Cloud and Edge Infrastructures - Practical Work

Solution to summarize and consolidate the data of the daily sales

Cyber-Physical Social Systems (CPS2 - M2)

Submitted by

Sivaratnam PACHAVA

Randika Wishwani Rathwaththa RATHWATHTHAGE

Professor

Luis Gustavo Nardin

Assistant Professor at École des Mines de Saint-Étienne, Saint Etienne

1 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. Everyday in the stores, it has sales on different products by different qualities and they receive different profits on those products. It is difficult to analyze everyday profits manually. We use AWS services to provide a solution to summarize the data of daily sales. In this document, we explained which AWS services are used and how we generate a solution for summarizing the data of the daily sales.

2 Proposed Solution Architecture

We proposed two solutions to get output values from the sales files. We created three applications to provide solutions to the sales files.

1. Client
2. Worker
3. Consolidator

We created two separate projects to run Worker application as a Lambda function and as a Java application. We used different AWS Services while running Worker in two different ways.

3 AWS Services

By using Aws Services we can easily select the operating system, programming language and other services we need in our work. In our project we used below AWS Services,

1. Amazon EC2
2. Amazon S3
3. Amazon Lambda
4. Amazon SQS

3.1 Amazon EC2

We created one EC2 Instance to run Worker as a Java application. EC2 Instance offers virtual servers to manage any kind of workload.

3.2 Amazon S3

We used the S3 service to create buckets to store the input and output files. In our project solution, We created totally three buckets to store csv files. We used one bucket to store output files and two buckets to store input files (one for worker as a java application and the other one for worker as a Lambda application).

3.3 Amazon Lambda

We created one Lambda function to run worker file. AWS Lambda allows you to add custom logic to AWS resources such as Amazon S3 buckets. So you can easily apply compute to data as it enters or moves through the cloud.

3.4 Amazon SQS

We used SQS Service to create queues to stores messages until any of our application process them. We created two queues in our project and used both while worker is running as a Java Application, Used only one queue while worker is running as a Lambda function.

4 Proposed solution

In this project we have implemented solution in two ways according to the worker application.

4.1 Client, Worker as a Lambda Function and Consolidator

In this solution we implemented worker as a Lambda function and developed the solution as mentioned in below steps.

1. Created three Java maven projects such as for the client (filename:ProjectClient), the worker (filename:HandlerS3) and for the consolidator (filename:Consolidator).
2. Created two buckets, one to upload the input csv files (bucketname:uploadbucket) and the other to store output csv files (bucketname:summary-bucket). We are using two separate buckets to avoid looping issue.
3. Created one SQS queue named with OutputFifo.fifo to store/pass messages from worker to client.
4. Created one lambda function to implement worker.

4.1.1 Worker

- We package the Worker code using mvn-package in the root directory of the project in terminal to generate a Jar file. It's created that jar file and placed it in inside the target folder.
- Placed triggering in our lambda function by putting source as S3 with suffix name .csv.
- Deploy the worker file by uploading generated Jar file in to the lambda function.
- So the code receives a S3 Event every time an .csv file is uploaded into the S3 bucket.
- Once it received, the code retrieves the bucket and file names, then downloads and processes the file content and calculates the required values such as total quantity, total sold, total profit for the stores and finally it uploads output file in summary-bucket.
- Once the worker uploads the output file into the summary-bucket it sends a message by using OutputFifo.fifo queue to the client.
- We pass the input file name as a command line argument as an example 01-10-2022-store5.csv. We upload only two CSV files to show our solution is working correctly. But we have 20 CSV files in our project and it is difficult to provide every file as an argument individually every time.
- So we upload all the input files into the bucket manually.
- Once worker application done the calculation, we can see all the output files in summary-bucket.
- Adding one output file below to show the format of how the worker file generates the output.

product/store	quantity	profit	sold
store5	146112	17224333	33765767
p10	3157	251297.2	718605.5
p12	2926	130382.6	324119.3
p11	2974	334961.6	975862.6
p14	2908	333605.8	697939.4
p13	2815	188295.4	331616.9
p16	3002	421240.6	1028155
p15	3255	606764.6	1325951
p18	2931	620903	1397327
p17	2871	413194.3	542866
p19	2808	283916.9	616452.1
p21	3035	561475	1389315
p20	3116	315370.4	715739.6
p23	2954	304646	801233.1
p22	3001	492374.1	986896.9
p25	2960	308165.6	497462.8
p24	2982	555337.9	987089.7
p27	2989	188785.2	561018.6
p26	2878	74511.42	213465.1
p29	2986	326937.1	711178.2

Figure 1: worker output file for store5 on 02-10-2022

4.1.2 Client

- Client uploads input file into the bucket ”uploadbucket” as a command line argument and wait for 10 seconds to check the queue for message.
- If it receives the message, then download the output file from ”summary-bucket” and delete the queue, Or else it will wait for 10 more seconds to check the queue.
- Client will uploads another input file and wait for some time to receive another message from worker to download next output file, Or else the client will terminate automatically if it’s don’t have any further input files to upload.

