

Localstack practices

Localstack reference:

<https://localstack.cloud>

Benefits:

- You can work offline
- You don't need a shared 'dev' bucket that everyone on your team uses
- You can easily wipe & replace your local buckets
- You don't need to worry about paying for AWS usage
- You don't need to log into AWS 😊

Stack:

- Nodejs
 - Express
 - [brew if is necessary](#)
 - Localstack (AWS suit)
 - Docker & Docker Compose
 - Jest & SuperTest for testing
 - Swagger for documentation
-

First Goal (Fake AWS locally with LocalStack)

Reference

Localstack within a node app

Localstack allows you to emulate a number of AWS services on your computer. We are going to use a S3 iin this example.

Initial setup

1. Install [docker](#) if you haven't already.
2. Install **AWS CLI** [official](#) or [using brew](#), even though we aren't going to be working with "real" AWS, we'll use this to talk to our local docker containers.
3. Once the AWS CLI is installed, run **aws configure** to create some credentials. Even though we're talking to our "fake" local service, we still need credentials. You can enter real credentials (as described [here](#)), or dummy ones. Localstack requires that these details are present, but doesn't actually validate them.
4. Create a new directory for you project, and within it a new nodejs project **npm init -y**.
5. Make a few files inside you project:
 - aws.js
 - docker-compose.yml
 - test-upload.js

- .env

6. Add an image to your project directory and rename it to `test-upload.jpg`.

7. Install aws-sdk, dotenv `npm install --save aws-sdk dotenv`

Docker config

Run Localstack with docker-compose to set this up.

`docker-compose.yml`

```
version: "3.2"
services:
  localstack:
    image: localstack/localstack:0.11.3
    container_name: "fake-aws-s3"
    network_mode: bridge
    ports:
      - "4566:4566"
      - "8055:8080"
    environment:
      - SERVICES=s3
      - DEBUG=1
      - DATA_DIR=/tmp/localstack/data
    volumes:
      - "./.localstack:/tmp/localstack"
      - "/var/run/docker.sock:/var/run/docker.sock"
```

The line `4566:4566` does the same thing, but binds a whole range of ports. These particular port numbers are what Localstack uses as endpoints for the various APIs. We'll see more about this in a little bit.

SERVICES=s3: You can define a list of AWS services to emulate. In our case, we're just using S3, but you can include additional APIs, i.e.

SERVICES=s3, lambda: There's more on this in the Localstack docs.

DEBUG=1:  Show me all of the logs!

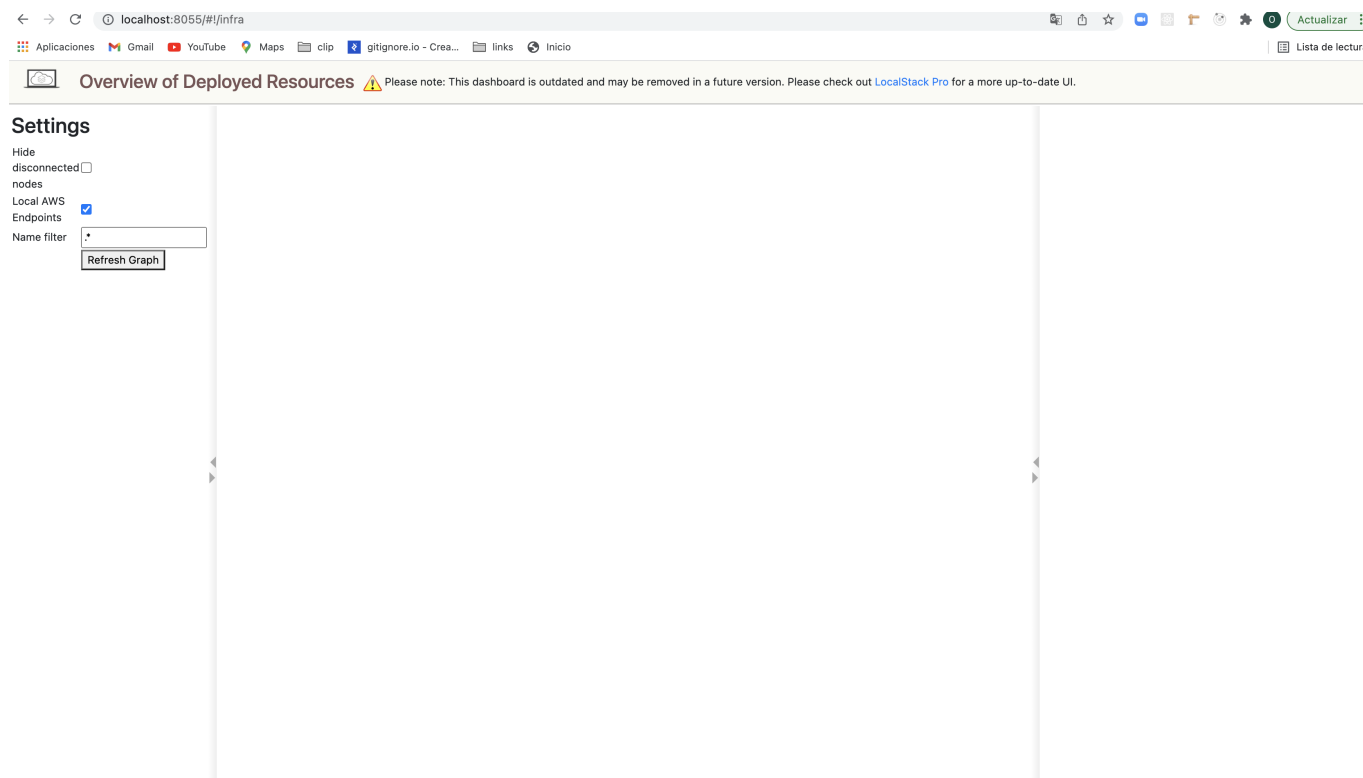
DATA_DIR=/tmp/localstack/data: This is the directory where Localstack will save its data internally.

