INFORMATION TECHNOLOGY UNIVERSITY

BS-Software Engineering

Software Development & Construction

AWS Architecture Diagram

Instructor: Dr Zunnurain Hussain **Teaching Assistant: Umair Makhdoom**

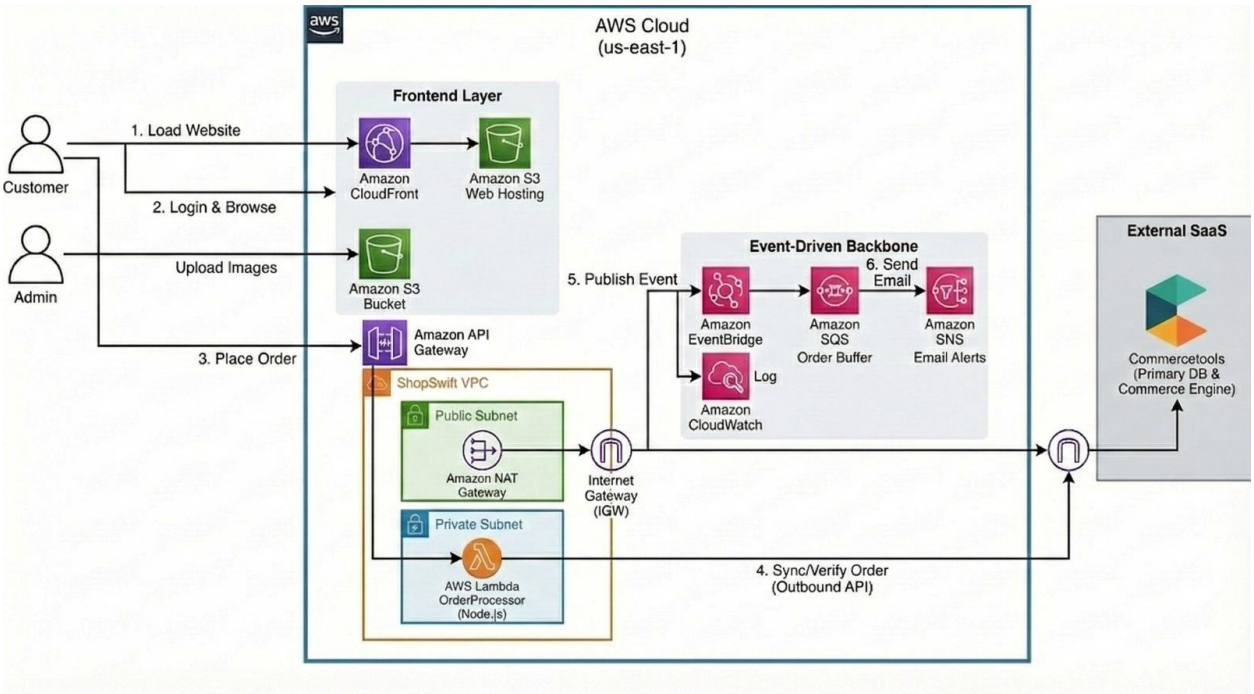
Project Title: Cloud Based E – Commerce Platform

Group Members:

- Muhammad Numan Tahir – BSSE23069
- Muhammad Zubair Akhter – BSSE23079

AWS Architecture Diagram:

A professional, AWS-standard architecture diagram was designed using correct AWS icons and clear data flows. The diagram accurately represents frontend delivery through Amazon S3 and CloudFront, backend processing via API Gateway and Lambda, secure networking using VPC subnets, and event-driven integration with EventBridge, SQS, SNS, and external Commercetools services. All components and interactions are logically connected and easy to understand.



