Information Technology Course
Module Cloud Computing
by Damir Dobric / Andreas Pech

Implementation of Scalar Encoder in Microsoft Azure Cloud

Komala Balakrishna
komala.balakrishna@stud.fra-uas.de

Prajwal Nagaraja
prajwal.nagaraja@stud.fra-uas.de

Sahana Prasad
sahana.prasad@stud.fra-uas.de

***Abstract*— Cloud Computing provides access to data and applications from anywhere and at any time. It acts as a virtual environment for deploying and maintaining the application. Cloud computing has gained immense publicity in the IT world and it is predicted to be the next big thing in the world of computer after internet. Hence this project aims at deploying the Scalar encoder to cloud with the help of Microsoft Azure Cloud.**

Keywords- Scalar Encoder, Microsoft Azure, cloud computing.

I. INTRODUCTION

Over the past decade there is an immense growth in adopting cloud computing at both consumer and enterprise level. Basically, cloud computing is nothing but delivering the services over the internet which includes services like storage, database, servers, networking, software, and intelligence. The benefits of this is lower operating cost, higher speed, productivity, reliability, performance, and security. Microsoft is the leading provider of cloud services for all sizes of businesses. In this cloud computing project Microsoft Azure is used to deploy our previous software engineering code which is Scalar encoder to cloud.

Overview of Scalar Encoder: Scalar encoder is a part of Hierarchical Temporal Memory (HTM) systems used in solving problems like classification, prediction, and anomaly detection for a wide range of datatypes. HTM provides a very flexible and biologically precise framework. The input for HTM systems should be in the form of Sparse Distributed Representations (SDRs). SDR consists of a large array of bits in which most of the bits are zeros and few bits are ones. The encoders are used to covert the data input into SDR in HTM systems. The encoders also decide which bits should be ones and which bits should be zeros in the output for any given input value in such a way that it captures the important semantic characteristics of data. Hence if the inputs are alike, then output is more likely to have highly overlapping SDRs.

II. PREREQUISITES FOR CLOUD

The important concepts in cloud which needs to be understood for implementing any project are as follows:

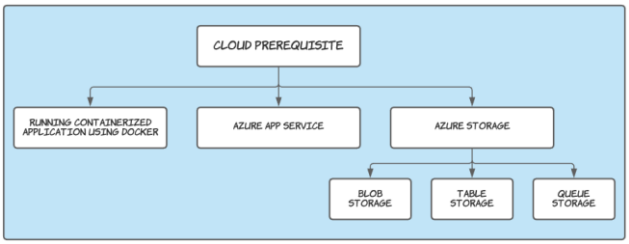

Fig.2.1. Cloud Prerequisite

a) Running containerized application using Docker: Containerizing an application involves deploying and running of distributed applications without launching the entire application environment for each app. This effectively reduces the cost and time as configuration of hardware, installation of software and operating systems is not required for the deployment.
To run this containerized apps docker is used. The docker app runs with the help of docker image where docker image is an environment which contains application code and the environment in which the code is executed. This docker image is finally stored in Azure Container Registry and this container can be executed using Azure Container Instances.

b) Azure App Service: Azure app service is the platform for hosting the web application. The focus is to design and build the application while infrastructure to run and scale the application is taken care by Azure. When compared to traditional hosting options app service helps to reduce the time and effort spent in running and managing the web app and provides excellent features in cloud which includes auto scaling and git integration.

c) Azure Storage: Azure storage acts as a platform for modern data storage which is a cloud storage by Microsoft. The azure storage is highly reliable, and feasible, secure, adaptable and it can be accessed from any part of the world over HTTP or HTTPS. The core azure storage services include blobs, files, queues, tables, and disks. In this project we mainly deal with three types of storage which are blobs, queues, and tables.

i. Blob Storage: It is an object storage solution by Microsoft which stores large amount of unstructured data where unstructured data means the data which does not follow any particular data model or definition. Blob is created to serve images or documents directly to browser, saving, files for distributed access, audio, and video streaming, to write log files and storing data for backup, restore and analysis. Blob storage is advantageous as its cost is low, highly available, and consistent and is capable of disaster recovery. The resources offered by blob storage are storage account, container, and blob. The relationship between the resources is shown in fig.2.2. For our data, the unique namespace in azure is provided by the storage account. The set of blobs is organised by the container which is similar to dictionary in the file system. Blobs might be audio, images and other multimedia objects even binary executable code is stored as blob sometimes.

