ASSET UPLOADER SERVICE APPLICATION

About this project

This microservice exposes all the required API that allows a user to upload an asset on AWS S3 object store and then request a time expiring URL to retrieve that asset.

Application is being developed using Java, Spring boot, Spring cloud, embedded H2 PostgreSQL and AWS cloud SDK integration.

Reason of Choice of Technologies used

Spring boot reduces lots of development time and increases productivity. It avoids writing lots of boilerplate Code, Annotations and XML Configuration. It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data etc. I decided to use an in-memory database to save the data and then manage the session based on use case.

Limitations on use of Technologies

The embedded H2 PostgreSQL is the not the best choice of database when designing similar application that will be shipped to production, but since the objective of this project is for assessment purpose, I used in memory database to achieve the purpose. All data saved will be lost on each restart of the system.

Setup plan

Below are pre-requisite for this application to run and download link

1. Java SDK (Version 8.0)
   (download)https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

2. Apache maven (download): https://maven.apache.org/download.cgi
3. IDE
   You can use any IDE of choice but for the cause of this project I used Spring tool suite
   https://spring.io/blog/2019/06/20/spring-tool-suite-3-9-9-released
4. You need to set the environment variable for java and maven
   Below link will give you a quick guide on how you can do it:
   Set up java environment variable for windows/Linux/Mac OS
   https://javatutorial.net/set-java-home-windows-10
   https://www.serverlab.ca/tutorials/linux/administration-linux/how-to-set-environment-variables-in-linux/
   http://www.sajeconsultants.com/how-to-set-java_home-on-mac-os-x/
5. To setup Apache maven environment variable follow this link
   https://www.tutorialspoint.com/maven/maven_environment_setup.htm
6. Postman for testing the API
   https://www.postman.com/downloads/
7. A valid AWS account

AWS set up

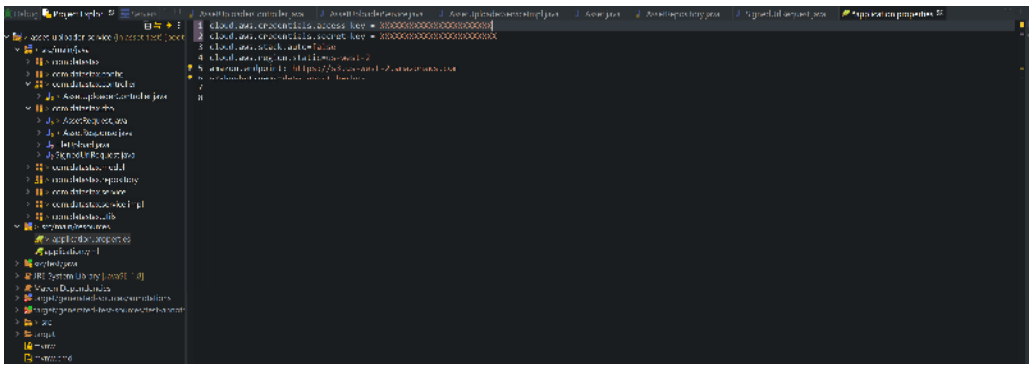
1. Log on to AWS and create IAM user and set the policy for the user, follow the link to learn how to create it https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html

2. create security credentials. If you don't know how to go about it, you can follow the link for a quick guide on creating security credentials
3. https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/getting-your-credentials.html

4. Copy out the access key and secret key generated and keep it in a safe place since you will need while running the app.

Application setup

1. Import the project on any IDE of your choice as a maven project
2. Open a folder called application.properties and set in the values you copied from AWS and replace it with the corresponding values for cloud.aws.credentials.access-key and cloud.aws.credentials.secret-key on the IDE respectively
3. Leave other parameter as it is a shown bellow



Run This project in Local Machine

Import the project on IDE and run as a Spring boot projects or use command bellow

>>$ mvn clean install in root parent pom

This will build a jar file and the service will be up, if everything is fine.

Testing the functionalities

For sake of quick testing, I integrated with **Swagger UI** to generate interactive **API** documentation that will allow your try out the **API** calls directly in the browser. Although, you can still use postman to validate your test.

Paste the link bellow on your browser and click enter button

http://localhost:9000/swagger-ui.html#/

STEP 1. Out of the scope:

You would have manually created the bucket (asset) on AWS but I have exposed an API that you can use to create bucket from the application. And the system will not allow you create a duplicate bucket.

a. Method name: createNewAsset

Request URL: POST *http://localhost:9000/s3/bucket/create /{bucketName}*

Sample Request:

http://localhost:9000/s3/assetName/create/wbp-test-001

Request parameter: assetName: wbp-test-001

Sample Success Response:

{

"assetName": "wbp-test-001",

"url": "",

"statusFlag": false,

"message": "Congratulation your bucket has been created!!!"

}

Sample Failed Request:

When a user tries to create asset with same name

http://localhost:9000/s3/assetName/create/wbp-test-001

Failed Response:

{

"statusFlag": false,

"message": "Bucket already exist"

}

STEP 2.

