# CSCI 5409
# Cloud Computing

Term Assignment – 2024 Summer

Name: Divy Bhavsar

BannerID: B00978267

# Overview

I have developed a comprehensive web-based stock trading application named **'SKEX'**. This application leverages modern web technologies to provide a seamless trading experience. The frontend is crafted using Next.js, ensuring a dynamic and responsive user interface. The backend is built with Node.js 18, offering robust and efficient server-side operations. Data management is handled using AWS RDS with a MariaDB database, ensuring scalable and reliable data storage and retrieval.

## Feature

### Stock Data

- **Data Source**: Stock data is fetched from a third-party API, Polygon.io.
- **Initial Data**: Initially, the application displays data for 50 stocks.
- **Public Access**: Unauthenticated users can view stock data but cannot perform any trading operations.
- **Authenticated Operations**: Logged-in users can buy and sell stocks. If an unauthenticated user attempts to perform a trade, they are redirected to the login page.

### Authentication

#### Sign Up

- **Fields Required**: Email, SIN number, and password.
- **User Experience**: The sign-up process is straightforward, collecting essential information to create a secure user account.

#### Login

- **Fields Required**: Email and password.
- **User Experience**: Users can log in using their email and password, gaining access to trading functionalities.

### Trading Operations

#### Buy and Sell Buttons

- **Location**: Buttons are present in each row of the stock information table.
- **Functionality**: When a user clicks the buy or sell button for a specific stock, they are redirected to the respective form.
- **Forms:**
  - **Buy Form**: Users need to specify the quantity of the stock they wish to purchase and hit the buy button.
  - **Sell Form**: Users need to specify the quantity of the stock they wish to sell and hit the sell button.

- **Validation:**
  - **Buy**: If the user is not logged in, they are redirected to the login page.
  - **Sell**: If the user does not have the stock in their portfolio, the sell request is not accepted.

## User Portfolio

- **Access**: The portfolio page can only be accessed by authenticated users.
- **Content**: The portfolio page displays the stocks purchased by the user, providing an overview of their holdings.

## User Flow

1. **View Stock Data**: All users can view stock data fetched from Polygon.io.
2. **Authentication**:
   - Sign Up: New users provide their email, SIN number, and password to create an account.
   - Login: Existing users log in with their email and password.
3. **Trading**:
   - Buy/Sell Operations: Authenticated users can buy or sell stocks by clicking the respective button in the stock information table.
   - Form Redirection: Upon clicking the buy or sell button, users are redirected to a form where they specify the quantity and confirm the transaction.
4. **Portfolio Management**: Authenticated users can access their portfolio page to view their purchased stocks.

Drawback: As per the API limit , The application can not fetch live price of stocks, which can be solved in future versions.

# AWS Services

### 1. AWS VPC (Virtual Private Cloud)

**Description:**

AWS VPC was used to create a logically isolated network environment where I could launch AWS resources in a virtual network that I define.

**Alternatives Considered:**

- **AWS Transit Gateway**: Provides scalable and easy-to-manage connections between VPCs and on-premises networks but is more suited for complex network architectures.
- **AWS Direct Connect**: Offers a dedicated network connection to AWS but is more suitable for hybrid environments requiring high-bandwidth connections.

**Why Chosen:**

- **Integration**: AWS VPC integrates seamlessly with other AWS services used in the project.
- **Control**: Provides granular control over the network environment, including subnets, route tables, and network gateways.
- **Security**: Enhanced security features, including security groups and network ACLs, ensured a secure environment for sensitive financial data.

### 2. Amazon EC2 (Elastic Compute Cloud)

**Description:**

Amazon EC2 was used to run the backend server of the application, providing scalable computing capacity in the cloud.

**Alternatives Considered:**

- **AWS Elastic Beanstalk**: Provides a managed service for deploying and scaling web applications, but offers less control over the underlying infrastructure.
- **AWS Fargate**: Manages serverless containers, simplifying infrastructure management, but was less suitable for running the specific backend processes required by **SKEX**.

**Why Chosen:**

- **Scalability**: EC2 allows for easy scaling up or down based on demand.
- **Flexibility**: Wide range of instance types to choose from based on performance needs.

- **Integration**: Seamless integration with AWS VPC, RDS, and other AWS services ensured efficient deployment and management.

### 3. AWS Lambda Functions

**Description:**

AWS Lambda was used to handle serverless functions for various background tasks, such as processing stock data and sending notifications.

