Project: AWS Cloud Sales Application

Group Members:

Tareq Md Rabiul Hossain CHY

Nushrat JAHAN

Alaleh Mohammadi GOLRANG

Group Number: 4

Submission Date:

31/12/2023

Course Title:

Cloud and Edge Infrastructures

Instructor:

Luis Gustavo Nardin

AWS Cloud Sales Application

Introduction

In this project, we have developed an innovative solution tailored for a large retailer with stores dispersed globally, including the headquarters in Saint-Étienne, to streamline the summarization and consolidation of their daily sales data. Recognizing the challenges associated with managing vast amounts of data from various geographical locations, our solution capitalizes on the power and flexibility of Amazon Web Services (AWS) to create a highly scalable and efficient cloud-based infrastructure.

Our solution architecture is comprised of three core applications: the Client, Worker, and Consolidator applications, each designed to play a critical role in the data processing pipeline. This system is engineered to handle the daily influx of diverse sales data from numerous retail outlets, simplifying the process of analyzing and interpreting this information for effective decision-making.

Solution Architecture Overview

Solution 1: Traditional Server-Based Approach

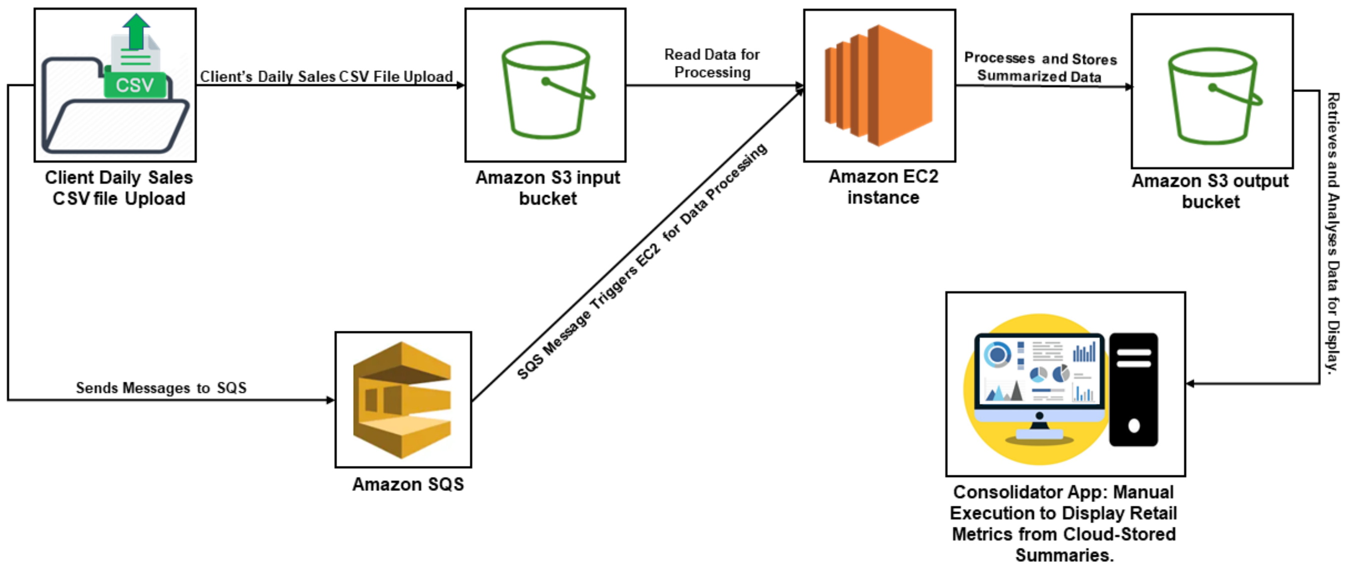
CSV
Client Daily Sales CSV file Upload
Client's Daily Sales CSV File Upload
Amazon S3 input bucket
Read Data for Processing
Amazon EC2 instance
Processes and Stores Summarized Data
Amazon S3 output bucket
Retrieves and Analyses Data for Display.
SQS Message Triggers EC2 for Data Processing
Sends Messages to SQS
Amazon SQS
Consolidator App: Manual Execution to Display Retail Metrics from Cloud-Stored Summaries.

