# Mass Ingestion and Analyzing of Articles

Table of Contents

[Mass Ingestion and Analyzing of Articles 1](#_Toc531621390)

[Overview 2](#_Toc531621391)

[Azure Components 2](#_Toc531621392)

[Data Flow 3](#_Toc531621393)

[Cosmos DB – Articles DB Collections 4](#_Toc531621394)

[Collection – Ingest 5](#_Toc531621395)

[Format: 5](#_Toc531621396)

[Property Bag Properties 5](#_Toc531621397)

[Collection – Processed 5](#_Toc531621398)

[Format: 5](#_Toc531621399)

[Property Bag Properties 6](#_Toc531621400)

[Collection – Inspection 7](#_Toc531621401)

[Format 7](#_Toc531621402)

[Required App Settings 7](#_Toc531621403)

[Example 8](#_Toc531621404)

[Raw Records 8](#_Toc531621405)

[Article 8](#_Toc531621406)

[Image 1 9](#_Toc531621407)

[Image 2 9](#_Toc531621408)

[Processed Records 9](#_Toc531621409)

[Article 10](#_Toc531621410)

[Image 1 11](#_Toc531621411)

[Image 2 11](#_Toc531621412)

[Epilogue 12](#_Toc531621413)

# Overview

This project is designed as a reference to help any organization that desires to glean information from a wide stream of digital articles that have been written in any human language using standard, or out of the box, Microsoft Azure Services.

A digital article, in this case, is a document that meets a certain criteria. That is, it may have a title, it does contain a body, and may have image or video files associated with it. From this document a rich amount of data can be obtained simply using Azure Cognitive Services and might be used to enhance business processes, identify new customers or identify customer satisfaction issues.

These tasks can be accomplished, as mentioned, with existing out of the box Azure services by building a data pipeline utilizing Azure CosmosDB and Azure Storage as the storage facility, Azure Cognitive Services to obtain intelligent information from the data, and a combination of Azure Service Bus Queues and durable Azure Functions to build the actual pipeline.

This example pipeline:

* Translates any text to the language of choice.
* Finds Key Phrases, Entities, and user Sentiment in the provided text fields.
* Detects objects, text, and landmarks in any image associated with the article.
* Detects people by their gender and age in any image associated with the article.

Further, the architecture does not preclude a user from injecting customized ML/AI models. If those models are built as RESTful endpoints, extending the pipeline is a trivial task.

# Azure Components

The following list of Azure components are used in this example.

|  |  |
| --- | --- |
| **Azure Component** | **Purpose** |
| [Azure Storage Account](https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview) | Used to hold raw image and video files associated with an article.  A secondary storage account will be created with the Azure App Service and is used to host the Azure Function code and logs. |
| [Azure CosmosDB](https://azure.microsoft.com/en-us/services/cosmos-db/) | Used to hold article textual content and image and video tracking information. Further, the results of the Cognitive Service steps are stored in this CosmosDB as well. |
| [Azure Service Bus](https://azure.microsoft.com/en-us/services/service-bus/) | Hosts the Azure Service Bus Queues utilized by a series of Azure Functions |
| [Azure App Service](https://azure.microsoft.com/en-us/try/app-service/) | Hosts several Azure Functions to process the records serially. |
| [Azure Application Insights](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview) | Provides insights into the functionality of the Azure App Service |
| [Azure Cognitive Services](https://azure.microsoft.com/en-us/services/cognitive-services/directory/) | Provides the AI element of the pipeline using out of the box implementations of the following services:  [Computer Vision](https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/)  [Face](https://azure.microsoft.com/en-us/services/cognitive-services/face/)  [Translator Text](https://azure.microsoft.com/en-us/services/cognitive-services/translator-text-api/) |

# Data Flow

This example does not specifically define the ingestion engine in which content is collected and fed into the pipeline. It is expected that an organization requiring such capacity has already determined how to collect the data and now wishes to process it.

The following steps, and diagram, outline the logical flow of data through the pipeline after the content has been inserted into an Azure CosmosDB:

***Diagram***



\* Note that there can be many generators/ingestion processes feeding the CosmosDB and Azure Storage Account.

