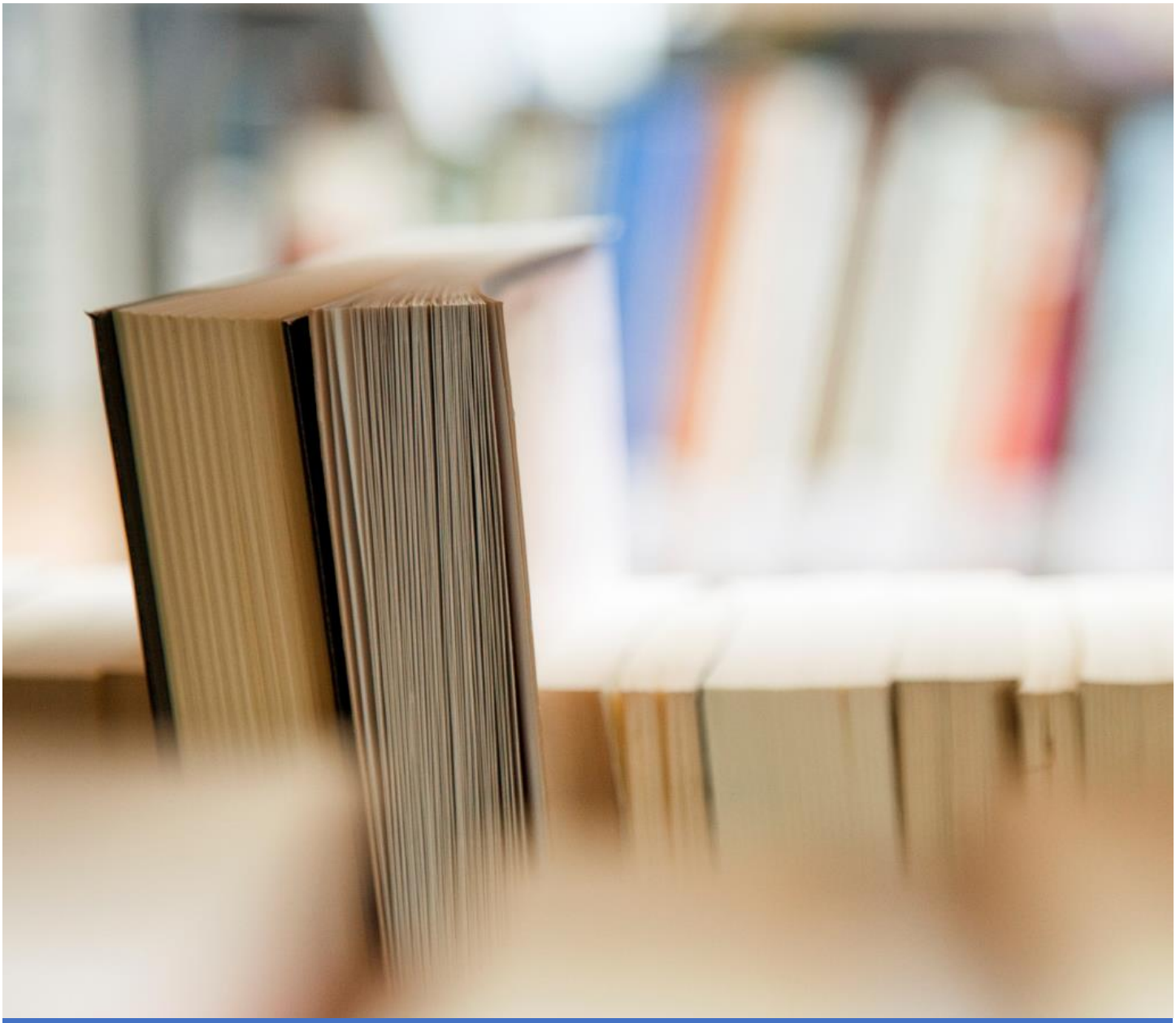

Storing Daily Data from SES to S3 Using AWS Lambda Consistently

Protecting it from Auto Deletion or Data Loss

Sheraz Iqbal

3/10/23

AWS Cloud Solutions Architect &
Full Stack DevOps

Contents

Problem Statement 2

Solutions: 2

Sol 1. Lambda Code Checking and Modification to Prevent Auto Deletion 2

1.2) Suggested Improvements and Modifications in Code 8

1.3) Some Problems and Solutions with Existing Upload Function Code: 15

1.4) Modified Chunk with Different approach of Appending and Uploading Data to S3 without PutObject() 18

