I have setup aws infrastructure as AWS SES Service sends emails to customers, then this data is stored into html type files as logs in S3 Bucket through AWS Lambda.

yes i am facing a problem that i save daily log files in html file in s3 but it gets deleted automatically after every other day

One possibility is that the Lambda function that you have set up to save the logs to S3 might be overwriting the logs with new data each day, rather than appending to the existing logs. This could result in the logs being deleted after a certain period of time.

Another possibility is that there's some other process or tool that's deleting the logs. For example, if you have set up a script or tool to clean up old data in your S3 bucket, it might be unintentionally deleting the logs.

To troubleshoot this issue, you can try enabling S3 object-level logging on your bucket. This will create a new set of log files that will show all the requests made to your S3 bucket, including any requests that might be deleting the HTML log files.

To enable S3 object-level logging, you can follow these steps:

1. Go to the Amazon S3 console.
2. Select your bucket from the list of buckets.
3. Click on the "Properties" tab.
4. Click on the "Server access logging" sub-tab.
5. Click on the "Edit" button.
6. Choose the "Enable logging" option.
7. Specify a target bucket and prefix for the log files.
8. Click "Save".

After you've enabled S3 object-level logging, you can monitor the log files to see if any requests are being made to delete your HTML log files. If you're not sure how to interpret the log files, you can also reach out to AWS Support for assistance.

I think there can be problem with lambda code i am sharing my lambda policy or function code check it and let me know if there is any problem with it

Sure, I'd be happy to take a look at your Lambda function code to see if there are any issues with it. Please share the code with me and I'll do my best to help you identify any problems.

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]); }; };

The code is written in Node.js and is intended to be deployed as an AWS Lambda function. It uses various AWS services such as SES, SQS, and S3.

The code is designed to process messages received in an SQS queue, which may contain various types of email delivery notifications, such as bounces and complaints. The code parses the messages, extracts the relevant information, and then logs the information and stores it in an S3 bucket.

The **exports.handler** function is the main function that is invoked by the Lambda service. It initializes an SQS queue, reads messages from the queue, processes them, and then deletes them from the queue.

The **initializeQueue** function initializes the SQS queue by retrieving its attributes, such as the number of messages in the queue.

The **deleteMessage** function is used to delete processed messages from the SQS queue.

The **s3upload** function uploads the processed information to an S3 bucket.

The code also uses various AWS SDKs, such as **aws-sdk** and **https**, to interact with AWS services.

check the above code and find the that is there anything that overwrites my old html log files into S3 Buckets with new html log files every 2nd day

Based on the code provided, there doesn't seem to be any code that overwrites old HTML log files in S3 with new ones every 2nd day.

However, without seeing the entire codebase or understanding the full context of the application, it's difficult to say for sure. There could be other parts of the codebase or other processes that could be responsible for overwriting files in S3.

If you're concerned about overwriting files in S3, it's important to review your codebase and all related processes to ensure that they are not overwriting files unintentionally. You may also want to consider implementing versioning on your S3 bucket to help prevent accidental file overwrites.

last day my html log file was saved with 600kb data but today new html file saved in S3 bucket with 400 kb and also my last day file data get deleted and file size reduced to 11kb only

Based on the information provided, it is difficult to determine the cause of the issue with certainty. However, there are a few possible explanations:

1. Overwriting the file: If the S3 bucket is configured to overwrite the existing file with a new one of a smaller size, it is possible that the previous file was overwritten with the new one. This could happen if the S3 bucket is configured to only store the latest version of a file and delete the older versions.
2. Compression: It is possible that the new file is compressed or encoded, resulting in a smaller file size. This could happen if the code or the S3 bucket is configured to compress the files before uploading them to the bucket.
3. Data loss: It is also possible that there was some data loss during the upload process, resulting in a smaller file size. This could happen if there were network or connection issues during the upload, causing some data to be lost.

To determine the cause of the issue, you may need to investigate the S3 bucket configuration, the code that uploads the files, and any logs or error messages that were generated during the upload process.

write a solution for every possible case to prevent getting old files deleted.