Starting our container

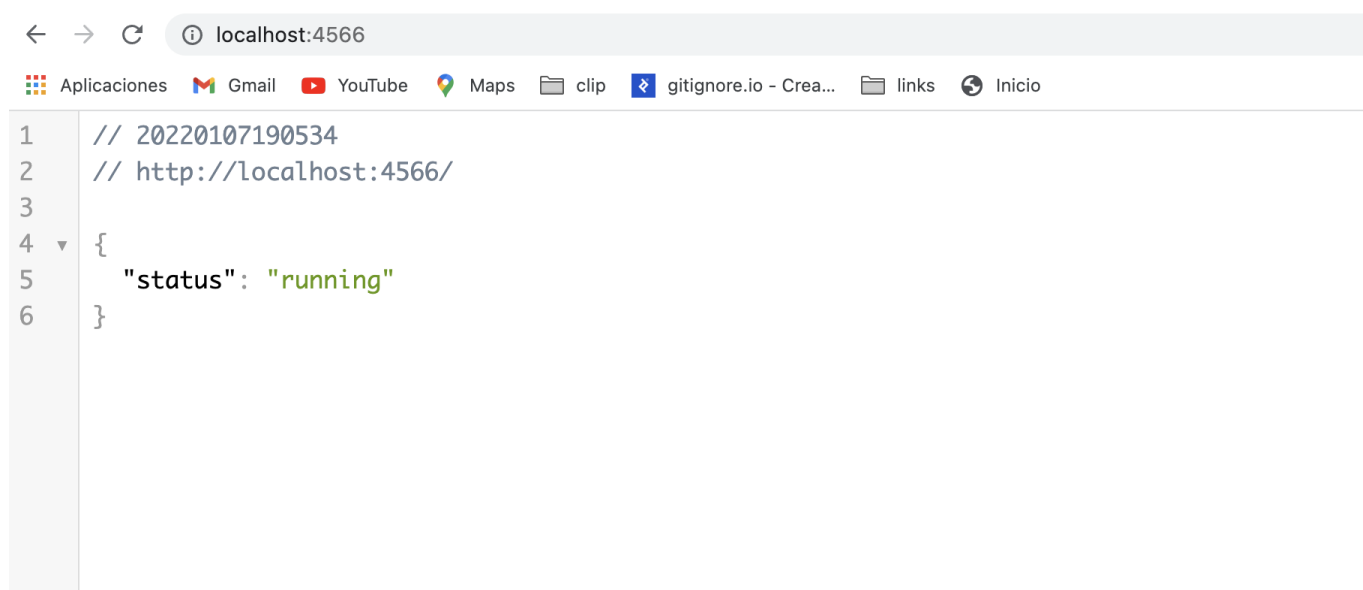
Run in terminal:

`docker-compose up`

To make sure it's working, we can visit <http://localhost:8055> to see localstack's web UI:



Similarly, our s3 endpoint `http://localhost:4566`:

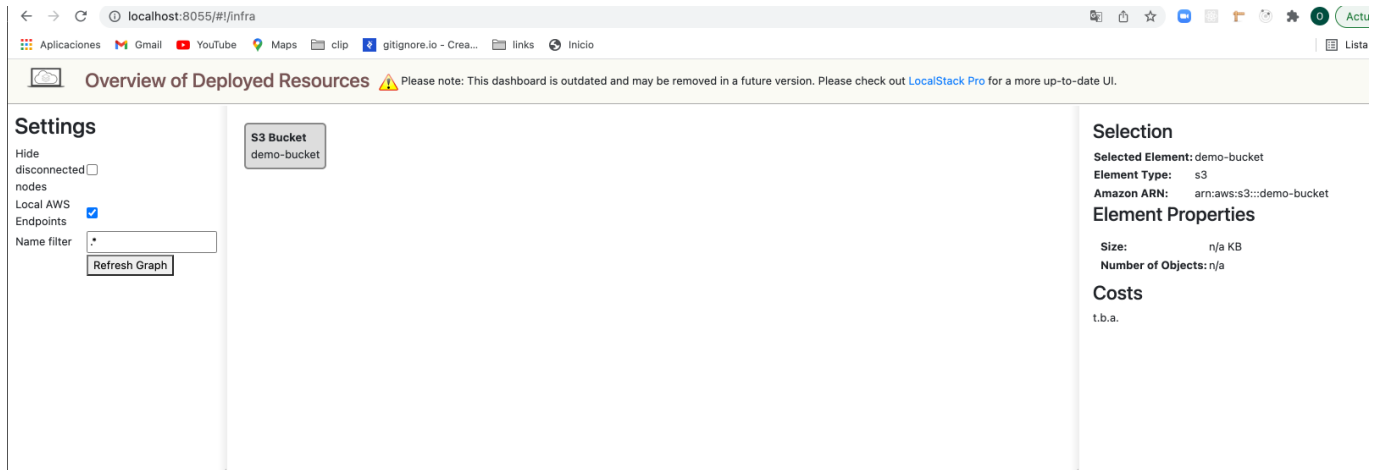


Working with Localstack

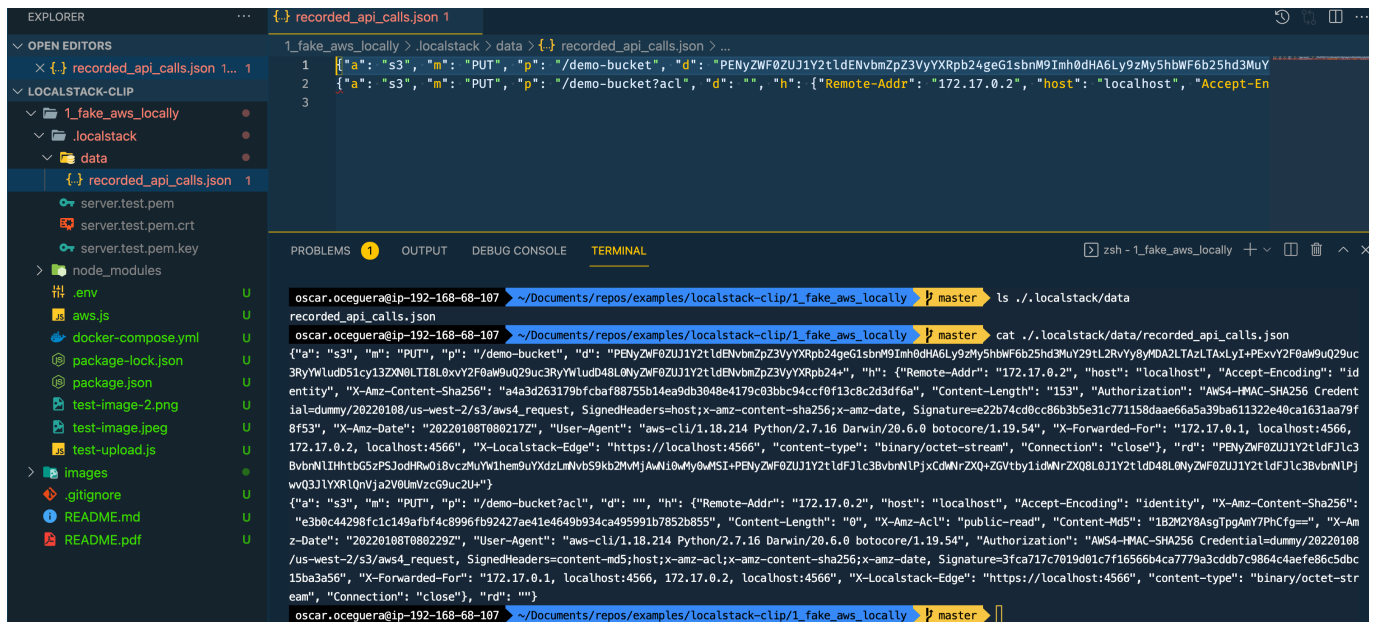
AWS is now inside our computer. After we start upload files, we need to create and configure a **bucket**. We will do this using the AWS CLI that we installed earlier, using the `--endpoint-url` flag to talk to Localstack instead.