Sol2) Upload Function for Appending New Data to Prevent Old Data from Deletion 20

Sol3) Lifecycle Policy for S3 for Additional Layer of Protection from Deletion or Data Loss 23

Sol4) Adding an IAM Policy to Deny any kind of Deletion 24

Sol5) Double Bucket Approach for Storing Data Consistently 25

Problem Statement

Ziya having an issue with the s3 bucket, the messages we are getting from SQS and triggered by lambda should get saved in s3 but it is overwriting the old data in s3 at some point and saving the new or it is auto deleting the data. Can u help me with this so that I can save all the data in s3 from SQS.

- Lambda Function is Triggered by EventBridge to Save html file into S3.

Solutions:

S3 is designed to store data reliably and securely, and it does not modify or delete data without explicit instructions from the user or a configured lifecycle policy.

There can be multiple solutions for solving similar issues based on different restrictions and senerios.

I'll list all possible solutions as per best to my knowledge step by step so you can use one or more than one of them at a time to overcome your problem.

It's possible that there is some issue with the **Lambda function** that is causing it to **overwrite or modify** the existing file instead of creating a new file for each day. Or maybe any other service or configuration is causing this issue so we have to check all possible scenarios. Here are a few things that I have Checked:

Sol 1. Lambda Code Checking and Modification to Prevent Auto Deletion

Checked the Lambda function code to Make sure that the code is correctly writing the new file to S3 with a unique key name that includes the date or some other identifier for each day. It's possible that the code is incorrectly overwriting the existing file instead of creating a new one.

