Overview:

Update your Note Keeper REST API application to create and store a single zip blob containing all the attachments **for a given note**. You will create a new project using your completed assignment #3 code base. All functionality in assignments 2 and 3 must still meet assignment 2 & 3's requirements and operate correctly.

You will use Azure Storage to store the resultant zip blob under a container name with the associated note's id and a **suffix** of **-zip**.

You shall use an **Azure Function** with **.NET 9 Isolated** to create the zip blob.

You shall create a new REST **sub-resource** that will support zip archive creation requests via the POST verb, a listing of all available zip archives via the GET verb, retrieval of a specific zip archive via the GET verb with a specific zip archive id, and deletion of a zip archive via the DELETE verb specifying the specific zip archive to delete.

Note: The process of compressing all the attachments is an **asynchronous operation**. This means the zip archive may not be immediately available. The caller is expected to poll the new REST sub-resource using GET verb and the specific zip archive id. There is no notification or tracking as part of the core requirements. There is a job tracking option as an extra credit option.

You will use Visual Studio to deploy your **API application** to a **NEW Azure App Service** hosted you're your **existing Linux** **Azure App Service Plan** that uses the **basic tier**.

Create a new database for this assignment so that you don't impact the grading that is occurring against assignment 3. Ensure that the seeded notes and attachments are properly populated in the database and azure storage, respectively.

You will use Visual Studio to deploy your **Azure Function** that compresses attachments to a **new Linux Azure Function App** in the **consumption plan**.

Be sure to read the **Implementation Notes** section, as it has important guidelines and restrictions.

Goals

- Setup and configure
    - Azure Functions with .NET 9 Isolated
    - Azure Storage Queue
- Familiarize yourself with
    - Azure Functions
    - Azure Storage Queues
    - Optionally: Azure Storage Tables
- Gain experience with:
    - Azure Storage
    - Azure Queues
    - Azure Blob Storage
    - Azure SQL Server & Database
    - Visual Studio 2022 development environment
    - .NET 9 ASP.NET Core Web API Projects
    - Building REST interfaces that run in Azure App Services
    - Azure App Services
    - Azure App Service Plans
    - Debugging
        - Web API App Services from Visual Studio
        - Web API App Services from the Azure Dashboard Logging

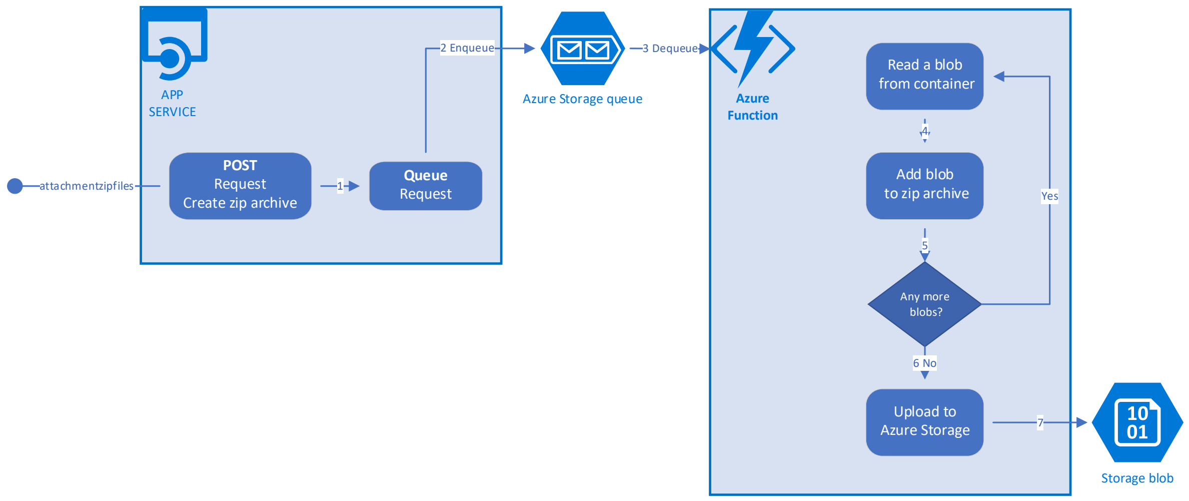


Table of contents

Solution, project naming, and submission information

You shall create one Solution containing **two** Visual Studio Projects. The solution shall be called **HW4NoteKeeperSolution**

The solution shall contain two projects:

1. **HW4NoteKeeper** shall contain the REST interface web app implementation.
2. **HW4AzureFunctions** shall contain the azure function implementation.

Note:

- Do not include any Azure credential information in your submission unless specified in the requirements. Double check your **appsettings json** files to be sure you have cleansed them of credentials prior to submission.
- Do not include files under the **bin** directory and the **Properties\PublishProfiles** directory.

As stated in the syllabus: The completed assignment shall be placed in a single zip file that preserves the directory structure. and must be submitted to the assignment's associated drop box on the course canvas website. Submission by any other means will not be accepted. The zip file shall contain:

a) All assets such as configuration files, images, initialization scripts and resources that are part of the application. The TA must be able to build and deploy your homework assignment to Azure and experience the full functionality of your application.
b) A text file named **ProjectNotes.txt** that includes:
    1) The homework assignment title and number
    2) Your name and email address
    3) **Attribution regarding the use of AI in the development of your assignment**
    4) Any notes for the TA that may be needed for the TA to install, set up, login to and operate your homework assignment.

