, RDS databases) incur costs. By using a private subnet for the backend and database, the VPC helps ensure secure, cost-effective networking.

Best Fit: The architecture consists of 6 subnets: 2 public subnets (for frontend) and 4 private subnets (for backend and database). Public subnets are able to access the internet through an internet gateway, while private subnets rely on NAT Gateways to allow internet access when needed (e.g., for updates or external API calls), ensuring security and control. It provides full control over networking, is highly scalable, and supports a multi-tier architecture for security and growth.

➢ API Gateway

Usage: API Gateway is used to handle HTTP requests from users, forwarding them to the Spring Boot backend for processing. It acts as an intermediary between the frontend (Next.js) and backend (Spring Boot).

Scalability: API Gateway scales automatically to handle increases in traffic, allowing roomM8 to maintain responsiveness even during peak traffic times (e.g., multiple users sending messages). It also supports rate limiting and throttling, which prevents excessive API requests from overwhelming the backend services.

Cost: API Gateway follows a pay-as-you-go pricing model, which means you pay only for the number of requests and the data transferred.

Best Fit: API Gateway is ideal for roomM8 because it allows you to efficiently manage API calls between the Next.js frontend and Spring Boot backend. It ensures secure, high-performance routing of user requests and simplifies the process of managing REST APIs. The ability to integrate with AWS Lambda also provides additional flexibility for serverless features like notifications or background tasks.

➢ RDS

Usage: MySQL is used to store data in roomM8 application. Managed through Amazon RDS, it provides high availability and easy integration with the Spring Boot backend.

Cost: RDS uses a pay-as-you-go pricing model with no upfront costs. Costs depend on instance size, storage, and usage.

Performance: MySQL on RDS supports complex queries and transactions, ideal for roomM8, which requires relational data handling. RDS Multi-AZ deployments ensure high availability.

Security: RDS MySQL provides encryption at rest and in transit. The database is deployed in a private subnet for enhanced security.

Scalability: Easily scalable with read replicas and vertical scaling to meet increased demand as roomM8 grows.

Best Fit: RDS MySQL securely stores all critical data like user accounts, room information, and interactions, and scales seamlessly to accommodate growing user activity and data as the app expands.

➢ SNS

Usage: SNS is used to send notifications to users about important actions, such as when they sign up or login into this application.

Scalability: SNS automatically scales to handle large volumes of notifications, whether sent via email, SMS, or other channels.

Cost: SNS operates on a pay-per-use pricing model based on the number of messages sent. This makes it cost-effective, especially for applications that need to send notifications on an as-needed basis.

Best Fit: SNS is perfect for roomM8 as it can handle high volumes of real-time notifications, ensuring users are promptly informed about key actions. This enhances user engagement and overall experience.

➢ Cloud Watch

Usage: CloudWatch is used in roomM8 for monitoring and logging AWS resources and application performance, particularly the EC2 instances running the frontend (Next.js) and backend (Spring Boot).

Best Fit: CloudWatch provides detailed monitoring for AWS cloud resources and application performance, ensuring that roomM8 remains performant and issues are detected early.

- Customizable Alerts allow you to set up alarms based on specific thresholds (e.g., CPU usage, request latency), enabling proactive issue detection and quick response.
- It enables the collection and tracking of metrics, logs, and events, ensuring comprehensive visibility into application behavior and resource health.

How CloudWatch Helps in roomM8: CloudWatch helps monitor the health of roomM8's EC2 instances, track performance metrics (e.g., CPU utilization, disk I/O), and store logs for both the frontend and backend. This ensures that any issues, such as high latency or errors, are quickly identified and addressed, helping maintain a smooth user experience.

➢ Secret Manager

Usage: Secrets Manager is used in roomM8 to securely store critical information such as MySQL database credentials and connection URLs, preventing hard-coded secrets in the codebase.

Best Fit:

- Secrets Manager helps protect sensitive data, ensuring that credentials are securely managed and rotated automatically. This reduces the risk of exposure from hardcoded secrets in the Spring Boot backend.

- Automatic Rotation allows credentials to be automatically updated on specified dates, reducing manual management overhead.
- Encryption of secrets ensures that sensitive information is stored securely, adding an extra layer of protection for important data.

