Serverless MapReduce using NodeJS, MongoDB, ReactJS and AWS

Abishek Arumugam Thiruselvi
40218896
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
abishekarumugam@icloud.com

***Abstract*— The main aim of the assignment is to implement the concepts and methods in MapReduce and cloud computing.**

Keywords—MongoDB, AWS, ReactJS, Function, Lambda, S3, Serverless, runtime, NodeJS, API, Stack, Postman.

I. INTRODUCTION

A MapReduce algorithm is developed to process big data with repeated map and reduction techniques.

All the datasets given in the specification are uploaded in MongoDB using MongoDB compass.

Code organization has the following items, and the repository is listed as a reference in the reference section.

The backend has two files, "handler.js" and "test1.js". The handler.js file is deployed in AWS and can be triggered from this endpoint [1]. The test1.js file is for running the program locally.

The front end used is ReactJS.

The runtime used is NodeJS16.x, and the library used is "mongodb".

II. ARCHITECTURE DESIGN

The application developed is serverless-microservices-based architecture. Fig. 1.1. Shows the architecture diagram of implemented MapReduce functionality.

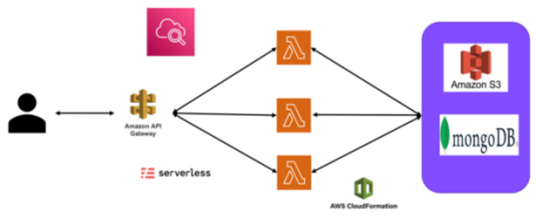


Fig. 1.1. Architecture Diagram

A. AWS services used

AWS Lambda, API Gateway, CloudFormation, S3 and CloudWatch.

B. MongoDB

MongoDB Atlas, MongoDB Compass, and MongoDB Aggregation functionality.

C. Client/Server

The endpoint [1] facilitates the communication between the client and server. From the repository-> backend-> event.json is used as a request body to perform post-operation. The response is logged in CloudWatch.

Local server communication is done with "test1.js".

D. Cloud Architecture

The Architecture follows the same design as from assignment 1 [2].

E. MapReduce Algorithm Design.

The algorithm is designed per the flow chart shown in Fig.1.2.

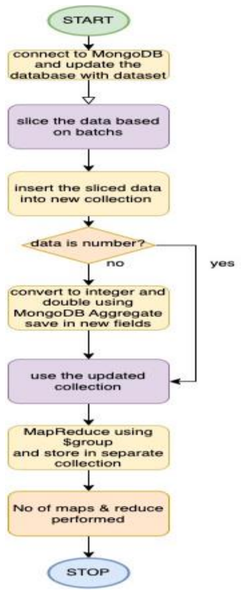

START
connect to MongoDB and update the database with dataset
slice the data based on batchs
insert the sliced data into new collection
data is number?
no
yes
convert to integer and double using MongoDB Aggregate save in new fields
use the updated collection
MapReduce using $group and store in separate collection
No of maps & reduce performed
STOP

Fig. 1. 2. Architecture Diagram

III. INSTRUCTION AND TECHNICAL IMPLEMENTATION

1. Uploading Datasets to MongoDB

Using MongoDB Atlas, connect to the MongoDB compass using the link "mongodb+srv://Abishek:1234test.frpr17t.mongodb.net/test as in Fig.3.1.1. Open the MongoDB Compass and connect to the cluster with the link on add datasets to the created database as shown in Fig.3.1.2. And use adds data option to upload data.

Fig.3.1.1. MongoDB Atlas Connect URL

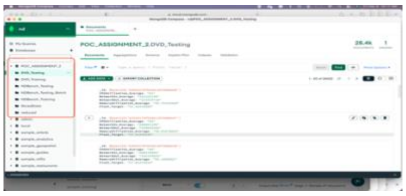


Fig.3.1.2. MongoDB Compass

2. *Run the program locally*

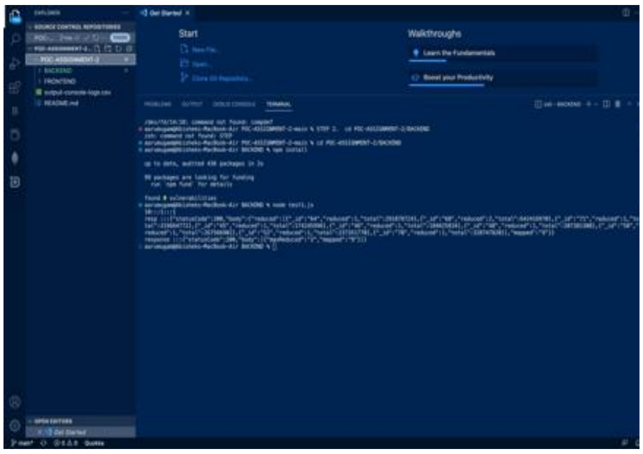
Clone the git repository [3]. Navigate to the backend folder using the terminal and follow the terminal commands below in Fig 3.2.1.