Existing Code is:

```
var AWS = require('aws-sdk');
var ses = new AWS.SES({
  apiVersion: '2010-12-01'
});
var sqs = new AWS.SQS({
  region: process.env.Region,
  httpOptions: {
    agent: agent
  }
});
var s3 = new AWS.S3();
var https = require('https');
var agent = new https.Agent({
  maxSockets: 150
});
```

```
var fs = require('fs');
var queueURLS = [process.env.DeliveryBounceURL, process.env.DBKQueueURL,
process.env.CEQueueURL, process.env.DCEQueueURL, process.env.ECQueueURL,
process.env.DECQueueURL, process.env.GMQueueURL, process.env.DGMQueueURL];
var bucket = process.env.BucketName;
var prefix = process.env.BucketPrefix;
var qSize = null;
var content = null;
// var queueParams = {AttributeNames: ["ApproximateNumberOfMessages"], QueueUrl:
queueURL};


exports.handler = (event, context, callback) => {
  var today = new Date();
  var date = today.toDateString();
  var url = null;

  function s3upload() {
    if (prefix == undefined) {
      prefix = "";
    }
    var fileKey = prefix + date + ".html";
    var param = {
      Bucket: bucket,
      Key: fileKey,
      Body: content,
      ACL: 'public-read',
      ContentType: "text/html"
    };
    s3.getObject({
      Bucket: bucket,
      Key: fileKey
    }, function(err, data) {
      if (err) {
        console.error(err);
        return;
      }
      var existingData = data.Body.toString();
      var newData = content.toString();
      var combinedData = existingData + newData;
      param.Body = combinedData;
      s3.putObject(param, function(err, data) {
        if (err) {
          console.error(err);
        } else {
```

```
        console.log('Data appended to file successfully');
      }
    });
  });


  s3.upload(param, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else console.log(data);
    url = data.Location;
    console.log("uploading to s3");

    //context.done();
  });
}


function initializeQueue(callbackQueue, queueURL) {
  var queueParams = {
    AttributeNames: ["ApproximateNumberOfMessages"],
    QueueUrl: queueURL
  }
  console.log("Reading from: " + queueURL);
  sqs.getQueueAttributes(queueParams, (err, data) => {
    if (err) {
      console.log("Possible issue with SQS permissions or QueueURL wrong")
      callbackQueue(err, null, null);
    }
    qSize = data.Attributes.ApproximateNumberOfMessages;
    callbackQueue(null, qSize, queueParams);
  });
}

function deleteMessage(message, queueURL) {
  console.log("hello", message.ReceiptHandle)
  sqs.deleteMessage({
    QueueUrl: queueURL,
    ReceiptHandle: message.ReceiptHandle
  }, (err, data) => {
    if (err) {
      console.log(err);
      throw err;
    }
    console.log("Data removed. Response = " + data);
  });
```

```
}

//Start Receive message
for (let index = 0; index initializeQueue((err, queueSize, queueParams) => {
    console.log("Reading queue, size = " + queueSize);

    if (queueSize == 0) {
      callback(null, 'Queue is empty.');
    }

    var messages = [];
    var msgBouncePerm = [];
    var msgSuppres = [];
    var msgBounceTrans = [];
    var msgComplaint = [];
    var msgDeliver = [];

    for (var i = 0; i < queueSize; i++) {
      sqs.receiveMessage(queueParams, (err, data) => {
        if (err) {
          console.log(err, err.stack);
          throw err;
        }

        // console.log("data with message = " + data.Messages);
        if (data.Messages) {
          console.log(data.Messages[0])
          for (var j = 0; j < data.Messages.length; j++) {

            var message = data.Messages[j];
            var body = JSON.parse(message.Body);
            var msg = JSON.parse(body.Message);
            console.log("hello", msg)
            var source = msg.mail.source;
            var type = msg.notificationType;
            var time = msg.mail.timestamp;
            var source_ip = msg.mail.sourceIp;
            var id = msg.mail.commonHeaders.subject;
            var otr = "";
            var ftr = "";
            var oline = "";
            var cline = "";
            var btype = null;
            var bsubtype = null;
            var diagcode = null;
```

```
            //console.log(msg);


            if (type == "Bounce") {
              var destination = msg.bounce.bouncedRecipients[0].emailAddress;
              btype = msg.bounce.bounceType; // Permanent || Transient
              bsubtype = msg.bounce.bounceSubType; // General || Supressed
              if (btype == "Permanent" && bsubtype == "Suppressed") {
                diagcode = "Suppressed by SES";
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgSuppres.push(text);

              } else if (btype == "Permanent" && bsubtype == "General") {
                diagcode = msg.bounce.bouncedRecipients[0].diagnosticCode;
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgBouncePerm.push(text);

              } else if (btype == "Permanent" && bsubtype == "NoEmail") {
                diagcode = msg.bounce.bouncedRecipients[0].diagnosticCode;
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgBouncePerm.push(text);

              } else if (btype == "Undetermined") {
                diagcode = msg.bounce.bouncedRecipients[0].diagnosticCode;
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgBouncePerm.push(text);

              } else if (btype == "Transient") {
                diagcode = "soft-Bounce";
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
```

```
                msgBounceTrans.push(text);

              } else {
                console.log("it's an unknown bounce");
                diagcode = "unknown";
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgBouncePerm.push(text);
              }

            } else if (type == "Delivery") {
              console.log("bye there", msg.delivery)
              var destination = msg.delivery.recipients[0];
              btype = "null";
              bsubtype = "null";
              diagcode = "null";
              var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
              msgDeliver.push(text);



            } else if (type == "Complaint") {
              console.log("bye there", msg.complaint)
              var destination =
msg.complaint.complainedRecipients[0].emailAddress;
              btype = "null";
              bsubtype = "null";
              diagcode = "null";
              var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
              msgComplaint.push(text);

            } else {
              console.log("not identified");
            }

            messages.push(i);
```

```
              deleteMessage(message, queueURLS[index]);
              //console.log("Array size = " + messages.length + " with queue size
= " + queueSize);

              if (messages.length == queueSize) {

                var bp = msgBouncePerm.join('');
                var sp = msgSuppres.join('');
                var bt = msgBounceTrans.join('');
                var cp = msgComplaint.join('');
                var dm = msgDeliver.join('');
                var begin = fs.readFileSync('template/begin_new.html', 'utf8');
                var middle = bp + sp + bt + dm + cp;
                var end = fs.readFileSync('template/end_new.html', 'utf8');
                content = begin + middle + end;

                s3upload();

              }
            }
          } else {
            console.log("data without messages.");
          }
        });
      }
    }, queueURLS[index]);

  };
};
```

This code read messages from an SQS queue and handles different types of notifications related to email deliveries (such as bounces, deliveries, etc.). Then it uploads the contents of an email message to an S3 bucket and creates an HTML file with the contents of all the messages processed by the function on the current day.

1.2) Suggested Improvements and Modifications in Code

Following effective and possible improvements or modifications have been done to the code.

