**Shruti Shresth**

20 Followers · About Follow

# Uploading Files in AWS S3 Bucket through JavaScript SDK with Progress Bar

 Shruti Shresth Nov 19, 2018 · 4 min read


Amazon S3 Bucket is a cloud storage resource by Amazon Web Services (AWS) which is conveniently used to store simple objects and files easily. S3 is a sort of an acronym for Simple Storage Service.

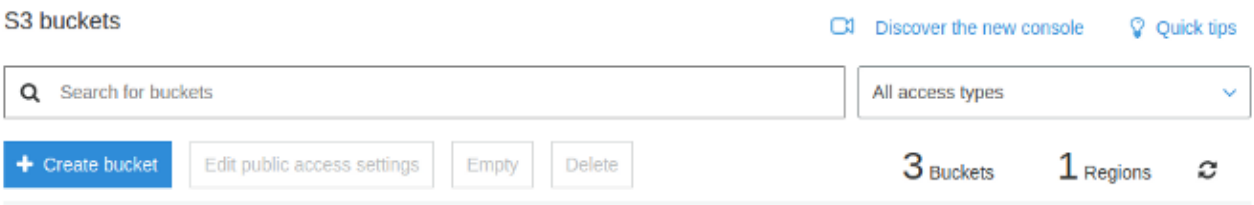
While data upload is mostly done in server side, it can easily be done in the browser or as we say, the front end through the Javascript SDK for AWS.

Here are a few simple but detailed steps to set up your own S3 Bucket and upload files on it using only Javascript and HTML.

## Step 1: Creating a S3 Bucket

1. Go to your AWS console and select the S3 service.

2. In the console, select the option to 'Create Bucket'.




AWS S3 Console

3. A form will open up. In the 'Name and Region' page, type your preferred bucket name in the 'Bucket Name' field. In the 'Region' field, select the region where you want your bucket to be.

4. On the 'Set Permissions' page, select your preferred permissions you want as per your requirements for the accessibility of your files.

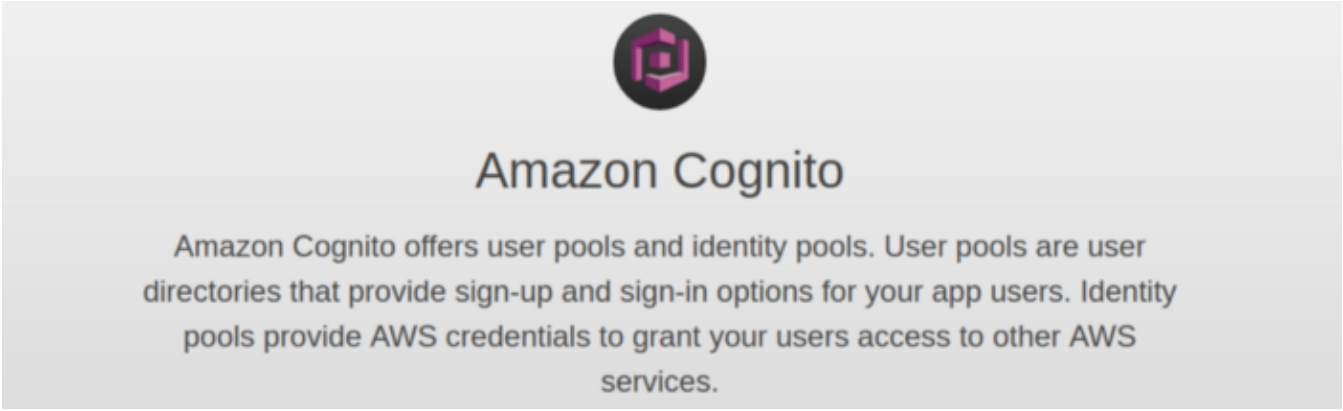
5. After filling the form, click on 'Create Bucket' and your bucket is created!

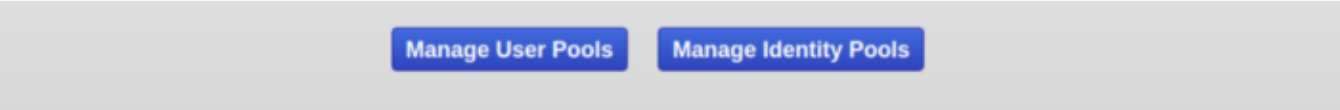
## Step 2: Create a Federated Identity Pool

Now we need to create a Federated Identity Pool for unauthenticated users and gets its ID to put in our browser script so that users can get access to upload data on the s3 bucket.

