Document Manager Solution

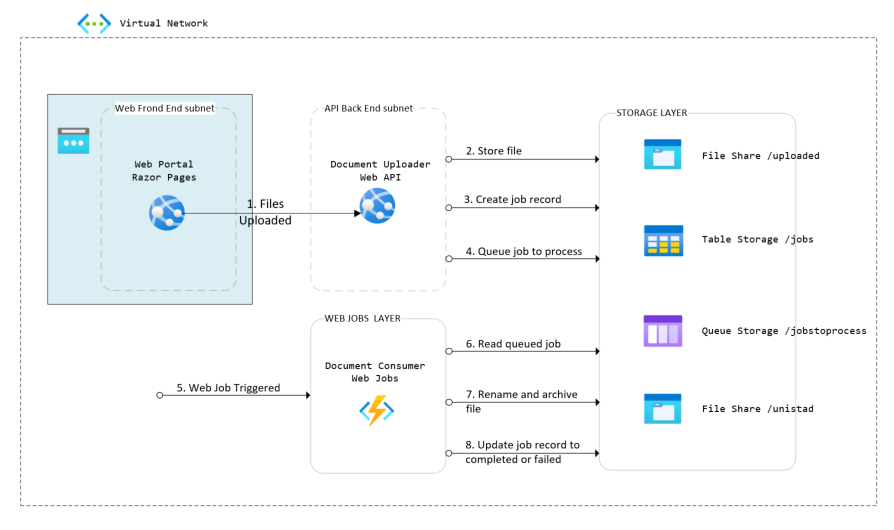
# Overview

This Web Application will be responsible to organize automatically documents in Azure Storage File folders and apply the naming and convention defined for the files.

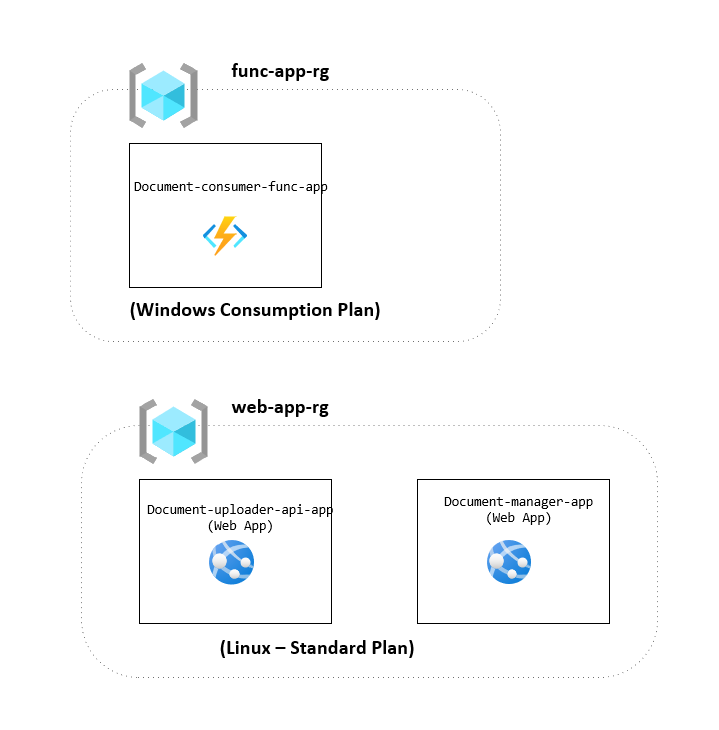
The application contains the following components:

* Document Uploader: Web API which receives the document uploaded to be processed.
* Document Consumer: Azure Function which stores the document uploaded in the correct Storage File Folder.
* Web application: Web page with the interface to upload the files.

# Architecture



# Hosting



The application requires one app service plan (Linux) with the following resources:

* Two Web Apps
* One Function App
* One Storage account with File Share, Table Storage, and Queue Storage.

# Document Uploader – Web API Specification

## API Upload Document

Creates the file uploaded in the File Storage. There is a default folder where the files uploaded are stored (see topic Storage Entities -> File Storage for details about the folders). Create a record in the table storage to track the status of the job. Enqueues a message in the Storage Queue indicating the file was uploaded, so the job needs to be processed by the Azure Function.

|  |
| --- |
| **HTTP POST** |
| https://documentuploader.azurewebsites.net/api/v1/uploadedimage?fileName=SC-I60-CAB-ORD-DBF-IT-00102.pdf |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **URI Parameters** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| fileName | True | String | 200 | Valid file name | File name |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Request Body** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| fileData | True | File | n/a | Not null | The file upload data  .NET Type: **IFormFile** |

Additional Requirements:

* + 1. The content type shall be automatically set using the content type provided by IFormFile fileData parameter.
    2. The id assigned to the Job shall be generated automatically by your implementation. You shall use a Guid converted to a string for the Job’s id. This id will be added in the Table storage RowKey column and also in the Queued message.
    3. Each file uploaded will be considered a Job.
    4. See Topic Storage Entities to get all details about the File Storage, Table Storage, and Queue Storage.

|  |
| --- |
| **Response 201 (Created)** |
| Returned if the file was stored in the uploaded folder successfully. Table job row was created, and a message was added to the Queue.  **Headers**  **Location: contains the storage address to the file uploaded, example** [https://**[service**](https://[service) **name]**.azurewebsites.net/api/v1/[uploaded folder name]/[file name uploaded]  See Storage topic for further details about the folders name used for the uploaded files.  **Body**  Empty (no content) |

|  |
| --- |
| **Response 400 (Bad Request)** |
| Returned if the request has no file data or already exist a file in the uploaded folder with the same name.  **Headers**  None, beyond standard  **Body**  Return an error response using the ErrorResponse JSON format, see topic about the Error Response topic for further details. |

## API Get Job Queue Message - Operation returns a job message stored in the table storage, based on the GUID of the message.

