Cloud and Edge Infrastructures

Cloud Solution to summarize and consolidate the data of the daily sales

Professor: Luis Gustavo Nardin

BY: Sara Assefa ALEMAYEHU
Soumya KUMBAR &

Date: 07/11/2022

Introduction

In this project, we provide a solution to summarize and consolidate the data of the daily sales generated by the stores of a large retailer. The stores are geographically distributed all over the world and the headquarter is located in Saint-Étienne. Everyday in the stores, they sell different products and they will have different profits on those products, we provide a solution to summarize and consolidate the data of the daily sales generated by the stores. The retailer will use a Pubic Cloud infrastructure rather than purchase, install, and maintain all the required infrastructure, for this we use AWS services to provide a solution.

General architecture of proposed solution

We proposed two solutions first solution includes :

- Client application
- Worker as java application
- Consolidator

In the first solution, every retailer store uploads daily sales(CSV) file using a client application to the S3 bucket and sends a message to the Simple Queue Service. A java worker application will be hosted in Amazon EC2 instance and get triggered by the SQS to process the file, then the consolidator will be run manually to display the summery data of the daily sales generated by the stores.

The second solution includes:

- Client application
- Worker lambda function
- Consolidator

The second solution will go through all the steps of the first one but here we use lambda function for worker application and this application will be triggered by SNS then consolidator will be run manually.

The proposed solution architecture used the following AWS services:

- Amazon EC2
- AWS Lambda

- Amazon S3
- Amazon SQS
- Amazon SNS

Proposed solution architecture and chosen AWS services

Fig1 : The first proposed solution general picture

In this architecture, we created three Java maven projects client, worker, and consolidator. To store the files we created two buckets, one to upload the daily sales CSV files and the other to store summary files CSV. We are using two separate buckets to avoid looping issues and in case of large data, this is the best approach to have a more stable system. Then after we created one SQS queue to store (pass) messages from the client application of several different stores to the worker java application.

Amazon EC2: Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, re-sizable compute capacity in the cloud. It makes computing easier for us and allows us to obtain and configure capacity with minimal friction. EC2 setup involves creating an Amazon Machine Image (AMI), which includes an operating system, apps, and configurations. That AMI is loaded to the Amazon Simple Storage Service (S3), and it's registered with EC2, at which point users can launch virtual machines as needed. Also, it can be created in

minutes, and since it is well-suited for applications that experience hourly, daily, or weekly variability in usage we use EC2.

Amazon S3: Every object in Amazon Simple Storage Service (Amazon S3) is stored in a bucket thus we created a bucket and uploaded the daily sales CSV file using a client application, the file stored in Amazon S3 is retrieved by worker java application which is hosted on EC2 instances in real time after an upload successfully done and the worker application get the message from created SQS , i.e Client application uploads the daily file passed as a parameter into the Cloud storage and notify the Worker application by SQS, this application is used independently by each store. In general, we used the S3 service to create buckets to store the input and output files we proposed this because the system can retrieve any amount of data at any time, from anywhere, and for the application that we are working on this is very important that knowing we will not face memory problem in a large scale.

Amazon SQS: we used SQS Service to create queues to store messages until any of our applications processes them. We created one queue in our project and used it to trigger the worker java application and to use the least amount of Cloud storage resources possible we deleted the message after that. The advantage of using SQS is that we can implement FIFO logic in case of several file arrival.

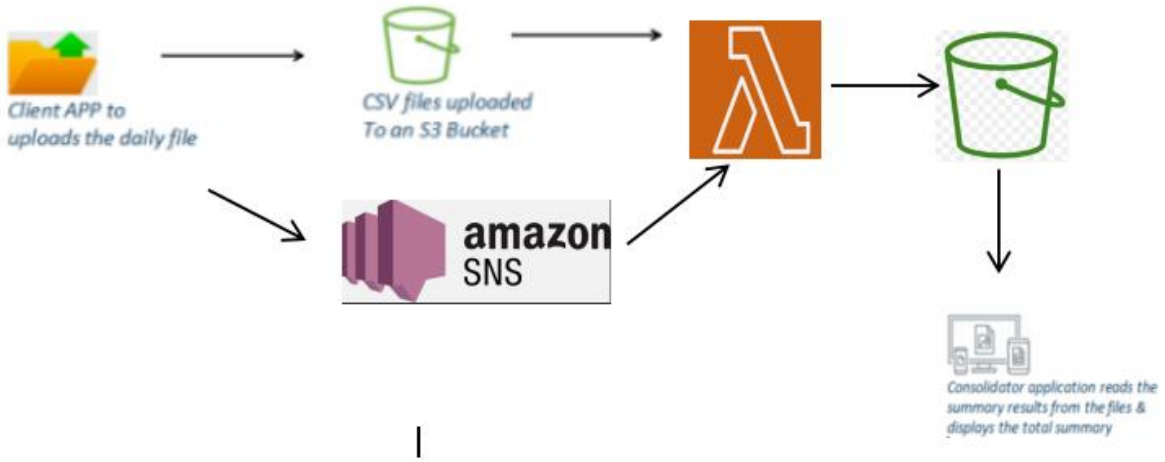

Fig1 : The second proposed solution general picture

In the second proposed solution, we created three Java maven projects as the first one client, worker and consolidator. To store the files we use the two buckets that we created, one to upload the input CSV files (this application should be run once and it will trigger both worker applications) and the other to store output CSV. After we created one SNS to worker lambda function, then after once the lambda function gets triggered it will do all the calculations and give the output CSV in the second bucket to be processed by the consolidator.

AWS Lambda: AWS Lambda is a serverless, event-driven compute service that lets us run code for virtually any type of application or back-end service without provisioning or managing servers, also it allows us to add custom logic to AWS resources such as Amazon S3 buckets. So you can easily apply it to compute data as it enters or moves through the cloud. In our case, we use this Lambda function to implement a worker application that reads the CSV file and summarizes the daily sales by store and by-product.

Amazon SNS: Amazon Simple Notification Service (Amazon SNS) is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication. In our case, we use Amazon SNS to publish messages by the Client application when it uploads the file to S3 bucket. Then the worker lambda function is configured to be triggered by this Simple Notification Service.

Quantitative and qualitative comparison

Worker implementation as Lambda function: Lambda is very good for many types of development. It is perfect for event-driven programming like in our case, the environment is multi-code capable, it is server-less architecture, and can easily handle micro-service architecture, also by combining AWS Lambda with other AWS services it is possible to create secure, stable, and scalable online experiences since we can trigger Lambda from AWS services and software as a service (SaaS) applications. To do the calculation in lambda the worker is taking

nearly 20 seconds. Setting up the Lambda and management of the environment is easier since it is fully it is completely automated than EC2 application setup. In general, our Lambda function is a piece of code that is executed whenever it is triggered by an event from an event source.

Worker implementation as Java application: For the java application we need to provisioning or managing servers and in our work, we use EC2 instance for that. To do the calculation in the worker java application it is taking 10 seconds and comparing it with the lambda function worker java application is faster, even if creating EC2 is not difficult to work managing EC2 is not, setting up includes logging in via SSH and manually installing Apache. Along with that, we need to install and configure all the required software(e.g putty) also in the virtual machine we need to terminate the EC2 instance manually after use.