5) Your Azure **AI Foundry Hub** and **Project name**
6) The name of the Azure SQL **Server** you created.
7) The name of the Azure SQL **Database** you created.
8) The **URL** to the **Azure SQL Server** with the **Database** name included
9) The name of your Azure Storage account
10) The name of your Azure Function
11) Any custom Azure resource abbreviations you used
    1. You can the abbreviation recommendations for Azure resources here: [Abbreviation recommendations for Azure resources - Cloud Adoption Framework | Microsoft Learn]
12) The **URL** to **Azure App Service Website**.
13) The target **Uri Azure OpenAI Service**
14) Don't include the compiled output
    1. **Do not include the publish, bin or obj directory**

If you have any questions regarding these guidelines, please ask a TA or the instructor.

1. Implement attachmentzipfiles sub resource

1.1 API Operation – Request zip file to be created

Requests a zip archive file to be created, which is stored as a blob in Azure Storage, of all the attachments associated with a note. The request is submitted to an **Azure Storage Queue** named **attachment-zip-requests**. The message enqueued contains the **note id** of the note whose attachments are requested to be compressed into a single zip archive and the **name** of the **zip archive**, which will become the blob id of the blob stored in Azure Storage. The zip archive shall be stored as a single blob. See the ZipArchive Class

HTTP
POST `https://[appservicename].azurewebsites.net/notes/{`**`noteId`**`}/attachmentzipfiles`

URI Parameters

Name	Data Type	.NET Type	Min Length	Max Length	Description
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist

Request Body

None

Operation

1.1.1 Create a message containing the noteId and the zipFileId to create as a JSON object. The zipFileId consists of a newly generated **guid** by your server code with a **suffix** of **.zip**.

Example:

{
"noteId":"ad698324-7b23-48ce-a13c-6e91f6664786",
"zipFileId":"4b7134af-6cc3-4aad-bb54-0da2e0f07928.zip"
}

1.1.2 After the message is enqueued successfully return an **HTTP 202 Accepted** with a **location header containing the full URL to retrieve the zip file**, which is stored as a blob. The blob will be available once it has been created by the Azure Function.

1.1.3 If the **note is NOT** present return a 404 (Not Found) and log the operation using a Log Warning

1.1.4. Handle the race condition scenario where a note has been deleted but a request to compress all attachment files is present in the queue by logging an error with the content: "**The note <note id> can't be found for the requested compression operation.**"

1.1.5. If there are no attachments associated with the note, return an HTTP 204 No Content to indicate that there is nothing to do.

Responses

202 (Accepted)

Returned if the request to create the zip file, which is stored as a blob, is successfully enqueued in the **attachment-zip-requests** queue.

Headers:
Location consists of the complete URL to retrieve the zip file of attachments.

Response Body:
None

Example:
`Assuming the` *`appservicename`* `is` **`noteswithattachments`**`, the` **`noteId`** `is` **`ad698324-7b23-48ce-a13c-6e91f6664786`** `and the` **`zipFileId`** `is "`**`4b7134af-6cc3-4aad-bb54-0da2e0f07928.zip`**`" the response shall be:`

Headers:
Location:

```
Location: https://noteswithattachments/notes/ad698324-7b23-48ce-a13c-
6e91f6664786/attachmentzipfiles/4b7134af-6cc3-4aad-bb54-0da2e0f07928.zip
```

Response Body:
None

204 (No content)
Returned if there are no attachments associated with the note.

Headers:
None beyond standard

Response Body:
No response body is returned.

400 (Bad Request)
Returned if there are one or more invalid parameters.

Headers:
None beyond standard

Response Body:
No custom response body is required, you can use the default response body returned by ASP.NET Core.

404 (Not Found)
Returned if the note could not be found.

Headers:
None beyond standard

Response Body:
No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.2 API Operation – Delete

Deletes an attachment zip file having the specified zip file id from Azure Blob Storage.

HTTP
DELETE `https://[appservicename].azurewebsites.net/notes/{`**`noteId`**`}/attachmentzipfiles/{`**`zipFileId`**`}`

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist
zipFileId	string	string	Not enforced	Not enforced	The zip file Id of the attachment zip file. This is the attachment zip file id associated with the note that was created on behalf of the POST operation. See 1.1 API Operation – Request zip file to be created **Constraints**: Required, Must Exist.

Operation

1.2.1 If the attachment zip blob is present but there is a failure to delete the attachment zip blob, return a 500 (Internal Server Error) and log the issue using a **Log Error**

1.2.2 If the attachment zip blob is present and deleted with success, return a 204 (No Content) and log the operation using a **Log Information**

1.2.3 If the attachment zip blob is **NOT** present, return a 204 (No Content) and log the operation using a Log Warning.

There is much controversy around this scenario however, to make things simpler for the client, we are opting for the success scenario as the end state is the desired state, the attachment is not present.

1.2.4 If the **note** is **NOT** present, return a 404 (Not Found) and log the operation using a Log Warning

Note: It is possible to have a race condition here where the zip archive has not been created yet, in this case you shall still return a 204 (No Content). This condition can occur if the request is put into the queue and the DELETE is called before the Azure Function has created the zip archive blob. Additionally, it's possible to run into an error if you are trying to delete the blob while the blob is being created, this could result in a 500-server error. You do not have to implement anything to handle these scenarios as part of the 1.2 Operation Delete requirements other than returning a 500-server error as described.

Request Body

None

Responses

204 (No Content)

Returned if the attachment zip blob was deleted successfully or if the attachment zip blob does not exist when attempting to delete it.

Headers:

None beyond standard

Response Body:

None

404 (Not Found)

Returned if the note could not be found.

Headers:

None beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

500 (Internal Server Error)

Returned if the attachment zip blob is present but could not be deleted or you run into any other server side error that is not a result of bad input from the caller.

Headers:

None beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.3 API Operation – Retrieve by Id

Retrieves an attachment zip blob having the specified attachment zip file id from blob storage.