**Alternatives Considered:**

- **Amazon EC2**: Offers more control over the computing environment but requires management of servers.
- **AWS Batch**: Manages batch computing jobs, providing scheduling and resource allocation, but is more suited for large-scale batch processing.

**Why Chosen:**

- **Cost-Effectiveness**: Pay-as-you-go pricing model reduced costs for infrequent background tasks.
- **Ease of Use**: Simplified the management of code execution without provisioning servers.
- **Integration**: Easily integrates with AWS SNS, RDS, and other services used in the application.

### 4. Amazon RDS (Relational Database Service)

**Description:**

Amazon RDS with MariaDB was used to manage the application's database, providing a scalable and reliable relational database service.

**Alternatives Considered:**

- **Amazon Aurora**: Provides higher performance and availability but at a higher cost, which was not necessary for the initial requirements of SKEX.
- **Amazon DynamoDB**: A fully managed NoSQL database service, which is highly scalable but not suitable for the relational data model used in SKEX.

**Why Chosen:**

- **Managed Service**: Reduced administrative tasks such as backups, patching, and scaling.
- **Reliability**: High availability and automated backups ensured data integrity and availability.
- **Compatibility**: MariaDB was chosen for its compatibility and performance, fitting well with the application's needs.

### 5. AWS SNS (Simple Notification Service)

**Description:**

AWS SNS was used to send notifications to admin about stock transactions and important updates.

**Alternatives Considered:**

- **Amazon SQS (Simple Queue Service)**: Provides message queuing but does not directly handle the publishing and subscribing model needed for notifications.
- **AWS SES (Simple Email Service)**: Focuses on email sending capabilities, which could be part of the notification system but lacks the broader messaging capabilities of SNS.

**Why Chosen:**

- **Ease of Use**: Simple to set up and manage notifications.
- **Integration**: Seamless integration with AWS Lambda and other services.
- **Scalability**: Automatically scales to handle a large number of messages.

### 6. AWS Backup

**Description:**

AWS Backup was used to automate and centrally manage backups across AWS services, ensuring data protection and recovery.

**Alternatives Considered:**

- **Amazon S3**: Provides scalable object storage, but requires custom implementation for backup scheduling and management.
- **AWS Data Lifecycle Manager**: Manages EBS snapshots, providing automated backup management, but does not offer the same centralized management across multiple AWS services as AWS Backup.

**Why Chosen:**

- **Centralized Management**: Allows for centralized management of backups across multiple AWS services.
- **Automation**: Simplifies the process of scheduling and managing backups.
- **Reliability**: Ensures data protection and easy recovery options.

# Deployment Model

The deployment model chosen for the SKEX Trading Application involves a hybrid cloud infrastructure that leverages both public and private resources within AWS. This model provides the flexibility, scalability, and security required for a stock trading platform. The architecture consists of a Virtual Private Cloud (VPC) with public and private subnets, an EC2 instance, Lambda functions, an RDS database, SNS for notifications, and AWS Backup for data protection.

**Components**

1. **AWS VPC**
   - **Reasoning**:
     - **Isolation and Security**: The VPC provides a logically isolated network that ensures security and control over the network configuration.
     - **Customizability**: Ability to define multiple subnets, route tables, and network gateways to segregate public and private resources.
   - **Configuration**:
     - CIDR Block: 10.0.0.0/16
     - Subnets: 3 public and 3 private subnets across different availability zones.
2. **EC2 Instance**
   - **Reasoning**:
     - **Compute Resources**: Required for hosting the application frontend or backend components that need a persistent runtime environment.
     - **Flexibility**: Easy to scale vertically by changing instance types as demand changes.
   - **Configuration**:
     - Instance Type: t2.micro (for initial deployment)
     - Security Group: Allows traffic on required ports (e.g., 3000, 5000) and restricts access to only necessary IP ranges.
3. **Lambda Functions**
   - **Reasoning**:
     - **Serverless**: Reduces the need for managing infrastructure, with automatic scaling based on the load.

