IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 April 2023, accepted 22 April 2023, date of publication 25 April 2023, date of current version 1 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3270368

RESEARCH ARTICLE

Data Lake Architecture for Storing and Transforming Web Server Access Log Files

ELISABETA ZAGAN[1], (Member, IEEE), AND MIRELA DANUBIANU[1,2], (Member, IEEE)
[1]Faculty of Electrical Engineering and Computer Science, Ştefan cel Mare University of Suceava, 720229 Suceava, Romania
[2]Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Ştefan cel Mare University of Suceava, 720229 Suceava, Romania

Corresponding authors: Elisabeta Zagan (elisabeta.b@gmail.com) and Mirela Danubianu (mirela.danubianu@usm.ro)

This work was supported by project ANTREPRENORDOC, in the framework of Human Resources Development Operational Programme 2014–2020, financed by European Social Fund under Contract 36355/23.05.2019 HRD OP /380/6/13–SMIS Code: 123847.

ABSTRACT Web server access log files are text files containing important data about server activities, client requests addressed to a server, server responses, etc. Large-scale analysis of these data can contribute to various improvements in different areas of interest. The main problem lies in storing these files in their raw form, over long time, to allow analysis processes to be run at any time enabling information to be extracted as foundation for high quality decisions. Our research focuses on offering an economical, secure, and high-performance solution for the storage of large amount of raw log files. Proposed system implements a Data Lake (DL) architecture in cloud using Azure Data Lake Storage Gen2 (ADLS Gen2) for extract–load–transform (ELT) pipelines. This architecture allows large volumes of data to be stored in their raw form. Afterwards they can be subjected to transformation and advanced analysis processes without the need of a structured writing scheme. The main contribution of this paper is to provide a solution that is affordable and more accessible to perform web server access log data ingestion, storage and transformation over the newest technology, Data Lake. As derivative contribution, we proposed the use of Azure Blob Trigger Function to implement the algorithm of transforming log files into parquet files leading to 90% reduction in storage space compared to their original size. That means much lower storage costs than if they had been stored as log files. A hierarchical data storage model has also been proposed for shared access to data over different layers in the DL architecture, on top of which Data Lifecycle Management (DLM) rules have been proposed for storage cost efficiency. We proposed ingesting log files into a Data Lake deployed in cloud due to ease of deployment and low storage costs. The aim is to maintain this data in the long term, to be used in future advanced analytics processes by cross-referencing with other organizational or external data. That could bring important benefits. While the proposed solution is explicitly based on ADLS Gen2, it represents an important benchmark in approaching a cloud DL solution offered by any other vendor.

INDEX TERMS Cloud data lake, ADLS Gen2, data lake architecture, web server access log data, Azure function Blob trigger.

I. INTRODUCTION

The Internet of Things (IoT) paradigm become a revolutionary transformative element in our knowledge and interaction with our environment. It involves the development of diverse applications and solutions in different domains such as smart homes, smart cities, digital health surveillance, industrial processes, etc. that need to be constantly monitored and analyzed. A good indicator of the performances of a system are log files from the network, servers, security devices, services, applications etc. Large-scale analysis of these logs brings a great deal of added insight into the different business domain. In [1] the authors have provided a systematic review of recent literature on the different types of log files that are used in research area. The importance of these log files is proven in all fields of activity, thus, nowadays,

The associate editor coordinating the review of this manuscript and approving it for publication was Giacomo Fiumara.

identifying a powerful storage and analysis environment is imperative.

Web server log files store all events that occur as a result of requests issued by online users accessing information available on a server. Recording these events can lead to the identification of user visiting behavior [2] over certain periods of time, which can lead to a better understanding of customer requests. Companies can then make smart, personalized decisions to improve user experience.

Servers keep these records in different log files, namely, web server access log files, which we hereafter refer to as Web Server Access Logs (WSAL) files. By analyzing these log files, it is possible to discover ''visiting behaviors'', which can lead to major improvements and allow the services offered to users to be perfectly customized.

