NAAN MUDHALVAN

FreshBasket: Scalable E-commerce Platform Deployment with Flask on AWS EC2 and RDS

Project Created by: Keerthivasan G,Akhilan LH,Kumar D,Kumar R,Manish ESJ

Project Created Date: 21/Nov/2024

College Code: 3135

College Name: Panimalar Engineering College Chennai City Campus

Team Name: NM2024TMID14450

NAAN MUDHALVAN

FreshBasket: Scalable E-commerce Platform Deployment with Flask on AWS EC2 and RDS

Project Created by: Keerthivasan G,Akhilan LH,Kumar D,Kumar R,Manish ESJ

Project Created Date: 21/Nov/2024

College Code: 3135

College Name: Panimalar Engineering College Chennai City Campus

Team Name: NM2024TMID14450

BONAFIDE CERTIFICATE

Certified that this Naan Mudhalvan project report **"FreshBasket: Scalable E-commerce Platform Deployment with Flask on AWS EC2 and RDS"** is the bonafide work of _____ who carried out the project work under my supervision.

SIGNATURE
Project Coordinator
Naan Mudhalvan

SIGNATURE
SPoC
Naan Mudhalvan

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

FreshBasket: Scalable E-commerce Platform Deployment with Flask on AWS EC2 and RDS

The project focuses on the development and deployment of a scalable e-commerce platform for selling fresh vegetables and fruits. Using Flask as the backend framework, the platform ensures efficient handling of user interactions and seamless integration of features such as product catalog management, order processing, and user authentication. To achieve scalability and high availability, the platform leverages Amazon Web Services (AWS) infrastructure. AWS EC2 is used for hosting the application, providing a flexible and robust environment to handle varying user demands, while Amazon RDS ensures reliable and efficient database management for handling product and transactional data.

The project demonstrates the advantages of cloud-native solutions in addressing critical challenges of e-commerce platforms, including scalability, fault tolerance, and real-time performance. With its modular architecture, the platform can dynamically scale to accommodate traffic surges and maintain smooth operations during peak usage. The implementation highlights the integration of modern cloud technologies and their impact on enhancing operational efficiency and user experience. This work provides a scalable blueprint for building cloud-based e-commerce platforms and sets the foundation for future enhancements such as machine learning-driven product recommendations and advanced analytics.

Project Description

The "FreshBasket" project involves developing and deploying a scalable e-commerce platform for selling vegetables and fruits. The platform uses Flask for backend development, AWS EC2 for hosting, and Amazon RDS for database management. This cloud-native solution ensures high availability, scalability, and efficient management of the platform’s operations. The project demonstrates how leveraging AWS services can create a robust infrastructure for managing user interactions, product catalogs, and order processing in a seamless manner.

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Overview of E-commerce Platforms and Their Growth: The evolution of e-commerce platforms has transformed traditional shopping into a digital-first experience. With the proliferation of the internet and digital payment systems, e-commerce platforms have become a cornerstone of the retail industry, enabling businesses to reach a global audience.

Motivation Behind FreshBasket: The motivation behind **FreshBasket** is to bridge the gap between local produce sellers and consumers by offering a reliable and scalable platform. By leveraging cloud technologies like AWS, the project aims to overcome traditional e-commerce challenges such as scalability, high availability, and cost efficiency.

1.2 Problem Statement

E-commerce platforms for fresh produce face several challenges:

1. **Scalability Issues:** Seasonal fluctuations in demand and regional festivals can cause sudden spikes in traffic, making it difficult to ensure smooth platform operations.
2. **Reliability Concerns:** Even minor downtimes can result in loss of sales and customer trust.
3. **Operational Efficiency:** Managing product catalogs, inventory, and order fulfillment in real-time while ensuring data integrity is complex.
4. **Cost Constraints:** Many businesses, especially small-scale sellers, struggle to afford robust e-commerce solutions.

The project addresses these challenges by developing a **cloud-native solution** utilizing **AWS EC2** for hosting and **RDS** for database management. This ensures seamless operations while maintaining cost-effectiveness.