        - **Cost Efficiency**: Charges only for the compute time used, making it cost-effective for event-driven operations.
    - **Functions**:
        - buy-stock-test: Handles stock purchase requests.
        - sell-stock-test: Handles stock sell requests.
    - **Integration**: Connected to API Gateway to expose these functions as RESTful APIs.
4. **RDS Database**
    - **Reasoning**:
        - **Managed Service**: Simplifies database management tasks such as backups, patching, and scaling.
        - **Reliability**: Provides high availability and durability with multi-AZ deployments.
    - **Configuration**:
        - Engine: MariaDB
        - Instance Class: db.r5.large (memory optimized)
        - Security: Deployed in private subnets with restricted access through security groups.
5. **AWS SNS (Simple Notification Service)**
    - **Reasoning**:
        - **Real-time Notifications**: Facilitates real-time notifications to users about significant events like stock transactions.
        - **Scalability**: Can handle a high throughput of messages, ensuring reliable delivery.
    - **Configuration**:
        - Topic: skex-sns
        - Subscription: Email notification to dibhavsar214@gmail.com
6. **AWS Backup**
    - **Reasoning**:
        - **Automated Backups**: Ensures regular, automated backups of RDS databases, enhancing data protection and recovery.

    - **Compliance**: Helps meet compliance requirements by maintaining backups as per defined policies.
  - **Configuration**:
    - Backup Vault: skex-backup-vault
    - Backup Plan: skex-backup-plan with daily backups retained for 30 days.

**Deployment Model**

The deployment model follows a multi-tier architecture with a separation of concerns between different layers:

1. **Network Layer (VPC and Subnets)**
   - Public subnets host resources that need direct internet access, such as the EC2 instance for web applications.
   - Private subnets host the RDS database and other internal services, enhancing security by restricting direct internet access.
2. **Compute Layer (EC2 and Lambda)**
   - The EC2 instance provides a traditional compute environment for hosting persistent applications.
   - Lambda functions handle specific, event-driven tasks like stock transactions, ensuring efficient resource utilization.
3. **Database Layer (RDS)**
   - A managed database service (MariaDB) provides reliable and scalable data storage.
4. **Notification and Backup Layer (SNS and AWS Backup)**
   - SNS facilitates user notifications, ensuring timely communication of critical events.
   - AWS Backup automates data protection, ensuring regular backups and quick recovery options.

**Benefits of Chosen Deployment Model**

1. **Scalability**: Both EC2 instances and Lambda functions can be scaled to meet demand, ensuring the system can handle varying loads efficiently.
2. **Security**: The VPC ensures isolation, and security groups restrict access, protecting resources from unauthorized access.

3. **Cost Efficiency**: The combination of on-demand EC2 instances and serverless Lambda functions optimizes costs by paying only for what is used.
4. **Manageability**: AWS managed services like RDS and Backup reduce the operational overhead, allowing the focus to remain on application development and improvement.
5. **Flexibility**: The architecture can be easily adapted to include more services or scale existing ones as the application grows.

This deployment model provides a robust, scalable, and secure foundation for the SKEX Trading Application, ensuring optimal performance and reliability for users.

# Delivery Model

The chosen delivery model for the SKEX Trading Application relies on **Docker-based deployment** combined with **Infrastructure as Code (IaaC)** using **AWS CloudFormation**. This model ensures that the application is consistently deployed in a containerized environment, providing portability, scalability, and ease of management.

**Components of the Delivery Model**

1. **Docker-based Deployment**
    - **Reasoning**:
        - **Portability**: Docker containers encapsulate all the dependencies and environment configurations, ensuring the application runs consistently across different environments.
        - **Isolation**: Each service runs in its own container, providing isolation and reducing the risk of conflicts between services.
        - **Scalability**: Containers can be easily scaled up or down based on demand.
        - **Efficiency**: Docker images can be pulled from a repository, making deployments faster and more efficient.
    - **Tools Used**:
        - **Docker**: To create, manage, and run containers.
        - **Docker Hub**: To store and distribute Docker images.
2. **Infrastructure as Code (IaC) using AWS CloudFormation**
    - **Reasoning**:
        - **Reproducibility**: CloudFormation templates ensure consistent and repeatable deployments of the entire infrastructure.
        - **Version Control**: Infrastructure code can be versioned and managed similarly to application code.
    - **Tools Used**:
        - **AWS CloudFormation**: To define and provision the AWS infrastructure including VPCs, subnets, EC2 instances, RDS, Lambda functions, SNS topics, and backup configurations.

**Docker-based Deployment Workflow**

1. **Build Docker Images**
    - Docker images for the application are built and tested locally.

   - Images are tagged and pushed to Docker Hub .

2. **Pull Docker Images**
   - During deployment, Docker images are pulled from the repository into the EC2 instances.
   - EC2 instances run the Docker containers using the pulled images.