Returns a Job stored in the Table storage, the id is the GUID of the job.

|  |
| --- |
| **HTTP GET** |
| https://documentuploader.azurewebsites.net/api/v1/job/{id} |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **URI Parameters** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| id | True | String | 200 |  | GUID of the job. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Request Body** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| Not used. | Not used. | Not used. | Not used. | Not used. | Not used. |

|  |
| --- |
| **Response 200 (Ok)** |
| Return in JSON format the job record.  **Headers**  **Location: not provided**  **Body**  Job record in JSON format.  {  "jobId":"11bd738c-005c-474d-80e0-d96180235c5a",  "partition":<constant value defined in the application code>,  "status":3,  "statusDescription":"Conversion completed",  "fileSource":"SC-I60-CAB-ORD-SPE-IT-00158\_sign.pdf",  "fileResult":"c:\\azurite\\unistad\\07. KH\\Package 1\\Base\\MS-2\\KH-ACS-BRS-SC-I60-CAB-ORD-SPE-IT-00158.pdf",  "user":"demo@rjnfrazaooutlook.onmicrosoft.com"  } |
| Comments  Partition: In the Configuration Library the constant TABLE\_PARTITION\_KEY holds the value for the partition. |

|  |
| --- |
| **Response 400 (Bad Request)** |
| Returned in case of any internal error. As the application makes use of a middleware to response for any exception. We don’t differentiate internal errors and bad requests.  **Headers**  None, beyond standard  **Body**  Return an error response using the ErrorResponse JSON format, see topic about the Error Response topic for further details. |

|  |
| --- |
| **Response 404 (Not found)** |
| Returned if the requested job record wasn’t found.  **Headers**  None beyond standard.  **Body**  None |

## API Get All Jobs - Operation returns the jobs stored in the table storage. In case a user name is provided, filter the jobs per user.

Returns all the Jobs stored in the Table storage, if the user name is provided returns the jobs uploaded by the user name provided.

|  |
| --- |
| **HTTP GET** |
| https://documentuploader.azurewebsites.net/api/v1/jobs/{user name} |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **URI Parameters** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| User name | optional | String | 200 |  | User name |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Request Body** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| Not used. | Not used. | Not used. | Not used. | Not used. | Not used. |

|  |
| --- |
| **Response 200 (Ok)** |
| Return in JSON format the list of job records.  **Headers**  **Location: not provided**  **Body**  Job records in JSON format uploaded by user ":"demo@rjnfrazaooutlook.onmicrosoft.com"  [  {  "jobId":"11bd738c-005c-474d-80e0-d96180235c5a",  "partition":<constant value defined in the application code>,  "status":3,  "statusDescription":"Conversion completed",  "fileSource":"SC-I60-CAB-ORD-SPE-IT-00158\_sign.pdf",  "fileResult":"c:\\azurite\\unistad\\07. KH\\Package 1\\Base\\MS-2\\KH-ACS-BRS-SC-I60-CAB-ORD-SPE-IT-00158.pdf",  "user":"demo@rjnfrazaooutlook.onmicrosoft.com"  },  {  "jobId":"928ba9f6-1374-46f8-a013-3d951f30d06c",  "partition":<constant value defined in the application code>,  "status":4,  "statusDescription":"Internal Error. Not able to move the file c:\azurite\\_jobs\_uploaded\SC-I60-CAB-ORD-SPE-IT-00158\_sign.pdf successfuly converted to the destination folder c:\azurite\root\07. KH\Package 1\Base\MS-2\KH-ACS-BRS-SC-I60-CAB-ORD-SPE-IT-00158.pdf",  "fileSource":"SC-I60-CAB-ORD-SPE-IT-00158\_sign.pdf",  "fileResult":"c:\azurite\\_jobs\_failed\SC-I60-CAB-ORD-SPE-IT-00158\_sign-0d06c.pdf",  "user":"demo@rjnfrazaooutlook.onmicrosoft.com"  }  ] |
| Comments  Partition: In the Configuration Library the constant TABLE\_PARTITION\_KEY holds the value for the partition. |

|  |
| --- |
| **Response 400 (Bad Request)** |
| Returned in case of any internal error. As the application makes use of a middleware to response for any exception. We don’t differentiate internal errors and bad requests.  **Headers**  None, beyond standard  **Body**  Return an error response using the ErrorResponse JSON format, see topic about the Error Response topic for further details. |

|  |
| --- |
| **Response 404 (Not found)** |
| Returned if the requested job records weren’t found to the user name provided.  **Headers**  None beyond standard.  **Body**  None |