1.3 Objectives of the Project

The primary objectives of the **FreshBasket** project are:

1. **Develop a Cloud-Native Platform:** Build a scalable and efficient backend using Flask, hosted on AWS EC2 instances.
2. **Ensure High Availability and Fault Tolerance:** Design the system to minimize downtimes and guarantee reliable user interactions.
3. **Enable Real-Time Operations:** Implement real-time updates for user transactions, product inventory, and order tracking.
4. **Integrate AWS RDS for Data Management:** Utilize AWS Relational Database Service (RDS) to handle customer data, product catalogs, and transactional records efficiently.
5. **Demonstrate Scalability and Cost-Effectiveness:** Create a system that can handle growing user demand while optimizing costs using AWS services.

1.4 Scope of the Project

The scope of this project is focused on the following key areas:

1. **Backend Architecture and Deployment:**
    - Development of the backend using Flask, a lightweight and efficient web framework.
    - Deployment of the application on AWS EC2 to ensure a scalable and reliable hosting environment.

2. **Integration of AWS Services:**
    - AWS RDS will be used for relational database management to store product and transactional data.
    - Other AWS tools, such as AWS CloudWatch, may be integrated for monitoring and performance analysis.
3. **Limitations:**
    - The project will not focus extensively on frontend development; only a basic interface is provided for demonstration purposes.
    - Advanced features, such as AI-driven recommendations or multi-regional support, are outside the current scope.
4. **Future Expansion Opportunities:**
    - Integration of payment gateways for smoother transactions.
    - Addition of advanced analytics for inventory prediction and demand forecasting.
    - Development of a mobile application to enhance accessibility for users.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing E-commerce Solutions

Overview of Existing Platforms: Popular e-commerce platforms like Amazon, Flipkart, Instacart, and BigBasket have set benchmarks in the online retail market. These platforms excel in:

1. **Product Catalog Management:** Maintaining a vast and dynamic product inventory.
2. **User Experience:** Offering seamless navigation, personalized recommendations, and secure payment systems.
3. **Scalability:** Handling millions of users and transactions daily without performance degradation.

Challenges Faced by Smaller Platforms: While large platforms leverage advanced technologies, smaller platforms face significant challenges:

- High setup and maintenance costs.
- Lack of scalability to handle peak traffic.
- Difficulty integrating advanced analytics and automation.

FreshBasket aims to address these issues by providing a scalable and cost-effective solution for niche markets, specifically fresh produce e-commerce.

2.2 Role of Cloud Computing in E-commerce

Scalability and Flexibility: Cloud computing allows e-commerce platforms to scale resources up or down based on user demand. Services like AWS EC2 ensure that platforms can handle sudden traffic surges during events like sales or festivals.

Cost Efficiency: Traditional hosting methods require significant upfront investments in hardware. Cloud solutions operate on a **pay-as-you-go** model, reducing costs significantly for startups and small businesses.

High Availability and Reliability: Cloud providers like AWS offer multiple availability zones, ensuring minimal downtime and disaster recovery capabilities. This is critical for e-commerce platforms to maintain customer trust.

Real-Time Data Management: Cloud computing enables real-time data processing, which is essential for inventory updates, order tracking, and user interaction.

Case Study: Platforms like Shopify have leveraged cloud computing to empower small-scale sellers with reliable and scalable solutions. FreshBasket adopts a similar approach, focusing on fresh produce e-commerce.

2.3 Overview of AWS Services

AWS EC2 (Elastic Compute Cloud): A web service that provides secure, resizable compute capacity in the cloud. EC2 is used in FreshBasket for hosting the backend application, ensuring scalability and reliability.

AWS RDS (Relational Database Service): RDS simplifies the setup, operation, and scaling of relational databases in the cloud. FreshBasket uses RDS to manage the product catalog, user data, and transactions.

AWS CloudWatch: Enables monitoring and logging of system performance, allowing the team to analyze metrics and optimize the application.

AWS S3 (Simple Storage Service): Provides scalable storage for application data, such as images and static files, ensuring durability and accessibility.

Advantages of Using AWS for FreshBasket:

1. **Global Reach:** Ensures the platform can serve users from multiple regions.
2. **Scalable Infrastructure:** Supports growth in user demand without manual intervention.
3. **Security:** Offers built-in security features, such as encryption and identity management.

2.4 Flask as a Backend Framework

Introduction to Flask: Flask is a lightweight, Python-based web framework known for its simplicity and flexibility. It follows a microservices architecture, making it ideal for building modular and scalable applications.