Implementation Steps:

1. An **Amazon S3 bucket** was created to store all website-related static resources, including HTML files, CSS, JavaScript, and product images. S3 was chosen due to its high durability, scalability, and cost-effective storage. This allowed the website content to be securely stored and accessed without the need for managing physical servers.
2. **Amazon CloudFront** was configured with the S3 bucket as its origin to deliver website content to users with low latency. CloudFront uses a global

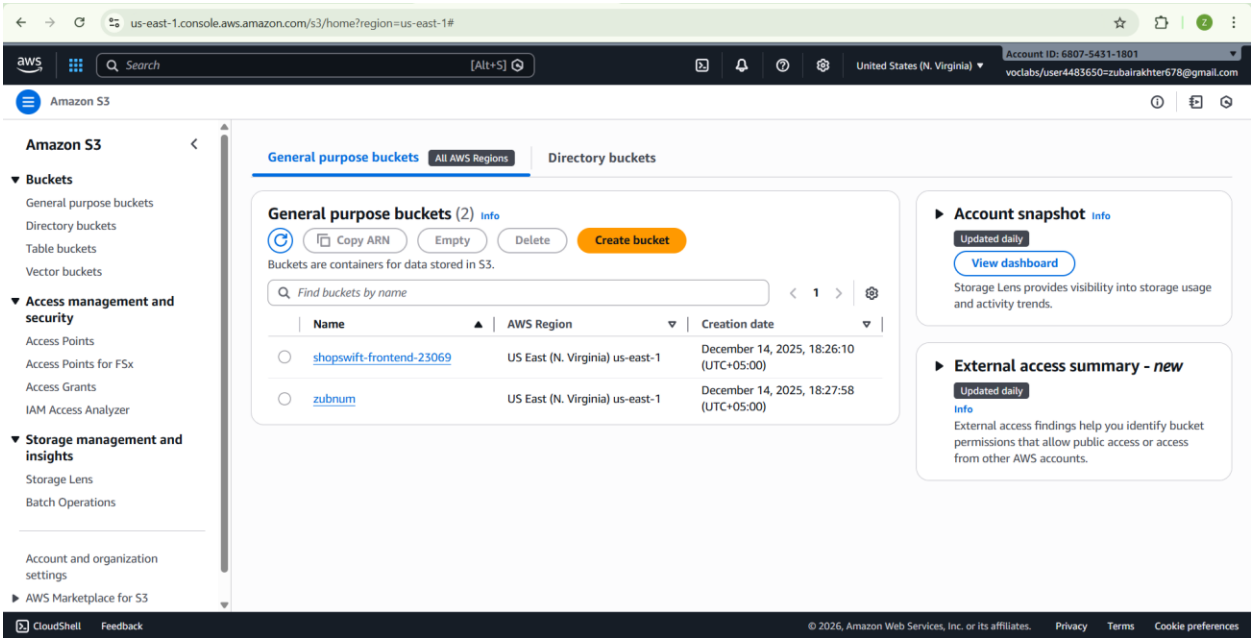
network of edge locations to cache content closer to users, which significantly improved website loading speed and reduced the load on the origin server.

3. A **Virtual Private Cloud (VPC)** was set up with both public and private subnets to ensure network security and isolation. Public subnets were used for services that needed internet access, while private subnets were used for backend services. This separation improved security by limiting direct access to critical components.
4. **Amazon API Gateway** was deployed to act as the main entry point for all backend requests from the frontend. It handled request routing, authentication, and traffic management. API Gateway enabled secure and scalable communication between the frontend application and backend services.
5. Backend business logic was implemented using **AWS Lambda (or Amazon ECS)** within private subnets. Lambda allowed serverless execution of backend code without managing servers, automatically scaling based on demand. This approach reduced operational overhead and ensured high availability during traffic spikes.
6. **Commercetools APIs** were integrated to manage core e-commerce functionalities such as product catalog, shopping carts, order processing, and payments. Using Commercetools provided a headless commerce solution that allowed ShopSwift to scale and update features independently from the frontend.
7. Customer session data and profile information were stored in **Amazon DynamoDB**, a fully managed NoSQL database service. DynamoDB was selected for its low latency, automatic scaling, and high availability, ensuring fast access to user data even during high traffic periods.
8. **Amazon EventBridge** was configured to publish events whenever a new order was placed. This enabled an event-driven architecture where different services could react to order events independently, improving system flexibility and decoupling services.