Existing problems such as WSALs are difficult to store, process and analyze at large scale, so we proposed storing them in a DL architecture. The analysis of WSAL files together with the DL design methodology, respectively the procedure of transforming WSAL files into parquet files using the azure blob trigger function written in python are a novelty in the field and are the main contribution of this paper. The derivative contributions resulting from this research are both theoretical and practical, since the processes of data ingestion, transformation and data preparation for the next analysis step are implemented and well described. Also, based on the theoretical considerations we described in paper [3], in the present paper we are proposing also a hierarchical structuring model for the organization of WSAL data in order to serve the ELT pipelines, specific to the DL architecture.

The emergence of new types of data, both unstructured and semi-structured, from web environments, social media, comments, servers, sensors and various devices has made traditional storage methods obsolete. An example might be that 15 to 20 years ago it was not foreseeable that in the near future the record of 'likes' on different social media would be so important although they can provide vital information as they are direct feedback from users in the virtual environment.

Another example is web server access logs, which we will refer to throughout the paper as WSAL. Web servers are accessed by millions of users every day. Users leave behind their visiting behavior by recording their activities online in log files. Servers keep these records in different files, such as access logs, error logs, piped logs, script logs, etc. Log files are often automatically deleted from servers due to their constant increase in volume. There are various techniques for analyzing WSAL files on the market [4], [5], but the main problem lies in storing them in their raw form in order to allow various analysis processes to be run in the future enabling information to be extracted for different purposes. Transforming this logs into structured data for later storage, using traditional methods, would involve high storage costs. As a consequence, in recent years, the explosion of these new data types has led to the emergence and development of new concepts, technologies and techniques for data management. Data Lake (DL) is a revolutionary concept that has been in the spotlight recently. DL, as the name suggests, can be seen as a structure where absolutely all data can be stored in its raw form [6].

The **research gap** could be filled by implementing a generally valid model of hierarchical data storage with distribution of data to different areas. DLM techniques for cost optimization, data security mechanisms, data redundancy aspects, built-in DL data processing techniques are presented. Several important technical aspects have been also highlighted and should be considered when building a data lake in the cloud. Thus, in this article we presented the design and implementation of a DL architecture in cloud for WSAL files long-term storage. WSAL files, contain large volumes of unstructured data for which a self-scalable storage environment is required.

The **research motivation** behind this project is the implementation of a DL architecture in cloud using the technology made available by Microsoft, namely Azure Data Lake Storage Gen2. ADLS Gen2 is based on Apache Hadoop and Apache YARN and is a solution that does not require the installation of any hardware or software systems on the user's side. It is a cost-effective cloud service where you pay only for the used storage space. These costs can be optimized as needed through data lifecycle management (DLM). Our paper presents an ADLS Gen2 architecture with different levels of data processing. Important technical aspects underlying the implementation of a DL in Azure are also covered, technical aspects defining each level within the DL architecture.

This paper, is a **scientific and practical contribution** to the field. An ADLS Gen2 architecture with different levels of data processing is presented. Important technical aspects defining each level within the DL architecture are presented. A generally valid model of hierarchical data storage is proposed and presented, with distribution of data across different areas. These issues aim to highlight some important features in the process of implementing a DL. Even if they are explicitly presented for the ADLS Gen2 environment, they are an important benchmark in approaching a cloud solution provided by any other vendor.

Most servers generate WSAL files that can be stored in different formats, such as Common Log Format or Combined Log Format. A study on the content of these files has led to the identification of the following categories of data [7], [8], [9]:

- Requested resource file location, name, and size;
- Request time and date;
- Request method;
- Identity of client device that made the request to the server;
- Client browser information;
- Page from which the client left the site;
- Server response to client request;
- Number of pages visited by client;
- What search engine was used and search terms used;

- Whether it was a direct access by a user or a redirect from another site, or part of an advertising campaign.

WSAL files are raw data files that are difficult to understand without a proper log parsing tool. Another major problem is storing these files, because they need a lot of storage space. In a short period of time, depending on the frequency with which the servers are accessed, huge numbers of data can be generated and stored on them, which is why they are automatically deleted after a period or after they exceed a certain size [10]. It is estimated that 80% [11] of all companies' data is lost, because organizations cannot keep up with the volume, speed, and variety of data. Inside companies' it is being estimated that 80% of data is unstructured. On average, between 60% and 73% of all this data is never analyzed [12]. This is so-called auxiliary data or dark data, collected from various sources like data from sensors, statistics on processes, social media, monitoring services, log files, raw survey data, audio, and video, etc. [13]. There is a large percentage of lost unstructured data not being analyzed inside a corporation, data from which valuable information, that could lead to multi-level improvements through advanced analysis processes, could be extracted.

