Laterpay – Innovation Project

About the project

In this development project, Microsoft teamed up with LaterPay to offer intelligence for recommendations and new services for publishers and users within LaterPay's micropayment system. We started this project after a first contact at a Microsoft conference. We introduced LaterPay to the power of Cognitive Services, which can add deeper insights to a broad set of content types, text, images, videos.

We are using the lightweight and powerful features of Azure Functions, to do frequent crawler tasks, and start from there the intensive process analyzing content more intensive. And through the scalability of Azure Functions, we can do this in a very fast and efficient way.

Cognitive Services supports us in several ways. It enables us to detect, what content is on photos, used in this article. In case persons are on the photo, we try to detect their emotions. We can automatically detect the language. In case it is english, we added sentiment and key phrase detection to the articles.

By using these technologies, and through better access to a news article, you can better create a profile of the reader of this content. Therefore it is key to get a structured access to the content, to better analyze the content, for example by sentiment of text, or content of images.

LaterPay is keen to use these possibilities to improve and extend their business models and offerings for their clients.

Core team

- Cosmin Ene – Founder and CEO, LaterPay
- Kristian Glass - CTO, LaterPay
- Tobias Woldrich - Product Manager, LaterPay
- Jennifer Becker - Project Manager, LaterPay
- Oliver Scheer - Senior Technical Evangelist, Microsoft

Customer profile

LaterPay is a two-click system that enables impulse purchasing of digital content across platforms, websites and content types. LaterPay provides a SaaS payment infrastructure that enables publishers to get users over the hurdle of paying for content. By not requiring either upfront registration or payment, and by aggregating micropayments, LaterPay facilitates converting users into paying customers.

LaterPay believes that good content is valuable and that this value should have a price. High quality content should be profitable again!

Used Technologies

In our solution we used Azure Queues, SQL Azure, Azure Web Apps, Power BI embedded, and Cognitive Services.

Detailed Cognitive Services:

- Computer Vision API
- Emotion API
- Text Analytics API

Pain point or problem area to be addressed

LaterPay is the payment provider for paid content on various publisher webpages and news portals. This means that paid content is purchased and paid via LaterPay. LaterPay is interested in exploring additional services they can offer their users and merchants.

Recommendation services are commonly used by publishers to help users discover more content they might be interested in. By offering publishers the ability to say to users "You bought our article X, here is similar paid content we offer that you might be interested in", LaterPay can help its merchants increase their sales and value delivered to their users, at no extra integration effort.

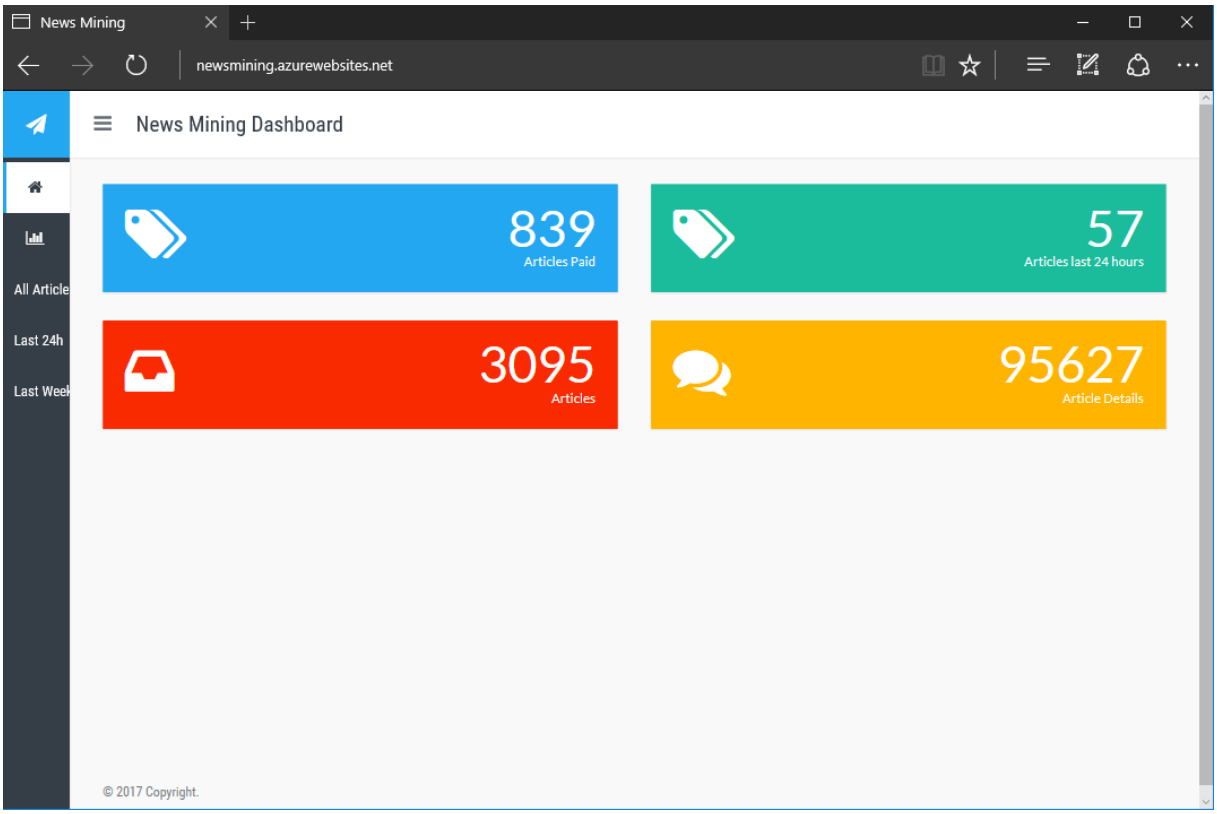
Solution, steps, and delivery

The entire solution is up and running. It is finalized, but it can be easily extended with new Cognitive Services, if available. It contains the following building blocks:

- Azure Queue
- Azure Functions
- Azure Web Apps
- SQL Azure
- Power BI Workspace
- Power BI embedded
- Cognitive Services

Solution

The solution is live and running on Azure. The URL for the next weeks is https://newsmining.azurewebsites.net.