To prevent getting old files deleted in the S3 bucket, you can implement the following solutions:

1. Enable versioning: Enabling versioning in the S3 bucket will ensure that each object in the bucket has a unique version ID. This way, you can keep track of the different versions of your files and restore previous versions if needed.
2. Implement a retention policy: You can set up a retention policy in the S3 bucket to prevent the deletion of old files. This can be set based on a specific time frame or event, and it will prevent the files from being deleted even if someone tries to delete them manually.
3. Use lifecycle rules: S3 bucket lifecycle rules can be set up to automatically move or delete objects based on specific criteria. You can create rules to move old files to a different storage class or to another S3 bucket instead of deleting them.
4. Implement versioning with a lifecycle policy: You can combine versioning and lifecycle policies to retain different versions of your files for a specific period. This will ensure that all versions of the file are available for a specific duration, and after that period, they can be deleted automatically.
5. Enable MFA delete: You can enable Multi-Factor Authentication (MFA) Delete to add an extra layer of security to your S3 bucket. This will require users to provide an authentication code before deleting any object in the bucket, including old files.

By implementing these solutions, you can prevent getting old files deleted in your S3 bucket, ensuring the safety and availability of your data.

Implement a retention policy: You can set up a retention policy in the S3 bucket to prevent the deletion of old files. This can be set based on a specific time frame or event, and it will prevent the files from being deleted even if someone tries to delete them manually.

Implementing a retention policy in the S3 bucket is a good solution to prevent getting old files deleted. Here's how to do it:

1. Log in to your AWS console and navigate to the S3 bucket that contains the log files.
2. Click on the bucket and select the Properties tab.
3. Under the Properties tab, select the Object retention option.
4. Enable the retention by selecting "Enable legal hold" option.
5. Select "Days" or "Years" option to specify the retention period for the objects.
6. Click on the "Save" button to save the retention policy.

With this retention policy in place, the S3 bucket will not allow the deletion of the log files until the retention period has expired. This ensures that the log files are kept for the required duration and are available for future analysis or audits. Even if someone tries to delete the files manually, they will be prevented from doing so until the retention period is over.

Use lifecycle rules: S3 bucket lifecycle rules can be set up to automatically move or delete objects based on specific criteria. You can create rules to move old files to a different storage class or to another S3 bucket instead of deleting them.

Yes, using S3 bucket lifecycle rules is another effective solution to prevent getting old files deleted. Here's how you can set it up:

1. Open the S3 Management Console and select the S3 bucket you want to set the lifecycle rule for.
2. Click on the "Management" tab and then select "Lifecycle".
3. Click on "Add lifecycle rule".
4. Enter a name for the rule and then select the objects you want to apply it to. You can apply it to all objects in the bucket or to objects with specific prefixes or tags.
5. Select the action you want to take on the objects. In this case, you want to prevent old files from being deleted. To do this, you can set up a rule that moves the objects to the Glacier storage class or to another S3 bucket.
6. Set the time period for the rule. For example, you can set it to move objects to Glacier after 30 days or to move them to another bucket after 90 days.
7. Save the rule.

Once the lifecycle rule is set up, it will automatically apply to the objects in the bucket based on the criteria you defined. This will prevent the old files from being deleted, and they will either be moved to a different storage class or to another S3 bucket instead.

Use lifecycle rules: S3 bucket lifecycle rules can be set up to automatically move or delete objects based on specific criteria. You can create rules to move old files to a different storage class or to another S3 bucket instead of deleting them.

Yes, setting up lifecycle rules in an S3 bucket can help prevent old files from getting deleted. Here's how it works:

1. Navigate to the S3 console and select the desired bucket.
2. Click on the "Management" tab and select "Lifecycle".
3. Click on "Create lifecycle rule".
4. Give the rule a name and click "Next".
5. Set the lifecycle rule configuration options, including when the rule should be applied, the actions to be taken, and any filters that should be applied.
6. Save the lifecycle rule.

For example, you could create a lifecycle rule that moves all objects older than 30 days to a Glacier storage class, rather than deleting them. This ensures that the data is still accessible if needed, but is stored in a lower-cost storage class. Alternatively, you could create a rule that moves all objects to a different S3 bucket, such as an archive bucket, after a certain amount of time has passed.