HTTP
GET `https://[appservicename].azurewebsites.net/notes/{`**`noteId`**`}/attachmentzipfiles /{`**`zipFileId`**`}`

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist
zipFileId	string	string	Not enforced	Not enforced	The zip file id of the attachment zip file This is the filename associated with the attachment that was created on behalf of the POST operation. See 1.1 API Operation – Request zip file to be created **Constraints**: Required, Must Exist.

Request Body

None

Responses

200 (OK)

Returned if the attachment zip file was successfully retrieved.

Headers:

None beyond standard

Response Body:

The attachment zip file retrieved from Block Blob storage with content type set to the **application/zip**

404 (Not Found)

Returned if the **note** or the **attachment zip** file could not be found.

Note: this is not an ideal response as the process of creating the zip blob is asynchronous, requiring the caller to poll this endpoint to retrieve the zip archive. The extra credit provides you with an opportunity to gain experience with Azure Storage Tables and implement this properly, we highly encourage you to take advantage of this learning opportunity.

Headers:

None beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.4 API Operation – Retrieve all

Retrieves a list of all attachment zip file blob Ids and associated summary information for the associated note

HTTP
GET `https://[appservicename].azurewebsites.net/notes{`**noteId**`}/attachmentzipfiles`

URI Parameters

Name	**JSON Type**	**.NET Type**	**Min Length**	**Max Length**	**Description & Notes**
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist

Request Body

None

Responses

200 (OK)

Returned regardless of whether there are any notes to return

Headers:

None beyond standard

Response Body:

The details of all the notes available are to be returned

```
[
  {
    "zipFileId": "4b7134af-6cc3-4aad-bb54-0da2e0f07928.zip",
    "contentType": "application/zip",
    "created":"{3/1/2025 10:53:09 PM +00:00}",
    "lastModified":"{3/2/2025 10:53:09 PM +00:00}",
    "length":2185980
  },
  {
    "zipFileId": "4f0442ab-3885-4512-b083-71911118a067.zip",
    "contentType": "application/zip",
    "created":"{3/3/2025 1:22:01 PM +00:00}",
    "lastModified":"{3/4/2025 12:33:21 PM +00:00}",
    "length":180152
  }
```

Name	**JSON Type**	**.NET Type**	**Description & Notes**
zipFileId	string	string	The id of the zip file in blob storage
contentType	string	string	The content type of the blob
createdDate	string	DateTimeOffset	The date/time created as recorded by the blob SDK
lastModifiedDate	string	DateTimeOffset	The last modified date as recorded by the blob SDK
length	string	long	The length of the zip file in blob storage as recorded by the blob SDK

404 (Not Found)
Returned if the note could not be found.

Headers:

None beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.5 API Operation – Note Delete Enhancement

Enhance the previous implementation of the deletion of the note and add support **for the deletion of attachment zip files**. You shall continue to delete the note and tag(s) from the Azure SQL Database **note & tag** tables and deletion of attachments from blob storage. You will add support for deleting the note's attachment zip files from blob storage. **Be sure to delete all attachment zip files and the container the attachment zip files reside in.**

HTTP
DELETE `https://[appservicename].azurewebsites.net/notes/`**`<noteId>`**

URI Parameters

Name	**JSON Type**	**.NET Type**	**Min Length**	**Max Length**	**Description & Notes**
noteId	string	string	Not enforced	Not enforced	The noteId of the note **Constraints:** Required, Must Exist.

Operation

1.5.1 Delete the note and tags from the database, if the note does not exist return a 404

1.5.2 Delete all attachments from blob storage, if there are no attachments no error is raised.

1.5.3 Delete the attachment container from blob storage, if there is no associated container no error is raised.

1.5.4 Delete all attachments zip files from blob storage, if there are no attachment zip files no error is raised.

1.5.5 Delete the attachment zip container from blob storage, if there is no associated container no error is raised.

1.5.6.1 If there is a failure to delete an existing attachment, attachment zip or a failure to delete the related containers, log an error with the details of the failure(s), but do not report a failure to the caller. Additionally ensure the note and tags are still deleted.

1.5.6.2 If there is a failure to delete the note or tags from the database then return an HTTP 500 error and log an error indicating the note or tags were not able to be deleted from the database with details of the failures(s) also **do not attempt to delete the attachments or attachment zip files if there is a failure to delete the note or tags from the database**.

A separate log entry shall be logged for each storage related failure that occurs;

For example, if there are three blobs and two blobs fail to be deleted then log to errors, one for each blob that failed to be deleted. Also, in this scenario it's likely that the container failed to be deleted so a third log entry indicating the details of the container deletion failure shall also be logged. This applies to attachments and attachment zip files.

Note: The Storage API does provide the ability to delete the container which will also delete all blobs within the container. You may use this technique, just note that you did this in your project notes.txt file.

Request Body

None

Responses

204 (No Content)

Returned if the note was deleted successfully.

Headers:

None, beyond standard

Response Body:

None

404 (Not Found)

Returned if the note could not be found.

Headers:

None, beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

500 (Not Found)

Returned if the note or tags, if tags exist, could not be deleted from the database.

Headers:

None, beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

2. Ensure seeding functionality still works as it did from assignment 3

This requirement does not require additional implementation as this requirement was already performed in assignment 3

Your API app shall automatically seed Azure Storage with the attachments associated with the following notes. These files are in the **SampleAttachments** directory of the **03-Assignment.zip** file.

Be sure that that the **NoteId** metadata entry is set to the associated note Id that is auto assigned.

You can start with an empty set of tables so that you seed the azure table and blobs in azure storage using the same conditional logic you used to seed the table in Azure Storage.