Most important steps of the solutions

1. Accessing the content
2. Parsing the content
3. Adding deeper analyzing tools like Cognitive Services
4. Aggregating the information
5. Accessing the new insights very easily

Step 1: Getting the content

The solution starts with getting access to the content of news portals and publishers. Therefore a web crawler and parser are needed, which frequently scan for new articles.

This is done by Azure Functions, which checks the RSS-Feed of the respective webpages every 15 minutes. Documentation how to set timing on Azure Functions

```
#load "..\Shared\ArticleQueueItem.csx"

// Needed for rss features
#r "System.ServiceModel"

using System;
using System.Xml;
using System.ServiceModel.Syndication;

public static void Run(TimerInfo myTimer, TraceWriter log, ICollector<ArticleQueueItem>
outputQueueItem)
{
    log.Info($"Trigger function - AddRssFeedItemsToQueue: {DateTime.Now}");

    // List of rss feeds
    var onlineFeeds = new List<string>();
    onlineFeeds.Add("http://www.anewsprovideringermany.de/schlagzeilen/index.rss");
```

```
    foreach (var feedUrl in onlineFeeds)
    {
        IEnumerable<SyndicationItem> feedItems;
        using(XmlReader reader = XmlReader.Create(feedUrl))
        {
            var feed = SyndicationFeed.Load(reader);
            reader.Close();
            feedItems = feed.Items;
        }

        foreach (var item in feedItems)
        {
            // Create new queue element and add it to the outputQueueItem
            ArticleQueueItem a = new ArticleQueueItem();
            a.Url = item.Id;
            outputQueueItem.Add(a);

            // Logging
            log.Info("new url from rss-feed: " + item.Id);
        }
    }
}
```

Code for Azure Function: CheckRssFeedForNewItems

The class `ArticleQueueItem` is used to add an Element into a new queue, and store it in an external shared file to be used in different Azure Functions.

Information about the content of paid articles is especially interesting for LaterPay to provide a basis for recommendations. Therefore it is necessary to scan the public parts of all existing paid articles from publishers in an easy and fast manner. Up until now (March 2017), we have scanned the content of over 800 paid content articles. To scan them fast, the process scans several pages to get the URLs of paid content articles, and queue them in an Azure Queue.

Step 2: Parsing the new content

This Azure Queue `articleurl` is watched by an Azure Function to parse the queued URLs. An insert will trigger the execution of the Azure Function `ParseSpiegelPage`.

Every new URL in the queue is downloaded and parsed. The process of parsing needs access to the DOM(Document Object Model) and pure HTML code. Here we are using the library HTMLAgilityPack, which is easy to integrate in Azure Functions via NUGet, and enables a solid access to needed properties of a HTML-Document and the content.

Here is a part of the Azure Functions, to download and parse a page.

```
public static void ParseThePage(string url, string html, TraceWriter log)
{
    // add new or get existing id of an article base on the url
    var articleId = InsertOrUpdateArticle(url, "", log);

    // HtmlDocument enables parsing, load text in to HtmlDocument
    var doc = new HtmlDocument();
    doc.LoadHtml(html);

    // Headline
    nodes = doc.DocumentNode.SelectNodes("//span[@class='headline']");
    if (nodes != null)
    {
        for (int i = 0; i < nodes.Count; i++)
        {
            var node = nodes[i];
            var key = C_Headline;
            var value = node.InnerText;
            if (value != null || value.Length > 0)
```

```
            {
                InsertOrUpdateArticleProperty(log, articleId, key, value);
            }
        }
    }
    ...
}
```

The method `InsertOrUpdateArticleProperty` saves the parsed information and key in a Database with an association to the root article.

![LaterPay Simplified Database Schema]({{ site.baseurl }}/images/laterpay/dbdiagram.png)

In the later process the tables `Articles`and `ArticleProperties` will be aggregated into the 3rd Table `ArticleAggregate` for performance and reporting reasons.

Step 3: Adding deeper analyzing tools like Cognitive Services

After parsing the content of a webpage (=article), the custom analyzing tools start. For a proof of concept, we integrated the Computer Vision API of the Microsoft Cognitive Services and in case that persons are in images the Emotion API There are several usage scenarios for the Computer Vision API:

- What is on the photo of the news article? e.g. a car with happy people, a building, politicians
- Is this adult content?
- Are people in the image? How are the emotions of the people on the photos?

The following snippets, shows the code for getting information of a photo, and in case of Persons on the photo, more details about their emotions.