The main advantages of the long-term analysis of WSAL files are the following:

- Improved website security and structure;
- Improved web server performance;
- Increased website traffic;
- Elimination of navigation errors on website;
- SEO improvement and increased search engine rankings;
- High-quality marketing strategies based on customer preferences.

Thus, it becomes a necessity for most large companies to be able to store and analyze these log files to ensure reliability, security, and performance, especially for the financial benefits. For small sites, there are already a variety of online or open-source services on the market [14] that can extract various reports based on standard analyzing techniques, but with several limitations, either related to file size or to the type of analyzing processes, which usually return specific reports and statistics. As mentioned above, the major problem lies in storing the log files on long term. One ideal candidate for storing and analyzing large volumes of unstructured data in their raw form is the latest technology: a DL with unlimited storage space with auto scaling at low costs.

In [3] a conceptual model of hierarchical structuring was developed; the process of ingestion was also discussed extensively but the process of transformation was only mentioned. The results obtained by extending the mentioned research, presented in this paper, have been materialized in the following contributions in the area of DLs and WSAL files:

- We designed and implemented a DL architecture to efficiently support storing and transforming WSAL files;
- We designed an effective hierarchical structuring model of WSAL data in the storage layer of the DL architecture. Serving in ELT processes, this model can improve the performance of data storage and access;
- We implemented a serverless Azure Function Blob trigger, written in Python, for automatically converting log files into Parquet files once a new file is uploaded to the raw DL area.

This paper begins with a brief introduction; in section II, we present the most important advantages of saving WSAL files in DLs for further performant analytics processes. Section III presents the proposed DL architecture and its physical implementation in the cloud using ADLS Gen2 technology. A hierarchical structuring of the data within the DL is also proposed to provide structured data for consumption that feed the different processes within the DL. Section IV describes the experimental results, and Section V reviews recent research in the DL domain. Section VI presents the conclusions and future research directions.

II. ADVANTAGES OF SAVING WEB SERVER ACCESS LOG FILES IN DLS

The ability to examine log files even after long periods of time becomes a necessity. Companies' do not store these logs for reasons related to hardware resources, which can grow unreasonably large and require significant storage capacities. Therefore, an optimal solution for archiving these log files for long periods of time may be to use on-premise or cloud DLs.

We identified and summarized the main advantages of saving log files in a DL, among which the following are worth highlighting:

- Since WSAL files are locally stored on servers for limited periods of time, the use of a DL offers maximum flexibility in storing data for long periods of time at affordable costs;
- Control over hierarchically structured data within the DL storage layer can be successfully managed by assigning different access roles to each area;
- Existing web analytics tools rely on Java script codes or cookies, which can cause delays in loading a web page and are more susceptible to various technical problems. If the user has scripts and cookies blocked in their personal browser, then reports from services such as Google Analytics may not be conclusive. Thus, storing log files in a DL offers the possibility to implement dedicated analysis processes to extract customized information and reports without influencing the loading of a web page and without depending on the client browser settings.

In terms of applications, the implementation proposed in this paper, based on DL technology in the cloud, is a low-cost solution with remarkable performance in terms of storing and analyzing WSAL data to obtain valuable information from web servers.