Architecture

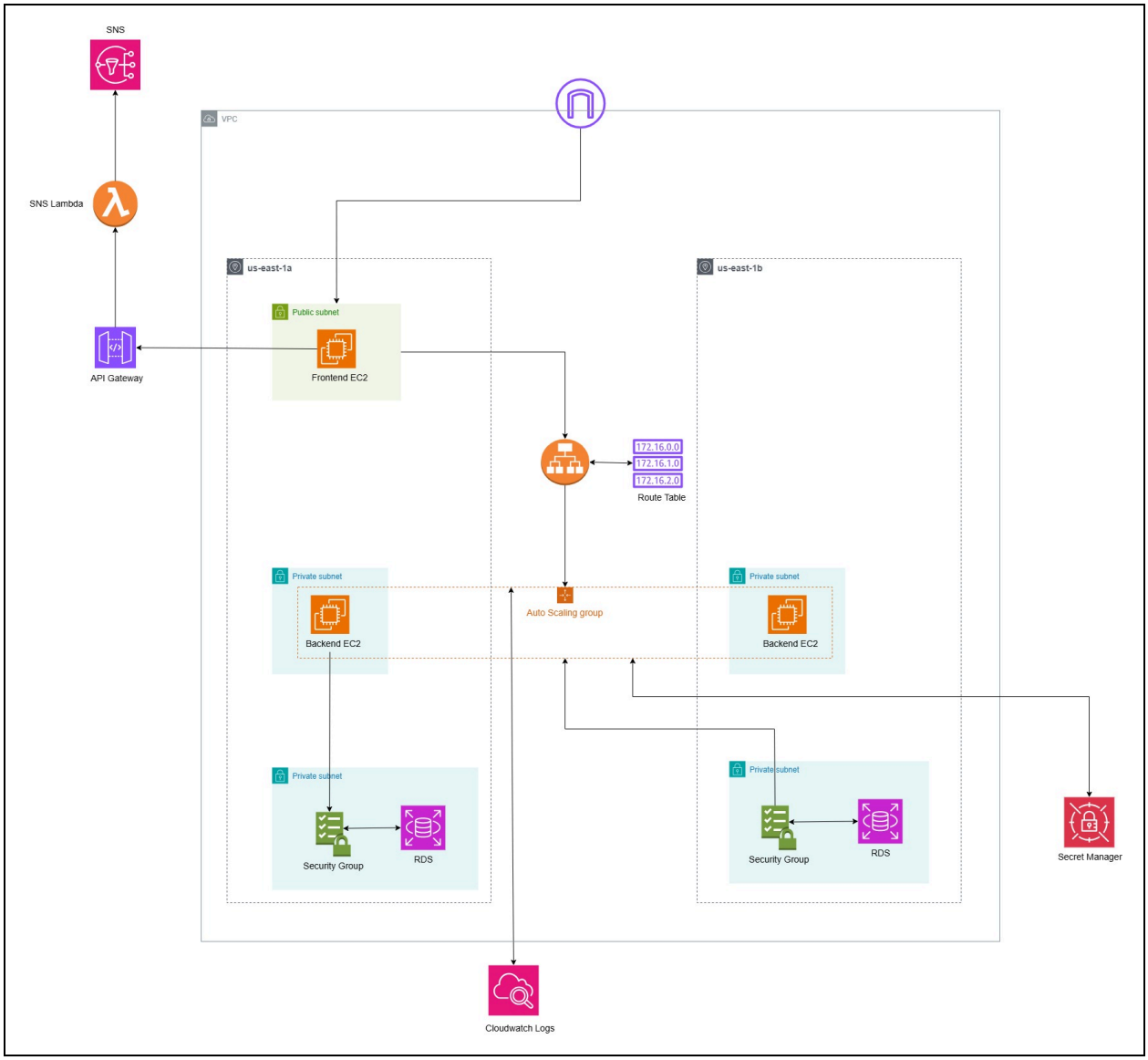

Figure 1: RoomM8 Architecture

Application Flow

VPC:

- When a user accesses the **roomM8** frontend, the request goes through the **VPC**, an isolated network. The **public subnet** allows access from the internet via the **Internet Gateway**, while the **private subnet** ensures secure backend communication.

Frontend EC2 (Next.js):

- The **EC2 instance** in the **public subnet** (us-east-1a) runs the **Next.js** frontend, which is accessible to users via the internet. It interacts with the backend through a load balancer and API Gateway for sending notifications.

API Gateway:

- **API Gateway** is used to trigger **email notifications** when certain events occur, like user registration or login. It routes requests from the frontend to **AWS Lambda**, which handles the email sending via **Amazon SNS**.

Backend EC2 (Spring Boot):

- The **EC2 instances** in the **private subnet** run the **Spring Boot** backend, which handles core business logic, and database connections.

Load Balancer:

- The **Elastic Load Balancer (ELB)** distributes incoming traffic between the backend EC2 instances in the Auto Scaling group, ensuring availability and even load distribution.

RDS MySQL (Database):

- The **MySQL database** is hosted in **RDS** in a **private subnet**, storing all critical data such as user profiles, room listings, and messages. **RDS** supports high availability and vertical scaling via read replicas.

CloudWatch:

- **CloudWatch** monitors the health of **EC2 instances**, and tracks logs which helps in the continuous tracking of application health.

Secrets Manager:

- **AWS Secrets Manager** is used to securely store sensitive credentials like database access details, ensuring they are encrypted and managed without hardcoding them into the backend code.

Auto Scaling Group:

- **Auto Scaling** automatically adjusts the number of **EC2 instances** in the backend to match traffic demands, ensuring that the application can handle variable loads and maintain high performance.

AWS Well-Architected Framework

1. Operational Excellence Pillar

The **Operational Excellence** pillar focuses on the ability to run and monitor systems effectively while continuously improving processes and gaining insights from operations.

- **Perform Operations as Code:**
  - **Infrastructure as Code (IaC):** In **roomM8**, the entire infrastructure is defined using **AWS CloudFormation**. This ensures that the deployment of resources such as EC2 instances, RDS, load balancers, and security configurations is automated and repeatable.
  - This approach reduces manual intervention, ensuring consistency across environments and minimizing the risk of human errors. By managing infrastructure as code, any changes to the architecture can be easily tracked, tested, and rolled back if needed.
- **Monitoring and Logging:**
  - Real-time monitoring of the **roomM8** application is achieved through **AWS CloudWatch**, which collects logs and metrics from the frontend (Next.js) and

backend (Spring Boot) EC2 instances. This enables centralized logging and monitoring of application performance.

- **CloudWatch Logs** are enabled for both frontend and backend instances, ensuring that any issues are identified and resolved quickly. Logs from the backend EC2 instances, such as error messages or performance metrics, are stored for troubleshooting and analysis.

