aws-sdk-java

- Generate presigned URLs

import { Render } from "~/components";

This example uses version 2 of the aws-sdk-java package. You must pass in the R2 configuration credentials when instantiating your `S3` service client.

:::note

You must set `chunkedEncodingEnabled(false)` in the `S3Configuration` when building your client. The AWS SDK for Java v2 uses chunked transfer encoding by default for `putObject` requests, which causes a signature mismatch error (HTTP 403) with R2. Disabling chunked encoding ensures the request signature is calculated correctly. :::

```
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;
import software.amazon.awssdk.services.s3.S3Configuration;
import software.amazon.awssdk.core.sync.RequestBody;
import java.net.URI;
import java.util.List;

/**
 * Client for interacting with Cloudflare R2 Storage using AWS SDK S3 compatibility
 */
public class CloudflareR2Client {
    private final S3Client s3Client;

    /**
     * Creates a new CloudflareR2Client with the provided configuration
```

```
     */
    public CloudflareR2Client(S3Config config) {
        this.s3Client = buildS3Client(config);
    }

    /**
     * Configuration class for R2 credentials and endpoint
     * - accountId: Your Cloudflare account ID
     * - accessKey: Your R2 Access Key ID (see:
https://developers.cloudflare.com/r2/api/tokens)
     * - secretKey: Your R2 Secret Access Key (see:
https://developers.cloudflare.com/r2/api/tokens)
     */
    public static class S3Config {
        private final String accountId;
        private final String accessKey;
        private final String secretKey;
        private final String endpoint;

        public S3Config(String accountId, String accessKey, String secretKey) {
            this.accountId = accountId;
            this.accessKey = accessKey;
            this.secretKey = secretKey;
            this.endpoint = String.format("https://%s.r2.cloudflarestorage.com", accountId);
        }

        public String getAccessKey() { return accessKey; }
        public String getSecretKey() { return secretKey; }
        public String getEndpoint() { return endpoint; }
    }

    /**
     * Builds and configures the S3 client with R2-specific settings
     */
    private static S3Client buildS3Client(S3Config config) {
        AwsBasicCredentials credentials = AwsBasicCredentials.create(
            config.getAccessKey(),
            config.getSecretKey()
        );
```

```
        S3Configuration serviceConfiguration = S3Configuration.builder()
            .pathStyleAccessEnabled(true)
            .chunkedEncodingEnabled(false)
            .build();

        return S3Client.builder()
            .endpointOverride(URI.create(config.getEndpoint()))
            .credentialsProvider(StaticCredentialsProvider.create(credentials))
            .region(Region.of("auto")) // Required by SDK but not used by R2
            .serviceConfiguration(serviceConfiguration)
            .build();
    }

    /**
     * Lists all buckets in the R2 storage
     */
    public List<Bucket> listBuckets() {
        try {
            return s3Client.listBuckets().buckets();
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to list buckets: " + e.getMessage(), e);
        }
    }

    /**
     * Lists all objects in the specified bucket
     */
    public List<S3Object> listObjects(String bucketName) {
        try {
            ListObjectsV2Request request = ListObjectsV2Request.builder()
                .bucket(bucketName)
                .build();

            return s3Client.listObjectsV2(request).contents();
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to list objects in bucket " + bucketName + ": "
+ e.getMessage(), e);
        }
```

```
    }

    /**
     * Uploads an object to the specified bucket
     */
    public void putObject(String bucketName, String key, String content) {
        try {
            PutObjectRequest request = PutObjectRequest.builder()
                .bucket(bucketName)
                .key(key)
                .build();

            s3Client.putObject(request, RequestBody.fromString(content));
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to put object " + key + " in bucket " +
bucketName + ": " + e.getMessage(), e);
        }
    }

    public static void main(String[] args) {
        S3Config config = new S3Config(
            "<ACCOUNT_ID>",
            "<ACCESS_KEY_ID>",
            "<SECRET_ACCESS_KEY>"
        );

        CloudflareR2Client r2Client = new CloudflareR2Client(config);

        // List buckets
        System.out.println("Available buckets:");
        r2Client.listBuckets().forEach(bucket ->
            System.out.println("* " + bucket.name())
        );

        // Upload an object to a bucket
        String bucketName = "demos";
        r2Client.putObject(bucketName, "example.txt", "Hello, R2!");
        System.out.println("Uploaded example.txt to bucket '" + bucketName + "'");
```

```
        // List objects in a specific bucket
        System.out.println("\nObjects in bucket '" + bucketName + "':");
        r2Client.listObjects(bucketName).forEach(object ->
            System.out.printf("* %s (size: %d bytes, modified: %s)%n",
                object.key(),
                object.size(),
                object.lastModified())
        );
    }
}
```

Generate presigned URLs

You can also generate presigned links that can be used to temporarily share public write access to a bucket.

```
// import required packages for presigning
// Rest of the packages are same as above
import software.amazon.awssdk.services.s3.presigner.S3Presigner;
import software.amazon.awssdk.services.s3.presigner.model.PutObjectPresignRequest;
import software.amazon.awssdk.services.s3.presigner.model.PresignedPutObjectRequest;
import java.time.Duration;

public class CloudflareR2Client {
  private final S3Client s3Client;
  private final S3Presigner presigner;

    /**
     * Creates a new CloudflareR2Client with the provided configuration
     */
    public CloudflareR2Client(S3Config config) {
        this.s3Client = buildS3Client(config);
        this.presigner = buildS3Presigner(config);
    }

    /**
     * Builds and configures the S3 presigner with R2-specific settings
     */
    private static S3Presigner buildS3Presigner(S3Config config) {
```

```
        AwsBasicCredentials credentials = AwsBasicCredentials.create(
            config.getAccessKey(),
            config.getSecretKey()
        );

        return S3Presigner.builder()
            .endpointOverride(URI.create(config.getEndpoint()))
            .credentialsProvider(StaticCredentialsProvider.create(credentials))
            .region(Region.of("auto")) // Required by SDK but not used by R2
            .serviceConfiguration(S3Configuration.builder()
                .pathStyleAccessEnabled(true)
                .build())
            .build();
    }

    public String generatePresignedUploadUrl(String bucketName, String objectKey, Duration
expiration) {
        PutObjectPresignRequest presignRequest = PutObjectPresignRequest.builder()
            .signatureDuration(expiration)
            .putObjectRequest(builder -> builder
                .bucket(bucketName)
                .key(objectKey)
                .build())
            .build();

        PresignedPutObjectRequest presignedRequest =
presigner.presignPutObject(presignRequest);
        return presignedRequest.url().toString();
    }

    // Rest of the methods remains the same

    public static void main(String[] args) {
      // config the client as before

      // Generate a pre-signed upload URL valid for 15 minutes
        String uploadUrl = r2Client.generatePresignedUploadUrl(
            "demos",
            "README.md",
            Duration.ofMinutes(15)
        );
```

```
        System.out.println("Pre-signed Upload URL (valid for 15 minutes):");
        System.out.println(uploadUrl);
    }

}
```

□□□ #2
□ □□□ □□□ 2026-11-01 03:09:09 UTC
□ □□□ □□□ 2026-11-01 03:09:10 UTC