Figure 1: Traditional Server-Based Solution Architecture Using EC2 and Amazon SQS

1. **Client Application**:
    - *Purpose*: Manage daily sales data for each store.
    - *Functionality*:
        - Uploads daily sales data in CSV format to Amazon S3 input bucket.
        - Notifies the Worker application via Amazon SQS.
    - *AWS Services Used*: Amazon S3 bucket, Amazon SQS.
2. **Worker Application (Java on EC2)**:
    - *Java Application on EC2* : Triggered by SQS messages, runs on an Amazon EC2 (Elastic Compute Cloud) instance. A traditional server-based approach.

- *Purpose*: Process and summarize sales data.
- *Functionality*:
  - Triggered by messages from Amazon SQS.
  - Reads CSV files from the Amazon S3 input bucket.
  - Summarizes data by store and product.
    - *By Store* : Total profit.
    - *By Product* : Total quantity sold, total sales, total profit.
  - Saves summarized data in a new CSV file in an Amazon S3 output bucket.
  - Ensures robust processing, handling cases of unavailability or high load.
- *Resilience*: Handles unavailability or high load by ensuring eventual processing of sales files.
- *AWS Services Used*: Amazon EC2, Amazon S3 bucket, Amazon SQS.

3. **Consolidator Application**:
   - *Purpose*: Aggregates the summarized data. Compiles and displays comprehensive sales information.
   - *Functionality*:
     - Manually executed by an operator.(Operator will provide a date as an input)
     - Takes a date as an argument to identify which files to process.
     - Reads summary results from Amazon S3 output bucket and displays data such as total retailer profit, most/least profitable stores, and total sales/profit per product.
   - *AWS Services Used*: Amazon S3 bucket.

Solution 2: Serverless Approach

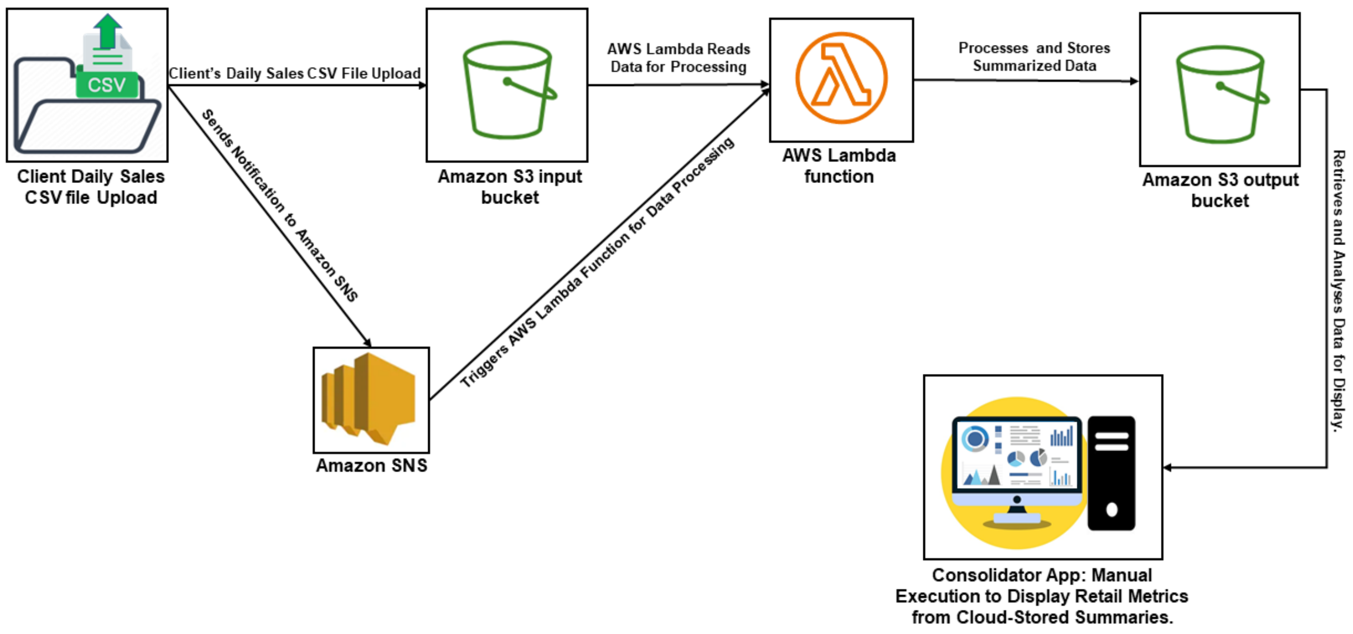
CSV
Client Daily Sales CSV file Upload
Client's Daily Sales CSV File Upload
Amazon S3 input bucket
AWS Lambda Reads Data for Processing
AWS Lambda function
Processes and Stores Summarized Data
Amazon S3 output bucket
Sends Notification to Amazon SNS
Triggers AWS Lambda Function for Data Processing
Amazon SNS
Retrieves and Analyses Data for Display.
Consolidator App: Manual Execution to Display Retail Metrics from Cloud-Stored Summaries.

