Mass ingestion and analysis of news feeds on Azure

02/01/2019 • 5 minutes to read • Contributors

In this article

This example scenario describes a pipeline for mass ingestion and near real-time analysis of documents using public RSS news feeds. It uses Azure Cognitive Services to offer useful insights including text translation, facial recognition, and sentiment detection.

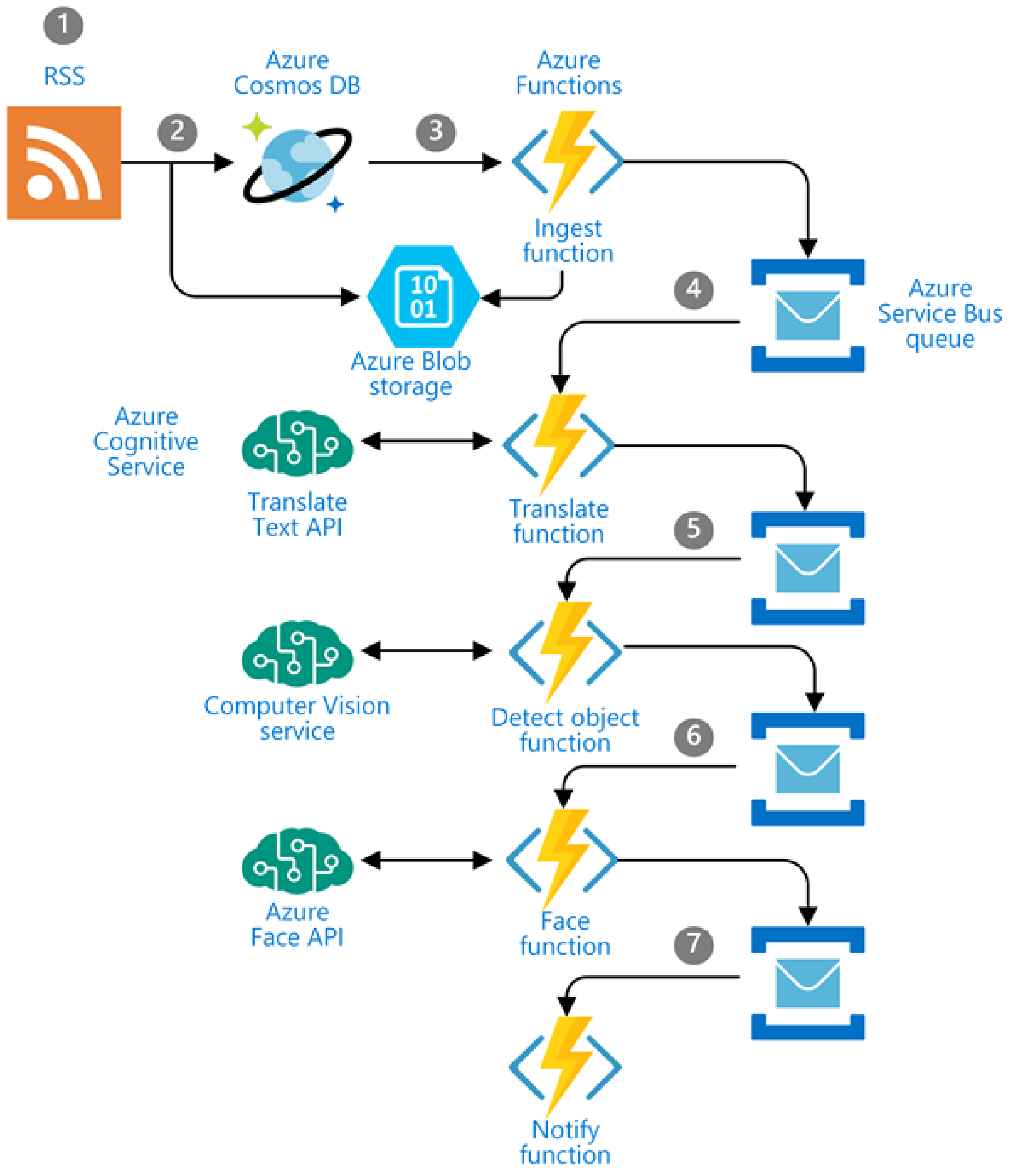
This scenario contains examples for English, Russian, and German news feeds, but you can easily extend it to other RSS feeds. For ease of deployment, the data collection, processing, and analysis are based entirely on Azure services.

Relevant use cases

While this scenario is based on processing of RSS feeds, it's relevant to any document, website, or article where you would need to:

- Translate any text to the language of choice.
- Find key phrases, entities, and user sentiment in digital content.
- Detect objects, text, and landmarks in images associated with a digital article.
- Detect people by their gender and age in any image associated with digital content.