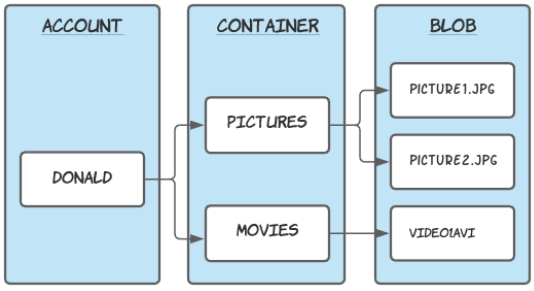

Fig.2.2. Blob Storage

ii. Table Storage: It is used to store the structured NoSQL data in the cloud where the design is schema less. Adapting the table storage to our data is easy as it is schema less. It is used for storing flexible datasets for web applications and querying huge structured, non-relational data. Accessing the table storage data is fast and it is also cost effective for many types of application. The table storage contains storage account, table, and entity.
The storage account is used to access the azure storage. Collection of entities forms a table and entity is set of properties which are similar database row.

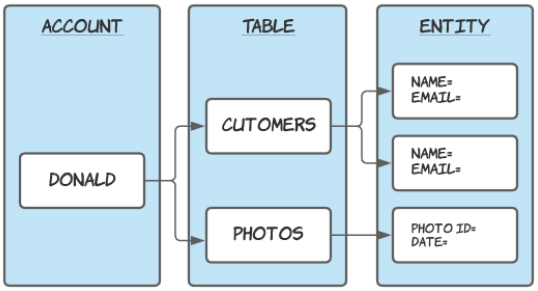

Fig.2.3. Table Storage

iii. Queue Storage: It is used to store large number of messages which can be accessed from any part of the world with the help of authentication using HTTP or HTTPS. To create the work backlog and to process asynchronously the queue storage is used. The queue storage contains storage account, queue, and message.
The storage account is used to access the azure storage. Queue consists of set of messages and all the messages must be in a queue. The message can remain in queue for maximum of seven days.

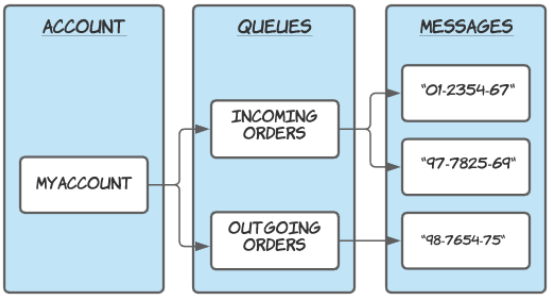

Fig.2.4. Queue Storage

III. CLOUD IMPLEMENTATION

The general architecture of cloud is shown below in the fig.3.1. The steps involved while deploying the Scalar encoder to cloud is are as follows:

1) The docker image is produced for the scalar encoder code. This docker image is stored in azure container registry and using the azure container instances the container is deployed.
2) The input data is uploaded as a file by the user to the blob storage.
3) To trigger the process a message is sent by the user.
4) After receiving the message in the queue, the code is triggered, as a result it downloads and extracts the file.
5) Scalar encoder test cases will use the data from the downloaded file and then it starts executing.
6) After the execution of scalar encoder test cases the obtained results will be uploaded to the Blob and Table