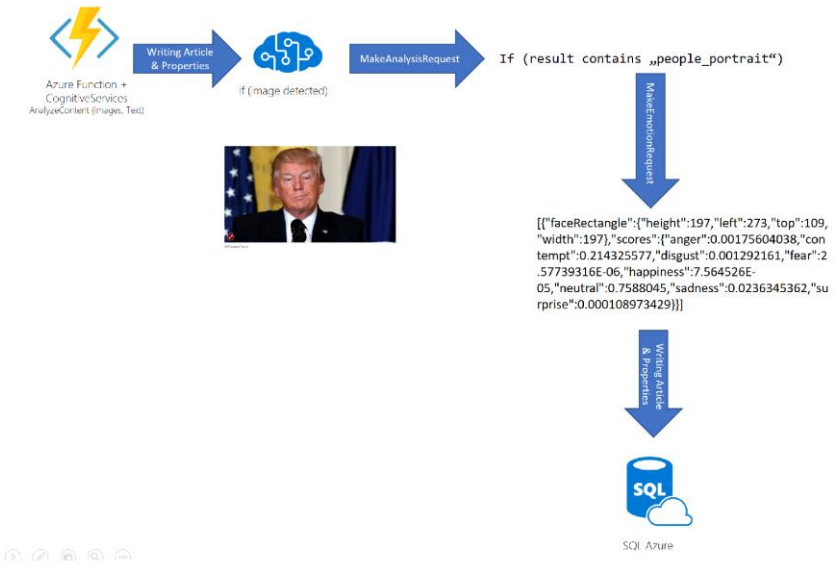
```
static async void MakeAnalysisRequest(TraceWriter log, Guid articleId, string
imageWebFilePath)
{
    var client = new HttpClient();
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key",
C_CognitiveServices_VisionAPI_Key);

    string requestParameters = "visualFeatures=Categories&language=en";
    string uri = "https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?" +
requestParameters;

    HttpResponseMessage response;
    string json;

    var imageDownloaderClient = new HttpClient();
    byte[] byteData = await imageDownloaderClient.GetByteArrayAsync(imageWebFilePath);

    using (var content = new ByteArrayContent(byteData))
    {
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");
        response = await client.PostAsync(uri, content);
        json = await response.Content.ReadAsStringAsync();

        VisionResult result = JsonConvert.DeserializeObject<VisionResult>(json);

        foreach(var cat in result.categories)
        {
            InsertOrUpdateArticleProperty(log, articleId, cat.name, cat.score.ToString());

            if (cat.name == "people_portrait")
```

```
            {
                MakeEmotionRequest(log, articleId, byteData);
            }
        }
        Debug.WriteLine(json);
    }
}

public static async void MakeEmotionRequest(TraceWriter log, Guid articleId, byte[]
byteData)
{
    var client = new HttpClient();

    // Request headers
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key",
C_CognitiveServices_EmotionAPI_Key);

    string uri = "https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize?";
    HttpResponseMessage response;
    string json;

    using (var content = new ByteArrayContent(byteData))
    {
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");
        response = await client.PostAsync(uri, content);
        json = response.Content.ReadAsStringAsync().Result;

        EmotionResult faces = JsonConvert.DeserializeObject<EmotionResult>(json);

        int faceId = 0;
        foreach(var f in faces.Property1)
        {
            InsertOrUpdateArticleProperty(log, articleId, $"Face {faceId} - Anger",
f.scores.anger.ToString());
            // ... surprise, sadness, etc.
        }
        Debug.WriteLine(json);
    }
}
```

SourceCode on GitHub

Again the Method `InsertOrUpdateArticleProperty` is used to store the data collected by the features of Cognitive Services API, in the database.

The shown code is very littled customized, based on this official Computer Vision API Sample Code and Emotion API Sample Code.



Analyzing the text was a very interesting part of the project. But because of the missing language support for German, it is currently disabled for german articles. We're enabling this feature whenever it becomes available.

Step 4: Aggregating the informations

This step is not absolutely required, but enables easier and faster access to the data. Especially for extremely fast access to all properties of an article, or selecting all articles with a specific value for one property, e.g. Author, Publishing date.

This is done with the Azure Function `UpdateAggregate` which calls every hour an aggregation process implemented in the stored procedure `sp_Aggregate` in the SQL Azure Database. This stored procedure creates a new table if it not already exists, and joins all properties of an article into one single row. This procedure is very fast, when you try to join up to a few hundred articles, which is succificent for an update every hour. It runs only for a few seconds. If you want to aggregate more than 1000 articles it takes up to 10 minutes.

