
Multer S3

Streaming multer storage engine for AWS S3.

This project is mostly an integration piece for existing code samples from Multer's storage engine documentation with a call to S3 as the substitution piece for file system. Existing solutions I found required buffering the multipart uploads into the actual filesystem which is difficult to scale.

AWS SDK Versions

3.x.x releases of multer-s3 use AWS JavaScript SDK v3. Specifically, it uses the Upload class from @aws-sdk/lib-storage which in turn calls the modular S3Client.

2.x.x releases for multer-s3 use AWS JavaScript SDK v2 via a call to s3.upload.

Installation

```
1 npm install --save multer-s3
```

Usage

```
1 const { S3Client } = require('@aws-sdk/client-s3')
2 const express = require('express')
3 const multer = require('multer')
4 const multerS3 = require('multer-s3')
5
6 const app = express()
7
8 const s3 = new S3Client()
9
10 const upload = multer({
11   storage: multerS3({
12     s3: s3,
13     bucket: 'some-bucket',
14     metadata: function (req, file, cb) {
15       cb(null, {fieldName: file.fieldname});
16     },
17     key: function (req, file, cb) {
18       cb(null, Date.now().toString())
19     }
20   })
21 })
22
```

```
23 app.post('/upload', upload.array('photos', 3), function(req, res, next)
    {
24   res.send('Successfully uploaded ' + req.files.length + ' files!')
25 })
```

File information

Each file contains the following information exposed by `multer-s3`:

| Key | Description | Note |
|---------------------------------|---|------------------------|
| <code>size</code> | Size of the file in bytes | |
| <code>bucket</code> | The bucket used to store the file | <code>S3Storage</code> |
| <code>key</code> | The name of the file | <code>S3Storage</code> |
| <code>acl</code> | Access control for the file | <code>S3Storage</code> |
| <code>contentType</code> | The <code>mimetype</code> used to upload the file | <code>S3Storage</code> |
| <code>metadata</code> | The <code>metadata</code> object to be sent to S3 | <code>S3Storage</code> |
| <code>location</code> | The S3 <code>url</code> to access the file | <code>S3Storage</code> |
| <code>etag</code> | The <code>etag</code> of the uploaded file in S3 | <code>S3Storage</code> |
| <code>contentDisposition</code> | The <code>contentDisposition</code> used to upload the file | <code>S3Storage</code> |
| <code>storageClass</code> | The <code>storageClass</code> to be used for the uploaded file in S3 | <code>S3Storage</code> |
| <code>versionId</code> | The <code>versionId</code> is an optional param returned by S3 for versioned buckets. | <code>S3Storage</code> |
| <code>contentEncoding</code> | The <code>contentEncoding</code> used to upload the file | <code>S3Storage</code> |

Setting ACL

ACL values can be set by passing an optional `acl` parameter into the `multerS3` object.

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     acl: 'public-read',
6     key: function (req, file, cb) {
7       cb(null, Date.now().toString())
8     }
9   })
10 })
```

Available options for canned ACL.

| ACL Option | Permissions added to ACL |
|----------------------------------|--|
| private | Owner gets <code>FULL_CONTROL</code> . No one else has access rights (default). |
| public-read | Owner gets <code>FULL_CONTROL</code> . The <code>AllUsers</code> group gets <code>READ</code> access. |
| public-read-write | Owner gets <code>FULL_CONTROL</code> . The <code>AllUsers</code> group gets <code>READ</code> and <code>WRITE</code> access. Granting this on a bucket is generally not recommended. |
| aws-exec-read | Owner gets <code>FULL_CONTROL</code> . Amazon EC2 gets <code>READ</code> access to <code>GET</code> an Amazon Machine Image (AMI) bundle from Amazon S3. |
| authenticated-read | Owner gets <code>FULL_CONTROL</code> . The <code>AuthenticatedUsers</code> group gets <code>READ</code> access. |
| bucket-owner-read | Object owner gets <code>FULL_CONTROL</code> . Bucket owner gets <code>READ</code> access. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it. |
| bucket-owner-full-control | Both the object owner and the bucket owner get <code>FULL_CONTROL</code> over the object. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it. |

| ACL Option | Permissions added to ACL |
|---------------------------------|---|
| <code>log-delivery-write</code> | The <code>LogDelivery</code> group gets <code>WRITE</code> and <code>READ_ACP</code> permissions on the bucket. For more information on logs. |

Setting Metadata

The `metadata` option is a callback that accepts the request and file, and returns a metadata object to be saved to S3.

Here is an example that stores all fields in the request body as metadata, and uses an `id` param as the key:

```
1 var opts = {
2   s3: s3,
3   bucket: config.originalsBucket,
4   metadata: function (req, file, cb) {
5     cb(null, Object.assign({}, req.body));
6   },
7   key: function (req, file, cb) {
8     cb(null, req.params.id + ".jpg");
9   }
10  };
```

Setting Cache-Control header

The optional `cacheControl` option sets the `Cache-Control` HTTP header that will be sent if you're serving the files directly from S3. You can pass either a string or a function that returns a string.

Here is an example that will tell browsers and CDNs to cache the file for one year:

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     cacheControl: 'max-age=31536000',
6     key: function (req, file, cb) {
7       cb(null, Date.now().toString())
8     }
9   })
10  });
```

Setting Custom Content-Type

The optional `contentType` option can be used to set Content/mime type of the file. By default the content type is set to `application/octet-stream`. If you want multer-s3 to automatically find the content-type of the file, use the `multerS3.AUTO_CONTENT_TYPE` constant. Here is an example that will detect the content type of the file being uploaded.

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     contentType: multerS3.AUTO_CONTENT_TYPE,
6     key: function (req, file, cb) {
7       cb(null, Date.now().toString())
8     }
9   })
10 })
```

You may also use a function as the `contentType`, which should be of the form `function(req, file, cb)`.

Setting StorageClass

`storageClass` values can be set by passing an optional `storageClass` parameter into the `multerS3` object.

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     acl: 'public-read',
6     storageClass: 'REDUCED_REDUNDANCY',
7     key: function (req, file, cb) {
8       cb(null, Date.now().toString())
9     }
10   })
11 })
```

Setting Content-Disposition

The optional `contentDisposition` option can be used to set the `Content-Disposition` header for the uploaded file. By default, the `contentDisposition` isn't forwarded. As an example below, using the value `attachment` forces the browser to download the uploaded file instead of trying to open it.

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     acl: 'public-read',
6     contentDisposition: 'attachment',
7     key: function (req, file, cb) {
8       cb(null, Date.now().toString())
9     }
10  })
11 })
```

Using Server-Side Encryption

An overview of S3's server-side encryption can be found in the [S3 Docs] (<http://docs.aws.amazon.com/AmazonS3/latest/dev/using-server-side-encryption.html>); be advised that customer-managed keys (SSE-C) is not implemented at this time.

You may use the S3 server-side encryption functionality via the optional `serverSideEncryption` and `sseKmsKeyId` parameters. Full documentation of these parameters in relation to the S3 API can be found [here] (<http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html#upload-property>) and [here] (<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingServerSideEncryption.html>).

`serverSideEncryption` has two valid values: 'AES256' and 'aws:kms'. 'AES256' utilizes the S3-managed key system, while 'aws:kms' utilizes the AWS KMS system and accepts the optional `sseKmsKeyId` parameter to specify the key ID of the key you wish to use. Leaving `sseKmsKeyId` blank when 'aws:kms' is specified will use the default KMS key. **Note:** You must instantiate the S3 instance with `signatureVersion: 'v4'` in order to use KMS-managed keys [[Docs]] (<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingAWSSDK.html#specify-signature-version>), and the specified key must be in the same AWS region as the S3 bucket used.

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     acl: 'authenticated-read',
6     contentDisposition: 'attachment',
7     serverSideEncryption: 'AES256',
8     key: function(req, file, cb) {
9       cb(null, Date.now().toString())
10    }
11  })
12 })
```

Setting Content-Encoding

The optional `contentEncoding` option can be used to set the `Content-Encoding` header for the uploaded file. By default, the `contentEncoding` isn't forwarded. As an example below, using the value `gzip`, a file can be uploaded as a gzip file - and when it is downloaded, the browser will uncompress it automatically.

```
1 var upload = multer({
2   storage: multerS3({
3     s3: s3,
4     bucket: 'some-bucket',
5     acl: 'public-read',
6     contentEncoding: 'gzip',
7     key: function (req, file, cb) {
8       cb(null, Date.now().toString())
9     }
10  })
11 })
```

You may also use a function as the `contentEncoding`, which should be of the form `function(req, file, cb)`.

Testing

The tests mock all access to S3 and can be run completely offline.

```
1 npm test
```