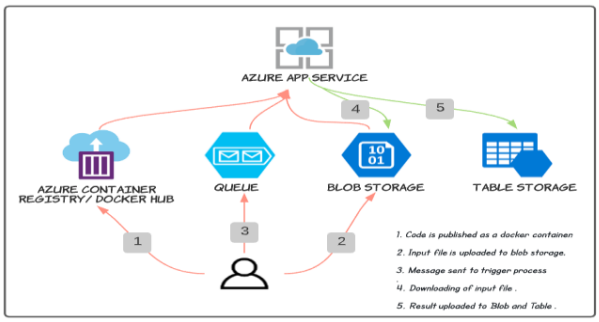

Fig.3.1. Cloud Architecture

IV. CODE DESCRIPTION

This project and their code solutions are based on the class Expriment.cs. When this program is run then it will be called by the main Program.cs file.

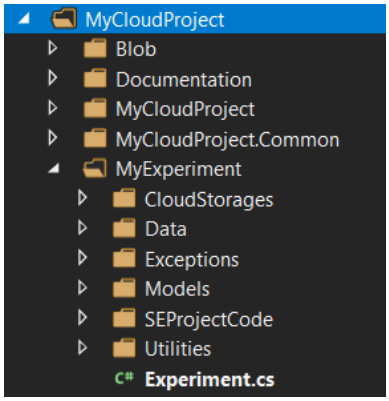

Fig. 4.1. Location of Experiment.cs class

The main aim of this Experiment.cs is to create a local folder in desktop path where the files will be first downloaded, and then when the files are executed on Visual Studio, then they are stored in azure within blob and Table. The Objects that are being taken in this *Experiment.*cs code is unchanged and needs to be used throughout. They are:

- StorageProvider
- Logger(Log)
- configSection

In the Fig. 4.2 Input data.json file is uploaded as a file to blob and in Fig.4.3 Message is sent to the queue by user to trigger the process.

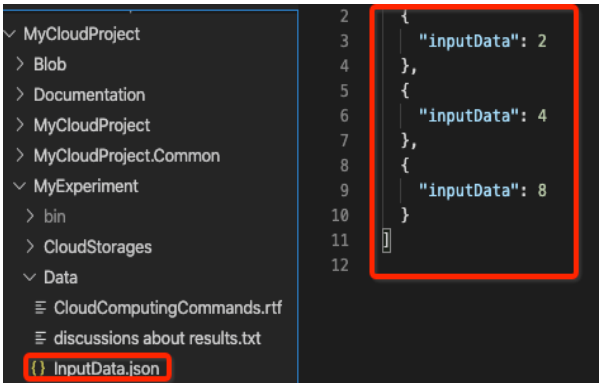

Fig. 4.2. Input data.json file

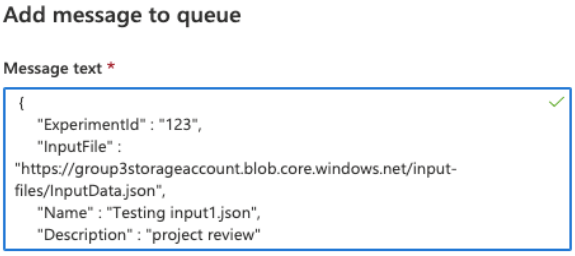

Fig. 4.3. Message given to trigger Scalar Encoder Test Cases

Here configSection is used with IConfigurationSection configuration and InitHelpers class to retrieve the details from appconfig.Json file. Once the Constructor is created then RunQueueListener method is created after that. This basically aims at listening the queue messages from the program that is pipelined next. Below figure shows how the input file is being triggered when code is run on Visual studio console and generate the output file:

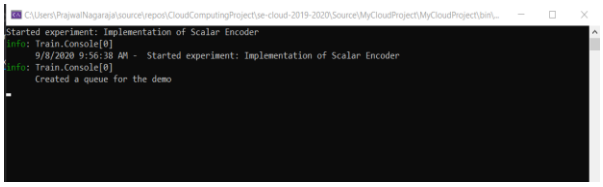

Fig. 4.4. Queue is created and waiting for message from user to trigger the process

```
public void MyFunction()
{
/// STEP 2. Downloading the input file from the blob storage

var fileToDownload = experimentRequestMessage.InputFile;
var localStorageFilePath = await
storageProvider.DownloadInputFile(fileToDownload);

logger?.LogInformation(
$"File download successful. Downloaded file link:
{localStorageFilePath}");


///STEP 3. Running SE experiment with inputs from the input file
var result = await Run(localStorageFilePath);
result.Description = experimentRequestMessage.Description;
var resultAsByte =
Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(result));
}
```