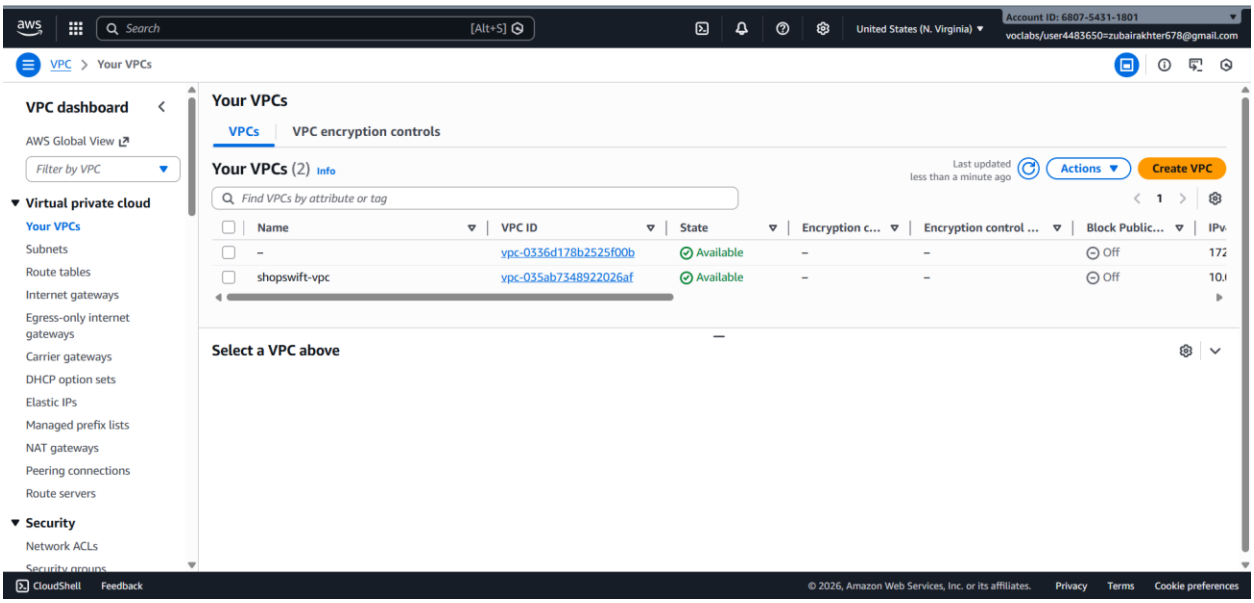
9. **Amazon SQS** was used to buffer order messages to ensure reliable processing, even during high load. At the same time, **Amazon SNS** was used to send notifications such as order confirmations to customers. This combination ensured fault tolerance and reliable message delivery.
10. Finally, **Amazon CloudWatch** was enabled to monitor system performance, collect logs, and track metrics across all services. CloudWatch provided visibility into application health, allowed real-time monitoring, and helped in quickly identifying and resolving issues.

Screenshots:

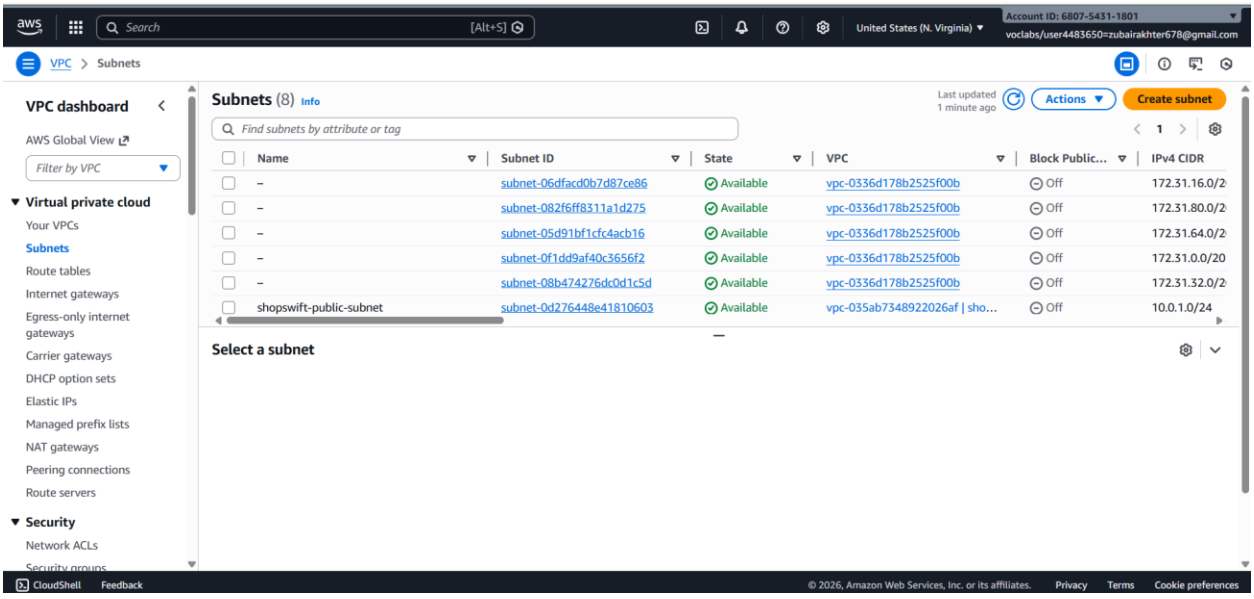
1) S3:



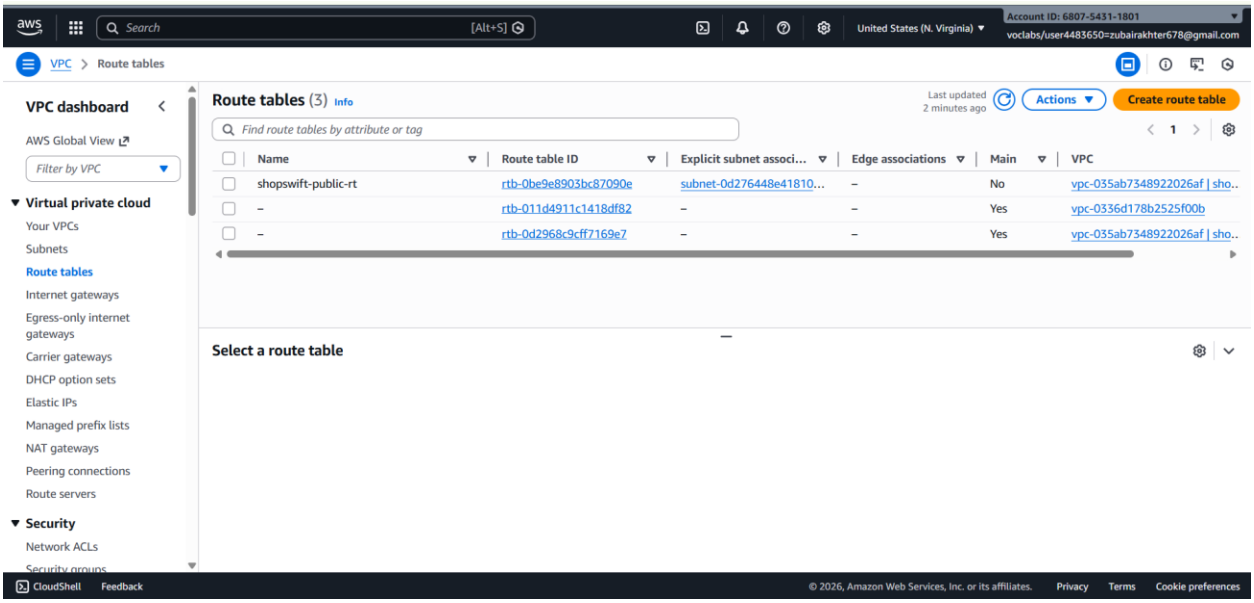
2) VPC



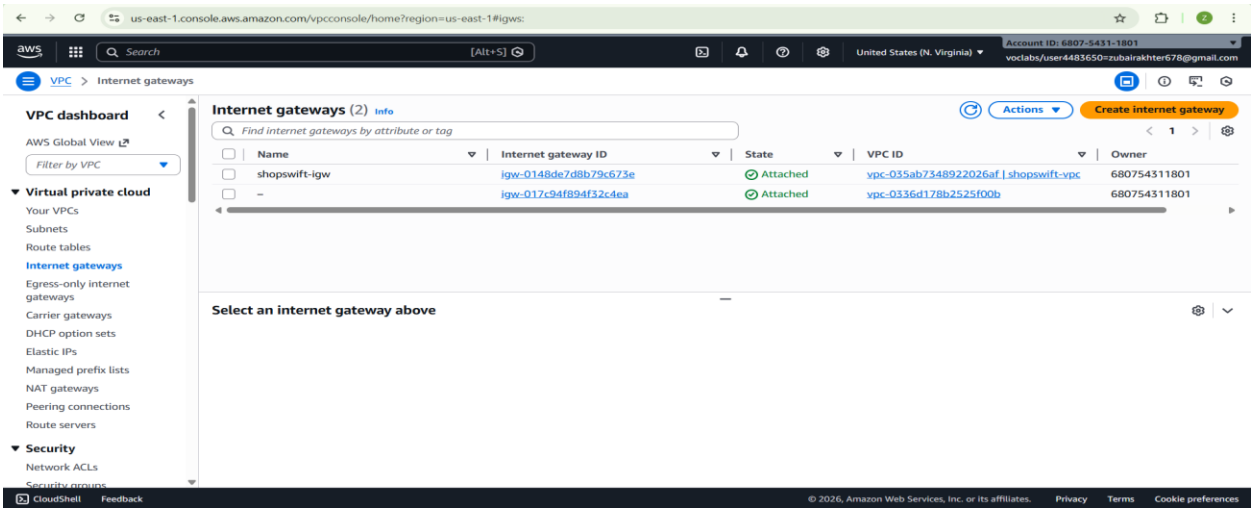
3) Subnets:



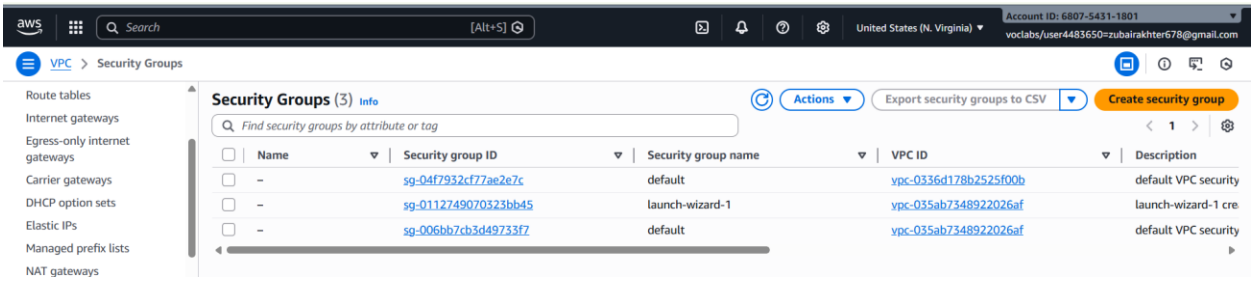
4) Route Tables:



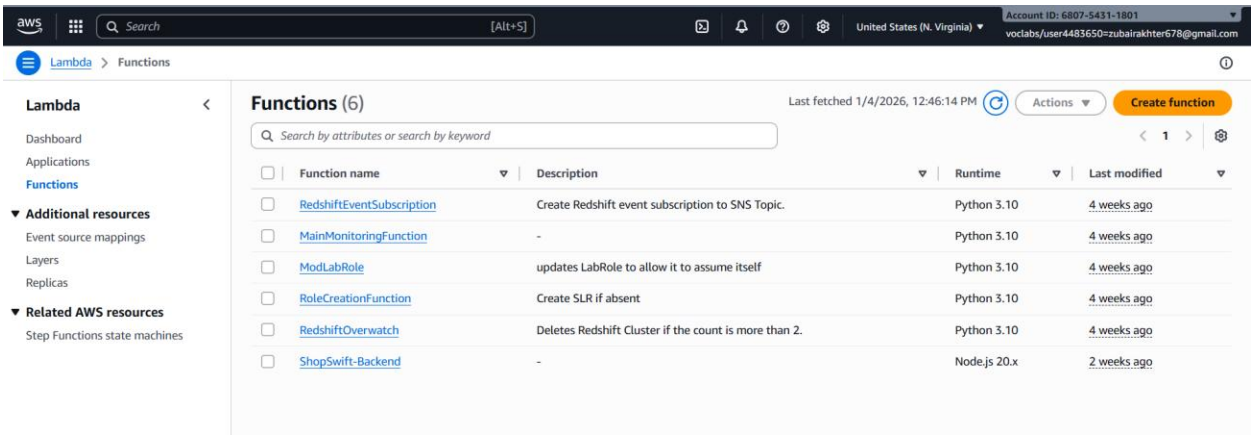
5) Internet Gateway:



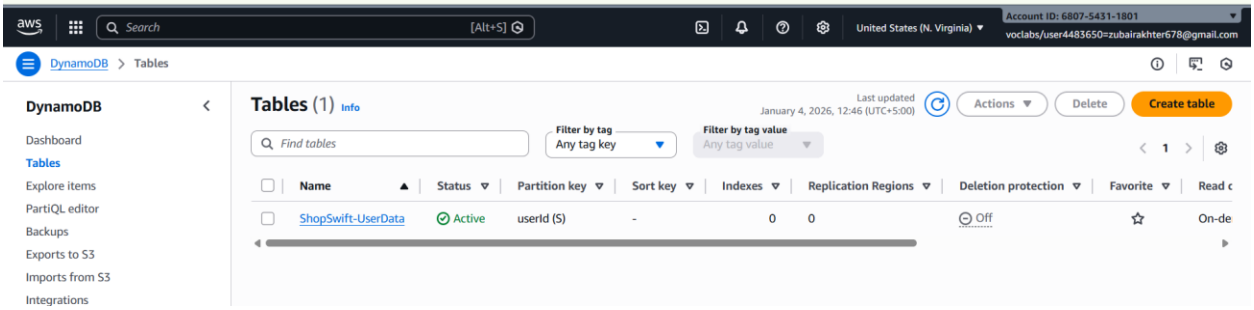
6) Security Groups:



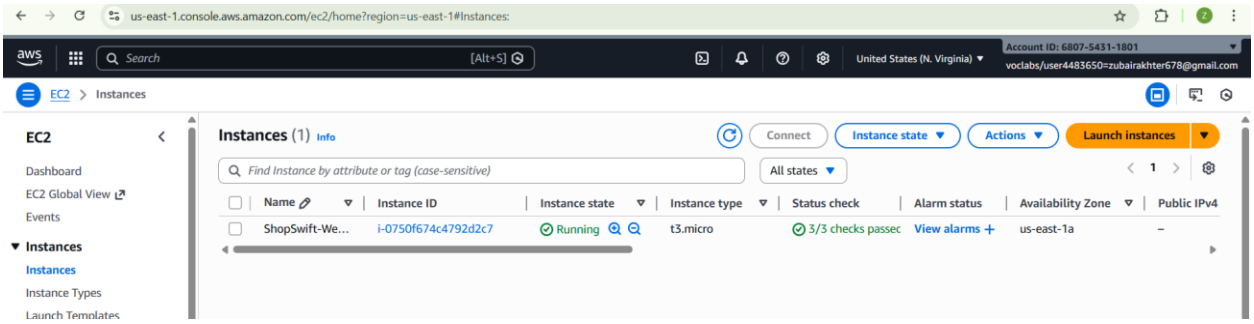
7) Lambda



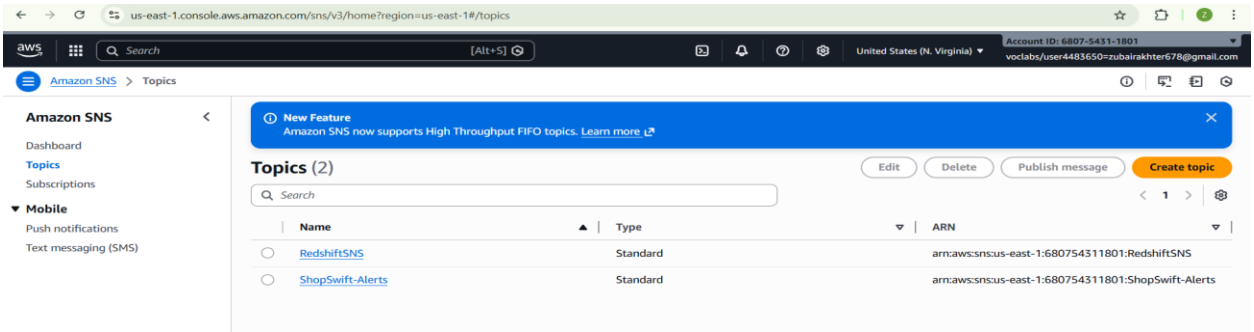
8) Tables (DynomoDB):