Figure 2: Serverless Approach Solution Architecture Using AWS Lambda Function and Amazon SNS

1. **Client Application**:

    - *Purpose*: Same as in Solution 1.
    - *Functionality*:
        - Identical to Solution 1, but notifies Worker application via Amazon SNS.
    - *AWS Services Used*: Amazon S3 bucket, Amazon SNS.

2. **Worker Application (AWS Lambda Function)**:

    - *AWS Lambda Function* : Automatically triggered by SNS notifications. A serverless option that automatically triggers upon notification.
    - *Purpose*: Same as in Solution 1.
    - *Functionality*:
        - Automatically triggered by SNS notifications.
        - Reads the uploaded CSV file from Amazon S3 input bucket.
        - Summarizes data by store and product.
            - *By Store* : Total profit.
            - *By Product* : Total quantity sold, total sales, total profit.
        - Saves the summarized data in a new CSV file in the Amazon S3 output bucket.
        - Ensures robust processing, handling cases of unavailability or high load.
    - *Resilience*: Built-in scalability and fault tolerance.
    - *AWS Services Used*: AWS Lambda, Amazon S3 bucket, Amazon SNS.

3. **Consolidator Application**:

    - *Purpose* and *Functionality*: Identical to Solution 1.
    - *AWS Services Used*: Amazon S3 bucket.

Justification of Proposed Solution Architecture and Chosen AWS Services

Rationale Behind the Solution Architecture

1. **Modular Design with Three Core Applications (Client, Worker, Consolidator)**:

    - *Justification*: This design facilitates specialized handling of distinct stages in the data processing pipeline, enhancing efficiency and manageability. Each application focuses on specific tasks, allowing for optimization and scalability tailored to those functions.

2. **Global Scalability and Efficiency**:

    - *Justification*: Considering the retailer's global presence, the architecture is designed to handle large volumes of data from various locations efficiently. The cloud-based approach enables scaling according to demand, ensuring that the system can manage peak loads without unnecessary overhead during quieter periods.

3. **Flexibility in Approach (Traditional Server-Based vs. Serverless)**:

    - *Justification*: Providing two architectural solutions offers flexibility. The traditional server-based approach suits scenarios where more control over the computing environment is required, while the serverless approach is ideal for unpredictable workloads, reducing operational costs and complexity.

Justification for Specific AWS Services

1. **Amazon EC2 (for Java Application in Worker - Traditional Approach)**:
    - *Justification*: EC2 provides a flexible and controlled computing environment, necessary for complex or legacy applications that may not be easily adapted to a serverless architecture. It also offers the ability to customize the computing, storage, and networking configurations as per the application's specific needs.
2. **AWS Lambda (for Worker Application in Serverless Approach)**:
    - *Justification*: Lambda is chosen for its scalability and cost-effectiveness. It allows the Worker application to automatically scale with the incoming workload and incurs costs only when the function is running, making it ideal for varying workloads.
3. **Amazon S3 Bucket (for Data Storage)**:
    - *Justification*: S3 is used for its scalability, durability, and security in storing large volumes of data. It's an ideal choice for storing both input (daily sales CSV files) and output (summarized data), providing a centralized and secure data repository.
4. **Amazon SQS (for Message Queueing in Traditional Approach)**:
    - *Justification*: SQS ensures reliable delivery and decoupling of the Client and Worker applications. It manages message queues, which helps in maintaining the integrity of data processing even in the event of component failures or high load.
5. **Amazon SNS (for Notifications in Serverless Approach)**:
    - *Justification*: SNS is suitable for triggering the Lambda-based Worker application, especially in a serverless architecture. It efficiently handles message broadcasting to multiple subscribers, which is useful when scaling the application to handle high volumes of data.