- Verified that code is correctly writing the new file to S3 with a unique key name that includes the date timestamp each day. It's possible that the code is incorrectly overwriting the existing file instead of creating a new one.
- S3 allows multiple objects to have the same key (name) as long as they are stored in different folders or prefixes. However, if you try to upload an object with the same key to the same folder or prefix as an existing object, it will overwrite the existing object. So, the improvement is to use

a combination of date/time stamps and a unique identifier to create unique key names for each object.

- To save all the data from SQS to S3, I Modified the Lambda function to append the data to the existing object in the S3 bucket rather than overwriting it. You can use the S3 API to read the existing object, append the new data to it, and write it back to the same object. Alternatively, you can create a new object for each message received from SQS, using a unique key name based on the message ID or any other identifier. This will ensure that each message is saved as a separate object in the S3 bucket, and you won't have to worry about overwriting existing data.

Modified Code is:

```
var AWS = require('aws-sdk');
var ses = new AWS.SES({
  apiVersion: '2010-12-01'
});
var sqs = new AWS.SQS({
  region: process.env.Region,
  httpOptions: {
    agent: agent
  }
});
var s3 = new AWS.S3();
var https = require('https');
var agent = new https.Agent({
  maxSockets: 150
});
var fs = require('fs');
var queueURLS = [process.env.DeliveryBounceURL, process.env.DBKQueueURL,
process.env.CEQueueURL, process.env.DCEQueueURL, process.env.ECQueueURL,
process.env.DECQueueURL, process.env.GMQueueURL, process.env.DGMQueueURL];
var bucket = process.env.BucketName;
//var prefix = process.env.BucketPrefix;
var prefix = process.env.BucketPrefix + "logs/";
var qSize = null;
var content = null;
// var queueParams = {AttributeNames: ["ApproximateNumberOfMessages"], QueueUrl:
queueURL};

exports.handler = (event, context, callback) => {
  var today = new Date();
  var date = today.toDateString();
  var url = null;

  function s3upload() {
    if (prefix == undefined) {
      prefix = "";
```

```
  }
  //var fileKey = prefix + date + ".html";
  var fileKey = prefix + date.getTime() + ".html";

  var param = {
    Bucket: bucket,
    Key: fileKey,
    Body: content,
    ACL: 'public-read',
    ContentType: "text/html"
  };
  s3.getObject({
    Bucket: bucket,
    Key: fileKey
  }, function(err, data) {
    if (err) {
      console.error(err);
      return;
    }
    var existingData = data.Body.toString();
    var newData = content.toString();
    var combinedData = existingData + newData;
    param.Body = combinedData;
    s3.putObject(param, function(err, data) {
      if (err) {
        console.error(err);
      } else {
        console.log('Data appended to file successfully');
      }
    });
  });


  s3.upload(param, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else console.log(data);
    url = data.Location;
    console.log("uploading to s3");

    //context.done();
  });
}
```

```
function initializeQueue(callbackQueue, queueURL) {
  var queueParams = {
    AttributeNames: ["ApproximateNumberOfMessages"],
    QueueUrl: queueURL
  }
  console.log("Reading from: " + queueURL);
  sqs.getQueueAttributes(queueParams, (err, data) => {
    if (err) {
      console.log("Possible issue with SQS permissions or QueueURL wrong")
      callbackQueue(err, null, null);
    }
    qSize = data.Attributes.ApproximateNumberOfMessages;
    callbackQueue(null, qSize, queueParams);
  });
}

function deleteMessage(message, queueURL) {
  console.log("hello", message.ReceiptHandle)
  sqs.deleteMessage({
    QueueUrl: queueURL,
    ReceiptHandle: message.ReceiptHandle
  }, (err, data) => {
    if (err) {
      console.log(err);
      throw err;
    }
    console.log("Data removed. Response = " + data);
  });
}

//Start Receive message
for (let index = 0; index initializeQueue((err, queueSize, queueParams) => {
    console.log("Reading queue, size = " + queueSize);

    if (queueSize == 0) {
      callback(null, 'Queue is empty.');
    }

    var messages = [];
    var msgBouncePerm = [];
    var msgSuppres = [];
    var msgBounceTrans = [];
    var msgComplaint = [];
```

```
      var msgDeliver = [];

      for (var i = 0; i < queueSize; i++) {
        sqs.receiveMessage(queueParams, (err, data) => {
          if (err) {
            console.log(err, err.stack);
            throw err;
          }

          // console.log("data with message = " + data.Messages);
          if (data.Messages) {
            console.log(data.Messages[0])
            for (var j = 0; j < data.Messages.length; j++) {

              var message = data.Messages[j];
              var body = JSON.parse(message.Body);