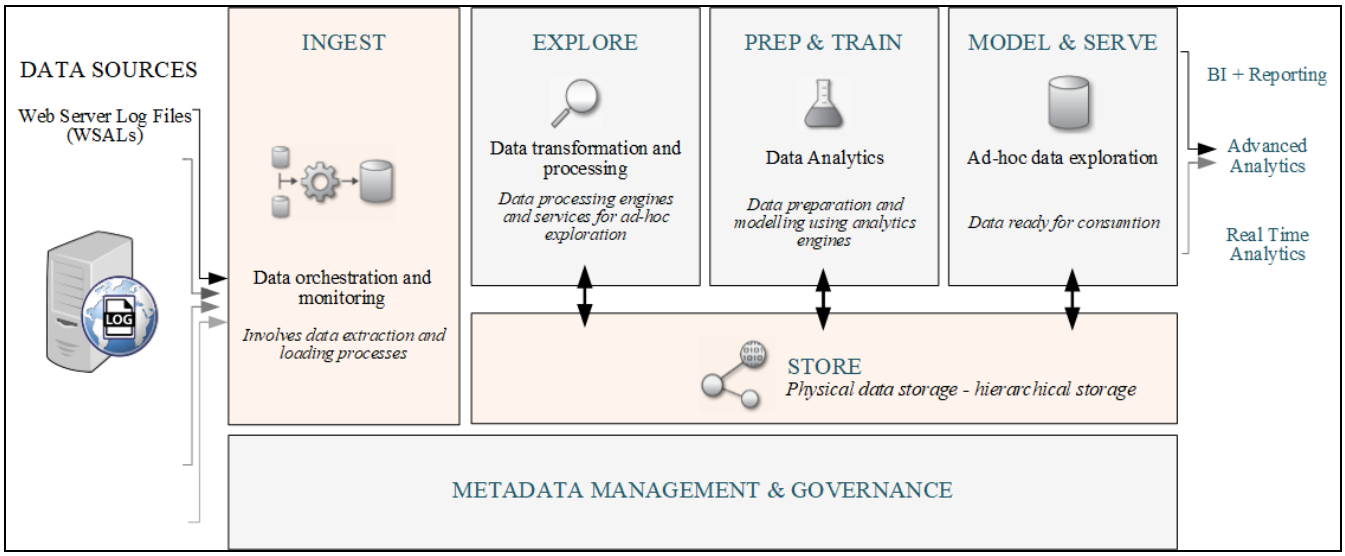


FIGURE 1. Proposed DL architecture and various ELT processes serving WSAL files.

III. DESIGNING AND IMPLEMENTING A DL FOR WEB SERVER LOG FILE ANALYSIS BASED ON ADLS GEN2 ARCHITECTURE

This paper addresses the technique of cloud DL storage of WSAL data, a technique that is more accessible without the need for hardware, software or engineering resources that are not always affordable and available to everyone and can be expensive as time consuming to implement. DL in the Cloud is a leading technology for managing large data sets with advanced real-time data analytics. With this new technology, the huge potential of previously unanalyzed data is revealed.

We present and validate an original project implemented in Azure Cloud for storing WSAL files without storage limits at low costs, so that they can then undergo advanced analysis processes. Thus, we propose a solution using ADLS Gen2 cloud storage, a service provided by Microsoft that allows the customer to create and own a DL where large volumes of data can be stored, transformed, and analyzed.

ADLS Gen2, is a hyper-scalable system which relies on archiving service to keep the cost of storing unstructured Big Data as low as possible. It includes the following capabilities.

- Hadoop-compatible access (relies heavily on the Hadoop and YARN framework)
- Hierarchical directory structure (the hierarchical namespace is a key feature that enables ADLS Gen2 to provide high-performance data access at object storage scale and price)
- Ideal for big data analytics workloads
- Optimized cost and performance (ADLS Gen2 comes with the same storage price as blob storage, which is already known to be the most economical storage system for large data volumes)
- Storage Tiers (Different Blob tiers Hot, Cool, Archive have been created to provide an efficient storage option)
- Finer grain security model
- Massive scalability (ADLS Gen 2 doesn't impose any limits on storage size)
- Azure Portal is user friendly (Azure Portal is a graphical user interface that is used to manage Azure services)
- Allows the design of Hybrid systems.

A strength of the ADLS Gen2 concept is that it allows data to be organized in a hierarchy of directories and subdirectories for efficient data access (hierarchical namespace). Until ADLS Gen2, data was stored in flat namespace mode, this hierarchy is new and comes with multiple advantages. As data volumes grow, the hierarchy allows data to be maintained in a more organized way and provides better performance for data analysis tasks.

We are proposing a DL architecture for the implementation of different processes within the ELT pipeline, storage of WSAL files, transformation and preparation of data for consumption, once they have been submitted to different analysis processes. As a result of our research, we were able to design and propose a stable, reliable and economical architecture shown in Figure 1.

