Practical: S3 blob storage service (Javascript)

Table of Contents

S3 is AWS's blob storage service. In this practical you will learn how to read and write data with S3.

Prerequisites and references

- AWS S3 documentation (https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html)
- AWS SDK code samples for S3 (https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/javascript_s3_code_examples.html)
- Complete code for this prac: s3demo.zip (https://canvas.qut.edu.au/courses/20367/files/6583675/download)

Creating an S3 bucket

S3 uses the concept of a *bucket* to organise objects. Objects are stored within a single bucket, and it is typical for an application to have multiple buckets to organise objects according to their

usage.

It is straightforward to create buckets and perform other S3 operations using the AWS console. In this practical we will concentrate on using the JS API.

1. Get authenticated

You can do this practical on an EC2 instance or on your local development environment, provided you have authentication set up.

- If you are using an EC2 instance, ensure that node is installed, that you are the `ubuntu` user, etc. as in previous practicals.
- Refer to **Practical: Installing and using the AWS SDK for Javascript (https://canvas.qut.edu.au/courses/20367/pages/practical-installing-and-using-the-aws-sdk-javascript)** to get started with appropriate authentication for the AWS SDK.

2. Create the node app and install packages

Each AWS service has a separate SDK client that we need to install with npm. In this case we need `@aws-sdk/client-s3`. To get set up with everything we need run these commands:

```
mkdir s3demo
cd s3demo
npm init -y
npm i @aws-sdk/client-s3
npm i dotenv
```

Here we have installed `dotenv` in case you need it for authentication, but you can leave that off if using the IAM role authentication method with an EC2 instance.

3. Write the code

In the `s3demo` directory, create `index.js` with the following contents, changing `n1234567` to your own username:

```
S3 = require("@aws-sdk/client-s3");

const bucketName = 'n1234567-test'

async function main() {
    // Creating a client for sending commands to S3
    s3Client = new S3.S3Client({ region: 'ap-southeast-2' });

    // Command for creating a bucket
    command = new S3.CreateBucketCommand({
        Bucket: bucketName
    });

    // Send the command to create the bucket
    try {
```

```
        const response = await s3Client.send(command);
        console.log(response.Location)
    } catch (err) {
        console.log(err);
    }
}

main();
```

Be sure to change the username in the bucket's name to your own, as buckets need to have unique names.

4. Run the app

- Run `node index.js`

If all goes well then you should see a URL printed, which is the location of the bucket. By default this will not be accessible from the public internet, but you will be able to see the bucket in the list of buckets.

If the bucket already exists (for example, if you've run this already or didn't change the username in the code) then you will see a `BucketAlreadyOwnedByYou` error. This will be OK for the next steps because the bucket is already there and we caught the error.

Tagging an S3 bucket

Like EC2 instances, we require that you tag all S3 buckets that you create. This can be done with the SDK.

1. Add code for tagging

- Near the top of `index.js`, after the `bucketName` line, add the following:

```
const qutUsername = 'n1234567@qut.edu.au'
const purpose = 'prac'
```

- Change the username to your own.
- Next, add the following code after the `try/catch` statements in `index.js` (within the `main` function)

```
command = new S3.PutBucketTaggingCommand({
    Bucket: bucketName,
    Tagging: {
        TagSet: [
            {
                Key: 'qut-username',
                Value: qutUsername,
            },
            {
```

```
                Key: 'purpose',
                Value: purpose
            }
        ]
    }
});
// Send the command to tag the bucket
try {
    const response = await s3Client.send(command);
    console.log(response)
} catch (err) {
    console.log(err);
}
```

2. Run the app

- `node index.js`

At this point the first command (to create the bucket) will fail with a `BucketAlreadyOwnedByYou` error because you've already created the bucket. That is OK since we caught the error. The next command should print out a repsponse with `httpStatusCode: 204`.