              var msg = JSON.parse(body.Message);
              console.log("hello", msg)
              var source = msg.mail.source;
              var type = msg.notificationType;
              var time = msg.mail.timestamp;
              var source_ip = msg.mail.sourceIp;
              var id = msg.mail.commonHeaders.subject;
              var otr = "";
              var ftr = "";
              var oline = "";
              var cline = "";
              var btype = null;
              var bsubtype = null;
              var diagcode = null;
              //console.log(msg);


              if (type == "Bounce") {
                var destination = msg.bounce.bouncedRecipients[0].emailAddress;
                btype = msg.bounce.bounceType; // Permanent || Transient
                bsubtype = msg.bounce.bounceSubType; // General || Supressed
                if (btype == "Permanent" && bsubtype == "Suppressed") {
                  diagcode = "Suppressed by SES";
                  var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                  msgSuppres.push(text);
```

```
                } else if (btype == "Permanent" && bsubtype == "General") {
                  diagcode = msg.bounce.bouncedRecipients[0].diagnosticCode;
                  var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                  msgBouncePerm.push(text);


                } else if (btype == "Permanent" && bsubtype == "NoEmail") {
                  diagcode = msg.bounce.bouncedRecipients[0].diagnosticCode;
                  var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                  msgBouncePerm.push(text);

                } else if (btype == "Undetermined") {
                  diagcode = msg.bounce.bouncedRecipients[0].diagnosticCode;
                  var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                  msgBouncePerm.push(text);

                } else if (btype == "Transient") {
                  diagcode = "soft-Bounce";
                  var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                  msgBounceTrans.push(text);

                } else {
                  console.log("it's an unknown bounce");
                  diagcode = "unknown";
                  var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                  msgBouncePerm.push(text);
                }

              } else if (type == "Delivery") {
                console.log("bye there", msg.delivery)
```

```
                var destination = msg.delivery.recipients[0];
                btype = "null";
                bsubtype = "null";
                diagcode = "null";
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgDeliver.push(text);



              } else if (type == "Complaint") {
                console.log("bye there", msg.complaint)
                var destination =
msg.complaint.complainedRecipients[0].emailAddress;
                btype = "null";
                bsubtype = "null";
                diagcode = "null";
                var text = otr + oline + type + cline + oline + btype + cline +
oline + bsubtype + cline + oline + source + cline + oline + destination + cline +
oline + diagcode + cline + oline + source_ip + cline + oline + time + cline +
oline + id + cline + ftr;
                msgComplaint.push(text);

              } else {
                console.log("not identified");
              }

              messages.push(i);

              deleteMessage(message, queueURLS[index]);
              //console.log("Array size = " + messages.length + " with queue size
= " + queueSize);

              if (messages.length == queueSize) {

                var bp = msgBouncePerm.join('');
                var sp = msgSuppres.join('');
                var bt = msgBounceTrans.join('');
                var cp = msgComplaint.join('');
                var dm = msgDeliver.join('');
                var begin = fs.readFileSync('template/begin_new.html', 'utf8');
                var middle = bp + sp + bt + dm + cp;
                var end = fs.readFileSync('template/end_new.html', 'utf8');
```

```
                content = begin + middle + end;

                s3upload();

              }
            }
          } else {
            console.log("data without messages.");
          }
        });
      }
    }, queueURLS[index]);

  };
};
```

1.3) Some Problems and Solutions with Existing Upload Function Code:

By Reviewing the upload function in depth and comparing it with all possible problems I found that there might be a reason to the existing code of overwriting the data is because you are using the **putObject()** method **to write the entire file to S3 every time the Lambda function runs.** This method replaces the existing file with the new data, which is why you are losing the previous data.

