Apex code:-

public class XFP\_S3FileSyncBatch implements Database.Batchable<SObject>, Database.Stateful,Database.AllowsCallouts {

private Boolean convert18To15;

private String objName;

private XFILES\_\_Cloud\_Storage\_Provider\_\_c storageProvider;

private String accessKey;

private String secretKey;

private String keyPrefix;

private String service = 's3';

private String region;

private Datetime now = Datetime.now();

private Integer failedCount = 0;

private List<Id> ecdIds;

private Map<String, String> failedRecordsMap = new Map<String,

String>();

public XFP\_S3FileSyncBatch(Boolean convert18To15, String objName,List<Id> ecdIds) {

this.convert18To15 = convert18To15;

this.objName = objName;

this.ecdIds = ecdIds;

storageProvider = [

SELECT Id, XFILES\_\_Root\_Folder\_Name\_\_c,

XFILES\_\_GDrive\_Root\_Folder\_Id\_\_c, XFILES\_\_Password\_\_c,

XFILES\_\_Username\_\_c

FROM XFILES\_\_Cloud\_Storage\_Provider\_\_c

WHERE XFILES\_\_Provider\_Type\_\_c = 'Amazon-S3'

LIMIT 1

];

keyPrefix = Schema.describeSObjects(new

List<String>{objName})[0].getKeyPrefix();

accessKey =

XFILES.XfileCryptoController.decrypt(storageProvider.XFILES\_\_Username\_\_c);

secretKey =

XFILES.XfileCryptoController.decrypt(storageProvider.XFILES\_\_Password\_\_c);

region = storageProvider.XFILES\_\_GDrive\_Root\_Folder\_Id\_\_c;

}

public Database.QueryLocator start(Database.BatchableContext bc) {

String query = 'SELECT Id, XFILES\_\_Content\_Remote\_Path\_\_c, XFILES\_\_Client\_Details\_\_c ' +

'FROM XFILES\_\_External\_Content\_Detail\_\_c ';

if (ecdIds != null && !ecdIds.isEmpty()) {

query += 'WHERE Id IN :ecdIds';

} else {

query += 'WHERE XFILES\_\_Client\_Details\_\_c LIKE \'' + keyPrefix + '%\'';

}

return Database.getQueryLocator(query);

}

public void execute(Database.BatchableContext bc,

List<XFILES\_\_External\_Content\_Detail\_\_c> contentDetails) {

List<XFILES\_\_External\_Content\_Detail\_\_c> updatedContentDetails =

new List<XFILES\_\_External\_Content\_Detail\_\_c>();

for (XFILES\_\_External\_Content\_Detail\_\_c contentDetail :

contentDetails) {

try {

String clientId = contentDetail.XFILES\_\_Client\_Details\_\_c;

String updatedClientId = convert18To15 ? clientId.substring(0,

15) : clientId;

String updatedPath;

if (convert18To15) {

if

(contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c.contains(clientId) &&

!contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c.contains(clientId.substring(0, 15) + '000')) {

updatedPath =

contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c.replace(clientId,clientId.substring(0, 15));

} else {

updatedPath =

contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c;

}

} else {

if

(contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c.contains(clientId.substring(0, 15)) &&!contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c.contains(clientId)) {

updatedPath =

contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c.replace(clientId.substring(0, 15), clientId);

} else {

updatedPath =

contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c;

}

}

if (contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c !=

updatedPath) {

String destinationUrl = 'https://' +

storageProvider.XFILES\_\_Root\_Folder\_Name\_\_c +

'.s3-' + region + '.amazonaws.com/' +

EncodingUtil.urlEncode(updatedPath, 'UTF-8');

HttpRequest request = signedRequest('PUT', new

URL(destinationUrl),

new Map<String,

String>{'x-amz-copy-source' => '/' +

storageProvider.XFILES\_\_Root\_Folder\_Name\_\_c + '/' +

EncodingUtil.urlEncode(contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c

, 'UTF-8')},

null, false);