So using this method every hour is very fast, and can be done in an Azure Function. For the first use, when you already have thousands of articles stored, you should better use SQL Management Studio to call this function, because you don't have the small timeout number that an Azure Function has.

```
create procedure [dbo].[sp_Aggregate]
as
begin

    -- Create Table if not extist
    IF NOT EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
               WHERE TABLE_NAME = N'ArticleAggregate')
    BEGIN

        -- drop table [ArticleAggregate]
```

```
          CREATE TABLE [dbo].[ArticleAggregate] (
               [Article_Id]   UNIQUEIDENTIFIER NOT NULL,
               [Url]  NVARCHAR (MAX)   NULL,
               [Html] NVARCHAR (MAX)   NULL,
               PublishDate DATETIME NULL,
               PublishTime DATETIME NULL,
               Author NVARCHAR (MAX)   NULL,
               Channel NVARCHAR (MAX)   NULL,
               Headline NVARCHAR (MAX)   NULL,
               HeadlineIntro NVARCHAR (MAX)   NULL,
               ArticleIntro NVARCHAR (MAX)   NULL,
               ArticleLength NVARCHAR (MAX)   NULL,
               ArticlePhotoUrl NVARCHAR (MAX)   NULL,
               ArticleComplete NVARCHAR (MAX)   NULL,
               Description NVARCHAR (MAX)   NULL,
               PaidContent NVARCHAR (MAX)   NULL,
               PublisherId NVARCHAR (MAX)   NULL,
               Source NVARCHAR (MAX)   NULL

               CONSTRAINT [PK_dbo.AggregateArticle] PRIMARY KEY CLUSTERED ([Article_Id]
ASC)
          );

     END

     select article_id
     into #tempids
     from articleaggregate

     INSERT INTO ArticleAggregate
     (
          Article_Id,
          Url,
          publishdate,
          PublishTime,
          Author,
          Channel,
          Headline,
          HeadlineIntro,
          ArticleIntro,
          ArticleLength,
          ArticlePhotoUrl,
          ArticleComplete,
          Description,
          PaidContent,
          PublisherId,
          Source
     )
     SELECT Articles.Id                                      as Article_Id
          --, COUNT(*)
          , Articles.Url                                     as Url
           , convert(datetime, tPublishDate.propertyvalue) as PublishDate
           , convert(datetime, tPublishTime.propertyvalue) as PublishTime
           , tAuthor.propertyvalue                          as Author
           , tChannel.propertyvalue                         as Channel
           , tHeadline.propertyvalue                        as Headline
           , tHeadlineIntro.propertyvalue                   as HeadlineIntro
           , tArticleIntro.propertyvalue                    as ArticleIntro
           , tArticleLength.propertyvalue                   as ArticleLength
           , tArticlePhotoUrl.propertyvalue                 as ArticlePhotoUrl
           , tArticleComplete.propertyvalue                 as ArticleComplete
           , tDescription.propertyvalue                     as Description
           , tPaidContent.propertyvalue                     as PaidContent
           , tPublisherId.propertyvalue                     as PublisherId
           , tSource.propertyvalue                          as Source
       from Articles
```

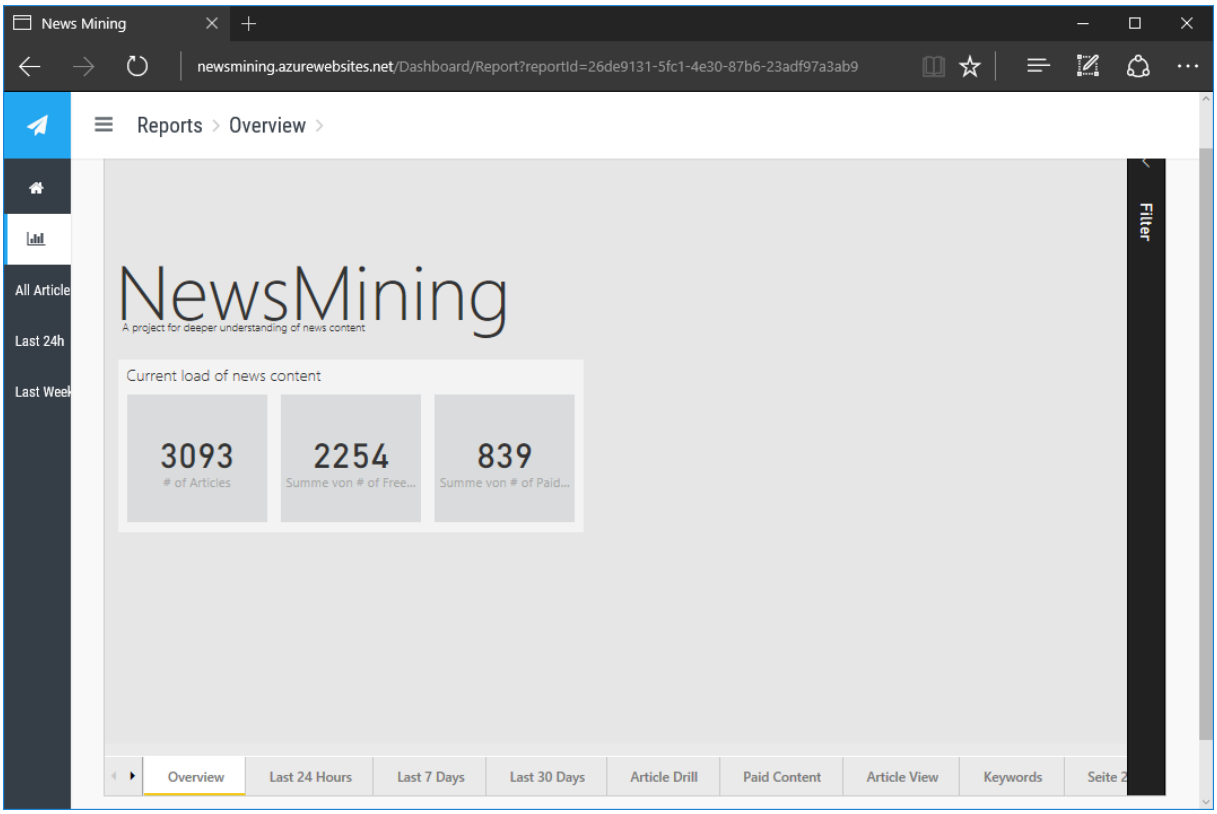
```
            left outer join ArticleProperties tPublishDate        on Articles.Id =
tPublishDate.Article_Id         and tPublishDate.propertyname       = 'PublishDate'
            left outer join ArticleProperties tPublishTime        on Articles.Id =
tPublishTime.Article_Id         and tPublishTime.propertyname       = 'PublishTime'
            left outer join ArticleProperties tAuthor             on Articles.Id =
tAuthor.Article_Id              and tAuthor.propertyname            = 'Author'
            left outer join ArticleProperties tChannel            on Articles.Id =
tChannel.Article_Id             and tChannel.propertyname           = 'Channelname'
            left outer join ArticleProperties tHeadline           on Articles.Id =
tHeadline.Article_Id            and tHeadline.propertyname          = 'Headline'
            left outer join ArticleProperties tHeadlineIntro      on Articles.Id =
tHeadlineIntro.Article_Id       and tHeadlineIntro.propertyname     = 'HeadlineIntro'
            left outer join ArticleProperties tArticleIntro       on Articles.Id =
tArticleIntro.Article_Id        and tArticleIntro.propertyname      = 'ArticleIntro'
            left outer join ArticleProperties tArticleLength      on Articles.Id =
tArticleLength.Article_Id       and tArticleLength.propertyname     = 'ArticleLength'
            left outer join ArticleProperties tArticlePhotoUrl    on Articles.Id =
tArticlePhotoUrl.Article_Id     and tArticlePhotoUrl.propertyname   = 'ArticlePhotoUrl'
            left outer join ArticleProperties tArticleComplete    on Articles.Id =
tArticleComplete.Article_Id     and tArticleComplete.propertyname   = 'ArticleComplete'
            left outer join ArticleProperties tDescription        on Articles.Id =
tDescription.Article_Id         and tDescription.propertyname       = 'Description'
            left outer join ArticleProperties tPaidContent        on Articles.Id =
tPaidContent.Article_Id         and tPaidContent.propertyname       = 'PaidContent'
            left outer join ArticleProperties tPublisherId        on Articles.Id =
tPublisherId.Article_Id         and tPublisherId.propertyname       = 'PublisherId'
            left outer join ArticleProperties tSource             on Articles.Id =
tSource.Article_Id              and tSource.propertyname            = 'Source'
       where Articles.Id not in (select article_id from #tempids)

     -- Tuning some fields
      update ArticleAggregate
        set PaidContent = 'No'
       where PaidContent is null

     -- Check Keywords
     IF EXISTS (SELECT *
                  FROM INFORMATION_SCHEMA.TABLES
                 WHERE TABLE_NAME = N'ArticleKeywords')
     BEGIN
          -- Drop Table
         drop table ArticleKeywords
     END

     select Id
          , Article_id
          , PropertyValue Keyword
       into ArticleKeywords
       from ArticleProperties
      where propertyname = 'Keyword'

end
GO
```