**Infrastructure Provisioning Workflow using AWS CloudFormation**

1. **Define Infrastructure as Code**
   - CloudFormation templates define the entire infrastructure required for the SKEX application, including:
     - VPC with public and private subnets.
     - Security groups for EC2 instances and RDS.
     - EC2 instances configured to pull and run Docker containers.
     - RDS instance for the database.
     - Lambda functions for specific operations.
     - SNS topics for notifications.
     - Backup configurations for data protection.
2. **Deploy Infrastructure**
   - CloudFormation stacks are deployed to provision the defined infrastructure.
   - The stack manages the lifecycle of the infrastructure, allowing updates, deletions, and monitoring.

**Benefits of the Chosen Delivery Model**

- **Consistency**: Ensures that the application and infrastructure are consistently deployed across different environments.
- **Efficiency**: Reduces the time and effort required for deployment and configuration, allowing developers to focus on application development.
- **Scalability**: Easily scales the application and infrastructure based on demand.
- **Portability**: Docker containers provide a portable environment, making it easier to move the application across different environments or cloud providers.
- **Manageability**: Infrastructure as Code with CloudFormation simplifies the management and monitoring of the AWS resources.

# Security Analysis

Security is a critical component of the SKEX stock trading application. Ensuring data security through all stages of the system—whether in transit or at rest—was a key consideration during the development process. This analysis outlines the security measures implemented to protect data and ensure compliance with industry standards.

**Data Security in Transit**

**HTTP Encryption**

All data transmitted between the client and the server is secured using HTTP. HTTP ensures that the data is encrypted using Transport Layer Security (TLS), preventing attacks. Key measures include:

**API Gateway**

For the serverless components, AWS API Gateway is used to expose APIs securely. Key features include:

- **Request Validation**: Validates incoming requests to ensure they meet the defined API specifications.
- **Rate Limiting**: Protects against denial-of-service (DoS) attacks by limiting the number of requests a client can make within a specified time period.
- **Usage Plans and API Keys**: Provides controlled access to the APIs, ensuring only authorized users can interact with the backend services.

**Data Security at Rest**

**Amazon RDS**

The application's database is hosted on Amazon RDS, which provides robust security features for data at rest.

**Amazon S3 and AWS Backup**

For backups and other stored data, Amazon S3 is used along with AWS Backup to ensure data security.

**EC2 Instances**

The EC2 instances hosting the backend server also follow strict security protocols for data at rest.

**IAM Roles**: EC2 instances use IAM roles to securely access other AWS services without the need for storing credentials on the instances.

### Network Security

#### AWS VPC

AWS VPC provides a secure network environment for all components of the application. Key measures include:

- **Subnets**: Public and private subnets are used to isolate resources based on their exposure to the internet.
- **Security Groups**: Act as virtual firewalls to control inbound and outbound traffic to the instances.

### Authentication and Authorization

#### AWS IAM

AWS Identity and Access Management (IAM) is used to manage access to AWS services and resources. Key measures include:

- **Fine-Grained Permissions**: IAM policies are used to grant the least privilege necessary to users and services.
- **Multi-Factor Authentication (MFA)**: MFA is enabled for administrative access to the AWS management console, ensuring an additional layer of security.

#### Application-Level Security

The application itself implements robust authentication and authorization mechanisms. Key measures include:

- **User Authentication**: Users sign up with an email, SIN number, and password. Passwords are hashed and stored securely.
- **Session Management**: Secure session management ensures that user sessions are handled securely, preventing session hijacking.

### Monitoring and Incident Response

#### CloudWatch

To ensure continuous monitoring and quick incident response, CloudWatch are utilized. Key measures include:

- **Logging and Monitoring**: All API calls, changes to resources, and operational logs are monitored and recorded.

# Cost Analysis

**1. AWS VPC**

**Upfront Costs:**

- **Creation of VPC and subnets**: No upfront cost; included in the ongoing costs.

**Ongoing Costs:**

- **VPC**: No direct cost for the VPC itself.
- **Subnets**: No direct cost for subnets.
- **Internet Gateway**: No direct cost.
- **NAT Gateway**: Not mentioned in the template but if used, costs $0.045 per hour and $0.045 per GB data processed.
- **Data Transfer**: Data transfer out to the internet costs $0.09 per GB.

**2. AWS EC2**

**Upfront Costs:**

- **EC2 Instance**: No upfront cost if using On-Demand instances. Reserved instances have upfront payment options.

