Documentation: ImageProcessor Lambda Function

Overview

The `ImageProcessor` is an AWS Lambda function designed to process images stored in an Amazon S3 bucket. It performs the following tasks:

1. Fetches an image from S3.
2. Processes the image (resizes it to 200x200 pixels).
3. Saves the processed image back to S3.
4. Determines the next image to process using metadata stored in S3.

This workflow operates in a manner conceptually similar to a **linked list**, where each image contains metadata pointing to the next image to process.

Code Structure

Handler Interface

- The class implements `RequestHandler<ImageEvent, ImageResponse>`.
  - **Input:** An `ImageEvent` object containing the image key.
  - **Output:** An `ImageResponse` object containing the next image key or a status message.

Key Components

AWS S3 Client

- Uses `AmazonS3ClientBuilder.defaultClient()` to create an S3 client for interacting with the S3 bucket.

Bucket Name

- The S3 bucket name is hardcoded: `"image-processing-bucket-spring-boot-backend"`.

Processing Workflow

1. **Fetch the Image:**
   - The function retrieves the specified image from the bucket using the `GetObjectRequest` API.

    - Defaults to `"image1.jpg"` if no key is provided in the input.
2. **Resize the Image:**
    - The image is resized to `200x200` pixels using the `Thumbnailator` library.
    - The resized image is stored in memory as a byte stream.
3. **Save Processed Image:**
    - The processed image is saved in the `processed/` subdirectory within the same bucket.
4. **Retrieve Next Image Key:**
    - The function retrieves metadata of the current image.
    - It checks for a custom metadata field, `next`, which points to the next image key.
5. **Return Result:**
    - If a `next` key exists, it is returned in the `ImageResponse`.
    - If no `next` key is present, the response indicates that all images have been processed.

Input Format

The Lambda function accepts an `ImageEvent` object with the following structure:

```
public class ImageEvent {
    private String imageKey; // The S3 key of the image to process

    // Getters and setters
}
```

- Example input:
- {
- "imageKey": "image1.jpg"
- }

If `imageKey` is null, the function defaults to processing `"image1.jpg"`.

Output Format

The function returns an `ImageResponse` object with the following structure:

```
public class ImageResponse {
    private String message; // Status or the key of the next image

    // Constructor, getters, and setters
}
```

- Example output:
- {
- "message": "image2.jpg"
- }

- If no `next` key is found in metadata:
- `{`
- `    "message": "All images processed"`
- `}`

Error Handling

- Logs errors using the `Context` logger.
- Throws a `RuntimeException` if image processing fails.

Conceptual Linked List Workflow

- Each image in the S3 bucket serves as a **node**.
- The custom metadata field `next` acts as a **pointer** to the next node (image).
- If the `next` field is `null`, it signifies the **end of the list**.

Example Workflow

1. **Input:** `"image1.jpg"`.
2. **Steps:**
   - Download `"image1.jpg"`.
   - Resize to `200x200`.
   - Save as `"processed/image1.jpg"`.
   - Retrieve metadata `next: "image2.jpg"`.
3. **Output:**
   - `ImageResponse: { "message": "image2.jpg" }`
4. **Repeat for `image2.jpg`, `image3.jpg`, ... until no `next` field exists.**

Dependencies

Libraries

1. **AWS SDK for Java**
   - For S3 operations (upload/download, metadata).
2. **Thumbnailator**
   - For image resizing.

Limitations

1. Hardcoded bucket name: Update if deploying in different environments.
2. Assumes the `next` metadata is always correctly set.
3. No handling of image format inconsistencies.

Enhancements (Future Scope)

1. Make the bucket name configurable via environment variables.
2. Add support for logging to AWS CloudWatch.
3. Handle unsupported image formats gracefully.
4. Implement error retries for S3 operations.

How to Deploy

1. Package the code as a `.jar` file.
2. Upload it to an AWS Lambda function.
3. Set necessary IAM roles for S3 access.
4. Configure the handler to
   `com.example.image.configure.ImageProcessor::handleRequest`.