Designing a Cloud-Based Inventory Management System for Azure's E-Commerce Platform on AWS

CSA1594-Cloud Computing and Big Data Analytics Using Data Centre Network

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

By

S.KAVIYA

192221103

Problem Statement

Design and deploy an Inventory Management System for Azura's e-commerce platform that can scale efficiently with varying user traffic, ensuring high availability, fault tolerance, and optimized performance. The system will leverage AWS services to achieve these goals.

Requirement Analysis Identifying the Specific Requirements Stakeholders

- E-commerce Managers
- Inventory Managers
- Warehouse Staff
- System Administrators
- DevOps Teams
- Business Managers

Key Requirements

- Handle varying levels of user traffic.
- Ensure high availability and fault tolerance.
- Optimize performance and cost.
- Implement robust monitoring and logging. **Determine the Necessary Features**
- **User Authentication and Authorization:** Secure user login and role based access control.
- **Real-time Data Processing:** Stream and process inventory data in real time for quick insights.
- **Load Balancing:** Distribute incoming traffic evenly across multiple servers.
- **Auto-scaling:** Automatically scale resources up or down based on traffic.
- **Database Management:** Efficiently manage structured and unstructured inventory data.

  CI/CD Pipeline: Automate code integration, testing, and deployment.

-

- **Monitoring and Alerting:** Continuously monitor performance and set up alerts for anomalies.
- **Logging and Reporting:** Collect and logs for troubleshooting and reporting.

Architecture Design System Architecture

- **Front-End:** React.js for a responsive user interface.
- **Back-End:** Node.js for handling API requests and business logic. ○ **Primary Database:** Amazon RDS (MySQL) for relational data.
    - ○ **NoSQL Database:** DynamoDB for handling unstructured data.
- **Load Balancer:** AWS Elastic Load Balancer (ELB) to distribute traffic.
- **Auto-Scaling:** AWS Auto Scaling Groups to manage traffic spikes.
- **Containerization:** Docker for packaging and deploying application components.
- **Orchestration:** Kubernetes for managing containerized applications.

Data Ingestion and Processing

- **Data Sources:** Supplier records, sales data, inventory logs.
- **Ingestion Pipeline:** Amazon Kinesis for real-time data streaming.
- **Processing Framework:** AWS Lambda for serverless data processing.

Anomaly Detection

- **Algorithms:** Isolation Forest, DBSCAN for detecting anomalies in inventory patterns.
- **Implementation:** Using Python and Scikit-learn for algorithm implementation.
- **Evaluation:** Confusion matrix, precision, and recall metrics to evaluate performance.

Real-time Monitoring and Alerting

- **Monitoring Tools:** Prometheus for collecting metrics.

- **Alerting Tools:** Grafana for setting up dashboards and alerts.
- **Notifications:** Integrate with Slack and email for real-time notifications.

Visualization and Reporting Design Interactive Dashboards

- **Front-End:** React.js with D3.js for dynamic data visualizations.
- **Back-End:** Node.js to serve data to the front-end.

Use Cloud-native Analytics and Visualization Services

- **Service:** AWS Quick Sight for advanced analytics and visualization.
- **Features:** Customizable dashboards, secure access, integration with AWS data sources.

Cloud Integration and Deployment

- **Cloud Provider:** AWS.
- **Infrastructure Setup:** ○ **EC2 instances:** For compute resources. ○ **S3:** For object storage.
    - ○ **VPC:** For networking.
- **Containerization:** Docker for creating and managing container images.
- **Orchestration:** Kubernetes for deploying and managing containers.
- **CI/CD Pipeline:** Jenkins for continuous integration and delivery.

Testing

- **Unit Testing:** Jest for front-end testing, Mocha and Chai for back-end testing.
- **Integration Testing:** Ensure seamless interaction between front-end, back-end, and database.
- **Load Testing:** Apache JMeter to simulate user load and test system scalability.

Performance Evaluation

- **Metrics:** Latency, throughput, error rates.
- **Tools:** Prometheus and Grafana for real-time performance monitoring.

Optimization: Identify and resolve performance bottlenecks.

Conclusion

- **Summary:** Successfully designed and deployed a scalable Inventory Management System for Azura's e-commerce platform on AWS.
- **Challenges:** Managing real-time data processing, ensuring high availability and scalability.
- **Key Learnings:** Importance of a robust architecture, effective use of cloud services.
- **Future Improvements:** Incorporate machine learning for predictive analytics, extend monitoring to additional infrastructure components.