IMPLEMENTATION OF IMAGE BINARIZER IN MICROSOFT AZURE CLOUD

Arinze Okpagu
Matriculation No: 1324112
arinze.okpagu@stud.fra-uas.de

John Uchechukwu Emenike
Matriculation No: 1324701
john.emenike@stud.fra-uas.de

Abstract: ***Cloud computing offers quick innovation, improved efficiency to manage resources, and economies of scale through on-demand availability of computer system resources, especially data storage and computing power. This makes it easy means to run infrastructure more efficiently and lower operating costs. This project demonstrates the deployment of the existing implementation of the image binarization module in Microsoft Azure cloud.***

Keywords: Blob, Image Binarizer, Container, Storage, Table, Microsoft Azure.

I. INTRODUCTION

This project was carried out as part of the Cloud Computing module of the Frankfurt University of Applied Sciences. The project aims to demonstrate the implementation of image binarization using cloud services most especially the Infrastructure as a service (IaaS) model. Using the cloud services allows for processing flexibility and instant access to storage infrastructure. The image binarization module typically processes each training image and converts it to a binary image. The binary image provides sharper and clearer contours of various objects present in the original image. This feature extraction could be referred to as image thresholding or image segmentation [1]. The process of segmentation works by identifying a threshold value that effectively divides the composite pixel intensities of the image into two parts, each representing one of two objects: background and foreground. This implementation uses global thresholding approach to account for wide range of images and their corresponding pixel distributions. The global thresholding algorithms work well on images that contain objects with uniform intensity values on a contrasting background and perform poorly where there is a low contrast between the object and the background.

II. PROJECT BACKGROUND

The following key components and underlying processes are necessary to the implementation of the project:

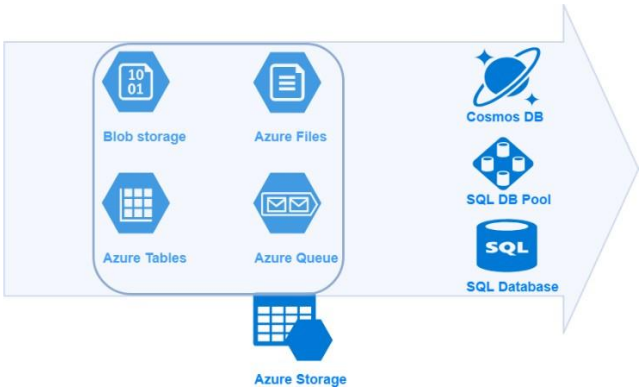

Figure 1: Cloud Storage Infrastructure [2]

A. Azure Storage Account

An Azure storage account contains all storage data objects such as blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure storage account is durable and highly available, secure, and massively scalable. Microsoft provides utilities and libraries for importing data from on-premises storage devices or third-party cloud storage providers [3].

B. Azure Blob Storage

Azure Blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that does not adhere to a particular data model or definition, such as text or binary data. They are optimized for serving images directly to a browser and storing files for distributed access [4].

C. Azure Table Storage

Azure Table storage is a service that stores structured NoSQL data in the cloud, providing a key/attribute store with a schemaless design that makes it easy to adapt data as the needs of application evolve. Azure tables are ideal for storing and querying huge sets of structured, and non-relational data [5].

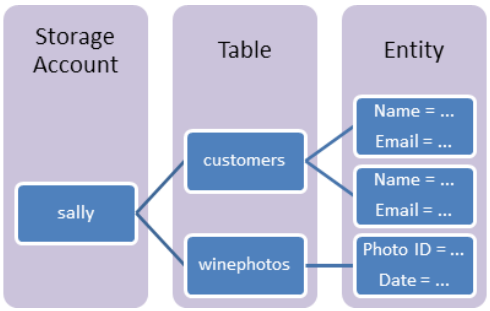

Figure 2: Table Storage [5]

A table is a collection of entities. Tables do not enforce a schema on entities, which means a single table can contain entities that have different sets of properties [5]. An entity is a set of properties, such as a row in a database.

D. Azure Queue Storage

Queues are commonly used to create a backlog of work to process asynchronously. Azure Queue storage provides cloud messaging between application components. Queue storage delivers asynchronous messaging between application components, whether they are running in the cloud, on the desktop, on an on-premises server, or on a mobile device [6]. The queue service concept is characteristically described as shown in figure 3 below:

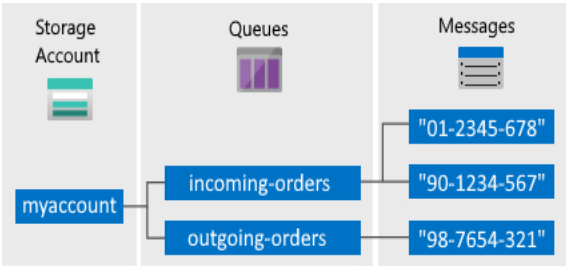

Figure 3: Queue Storage [6]

E. Experiment ML19/20 - 3.31

ML19/20-3.31 is the most recent console application of the image binarization module using C#, a type-safe object-oriented language that runs on the .NET framework. The console application typically takes a pixel image as an input and converts it to a binary image, comprising of a matrix 0s and 1s, and subsequently saved in a text file (.txt) in the user's computer. The input image is provided by the user as an argument by giving the file location path on the computer [7]. Several of these components were redeployed to enable seamless adaptation with the Microsoft Azure cloud resources.