# Document Consumer – Azure Function

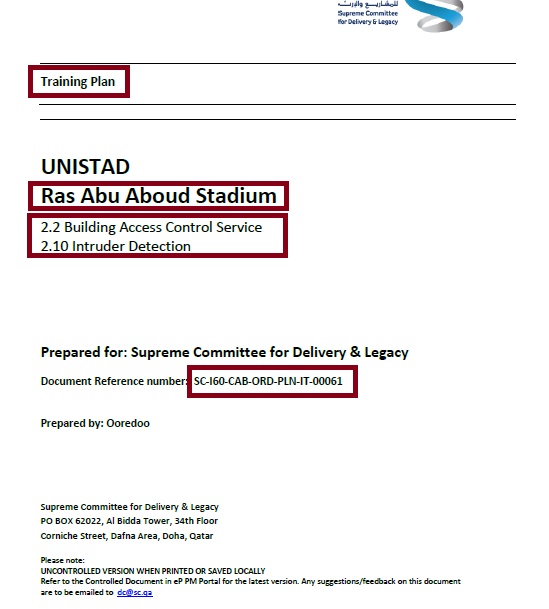
This Azure Function is triggered when a new message is queued. The function will be responsible to scan the document, identify all elements which produces the correct file name and the correct folder to be stored, if all elements are found the file is moved to the correct folder using the defined files name.

The file name pattern to be worked out is [Stadium Code]-[Services Code]-[Document Type]-[EDRMS reference number]-[Revision number]. All details regarding the codes used and also the folder structure can be found in this file.

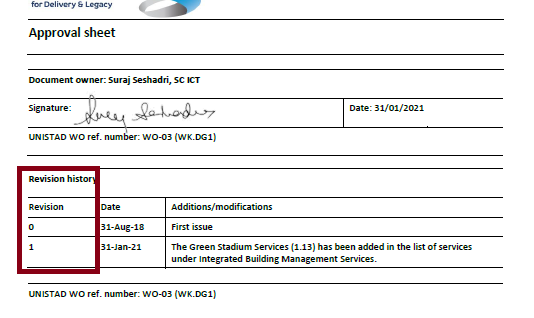


In order to work out the file name and folder name, it’s required that the following configuration map to be added into the Application Settings configuration, please attention to the topic 4. Configurations - Document Consumer (Azure Function). You will find there the file with all configuration needed.

The following information is captured from the first page of the pdf file Stadium name, Service name, Document Type, and EDRMS reference number.



The revision number is captured on the second page. The patterns to be considered are the numbers after “Revision” word where a number is surrounded by spaces or using the following pattern “0.[number]” (0.1 0.2 etc). When the document has only the revision 0, the version is not added to the file name. The application consider the biggest number.



Once the operation is completed, the job record (storage table) is updated from 2 – Job running to status 3 – process completed, or 4 – process failed. If the process failed, the file is moved from the \_jobs\_uploaded folder to \_jobs\_failed folder, part of the GUID of the job process is added to the file before move it. All details are stored in the Job Table. See API Upload Document above and 3. Storage Entities (3.1) for more details about the Job Record.

The function assures to create the subfolders needed, when archiving under the “Root” folder, the files converted successfully.

# Storage Entities

## Table Storage Specification

Table storage name will be **unistadjobs**, If the queue **unistadjobs** does not exist, then it shall be created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column | Data Type | Length | Description | Value |
| PartitionKey | String | 20 | Project name | UNISTAD |
| RowKey | String | 36 | The id of the job. | The id will be a Guid. |
| Status | Number | n/a | A number indicating the status of the job  1 = File uploaded  2 = Job is running  3 = Job completed with success  4 = Job failed | 1 |
| StatusDescription | String | 512 | If no error occurred this value contains the text associated with the status defined above.  Ex: File uploaded.  If an error occurred a human readable description of the problem running converting the image. Always prefix this with "Job failed:" followed by a more descriptive message. This can be the message from an exception that occurred but NOT THE STACK TRACE.  Ex: Job failed: The uploaded image could not be converted. | File uploaded |
| FileSource | String | 255 | Original name of the file. | SC-I60-CAB-ORD-DBF-IT-00102.pdf |
| FileResult | String | 255 | The path where the file was stored and the new file name.  In case of any failure, the file is moved to the failed folder. | /HB/Package 1/Base/MS – 2/HB-ACS-HLFD- SC-I60-CAB-ORD-DBF-IT-00102.pdf  Or  /\_Failed/ SC-I60-CAB-ORD-DBF-IT-00102.pdf |

## Queue Storage Specification

Azure Storage Queue shall be called: **unistadprocessqueue,** If the queue **unistadprocessqueue** does not exist, then it shall be created.

The message will have the following JSON format:

|  |
| --- |
| {  “PartitionKEy” : “UNISTAD”  “RowId” : “[Guid]”  “FileName” : “[Original file name]”  } |

RowId is the link with the Jobs table.

FileName is the name of the file stored in the uploaded folder.

When the process is completed, the message is removed from the queue.

## File Storage Specification

Azure Storage File will have the following folders:

* /jobs/uploaded : Folder where the files uploaded are stored temporally.
* /jobs/failed : Folder where the files, where the consuming process failed are stored.
* /unistad : Root folder where the files consumed will be stored.