As we can see, the proposed architecture consists of 5 different layers. The Store layer is the core level of the entire architecture. Above this layer there are the Explore, Prep & Train, and Model & Serve layers, which serve different transformation, enrichment, and processing steps that data undergo for valuable information to be extracted with different advanced analysis processes. Over the ingestion layer, fast processes are running to load different types of raw data from various external sources into the DL. There is no data alteration within this layer. Raw data can be ingested in real time or in batch mode. Following ingestion, the data are saved into the DL storage layer to a dedicated raw data area. Thus, the data are stored in their natural form within this area of the DL. In the exploration layer, transformation processes are performed over the stored data,

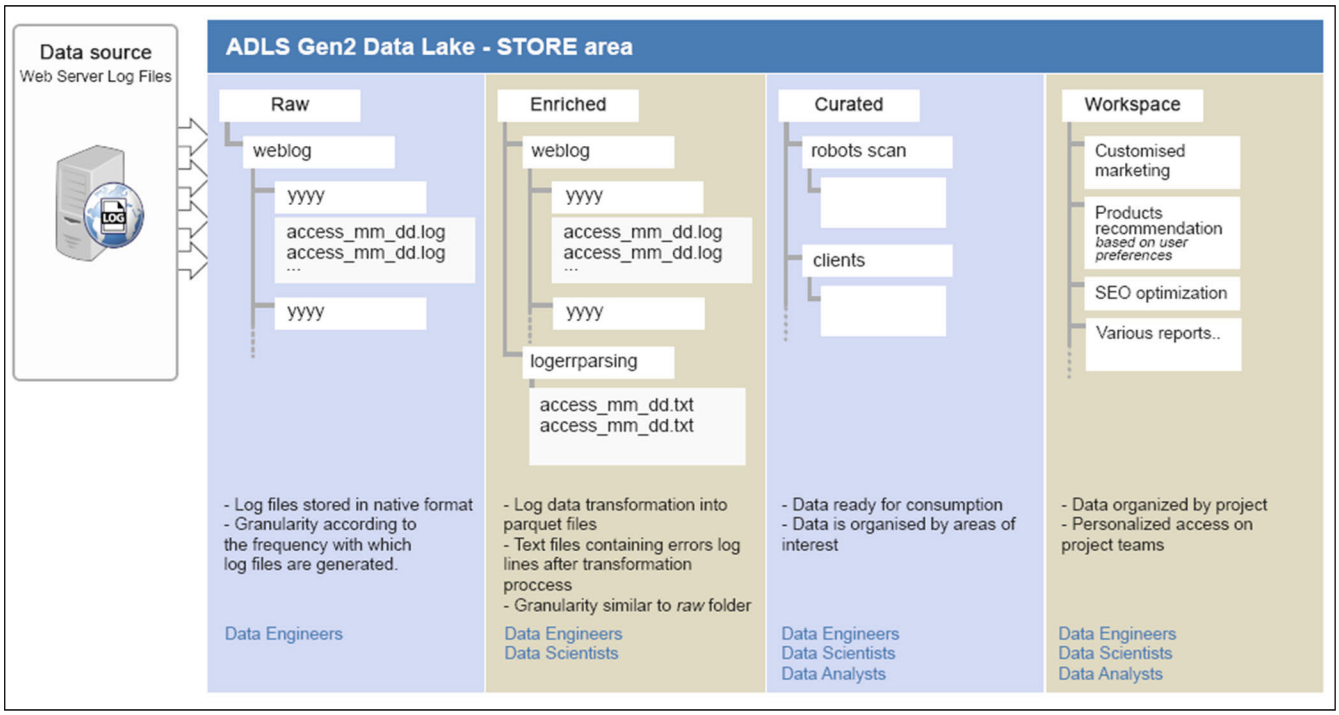


FIGURE 2. ADLS Gen2 hierarchical structuring of web server log data.

thus the data are cleaned and standardized into proper data types. This is where raw data are being transformed into more structured datasets that are being placed into well-organized directories and subdirectories into a different area in the storage layer. The Prep & Train layer of the architecture is where previously transformed data are prepared and modelled using advanced analytics engines. A new area within the DL storage layer is dedicated to the data derived out of these processes. At the Model & Serve layer, data are accessed by users who know the data and the needs of an organization. New modelling processes are carried out using appropriate analysis techniques in order to obtain information to be used for final statistics and reports. Each of these layers is served by different processes that can be implemented using either integrated technologies provided by Microsoft, such as Azure DF, Azure Databricks or through ad-hoc developed processes written in different programming languages that can be easily integrated using Azure Logics or Azure Functions.

