Cloud Architecture Design Document

Overview

The **Cloud-Based Retail Inventory Management System** is built on **Amazon Web Services (AWS)** to provide a scalable, secure, and cost-effective solution for managing inventory, tracking sales, and generating real-time analytics. The system leverages a combination of AWS services, including **EC2**, **S3**, **RDS**, **Lambda**, **VPC**, and **CloudWatch**, to deliver a robust and efficient cloud infrastructure.

System Architecture

The system architecture is divided into the following components:

1. **Frontend**: A web-based dashboard hosted on an EC2 instance for real-time inventory and sales analytics.
2. **Backend**: Python scripts and Lambda functions for automating inventory updates, sales reporting, and low-stock alerts.
3. **Database**: A relational database (RDS) for storing inventory, sales, and user data.
4. **Storage**: S3 buckets for storing inventory and sales data, as well as backups.
5. **Networking**: A VPC with public and private subnets for secure communication between components.
6. **Monitoring**: CloudWatch for monitoring system performance and setting up alarms.

AWS Services Used

1. Amazon EC2

- **Purpose**: Hosts the web application and Python scripts for inventory management.
- **Justification**: EC2 provides scalable compute capacity, allowing the system to handle varying loads. Auto Scaling groups ensure high availability and performance.

2. Amazon S3

- **Purpose**: Stores inventory and sales data, as well as backups.

- **Justification**: S3 is highly durable and scalable, making it ideal for storing large volumes of data. It also integrates seamlessly with other AWS services like Lambda and RDS.

3. Amazon RDS

- **Purpose**: Manages a relational database for inventory, sales, and user data.
- **Justification**: RDS was chosen over DynamoDB because the system requires complex queries and relationships between tables (e.g., products, inventory, sales). RDS supports SQL-based databases like MySQL and PostgreSQL, which are better suited for this use case.

4. AWS Lambda

- **Purpose**: Executes serverless functions for automating inventory updates, sales reporting, and low-stock alerts.
- **Justification**: Lambda eliminates the need to manage servers, reducing operational overhead. It is cost-effective for event-driven tasks and scales automatically.

5. Amazon VPC

- **Purpose**: Provides a secure and isolated network environment for the system.
- **Justification**: VPC allows for fine-grained control over network traffic, ensuring that sensitive data is protected. Public and private subnets are used to separate web-facing components from backend services.

6. Amazon CloudWatch

- **Purpose**: Monitors system performance, sets up alarms, and logs events.
- **Justification**: CloudWatch provides real-time insights into system health, enabling proactive troubleshooting and optimization.

Network Configuration

Virtual Private Cloud (VPC)

- **CIDR Block**: 10.0.0.0/16
- **Subnets**:
  - **Public Subnets**: 10.0.1.0/24 and 10.0.2.0/24 (for EC2 instances and web-facing components).
  - **Private Subnets**: 10.0.3.0/24 and 10.0.4.0/24 (for RDS and Lambda functions).
-
- **Internet Gateway**: Attached to the VPC for internet access.

- **Route Tables**: Configured to route traffic between subnets and the internet.

Security Groups

- **EC2 Security Group**: Allows SSH (port 22) and HTTP (port 80) traffic.
- **RDS Security Group**: Allows inbound traffic from EC2 instances and Lambda functions.
- **Lambda Security Group**: Allows outbound traffic to RDS and S3.

Security Measures

1. IAM Roles and Policies

- **EC2 Role**: Grants permissions to access S3, RDS, and CloudWatch.
- **Lambda Role**: Grants permissions to access S3, RDS, and CloudWatch.
- **RDS Role**: Grants permissions for automated backups and encryption.

2. Encryption

- **S3**: Server-Side Encryption (SSE-S3) for data at rest.
- **RDS**: Encryption enabled for the database.
- **Data in Transit**: SSL/TLS encryption for communication between components.

3. Security Groups

- Restrict access to EC2 instances and RDS databases to only trusted IP addresses.
- Use private subnets for backend services to minimize exposure to the internet.

Scalability and Cost Optimization Strategies

1. Scalability

- **EC2 Auto Scaling**: Automatically adjusts the number of EC2 instances based on traffic.
- **RDS Read Replicas**: Improves database performance by offloading read queries.
- **Lambda**: Scales automatically to handle varying workloads.

2. Cost Optimization

- **Right-Sizing EC2 Instances**: Use instance types that match the workload requirements.
- **Reserved Instances**: Purchase reserved instances for long-term cost savings.

- **S3 Lifecycle Policies**: Move infrequently accessed data to S3 Glacier for cost savings.
- **AWS Cost Explorer**: Monitor and optimize cloud expenses.

Conclusion

The **Cloud-Based Retail Inventory Management System** leverages AWS services to provide a scalable, secure, and cost-effective solution for retail businesses. The architecture is designed to handle varying workloads, ensure data security, and optimize costs. By using a combination of EC2, S3, RDS, Lambda, VPC, and CloudWatch, the system delivers real-time inventory tracking, automated stock replenishment, and detailed sales analytics.

Diagrams

Architecture Diagram

![alt text](architecture_diagram.png)

References

- AWS Documentation
- AWS Well-Architected Framework