Additional Requirements:

* + 1. The unistad folder will be organized using the following sub folders: /unistad/[Stadium]/[Package + number]/[\*Release name]/[Milestone number].
       - The “\*” means optional, as for Package 2 there are no releases at the moment, so this is omitted.
       - Packages names are: Package 1 or Package 2.
       - Release names are Base, Release 1, Release 2.
       - Milestones numbers are MS – 1, MS – 2, MS – 3, …, until MS – 6.
    2. When file is saved in the **target folder**, in case the file already exists the job will be considered failed, the file will be stored in the filed folder.
    3. When file is saved in the **failed folder**, the last 4 characters of the job are added as suffix of the file, as the same file name can be uploaded several times.

# Error Response JSON

This is the standard error response that shall be returned.

|  |
| --- |
| {  "errorNumber":<error number>,  "parameterName":"<name of parameter that caused the error>",  "parameterValue":"<value of parameter that caused the error>",  "errorDescription":"<Description of the error intended developer consumption>"  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | Required | Data Type | Length | Description |
| errorNumber | True | number | n/a | Numeric error that represents the issue  .NET Type: int |
| parameterName | False | String | 1024 | The name of the parameter that has the issue.  If the error is not tied to a specific parameter, then this value can be null  .NET Type: string |
| parameterValue | False | string | 2048 | The value of the parameter caused the error  If the error is not tied to a specific parameter, then this value can be null  .NET Type: string |
| errorDescription | True | String | 1024 | A description of the error, not localized, intended for developer consumption  .NET Type: string |

# API Error Response Codes

Below are the common codes and error descriptions to be used by the APIs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **errorNumber** | parameterName | parameterValue | errorDescription | Notes |
| 1 | Required | Required | The document was already uploaded. | This error is raised when file is uploaded. |
| 2 | Required | Required | The parameter is required. |  |
| 3 | Required | Required | The entity could not be found. |  |
| 4 | Required | Null | The parameter cannot be null. |  |

Below are the common codes and error descriptions to be used when consuming the document uploaded.

|  |  |  |
| --- | --- | --- |
| **errorNumber** | errorDescription | Notes |
| 21 | The stadium name wasn’t found. | Raised when searching the **name of the stadium** in the pdf. |
| 22 | The service name wasn’t found. | Raised when searching the **name of the service** in the pdf. |
| 23 | The document type wasn’t found. | Raised when searching the **document type** in the pdf. Examples: Business Requirement, High Level Functional Document, etc. |
| 24 | The document reference number wasn’t found. | Raised when searching the **document reference number** in the pdf. Example : SC-I60-CAB-ORD-DBF-IT-00102 |
| 25 | The version wasn’t found. |  |
|  |  |  |
| 25 | File already exists. | When the file result already exist in the folder. |

# Log Error Codes

This is the list of errors listed in the log. The codes follow this pattern :

100-149 : Errors at the UnistadDocumentLibrary Project.

200-249 : Errors at the DocumentConsumer Project.

250-299 : Errors at the StorageLibrary Project.

| **Error Number** | **Location** | **Raised When** | **Cause** |
| --- | --- | --- | --- |
| 114 | Unistad Document Library :  UnistadDocument Class | Not found in the pdf the following information: Stadium, Service, or Document Type. The application is not able to work out the document name. | Document is not an UNISTAD document managed by the application. Ex:PM & DM Plan.  Missing a configuration related to the mapping between value and code in the host.json. |
| 115 | Unistad Document Library :  UnistadDocument Class | Not able to find the mapping between the code (stadium, service, document type) and target folder or sub-folder to store the file.  When 114 is raised 115 is raised also as the code wasn’t found | When error 114 is raised, the error 115 is raised also.  Missing a configuration related to the mapping between code and folder location in the host.json file. |
| 200 | DocumentConsumer : DocumentConsumer Class | Unknown error raised when UnistadConsumer function was processing the file provided in the triggered queued message. | Check it out the log. More details of this errors are recorded. |
| 201 | DocumentConsumer : DocumentConsumer Class | When archiving the file at the target folder, a file with the same name already exists in the target folder. In this case the process fails, the file being processed is stored in the Failed folder, using the original name + 5 last GUID letters. | Same file was already uploaded before.  Application wasn’t able to define correctly the name of the file. Possible cause can be the version or service name. |
| 251 | StorageLibrary : Repositories. FileSystem | The file to be deleted or moved can’t be found.  I don’t believe this error will happen, but when removing the file from the Source Uploaded Folder after being process, the file wasn’t found. | The file was removed by another process.  The folder or file was renamed while file was being processed.  Please analyze carefully as this error wouldn’t happen. |
| 253 | StorageLibrary : Repositories. FileSystem | Unknown error moving the file. | Check error log for further details. |

# Class Unistad Document

Class responsible to implement all operations needed to rename the original uploaded file to the correct naming and convention, it also has the responsibilities to implement the methods required to store the file in the correct folder. This class will make use of the package which implements all operation into pdf file.