- **Continuous Improvement:**
  - With **CloudFormation**, changes to the infrastructure can be implemented in a controlled and automated way, ensuring that the application architecture is continuously evolving with minimal disruption.
  - The system's health and operational insights are continuously monitored, enabling iterative improvements and proactive issue resolution.

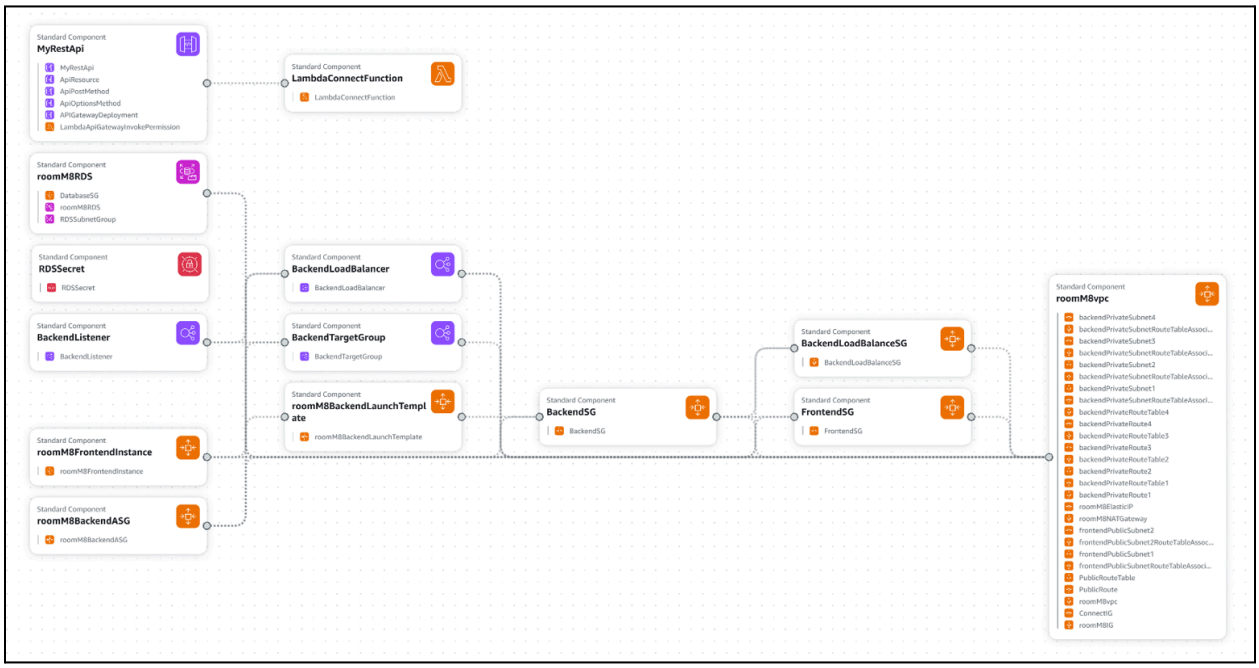

Figure 2: Diagram created from CloudFormation

2. Security Pillar

The **Security** pillar focuses on protecting data, and systems, and ensuring robust security measures are applied at every level.

- **Apply Security at All Layers:**
  - **Infrastructure Protection: roomM8** is deployed in an isolated **VPC**, with **public** and **private subnets**. The **frontend (Next.js)** runs in the **public subnet**,

while the **backend (Spring Boot)** and **MySQL RDS** are placed in a **private subnet** for additional security. This ensures that critical resources are not directly accessible from the internet.

- **Security Groups:** The **Security Group** of the **RDS MySQL** database is configured to allow inbound access only from the backend **EC2 instances** and **API Gateway**, ensuring that unauthorized access is blocked. Similarly, **backend EC2 instances** are protected by **Security Groups** that limit inbound and outbound traffic to only necessary sources.