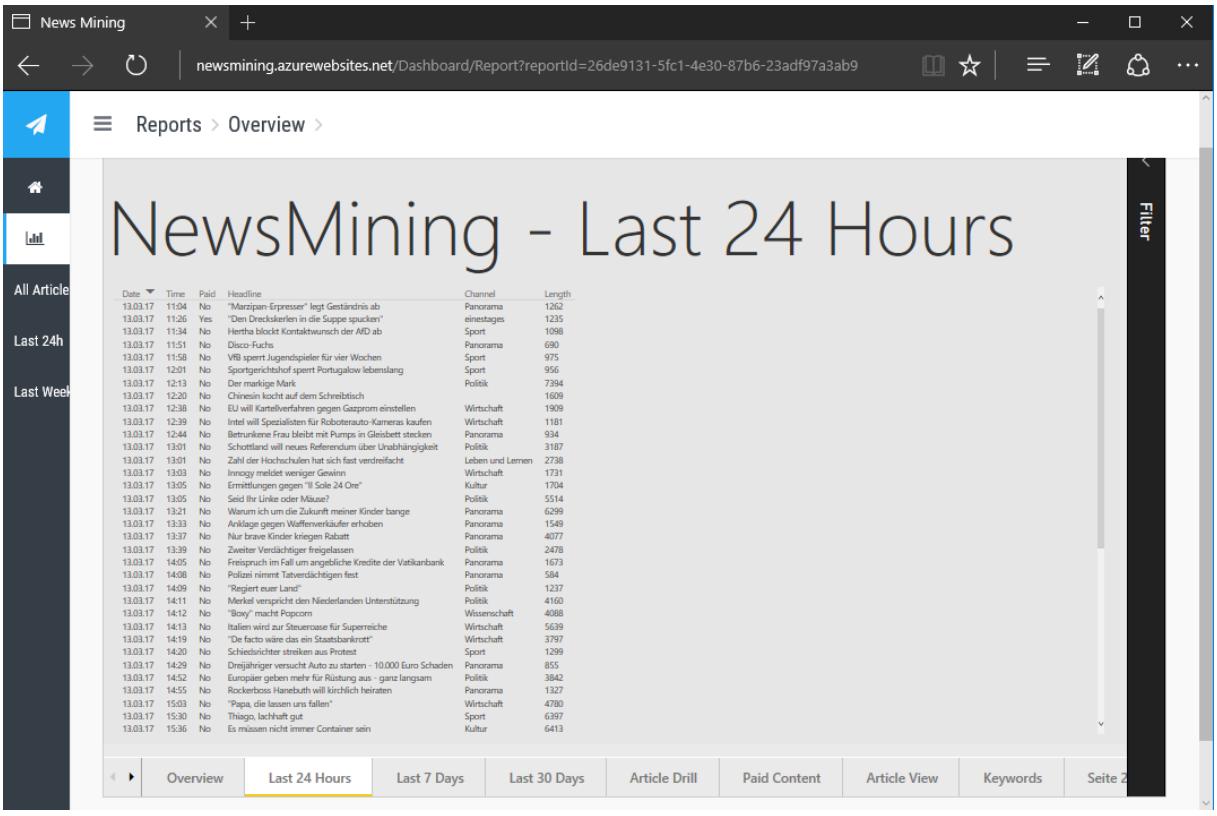
Step 5: Accessing the new insights very easly