The following were carefully considered while redesigning the implementation to make use of the Microsoft Azure cloud services:

1. How would the user submit the test image?
2. How and where to store a copy of the test image?
3. How and where to store the binary image after conversion?
4. How to keep track of the details each experiment?

To implement the above, the source code was defined to accept image inputs from any image or file hosting platforms using the uniform resource identifier of the subject image submitted via a trigger message, download and process the image, and store the results using blob storage and table storage. The Microsoft Azure Storage Emulator which emulates the Azure Blob, Queue, and Table services for local development purposes proved very useful towards successful implementation of this experiment.

III. PROJECT WORKLOW

The current implementation of the image binarization library; *ImageBinarizerLib.1.0.0* was packed as a NuGet package coupled with the LearningApi.1.3.1 [7] and added to the cloud computing project template [8] while making use of the *unicloud* storage account. The following systematically describes the steps in executing the experiment:

A. Trigger Message

The Trigger message serves to provide some of the key parameters such as the experiment identification number, description of the experiment, input file URI needed to trigger the experiment. A sample trigger message is given below:

{
"ExperimentID" : "EX1",
"InputFile" :
"https://cdn.pixabay.com/photo/2020/09/17/13/59/cat-5579221_960_720.jpg",
"Name" : "Arinze_Okpagu",
"Description" : "Black_Cat",
}

The trigger message is added to the *seivxgroup-trigger-queue* and passed as parameters for deserialization in the main code.

B. Deserialization

Upon receiving the message from the queue, the deserialization function decouples the message into their respective objects which can be stored in memory and further passed on as parameters to the experiment functions.

```
// Read message from the Queue and Deserialize
ExperimentRequestMessage msg = JsonConvert.DeserializeObject<ExperimentRequestMessage>(message.AsString);
```

Figure 4: Deserialization function

C. Download Input Image

The experiment aims to demonstrate flexibility with how the input image would be accessed. Considering this, the experiment allows the user to submit images from virtually any image hosting sites given the valid URI was provided as input via the trigger message.

```
// Use Webclient to accept Image from any Image Hosting Platform
WebClient myWebClient = new WebClient();
Uri imageUrl = new Uri(fileToDownload);

// Save the image file to local directory
myWebClient.DownloadFile(imageUrl, localFilePath);
```

Figure 5: WebClient function

D. Save a copy of input image to Blob Storage

A copy of the input image would be uploaded to the training container; *seivxgroup-training-files*, as a blob for reference purposes.

```
// Store the sample image in Blobstorage
string trainingImageUri = await this.storageProvider.UploadTestFileAsync(TestcontainerName, localFilePath, localfilename);
```

Figure 6: Upload file to Blob Storage

E. Download input image as a Blob

This step simply demonstrates the download function from the training container; *seivxgroup-training-files, with the DownloadInputFileAsync* method.

```
// Download the blob from BlobStorage for the experiment
string inputImageUri = await this.storageProvider.DownloadInputFileAsync(TestcontainerName, downloadedFilePath, localfilename);
```

Figure 7: Downloading Blob for Experiment

F. Run the Image Binarizer Experiment

The downloaded blob is being fed as an argument to the instance of the image binarizer module; *ImageBinarizerTest* with the aid of the *Run method* which defines other key parameters such as the experiment start time and stop time.

```
public Task<ExperimentResult> Run(string experimentImageFile)
{

    ExperimentResult res = new ExperimentResult(this.config.GroupId, Guid.NewGuid().ToString());

    // Recording start time
    res.StartTimeUtc = DateTime.UtcNow;
    Console.WriteLine("Image Binarization Experiment Started...\n-------------\n");

    // Reference to the SE Project Experiment
    Unitest ImageBinarizerTest = new Unitest();
    resultFilePath = ImageBinarizerTest.TestMethod1(experimentImageFile);

    //  Recording stop time
    res.EndTimeUtc = DateTime.UtcNow;
    Console.WriteLine("Image Binarization Experiment Finished...\n-------------\n");

    //UpdateExperimentResult(res, res.StartTimeUtc, res.EndTimeUtc, experimentImageFile);
    return Task.FromResult<ExperimentResult>(res);
}
```

Figure 8: Reference to the Image Binarizer Method

The *ImageBinarizerTest* applies threshold values to the input image using the fundamental algorithm defined by *ImageBinarizerLib.1.0.0* and *LearningApi 1.3.1* packages to convert the pixel image to its binary counterpart. The final binary image, the output of the binarization process, would be returned as a text file temporarily stored in the test device before being uploaded to the blob storage.

```
string resultFilePath = imageLocalPath.Replace(".png", ".txt");

using (StreamWriter writer = File.CreateText(resultFilePath))
{
    writer.Write(stringArray.ToString());
}

return resultFilePath;
```