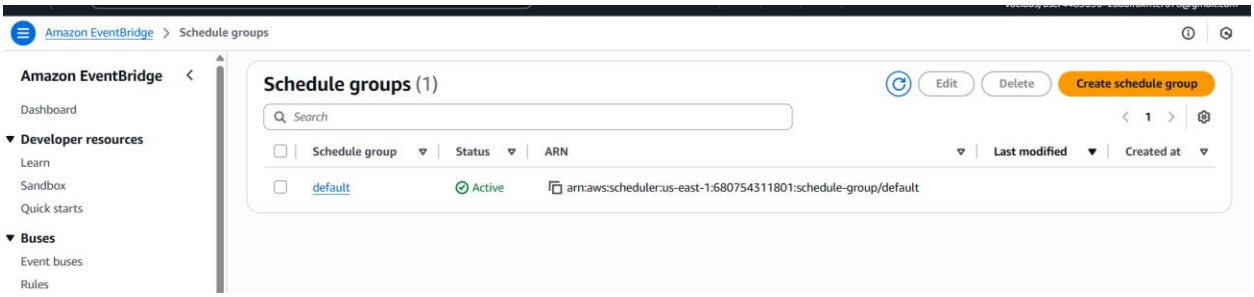
9) EC2 Instances:



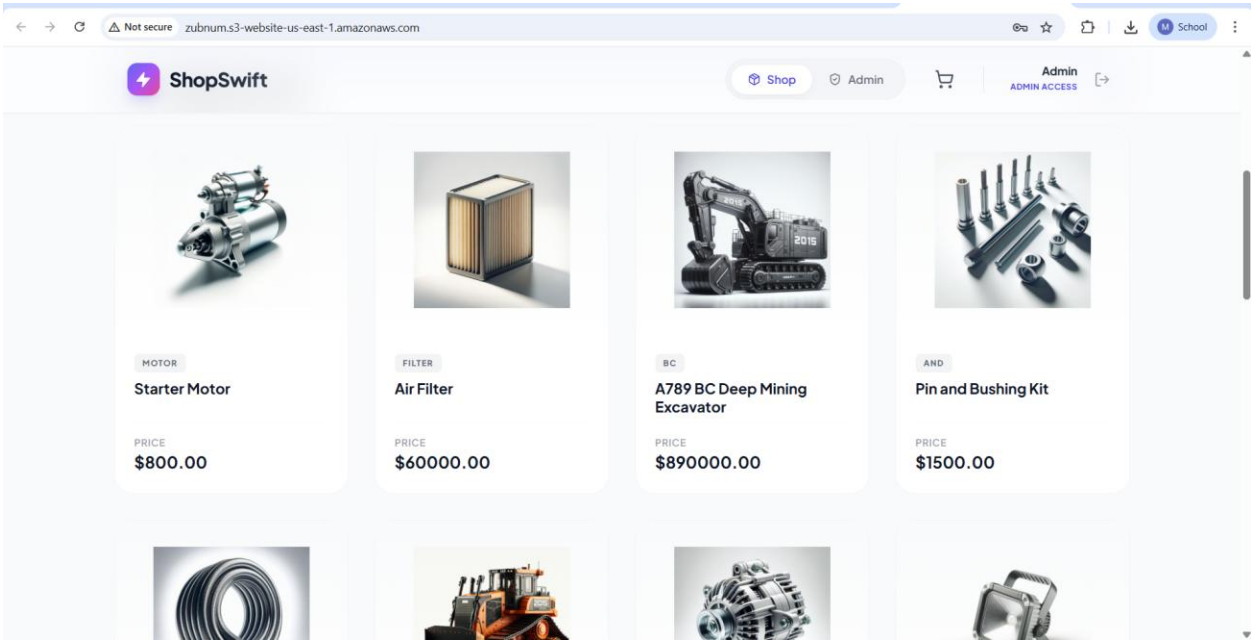
10) SNS



11) Event Bridge:



12) Final Output



Security & IAM:

Security best practices were followed throughout the system. IAM roles were configured using the principle of least privilege, VPC isolation was implemented with public and private subnets, and outbound access was controlled using a NAT Gateway. Logging and monitoring were enabled through CloudWatch, and data access was secured to ensure reliability and compliance.