Why Flask for FreshBasket?

1. **Ease of Use:** Flask's intuitive design allows rapid development and easy integration with other tools.
2. **Scalability:** Its modular approach enables seamless scaling as the platform grows.
3. **Integration Capabilities:** Flask can easily integrate with AWS services, databases, and third-party APIs.

Features of Flask:

- Built-in development server and debugger.
- Jinja2 templating for dynamic web page rendering.
- RESTful request handling for API development.
- WSGI compliance for deployment with production servers.

Comparison with Other Frameworks: While frameworks like Django offer more out-of-the-box features, Flask's minimalistic nature makes it more suitable for cloud-native projects like FreshBasket, where customization and modularity are essential.

CHAPTER 3

TECHNOLOGIES USED

3.1 System Architecture

Overview of FreshBasket Architecture: The system architecture of FreshBasket is designed to ensure scalability, high availability, and efficient management of operations. It follows a three-tier architecture comprising:

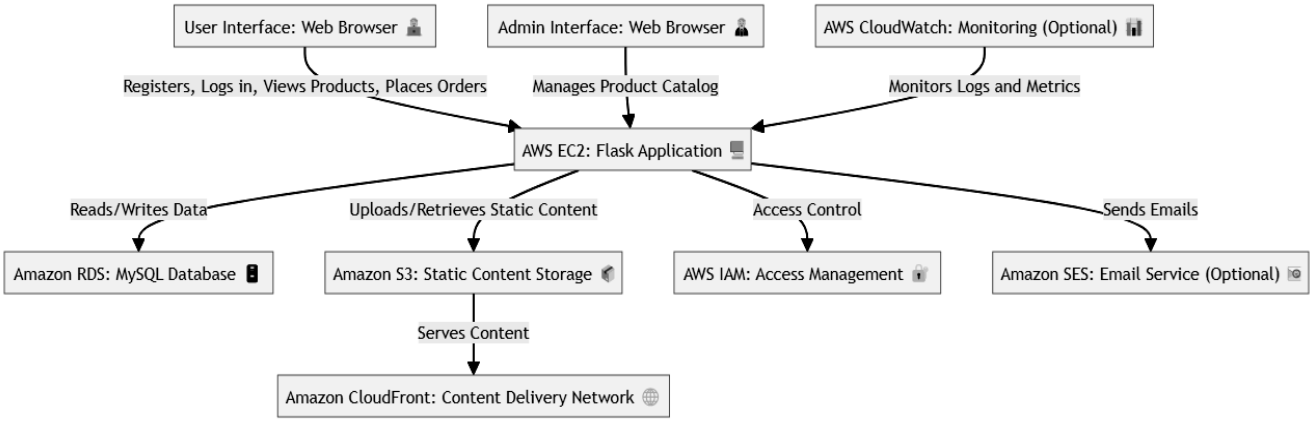
1. **Presentation Layer:** The user interface (UI) for customers to browse products, add items to the cart, and place orders.
2. **Application Layer:** Flask-based backend handling business logic, API endpoints, and interactions with the database.
3. **Data Layer:** AWS RDS for secure and efficient storage and retrieval of data.

Key Components of the Architecture:

- **Frontend:** A basic web interface developed using HTML, CSS, and JavaScript for demonstration purposes.
- **Backend:** Flask framework for handling requests and responses, managing product catalogs, and processing orders.
- **Database:** Amazon RDS (MySQL) for storing user data, product information, and transaction details.
- **Cloud Infrastructure:** AWS EC2 for hosting, ensuring scalability and reliability.

Architecture Diagram: A visual representation of the architecture includes:

1. User interactions via a web browser.
2. API calls routed through Flask.
3. Data storage and retrieval from AWS RDS.
4. Monitoring and scaling through AWS CloudWatch and Auto Scaling Groups.



3.2 AWS Services Used

1. AWS EC2 (Elastic Compute Cloud):

- Deployed the backend application on EC2 instances.
- Auto-scaling enabled to handle traffic fluctuations.

2. AWS RDS (Relational Database Service):

- Used for managing structured data, including user information, product catalogs, and transactions.
- Ensured high availability using Multi-AZ deployment.

3. AWS CloudWatch:

- Monitored system performance and resource utilization.
- Set alarms to notify in case of unusual behavior or resource overuse.