By setting up lifecycle rules, you can automate the process of managing old files and ensure that they are retained for as long as necessary, without the risk of accidental deletion.

write that rule

Sure, here's an example lifecycle rule that moves files older than 30 days to a different storage class:

jsonCopy code

{ "Rules": [ { "ID": "Move files older than 30 days", "Filter": { "Prefix": "" }, "Status": "Enabled", "Transitions": [ { "Days": 30, "StorageClass": "STANDARD\_IA" } ], "NoncurrentVersionTransitions": [], "Expiration": { "Days": 3650 }, "NoncurrentVersionExpiration": null } ] }

This rule specifies that any file in the bucket that is older than 30 days should be transitioned to the "STANDARD\_IA" storage class. The rule also includes an expiration of 10 years (3650 days) for the files. This means that if a file is older than 10 years, it will be automatically deleted from the bucket.

You can modify this rule to fit your specific requirements, such as changing the time frame for retention or the storage class for transitioned files. Just make sure to test any changes thoroughly before implementing them in a production environment.

it should store everyday new file with date and keep all the data save in the original file and keep the each file with new name cintaining date in the name and no file should be deleted automatically

To achieve this, you can modify the code to include the date in the filename when storing new files. You can also create a retention policy or lifecycle rule to prevent files from being automatically deleted. Here's an example of how you can modify the code:

pythonCopy code

import datetime import boto3 # Set up S3 client s3 = boto3.client('s3') # Get today's date today = datetime.datetime.now().strftime('%Y-%m-%d') # Set filename with today's date filename = f'log-{today}.html' # Open and read the log file with open('/path/to/log.html', 'rb') as file: log\_data = file.read() # Upload file to S3 bucket with today's date in filename s3.put\_object(Bucket='my-bucket', Key=filename, Body=log\_data) # Set up retention policy for S3 bucket to prevent automatic deletion of files s3.put\_bucket\_lifecycle\_configuration( Bucket='my-bucket', LifecycleConfiguration={ 'Rules': [ { 'ID': 'Retain All Files', 'Status': 'Enabled', 'Prefix': '', 'Expiration': { 'Days': 365 } } ] } )

In this example, the filename includes the current date in the format **log-YYYY-MM-DD.html**. The file is uploaded to the S3 bucket with this filename. A retention policy is set up to retain all files in the bucket for 365 days. This ensures that no files are automatically deleted, and all log data is kept for at least a year.

Enable MFA delete: You can enable Multi-Factor Authentication (MFA) Delete to add an extra layer of security to your S3 bucket. This will require users to provide an authentication code before deleting any object in the bucket, including old files.

Yes, enabling MFA Delete is another effective way to prevent old files from being deleted. With MFA Delete enabled, a user would need to provide an authentication code generated by an MFA device before being able to delete any object in the bucket, including old files.

To enable MFA Delete in your S3 bucket, you can follow these steps:

1. Go to the S3 management console and select your bucket.
2. Click on the "Properties" tab.
3. Click on the "Versioning" option.
4. Click on the "Edit" button to edit the versioning settings.
5. Scroll down to the "MFA Delete" section and click on the "Enable MFA Delete" checkbox.
6. Follow the prompts to set up MFA Delete with an MFA device.

Once MFA Delete is enabled, any user attempting to delete an object, including old files, will need to provide an authentication code generated by an MFA device in addition to their normal credentials. This adds an extra layer of security and helps prevent accidental or unauthorized deletions.

aws s3 is dropping or deleting our files continuously for example S3 daily saves HTML files with important data and this file is stored by lambda function in S3 Bucket daily. but on every 2nd day previous file size / data gets deleted or reduces

If S3 is dropping or deleting files, there could be several reasons why this is happening. Here are some possible causes and solutions:

1. Misconfigured lifecycle rules: If there are lifecycle rules set up in the S3 bucket, they could be configured incorrectly, causing the files to be deleted prematurely. Double-check the rules and adjust them as necessary.
2. Overwriting files: If the same file name is used for each day's log, it could be possible that the new log is overwriting the old log file instead of creating a new one. Make sure that the file names are unique and include a date or timestamp.
3. Misconfigured Lambda function: It's possible that the Lambda function is not functioning correctly, causing it to overwrite or delete files unintentionally. Check the code for any errors or bugs and test it thoroughly.
4. Accidental deletion: It's also possible that someone with access to the S3 bucket is accidentally deleting files. Enable versioning to prevent accidental deletion of files and keep track of changes.
5. Storage class changes: If the storage class of the S3 object is set to Glacier or Deep Archive, it could be possible that the data retrieval process is not optimized, resulting in data loss. Check the retrieval options for these storage classes and ensure that they are configured correctly.
6. Insufficient permissions: Check the permissions for the S3 bucket and ensure that only authorized users have access to it. Unauthorized users could be deleting the files without realizing the impact.