Method Name: generatePresignedUrl

- **Create a preassigned URL to be use to upload bucket (asset):**

Instruction: Use the bucketName returned on STEP 1 to initiate the request as follows:

Request URL: POST http://localhost:9000/s3?assetName=wbp-test-001

Request parameter: assetName: wbp-test-001

Sample Success Response:

{
"id": "bd0c1b7e-adf2-417b-8751-3f0aa4f17ab4wbp-test-001",
"url": "https://wbp-test-001.s3.us-west-2.amazonaws.com/bd0c1b7e-adf2-417b-8751-3f0aa4f17ab4wbp-test-001?X-Amz- Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210621T092459Z&X-Amz-SignedHeaders=host&X-Amz-Expires=3599&X-Amz-Credential=AKIASTMPXL2EMH7J3KTR%2F20210621%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Signature=5131743a57f042216f03179437a0d0bbd04608b823c4d21b6d5bce819cab6083",
"assetName": "wbp-test-001",
"expiresAt": "2021-06-21T10:24:59.167+00:00",
"timeout": "",
"statusFlag": true
}

Sample Failed Request:

http://localhost:9000/s3?assetName=wbp-test-002

Sample Failed Response

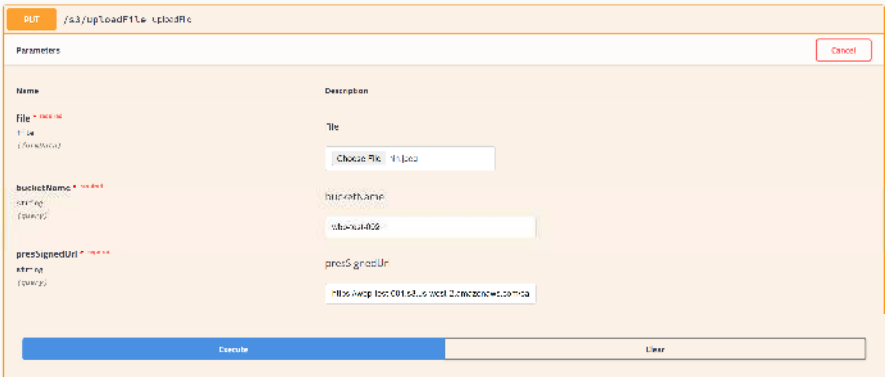
{
"url": "wbp-test-002",
"message": "Bucket does not exist in aws s3, create a bucket before generating presigned URL",
"statusFlag": false
}

STEP 3.

Method Name: UploadFile

With the presigned generated URL, you can now make a call to upload file on the asset already created on STEP 2. Copy the URL and use it to make a PUT call with attach the file you want to upload as shown below:

Request URL: PUT http://localhost:9000/s3/uploadFile



Sample Success Response:

{

http://localhost:9000/s3/uploadFile?bucketName=wbp-test-002&presSignedUrl=https%3A%2F%2Fwbp-test-001.s3.us-west-2.amazonaws.com%2Fca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001%3FX-Amz-Algorithm%3DAWS4-HMAC-SHA256%26X-Amz-Date%3D20210621T093242Z%26X-Amz-SignedHeaders%3Dhost%26X-Amz-Expires%3D3599%26X-Amz-Credential%3DAKIASTMPXL2EMH7J3KTR%252F20210621%252Fus-west-2%252Fs3%252Faws4_request%26X-Amz-Signature%3De3833fa15098755ece94b8f88d32a563ad452977a399645ca96aa2200a4575f2

}

Once done, you can manually log on to AWS console to confirm if the file was added to the asset already created, if you can find your file. Then get back to your program to update the status of upload.

STEP 4.

Method name: getAssetStatus

Now, we will use system to update the uploaded file.

Sample Request:

PUT: http://localhost:9000/s3/check/status/wbp-test-001?statusFlag=true&id=ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001

Request parameter:

1. assetName: Asset name returned after creating the asset in STEP 1
2. statusFlag: Will be set to true to update the uploaded asset in the dB
3. id: This is the id generated and returned after calling generatePresignedUrl in STEP 2

Sample Success Response:

{

"statusFlag": false,

"message": "Your upload is completed!!! Upload status has been set to true"

}

Sample Failed Request:

PUT: http://localhost:9000/s3/check/status/wbp-test-001?statusFlag=true&id=null

Failed Response:

{

"statusFlag": false,

"message": "Pass in a valid ID"

}

STEP 5.

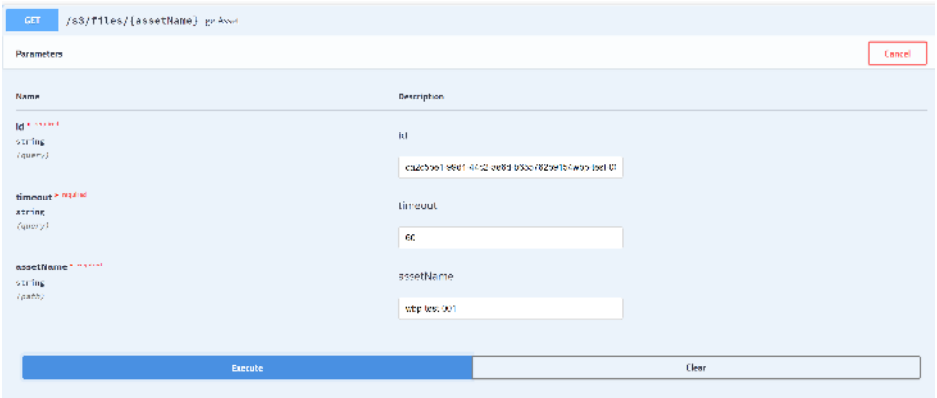
We will now confirm if the update we did on STEP worked.