1. Go to Amazon Cognito Console in the same region as the bucket is created.

2. Click on 'Manage Identity Pool'.






AWS Cognito Console

3. In the next page, click on ‘Create New User Pool’.

4. Create a new pool with access for unauthenticated user.

## Step 3: Add Policy role in the IAM Console

1. In the IAM console, find the role for the unauthenticated user created by the federated identity pool and attach the following policy to enable permissions for the users.

```
{
     "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:*"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME/*"
         ]
      }
   ]
}
```

2. Here the BUCKET_NAME is the name of your new bucket.

## Step 4: Configure CORS

Go to your bucket and in the ‘Permissions’ tab, write the CORS configuration as follows.

```
<?xml version="1.0" encoding="UTF-8"?>

<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">

<CORSRule>

<AllowedOrigin>*</AllowedOrigin>
<AllowedMethod>POST</AllowedMethod>
<AllowedMethod>GET</AllowedMethod>
```

```
<AllowedMethod>PUT</AllowedMethod>
<AllowedMethod>DELETE</AllowedMethod>
<AllowedMethod>HEAD</AllowedMethod>

<AllowedHeader>*</AllowedHeader>

</CORSRule>

</CORSConfiguration>
```

## Step 5: Writing JavaScript Script

After all the configurations have been completed in the Amazon console, there is only JS script to be written in the browser code to configure the bucket.

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-2.1.24.min.js">
</script>

<script type="text/javascript" src="https://code.jquery.com/jquery-
3.2.1.min.js"></script>

<script type="text/javascript">

//Bucket Configurations

var bucketName = BUCKET_NAME;
var bucketRegion = BUCKET_REGION;
var IdentityPoolId = IDENTITY_POOL_ID;

 AWS.config.update({
                region: bucketRegion,
                credentials: new AWS.CognitoIdentityCredentials({
                    IdentityPoolId: IdentityPoolId
                })
            });

            var s3 = new AWS.S3({
                apiVersion: '2006-03-01',
                params: {Bucket: bucketName}
        });

</script>
```

Here BUCKET_NAME is the name of the bucket that you have just created, IDENTITY_POOL_ID is the ID of the Federated Identity Pool which can be obtained from its console and BUCKET_REGION is the region of the bucket where it is created in and currently resides.

## Step 6: Simple HTML Code

Here is a basic simple HTML Code having an input field to upload a file along with a simple HTML progress bar.

```
<div>
   <input type="file" id="fileUpload">
</div>
<div>
   <button onclick="s3upload()">Submit</button>
</div>
<progress max=”100” value=”0”></progress>
```

## Step 7: S3 function to upload a file in the bucket

```
<script type="text/javascript">

function s3upload() {

   var files = document.getElementById('fileUpload').files;

   if (files)
   {
     var file = files[0];
     var fileName = file.name;
     var filePath = 'my-first-bucket-path/' + fileName;
     var fileUrl = 'https://' + bucketRegion + '.amazonaws.com/my-
first-bucket/' +  filePath;

     s3.upload({

        Key: filePath,
        Body: file,
        ACL: 'public-read'
        }, function(err, data) {
        if(err) {
        reject('error');
        }
        alert('Successfully Uploaded!');
        }).on('httpUploadProgress', function (progress) {
        var uploaded = parseInt((progress.loaded * 100) /
progress.total);
        $("progress").attr('value', uploaded);

      });
   }
};

</script>
```

Here we are uploading the data in a simple S3 upload function. We can define our own custom path inside the bucket. The paths are nested inside each other and separated by '/'.

Now your data has been successfully uploaded onto the S3 bucket! You can access it in your S3 console by going into the bucket that you have uploaded your file into.

And the tutorial is complete!

You can access the above example on Github.

There are many other ways in which data that can be manipulated through the browser on the S3 bucket including listing, deleting and views files and folders. They are explained in the AWS documentation with an easy example here.

AWS S3 Cloud Storage JavaScript Browsers