It is not necessary to handle cases where one or more of the notes or attachments are missing, your logic can seed only when there are no note entries in the Azure SQL Database table.

When submitted the table must have all the notes in the database and all attachments stored as blobs in blob storage under a container with the name of the associated note id. Additionally, you shall also have all the tags generated via the o4 mini model as described in assignment 2.

The relationship between the note in the database and attachment in blob storage is established by the container name the blob resides in. The container name has the same value as the associated note id.

There are no changes to the database schema, which consists of the table definition, from assignment 2.

You may perform the **seeding by running locally and connecting to azure**, which means you don't have to deploy the seeding attachment files to your website if you don't want to.

Table Schema Definition				
Id	Name	Details	CreatedDateUtc	ModifiedDateUtc
Auto Assigned Value	Running grocery list	Milk Eggs Oranges	Initialized with current utc time at run	Null
Auto Assigned Value	Gift supplies notes	Tape & Wrapping Paper	Initialized with current utc time at run	Null
Auto Assigned Value	Valentine's Day gift ideas	Chocolate, Diamonds, New Car	Initialized with current utc time at run	Null
Auto Assigned Value	Azure tips	portal.azure.com is a quick way to get to the portal Remember double underscore for linux and colon for windows	Initialized with current utc time at run	Null

Not included in table schema
associated attachments stored in blob storage
MilkAndEggs.png Oranges.png
WrappingPaper.png Tape.png
Chocolate.png Diamonds.png NewCar.png
AzureLogo.png AzureTipsAndTricks.pdf

3. Implement an Azure Function that listens to the attachment-zip-requests queue and creates zip files as requested by the messages in the queue

3.0. You shall build an isolated worker-based .NET 9 Isolated Azure Function

3.1. The Azure Function shall create the container for the attachment zip file if it does not exist. The **container name** is the **noteId** with **-zip** as the **suffix**

3.2. The Azure Function shall compress all the blobs in the attachment container associated with the noteId into a single zip file and store that zip file as a blob with a blob id consisting of the zipFileId provided in the message to the container created in step 3.1.

3.2.1 Log an entry, using log information, that includes the zipFileId and residing container indicating success.

3.2.2. If the zip file failed to be created, then log an error that includes the zipFileId and the intended container where the zipFileId would have been created in.

3.2.3. If the zip file fails to be created, ensure that the associated message resides in the poison message queue attachment-zip-requests-poison

4. Ensure your swagger UI updated in this assignment still operates as it did for assignment 3 with the inclusion of the attachmentzipfiles resource operations

a) Ensure the swagger UI is presented when the user navigates to your applications base URL.
   a. When running in Azure the base url is:
      `https://[appservicename].azurewebsites.net/index.html`
      `Where [appservicename] is your app services name`
   b. When running locally from Visual Studio the base url is:
      `https://localhost:[portnumber]/index.html`
      `Where [portnumber] is the port number visual studio assigned to you .NET Core Web API app`
   c. `Do not require or redirect to the /swagger sub-resource to display the UI`

b) Ensure that all REST operations are present and can be exercised from the swagger user interface.

c) Ensure that all REST operations have a description.

d) Ensure that all input and output payload fields have a description.
   a. Exception the "**`ProblemDetails`**" payload is created automatically by ASP.NET Core you don't have to add documentation to it.

e) Ensure that all Http Status codes your application can return for each REST operation are present and described with the corresponding http status code and text. 200 Success, 201 Created, 204 Success, 400 Bad Request, 404 Not Found and so forth.

f) Ensure that the response is of type application/JSON

5. Create an Azure App Service for the REST API

Create a new **Azure App Service** on your **existing Azure Linux Azure App Service Plan which is configured to the basic tier** and deploy your note keeper Web API implementation your new Azure App Service. Be sure to provide the URL to your REST interface endpoint in your Project Notes.TXT file. Remember to follow recommended abbreviations for Azure resource types. See

Implementation notes:

6. Create an Azure Function App for the Azure Function

Create a new **Azure Function App using Linux and a new Consumption Plan.** Deploy your Azure Function your new Azure Function App. **Be sure to provide the name of your Azure Function App in your Project Notes.TXT file**. Remember to follow recommended abbreviations for Azure resource types. See

Implementation notes:

Implementation notes:

1. You may not use / employ.
    a. Any samples from the Azure Dashboard to create this solution.
2. You can use the Visual Studio ASP.NET Core Web API Application Template
    a. You must delete all non-relevant items as described in the lecture.
3. All exceptions within your code must be handled, the service must not return any exception information to the caller raised inside of your code. Exceptions raised before your code is reached don't need to be captured in this assignment.
4. Use built in exceptions to handle error exceptional cases wherever possible.
5. When creating azure resources, including the resource group your Azure resources will reside in, be sure to use the recommended abbreviations for Azure resource types. See: Recommended abbreviations for Azure resource types - Cloud Adoption Framework | Microsoft Docs
6. Map App Service REST errors to standard HTTP Error codes as defined in the requirements.
    a. If a situation is not specified in the requirements pick a standard HTTP status code and document under what conditions, it is returned in your projectnotes.txt file.

Other Notes:

- Up to 15 Points shall be deducted for lack of comments in code.
- All methods, classes etc. must be documented.
    a. C# with XML comments.
- Your project must build with no errors or warnings.

Extra Credit:

To qualify for extra credit, you must first implement all core requirements of the assignment. Only the extra credit options specified are eligible for additional points, aside from points awarded for early submission. To earn extra credit, you must completely implement an extra credit option.

Students registered for graduate credit are required to complete one extra credit option. The selected option will be included in your base grade calculation but will not raise your overall score above 100%. Failure to complete an extra credit option will result in a deduction equivalent to the points assigned to the lowest-value extra credit option. For specific point values and details, refer to the grading rubric for this assignment.