Fig. 4.5. Code to Download and run the input file from blob storage

The Fig. 4.5. demonstrates working of method where the arguments in the method receives a token that should be cancelled while executing with *CancellationToken*. A queue is created from storage account of user.

Step I: The input that is being triggered by queue is executed and read (Fig. 4.4.). The input is then deserialized or converted from byte stream to object memory. The log is then logged on the console which shows queue messages from the input file that was provided such as name, description etc. (Fig. 4.3).

Step II: The name of the input file that is present in queue (in this case 2,4 and 8) is first downloaded from blob and then the local file that is returned, is stored in local path. The method *DownloadInputFile* in class *AzureBlobOperations.cs* is used to download input file from path. The location of the class is shown in Fig. 4.6. Then the filename is passed to arguments for downloading.

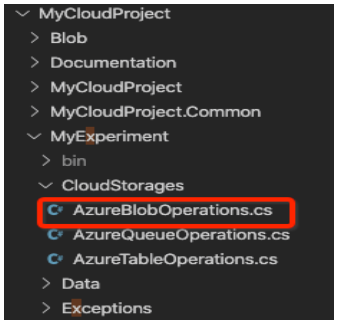

Fig. 4.6. Location of *AzureBlobOperations.cs* class

Step III: The program is run with input file that is present in local repository, for further processing. The path of the file that is stored needs to be mentioned here. The result that is obtained here from *Run* after program is debugged, is serialized or again converted into byte stream. The location (Fig. 4.7.) of the method *'Run'* and code (Fig. 4.8.) is given below:

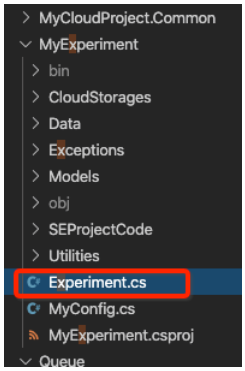

Fig. 4.7. Location of 'Run' method

```
public void MyFunction()
{
        /// <summary>
        /// Runs the project via data locally and update record of
results received in addition
        /// to time stamps and duration
        /// </summary>
        /// <param name="localFileName"></param>
        /// <returns></returns>
    public async Task<ExperimentResult> Run(string localFileName)
    {
        var seProjectInputDataList =

JsonConvert.DeserializeObject<List<SeProjectInputDataModel>>(File
Utilities.ReadFile(localFileName));

        var startTime = DateTime.UtcNow;

            /// running until the input ends
        string uploadedDataURI = await
RunSoftwareEngineeringExperiment(seProjectInputDataList);

            /// Added delay
        Thread.Sleep(5000);
        var endTime = DateTime.UtcNow;

        logger?.LogInformation(
          $"Ran all test cases as per input from the blob storage");

long duration = endTime.Subtract(startTime).Seconds;
        var res = new ExperimentResult(this.config.GroupId,
Guid.NewGuid().ToString());
        UpdateExperimentResult(res, startTime, endTime, duration,
localFileName, uploadedDataURI);
return res;
            }
}
```

Fig. 4.8. The method 'Run' in Experiment.cs

When *Run* method is executed it first deserializes the data from the output file that is downloaded. After that it runs software engineering experiment using the method *RunSoftwareEngineeringExperiment.*

RunSoftwareEngineeringExperiment shows the steps where the test cases are processed by pointing to *ScalarEncoderTests*. Then a file is created to output the results into another file.

```
public voiud MyFunction()
{
 // Step 2: Uploading output to blob storage
            var uploadedUri =
                await
storageProvider.UploadResultFile("EncodedOutput.txt",
                  null);
            logger?.LogInformation(
              $"Test cases output file uploaded successful. Blob URL:
{Encoding.ASCII.GetString(uploadedUri)}");

        // return a string and delete the combined file if possible
        return "Project completed";
}
```

Fig. 4.9. Uploading the output to blob storage

In Fig. 4.9, it shows the code for how to output the test cases. For example, one of the test cases is shown above. The output file is uploaded to blob in azure account using the method *UploadResultFile*.