Writing to an S3 bucket

We can write to a bucket by creating (or updating) an object. For this we need a key, which is the object's name, and the data for the object. We'll use a string as object's data, but it can also be a byte array or other types.

1. Add code for writing

- Add the following at the top of `index.js`, just below the `purpose` line. You can change the key and value text if you like.

```
const objectKey = 'myAwesomeObjectKey'
const objectValue = 'This could be just about anything.'
```

- Add the following to `index.js` after the tagging code (within the `main` function)

```
// Create and send a command to write an object
try {
    const response = await s3Client.send(
        new S3.PutObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
            Body: objectValue
        })
    );
    console.log(response);
} catch (err) {
```

```
    console.log(err);
}
```

2. Run the app

- `node index.js`

In the response you should see `httpStatusCode: 200`, indicating that the object was successfully written.

Reading from an S3 bucket

Like writing, we can read from an object in a bucket. Again, we need the key so we know which object we are trying to read.

1. Add code for reading

- in `index.js`, under the code for writing the object (within the `main` function) add the following

```
// Create and send a command to read an object
try {
    const response = await s3Client.send(
        new S3.GetObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
        })
    );
    // We need to transform the response's value to a string or other
type.
    str = await response.Body.transformToString();
    console.log(str);
} catch (err) {
    console.log(err);
}
```

Note that we are using the same `bucketName` and `objectKey` variables that we added in the writing step, so these are already set.

2. Run the app

- `node index.js`

You should see the value you wrote to the bucket printed out.

Accessing S3 with Pre-signed URLs

Since S3 uses HTTPS for communication it is very convenient to have clients up/download data directly from S3 rather than going through your server. However, this can cause problems if that data should only be read or written two by authorised entities. S3 fixes this problem with *pre-*

signed URLs. Basically, this is a special URL for a particular object in a bucket that contains an authentication token to read/write specifically to that object.

Here is a typical workflow:

- Client requests a resource
- Server checks that the client is authorised
- Server generates a pre-signed URL for the object corresponding to the requested resource
- Server responds to the client with the URL
- Client uses `fetch()` or similar to retrieve the object using the URL

Pre-signed URLs can also be used for writing to an object. Note that pre-signed URLs have a fixed lifetime after which they become invalid. The time until expiry can be changed.

The SDK has two different methods for generating pre-signed URLs. We'll use the easier one which starts in the same way as the S3 operations we have seen so far, by creating a command object. Then this command object, along with the client object, are passed to the pre-signing function to obtain the URL.

1. Add the code for generating presigned URLs

- Install the presigner module: `npm i @aws-sdk/s3-request-presigner`
- in `index.js` at the top, insert the following line:

```
const S3Presigner = require("@aws-sdk/s3-request-presigner");
```

- in `index.js`, under the code for reading the object (within the `main` function) add the following:

```
// Create a pre-signed URL for reading an object
try {
    const command = new S3.GetObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
        });
    const presignedURL = await S3Presigner.getSignedUrl(s3Client, com
mand, {expiresIn: 3600} );

    console.log('Pre-signed URL to get the object:')
    console.log(presignedURL);

    // fetching the object using an HTTP request to the URL.
    const response = await fetch(presignedURL);
    const object = await response.text();
    console.log('Object retrieved with pre-signed URL: ');
    console.log(object);

} catch (err) {
    console.log(err);
}
```

Note that URLs for putting objects can be created by first creating the appropriate `command` object to be passed to the presigner. When making a request to put and object, use the `PUT` method and set the body to the data to be written to the object, like so:

```
await fetch(presignedURL, { method: "PUT", body: objectValue});
```

2. Run the code

- `node index.js`

After the output you saw in previous steps, you will see the URL printed out. This will be very long, spanning multiple lines. You can copy/paste this into your web browser and retrieve the object if you like. After the URL you should see the contents of the object resulting from the request to the URL.

Delete your bucket

After you are done, please delete your bucket. You can do this through the AWS console. While you are there, you can see your table and the objects stored in it.