Figure 9: Save the Output to a file path

G. Upload Results to the Blob Storage

The experiment result which comprises of the specific experiment ID, the description of the experiment, the URI of the input image, name of the binary image (output) file, and URI of the result blob (output) would be uploaded to the table storage; *seivxgroupresults*

```
// Upload result file and update ExperimentResult object
experiment.RowKey = msg.ExperimentID;
experiment.ExperimentID = msg.ExperimentID;
experiment.Name = msg.Name;
experiment.Description = msg.Description;
experiment.TrainingImageUrl = msg.InputFile;
experiment.InputBlobUri = inputImageUri;
experiment.ResultFile = resultFilename;
experiment.ResultBlobUri = await this.storageProvider.UploadResultFileAsync(config.ResultContainer, resultFilePath, resultFilename);
this.logger?.LogInformation("Finished uploading Image Binarization result");
```

Figure 10: Upload Result to Blob Storage

IV. RESULTS

Each successful experiment could be referenced by looking up the blob files of the input image and corresponding result in their respective containers alongside the table storage for the detailed experiment result. In addition to looking up the blob files, the source code was well documented to log every successful step even as it switches to the next step. This helps to monitor the program execution and follow through the different stages of the experiment.

Figure 11: Input Image

```
... Created Azure Queue seivxgroup-trigger-queue
info: Train.Console[0]
      Received the message {
"ExperimentID" : "EX1",
"InputFile" : "https://cdn.pixabay.com/photo/2020/09/17/13/59/cat-5579221_960_720.jpg",
"Name" : "Arinze_Okpagu",
"Description" : "Black_Cat",
}
```

Figure 12: Experiment - Receiving Trigger Message

```
Uploading image as blob: "image-e7dbe501-fee0-4e29-8325-545d2d83aa0a.png"
(URI: https://blobxstorage.blob.core.windows.net/seivxgroup-training-files/image-e7dbe501-fee0-4e29-8325-545d2d83aa0a.png)
-------------


Downloading input blob:"image-e7dbe501-fee0-4e29-8325-545d2d83aa0a.png"
(URI: https://blobxstorage.blob.core.windows.net/seivxgroup-training-files/image-e7dbe501-fee0-4e29-8325-545d2d83aa0a.png)
-------------
```

Figure 13: Experiment - Uploading and Downloading blob

```
Image Binarization Experiment Started...
-------------

Image Binarization Experiment Finished...
-------------


Uploading the binarized image as a blob: "image-e7dbe501-fee0-4e29-8325-545d2d83aa0a.txt"
(URI: https://blobxstorage.blob.core.windows.net/seivxgroup-result-files/image-e7dbe501-fee0-4e29-8325-545d2d83aa0a.txt)
```

Figure 14: Experiment - Image Binarization

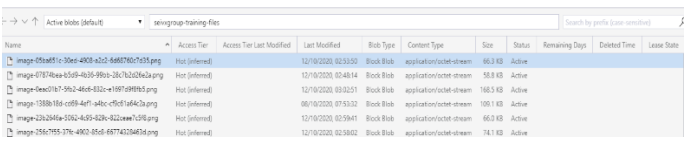

Figure 15: Training Container (seivxgroup-training-files)

Figure 16: Binary Image of the Black Cat (Result Blob)

V. Conclusion

The purpose of the experiment to couple the existing implementation of the image binarization application with cloud computing functionality was clearly demonstrated. The source code could do better to allow experimenters to choose their threshold value and image height dimensions for each experiment on the fly. Other possible modifications to this implementation could be to expose same image to varying segmentation features and as such give the experimenter an option to select the most satisfactory of the options.

VI. References

[1] M.Wirth, "https://craftofcoding.wordpress.com/," February 2017. [Online]. Available: https://craftofcoding.wordpress.com/tag/image-binarization/. [Accessed 20 March 2020].

[2] K. Nojman, "SPYTHESKY," 6 April 2019. [Online]. Available: https://spythesky.com/why-do-i-need-azure-storage/. [Accessed 20 September 2020].

[3] Microsoft, "Microsoft Docs," 17 January 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview. [Accessed 10 August 2020].

[4] Microsoft, "Microsoft Docs," 24 June 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction. [Accessed 10 August 2020].

[5] Microsoft, "Microsoft Docs," 20 May 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/cosmos-db/table-storage-overview. [Accessed 10 August 2020].

[6] Microsoft, "Microsoft Docs," 18 March 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/queues/storage-queues-introduction. [Accessed 8 August 2020].

[7] LearningApi, "Image Binarizer," Frankfurt University of Applied Sciences, Frankfurt.

[8] D. Dobric, "GitHub," July 2020. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2019-2020/blob/master/Source/MyCloudProjectSample/MyCloudProject.sln. [Accessed 20 July 2020].