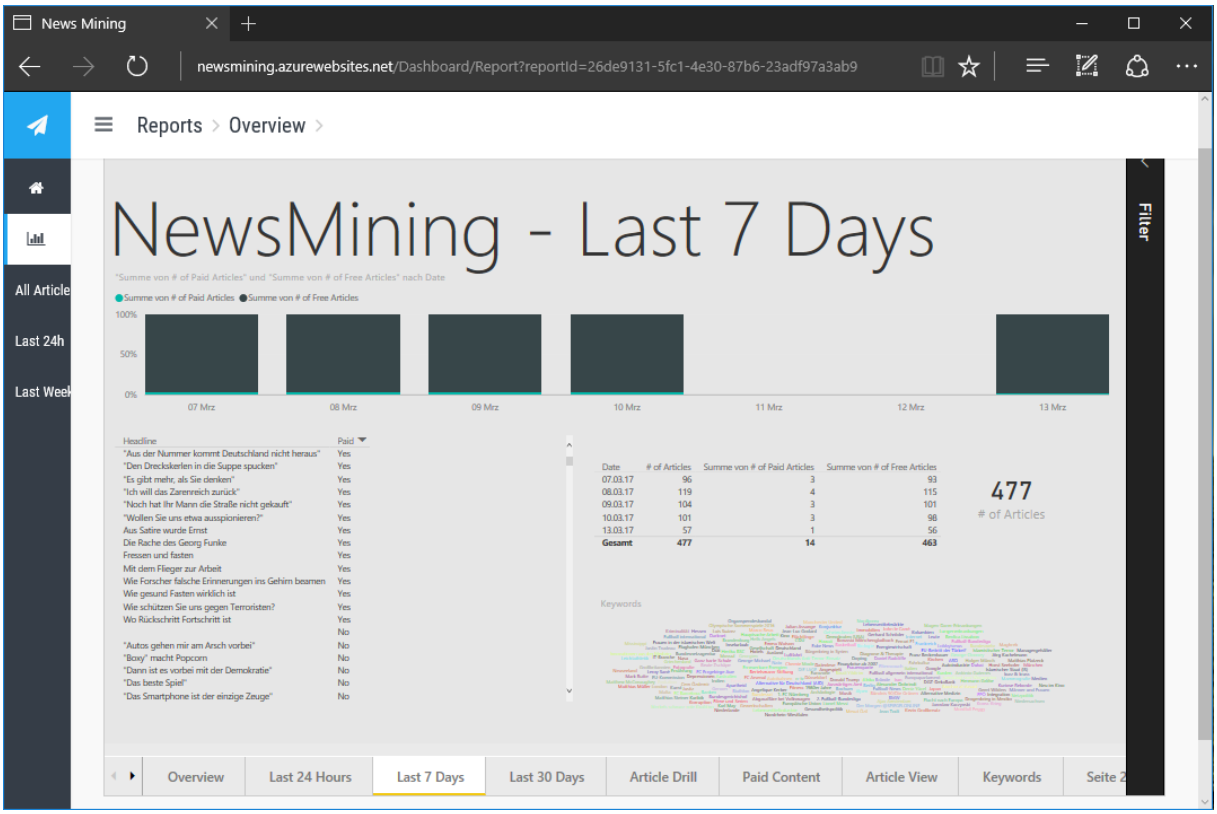
The easiest way to access the data is the webpage http://newsmining.azurewebsites.net. On the start dashboard, you can see the "important" numbers, like how many articles in total are in the database, and the numbers of paid content articles.

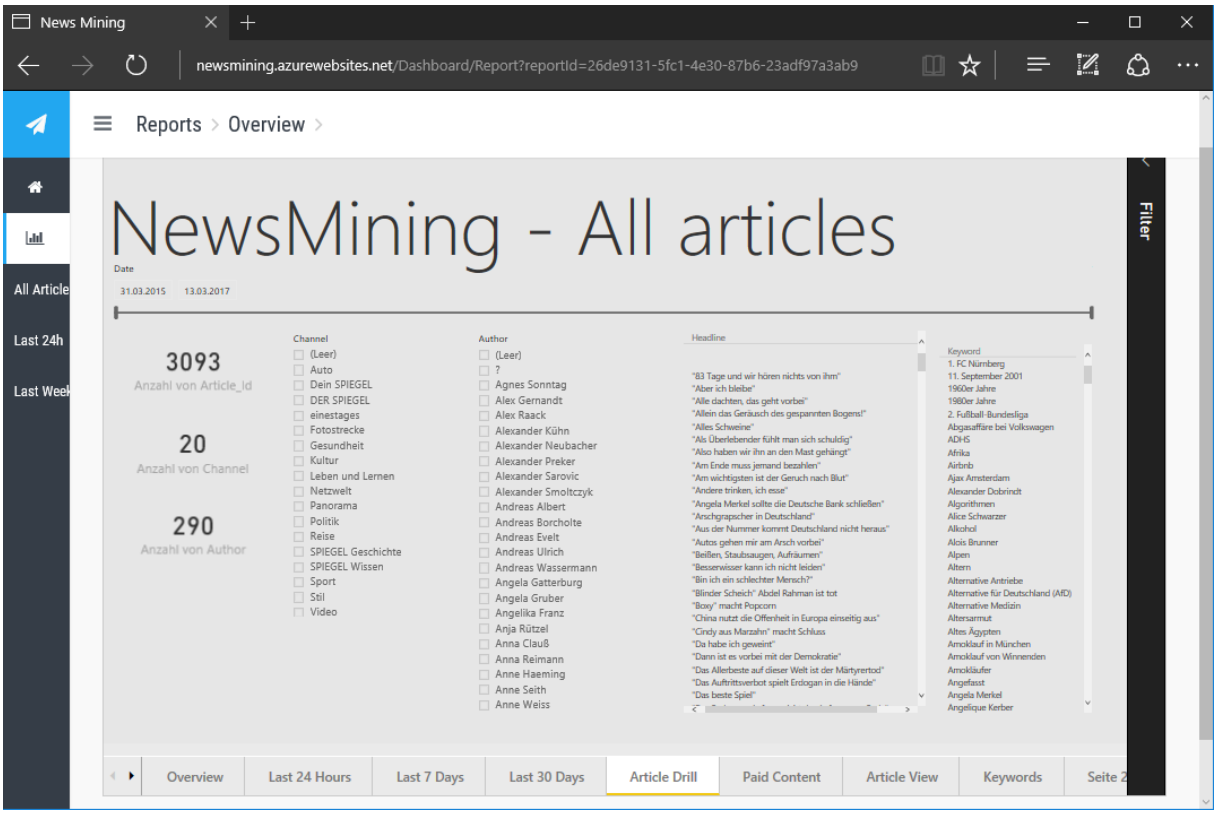
However, to get the real insights, Power BI Dashboards are created in order to drill into different aspects of the data and to get new insights. These dashboards are "uploaded" to Azure and hosted in a Power BI Workspace. From there, they could be embedded in any client-technology (iOS, Android, Windows, Web). In this project we only "embedded" them into the website.