To append new data to the existing file, We need to first read the existing data from S3 using the **getObject()** method, then **append the new data** to the existing data, and finally **write the combined data back to S3 using the putObject() method.**

```
exports.handler = async (event, context) => {
  const s3 = new AWS.S3();
  const bucketName = 'your-bucket-name';
  const fileName = `data-${new Date().toISOString().slice(0, 10)}.html`; // use
the current date in the file name   // read existing data from S3
  const params = {
    Bucket: bucketName,
    Key: fileName,
  };
  let existingData = '';
  try {
    const data = await s3.getObject(params).promise();
    existingData = data.Body.toString();
  } catch (err) {
    // if the file doesn't exist yet, ignore the error
    if (err.code !== 'NoSuchKey') {
      throw err;
    }
```

```
  }   // generate new data
  const newData = await generateData();   // combine existing and new data
  const combinedData = existingData + newData;   // write combined data back to
S3
  const putParams = {
    Bucket: bucketName,
    Key: fileName,
    Body: combinedData,
    ContentType: 'text/html',
  };
  await s3.putObject(putParams).promise();   return {
    statusCode: 200,
    body: 'Data written to S3',
  };
}; async function generateData() {
  // generate new data here
  return 'new data';
}


/*

This Function will first read the file from S3 using the getObject() method, then
append the new data to the existing data, and finally will write the combined
data back to S3 using the putObject() method.

*/
```

1.3.1) More issues and improvements with S3 Upload ()

Here are some more improvements in code for better and error-free process flow to solve some issues:

- The **s3.getObject() and s3.putObject()** calls are both made in the s3upload() function. This means that **param** is not defined when **s3.getObject()** is called. I think we should define param before calling s3.getObject().

- The **s3.upload()** call is made outside the s3upload() function, so the param is not defined when the s3.upload() is called. We can move the s3.upload() call inside the s3upload() function, after s3.putObject().

Here's an updated S3Upload():

```
exports.handler = (event, context, callback) => {
        var today = new Date();
        var date = today.toDateString();
        var url = null;
        var prefix = '';
        var fileKey = prefix + date + ".html";
        var content = 'your-new-data';
        var param = {
            Bucket: 'your-bucket-name',
            Key: fileKey,
            ACL: 'public-read',
            ContentType: "text/html"
        };     function s3upload() {
            s3.getObject({ Bucket: param.Bucket, Key: param.Key }, function(err,
data) {
                if (err) {
                    console.error(err);
                    return;
                }
                var existingData = data.Body.toString();
                var newData = content.toString();
                var combinedData = existingData + newData;
                param.Body = combinedData;
                s3.putObject(param, function(err, data) {
                    if (err) {
                        console.error(err);
                    } else {
                        console.log('Data appended to file successfully');
                    }
                    s3.upload(param, function (err, data) {
                        if (err) console.log(err, err.stack); // an error
occurred
                        else console.log(data);
                        url = data.Location;
                        console.log("uploading to s3");
                        //context.done();
                    });
                });
            });
        }
        s3upload();
    };
```

```
/*
The s3.getObject() and s3.putObject() calls are both made in the s3upload()
function. This means that param is not defined when s3.getObject() is called. I
think we should define param before calling s3.getObject().

The s3.upload() call is made outside the s3upload() function, so the param is not
defined when the s3.upload() is called. We can move the s3.upload() call inside
the s3upload() function, after s3.putObject().
*/
```

1.4) Modified Chunk with Different approach of Appending and Uploading Data to S3 without PutObject()

This Chunk is explained in a Multiline Comment at the end of the code.

```
exports.handler = (event, context, callback) => {
    const today = new Date();
    const date = today.toDateString();     if (prefix === undefined) {
        prefix = "";
    }
    const fileKey = prefix + date + ".txt";     const params = {
        Bucket: bucket,
        Key: fileKey
    };     // Get the current data from the object
    s3.getObject(params, (err, data) => {
        let newData = content;
        let combinedData;         if (err) {
            // If the object doesn't exist, create it
            if (err.code === 'NoSuchKey') {
                combinedData = newData;
            } else {
                console.error(err);
                return;
            }
        } else {
            // If the object exists, append to it
            const existingData = data.Body.toString();
            combinedData = existingData + newData;
        }         // Upload the combined data back to S3
        const uploadParams = {
            Bucket: bucket,
            Key: fileKey,
            Body: combinedData,
            ContentType: 'text/plain'
```

```
        };         s3.upload(uploadParams, (err, data) => {
            if (err) {
                console.error(err);
            } else {
                console.log('Data appended to file successfully');
            }
        });
    });
};

/*

It defines an AWS Lambda function that will append new data to an existing object in an S3 bucket and upload the concatenated data back to S3.The today and date variables are used to construct the object key.If the prefix variable is undefined, it is set to an empty string.The params variable is an object that contains the S3 parameters for retrieving the object data.

The s3.getObject() method is called to retrieve the existing object data from S3.If there is an error retrieving the object and the error code is NoSuchKey, this means that the object does not exist yet. In this case, the newData is used as the initial data.If there is an error retrieving the object and the error code is not NoSuchKey, the error is logged and the function returns.

If the object exists, the existing data is converted to a string and stored in the existingData variable, and the newData is stored in the newData variable.The existing and new data are concatenated and stored in the combinedData variable.The uploadParams variable is an object that contains the S3 parameters for uploading the combined data back to S3.The s3.upload() method is called to upload the combined data back to S3. If there is an error, it is logged. Otherwise, a success message is logged.I hope this updated code helps you achieve your desired result.

*/
```