4. AWS S3 (Simple Storage Service):

- Stored product images and static assets.
- Provided secure, scalable, and cost-effective storage.

5. AWS IAM (Identity and Access Management):

- Managed user permissions and access to AWS resources.
- Implemented role-based access control to secure the platform.

6. AWS Auto Scaling:

- Automatically adjusted the number of EC2 instances based on demand.
- Reduced operational costs by optimizing resource usage.

7. AWS Elastic Load Balancer (ELB):

- Distributed incoming traffic across multiple EC2 instances.
- Improved application fault tolerance and scalability.

3.3 Backend Development

Flask Framework:

- Flask was chosen for its simplicity, flexibility, and suitability for REST API development.
- Developed API endpoints for functionalities like user registration, product browsing, and order placement.

Database Integration:

- Connected Flask application to AWS RDS using the SQLAlchemy ORM.
- Designed database schema to handle users, products, orders, and inventory efficiently.

API Design:

- Developed RESTful APIs for seamless interaction between the frontend and backend.
- Example Endpoints:
    - GET /products: Fetch the product catalog.
    - POST /orders: Place an order.
    - GET /order-status: Check the status of an order.

Middleware and Error Handling:

- Implemented middleware for authentication and logging.
- Added error-handling mechanisms to manage exceptions gracefully.

Performance Optimization:

- Utilized caching mechanisms like Flask-Caching to reduce database load.
- Optimized database queries for faster response times.

3.4 Deployment Strategy

1. EC2 Deployment:

- The Flask application was containerized using Docker for consistency across development and production environments.
- Deployed Docker containers to AWS EC2 instances for scalable hosting.

2. Continuous Integration/Continuous Deployment (CI/CD):

- Configured CI/CD pipelines using AWS CodePipeline and CodeDeploy.
- Ensured automated testing and deployment to reduce manual intervention.

3. High Availability:

- Utilized Elastic Load Balancer (ELB) to distribute traffic across multiple EC2 instances.
- Deployed instances across multiple Availability Zones for fault tolerance.

4. Security Measures:

- Configured security groups to allow only necessary inbound traffic.
- Used HTTPS for secure communication.
- Enabled encryption for data at rest and in transit.

CHAPTER 4

PROJECT FLOW

4.1 System Architecture

Overview of FreshBasket Architecture: The system architecture of FreshBasket is designed to ensure scalability, high availability, and efficient management of operations. It follows a three-tier architecture comprising:

1. **Presentation Layer:** The user interface (UI) for customers to browse products, add items to the cart, and place orders.
2. **Application Layer:** Flask-based backend handling business logic, API endpoints, and interactions with the database.
3. **Data Layer:** AWS RDS for secure and efficient storage and retrieval of data.

Key Components of the Architecture:

- **Frontend:** A basic web interface developed using HTML, CSS, and JavaScript for demonstration purposes.
- **Backend:** Flask framework for handling requests and responses, managing product catalogs, and processing orders.
- **Database:** Amazon RDS (MySQL) for storing user data, product information, and transaction details.
- **Cloud Infrastructure:** AWS EC2 for hosting, ensuring scalability and reliability.

Architecture Diagram: A visual representation of the architecture includes:

1. User interactions via a web browser.
2. API calls routed through Flask.
3. Data storage and retrieval from AWS RDS.
4. Monitoring and scaling through AWS CloudWatch and Auto Scaling Groups.

4.2 AWS Services Used

1. AWS EC2 (Elastic Compute Cloud):

- Deployed the backend application on EC2 instances.
- Auto-scaling enabled to handle traffic fluctuations.

2. AWS RDS (Relational Database Service):

- Used for managing structured data, including user information, product catalogs, and transactions.
- Ensured high availability using Multi-AZ deployment.

3. AWS CloudWatch:

- Monitored system performance and resource utilization.
- Set alarms to notify in case of unusual behavior or resource overuse.

4. AWS S3 (Simple Storage Service):

- Stored product images and static assets.
- Provided secure, scalable, and cost-effective storage.

5. AWS IAM (Identity and Access Management):

- Managed user permissions and access to AWS resources.
- Implemented role-based access control to secure the platform.

