**DESIGN STATEMENT**

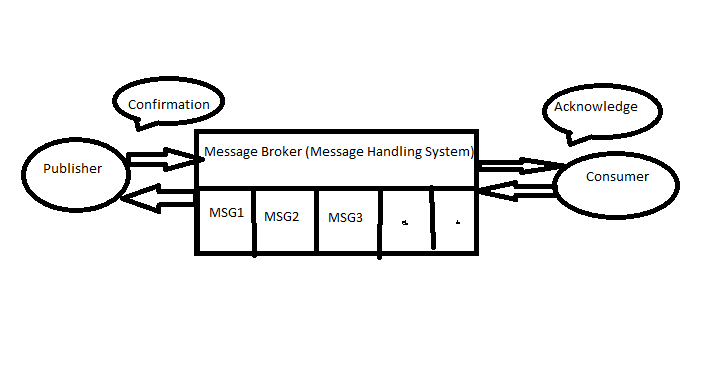
Handling a JSON content which is flowing in gigabytes of data, challenge will be to process continuously and persist also. If it only about saving the data which is coming in Gigabytes then Cloud based solution can be chosen. However here additional requirement is to process them further with Valid/Invalid schema and persist also. This scenario requires a system with distributed processing capability which should be highly scalable with distributed database. For Message Handling and maintenance RabbitMQ, ActiveMQ, SQS, etc are some of the options. For database MongoDB, Cassandra, HBase, DynamoDB, Redis(in-memory) can be a recommended option for distributed database.

There will be a Message Broker/Queue or a Message Handling system which handles the messages will be coming from the Publisher and Consumer will fetch the same for further processing with confirm and acknowledge system in place.

**Message Handling System**

Message Queuing is recommended to choose when client application may goes for offline or out of network, or an environment where heavy data load need to handle. For this scenario Confirm/Acknowledge based system. Recommended message Queuing system (RabbitMQ, ActiveMQ etc)

Below is diagram



**Basic/Simple Message Handler**

public abstract class BasicMessage

{

protected String id;

protected Vector argList;

public BasicMessage()

{

argList = new Vector();

}

public BasicMessage(String msgid)

{

Id = msgid;

argList = new Vector();

}

public void addArg(String arg)

{

argList.addElement(arg);

}

pubic Vector argList() {

Vector lstCopy = (Vector)argList.clone();

return lstCopy;

}

public abstract boolean do();

}

public abstract class BasicMsgHandler implements Runnable

{

// Static message handler for applications where only one message

// handler is used and needs to be globally accessible.

public static BasicMsgHandler current = null;

InputStream msgIn;

OutputStream msgOut;

StreamTokenizer tokenizer;

String msgEndToken = "END";

public BasicMsgHandler(InputStream in, OutputStream out) {

setStreams(in, out);

current = this;

}

public BasicMsgHandler(InputStream in, OutputStream out,

String endToken) {

msgEndToken = endToken;

setStreams(in, out);

current = this;

}

protected void setStreams(InputStream in, OutputStream out) {

msgIn = in;

msgOut = out;

}

public BasicMessage readMsg() throws IOException {

BasicMessage msg;

String token;

DataInputStream din = new DataInputStream(msgIn);

token = din.readUTF();

msg = buildMessage(token);

if (msg != null) {

boolean msgEnd = false;

while (!msgEnd) {

token = din.readUTF();

if (token.compareTo(msgEndToken) == 0)

msgEnd = true;

else {

msg.addArg(token);

}

}

}

return msg;

}

public void sendMsg(BasicMessage msg) throws IOException {

boolean success = true;

DataOutputStream dout = new DataOutputStream(msgOut);

dout.writeUTF(msg.messageID());

Vector args = msg.argList();

int acnt = args.size();

for (int i = 0; i < acnt; i++) {

dout.writeUTF((String)args.elementAt(i));

}

dout.writeUTF(msgEndToken);

}

public void run() {

try {

while (true) {

BasicMessage msg = readMsg();

if (msg != null)

msg.Do();

}

}

// Treat an IOException as a termination of the message

// exchange, and let this message-processing thread die.

catch (IOException e) {}

}

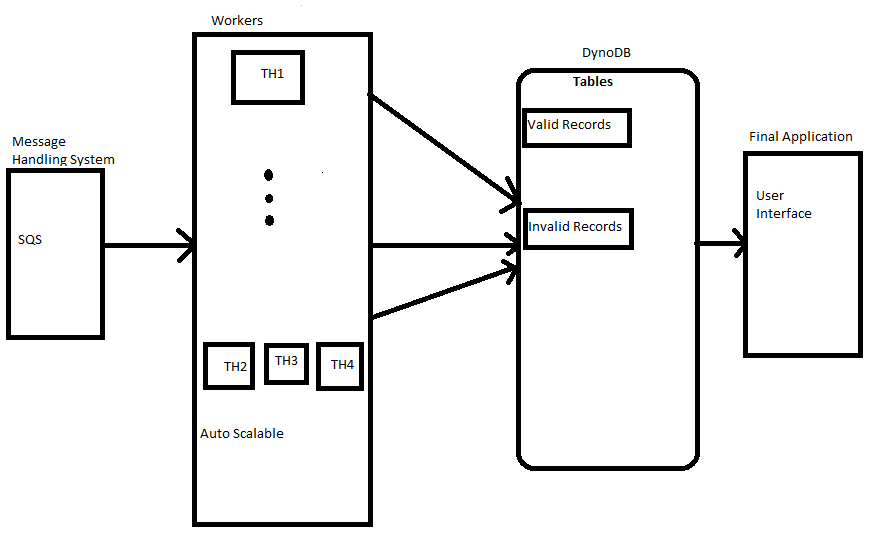
protected abstract BasicMessage buildMessage(String msgId);

}

In the below diagram, each server instances auto scalable based on the number of JSON data in the queue, if queue has more JSON data (can be set by total no. of JSON in terms of size) then create another instance of server. Benefits of this system is it can handle any level of throughput, Auto Scalable, Fully Managed Queue Service.