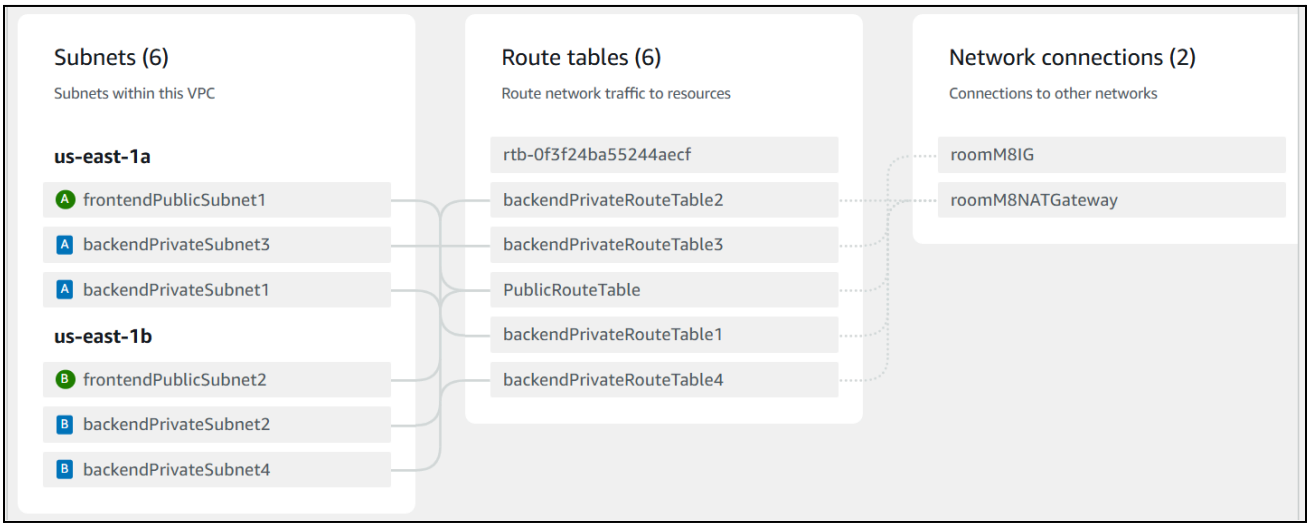

Figure 3: Subnet, Route Table, Internet Gateway, and Nat Gateway

- **Keep People Away from Data:**
  - **Database Encryption: MySQL RDS** provides **encryption at rest** and **in transit** by default, ensuring that sensitive user data is protected both while stored and during transmission.
  - **Secrets Management: AWS Secrets Manager** is used to securely store and manage **MySQL database credentials** and other sensitive information, preventing hardcoded credentials in the application code.
- **Access Management:**
  - **Controlled Network Access:** The **VPC** setup ensures that the backend and database are not publicly exposed. Only the **frontend EC2 instances** in the **public subnet** are accessible via the internet, while the **backend EC2 instances** in the **private subnet** are shielded from direct access.
  - **Load Balancer Security:** The **Elastic Load Balancer (ELB)** allows traffic on **HTTP (80)** and **HTTPS (443)** ports for user access to the frontend, and restricts unnecessary ports. SSH access to the backend instances is only permitted for necessary administrative actions.

3. Reliability Pillar

The **Reliability** pillar ensures that **roomM8** performs its intended function consistently, can recover from failures, and meets customer demands.

- **High Availability & Fault Tolerance:**
  - The **backend EC2 instances** for **roomM8** are managed in an **Auto Scaling group** in a **private subnet**, distributed across multiple **Availability Zones** (e.g., us-east-1a and us-east-1b) to ensure high availability.
  - **Elastic Load Balancer (ELB)** ensures that traffic is distributed evenly across **healthy backend EC2 instances**. If an EC2 instance becomes unhealthy or fails, the load balancer automatically reroutes traffic to healthy instances, ensuring minimal disruption to users.

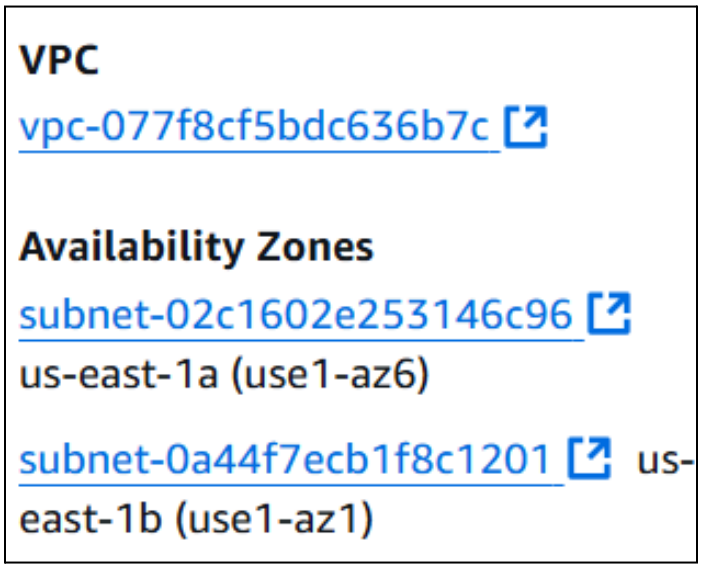

Figure 4: Load Balancer Availability Zones

- **Auto Scaling:**
  - The **Auto Scaling group** ensures that the **backend EC2 instances** scale up or down based on traffic demand. By default, the system runs **one EC2 instance** with **Spring Boot** on port 8080. If the **health check** fails, **Auto Scaling** automatically replaces the unhealthy instance with a new one, maintaining application availability.
  - The **Auto Scaling group** is configured with a **desired capacity of 1** and a **maximum capacity of 3** to ensure that at least one instance is always running and ready to handle requests, with the ability to scale to two instances if needed.

- **Load Balancer:**
    - The **Elastic Load Balancer** distributes incoming user traffic between **EC2 instances**, ensuring that the application remains highly available. If any backend instance becomes unhealthy, the load balancer stops sending traffic to it and routes requests only to healthy instances.
    - **Health checks** are set up to ensure that only instances that are functioning correctly handle user requests, helping maintain consistent performance.
- **Disaster Recovery:**
    - **CloudFormation (IaC): CloudFormation** is used to define the entire infrastructure as code, enabling quick and reliable redeployment of the **roomM8** architecture. In case of a failure or disaster, the infrastructure can be easily recreated in a different **Availability Zone** or even a different **region** by modifying the CloudFormation template parameters.
    - This ensures that **roomM8** can be quickly restored with minimal downtime, ensuring business continuity.