Additional Considerations in Service Selection

1. **Scalability Concerns**:
    - *Justification*: Both AWS Lambda and EC2 were selected with scalability in mind. Lambda offers automatic scaling, and EC2 provides the ability to manually scale as per the application's demand.
2. **Cost-Effectiveness**:
    - *Justification*: The serverless architecture, particularly AWS Lambda, provides a cost-effective solution for handling variable workloads due to its pay-per-use pricing model.
3. **Data Security and Compliance**:
    - *Justification*: Ensuring data security and compliance is a priority. AWS services like S3 offer robust security features and compliance with various data protection regulations, which is essential for handling sensitive retail sales data.

4. **Monitoring and Logging with AWS CloudWatch**:

    - *Justification*: CloudWatch is recommended for its comprehensive monitoring and logging capabilities. It enables real-time monitoring of AWS resources and applications, helping in quick identification and resolution of issues.

Quantitative and Qualitative Comparison of Worker Application: AWS Lambda vs. Java on EC2

Quantitative Comparison

1. **Performance Metrics**:

    - *EC2-Based Java Application*: Performance depends on the EC2 instance type chosen. Metrics like CPU utilization, memory usage, and network throughput are key indicators.
    - *AWS Lambda*: Performance is measured in terms of execution time and memory usage. AWS Lambda automatically scales, but it has execution time and memory limits per function.

2. **Cost**:

    - *EC2-Based Java Application*: Costs are incurred based on the instance type, storage, and network resources used, regardless of whether the application is actively processing data.
    - *AWS Lambda*: Costs are based on the number of requests and the duration of code execution. This pay-per-use model can be more cost-effective for intermittent workloads.

3. **Scalability**:

    - *EC2-Based Java Application*: Scalability is manual or auto-scaled within the limits of the EC2 instance types.
    - *AWS Lambda*: Automatically scales with the number of incoming requests, offering high scalability without manual intervention.

Qualitative Comparison

1. **Ease of Management**:

    - *EC2-Based Java Application*: Requires more hands-on management for scaling, patching, and securing the server environment.
    - *AWS Lambda*: Requires minimal management as it abstracts away the server and operating system maintenance.

2. **Development and Deployment Complexity**:

    - *EC2-Based Java Application*: Potentially more complex due to the need to manage the entire application stack and deployment process.
    - *AWS Lambda*: Simplifies deployment with a focus on code alone, reducing the operational burden.

3. **Flexibility and Control**:

    - *EC2-Based Java Application*: Offers more control over the computing environment, which is beneficial for complex applications or specific compliance requirements.
    - *AWS Lambda*: Provides less control over the environment but offers greater flexibility in scaling and operation.

4. **Resilience and Fault Tolerance**:

    - *EC2-Based Java Application*: Requires explicit strategies for handling unavailability and high load.
    - *AWS Lambda*: Offers built-in fault tolerance and handles high load scenarios seamlessly.

5. **Startup Time**:

    - *EC2-Based Java Application*: Longer startup times due to the need to boot up an EC2 instance.
    - *AWS Lambda*: Typically has faster startup times, especially relevant for short and intermittent tasks.

6. **Integration with AWS Services**:

    - Both the EC2-based Java application and AWS Lambda offer robust integration with other AWS services like S3, SQS, and SNS. However, Lambda might have an edge in serverless architectures due to its native integration and automatic triggering capabilities.

Conclusion

- **Solution 1** is less efficient in terms of cloud resource usage due to the continuous operation of EC2 instances.Continuous operation of EC2 instances might not be ideal for companies looking to minimize their cloud footprint or manage costs tightly.
- **Solution 2** is more efficient, especially with its use of AWS Lambda, reducing the need for continuously running resources. The serverless architecture aligns well with companies aiming for cost-effectiveness, scalability, and reduced operational overhead.

Based on the requirement to use the least amount of cloud services, **Solution 2** is a better fit. However, further optimizations, particularly around the use of EC2 for the Client and Consolidator applications, could enhance its efficiency.