Constructor

* UnistadDocument (FileSourceName, FileShareClient)

Methods:

* Private GetStadiumCode
* Private GetServiceCode
* Private GetDocumentTypeCode
* Private GetDocumentReference
* Private GetDocumentName
* Private GetDocumentFolder
* Public StoreDocument

# Application Deployment in production

## Azure Resources

1. Create the storage account : “sascdocument”.
2. Create the storage : “unistad” (Private networking access only to restrict the access to the network).
3. Configure the networking (storage -> networking)
   1. Configure the firewall by allowing access via the Firewall from the IP used to configure the environment.
   2. Exceptions must be selected to allow Azure Services on the trusted services list.
4. Create the Azure File Share: “unistad”.
5. Create the Azure Queue: “jobstoprocess”.
6. Create the Azure Table: “jobs”.
7. Create App Service Plan using Linux.
8. Application Registration in the Azure AD. Follow the configuration explained in this link -> <https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-v2-aspnet-core-webapp>

## Application Configurations

**UNISTAD Document Manager (Web App)**

|  |  |  |
| --- | --- | --- |
| Parameter | Location | Description |
| ApiConsumerUrl | appsettings. {environment}. json | Consumer API URL end point.  Example development environment: https://localhost:44326/api/v1/**upload** |

**Document Uploader Project (Web API)**

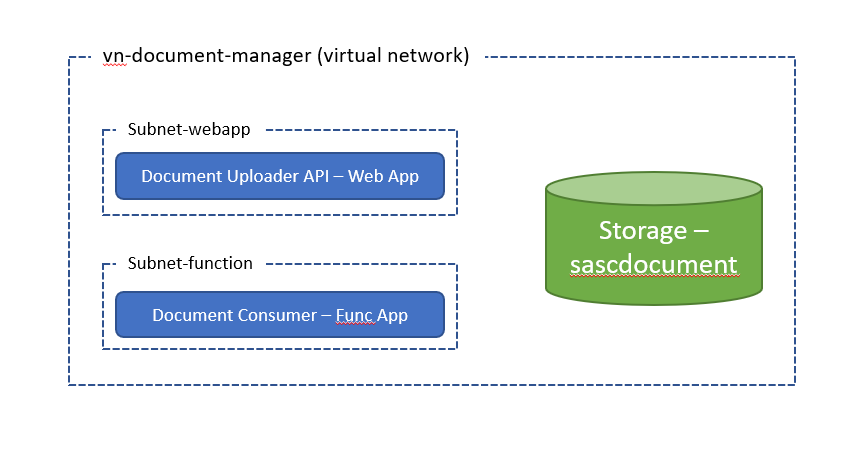
|  |  |  |
| --- | --- | --- |
| Parameter | Location | Description |
| ApplicationSettings : UseDevelopmentStorage | appsettings. {environment}. json | True means the solution will use the local file system to store the files uploaded and converted. This is used for local test when the Azure is not available.  False means the Azure File Share is used. |
| ApplicationSettings : DevelopmentFileSystemRoot | appsettings. {environment}. json | Local folder where the files are going to be stored.  Example: “c:\\azurite\\”, this is used when needed to test the solution locally. |
| ApplicationSettings : DefaultVisibilityInSeconds | appsettings. {environment}. json | Visibility time for the message in the Queue. |
| ApplicationSettings : DefaultTimeToLiveInSeconds | appsettings. {environment}. json | Duration the message will be live in the Queue. In this application should be a long period. |
| ConnectionStrings : DefaultStorageConnection | appsettings. {environment}. json | Connection string to access the storage. When tested locally this configuration is stored in “User Secrets” (Visual Studio). |

**Document Consumer (Azure Function) -**

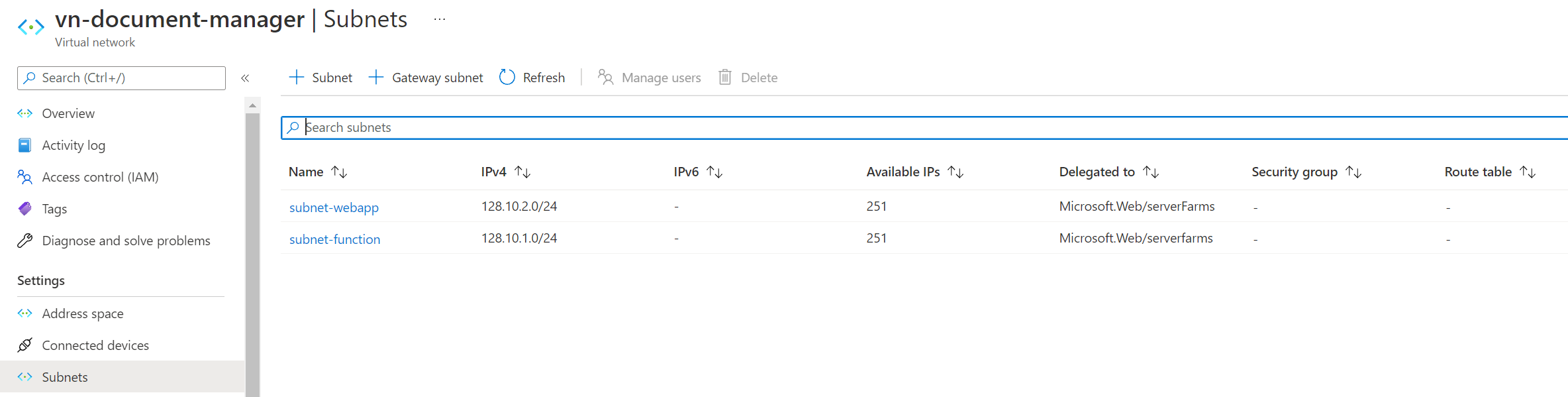
|  |  |  |
| --- | --- | --- |
| Parameter | Location | Description |
| ConnectionStrings : DefaultStorageConnection | Azure App Settings Configuration (production) and  Local.setting.json (local development) or | Connection string to access the storage. This configuration parameter can be changed. Check the Configuration Library constant “QUEUE\_CONNECTIONSTRING\_NAME” this is the constant used by the connection parameter used to trigger the function. |
| File: DictionaryMapping.settings.json | Root folder of the project | This JSON file contains all data mapping needed to work out folder destination and file name. This file will be published also in Azure Function App.  When the function runs, this file is loaded as one of the providers of the configuration object.  Environment variables such as the ones configured in Azure App Settings Portal will override the ones existing in this file. Keep in mind that the ordering of the elements must not be broken, so don’t leave gaps in the numbering. |
|  |  |  |
|  |  |  |
|  |  |  |

