Atlantis File Processing

Incidents

Files Were Not Being Sent to File Processing

We had an incident where files were not sent to file processing for about 1 hour. We wrote a PostgreSQL query to get all the required details (that are required for sending to file processing) as a JSON array for the files uploaded in that time period, saved the JSON array in a file, then queued them one at a time just using a C# console app.

PostgreSQL Query

```
select json_agg(
  json_build_object(
'NodeID', public."DbNodes"."Id",
'AccountID', public."DbNodes"."AccountID",
'AccountIdentifier', public."DbAccounts"."Identifier",
'FileInfoID', public."DbNodes"."FileInfoID",
'FileSize', public."DbFileInfoes"."SizeInBytes",
'FileIdentifier', public."DbFileInfoes"."FileIdentifier",
'CreatedByUserID', public."DbNodes"."CreatedByUserID",
'FileName', public."DbNodes"."Name",
'EncryptionVersion', public."DbFileInfoes"."EncryptionVersion",
'TryCountRemaining', 3))
from public."DbNodes"
left outer join public."DbAccounts" on public."DbNodes"."AccountID" = public."DbAccounts"."Id"
left outer join public."DbFileInfoes" on public."DbNodes"."FileInfoID" = public."DbFileInfoes"."Id"
where public."DbNodes"."Id" in (
select "NodeID" from public."DbAuditLogsToNodes" where "AuditLogID" in (
select "Id" from public."DbAuditLogs"
where public."DbAuditLogs"."ItemType" = 0
and public."DbAuditLogs"."ActionType" = 100
and public."DbAuditLogs"."Date" > '2024-06-10 16:50:00.00'
and public."DbAuditLogs"."Date" < '2024-06-10 17:53:00.00'))
```

C# Code from Console App

I put breakpoints where the comments indicate, so I could manually retry when errors occured (which did happen several times due to 429 errors w/ Cosmos)

```
using ConsoleApp1;
using Newtonsoft.Json;
using System.Text;

using HttpClient client = new HttpClient();
var nodesToProcess = JsonConvert.DeserializeObject<IndexOCRInput[]>(File.ReadAllText(@"path to file with
JSON data"));
foreach (var node in nodesToProcess)
{
    try
    {
        string json = JsonConvert.SerializeObject(node);
        var httpContent = new StringContent(json, Encoding.UTF8, "application/json");

        string url = $@"https://revver-fileprocessingmanager-
us.azurewebsites.net/api/account/{node.AccountIdentifier}/node/{node.NodeID}/index?code={app key here}";
        var response = await client.PostAsync(url, httpContent);

        if (!response.IsSuccessStatusCode)
        {
            var v = ""; // put a breakpoint here
        }

        await Task.Delay(100); // this may not be needed if there is not much traffic
    }
    catch (Exception ex)
    {
        var w = ""; // put a breakpoint here
    }
}

var s = ""; // put a breakpoint here

/* Classes for Main Method to Function */

public class IndexOCRInput
{
```

```
    public long NodeID { get; set; }
    public int AccountID { get; set; }
    public Guid AccountIdentifier { get; set; }
    public AccountFeatureEnum AdvancedOcrAccountFeature { get; set; } = AccountFeatureEnum.AccusoftOCR;
    public PermanentFileStorageTypeEnum AccountStorageType { get; set; } =
PermanentFileStorageTypeEnum.AWS_S3;
    public EncryptionVersionEnum EncryptionVersion { get; set; }
    public long FileInfoID { get; set; }
    public long? FileSize { get; set; }
    public string FileIdentifier { get; set; }
    public long CreatedByUserID { get; set; }
    public string FileName { get; set; }
    public int TryCountRemaining { get; set; }
}

public enum AccountFeatureEnum
{
    FullUserLicense = 1,
    GuestUserLicense = 2,
    Governance = 3,
    FileStorageSize = 4, // Specifies the storage size limit
    FileVersioning = 5,
    Templates = 6,
    CheckInOut = 7,
    AccusoftPreview = 8,
    //SideKickLicense = 9,
    //MobileAppLicense = 10,
    ZonalOCRLicense = 11,
    AccountTemplateFeature = 12,
    FullTextSearch = 13,
    PersonalProviders = 14,
    DocumentRequests = 15,
    GovernanceLock = 16,
    AccusoftOCR = 17,
    Workflow = 19,
    Salesforce = 20,
    ItemStatus = 21,
    EnterpriseSecurity = 22,
    GovernanceUnlock = 23,
    Branding = 24,
```

```
    GuestUserPreview = 25,
    FilePassword = 26,
    GuestUserSearch = 27,
    ConcurrentLicensing = 28,
    EmailFiles = 29,
    BetaAccount = 30,
    Records = 31,
    SSO_SAML = 32,
    EmailImport = 33,
    PreviewerImageStamps = 34,
    ESignature = 35,
    DualScreenPreview = 36,
    ESignatureKBA = 37,
    ESignatureOTP = 38,
    AnonymousAccessLinks = 39,
    SearchReports = 40,
    Watermark = 41,
    Reporting = 42,
    FormFill = 43,
    PriorityOCR = 44,
    O365 = 45,
    WorkflowStepsPerWorkflow = 46,
    AbbyFineReaderOCR = 47,
    //LDAP = 48, //LDAP may have already been issued in the hub, so don't reuse this number and if we need to re-key LDAP, just use this number
    EssentialsUserLicense = 49,
    PublicShareSearch = 50,
    ComplianceUserLicense = 51,
}

public enum PermanentFileStorageTypeEnum
{
    AzureBlobStorage = 0,
    AWS_S3 = 1,
    Local = 2,
}

public enum EncryptionVersionEnum
{
    UtopiaV1 = 0,
```

```
    LegacyDesktop = 1,
    None = 3
}
```