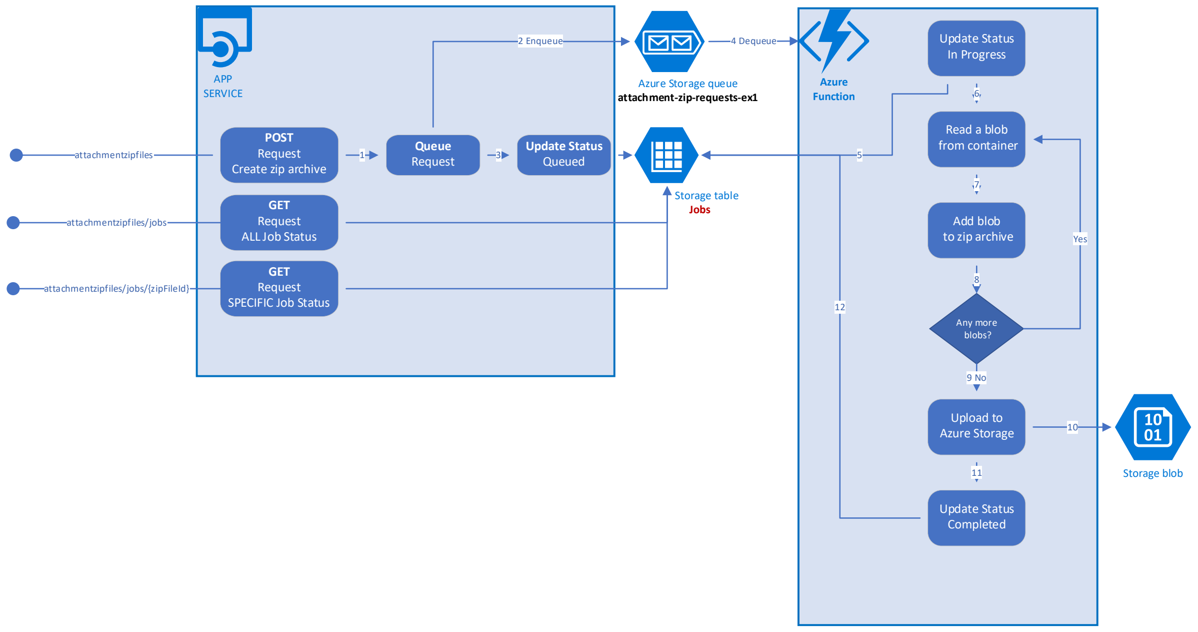
Use the same database for the core requirements of this assignment and the extra credit options. **The core requirements and each extra credit option will use unique queues so they can be run simultaneously.**

Extra Credit 1: Add support for a job status tracking table.

The solution shall contain two projects:

1. **HW4NoteKeeperEx1** shall contain the REST interface web app implementation.
    a. **Deploy to a new azure app service with a suffix of ex1.**
2. **HW4AzureFunctions Ex1** shall contain the azure function implementation.
    a. **Deploy to a new azure function with a suffix of ex1.**

The implementation shall use a queue named **attachment-zip-requests-ex1**



1. Utilize an Azure Storage Table to keep track of the status of the requested zip file creation jobs.
1.1 The table's **row key** shall be the **attachment zip file id**.
1.2 The **partition key** of table shall be the **note id**.
2. The **table** shall be **named** **Jobs** and shall contain the following columns
2.1. The inserted/last modified timestamp, called **Timestamp**
2.2. The status, named **Status**, with the following status values base on the operation
2.3.1. **Queued**

This status is set when the message is enqueued.

2.3.2. **InProgress**

This status is set when the message is retrieved by the azure function or webjob (see extra credit option 2).

2.3.3. **Completed**

This status is set when the attachment zip file has been created with success

2.3.4. **Failed**

This status is set when the attachment zip file or the attachment zip file's container fails to be created

2.4. The status details in a column called **StatusDetails** with the following content
2.4.1. If the status is **Queued** the **StatusDetails** shall contain "Queued: Zip File Id: <The zipFileId> NoteId: <the note id>"
2.4.2. If the status is **InProgress** the **StatusDetails** shall contain "In Progress: Zip File Id: <The zipFileId> NoteId: <the note id>"
2.4.3. If the status is **Completed** the **StatusDetails** shall contain "Completed: zipFileId: <zipFileId of the zip file created> containerId: <The residing container Id of where the zipFileId blob was created>"
2.4.4. If the status is failed, the **StatusDetails** shall contain the "Failed: Zip File Id: <The zipFileId> NoteId: <the note id>"
2.5. Create a new sub-resource that returns the status of the associated zipFileId compression job from the Azure Storage Job's table. See the following details:

HTTP
GET https://[appservicename].azurewebsites.net/notes/{**noteId**}/attachmentzipfiles/jobs/{**zipFileId**}

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist
zipFileId	string	string	Not enforced	Not enforced	The zip file id of the attachment zip file This is the filename associated with the attachment that was created on behalf of the POST operation. See 1.1 API Operation – Request zip file to be created **Constraints:** Required, Must Exist.

Request Body

None

Responses

200 (OK)

Returned if the attachment zip job status was successfully retrieved.

Headers:

None beyond standard

Response Body:

```
{
  "ZipFileId": "4b7134af-6cc3-4aad-bb54-0da2e0f07928.zip",
  "TimeStamp": " 2022-02-23T13:53:20.7319029+00:00",
  "Status":"InProgress",
  "StatusDetails ":"In Progress: Zip File Id: 4b7134af-6cc3-4aad-bb54-
0da2e0f07928.zip NoteId: 22f9d506-7b13-4763-82fc-1e2d154ed7fb"
}
```

Name	**JSON Type**	**.NET Type**	**Description & Notes**
ZipFileId	string	string	The id of the zip file in blob storage
TimeStamp	string	DateTimeOffset	The insertion time in UTC
Status	string	string	The status of the job
StatusDetails	string	string	A more detailed representation of the status

404 (Not Found)

Returned if the note or the Job row was no found.