HttpResponse response = new Http().send(request);

if (response.getStatusCode() == 200) {

contentDetail.XFILES\_\_Content\_Remote\_Path\_\_c =

updatedPath;

updatedContentDetails.add(contentDetail);

} else {

failedRecordsMap.put(contentDetail.Id, 'HTTP ' +

response.getStatusCode() + ' - ' + response.getStatus());

failedCount++;

}

}

if (failedCount > 10) {

System.abortJob(bc.getJobId());

}

} catch (Exception e) {

failedRecordsMap.put(contentDetail.Id, e.getMessage());

failedCount++;

System.debug('Error updating file: ' + e.getMessage());

}

}

if (!updatedContentDetails.isEmpty()) {

update updatedContentDetails;

}

}

public void finish(Database.BatchableContext bc) {

if (failedCount > 0) {

sendFailureEmail();

}

System.debug('Batch process completed.');

}

private void sendFailureEmail() {

String adminEmail =

XFILES\_\_xFiles\_custom\_setting\_\_c.getOrgDefaults().XFILES\_\_Admin\_Mail\_Address\_\_c;

List<String> emailBodyLines = new List<String>{

'Dear Admin,',

'','We wanted to inform you that the recent S3 File Sync Batchprocess completed successfully, but a few files could not be updated. Hereare the details:',''};

for (String recordId : failedRecordsMap.keySet()) {

emailBodyLines.add('ECD Id: ' + recordId + ' | Error: ' +

failedRecordsMap.get(recordId));

}

Messaging.SingleEmailMessage email = new

Messaging.SingleEmailMessage();

email.setToAddresses(new List<String>{adminEmail});

email.setSubject('Action Required: Some Files Failed in S3 SyncProcess');

email.setPlainTextBody(String.join(emailBodyLines, '\n'));

Messaging.sendEmail(new

List<Messaging.SingleEmailMessage>{email});

}

private HttpRequest signedRequest(String method, Url endpoint,

Map<String, String> headers, Blob payload, Boolean presign) {

if (headers == null) headers = new Map<String, String>();

if (payload == null) payload = Blob.valueOf('');

if (presign == null) presign = false;

String termination = 'aws4\_request';

String iso8601date = now.formatGmt('yyyyMMdd');

String iso8601time = now.formatGmt('yyyyMMdd\'T\'HHmmss\'Z\'');

String credentialScope = iso8601date + '/' + region + '/' + service + '/' +

termination;

headers.put('Host', endpoint.getHost());

String signedHeaders = signedHeadersFor(headers);

String spec = endpoint.getProtocol() + '://' + endpoint.getHost() +

rfc3986For(endpoint.getPath(), false);

if (endpoint.getQuery() != null) spec += '?' + endpoint.getQuery();

PageReference pr = new PageReference(spec);

Map<String, String> parameters = pr.getParameters();

parameters.put('X-Amz-Algorithm', 'AWS4-HMAC-SHA256');

parameters.put('X-Amz-Credential', accessKey + '/' +

credentialScope);

parameters.put('X-Amz-Date', iso8601time);

parameters.put('X-Amz-Expires', '86400');

parameters.put('X-Amz-SignedHeaders', signedHeaders);

String canonicalRequest = canonicalMethodFor(method)

+ '\n' + canonicalUriFor(endpoint.toExternalForm())

+ '\n' + canonicalQueryStringFor(parameters)

+ '\n' + canonicalHeadersFor(headers)

+ '\n' + signedHeadersFor(headers)

+ '\n' + (presign ? 'UNSIGNED-PAYLOAD' :

hexEncodedHashFor(payload));

String algorithm = 'AWS4-HMAC-SHA256';

String canonicalRequestHash =

hexEncodedHashFor(Blob.valueOf(canonicalRequest));

String stringToSign = algorithm + '\n' + iso8601time + '\n' +

credentialScope + '\n' + canonicalRequestHash;

Blob keySecret = Blob.valueOf('AWS4' + secretKey);

Blob keyDate = Crypto.generateMac('hmacSHA256',

Blob.valueOf(iso8601date), keySecret);

Blob keyRegion = Crypto.generateMac('hmacSHA256',

Blob.valueOf(region), keyDate);