**Azure Virtual Networks**

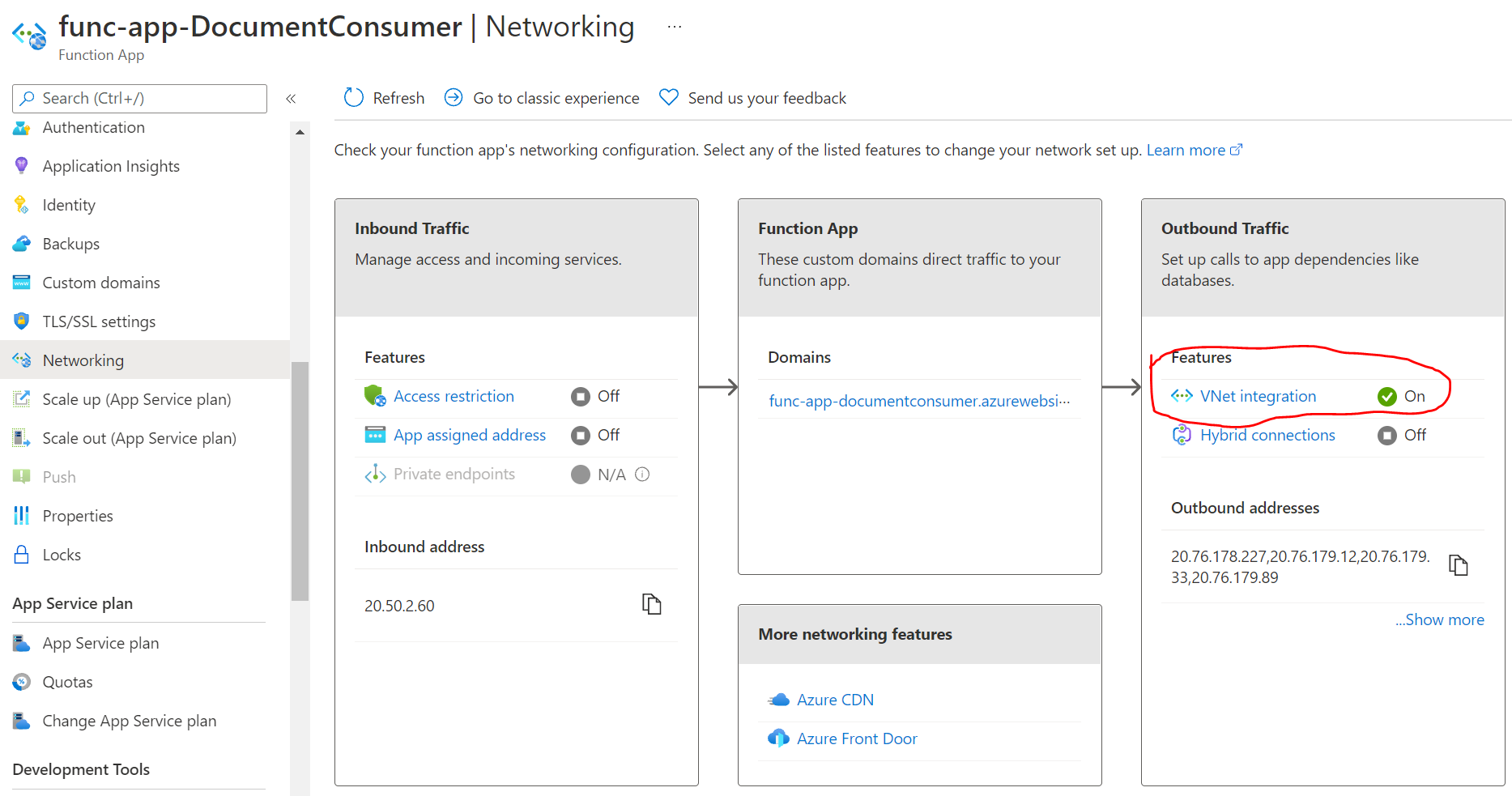
In order to allow the Document Uploader API and Document Consumer to access the storage is needed to create two virtual networks according this design.