STEP 1. git clone https://github.com/abishekat/POC-ASSIGNMENT-2.git

STEP 2. cd POC-ASSIGNMENT-2/BACKEND

STEP 3. "npm install"

STEP 4. "node test1.js"



```
[illegible]
up to date, audited 438 packages in 2s

99 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
[illegible]
```

Fig 3.2.1. Running Locally

3. *Run the program in AWS*

Clone the git repository [3]. Navigate to the backend folder using the terminal and follow the procedures below in Fig 3.3.1.

STEP 1. Install AWS CLI in the local terminal with the necessary user access [4].

STEP 2. the local terminal, run the command "npm install -g serverless"

STEP 3. the local terminal, run the command, "sls create -t aws-nodejs -p <project-name>"

STEP 4. "npm init -y"

STEP 5. "npm install"

STEP 6. "sls deploy"

STEP 7. Get the endpoint generated from the terminal and test it in postman using event.json in [3].

STEP 8. The same can be tested in AWS Console in "aws.amazon.com".

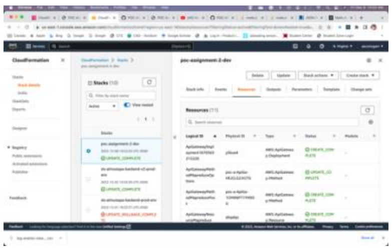


Fig 3.3.1. Running in AWS

IV. RESULTS OF ALGORITHM

A. *Request Parameters*

The deployed codes end point[1] is used in postman with post params as in the event.json[3], as shown in Fig 4.1

B. *Mapped Results*

Using the Aggregate function, MapReduce is performed. The screenshots are provided below with a short description.

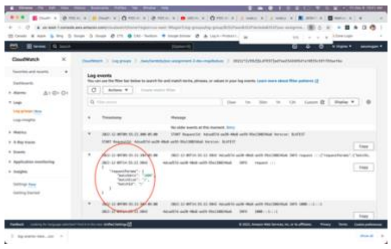


Fig 4.1. Request parameters

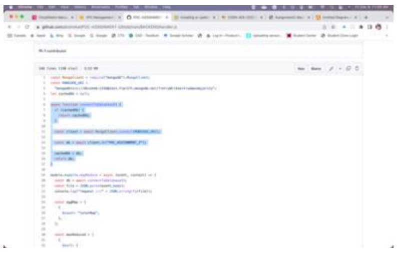


Fig 4.2. DB connection establishment [3]

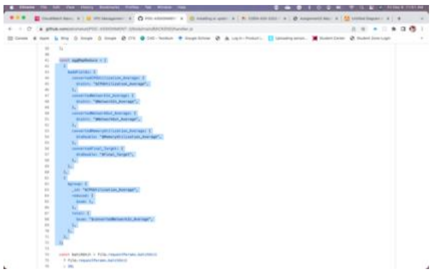


Fig 4.3. MapReduce Aggregation [3]

The pipeline for the mapReduce is as follows.

- "$addFields" – converting string to int and double.
- "$group" – mapping and reducing with id.
- "$sum"- calculation sum of all the maps with the same id

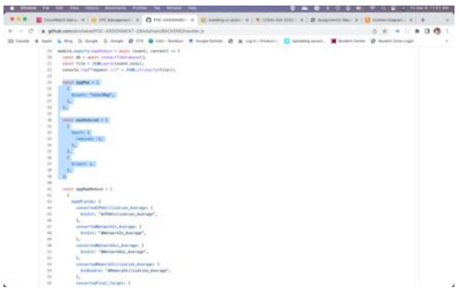


Fig 4.4. Aggregate for calculating map getting maximum reduced value [3]

Fig 4.4 shows two aggregations. "aggMap" helps calculate how many maps have been performed. "maxReduced" maximum reduce has been performed.

C. Frontend

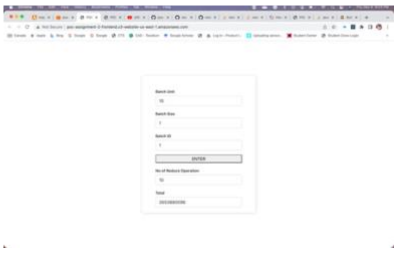


Fig 4.5 frontend developed.

D. Final Result

Where all the data sets are balanced across all map and reduced tasks, the number of maps and reductions will vary according to the request parameters posted in the endpoint [1].