**Ongoing Costs:**

- **EC2 Instance (t2.micro)**:
  - On-Demand Pricing: Approximately $0.0116 per hour.
  - Monthly Cost: $0.0116 * 24 * 30 = $8.35 (approx.)
- **EBS Storage**:
  - Assuming 30 GB of General Purpose SSD (gp2): $0.10 per GB-month.
  - Monthly Cost: 30 * $0.10 = $3.00
- **Data Transfer**:
  - Data transfer out to the internet costs $0.09 per GB after the first 1 GB per month.

**3. AWS Lambda Functions**

**Upfront Costs:**

- **Lambda Function Creation**: No upfront cost.

**Ongoing Costs:**

- **Lambda Execution**:
  - Memory: 128 MB
  - Duration: 100 ms
  - Number of Requests: Assume 1 million requests per month
- **Cost Calculation**:
  - Compute Charges: 1M requests * 128 MB * 100 ms = 12.8 GB-seconds.
  - Monthly Cost: First 1M requests and 400,000 GB-seconds are free. Beyond that, $0.20 per 1M requests and $0.00001667 per GB-second.

Since the example usage falls within the free tier, the monthly cost is $0.

**4. AWS RDS**

**Upfront Costs:**

- **RDS Instance Creation**: No upfront cost if using On-Demand instances. Reserved instances have upfront payment options.

**Ongoing Costs:**

- **RDS Instance (db.r5.large)**:
  - On-Demand Pricing: Approximately $0.283 per hour.
  - Monthly Cost: $0.283 * 24 * 30 = $203.76 (approx.)
- **Storage**:
  - Allocated Storage: 20 GB
  - Cost: $0.115 per GB-month.
  - Monthly Cost: 20 * $0.115 = $2.30
- **Backup Storage**:
  - 100% of the allocated storage for free; beyond that, $0.095 per GB-month.

**5. AWS SNS**

**Upfront Costs:**

- **SNS Topic Creation**: No upfront cost.

**Ongoing Costs:**

- **SNS Usage**:
  - Number of messages published: Assume 1,000 per month
  - Cost: First 1 million requests per month are free. Beyond that, $0.50 per 1M requests.
  - Email Delivery: $0.0004 per email
- **Monthly Cost**: Assuming 1,000 emails.
  - SNS Requests: Free
  - Email Cost: 1,000 * $0.0004 = $0.40

**6. AWS Backups**

**Upfront Costs:**

- **Backup Plan Creation**: No upfront cost.

**Ongoing Costs:**

- **Backup Storage**:
  - Assume 20 GB of RDS snapshot storage
  - Cost: $0.095 per GB-month.
  - Monthly Cost: 20 * $0.095 = $1.90

**Total Monthly Costs:**

- **VPC**: Minimal costs, mostly from data transfer.
- **EC2 Instance**: $8.35 + $3.00 = $11.35
- **Lambda Functions**: $0 (within free tier)
- **RDS Instance**: $203.76 + $2.30 = $206.06
- **SNS**: $0.40
- **Backups**: $1.90

**Total Monthly Estimate**: $11.35 + $206.06 + $0.40 + $1.90 = $219.71 (approx.)

**Alternative Approaches**

1. **EC2 Reserved Instances**:
   - Switching to Reserved Instances for EC2 could save up to 75% over On-Demand prices, depending on the term and payment options.

2. **Using Smaller RDS Instances**:
    - Downgrading the RDS instance to a smaller class (e.g., db.t3.medium) could significantly reduce costs if the performance requirements are not as high.
3. **Lambda Cost Optimization**:
    - Ensuring Lambda functions are optimized to run in the least amount of time and with the minimum necessary memory can help reduce costs further.
4. **Use of Spot Instances for EC2**:
    - Spot Instances can be used for non-critical workloads, potentially reducing EC2 costs by up to 90%.
5. **Consolidating SNS and Lambda Functions**:
    - Combining multiple related functionalities into fewer Lambda functions and minimizing SNS usage could further optimize costs.

**Justification for the Solution**

1. **Scalability and Flexibility**:
    - Using On-Demand and t2.micro instances ensures flexibility and scalability for fluctuating workloads.
2. **High Availability and Performance**:
    - The choice of db.r5.large for RDS ensures high performance and reliability, which is crucial for database operations.
3. **Compliance and Data Protection**:
    - Implementing AWS Backups with regular snapshots ensures data protection and compliance with best practices for disaster recovery.