There will be dedicated Worker thread will be there to process Invalid JSON records periodically.

**Architecture Diagram**

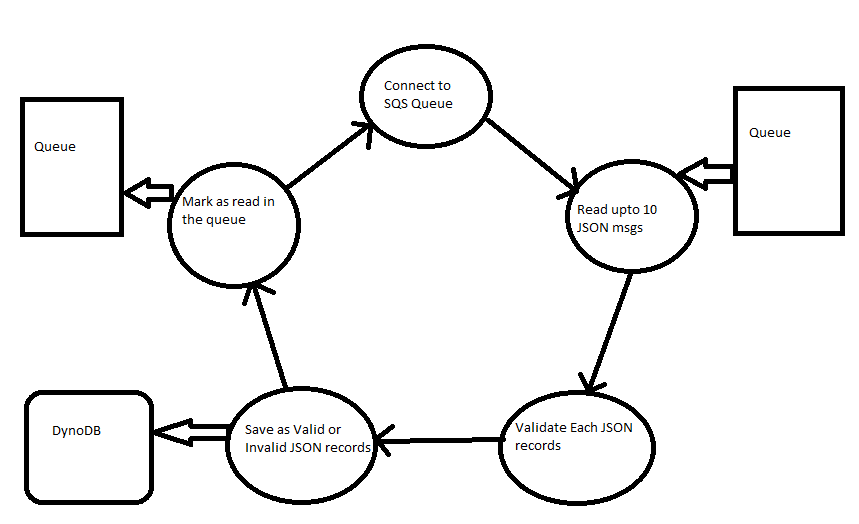


**How it works**

* Each thread will connect to SQS
* Thread is going to read the JSON record for 10 times from the Queue(assuming each thread is responsible to read 10 times which can be configurable)
* Validate each JSON data
* Save the JSON data into DynaMo DB
* Delete the message from Queue if reading is complete

SQS has a feature while reading the message from queue, item will not be visible to other workers. Before start read the data timeout can be set, during the timeot if anyone unable to read the message then message will be visible again.

**Thread Behavior**

****

**Pseudo Code/ Important Functions**

1. Connect to SQS Queue and Read the Message

public static void main(String[] args) {

//GET ENVIRONMENT PARAMETERS

getEnvParameters();

//CREATING POOL OF THREADS

ExecutorService execService = Executors.newFixedThreadPool(NO\_THREAD);

//Launching number of Threads

for (int i=0; I <= NO\_THREAD; i++){

execService.execute(new MessageProcessingTask(QUEUE\_NAME, VALID\_JSON\_RECORD, INVALID\_JSON\_RECORD);

}

execService.shutdown();

}

//ThreadProc

class MessageProcessingTask extends Thread{

@Override

public void run(){

for(;;) {

try {

//Read JSON Records from Queue

List <Message> dataJSON = this.sqsService.receiveMessage();

//PROCESS JSON messags

processMessages(dataJSON);

}

catch(exception e)

{

}

}

}

}

//Reads Messages from Queue

public List<Message> receiveMessage(){

ReciveMessageRequest rcvMsgRqst = new ReceiveMessageRequest(QUEUENAME)

rcvMsgRqst.setMaxNumberofMessages(10);

ReceiveMessageResult msgResult = this.sqs.receiveMessage(rcvMsgRrst);

List <Messages> messages = msgResult.getMessages();

return Messages;

}

//Processing JSON Data

Private void processMessage(List<Message> messages) {

try {

//For Each JSON Record

for (Message msg : messages) {

JSONObject jsonObject = null;

If ((jsonObject == parseMessage(msg) != null){

//SAVE RECORD

this.dynaService.saveRecord(jsonObject);

}

else {

//SAVE INVALID

this.dynaService.saveInvalidMessage(msg.getMessageID(), msg.Content());

}

//SET RECORD PROCESSED in THE QUEUE

sqsService.setMessageProcessed(msg);

}

}

}

//Saving the records in the DynoDB

public void saveRecord(JSON jsonObject)

{

try{

Map <String, AttributeValue> item = new Hashmap<String, AttributeValue>

//SET KEY and Value

fillMap(jsonObject, item);

//CREATE REQUEST OBJECT

PutItemItemRequest puItemRequest = new PutItemRequest(this.validJSONRecordTable, items);

//SAVE

dynamoDB.putItem(putItemRequest);

}

}

//Remove the Message from Queue

public void setMessageProcessed(Message msg)

{

DeleteMessageRequest deleteRequest = new DeleteMessageRequest(QUEUENAME, msg.getReceiptHandle());

sqs.deleteMessage(deleteRequest);

}