1. Create a bucket: `aws --endpoint-url=http://localhost:4566 s3 mb s3://demo-bucket`
2. Attach an [ACL](#) (Access control list) to the bucket so it's readable: `aws --endpoint-url=http://localhost:4566 s3api put-bucket-acl --bucket demo-bucket --acl public-read`

Now, when we visit the web UI, we will see our bucket:



If you use **volumes** in your docker settings, let's pause for a moment to look at what's going on in `./localstack/data`:



Here, we can see that Localstack is recording all API calls in this JSON file. When the container restarts, it will re-apply these calls - this is how we are able to keep our data between restarts. Once we start uploading, we won't see new files appear in this directory. Instead, our uploads will be recorded in this file as raw data. (You could include this file in your repo if you wanted to share the state of the container with others - but depending on how much you upload, it's going to become a pretty big file)

Uploading from our app

We will just make a simple **upload** function and try uploading an image a few times to our S3 bucket.

`.env`, our environment variables

```
AWS_ACCESS_KEY_ID='123'
AWS_SECRET_KEY='xyz'
AWS_BUCKET_NAME='demo-bucket'
```

Note: it doesn't matter what your AWS key & secret are, as long as they aren't empty.

`aws.js`, the module for our upload function

```
const AWS = require("aws-sdk");
require("dotenv").config();

const credentials = {
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_KEY,
};

const useLocal = process.env.NODE_ENV !== "production";

const bucketName = process.env.AWS_BUCKET_NAME;

const s3client = new AWS.S3({
  credentials,
  endpoint: useLocal ? "http://localhost:4566" : undefined,
  s3ForcePathStyle: true,
});

const uploadFile = async (data, name) =>
  new Promise((resolve) => {
    s3client.upload(
      {
        Bucket: bucketName,
        Key: `${bucketName}/${name}`,
        Body: data,
      },
      (err, response) => {
        if (err) throw err;
        resolve(response);
      }
    );
  });

module.exports = uploadFile;
```