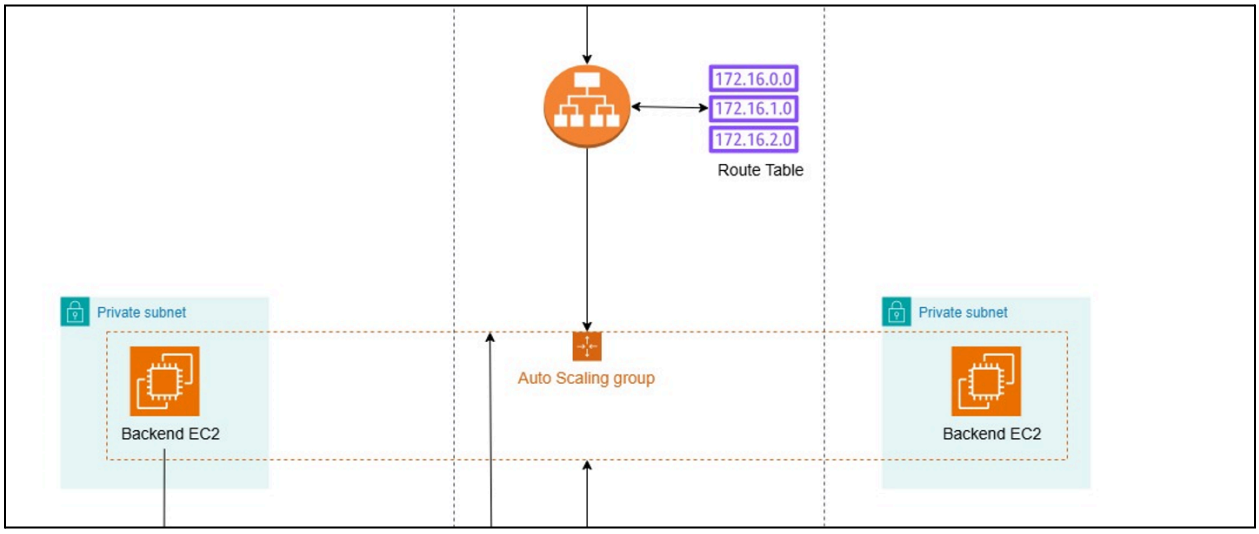

Figure 5: Auto scaling of Backend EC2

4. Performance Efficiency Pillar

The **Performance Efficiency** pillar focuses on using cloud resources efficiently to meet system requirements and optimize performance for your application.

- **Right-Sizing:**
    - **EC2 Instances:** The architecture uses **EC2 instances** for both the **frontend (Next.js)** and **backend (Spring Boot)**. The **t2.medium** instance type provides a good balance of **CPU**, **memory**, and **network performance**, making it suitable for the expected workload.
    - **Auto Scaling** ensures that **roomM8** only uses the necessary resources, reducing costs during periods of low traffic while ensuring availability and performance during peak times.
- **Simplified Management with AWS Services:**
    - **Managed Services: roomM8** utilizes **AWS RDS (MySQL)**, a fully managed relational database service, which offloads the complexity of database management. AWS handles **backups**, **patching**, **scaling**, and **high availability**, allowing you to focus on the application logic without worrying about database administration.
    - **Load Balancer** ensures that traffic is efficiently distributed across the backend **EC2 instances**, optimizing application performance and ensuring high availability even during peak traffic times.
- **Go Global in Minutes:**
    - Multi-AZ Deployment: With Auto Scaling groups deployed across multiple Availability Zones (AZs), roomM8 ensures high availability and consistent performance. This setup helps mitigate any potential failures in a single AZ, ensuring the application remains resilient and responsive to traffic from different geographic regions.

5. Cost Optimization Pillar

The **Cost Optimization** pillar focuses on controlling costs and maximizing the value of your AWS investment.

- **Adopt a Consumption Model:**
    - **Auto Scaling: roomM8** leverages **Auto Scaling** for both the **frontend (Next.js)** and **backend (Spring Boot)** EC2 instances. **Auto Scaling** dynamically adjusts the number of running instances based on real-time traffic. This ensures that **roomM8** only pays for the resources it needs. During low-demand periods, **Auto Scaling** scales down the number of instances, reducing unnecessary costs.
    - **EC2 Instance Types:** By using **t2.medium EC2 instances**, **roomM8** ensures that the resources are appropriately sized for the workload, balancing cost and

performance. **t2.medium** instances are cost-effective for handling the expected traffic while providing the required performance.

- **Pay-as-You-Go Pricing:**
  - **RDS (MySQL): Amazon RDS** operates on a **pay-as-you-go** pricing model, where **roomM8** only pays for the **database resources** used. This helps optimize costs by avoiding over-provisioning of database capacity. **RDS** also offers automatic scaling, which allows **roomM8** to scale its database as needed without manual intervention.
- **Efficient Resource Management:**
  - By using **Auto Scaling** and **Elastic Load Balancer (ELB)** to distribute traffic, **roomM8** ensures that resources are efficiently utilized, preventing over-provisioning of **EC2 instances** or unnecessary server capacity during periods of low traffic.