This research focuses mainly on the ingestion layer, the storage layer and the data transformation process, within the Explore layer, leading to standardization and preparation of data for further advanced analysis processes. The data ingestion process in ADSL Gen2 consists in uploading the log files holding the raw data into the DL by implementing a batch process on the server that automatically uploads the log files to the DL at regular intervals or using various built-in technologies provided by Microsoft [15]. Once data are saved in their raw form in the cloud DL, they undergo a transformation and standardization process that transforms log files into Parquet files. This process is fully automated with the use of Azure Blob Trigger, which automatically launches into execution a Python function that transforms log files into Parquet files. Once transformed, data can be subjected to various cleaning and enrichment processes to be prepared for the final analysis process that serves the ultimate interests.

A. HIERARCHICAL DATA STRUCTURING IN THE DL

The data that are ingested into the DL must be stored in a well-organized way so that they can be easily managed. The data should not be randomly thrown into the DL Store layer, but should be organized in a structured way. Each area of this structure serves different processing layers.

In Figure 2, we show the design of a data structuring model to serve the entire ADLS Gen2 architecture throughout the various layers. Thus, into the Store layer, we have the following areas:

- The *Raw* area is the area where raw data are loaded directly from the server source, with restricted access. This area provides storage for raw data obtained during the ingestion process. This area can be organized using a hierarchy of directories, such as yyyy/mm/dd, based on the frequency with which log files are generated. In our case, we use a rough hierarchy. All ingestion processes dedicated to different data sources, in case the data comes from different servers, have write-only access to the directory associated with the data source. In our case we have only one source so we designed

under Raw folder only one dedicated directory, labeled *weblog*.

- The *Enriched* area stores data that have undergone a transformation and standardization process and have subsequently been saved in *Enriched* area as Parquet files. Thus, original raw data remain in the *Raw* area, which, to minimize storage costs, can be transferred to a cooler or archive tier for long-term archiving. In addition, in the Enriched area, there is a folder named logerrparsing, where text files containing error log lines that could not be successfully transformed are uploaded. This is the area serving the Explore layer within the proposed DL architecture.
- The *Curated* area stores data from the enriched area that have been cleaned to make them ready for exploration. This area serves the processes running on Prep & Train layer in the proposed architecture. Different cleaning methods can be applied to the same dataset to serve different analysis techniques, depending on the purpose. In this area, data undergo a change in the storage hierarchy, which can serve different purposes. In our case study we move from a hierarchy organized according to the date on which the logs were generated to a hierarchy dedicated to different segments of interest, like clients, search engine scans, system errors, etc.

The *Workspace* area is the data exploration area serving the Model & Serve layer of the proposed architecture. The data in this area are organized by project and can be accessed by data engineers, data scientists, and data analysts who understand the data and business needs. Here, the data are structured into projects with dedicated access for different teams.

Data organized hierarchically into directories and sub-directories are given different access rights as needed to maintain their integrity as recommended in Figure 2.

The innovative solution proposed in this paper, as well as the contributions of the theoretical and practical research related to it, exists at the conceptual approach level; therefore, it should be specified that the resources used are limited but validate the proposal of implementing a DL based on ADLS Gen2. It is obvious that in a production level approach, the raw area holds log data uploaded from a web server to the DL at regular intervals. Thus, it is necessary to achieve a more granular hierarchy in the raw area by creating directories and subdirectories that group the logs coming from the server based on the yyyy/mm/dd format, according to the frequency with which they are generated. To avoid bottlenecks in the ELT transformation process, it is ideal that log files are uploaded to the server once they have reached a maximum size of $1 \div 1.5$ GBytes.