Headers:

None beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

3. Implement a GET all to return the status of all Jobs for a note. See the following details:

HTTP
GET https://[appservicename].azurewebsites.net/notes/{**noteId**}/attachmentzipfiles/jobs

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist

Request Body

None

Responses

200 (OK)

Returned if the attachment zip job status(es) are successfully retrieved.

Headers:

None beyond standard

Response Body:

```
[
    {
      "ZipFileId": "4b7134af-6cc3-4aad-bb54-0da2e0f07928.zip",
      "TimeStamp": " 2023-03-23T13:53:20.7319029+00:00",
      "Status": "InProgress",
      "StatusDetails ": " In Progress: Zip File Id: 4b7134af-6cc3-4aad-bb54-
0da2e0f07928.zip NoteId: 22f9d506-7b13-4763-82fc-1e2d154ed7fb "
     },
     {
      "ZipFileId": "7b2134ab-8cc3-3aad-ac21-0da2e0f07937.zip",
      "TimeStamp": " 2023-02-11T13:53:20.8119031+00:00",
      "Status": "Completed",
      "StatusDetails ": " Completed: zipFileId: 7b2134ab-8cc3-3aad-ac21-
0da2e0f07937.zip containerId: e7d1c4f8-29a4-4a4e-8c3b-9e9f206d1556-zip"
     }
]
```

Name	JSON Type	.NET Type	Description & Notes
ZipFileId	string	string	The id of the zip file in blob storage
TimeStamp	string	DateTimeOffset	The insertion time in UTC
Status	string	string	The status of the job
StatusDetails	string	string	A more detailed representation of the status

404 (Not Found)

Returned if the note was no found.

Headers:

None beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

4. API Operation – Note Delete Enhancement

Enhance the previous implementation of the deletion of the note and add support **for the deletion of all job entries associated with the note**. See the 1.5 API Operation – Note Delete Enhancement in this assignment for the core requirements.

HTTP
DELETE `https://[appservicename].azurewebsites.net/notes/`**`<noteId>`**

URI Parameters

Name	**JSON Type**	**.NET Type**	**Min Length**	**Max Length**	**Description & Notes**
noteId	string	string	Not enforced	Not enforced	The noteId of the note **Constraints:** Required, Must Exist.

Operation

4.1 In addition to the core requirements for the 1.5 API Operation – Note Delete Enhancement, delete all rows in the Azure Storage Job table associated with the NoteId being deleted.

4.1.2 If there are no rows corresponding to the NoteId in the Azure Storage Job table, log an information message indicating this, do not report an error to the caller.

4.1.3. If there is an error deleting from the Azure Storage Job table, log an error indicating this issue but do not report an error to the caller and continue with all other delete operations as specified in 1.5 API Operation – Note Delete Enhancement.

4.1.4. If a note attachment compression is in progress while a delete operation is requested be sure to handle this scenario and fail the delete request with a 409 Conflict.

Request Body

None

Responses

204 (No Content)

Returned if the note was deleted successfully.

Headers:

None, beyond standard

Response Body:

None

404 (Not Found)

Returned if the note could not be found.

409 (Conflict)

Returned if the note could not be deleted because there is a compression operation ***in progress***.

Headers:
None, beyond standard

Response Body:
No custom response body is required, you can use the default response body returned by ASP.NET Core.

500 (Not Found)
Returned if the note or tags, if tags exist, could not be deleted from the database.

Headers:
None, beyond standard

Response Body:
No custom response body is required, you can use the default response body returned by ASP.NET Core.

