TRANSFER-MANAGER

-High level utility for managing transfers to Amazon S3.

-TransferManager provides a simple API for uploading content to Amazon S3, and makes extensive use of Amazon S3 multipart uploads to achieve enhanced throughput, performance and reliability.

-When possible, TransferManager attempts to use multiple threads to upload multiple parts of a single upload at once. When dealing with large content sizes and high bandwidth, this can have a significant increase on throughput.

TransferManager is responsible for managing resources such as connections and threads; share a single instance of TransferManager whenever possible**. TransferManager, like all the client classes in the AWS SDK for Java, is thread safe.** Call TransferManager.shutdownNow() to release the resources once the transfer is complete.

Behind this simple API, TransferManager is doing a lot of work for you. Depending on the size and data source for your upload, TransferManager adjusts the algorithm it uses to process your transfer, in order to get the best performance and reliability. Whenever possible, uploads are broken up into multiple pieces, so that several pieces can be sent in parallel to provide better throughput. In addition to higher throughput, this approach also enables more robust transfers, since an I/O error in any individual piece means the SDK only needs to retransmit the one affected piece, and not the entire transfer.

TransferManager includes several more advanced features, such as recursively downloading entire sections of S3 buckets, or the ability to clean up pieces of failed multipart uploads. One of the more commonly used options is the ability to attach a progress listener to your uploads and downloads, which can run custom code at different points in the transfer's lifecycle. The following example demonstrates using a progress listener to periodically print out the transfer's progress, and print a final message when the transfer completes.

(Upload 1 GB file and pause after 500Mb.)

JAVA CODE TO PAUSE UPLOAD:

**public** **void** pauseUploading(TransferManager tm, Upload upload) **throws** Exception{

**long** MB = 1024 \* 1024 ;

TransferProgress progress = upload.getProgress();

System.*out*.println("The pause will occur once 5 MB of data is uploaded");

**while**( progress.getBytesTransferred() < 5\*MB )

Thread.*sleep*(2000);

**boolean** forceCancel = **true**;

**float** dataTransfered = (**float**) upload.getProgress().getBytesTransferred();

System.*out*.println("Data Transfered until now: " + dataTransfered);

PauseResult<PersistableUpload> pauseResult = ((Upload) upload).tryPause(forceCancel);

System.*out*.println("The upload has been paused. The code that we've got is " + pauseResult.getPauseStatus());

pauseResult = ((Upload) upload).tryPause(forceCancel);

PersistableUpload persistableUpload = (PersistableUpload) pauseResult.getInfoToResume();

System.*out*.println("Storing information into file");

File f = **new** File("resume-upload");

**if**( !f.exists() )

f.createNewFile();

FileOutputStream fos = **new** FileOutputStream(f);

persistableUpload.serialize(fos);

fos.close();

}

JAVA CODE TO RESUME UPLOAD:

**public** **void** resumeUploading(TransferManager tm) **throws** Exception{

FileInputStream fis = **new** FileInputStream(**new** File("resume-upload"));

System.*out*.println("Reading information from the file");

PersistableUpload persistableUpload;

persistableUpload = PersistableTransfer.*deserializeFrom*(fis);

System.*out*.println("Reading information completed");

System.*out*.println("The system will resume upload now");

Upload upload =tm.resumeUpload(persistableUpload);

**while**(!upload.isDone())

{

System.*out*.println("Progress... :" + upload.getProgress().getBytesTransferred());

Thread.*sleep*(2500);

}

fis.close();

//System.out.println("Upload complete.");

}

JAVA CODE FOR MAIN CLASS:

**public** **static** **void** main(String[] args) **throws** Exception {

String existingBucketName = "onkaarbucket";

String keyName = "EXE\_File1";

String filePath = "C:\\Users\\onkaar\_singh\\Downloads\\S3File";

AmazonS3Basic amazonS3Basic = **new** AmazonS3Basic();

AmazonS3 amazonS3Client = AmazonS3Basic.*getS3*();

TransferManagerConfiguration configuration = **new** TransferManagerConfiguration();

TransferManager tm = **new** TransferManager(amazonS3Client);

configuration.setMultipartUploadThreshold(1024 \* 1024);

tm.setConfiguration(configuration);

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\* Upload Manager \*\*\*\*\*\*\*\*\*\*\*\*\*");

**try** {

Upload upload = tm.upload(existingBucketName, keyName, **new** File(filePath));

System.*out*.println("Upload Started");

System.*out*.println("Transfer: " + upload.getDescription());

/\*name of the class containing above defined pause and resume methods is UploadObjectMultipartUploadUsingHighLevelAPI\*/

UploadObjectMultipartUploadUsingHighLevelAPI multipartPause =

**new** UploadObjectMultipartUploadUsingHighLevelAPI();

multipartPause.pauseUploading(tm, upload);

System.*out*.println("PAUSED...!!");

Thread.*sleep*(3000);

System.*out*.println("RESUME...!");

UploadObjectMultipartUploadUsingHighLevelAPI multipartResume =

**new** UploadObjectMultipartUploadUsingHighLevelAPI();

multipartResume.resumeUploading(tm);

}

**catch** (AmazonClientException amazonClientException) {

System.*out*.println("Unable to upload file, upload was aborted.");

amazonClientException.printStackTrace();

}

}