`test-upload.js`, which implements the upload function

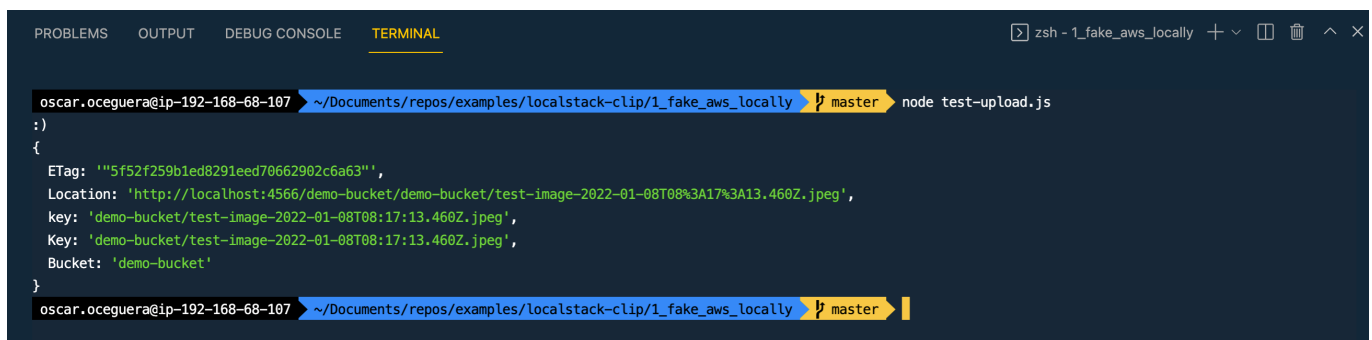
```
const fs = require("fs");
const path = require("path");
const uploadFile = require("./aws");

const testUpload = () => {
  const filePath = path.resolve(__dirname, "test-image.jpg");
  const fileStream = fs.createReadStream(filePath);
  const now = new Date();
  const fileName = `test-image-${now.toISOString()}.jpg`;
}
```

```
uploadFile(fileStream, fileName)
  .then((response) => {
    console.log(":)");
    console.log(response);
  })
  .catch((err) => {
    console.log(":|");
    console.log(err);
  });
};

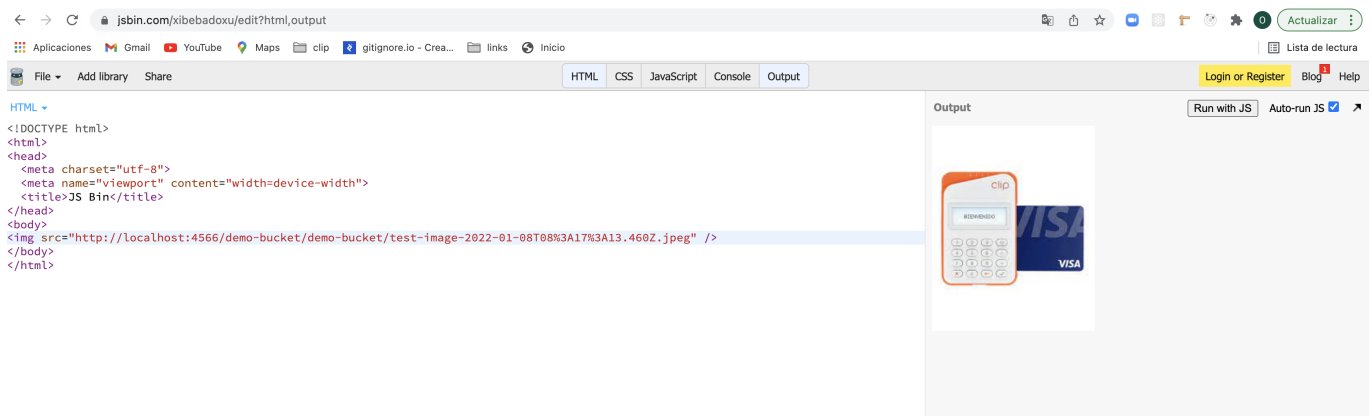
testUpload();
```