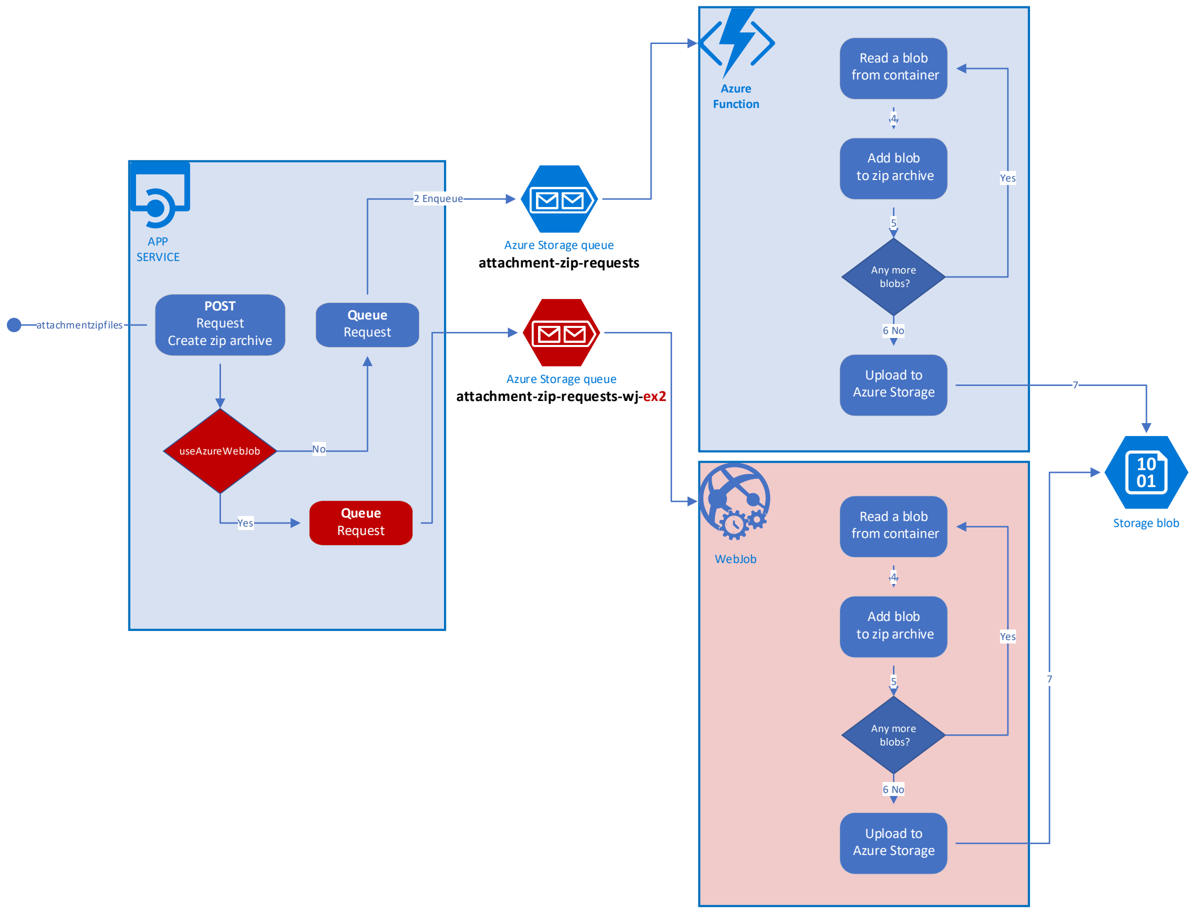
Extra Credit 2: Implement the Azure Function functionality as a WebJob.

The solution shall contain two projects:

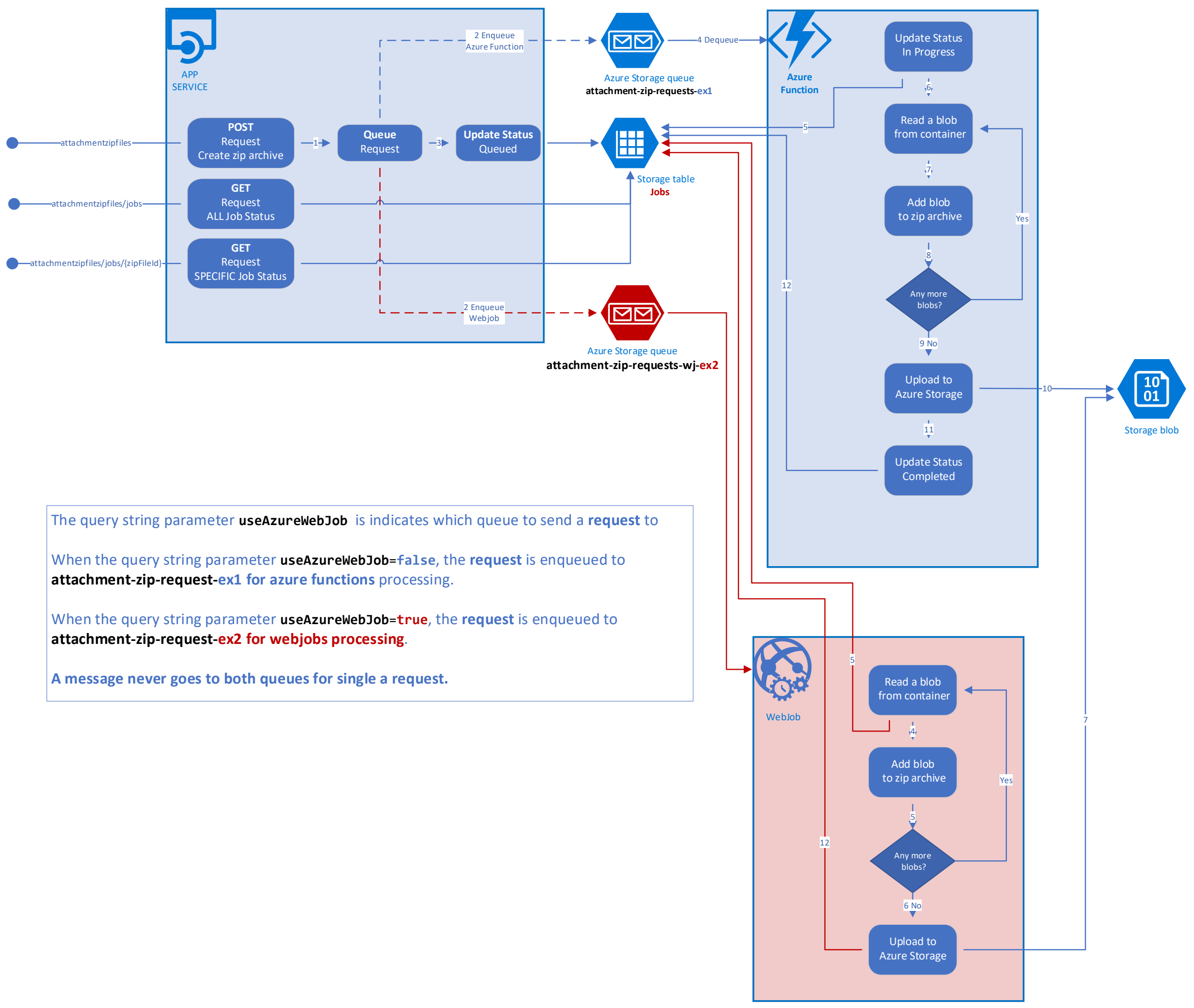
1. **HW4NoteKeeperEx2** shall contain the REST interface web app implementation.
    a. **Deploy to a new azure app service with a suffix of ex2.**
2. **HW4AzureFunctions Ex2** shall contain the azure function implementation.
    a. **Deploy to a new azure function with a suffix of ex2.**