1. Create the virtual networking and the two subnets.



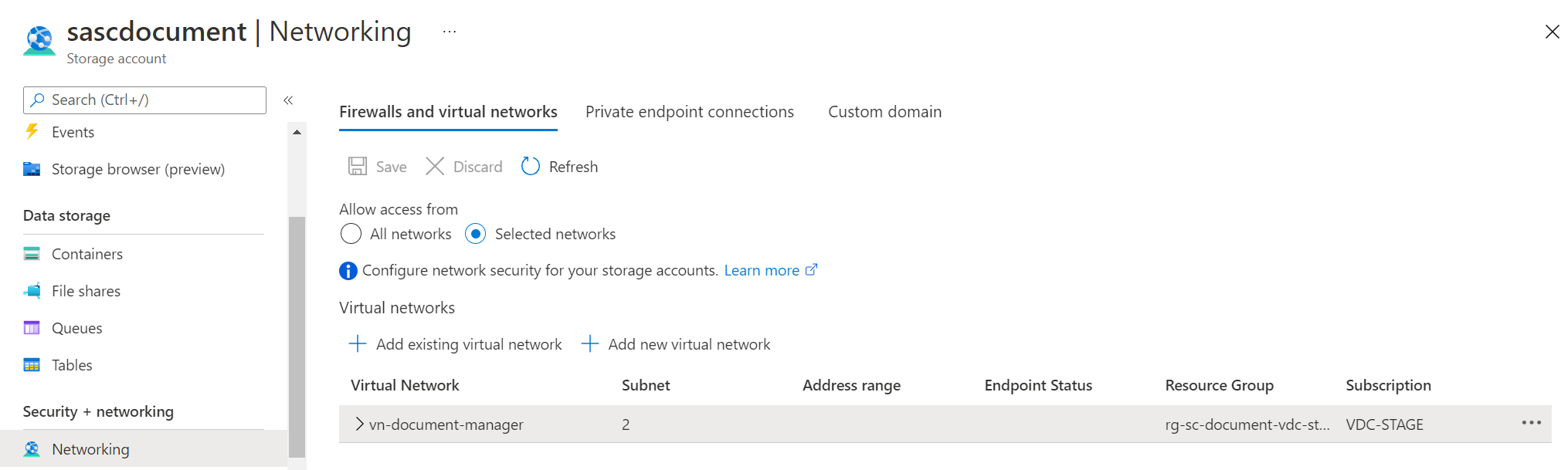
1. Add the virtual networking and the subnet (subnet-function) for the function app



1. Add the virtual networking and the subnet (subnet-webapp) for the Document Uploader API Web app

Do the same as explained above, but in the Web App you have to add the VNET Integration to the Web App subnet.

1. Add the two subnets to the networking configuration for the Storage.



After this configuration the Web App and the Function App will be able to reach the storage.

**Development Environment – Specific configuration**

**Connections String to the Storage - Development : User Secrets or appsettings.development.json file; Production : Azure Connection Strings Settings**

The challenge in this project is there are two projects the Web Api DocumentUploader and the Azure Function DocumentConsummer both projects make use of the common library StorageLibrary where can be find the classes responsible to operate the FileShare, Storage Queue, and Storage Table.

All of those make use of the Storage Connection String located at

"ConnectionStrings": {

    "DefaultStorageConnection": "DefaultEndpointsProtocol=https;AccountName=unistaddocumentssa;AccountKey=P5qz…gBFQ==;EndpointSuffix=core.windows.net"

}

However, considering the local development environment, the Document Consumer Azure Function has a different approach on the way the configuration is stored, this gets complicated due to the fact the ***StorageLibrary*** ***class*** is shared. The default location above is the correct place for the connection string in all projects using the Storage, with just one exception, the Document Consumer function trigger connection is stored in the location: ***local.settings.json -> Values : DefaultStorageConnection***.

**ConfigurationLibrary Project – ConfigSettings.cs Class**

This class stores some constant variables used across all projects.

|  |  |  |
| --- | --- | --- |
| Parameter | Default Value | Description |
| STORAGE\_CONNECTIONSTRING\_NAME | DefaultStorageConnection | Name of the storage connection string configuration used in the configuration files. |
| QUEUE\_CONNECTIONSTRING\_NAME | DefaultStorageConnection |  |
| QUEUE\_TOPROCESS\_NAME | unistad-toprocess |  |
| TABLE\_JOBS\_NAME | unistadjobs | Name of the table used to store the job status |
| TABLE\_PATITION\_KEY | unistad | Name of the table used to store the job status |
| FILE\_SHARE\_NAME | unistad-files | File share name used to store the files. |
| FILE\_SHARE\_UPLOADED\_FOLDER | \_jobs\_uploaded | Bear in mind Azure File Share delimiter is //, however when testing using the File System uses \\. |
| FILE\_SHARE\_FAILED\_FOLDER | \_jobs\_failed | Bear in mind Azure File Share delimiter is //, however when testing using the File System uses \\. |
| FILE\_SHARE\_UNISTAD\_FOLDER | unistad | Bear in mind Azure File Share delimiter is //, however when testing using the File System uses \\. |
| APP\_SETTINGS\_SECTION |  |  |

# Deployment Troubleshooting

The list of problems faced to deploy the solution using Docker Containers.