Blob keyService = Crypto.generateMac('hmacSHA256',

Blob.valueOf(service), keyRegion);

Blob keySigning = Crypto.generateMac('hmacSHA256',

Blob.valueOf('aws4\_request'), keyService);

Blob blobToSign = Blob.valueOf(stringToSign);

Blob hmac = Crypto.generateMac('hmacSHA256', blobToSign,

keySigning);

if (!presign) headers.put('X-Amz-Content-Sha256',

hexEncodedHashFor(payload));

String signature = EncodingUtil.convertToHex(hmac);

parameters.put('X-Amz-Signature', signature);

HttpRequest request = new HttpRequest();

request.setMethod(method);

request.setEndpoint(pr.getUrl());

if (payload != Blob.valueOf('')) request.setBodyAsBlob(payload);

for (String header : headers.keySet()) request.setHeader(header,

headers.get(header));

return request;

}

private static String canonicalMethodFor(String method) {

return method.toUpperCase();

}

private static String canonicalUriFor(String endpoint) {

Url uri = new Url(endpoint);

return rfc3986For(uri.getPath(), false);

}

private static String canonicalQueryStringFor(Map<String, String>

parameters) {

List<String> sortedKeys = new List<String>(parameters.keySet());

sortedKeys.sort();

List<String> canonicalParameters = new List<String>();

for (String sortedKey : sortedKeys)

canonicalParameters.add(sortedKey + '=' +

rfc3986For(parameters.get(sortedKey), true));

return String.join(canonicalParameters, '&');

}

private static String canonicalHeadersFor(Map<String, String>

key2value) {

Map<String, String> lower2value = new Map<String, String>();

for (String key : key2value.keySet())

lower2value.put(key.toLowerCase(),

key2value.get(key).trim().replaceAll('\\s+', ' '));

List<String> sortedKeys = new List<String>(lower2value.keySet());

sortedKeys.sort();

List<String> canonicalHeaders = new List<String>();

for (String sortedKey : sortedKeys) canonicalHeaders.add(sortedKey

+ ':' + lower2value.get(sortedKey) + '\n');

return String.join(canonicalHeaders, '');

}

private static String signedHeadersFor(Map<String, String> headers) {

List<String> keys = new List<String>(headers.keySet());

for (Integer i = 0; i < keys.size(); i++) keys.set(i,

keys[i].toLowerCase());

keys.sort();

List<String> signedHeaders = new List<String>();

for (String key : keys) signedHeaders.add(key);

return String.join(signedHeaders, ';');

}

private static String hexEncodedHashFor(Blob data) {

Blob hash = Crypto.generateDigest('SHA256', data);

return EncodingUtil.convertToHex(hash);

}

private static String rfc3986For(String characters, Boolean encodeSlash)

{

String result = '';

String encodedURL;

characters = EncodingUtil.urlDecode(characters, 'UTF-8');

for (Integer i = 0; i < characters.length(); i++) {

String character = characters.substring(i, i + 1);

if ((character.hashCode() >= 65 && character.hashCode() <= 90) ||

(character.hashCode() >= 97 && character.hashCode() <= 122) ||

(character >= '0' && character <= '9') ||

character == '\_' ||

character == '-' ||

character == '~' ||

character == '.' ) {

result += character;

} else if (character == '/') {

result += encodeSlash ? '%2F' : character;

} else {

encodedURL =

EncodingUtil.convertToHex(Blob.valueOf(character)).toUpperCase();

if (encodedURL.length() > 2) {

encodedURL = encodedURL.replaceAll('..(?!$)', '$0 ');

for (String sd : encodedURL.split(' ')) {

result += '%' + sd;

}

} else {

result += '%' + encodedURL;

}

}

}

return result;

}

}

**Run the following code in the anonymous window :**

Boolean convert18To15 = true; (true if files should be moved from 18 to 15 else false). String objName = ‘Account’;(Object which needs to be updated)

List<Id> ecdIds=new List<Id>{};( List of External\_Content\_Detail\_\_c IDs that need to be processed)  
Database.executeBatch(new XFP\_S3FileSyncBatch(convert18To15 ,objName))