1. Ingest function is notified on document inserts into the CosmosDB database in the ***Articles*** collection.
   1. Image and video content are stored in an Azure Storage Account as blobs during the ingest process Notifications can be for one or more inserts.
   2. Inserts can be of type article, image, or video where image and video documents are associated with the originating article.
   3. Image and video documents are inserted before the article record as the Cosmos Document Id needs to be recorded in the article document.
   4. Only documents of type article are passed along to the queue for further processing as images and videos are processed in later steps.
2. The Translate function is triggered by a queue event containing the article document id and utilizes the Translate Text Azure Cognitive Service.
   1. Detect the language of the existing title and body of the article content and determine if
      1. The language can be translated
      2. The language is different than the pre-defined translation language.
   2. When necessary, translate the title and body text of the article.
   3. Where present collect the sentiment, key phrases, and entities from both the body and the title.
   4. Write a record to the ***Processed*** collection in CosmosDB recording translation and analysis results and how long it took.
   5. Pass the article id on to the next queue. While this is inefficient if the article does not contain images or videos, it ensures this article, if there is anything interesting found, will be passed along at the end of the pipeline.
3. The Detect function is triggered by a queue event containing the article document id and utilizes the Vision Azure Cognitive service.
   1. Detect objects and landmarks in the image using the detect API.
   2. Detect written words in the image using the OCR API
   3. Write to, or create, a record in the ***Processed*** collection in CosmosDB recording the detection results by putting each detected object or text into the *tags* property on the processed record.
   4. Pass the article id to the next queue for processing.
4. The Face function is triggered by a queue event containing the article document id and utilizes the Face Cognitive service.
   1. Detect faces for gender and age in the image using the Face API.
   2. Write to, or create, a record in the ***Processed*** collection in CosmosDB recording the detection results by putting each detected face (gender/age) into the *tags* property on the processed record.
   3. Pass the article id to the next queue for processing.
5. The Notify function is triggered by a queue event containing the article document id.
   1. Load the processed records for the article, images, and videos.
   2. Scan the processed documents for “interesting” content.
   3. If an article or ANY of it’s children trigger an “interesting”\* flag, send a notification off to the system of choice.\*\*

\* Interesting is something that must be defined by the end user of the architecture.

\*\* The final destination for notification on “interesting” articles is left to the reader to determine.

# Cosmos DB – Articles DB Collections

The two core collections for this pipeline are Ingest and Processed. The design between the two are similar in that there are very few first class members of the document but potentially many ***properties*** associated with the document depending on the type.

Further, there is a third collection implemented in this solution – ***Inspection***. This simply records information about the processing of the original article and it’s children. This table can be customized or removed depending on how the reader implements a notification schema for interesting content.

## Collection – Ingest

### Format:

{

"id" : "GUID",

"asset\_hash" : "hash of the item",

"artifact\_type" : "article|image|video",

"properties" :

{

Dependent on artifact\_type

}

}

### Property Bag Properties

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Property** | **Type** | **Required** | **Article** | **Image** | **Video** |
| original\_uri | String | Y | X | X | X |
| retrieval\_datetime | DateTime | Y | X | X | X |
| post\_date | DateTime | N | X |  |  |
| body | String | N | X |  |  |
| title | String | N | X |  |  |
| author | String | N | X |  |  |
| hero\_image | String | N | X |  |  |
| child\_images | Array(object) | N | X |  |  |
| child\_videos | Array(object) | N | X |  |  |
| internal\_uri | String | N |  | X | X |

#### Media Object

The media object is used for both child\_images and child\_videos. The field media\_id is the Document ID of the media document in the Articles table.

{

"mediaId": "9d30724f5b8043e49552f4b8eb02f010",

"origUri": "https://dummy/thirdgrade.jpg",

"internalUri": "https://dangtestrepo.blob.core.windows.net/scraped/thirdgrade.jpg"

}

## Collection – Processed

### Format:

{

"id" : "GUID",

"artifact\_type" : "article|image|video", **<- Matches Ingest table type**

“parent” : “parent id”, **🡨 Paired with the original document in Articles collection**

"properties" : {

.... dependent on artifact type ......