Run `node test-upload.js` in your terminal

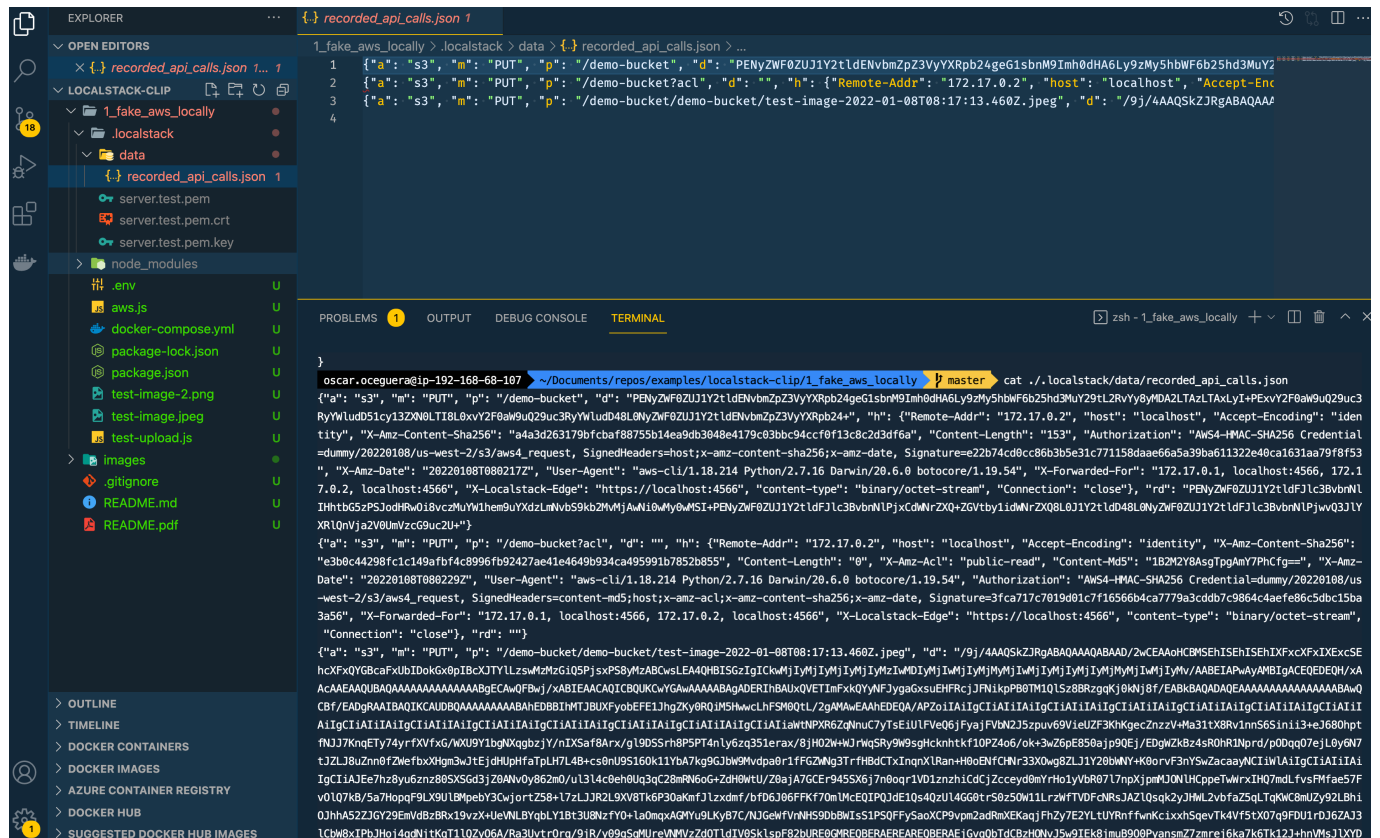


```
oscar.oceguera@ip-192-168-68-107 ~/Documents/repos/examples/localstack-clip/1_fake_aws_locally master node test-upload.js
:)
{
  ETag: '"5f52f259b1ed8291eed70662902c6a63"',
  Location: 'http://localhost:4566/demo-bucket/demo-bucket/test-image-2022-01-08T08:17:13.460Z.jpeg',
  key: 'demo-bucket/test-image-2022-01-08T08:17:13.460Z.jpeg',
  Key: 'demo-bucket/test-image-2022-01-08T08:17:13.460Z.jpeg',
  Bucket: 'demo-bucket'
}
```

Copy the URL in the Location property of the response and paste it into your browser. The browser will immediately download the image. If you want to see it in your browser, you can use something like JS Bin:



Then, if you look at `.localstack/data/recorded_api_calls.json` again, you'll see it filled up with the binary data of the image:



EXERCISE

Add a Endpoint to upload files (images) and other that list all images, install:

- express
- express-fileupload

POST http://localhost:5000

No Environment

Untitled Request

POST

http://localhost:5000

Send

Save

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	image	clip.jpeg	
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 82 ms

Size: 515 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "ETag": "\"5f52f259b1ed8291eed70662902c6a63\"",
3   "Location": "http://localhost:4566/demo-bucket/demo-bucket/2022-01-08T09%3A18%3A15.033Z-clip.jpeg",
4   "key": "demo-bucket/2022-01-08T09:18:15.033Z-clip.jpeg",
5   "Key": "demo-bucket/2022-01-08T09:18:15.033Z-clip.jpeg",
6   "Bucket": "demo-bucket"
7 }
```