Infrastructure of Code

CloudFormation Link:
https://github.com/Jay-Kumar-Patel/MACS/blob/main/ASDC/Project/group07/CloudFormation.yml

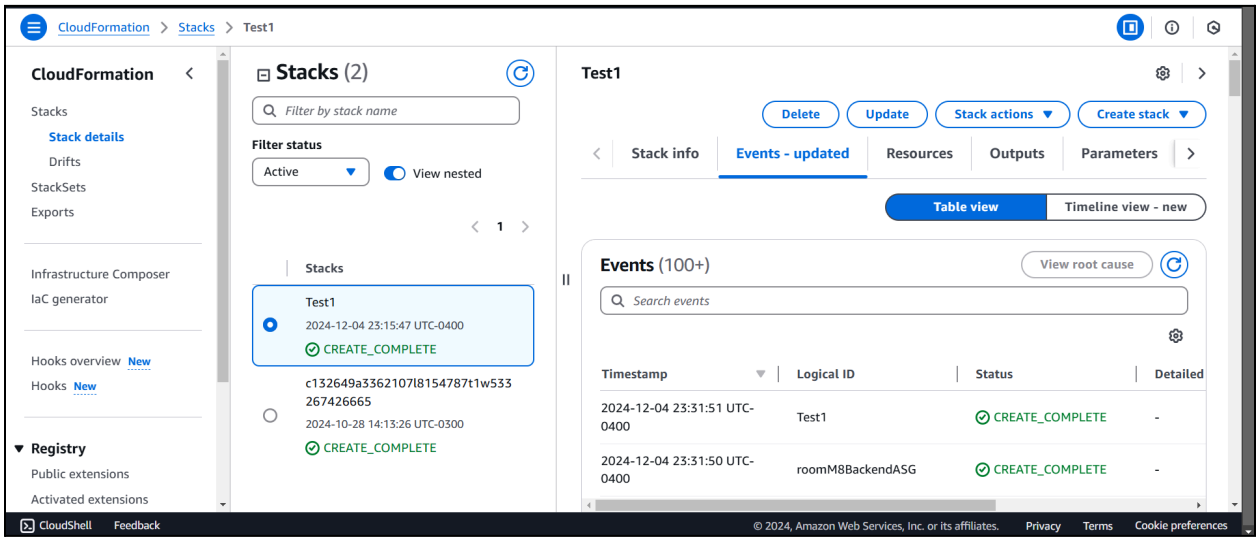

Figure 6: Successful CloudFormation

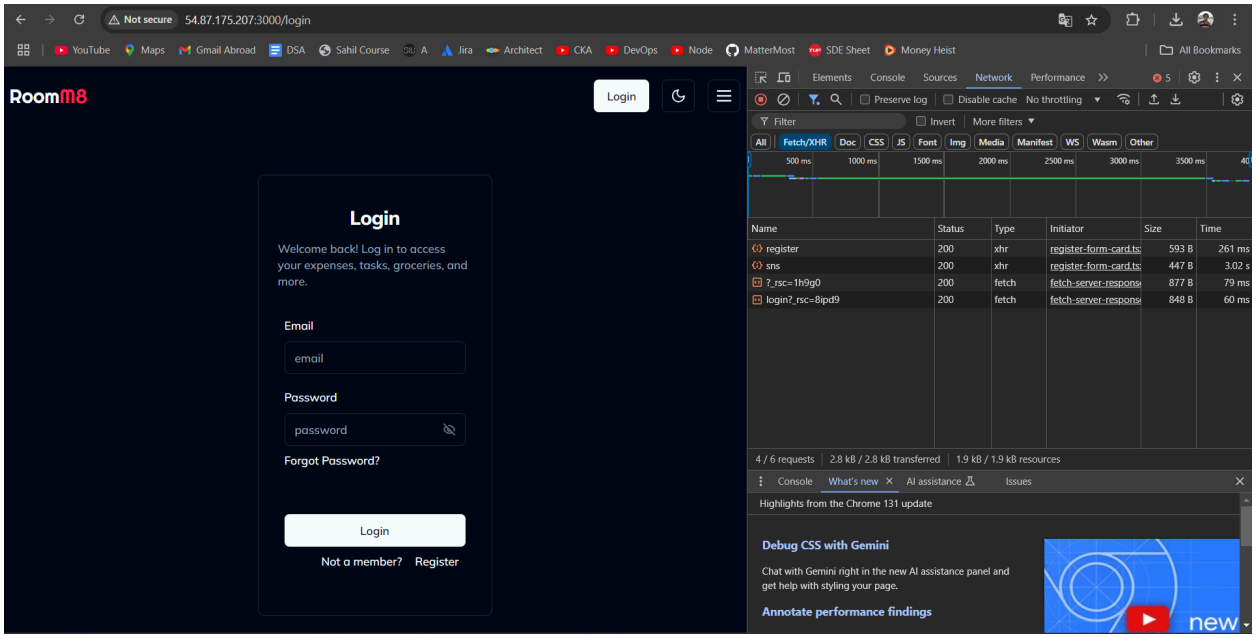

Figure 7: Successful Signup Process

Networking

1. **VPC and Subnets**
    - roomM8vpc: Creates a Virtual Private Cloud (VPC) with a CIDR block of 10.0.0.0/16.
    - **Public Subnets** (frontendPublicSubnet1, frontendPublicSubnet2): Used for frontend resources with public IPs.
    - **Private Subnets** (backendPrivateSubnet1, backendPrivateSubnet2, backendPrivateSubnet3, backendPrivateSubnet4): Used for backend resources and database with no direct internet access.

```yaml
roomM8vpc:
  Type: AWS::EC2::VPC
  Properties:
    CidrBlock: 10.0.0.0/16
    EnableDnsSupport: true
    EnableDnsHostnames: true
    Tags:
      - Key: Name
        Value: roomM8vpc

roomM8IG:
  Type: AWS::EC2::InternetGateway
  Properties:
    Tags:
      - Key: Name
        Value: roomM8IG

ConnectIG:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    VpcId: !Ref roomM8vpc
    InternetGatewayId: !Ref roomM8IG
```