By investigating these possible causes and taking appropriate actions, you should be able to prevent S3 from dropping or deleting your files.

"Misconfigured lifecycle rules: If there are lifecycle rules set up in the S3 bucket, they could be configured incorrectly, causing the files to be deleted prematurely. Double-check the rules and adjust them as necessary. .." I did not specified any lifecycle rules. please right the perfect rule for me that store daily file with newname and keep all the files stored without any auto deletion or data loss

Sure, here's an example of a lifecycle rule that you can use to store daily files with new names and prevent any automatic deletion or data loss:

1. Log in to the AWS Management Console and navigate to the S3 service.
2. Select the S3 bucket where you want to set up the rule.
3. Click on the "Management" tab and then click on "Lifecycle".
4. Click on "Add lifecycle rule" button.
5. In the "Name and scope" section, give your rule a name and select the prefix for the objects to which the rule will apply (e.g. "logs/").
6. In the "Transitions" section, select the time frame after which you want to transition the objects to a different storage class (e.g. "30 days after creation date").
7. In the "Expiration" section, select "Never" to prevent the objects from being deleted automatically.
8. In the "Review" section, review the rule configuration and click "Create rule" to save it.

This rule will transition the daily files to a different storage class after 30 days (which can help reduce storage costs), but it will never delete them automatically, so you can keep all the files stored without any data loss. Additionally, you can add a prefix to the file names to make it easier to identify them (e.g. "logs/2022-03-08.html").

Top of Form

Use lifecycle rules: S3 bucket lifecycle rules can be set up to automatically move or delete objects based on specific criteria. You can create rules to move old files to a different storage class or to another S3 bucket instead of deleting them.

Yes, setting up lifecycle rules in an S3 bucket can help prevent old files from getting deleted. Here's how it works:

Navigate to the S3 console and select the desired bucket.

Click on the "Management" tab and select "Lifecycle".

Click on "Create lifecycle rule".

Give the rule a name and click "Next".

Set the lifecycle rule configuration options, including when the rule should be applied, the actions to be taken, and any filters that should be applied.

Save the lifecycle rule.

For example, you could create a lifecycle rule that moves all objects older than 30 days to a Glacier storage class, rather than deleting them. This ensures that the data is still accessible if needed, but is stored in a lower-cost storage class. Alternatively, you could create a rule that moves all objects to a different S3 bucket, such as an archive bucket, after a certain amount of time has passed.

By setting up lifecycle rules, you can automate the process of managing old files and ensure that they are retained for as long as necessary, without the risk of accidental deletion.

Sure, here's an example lifecycle rule that moves files older than 30 days to a different storage class:

{

"Rules": [

{

"ID": "Move files older than 30 days",

"Filter": {

"Prefix": ""

},

"Status": "Enabled",

"Transitions": [

{

"Days": 30,

"StorageClass": "STANDARD\_IA"

}

],

"NoncurrentVersionTransitions": [],

"Expiration": {

"Days": 3650

},

"NoncurrentVersionExpiration": null

}

]

}

This rule specifies that any file in the bucket that is older than 30 days should be transitioned to the "STANDARD\_IA" storage class. The rule also includes an expiration of 10 years (3650 days) for the files. This means that if a file is older than 10 years, it will be automatically deleted from the bucket.

You can modify this rule to fit your specific requirements, such as changing the time frame for retention or the storage class for transitioned files. Just make sure to test any changes thoroughly before implementing them in a production environment.

it should store everyday new file with date and keep all the data save in the original file and keep the each file with new name cintaining date in the name and no file should be deleted automatically

To achieve this, you can modify the code to include the date in the filename when storing new files. You can also create a retention policy or lifecycle rule to prevent files from being automatically deleted. Here's an example of how you can modify the code:

import datetime

import boto3

# Set up S3 client

s3 = boto3.client('s3')

# Get today's date

today = datetime.datetime.now().strftime('%Y-%m-%d')

# Set filename with today's date

filename = f'log-{today}.html'

# Open and read the log file

with open('/path/to/log.html', 'rb') as file:

log\_data = file.read()

# Upload file to S3 bucket with today's date in filename

s3.put\_object(Bucket='my-bucket', Key=filename, Body=log\_data)