Step IV: When this program is executed then the result is uploaded to blob storage having a method of *UploadResultFile* in *AzureBlobOperations.cs* class. The file is later uploaded to blob storage where URI of the file (in blob) is returned. The Fig. 4.10. Demonstrates the running of the code and uploading the output result from local path to azure storage account:

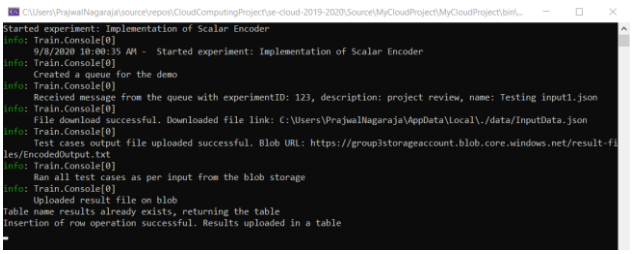

Fig. 4.10. Result obtained on the console for uploading the output file on blob and table

Step V: The obtained result is uploaded to blob storage by the means of *UploadExperimentResult* method in *AzureBlobOperations.cs* class. The result is inserted (if not already exists) into the Table created by the method *CreateTableAsync*. Once the resultant file is obtained then it is being named as *EncodedOutput*. The output of file that is uploaded to blob storage from local path is shown below:

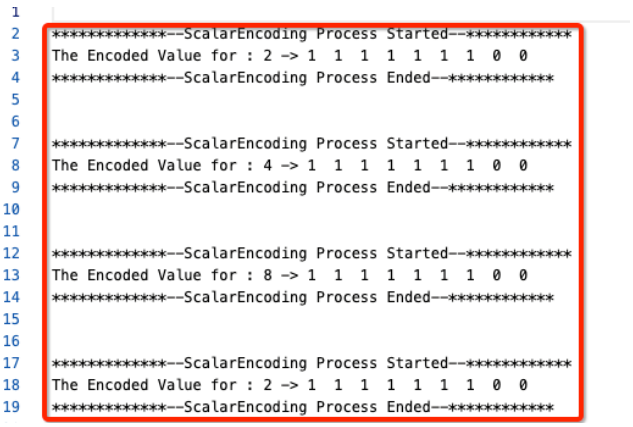

Fig. 4.11. Output of file that is uploaded to blob storage

Step VI: The result is inserted (if not already exists) into the Table created by the method *CreateTableAsync. Similar to blob storage,* the resultant file that is obtained is being named as *EncodedOutput.* Both blob storage and Table output files are saved in same resultant file.