Figure 8: Creating VPC, and Internet Gateway

2. **Routing**
    - PublicRouteTable and PublicRoute: Enables internet connectivity for public subnets via an Internet Gateway (roomM8IG).
    - roomM8NATGateway: Provides internet connectivity to private subnets via a NAT Gateway, enabling backend services to fetch and run docker images securely.

```yaml
PublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref roomM8vpc
    Tags:
      - Key: Name
        Value: PublicRouteTable

PublicRoute:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref roomM8IG
```

Figure 9: Creating Route and Route Table

Security Groups

1. **FrontendSG**: Allows HTTP, HTTPS, and SSH traffic to the frontend EC2 instances.
2. **BackendSG**: Restricts access to the backend instances only to specific sources, such as the frontend instances (through the FrontendSG security group) and the load balancer (through the BackendLoadBalanceSG security group).
3. **DatabaseSG**: Secures database access to only backend instances.

```yaml
FrontendSG:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Allow HTTP and HTTPS access for Frontend
    VpcId: !Ref roomM8vpc
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3000
        ToPort: 3000
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: Name
        Value: FrontendSG

BackendSG:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Allow access from Frontend to Backend
    VpcId: !Ref roomM8vpc
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 9090
        ToPort: 9090
        SourceSecurityGroupId: !Ref BackendLoadBalanceSG
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        SourceSecurityGroupId: !Ref FrontendSG
    Tags:
      - Key: Name
        Value: BackendSG

DatabaseSG:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Allow access from Backend to Database
    VpcId: !Ref roomM8vpc
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        SourceSecurityGroupId: !Ref BackendSG
    Tags:
      - Key: Name
        Value: DatabaseSG
```

Figure 10: Creating Security Groups

Compute and Containers

1. **Frontend Instance (roomM8FrontendInstance)**
    - Runs on an EC2 instance in the public subnet using a Docker container pulled from DockerHub.
    - Configures CloudWatch for monitoring and logging.
    - Exposes port 3000 for the frontend application.
2. **Backend Auto Scaling Group (ASG)**
    - Uses a launch template (roomM8BackendLaunchTemplate) to define backend instances.
    - Pulls a backend Docker image and runs it on port 9090.
    - Ensures scalability by auto-scaling between 1-5 instances based on demand.
    - Backend instances reside in private subnets.

```
roomM8BackendLaunchTemplate:
  Type: AWS::EC2::LaunchTemplate
  Properties:
    LaunchTemplateName: roomM8BackendLaunchTemplate
    LaunchTemplateData:
      InstanceType: t2.medium
      KeyName: roomM8
      ImageId: ami-0ba9883b710b05ac6
      SecurityGroupIds:
        - !Ref BackendSG
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          # Update packages
          sudo yum update -y

          sudo yum install -y amazon-cloudwatch-agent

          sudo bash -c 'cat <<EOF > /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
          ${CloudWatchConfig}
          EOF'
          sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json -s

          # Install Docker
          sudo yum install -y docker
          sudo service docker start

          # Pull Backend Image from DockerHub
          sudo docker pull jay411/backend:test

          # Run the container
          sudo docker run -d -p 9090:9090 \
            -e DB_HOST=${roomM8RDS.Endpoint.Address} \
            -e DB_USER=admin \
            -e DB_PASSWORD=jayspatel \
            jay411/backend:test
```

Figure 11: Creating Backend Launch Template

Database

1. **RDS Instance (roomM8RDS)**
    - A MySQL 8.0 database instance in a multi-AZ deployment for high availability.
    - Secured with a DatabaseSG and uses private subnets for enhanced security.
    - Credentials are stored securely in AWS Secrets Manager (RDSSecret).
2. **RDS Subnet Group**
    - Ensures the RDS instance is deployed in private subnets (backendPrivateSubnet1 and backendPrivateSubnet2).

```
roomM8RDS:
  Type: AWS::RDS::DBInstance
  Properties:
    DBInstanceClass: db.t3.micro
    Engine: mysql
    EngineVersion: 8.0.35
    DBInstanceIdentifier: roomM8DBInstance
    MasterUsername: admin
    MasterUserPassword: jayspatel
    DBName: room8database
    VPCSecurityGroups:
      - !Ref DatabaseSG
    DBSubnetGroupName: !Ref RDSSubnetGroup
    MultiAZ: true
    Port: 3306
    AllocatedStorage: 20
    StorageType: gp2

RDSSubnetGroup:
  Type: AWS::RDS::DBSubnetGroup
  Properties:
    DBSubnetGroupDescription: Subnet group for RDS in Private Subnet 1 and Private Subnet 2
    SubnetIds:
      - !Ref backendPrivateSubnet1
      - !Ref backendPrivateSubnet2
    Tags:
      - Key: Name
        Value: RDSSubnetGroup

RDSSecret:
  Type: AWS::SecretsManager::Secret
  Properties:
    Name: !Ref SECRETMANAGERNAME
    Description: "RDS Instance credentials"
    SecretString: !Sub |
      {
        "username": "${RDSMasterUsername}",
        "password": "${RDSMasterUserPassword}",
        "engine": "mysql",
        "host": "${roomM8RDS.Endpoint.Address}",
        "port": "3306",
        "dbname": "room8database"
      }
```