6. AWS Auto Scaling:

- Automatically adjusted the number of EC2 instances based on demand.
- Reduced operational costs by optimizing resource usage.

7. AWS Elastic Load Balancer (ELB):

- Distributed incoming traffic across multiple EC2 instances.
- Improved application fault tolerance and scalability.

4.3 Backend Development

Flask Framework:

- Flask was chosen for its simplicity, flexibility, and suitability for REST API development.
- Developed API endpoints for functionalities like user registration, product browsing, and order placement.

Database Integration:

- Connected Flask application to AWS RDS using the SQLAlchemy ORM.
- Designed database schema to handle users, products, orders, and inventory efficiently.

API Design:

- Developed RESTful APIs for seamless interaction between the frontend and backend.
- Example Endpoints:
    - GET /products: Fetch the product catalog.
    - POST /orders: Place an order.
    - GET /order-status: Check the status of an order.

Middleware and Error Handling:

- Implemented middleware for authentication and logging.
- Added error-handling mechanisms to manage exceptions gracefully.

Performance Optimization:

- Utilized caching mechanisms like Flask-Caching to reduce database load.
- Optimized database queries for faster response times.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Introduction

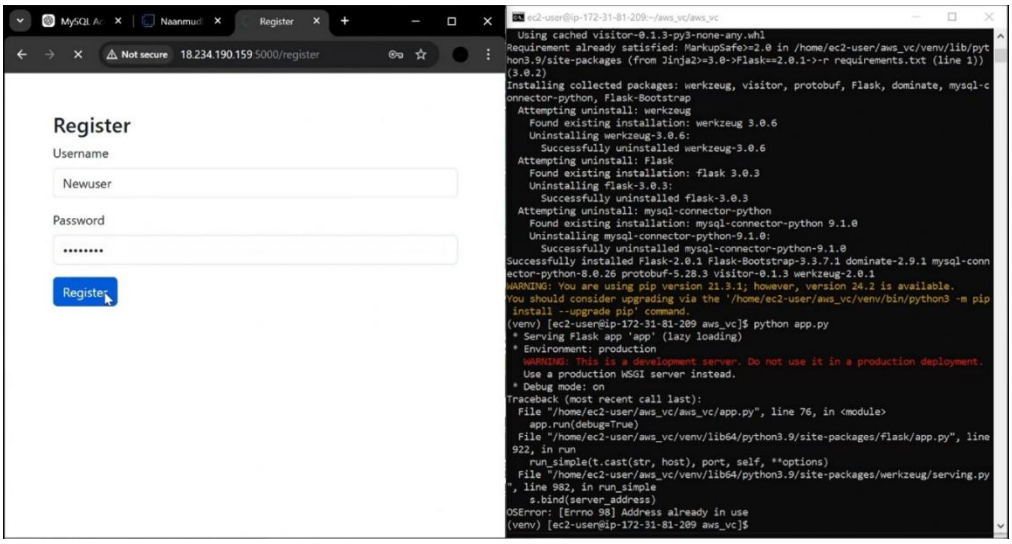
Testing ensures that the FreshBasket platform functions correctly, is scalable, and meets security and performance requirements. This chapter outlines the types of tests conducted, including unit, integration, load, security, and end-to-end testing.