Files Expired From Azure Storage Queues

QA did a mass apply that got out of hand, it ended up adding so many files to the file processing queues, that we had many files queued in file processing expire out of the queue after 7 days because file processing did not process them fast enough. Luckily, the state of each file that is being processed is stored in Cosmos DB so we added some functions that, given a specific time range, will go and requeue items that did not finish processing in that time period.

Steps to Do This

1. Use the query at the bottom of this page to see how many items were potentially dropped from the queue in a given time period.
   - Dates should be in UTC and formatted like this => '2024-08-05T00:00:00.000000Z'
   - When you run the http function in the next step, this is how many items will be put in the `processrequeuedexpireditemsqueue` queue in the `revvertextextractprocus` storage account.
   - If there are more than 200k items queued it will likely fail to queue all items successfully (which is why you are doing this step in the first place, to validate the expected number of items actually get queued), so you may need to narrow your time window and run this process multiple times if there are more than 200k items.
2. Make sure the `ProcessRequeuedExpiredItems` function in the `revver-fileProcessingManager` is disabled.
3. Run the http triggered `RequeueProcessingExpiredItemsFunction` function in the `revver-fileProcessingManager` function app with a request body like the following.
   - ```
     {
       "StartTime": "2024-07-31T00:00:00.000000Z",
       "EndTime": "2024-08-03T00:00:00.000000Z",
       "AccountIdsToExclude": [6878]
     }
     ```
   - The `AccountIdsToExclude` property allows you to not requeue items from a specific account (it's possible you just want this to be an empty list, in the first case, we didn't want to requeue items in the QA account)
   - The http triggered function will likely appear to time out, but it should continues processing regardless, but you will validate that the expected number of items were
     ```

queued in the next step for this reason.

4. Ensure the expected number of items are placed in the `processrequeuedexpireditemsqueue` queue in the `revvertextextractprocus` storage account
   - you validate this against the result from you query in step 1
   - if it didn't queue the expected number, clear the queue, and narrow your time window
5. disable the InitiateOCRExtraction function in the revver-ocrProcessing function app
6. disable the InitiateTextExtraction function in the revver-textExtractionProcessing function app
7. enable the ProcessRequeuedExpiredItems function in the revver-fileProcessingManager function app
8. monitor the `processrequeuedexpireditemsqueue` queue in the `revvertextextractprocus` storage account and wait for the queue to be empty
   - as items are processed in this queue, messages will be added to the initiate ocr and text extraction queues
   - not all items will be requeued to either queue, some items we cannot determine were not processed or not until this step, so often many of the items will not be requeued.
9. disable the ProcessRequeuedExpiredItems function in the revver-fileProcessingManager function app
10. enable the InitiateOCRExtraction function in the revver-ocrProcessing function app
11. enablethe InitiateTextExtraction function in the revver-textExtractionProcessing function app

Query for expected number of items to be queued (run in cosmos db file processing items container data explorer)