Figure 12: Creating RDS and storing its credentials into secret manager

Load Balancers

1. **Backend Load Balancer (BackendLoadBalancer)**
    - Distributes traffic to backend instances in the private subnets.
    - Associated with a target group (BackendTargetGroup) that monitors health checks on port 9090.

```yaml
BackendTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Protocol: HTTP
    Port: 9090
    VpcId: !Ref roomM8vpc
    TargetType: instance
    HealthCheckProtocol: HTTP
    HealthCheckPort: 9090
    HealthCheckPath: /
    Tags:
      - Key: Name
        Value: BackendTargetGroup

BackendListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    LoadBalancerArn: !Ref BackendLoadBalancer
    Protocol: HTTP
    Port: 80
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref BackendTargetGroup

BackendLoadBalancer:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name: roomM8BackendLoadBalancer
    Subnets:
      - Ref: frontendPublicSubnet1
      - Ref: frontendPublicSubnet2
    SecurityGroups:
      - Ref: BackendLoadBalanceSG
    LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: 60
    Tags:
      - Key: Name
        Value: BackendLoadBalancer
```

Figure 13: Creating Load Balancer

Serverless and API Gateway

1. **Lambda Function (LambdaConnectFunction)**
    - A Node. js-based function for processing SNS notifications.
    - Deployed via S3 (SNSLambdaCode.zip) with an IAM role for permissions.
2. **API Gateway**
    - Exposes the Lambda function via REST endpoints (/sns) for external applications to invoke.
    - Includes POST and OPTIONS methods with proper CORS configurations.
3. **Permissions**
    - LambdaApiGatewayInvokePermission: Grants API Gateway the permission to invoke the Lambda function.

```yaml
LambdaConnectFunction:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: "SNS"
    Runtime: "nodejs22.x"
    Architectures:
      - "x86_64"
    Handler: "index.handler"
    Code:
      S3Bucket: "snscode"
      S3Key: "SNSLambdaCode.zip"
    Description: "SNS Lambda"
    Role: "arn:aws:iam::533267426665:role/LabRole"
    Timeout: 200

MyRestApi:
  Type: AWS::ApiGateway::RestApi
  Properties:
    Name: "RoomM8APIGateway"
    Description: "API Gateway for SNS Lambda"
    EndpointConfiguration:
      Types:
        - REGIONAL

ApiResource:
  Type: AWS::ApiGateway::Resource
  Properties:
    ParentId: !GetAtt MyRestApi.RootResourceId
    PathPart: "sns"
    RestApiId: !Ref MyRestApi
```

Figure 14: Creating Lambda and API Gateway

Monitoring and Logging

1. **CloudWatch**
    - Configured to monitor disk and memory usage on both frontend and backend EC2 instances.
    - Collects logs from /var/log/messages for debugging and auditing.

```
CloudWatchConfig:
  Type: String
  Default: |
    {
      "metrics": {
        "namespace": "BackendMetrics",
        "metrics_collected": {
          "disk": {
            "measurement": [
              "used_percent"
            ],
            "metrics_collection_interval": 60
          },
          "mem": {
            "measurement": [
              "mem_used_percent"
            ],
            "metrics_collection_interval": 60
          }
        }
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "/var/log/messages",
                "log_group_name": "EC2InstanceLogGroup",
                "log_stream_name": "{instance_id}-messages",
                "timezone": "UTC"
              }
            ]
          }
        }
      }
    }
```

Figure 15: Configuration of cloud watch

References

[1] Amazon Web Services, "Amazon EC2 Reserved Instances Pricing," Amazon WebServices. [Online]. Available: https://aws.amazon.com/ec2/pricing/reserved-instances. [Accessed: December 4, 2024].

[2] Amazon Web Services, "Amazon EC2," Amazon Web Services. [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: December 4, 2024].

[3] Amazon Web Services, "Application Load Balancer Introduction," Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html. [Accessed: December 4, 2024].

[4] Amazon Web Services, "What is Amazon VPC?" Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html. [Accessed: December 4, 2024].

[5] Amazon Web Services, "Amazon CloudWatch," Amazon Web Services. [Online]. Available: https://aws.amazon.com/cloudwatch/. [Accessed: December 4, 2024].

[6] Amazon Web Services, "Operational Excellence Pillar– AWS Well-Architected Framework," Amazon Web Services. [Online]. Available: https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillar.operationalExcellence.en.html. [Accessed: December 4, 2024].

[7] Amazon Web Services, "AWS CloudFormation," Amazon Web Services. [Online]. Available: https://aws.amazon.com/cloudformation/. [Accessed: December 4, 2024].

[8] Amazon Web Services, "Security Pillar– AWS Well-Architected Framework," Amazon Web Services. [Online]. Available: https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillar.security.en.html. [Accessed: December 4, 2024].

[9] Amazon Web Services, "Reliability Pillar– AWS Well-Architected Framework," Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html. [Accessed: December 4, 2024].

[10] Amazon Web Services, "Performance Efficiency Pillar– AWS Well-Architected Framework," Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/performance-efficiency-pillar/design-principles.html. [Accessed: December 4, 2024].

[11] Amazon Web Services, "Cost Optimization Pillar– AWS Well-Architected Framework," Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/design-principles.html. [Accessed: December 4, 2024].

[12] diagrams.net, "diagrams.net," diagrams.net. [Online]. Available: https://app.diagrams.net/. [Accessed: December 4, 2024].

[13] AWS Lambda Documentation, Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/lambda/. [Accessed: December 4, 2024].

[14] Amazon API Gateway Documentation, Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/apigateway/. [Accessed: December 4, 2024].

[15] Amazon SNS Documentation, Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/sns/. [Accessed: December 4, 2024].