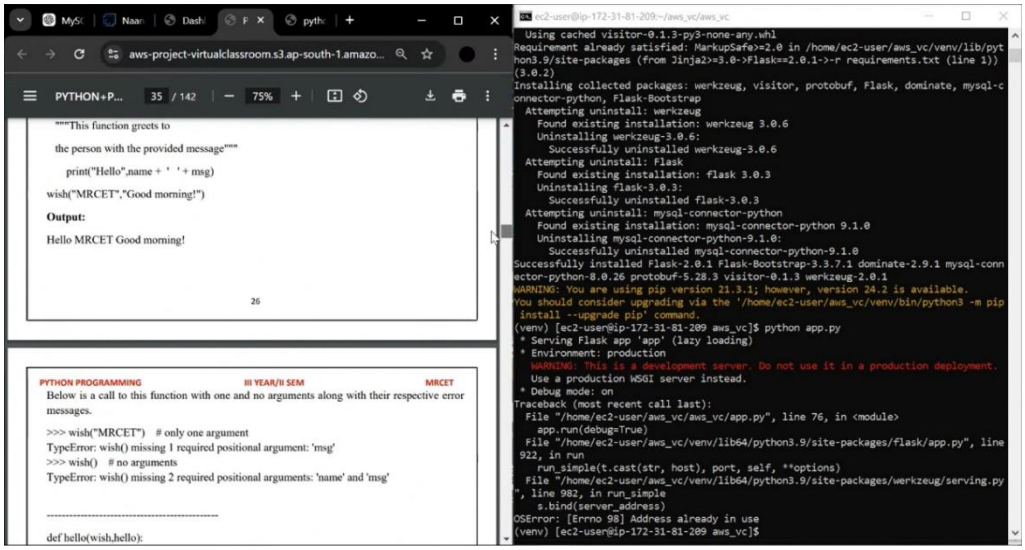
5.2 Types of Testing

1. **Unit Testing:** Focuses on individual components, such as Flask routes and database interactions, to ensure they behave as expected.
   Tools Used: PyTest, unittest.
2. **Integration Testing:** Verifies that the backend, database, and frontend work together seamlessly.
   Tools Used: Postman, MySQL Workbench.
3. **Load Testing:** Simulates high traffic to test platform scalability and performance under stress.
   Tools Used: Apache JMeter, AWS CloudWatch.
4. **Security Testing:** Ensures the platform is resilient to security vulnerabilities like SQL injection and unauthorized access.
   Tools Used: OWASP ZAP, Burp Suite.
5. **End-to-End Testing:** Tests the entire user flow, from registration to order confirmation, to ensure smooth operation.
   Tools Used: Selenium.

CHAPTER 6

TESTING AND OPTIMIZATION





CHAPTER 7

CONCLUSION

The **FreshBasket** project successfully demonstrated the deployment of a scalable e-commerce platform tailored for selling vegetables and fruits, utilizing a robust cloud infrastructure on AWS. By employing **Flask** for backend development, **AWS EC2** for hosting, and **Amazon RDS** for database management, the platform has proven to be efficient, scalable, and highly available, ensuring seamless customer experiences and smooth order processing.

Future Work

While the current implementation covers the core aspects of a scalable e-commerce platform, there are several opportunities for enhancement:

1. **Enhanced User Experience:**
    - Implementing advanced features such as personalized recommendations, product search optimization, and customer reviews could significantly improve user engagement and satisfaction.
    - Mobile application development can expand accessibility and convenience for users.
2. **Security Improvements:**
    - Strengthening the platform's security by integrating more advanced authentication methods like two-factor authentication (2FA) and encryption protocols to safeguard user data and transactions.
3. **AI and Machine Learning Integration:**
    - Introducing AI-based inventory management to predict stock levels and demand fluctuations could optimize operational efficiency.

- Machine learning algorithms could be used for dynamic pricing strategies, analyzing market trends and customer behavior.

4. **Payment Gateway and Multi-Currency Support:**
    - Incorporating multiple payment gateways and multi-currency support will open the platform to global users, enhancing its usability in different markets.
5. **Sustainability Features:**
    - Integration of features such as eco-friendly packaging options and sustainability scorecards for products could align with the growing demand for environmentally conscious business models.
6. **Microservices Architecture:**
    - Transitioning to a microservices architecture can provide more flexibility and maintainability, enabling independent scaling and deployment of platform components.
7. **Blockchain for Transparency:**
    - Blockchain technology could be integrated to provide transparency in product sourcing, especially for organic and sustainable produce, allowing users to trace the origin and freshness of their purchases.

References

- **AWS Account Setup: https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh**
- **Web Application Stack : FLask || MySQL Connector using flask || HTML/JS/CSS**
- **AWS EC2 Instance: https://www.youtube.com/results?search_query=aws+ec2+oneshot**
- **RDS Database: https://www.youtube.com/results?search_query=rds+oneshot**
- **MySQL: https://www.youtube.com/results?search_query=mysql+tutorial**
- **AWS Cost Management: https://youtu.be/OKYJCHHSWb4?si=aY3DQl1v26CfZxXA**


aws academy
KUMAR D
Certificate of Completion for
AWS Academy Graduate - AWS Academy Cloud Foundations
Course hours completed
20 hours
Issued on
10/30/2024
Digital badge
https://www.credly.com/go/yXyomuSD