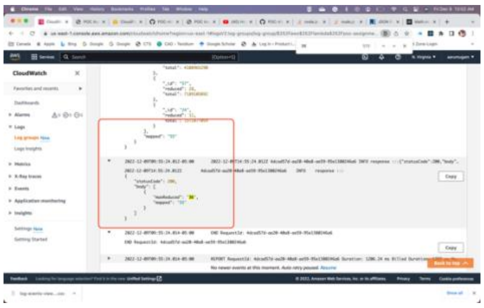


Fig 4.6. Result.

V. Cloud Deployment

Serverless and CloudFormation help in the formation of the stack and helps in cloud deployment. The serverless microservices-based architecture sets the infrastructure using the "serverless.yml" file.

VI. Testing

Testing of the application is done using,

- Postman
- serverless-offline plugin
- Lambda test functions.

VII. Review of Logs

I have attached the logs "output-console-logs.csv" in the git repository [3].

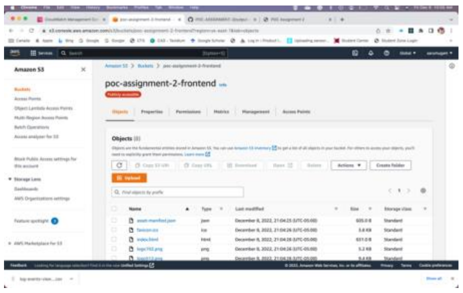


Fig 7.1 Frontend S3 hosting.

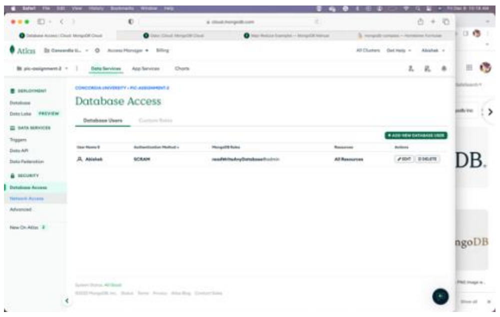


Fig 7.2 DB Network Access.

VIII. Resources

- REPOSITORY- https://github.com/abishekat/POC-ASSIGNMENT-2
- FRONTEND- https://github.com/abishekat/POC-ASSIGNMENT-2/tree/main/FRONTEND
- BACKEND- https://github.com/abishekat/POC-ASSIGNMENT-2/tree/main/BACKEND
    - LOCAL- https://github.com/abishekat/POC-ASSIGNMENT-2/blob/main/BACKEND/test1.js
    - AWS- https://github.com/abishekat/POC-ASSIGNMENT-2/blob/main/BACKEND/handler.js
- LOG- https://github.com/abishekat/POC-ASSIGNMENT-2/blob/main/output-console-logs.csv
- EVENT- https://github.com/abishekat/POC-ASSIGNMENT-2/blob/main/BACKEND/event.json

IX. Conclusions

I have implemented all the specifications in the assignment description and incorporated additional ones.

A. Additional VPC:

Figures 8.1 and 8.2 show the VPC connection of connecting mongodb and AWS securely through peering connections with corresponding security groups and subnet ids.

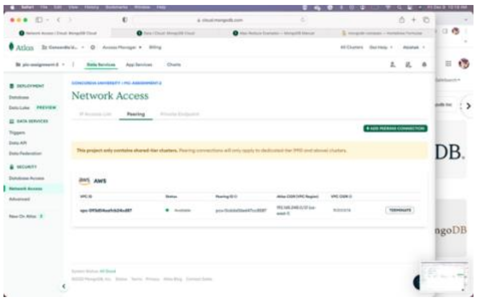


Fig. 8.1. MongoDB Peering Connection

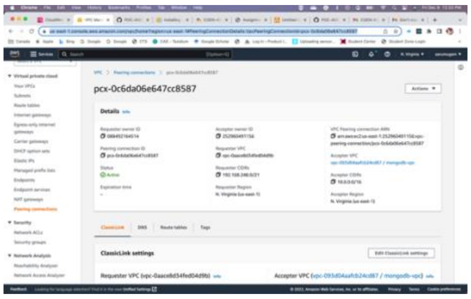


Fig. 8.2. AWS Peering Connection

References

[1] https://mmk8b66r7g.execute-api.us-east-1.amazonaws.com/dev/mapReduce

[2] https://github.com/abishekat/poc-assignment-1/blob/main/40218896_40188548_A1_report.pdf

[3] https://github.com/abishekat/POC-ASSIGNMENT-2

[4] https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

[5] https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-https-unixes.html

[6] https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html

[7] https://docs.aws.amazon.com/codepipeline/latest/userguide/pipelines-create.html

[8] https://s3.console.aws.amazon.com/s3/buckets/poc-assignment-1-data-source?region=us-east-1