Streaming Multipart Upload to S3 in Java

1. Set Up Your AWS S3 Client

Ensure you have the AWS SDK for Java dependency in your project. For Maven:

```
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
    <version>2.20.18</version> <!-- Replace with the latest version -->
</dependency>
```

Create the S3 client:

```
import software.amazon.awssdk.services.s3.S3Client;

S3Client s3Client = S3Client.create();
```

---

2. Initiate a Multipart Upload

Start a multipart upload session and retrieve the uploadId.

```
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;

String bucketName = "your-bucket";
```

```
String keyName = "your-key";

CreateMultipartUploadRequest createRequest = CreateMultipartUploadRequest.builder()
        .bucket(bucketName)
        .key(keyName)
        .build();

CreateMultipartUploadResponse createResponse = s3Client.createMultipartUpload(createRequest);
String uploadId = createResponse.uploadId();
```

---

3. Upload Parts in Chunks

Stream data in smaller chunks and upload each chunk as a part.

```
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.UploadPartResponse;
import software.amazon.awssdk.core.sync.RequestBody;

import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

List<CompletedPart> completedParts = new ArrayList<>();
int partNumber = 1;

try (InputStream inputStream = getInputStreamFromSource()) { // Replace with your data source
```

```
    byte[] buffer = new byte[5 * 1024 * 1024]; // 5 MB buffer
    int bytesRead;

    while ((bytesRead = inputStream.read(buffer)) != -1) {
        UploadPartRequest uploadPartRequest = UploadPartRequest.builder()
                .bucket(bucketName)
                .key(keyName)
                .uploadId(uploadId)
                .partNumber(partNumber)
                .build();

        UploadPartResponse uploadPartResponse = s3Client.uploadPart(uploadPartRequest,
                RequestBody.fromBytes(buffer, 0, bytesRead));

        completedParts.add(CompletedPart.builder()
                .partNumber(partNumber)
                .eTag(uploadPartResponse.eTag())
                .build());

        partNumber++;
    }
}
```

---

4. Complete the Multipart Upload

Once all parts are uploaded, complete the upload by combining the parts.

```
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;

CompletedMultipartUpload completedMultipartUpload = CompletedMultipartUpload.builder()
        .parts(completedParts)
        .build();

CompleteMultipartUploadRequest completeRequest = CompleteMultipartUploadRequest.builder()
        .bucket(bucketName)
        .key(keyName)
        .uploadId(uploadId)
        .multipartUpload(completedMultipartUpload)
        .build();

s3Client.completeMultipartUpload(completeRequest);
System.out.println("Multipart upload completed successfully!");
```

---

5. Abort Multipart Upload on Failure

If any part of the upload fails, abort the multipart upload to avoid leaving incomplete uploads in S3.

```
import software.amazon.awssdk.services.s3.model.AbortMultipartUploadRequest;

AbortMultipartUploadRequest abortRequest = AbortMultipartUploadRequest.builder()
```

```
        .bucket(bucketName)
        .key(keyName)
        .uploadId(uploadId)
        .build();

s3Client.abortMultipartUpload(abortRequest);
System.out.println("Multipart upload aborted.");
```

---

6. Helper: Streaming Data Source

Use a streaming data source, such as a database cursor, to feed chunks into the upload process.

```
private InputStream getInputStreamFromSource() {
    // Replace this with your data source logic (e.g., MongoDB cursor or file stream)
    // Example: Use a PipedInputStream to simulate streaming
    return new FileInputStream("path/to/your/large/file");
}
```

Key Points:

- Chunk Size: AWS S3 requires each part (except the last) to be at least 5 MB.
- Concurrency: To optimize performance, you can upload parts in parallel using CompletableFuture or ExecutorService.
- Error Handling: Ensure robust error handling to retry failed parts or abort the upload if necessary.