Group3	17a392aa-0113-450e-958b-f1366d18e7d7	Sat, 29 Aug 2020 23:27:55 GMT	project review	0000000000000000005	Sat, 29 Aug 2020 23:27:51 GMT
Group3	4cda287c-9155-4dfd-8124-14c40b4f1362	Mon, 31 Aug 2020 12:18:12 GMT	project review	0000000000000000006	Mon, 31 Aug 2020 12:18:07 GMT
Group3	294b0aee-08c1-4c6b-945c-c9a5b5a348ec	Mon, 31 Aug 2020 12:25:55 GMT	project review	0000000000000000005	Mon, 31 Aug 2020 12:25:52 GMT
Group3	5ab86c8b-3391-4611-b470-f1407eb010f0	Tue, 08 Sep 2020 08:01:01 GMT	project review	0000000000000000007	Tue, 08 Sep 2020 08:00:54 GMT
Group3	8f6a786a-c6af-411d-ae54-459f51faf392	Sat, 12 Sep 2020 09:41:22 GMT	project review	0000000000000000006	Sat, 12 Sep 2020 09:41:18 GMT
Group3	0506a254-36af-400d-9b87-6752c38b2104	Sat, 12 Sep 2020 09:48:38 GMT	project review	0000000000000000005	Sat, 12 Sep 2020 09:48:36 GMT
Group3	84cea6a2-1e85-4799-ba29-e4a5eac58ed1	Sat, 12 Sep 2020 09:56:28 GMT	project review	0000000000000000005	Sat, 12 Sep 2020 09:56:26 GMT
Group3	a98a720a-c441-455d-a166-b264b4fbba47	Sat, 12 Sep 2020 21:45:57 GMT	project review	0000000000000000006	Sat, 12 Sep 2020 21:45:52 GMT

Fig. 4.12. Output of file that is uploaded to table storage

V. CONCLUSION

Cloud Computing provides access to data as well as applications from anywhere in the world and at any time. It acts as a virtual environment to deploy and maintain the application. The important concepts in cloud which needs to be understood for implementing any project are as follows:

- Running containerized application using Docker
- Azure App Service
- Queue Storage
- Blob Storage
- Table Storage

The input file is uploaded as a zip file by the user to the blob storage where the Scalar encoder test cases will use data from the downloaded file and then it starts executing. After executing scalar encoder test cases, the obtained results will be uploaded to the Blob and Table storage.

VI. REFERENCES

[1] Purdy, Scott. (2016). Encoding Data for HTM systems https://www.researchgate.net/publication/301844094_Encoding_Data_for_HTM_Systems

[2] Hawkins, J. & Ahmad, S. (2015). Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. arXiv, 1511.00083. Neurons and Cognition; Artificial Intelligence. Retrieved from http://arxiv.org/abs/1511.00083

[3] Ahmad, S., & Hawkins, J. (2016). How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites. arXiv, 1601.00720. Neurons and Cognition; Artificial Intelligence. Retrieved from http://arxiv.org/abs/1601.00720

[4] Webber, F. D. S. (2015). Semantic Folding Theory And its Application in Semantic Fingerprinting, 57. Artificial Intelligence; Computation and Language; Neurons and Cognition. Retrieved from http://arxiv.org/abs/1511.08855

[5] Abu Zneit, Rushdi & Alqadi, Ziad & Mohammad, Abu & Zalatas, (2017). A Methodology to Create a Fingerprint for RGB Color Image. IJCSMC, Vol. 6, Issue. 1, January 2017, pg.197 – 204 Procedural Analysis of Procedural Analysis of RGB Color Image O. 61. 205-212. https://www.researchgate.net/publication/313199892_A_Methodology_to_Create_a_Fingerprint_for_RGB_Color_Image

[6] Vogl, Richard & Widmer, Gerhard & Knees, Peter. (2018). Towards multi-instrument drum transcription. https://www.researchgate.net/publication/325841542_Towards_multi-instrument_drum_transcription

[7] Power Consumption Excel Sheet https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2019-2020/tree/Group3/Source/HTM/UnitTestsProject/EncoderTests/Documentation

[8] Azure app service: https://docs.microsoft.com/en-us/learn/modules/host-a-web-app-with-azure-app-service/

[9] Blob storage: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview

[10] Table storage: https://docs.microsoft.com/en-us/azure/cosmos-db/tutorial-develop-table-dotnet

[11] Queue storage: https://docs.microsoft.com/en-us/azure/storage/queues/storage-dotnet-how-to-use-queues?tabs=dotnet

[12] Deploy and run and containerized web app with Azure App Services https://docs.microsoft.com/en us/learn/modules/deploy-run-container-app-service/

[13] Architect storage infrastructure in Azure https://docs.microsoft.com/enus/learn/paths/architect-storage-infrastructure/