Sol 2) Upload Function for Appending New Data to Prevent Old Data from Deletion

To ensure that all data for a given day is collectively saved in one file, you can modify the function **s3upload()** to check if a file for the current day already exists in the S3 bucket, and if so, append the new data to the end of the file. The main Concept here is that;

- It will check the file, If the file already exists, the function first retrieves its current contents using a **createReadStream()** on the object. The new data is then appended to the existing data, and the combined data is uploaded back to S3 using **s3.putObject()**.

Here's a breakdown of the function:

- The first few lines are unchanged from the original function. They set up the S3 parameters and construct the file key based on the current date.
- The function then calls **s3.headObject()** to check if the file already exists. This is done asynchronously using a callback function.
- If **headObject()** returns an error (i.e. the file does not exist), the function creates a new file with the current data using **s3.putObject()**.
- If **headObject()** succeeds, the function reads the current contents of the file using a **createReadStream()** on the object. This sets up an event listener for the 'data' event, which is emitted whenever a chunk of data is received from S3. The event listener concatenates the chunks of data into a string (**existingData**) as they arrive.
- Once all the data has been read from the file, the 'end' event is emitted, and the function appends the new data to **existingData** and uploads the combined data back to S3 using **s3.putObject()**.

Modified Upload Function with CreateReadStream:

```
function s3upload() {
    if (prefix == undefined) {
      prefix = "";
    }

    // Get the current date in ISO 8601 format, truncated to the day
    var isoDate = new Date().toISOString().substring(0, 10);

    // Construct the file key for the current date
    var fileKey = prefix + isoDate + ".html";

    var param = {
      Bucket: bucket,
      Key: fileKey,
      ACL: 'public-read',
      ContentType: "text/html"
```

```
    };

    // Check if the file already exists
    s3.headObject({
      Bucket: bucket,
      Key: fileKey
    }, function(err, data) {
      if (err) {
        // The file does not exist, so create a new file with the current data
        param.Body = content;
        s3.putObject(param, function(err, data) {
          if (err) {
            console.error(err);
          } else {
            console.log('New file created successfully');
          }
        });
      } else {
        // The file already exists, so append the new data to the end of the file
        var existingData = "";
        var readStream = s3.getObject({Bucket: bucket, Key:
fileKey}).createReadStream();
        readStream.on('data', function(chunk) {
            existingData += chunk.toString();
        });
        readStream.on('end', function() {
            var newData = content.toString();
            var combinedData = existingData + newData;
            param.Body = combinedData;
            s3.putObject(param, function(err, data) {
              if (err) {
                console.error(err);
              } else {
                console.log('Data appended to file successfully');
              }
            });
        });
      }
    });
  }
```

It ensure that all data for a given day is collectively saved in one file, you can modify the function **s3upload()** to check if a file for the current day already exists in the S3 bucket, and if so, append the new data to the end of the file. Here's an updated version of the function that implements this behavior:

1. The **isoDate** variable is computed using the **toISOString()** method of a new **Date** object, which returns a string in ISO 8601 format (e.g., "2023-03-09T13:28:00.000Z"). We then truncate this string to the first 10 characters to get just the date (e.g., "2023-03-09").
2. The **fileKey** variable is constructed using the ISO date string instead of the timestamp, so that all data for a given day will be written to the same file.
3. The **s3.headObject()** method is used to check if the file already exists. If it does, we read the existing data from the file and append the new data to it before writing it back to S3. Note that we use a **readStream** to read the existing data in chunks, which is more memory-efficient than reading the entire file into memory at once.