4.1.3 Consolidator

- Now for the Consolidator application we provide the dates (01-10-2022 and 02-10-2022) as command line arguments to calculate the total retailer’s profit, the most and least profitable stores, and the total quantity, total sold, and total profit per product on that particular dates.
- We considered total-Price in input file for calculate sold in output file. We multiply unit-profit with quantity in the input files for calculating total profit in output file.
- From the files in summary-bucket, Consolidator file generates the below output(In total profit the letter E indicates ”to get exact number multiply the number with 10 to the power 6”).

```
Summary Results for the application for the date01-10-2022/

Total Profit: 1.9123226910999998E8,Max Profit Store: store2,Min Profit Store:  store5

Product,Total Quantity,Total Profit,Total Sold
p10,33035,2629586.00,7519522.19
p12,32144,1432336.64,3560662.59
p11,33118,3730080.34,10867053.58
p14,31981,3668860.32,7675653.13
p13,33282,2226232.98,3920738.36
p16,32435,4551279.20,11108663.15
p15,31496,5871169.36,12830154.74
p18,32407,6865098.88,15449734.86
p17,33343,4798724.56,6304694.22
p19,32627,3298915.97,7162743.02
p21,33200,6142000.00,15197772.35
p20,32165,3255419.65,7388243.47
p23,32728,3375238.64,8877033.80
p22,32180,5279772.60,10582586.21
p25,32419,3375142.09,5448394.53
p24,32423,6038135.29,10732531.70
p27,32914,2078848.24,6177774.98
p26,32981,853878.09,2446245.17
p29,32384,3545724.16,7712926.22
p28,32788,2554185.20,5000789.27
p0,32404,6519684.80,10596554.23
p1,32388,3278961.12,5659983.02
p2,32538,6893175.30,13481686.30
p3,32809,3451506.80,8836302.92
p4,33616,310611.84,813753.98
p5,31696,1288442.40,3640145.17
p6,32618,2230745.02,5976958.74
p7,32126,4401262.00,8648818.76
```

Figure 2: summary file from consolidator

4.2 Client, Worker as a Java application and Consolidator

In this solution, we implemented worker as a Java application. We developed the solution as mentioned in the below steps.

1. Created three Java maven projects such as for the client (filename:Client), the worker (filename:Worker) and for the consolidator(filename:Consolidator).
2. Created one EC2 instance to implement worker as a Java application.
3. Created two S3 buckets to upload the input and output files. Used one bucket to upload input files with bucket name ”mybucket88888888777” and stored output files in the bucket name ”summary-bucket”.
4. Created two SQS queues named with InputFifo.fifo and OutputFifo.fifo to store/pass messages between client and worker. Client pass messages via InputFifo.fifo and Worker use OutputFifo.fifo to receive messages.

4.2.1 Worker

- We run the Worker code using maven build to generate a Jar file. It’s created that jar file and placed it in inside the target folder.
- After that we need to connect to the EC2 instance (by using SSH or any other method)and then enter our credentials to connect.
- Upload the Jar file to our EC2 instance using SFTP (Use suitable method according to your OS, we used FileZilla).
- Next we run the jar file in the EC2 instance using the command ”java -jar Worker-0.0.1-SNAPSHOT-jar-with-dependencies.jar” in the terminal.
- Once worker receive the message from inbox queue from the client, first it downloads the input file from bucket ”mybucket88888888777”.
- Next it do the necessary calculations mentioned in the code to provide total quantity, total sold, total profit for the stores and finally it uploads output file in the summary-bucket.
- It sends a message by using outputFifo queue to inform output file is in the summary-bucket. And it receives a message from inputFifo queue.
- Finally it deletes the existing messages in inbox queue and wait for to check new message in inbox queue to proceed further until we terminate the instance manually .

4.2.2 Client

- Run the Client file locally.
- First it uploads input file into the bucket ”mybucket88888888777” as a command line argument and sent message to the worker file.
- After sending message to the worker, client will wait for 10 seconds to check the outbox queue for receive message from worker.
- Once it receives the message, then download the output file from ”summary-bucket” and delete the queue, Or else it will wait for 10 more seconds to check the queue.
- Client will uploads another input file and wait for some time to receive another message from worker to download next output file or else it will terminate automatically if client has no input files to upload.

4.2.3 Consolidator

- Now for the Consolidator application we provide the dates (01-10-2022 and 02-10-2022) as command line arguments to calculate the total retailer's profit, the most and least profitable stores, and the total quantity, total sold, and total profit per product on that particular dates.
- We considered total-Price in the input file to calculate sold in output file. We multiply unit-profit with quantity in the input files for calculating total profit in output file.
- From the files in the summary bucket, the Consolidator file generates the final summary result file as same in the Lambda function part.

5 Comparison between Lambda function and Java application

In our project, in one implementation the worker is using AWS Lambda, triggering the function, and doing the calculation. Also in another implementation, the worker is running in the EC2 instance and doing the calculation part.

The following are the main differences between the two implementations

- In Lambda worker and client we are using one queue while in Java worker and client have two queues. Both clients in the two projects are the same except that queues are used in the application. The same consolidator has been used in both projects.
- For doing the calculation in lambda, the worker is taking nearly 25 seconds. And in the java application, it is taking 8 seconds.
- In Lambda setup and management of the environment is easier than EC2 application setup. Because it is completely automated. With EC2, setting up includes logging in via SSH and manually installing Apache. Along with that, we need to install and configure all the required software like putty.
- Our Lambda records a timeout of 300 seconds. This limits the type of tasks Lambda can deal with, Therefore so long-running functions and complex tasks aren't a good fit for that. But in our case functions are good enough to deal with the time-out period. In comparison with Lambda, the java application with EC2 has flexible options. It can work with long-running tasks. However, we encounter a connection timeout error in the EC2.
- For Java application, We need to run EC2 instance continuously, so the cost is high. The Lambda function is a piece of code that is executed whenever it is triggered by an event from an event source.
- In lambda, the function is automatically destroyed after the specified task is completed. But in the virtual machine, we need to terminate the EC2 instance manually after the use.

6 Conclusion

We proposed two solutions to summarize and consolidate the data of the daily sales generated by the stores of a large retailer by using AWS Services. Compare the implementation by running worker as a Lambda function and Java application.