Method name: getAsset

Sample Request:

Request URL: GET *http://localhost:9000/s3/files/wbp-test-001?id=ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001&timeout=60*

1. assetName: Asset name returned after creating the asset in STEP 1
2. id: This is the id generated and returned after calling generatePresignedUrl in STEP 2
3. timeout: Set the timeout to 60



Sample Success Response:

{

"url": "https://wbp-test-001.s3.us-west-2.amazonaws.com/60?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210621T095607Z&X-Amz-SignedHeaders=host&X-Amz-Expires=3599&X-Amz-Credential=AKIASTMPXL2EMH7J3KTR%2F20210621%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Signature=246f865da6f3f7d067fd8642dc47ae99327950dfc7502edb7316eb4cc097b18d",

"timeout": "60",

"message": "Asset details succesfully retrieved!!!, Upload status is true",

"statusFlag": false

}

STEP 6.

When a call made on an asset that has not been set to "status Upload" from STEP 3 should return an error to the user but if the status Upload has been set the user will get a message that the status upload has been completed.

Sample Request:

http://localhost:9000/s3/check/status/wbp-test-001

Sample Success Response:

{

Upload already completed for this bucket. Your Bucket Name is: wbp-test-001 & Upload status: true

}

Sample Failed Request:

http://localhost:9000/s3/files/wbp-test-00100?id=ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001&timeout=60

Failed Response:

When an asset that has not been created or updated tries to call the endpoint. It will throw an error

{

"message": "Oops!, Your request cannot be completed with buckect name: wbp-test-00100 has not been created",

"statusFlag": false

}

STEP 7.

Method Name: You can use the asset ID generated on STEP 1 to view the asset details

Sample Request:

http://localhost:9000/s3/view/asset/ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001

"id":" ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001",

Sample Success Response:

{

"id": "ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001",

"url": "https://wbp-test-001.s3.us-west-2.amazonaws.com/ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210621T093242Z&X-Amz-SignedHeaders=host&X-Amz-Expires=3599&X-Amz-Credential=AKIASTMPXL2EMH7J3KTR%2F20210621%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Signature=e3833fa15098755ece94b8f88d32a563ad452977a399645ca96aa2200a4575f2",

"assetName": "wbp-test-001",

"expiresAt": "2021-06-21T10:32:42.115+00:00",

"timeout": "",

"statusFlag": true

}

Sample Failed Request:

http://localhost:9000/s3/bucket/create/wbp-test-001

Failed Response:

{

{

"url": "wbp-test-00",

"message": "Bucket does not exist in aws s3, create a bucket before generating presigned URL",

"statusFlag": false

}

STEP 8.

Method name: getAssetPresignedUrl

You can view the asset with the generatedPresignedUrl.

Sample Request:

http://localhost:9000/s3/view/asset/6d645322-5467-4eb2-9575-334b6cdeb00fwbp-test-005

Sample Request:

{

"assetName": "wbp-test-001",

"id": "ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001",

"signedUrl": "https://wbp-test-001.s3.us-west-2.amazonaws.com/ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210621T093242Z&X-Amz-SignedHeaders=host&X-Amz-Expires=3599&X-Amz-Credential=AKIASTMPXL2EMH7J3KTR%2F20210621%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Signature=e3833fa15098755ece94b8f88d32a563ad452977a399645ca96aa2200a4575f2"

}

Sample Success Response:

{
"id": "ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001",
"url": "https://wbp-test-001.s3.us-west-2.amazonaws.com/ca2c5be1-99d1-44c2-ae6d-b655762b9154wbp-test-001?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210621T101213Z&X-Amz-SignedHeaders=host&X-Amz-Expires=3599&X-Amz-Credential=AKIASTMPXL2EMH7J3KTR%2F20210621%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Signature=e0fbe3338274b7b5e5d60f1f079f50133edab71cd8e0ef842327d2f515acbd4b",
"assetName": "wbp-test-001",
"expiresAt": "2021-06-21T11:12:13.883+00:00",
"timeout": "false",
"statusFlag": true
}

STEP 9.

Method Name: deleteAsset

Instead of manually deleting the asset in the console, you can call this endpoint to delete asset

Sample Request:

http://localhost:9000/s3/delete/wbp-test-001

" assetName":" wbp-test-001"

Sample Success Response:

{
Bucket deleted successfully
}

Sample Failed Request:

http://localhost:9000/s3/delete/wbp-test-001

Failed Response:

{
No Bucket Found
}