|  |  |  |
| --- | --- | --- |
| **Issue** | **Solution** | **Reference** |
| Deploy both containers together. | Added docker-compose to the solution so Web App and Web Api could be deployed together using the same network in Docker. |  |
| Docker-composed raising error “duplicate mount point /root/.aspnet/https” | Docker Desktop settings enabled experimental features “User Docker Compose v2” | Workaround mentioned in this link  <https://github.com/docker/compose-cli/issues/1811> |
| Connection refused when trying to communicate to the Document Uploader API, from the browser in the host computer or from the Web App in the Docker container. | The Kerstel web server in the Document Uploader API container wasn’t listening correctly the requests port forwarded to the Container’s network adapter by Docker. I changed the environment variable ASPNETCORE\_URLS from <http://localhost> to http://\*:80 (or [https://\*:443](https://*:443)), therefore Kerstel passed to listening all requests reaching the docker network adapter. (See docker compose yaml file)  Ps: I wasn’t able to understand why it was working for the Web App but not for the Web API. | <https://pythonspeed.com/articles/docker-connection-refused/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-5.0#server-urls> |
| Not able to establish ssl connection when Web App calls the Web API in another container. “The SSL connection could not be established, see inner exception.” When deployed using traditional approach under IIS or Kestrel works fine. | As the communication happens internally between the containers, and there is already encryption in this communication by Docker, the SSL was disabled in the API, it was used only HTTP communication between Web App and Web API.  The changes implemented were.   * Document Uploader API Docker File exposes only port 80 * Document Uploader Service section in the Docker Compose YAML file   + Environment Variables ASPNETCORE\_URLS HTTP only.   + Environment variable ASPNETCORE\_HTTPS\_PORT was removed.   + The ports mapping for the https (443) was removed, as this port isn’t exposed anymore.   + The application invokes the API using http. | <https://github.com/dotnet/AspNetCore.Docs/issues/6199>  <https://github.com/dotnet/dotnet-docker/blob/main/samples/run-aspnetcore-https-development.md>  <https://stackoverflow.com/questions/62489718/openssl-verify-pfx-password-ok>  <https://medium.com/@ma1f/docker-dotnet-3d979f56efe6>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel/endpoints?view=aspnetcore-5.0>  <https://docs.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-5.0&tabs=visual-studio#trust-the-aspnet-core-https-development-certificate-on-windows-and-macos> |
| Web API Container wasn’t able to reach the Azurite Storage Service located in the Host Machine. | Added the following parameter in ENTRYPOINT command at the docker file. “--add-host host.docker.internal:host-gateway” This command allowed the api inside the container access the service in the host.  The configuration of the storage (azurite) connection string in appsettings needs to point out to the following url http:// host.docker.internal | <https://github.com/containers/podman/issues/8466> |
| Microsoft Indentity wasn’t working in the Docker container. | Actually there was a configuration missing in appsettings.Development.Json file, in the AzureAD was missing the "Instance": "https://login.microsoftonline.com/" |  |
| Git Action wasn’t able to deploy DocumentManagerApp, because the SSL error raised by Kestrel as mentioned above, therefore I must solve the SSL certificate issue. |  | **Using New-SelfSignedCertificate to create an exportable root CA Certificate and SSL Certificate**  <http://woshub.com/how-to-create-self-signed-certificate-with-powershell/>  **Create CA Certificate and Signed Certificate using Powershell**  <https://spandothers.wordpress.com/2018/02/20/generating-a-root-ca-certificate-and-signed-certificates-for-dev-using-powershell/>  **Managing Windows PFX Certificates through PowerShell**  <https://dev.to/iamthecarisma/managing-windows-pfx-certificates-through-powershell-3pj>  Convert cer certificate to pem and key.pem certificates.  <https://stackoverflow.com/questions/15413646/converting-pfx-to-pem-using-openssl>  <https://medium.com/@ma1f/docker-dotnet-3d979f56efe6>  <https://docs.microsoft.com/en-us/powershell/module/pki/new-selfsignedcertificate?view=windowsserver2019-ps>  <http://woshub.com/how-to-create-self-signed-certificate-with-powershell/>  <https://stackoverflow.com/questions/4024393/difference-between-self-signed-ca-and-self-signed-certificate>  **Creating Certificate Authorities and self-signed SSL certificates**  <https://web.archive.org/web/20160403100211/https://metabrot.rocho.org/jan/selfsign.html>  **Ignoring SSL certificate errors On .Net Core On HttpClient**  <https://blog.roushtech.net/2016/12/20/ignoring-ssl-certificate-errors-net-core-httpclient/> |
| Publishing in Azure Unistad Document Manager doesn’t start in Linux. | The problem are the spaces in the project name. It requires to change the Configuration -> General Settings -> Start command -> by adding a “\” before the space so Linux can understand there are spaces in the file name. |  |
| When published in production, I faced “AuthorizationFailure” When API or function tried to access the Storage. | I don’t know why but from my laptop, my application was able to access the Storage, however the Api and Function published in Azure couldn’t (although I have added the Outbound Ips as restrictions to access the Storage). In the end I had to scale up the application (Standard Tier), so I was allowed to configure Virtual networks for the Web App, Function App, and connect to the Storage. | <https://docs.microsoft.com/en-us/azure/storage/common/storage-network-security?tabs=azure-portal#manage-exceptions> |