# Set up retention policy for S3 bucket to prevent automatic deletion of files

s3.put\_bucket\_lifecycle\_configuration(

Bucket='my-bucket',

LifecycleConfiguration={

'Rules': [

{

'ID': 'Retain All Files',

'Status': 'Enabled',

'Prefix': '',

'Expiration': {

'Days': 365

}

}

]

}

)

In this example, the filename includes the current date in the format **log-YYYY-MM-DD.html**. The file is uploaded to the S3 bucket with this filename. A retention policy is set up to retain all files in the bucket for 365 days. This ensures that no files are automatically deleted, and all log data is kept for at least a year.

aws s3 is dropping or deleting our files continuously for example S3 daily saves HTML files with important data and this file is stored by lambda function in S3 Bucket daily. but on every 2nd day previous file size / data gets deleted or reduces

If S3 is dropping or deleting files, there could be several reasons why this is happening. Here are some possible causes and solutions:

1. Misconfigured lifecycle rules: If there are lifecycle rules set up in the S3 bucket, they could be configured incorrectly, causing the files to be deleted prematurely. Double-check the rules and adjust them as necessary.
2. Overwriting files: If the same file name is used for each day's log, it could be possible that the new log is overwriting the old log file instead of creating a new one. Make sure that the file names are unique and include a date or timestamp.
3. Misconfigured Lambda function: It's possible that the Lambda function is not functioning correctly, causing it to overwrite or delete files unintentionally. Check the code for any errors or bugs and test it thoroughly.
4. Accidental deletion: It's also possible that someone with access to the S3 bucket is accidentally deleting files. Enable versioning to prevent accidental deletion of files and keep track of changes.
5. Storage class changes: If the storage class of the S3 object is set to Glacier or Deep Archive, it could be possible that the data retrieval process is not optimized, resulting in data loss. Check the retrieval options for these storage classes and ensure that they are configured correctly.
6. Insufficient permissions: Check the permissions for the S3 bucket and ensure that only authorized users have access to it. Unauthorized users could be deleting the files without realizing the impact.

By investigating these possible causes and taking appropriate actions, you should be able to prevent S3 from dropping or deleting your files.

"Misconfigured lifecycle rules: If there are lifecycle rules set up in the S3 bucket, they could be configured incorrectly, causing the files to be deleted prematurely. Double-check the rules and adjust them as necessary. .." I did not specified any lifecycle rules. please right the perfect rule for me that store daily file with newname and keep all the files stored without any auto deletion or data loss

Sure, here's an example of a lifecycle rule that you can use to store daily files with new names and prevent any automatic deletion or data loss:

1. Log in to the AWS Management Console and navigate to the S3 service.
2. Select the S3 bucket where you want to set up the rule.
3. Click on the "Management" tab and then click on "Lifecycle".
4. Click on "Add lifecycle rule" button.
5. In the "Name and scope" section, give your rule a name and select the prefix for the objects to which the rule will apply (e.g. "logs/").
6. In the "Transitions" section, select the time frame after which you want to transition the objects to a different storage class (e.g. "30 days after creation date").
7. In the "Expiration" section, select "Never" to prevent the objects from being deleted automatically.
8. In the "Review" section, review the rule configuration and click "Create rule" to save it.

This rule will transition the daily files to a different storage class after 30 days (which can help reduce storage costs), but it will never delete them automatically, so you can keep all the files stored without any data loss. Additionally, you can add a prefix to the file names to make it easier to identify them (e.g. "logs/2022-03-08.html").

Top of Form

Here are some potential solutions depending on your setup:

1. If your Lambda function is configured to delete files after a certain period of time, you can modify its code to remove the delete operation. If you're not sure where the code that deletes the files is, you can check the function's code in the AWS Lambda console, or ask the developer who wrote the function for help.
2. If your EventBridge rule is triggering the Lambda function to delete files, you can modify the rule to exclude the file types or directories you want to keep. To do this, you can update the filter pattern in your EventBridge rule to exclude events that correspond to the files or data you want to keep.
3. If you're using an S3 lifecycle policy to automatically delete objects in your bucket, you can modify the policy to exclude certain objects or modify the retention period for those objects. To do this, you can go to your S3 bucket's management console, click on "Lifecycle", and edit your existing policy or create a new one.