Some impressions, how the final reports look like.


News Mining
newsmining.azurewebsites.net/Dashboard/Report?reportId=26de9131-5fc1-4e30-87b6-23adf97a3ab9
Reports
Overview
All Article
Last 24h
Last Week
NewsMining
A project for deeper understanding of news content
Current load of news content
3093
# of Articles
2254
Summe von # of Free...
839
Summe von # of Paid...
Filter
Overview
Last 24 Hours
Last 7 Days
Last 30 Days
Article Drill
Paid Content
Article View
Keywords


News Mining
newsmining.azurewebsites.net/Dashboard/Report?reportId=26de9131-5fc1-4e30-87b6-23adf97a3ab9
Reports
Overview
All Article
Last 24h
Last Week
NewsMining - Last 24 Hours
Filter
Overview
Last 24 Hours
Last 7 Days
Last 30 Days
Article Drill
Paid Content
Article View
Keywords





To get a website, we used Azure Web App to host a simple Asp.Net MVC App based on a standard template for Power BI embedded (Link).

More challenging is the case that there is no easy click & play way to create a Power BI Workspace in Azure. Hopefully, the productgroup is working on that. Meanwhile you have to use PowerBI-CLI, which at least allows for easy automation. Power BI CLI can be easly installed via npm:

```
npm install powerbi-cli -g
```
After that, you have to create a Power BI Workspace with.

```
powerbi create-workspace -c <collection> -k <accessKey>
```
Then upload a Power BI report.

```
powerbi import -w <workspaceid> -c newsmining -k <accesskey> -f Overview.pbix -n Overview -
o
```
This command uploads and overrides an existing report. The next challenge is, that you need to update the connection in the report, which is already on azure. Therefore you can use the folling command:

```
powerbi update-connection -c newsmining -w <workspaceid> -k <accesskey> -d <datasetid> -s
<a normal connection string  -u <username> -p <password>
```
After that, you can take the keys from the portal and add the you Web Project web.config-file. And your almost done.

Summary of steps

All steps are covered through different Azure Services.

![LaterPay Azure Dashboard]({{ site.baseurl }}/images/laterpay/azuredashboard.png)

With this approach, LaterPay got familiar with different technologies on the Azure Cloud Platform for the first time.

- Automatically scanning for new content and paid content on demand by Azure Functions.
- Parsing content (free and paid) with Azure Functions and storing the results in a SQL Azure Database.
- Analyzing special properties with Cognitive Services to get even more information out of the article.
- Drilling into data with Power BI embedded.

Technical details

Defining user stories

The kickoff started with developing different story lines for the future products of LaterPay. Together with the LaterPay team we defined several User Stories, and picked the most important one: Collecting the base data, on which we could build more in the future.

![LaterPay User Stories]({{ site.baseurl }}/images/laterpay/UserStories.png

Selecting technologies and technologie benefits

LaterPay is very commited to Microsoft, because of its openness and innovations. They're very excited about the current development of the Cognitive Services, therefore part of the project was to get familiar with and evaluate different areas of Azure Cloud Services and Microsoft Development techniques.

We selected Azure Web App for the hosting of the solution for the LaterPay-Team. This enables an easy way to test, deploy, and reuse knowledge.

For tasks like checking publishers RSS Feed and downloading/parsing the pages, we used Azure Functions. They are lightweight and scale easily.

For better asynchronous and faster scaling, we used Azure Queue Storage. That enables us to optimize the process of parsing a huge number of articles in parallel.

In the first approach we used Azure Table Storage to store informations about the article and properties. That was very easy to implement, and works with Power BI. But it is necessary to do complex joining of tables, which is quiet a performance challenge for Azure Table Storage.

To optimize the access to Power BI, and enabling more kind of deeper queries and optimizations with stored procedures, we switched to SQL Azure Database.

Cognitive Services were the first touch point for LaterPay and Microsoft. So they are a key part of the solution. We started with Vision and Emotion. In the future, we will enable Text and Video analysis.

To design smart and easy to use dashboards, we used Power BI Desktop.