}

"tags" :[interesting/need alerting/other]

}

### Property Bag Properties

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Property** | **Type** | **Required** | **Article** | **Image** | **Video** |
| processed\_datetime | DateTime | Y | X | X | X |
| processed\_time\* | Int | Y | X | X | X |
| title\*\* | object | N | X |  |  |
| body\*\* | object | N | X |  |  |
| tags | Array(string) | N | X | X | X |

\*\* Text Field Analytics objects

#### Text Field Analytics

The body and title are represented on an article with the following text field analytics object. It contains

* The language that the original article was written in.
* The language in which the process was asked to translate to
* The translated value
* Key phrases found in translated value
* The sentiment of the message
* Entities that were detected in the translated value.

"body": {

"type": "Body",

"orig\_lang\_code": "es",

"lang\_code": "en",

"value": "The leaders meet in the framework of the G20 ……..",

"key\_phrases": [

"Argentine head of state",

"British premier",

"framework",

"country",

"time",

"G20 summit",

"leaders"

],

"sentiment": 0.5,

"entities": [

{

"OriginalText": "British premier",

"Name": "Prime Minister of the United Kingdom",

"BingId": "2570ebea-8c42-048a-3350-57c9e4169167",

"WikipediaUrl": "https://en.wikipedia.org/wiki/Prime\_Minister\_of\_the\_United...."

}

....

]

}

## Collection – Inspection

### Format

For this example, documents in this collection are simple JSON objects containing information about the processing steps that have occurred and is .

# Required App Settings

There are a number of application settings required in the Azure Function App to ensure that the code will run correctly. Given the source code, these values are required:

|  |  |
| --- | --- |
| **Key** | **Value** |
| ArticleIngestTrigger\_ConnectionString | Connection string to CosmosDB, I think created by linking the trigger to the app? Probably need something better. Example:  AccountEndpoint=https://dangcosmosdb.documents.azure.com:443/;AccountKey=X6tyyMXo3jbmm8iHFiMFkbkKR5Pev8FjQF0MQM04yiTHCaIa0qxkKdVIlAhRFVm1Doo3WyMTaMe7gK7P9YFSig==; |
| CosmosDbName | Database Name |
| CosmosCollectionName | Ingest collection name |
| CosmosProcessedCollectionName | Processed collection name |
| CosmosInspectionCollectionName | Inspection collection name |
| FaceAPIUri | Base URI to the Face Service  Example:  <https://eastus.api.cognitive.microsoft.com/face/v1.0> |
| FaceAPIKey | Api Key to Face service |
| TranslationAPIKey | ApiKey to translation service |
| TranslationAPIUri | Base URI to translation service, i.e.  <https://api.cognitive.microsoft.com/sts/v1.0>  But what is really used is for the global service:  <https://api.cognitive.microsofttranslator.com> |
| TranslationAPITargetLanguage | Target language to perform the translation to. i.e. en |
| VisionAPIKey | Key to the computer vision API |
| VisionAPIUri | Uri to the vision service, i.e.  <https://eastus.api.cognitive.microsoft.com/> |
| TextAPIKey | API Key To Text Service |
| TextAPIUri | Uri to the text service, i.e.  <https://eastus.api.cognitive.microsoft.com/text/analytics/v2.0> |

# Epilogue

This design uses the queue notification/Azure function pattern. This is not the only pattern that can be used to accomplish this data flow.

[Azure Service Bus Topics](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-how-to-use-topics-subscriptions) could be used as well in a different pattern that could process different parts of the article in a parallel as opposed to the serial processing done in this example. Topics would potentially be useful article inspection processing time is critical. A useful comparison between Azure Service Bus Queues and Azure Service Bus Topics can be found [here](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions).

Azure functions could also be implemented in an [Azure Logic App](https://azure.microsoft.com/en-us/services/logic-apps/) as well. A useful comparison between durable functions and Logic apps can be found [here](https://docs.microsoft.com/en-us/azure/azure-functions/functions-compare-logic-apps-ms-flow-webjobs).

Finally, all of the AI introduced in this article are out of the box services provided by Azure. There is nothing in this architecture that prevents an implementation that utilizes customized AI components in this process.

For example a customized model that detects certain people in an image as opposed to the generic people count, gender and age could easily be swapped out in this implementation assuming that the model was exposed as a RESTful API.