The implementation shall use a queue named **attachment-zip-requests-wj-ex2** for web job requests. If azure function requests are made send them to the core requirements queue **attachment-zip-requests** if you did not implement extra credit 1**. If you implemented extra credit 1 then send the azure function requests to attachment-zip-requests-wj-ex1.**

Architecture if you did not implement extra credit 1



Architecture if you DID implement extra credit 1



The query string parameter **`useAzureWebJob`** is indicates which queue to send a **request** to

When the query string parameter **`useAzureWebJob=false`**, the **request** is enqueued to **attachment-zip-request-ex1 for azure functions** processing.

When the query string parameter **`useAzureWebJob=true`**, the **request** is enqueued to **attachment-zip-request-ex2 for webjobs processing**.

A message never goes to both queues for single a request.

1. Create a WebJob to perform the same operations as the Azure Function, see 3. Implement an Azure Function that **listens** to the attachment-zip-requests queue and creates zip files as requested by the messages in the queue
2. Add a query string parameter called **useAzureWebJob** as an optional parameter to the POST operation see 1.1 API Operation – Request zip file to be created

```
HTTP
POST https://[appservicename].azurewebsites.net/notes/{noteId}/attachmentzipfiles
?useAzureWebJob=<true/false>
```

URI Parameters

Name	Data Type	.NET Type	Min Length	Max Length	Description
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist
useAzureWebJob	Bool	bool	N/A	N/A	**Default** false

2.1 If the **useAzureWebJob** is set to true then enqueue the message to a queue named **attachment-zip-requests-wj-ex2**. The Azure **WebJob** will listen to the **attachment-zip-requests-wj-ex2** for messages containing the instructions necessary to compress instead of the **attachment-zip-requests-ex2** queue that the Azure Function listens to. The format of the message shall be the same as the format used by the Azure Function.

3 Create an **Azure App Service** on a new **Windows App Service Plan in the Basic Tier** and deploy your completed azure **WebJob** to it.

4. If you implemented extra credit option 1 you will also need to have the web job update the azure storage job table meeting the requirements of extra credit option 1. ***You shall use the same table as you did for the extra credit 1 option.***

Extra Credit Option 3: Use managed identities for authentication to Azure Storage Queues in your Azure Function and Azure App Service

1. Update your existing Azure Function to utilize managed identities to access your Azure Storage Queue
1.2. Add a **user assigned** managed identity to your Azure Function. You can use the same user assigned managed identity from assignment 2 or 3 if you implemented the associated extra credit.
1.3. Grant your user assigned managed identity the **Storage Queue Data Contributor** role for access to the **storage queues**
1.4. Ensure that the **AzureWebJobsStorage** is not set or set to an empty string.
1.5 Set the environment variable **AzureWebJobsStorage__queueServiceUri** to the Uri of your storage account's queue end point on your Azure Function deployment.
1.6 Update your existing Azure App Service to use the same user assigned managed identity to access the azure storage queue. Hint look at the **ManagedIdentityQueueWebApi** example as a learning resource.

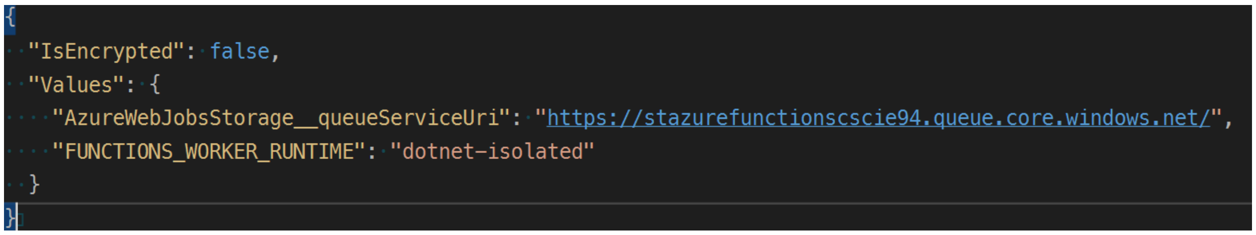
Example here the storage account name is **stazurefunctionscscie94** the setting for the environment variable would be "https://**stazurefunctionscscie94**.queue.core.windows.net/".

Replace **stazurefunctionscscie94** with your storage account name.

2. Ensure your development environment also uses managed identities to access the azure storage queue
2.1. Assign the user for the Microsoft Account you use to log into the Azure portal the **Storage Queue Data Contributor role on the Azure Storage Account**
2.3. **Repeat step 2.1 for all the TAs and Instructor's Azure Portal Accounts**
2.4. Ensure that the **AzureWebJobsStorage** is not set or set to an empty string in your **local.settings.json** file
2.5 Set the **local.settings.json** file to have an entry for **AzureWebJobsStorage__queueServiceUri** set to the Uri of your storage account's queue end point on your Azure Function deployment.
2.6 Update your existing Azure App Service to use the same user assigned managed identity to access the azure storage queue. Hint look at the **ManagedIdentityQueueWebApi** example as a learning resource.

Example here the storage account name is **stazurefunctionscscie94**

The **local.settings.json** file will look like this:



```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage__queueServiceUri": "https://stazurefunctionscscie94.queue.core.windows.net/",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet-isolated"
  }
}
```

Replace **stazurefunctionscscie94** with your storage account name.