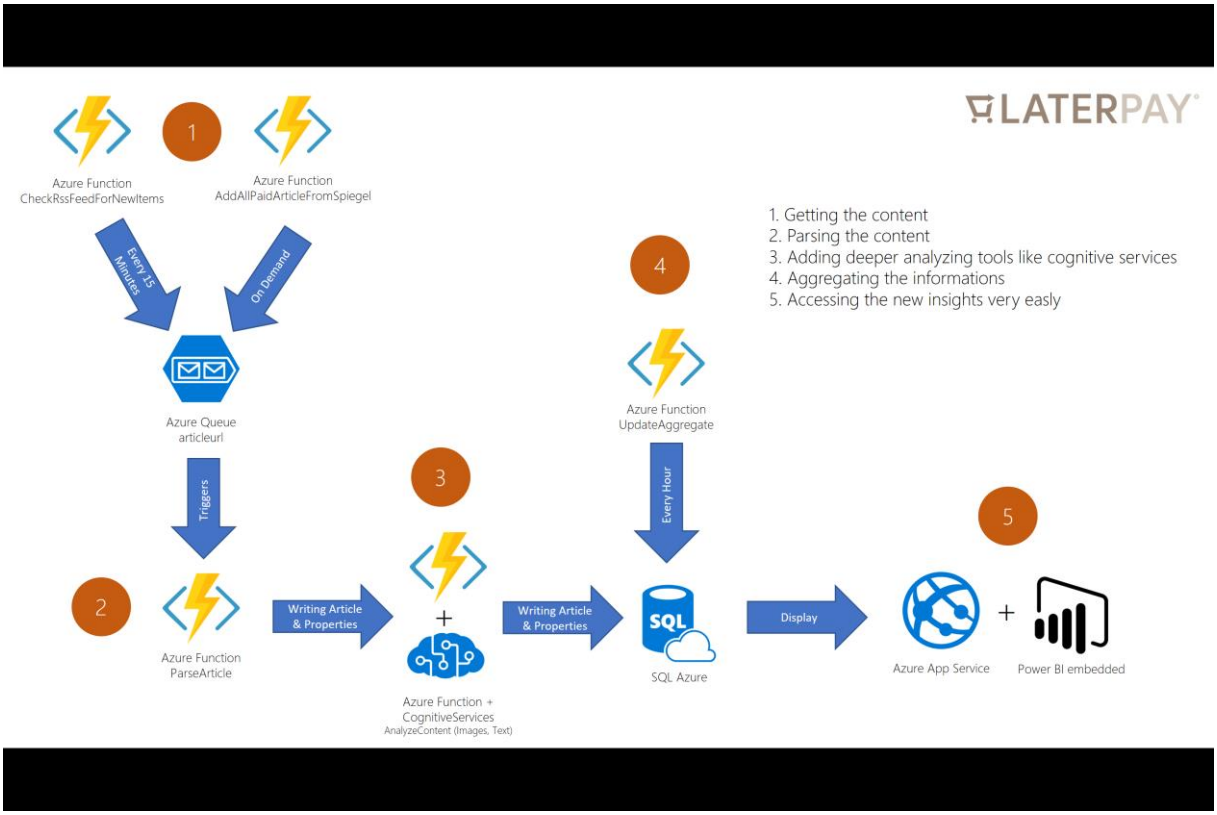
To embed those dashboards in other applications we used Power BI Embedded. This removes the need to install Power BI Desktop. It enables us to use it from any device on any client.

Architecture

We leveraged a number of Azure components in building this solution, including:

- Azure Queues
- Azure Functions
- Azure Power BI workspace
- Power BI Desktop
- Power BI embedded
- Azure SQL Database
- Azure App Service for Website hosting
- Cognitive Services

LaterPay Azure Value Stream



Key learnings

- Using Azure Table Storage seems very easy and lightweight, but it is really slow when used in Power BI Dashboards, where you need to pivot a significant number (more than 30) of different information about an article form several rows into one row. We switched from Azure Table Storage to a relation database, where we create and update an aggregation table every hour, for a much better performance.
- Cognitive Services are not really available for text in German language. Translating is possible, but a lot of "attitudes" are getting lost by this approach. So we removed it and wait for German language support.
- Handling Database connections correctly is key to avoid timeouts. Trying to optimize it with a serverless architecture such as Azure Functions didn't work with all known database patterns.
- Parsing news websites is easy, but from time to time they change their stylesheets, so you have to update the parser.

Conclusion

The developed solution enables LaterPay to use the collected public information of publishers to create a recommendation system for their users. This is the first step of using deeper insights on content to extend LaterPay's offerings to publishers and users. The project delivers also a very good start point and introduction to the Microsoft Azure Platform.

Quote Cosmin Ene - CEO, LaterPay: "By using Azure Functions and Power BI, Microsoft enables us to use the power of Cognitive Services in order to understand Paid Content consumption. This allows us to improve our services and serve content providers with innovative new features."

Quote Kristian Glass - CTO, LaterPay: "Azure Functions, with their open-source runtime, provide a great way of scaling to our exact need."

General lessons

Insights

- Developing with Azure Functions is extremely lightweight and fast, especially when using the online editor
- The process of aggregating the data for a small set of articles is very fast. For a larger number it takes more time than typical Azure Function should run. So we try to minimize the time, and for the setup the number of articles which are aggregrated.

How the learnings and insights can be applied elsewhere

The process of parsing and adding more insights to articles is absolutly reuseable for other news portals. The only thing you need to add is a different "parser" for a different news portal. This logic is encapsulated in one single file, and can be replaced or extended with other files for other news providers.