Architecture



The data flows through the solution as follows:

1. An RSS news feed acts as the generator that obtains data from a document or article. For example, with an article, data typically includes a title, a summary of the original body of the news item, and sometimes images.
2. A generator or ingestion process inserts the article and any associated images into an Azure Cosmos DB Collection.
3. A notification triggers an ingest function in Azure Functions that stores the article text in CosmosDB and the article images (if any) in Azure Blob Storage. The article is then passed to the next queue.
4. A translate function is triggered by the queue event. It uses the Translate Text API of Azure Cognitive Services to detect the language, translate if necessary, and collect the sentiment, key phrases, and entities from the body and the title. Then it passes the article to the next queue.
5. A detect function is triggered from the queued article. It uses the Computer Vision service to detect objects, landmarks, and written words in the associated image, then passes the article to the next queue.
6. A face function is triggered is triggered from the queued article. It uses the Azure Face API service to detect faces for gender and age in the associated image, then passes the article to the next queue.
7. When all functions are complete, the notify function is triggered. It loads the processed records for the article and scans them for any results you want. If found, the content is flagged and a notification is sent to the system of your choice.

At each processing step, the function writes the results to Azure Cosmos DB. Ultimately, the data can be used as desired. For example, you can use it to enhance business processes, locate new customers, or identify customer satisfaction issues.

Components

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

- Azure Storage is used to hold raw image and video files associated with an article. A secondary storage account is created with Azure App Service and is used to host the Azure Function code and logs.
- Azure Cosmos DB holds article text, image, and video tracking information. The results of the Cognitive Services steps are also stored here.
- Azure Functions executes the function code used to respond to queue messages and transform the incoming content. Azure App Service hosts the function code and processes the records serially. This scenario includes five functions: Ingest, Transform, Detect Object, Face, and Notify.
- Azure Service Bus hosts the Azure Service Bus queues used by the functions.
- Azure Cognitive Services provides the AI for the pipeline based on implementations of the Computer Vision service, Face API, and Translate Text machine translation service.
- Azure Application Insights provides analytics to help you diagnose issues and to understand functionality of your application.

Alternatives

- Instead of using a pattern based on queue notification and Azure Functions, use another pattern for this data flow. For example, Azure Service Bus Topics can be used to processes the various parts of the article in parallel as opposed to the serial processing done in this example. For more information, compare queues and topics.
- Use Azure Logic App to implement the function code and implement record-level locking such as Redlock (needed for parallel processing until Azure Cosmos DB supports partial document updates). For more information, compare Functions and Logic Apps.
- Implement this architecture using customized AI components rather than existing Azure services. For example, extend the pipeline using a customized model that detects certain people in an image as opposed to the generic people count, gender, and age data collected in this example. To use customized machine learning or AI models with this architecture, build the models as RESTful endpoints so they can be called from Azure Functions.
- Use a different input mechanism instead of RSS feeds. Use multiple generators or ingestion processes to feed Azure Cosmos DB and Azure Storage.

Considerations

For simplicity, this example scenario uses only a few of the available APIs and services from Azure Cognitive Services. For example, text in images can be analyzed using the Text Analytics API. The target language in this scenario is assumed to be English, but you can change the input to any supported language of your choice.

Scalability

Azure Functions scaling depends on the hosting plan you use. This solution assumes a Consumption plan, in which compute power is automatically allocated to the functions when required. You pay only when your functions are running. Another option is to use an Azure App Service plan, which allows you to scale between tiers to allocate a different amount of resources.

With Azure Cosmos DB, the key is to distribute your workload roughly evenly among a sufficiently large number of partition keys. There's no limit to the total amount of data that a container can store or to the total amount of throughput that a container can support.

Management and logging

This solution uses Application Insights to collect performance and logging information. An instance of Application Insights is created with the deployment in the same resource group as the other services needed for this deployment.

To view the logs generated by the solution:

1. Go to Azure portal and navigate to the resource group created for the deployment.
2. Click the **Application Insights** instance.
3. From the **Application Insights** section, navigate to **Investigate\Search** and search the data.

Security

Azure Cosmos DB uses a secured connection and shared access signature through the C# SDK provided by Microsoft. There are no other externally facing surface areas. Learn more about security best practices for Azure Cosmos DB.

Pricing

The estimated daily cost to keep this deployment available is approximately $20 U.S. with no data moving through the system.

Azure Cosmos DB is powerful but incurs the greatest cost in this deployment. You can use another storage solution by refactoring the Azure Functions code provided.

Pricing for Azure Functions varies depending on the plan it runs in.

Deploy the scenario

> **Note**
>
> You must have an existing Azure account. If you don't have an Azure subscription, create a free account before you begin.

All the code for this scenario is available in the GitHub repository. This repository contains the source code used to build the generator application that feeds the pipeline for this demo.