```
SELECT VALUE COUNT(1) FROM c
WHERE
c.AccountID NOT IN (6878)
AND (
   (c.Timestamp > '{start time here}' AND c.Timestamp < '{end time here}')
   OR
   (c.Pipeline.TextExtractionProcessing != null AND c.Pipeline.TextExtractionProcessing.Date != null and
(c.Pipeline.TextExtractionProcessing.Date > '{start time here}' and c.Pipeline.TextExtractionProcessing.Date <
'{end time here}')
   )
   OR
   (c.Pipeline.OCRProcessing != null AND c.Pipeline.OCRProcessing.Date != null and
(c.Pipeline.OCRProcessing.Date > '{start time here}' and c.Pipeline.OCRProcessing.Date < '{end time here}')
   )
)
AND (
```

```
    (
        c.Pipeline.TextExtractionProcessing != null AND (c.Pipeline.TextExtractionProcessing.Status = 'InProgress'
OR c.Pipeline.TextExtractionProcessing.Status = 'NotStarted')
    )
    OR
    (
        (c.Pipeline.TextExtractionProcessing != null AND c.Pipeline.OCRProcessing != null)
        AND
        (
            c.Pipeline.TextExtractionProcessing.Status = 'NotApplicable'
            OR
            c.Pipeline.TextExtractionProcessing.Status = 'Complete'
            OR
            c.Pipeline.TextExtractionProcessing.Status = 'Error'
        )
        AND
        (
            c.Pipeline.OCRProcessing.Status = 'InProgress'
            OR
            c.Pipeline.OCRProcessing.Status = 'NotStarted'
        )
    )
)
```

Files Didn't Finish Processing Because ElasticSearch DB Node was dropped

We had an incident close to when we migrated our ElasticSearch DB to Elastic Cloud, where we did not have replicas of our elastic search shards, and during regular maintenance (seriously, they said it was regular maintenance) they decided to remove a node from our cluster with a new one without copying over the data from the original node. This led to about 10% of are data being deleted and 100 or so unassigned shards that we could not read or write to until we restored our elastic search db from a backup.

This caused lots of files being processed to fail to process at the very end of processing because their extracted text could not be saved to elastic search. Also, because we were restoring the Elastic Search DB from a backup, the files that did process successfully would need to be reindexed in the elastic search db because all data that had been added since the incident was lost from the db (because we restored from a backup).

To remediate this situation, we added some functions to file processing, that given a specific time period, would reprocess all items, so that extracted text could be sent to elastic search. We add to modify the file processing process a little bit, and make it so that when reprocessing, a new file version wouldn't be created again if that had already happened. Note that this process ended up being quite finicky, and required quite a bit of manual effort so that the cosmos db was not overloaded.

We have since added replicas to our elastic search db, and don't expect an incident like this to happen again.

We have also made it so if a file is completely processed except for saving extracted text to elastic search, it will put the identifying file information in special queue with an infinite expiration, so that when elastic search is back up again, we can just process this queue and only send their extracted text to elastic search, instead of completely sending the item through file processing again.

How to do this?

It's been to long since we did this for me to document in great detail how you do this (likely you won't need to do this in the future again either because of the remediation steps we've taken) but I

will do my best to write a guide, just in case.

You'll want to have a dev build out a cosmos db query that you can run manually (sorry I don't know where it got put) from the query that is run the RequeueProcessingItemsFunction in the fileProcessingManager function app that gets all the items to be requeued, so you can use it to make sure everything gets queued.

You will want to disable other file processing while you queue things for processing so you don't overload cosmos db.

When you run the RequeueProcessingItemsFunction, you will want to disable the `ProcessRequeuedItems` function in the `revver-fileProcessingManager` function app, and make sure that expected number of items get placed in the `processrequeueditemsqueue` queue in the `revvertextextractprocus` storage account. (use the query generated previously to get expected number of items to queue).

Once you've validated the expected number of items were queued (you can clear the queue and re queue with a smaller time window if needed, but avoid if possible because the same item can potentially appear in different time windows just becuase of the way items are timestamped during file processing), enable the `ProcessRequeuedItems` function, but only do this for about a minute, after a minute disable and let things process for a few minutes. If you don't do this, Cosmos Db will be overwhelmed and you will start getting unexpected errors.

Once you've done the previous step, and the queue is empty, you should disable the ProcessRequeuedItems function, and re-enable the normal file processing functions that you disabled, and file processing should work per usual (you will just have quite a few items in the initiate ocr and text extraction queues that will need to be processed)