Sol3) Lifecycle Policy for S3 for Additional Layer of Protection from Deletion or Data Loss

We can protect our data by moving it or setting hard timeline for deletion through setting lifecycle policies for the **bucket** to **automatically transition older versions of the files to Glacier storage** or delete them after a 1 year period to keep old files consistent and secure from deletion or data loss.

For Doing this approach follow these instructions:

1. Open the Amazon S3 console and navigate to the bucket for which you want to set the lifecycle policy.
2. Click on the "Management" tab and select "Lifecycle".
3. Click "Add lifecycle rule" to create a new rule.
4. Give your rule a name and configure its settings as Transition of older data to another storage class or setting an autodeletion timeline for 1 year or more.
5. For the transition action, select "Transition current version to Glacier" and choose the appropriate number of days after which the transition should occur.
6. For the expiration action, select "Expire current version" and choose the appropriate number of days after which the object should be deleted.
7. Click "Review" to review your rule settings, and then click "Save" to create the rule.

Note that lifecycle policies apply to all versions of an object in a bucket. If you want to apply lifecycle policies to specific object versions, you can use versioning in Amazon S3.

Sol4) Adding an IAM Policy to Deny any kind of Deletion

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:PutObject"
         ],
         "Resource": [
            "arn:aws:s3:::<your-bucket-name>/*"
         ]
      },
      {
         "Effect": "Deny",
         "Action": [
            "s3:DeleteObject",
            "s3:DeleteObjectVersion",
            "s3:PutObject"
         ],
         "Resource": [
            "arn:aws:s3:::<your-bucket-name>/*"
         ],
         "Condition": {
            "Null": {
               "s3:VersionId": true
            }
         }
      }
   ]
}
```

Sol5) Double Bucket Approach for Storing Data Consistently

In this approach store we will be storing the daily email data created by AWS SES in an S3 bucket in HTML format and combine the files at the end of the day into a single file based on the date. The combined file will be stored in another S3 bucket, and the daily files will be deleted … and the daily files refer to the email data files that were created and stored in the S3 bucket on the current day. At the end of the day, after the combined file is created and stored in the separate S3 bucket, the Lambda function will delete the daily email data files from the original S3 bucket using the **s3.deleteObjects** method.

```
const AWS = require('aws-sdk');
const s3 = new AWS.S3();

exports.handler = async (event, context) => {
    try {
        const data = JSON.parse(event.Records[0].Sns.Message);
        const date = new Date();
        const fileName = `email-data-${date.toISOString()}.html`;

        const params = {
            Bucket: 'your-email-data-bucket',
            Key: fileName,
            Body: data
        };

        await s3.putObject(params).promise();

        // Combine files at the end of the day
        const endOfDay = new Date(date.getFullYear(), date.getMonth(),
date.getDate(), 23, 59, 59);
        if (date.getTime() === endOfDay.getTime()) {
            const prefix = `email-data-${date.getFullYear()}-${date.getMonth() +
1}-${date.getDate()}-`;
            const fileList = await s3.listObjects({ Bucket: 'your-email-data-
bucket', Prefix: prefix }).promise();
            const combinedData = fileList.Contents.reduce(async (accum, item) =>
{
                const fileData = await s3.getObject({ Bucket: 'your-email-data-
bucket', Key: item.Key }).promise();
                return (await accum) + fileData.Body.toString('utf-8');
            }, '');

            const combinedFileName = `combined-email-data-${date.getFullYear()}-
${date.getMonth() + 1}-${date.getDate()}.html`;
            const combinedParams = {
                Bucket: 'your-combined-data-bucket',
                Key: combinedFileName,
```

```
                Body: combinedData
            };
            await s3.putObject(combinedParams).promise();

            // Delete daily files
            const deleteParams = {
                Bucket: 'your-email-data-bucket',
                Delete: {
                    Objects: fileList.Contents.map(item => ({ Key: item.Key }))
                }
            };
            await s3.deleteObjects(deleteParams).promise();
        }

        // Return success status
        return {
            statusCode: 200,
            body: 'Email data saved successfully!'
        };
    } catch (err) {
        console.error(err);
        // Return error status
        return {
            statusCode: 500,
            body: 'Error saving email data!'
        };
    }
};


/*

store the daily email data created by AWS SES in an S3 bucket in HTML format and
combine the files at the end of the day into a single file based on the date. The
combined file will be stored in another S3 bucket, and the daily files will be
deleted.


*/
```

Thank You