B. LOG DATA TRANSFORMATION PROCESS

After loading the data into the DL, a process of transforming raw, unstructured data into structured data is automatically triggered. Thus, log files, once loaded into the DL, undergo an automatic process of transformation into Parquet files. We achieved this transformation process by developing an Azure Function Blob Trigger written in Python using the Visual Studio Code development environment. Azure Function is a serverless compute service that enables user to build and debug functions locally, deploy and operate at scale in the cloud, and integrate services using triggers and bindings. Functions can be written in different programming languages, such as compiled C#, C# script*, JavaScript, Java, PowerShell or Python. For the related research, Python was chosen to implement the transformation process from log files to parquet files due to the fact that Python libraries facilitate this process at the procedural level. We have chosen VS Code as the development environment, which enables the automatic generation of a new project to implement the Blob Trigger function using a template. A collection of directories and files are automatically generated locally. The files generated by the template can be edited and adapted to the needs of the project, as we explain below. Before creating a new project, it is important that in VS Code the connection to the Azure account to be already established, so that when the files for the new project are being generated, the Azure storage connection parameters are automatically inherited and therefore the access to the cloud resources will be granted. The project has a *local.settings.json* file containing the Azure account connection strings that include the Shared Key access keys associated to the owned storage account.

The *function.json* file defines the inputs and outputs for the main function that performs the data transformation process, therefore there are defined an input blobTrigger and two output blobs. The configuration of the input/output parameters for the Blob Trigger Function is shown in Table 1. The main function expects three parameters: one input blobTrigger and two output blobs.

TABLE 1. Defining the input/output parameters of the blob trigger transformation function.

Blob trigger transformation function	Input/output parameters
Binding 1	**myblob** *// one blob in* "name": "myblob" "type": "blobTrigger" "direction": "in" "path": "raw/{name}.log" "connection": "adlswebserver2_STORAGE"
Binding 2	**myblobout** *// first blob out* "name": "myblobout" "type": "blob", "direction": "out" "path": "enriched/{name}.parquet" "connection": "adlswebserver2_STORAGE"
Binding 3	**errmyblobout** *// second blob out* "name": "errmyblobout" "type": "blob", "direction": "out" "path": "enriched/logerrparsing/{name}.txt" "connection": "adlswebserver2_STORAGE

Algorithm 1 Blob Trigger Function to Transform Log Files Into.Parquet Files

Input: *myblob*
Output: *myblobout*,
errmyblobout

Begin main function
  regex =<define regular expression>
  columns =<define parquet columns>
  myblob =<read *myblob* bytes>
  mytxt_obj =<convert bytes to Unicode object in order to perform line by line reading>
  array_log_lines =<initialize array containing valid log lines>

(Check each line in the log file if it follows the format defined in the regex)
  For each log line
  begin
  If(check for regex format)
    Append valid log lines to array_log_lines
  else
    Append non valid log lines into ASCII text error file
  end for
   df_log_line=<put array_log_lines into a data frame>
   call **to_parquet** function to convert data frame into parquet file

(Pushing the transformed data to the output blobs)
   myblobout =<set *myblobout* with the new parquet file>
   errmyblobout =<set *errmyblobout* with the text file containing the transformation errors>
  end main function

The blobs in Table 1 are defined by the following:

- ''name'', which represents the name of the blob to which the main function refers;
- ''path'', which represents the address to the directory in the DL that serves the blob. If it is an input blob, then it represents the address to read the data from the DL; if it is an output blob, then it represents the address to write the data returned by the main function to the DL. One can also specify the type of the input or output data by specifying the log, parquet, or txt file type. For the input blob, it is important to specify the type of data expected by the main function as a.log file, in order to avoid launching the main function for files other than.log, which would lead to execution errors;
- ''direction'', where ''in/out'' defines the direction of the blob within the function, either input or output;
- ''connection'', where ''adlswebserver2_STORAGE'' represents the SK connection data to the Azure account that is saved in the *local.settings.json* file.

In the *__init__.py* file we wrote the main transformation function that automatically runs when a new file is uploaded to DL in the *raw/* directory which is being monitored by the trigger. This function performs the transformation of log files into Parquet files.

Algorithm 1 conceptually shows how to transform log files into.parquet files.

```
logline_regex = '^(?P<ClientIP>\S+) (?P<RemoteLogName>\S+) (?P<Au-
thUserName>\S+) \[(?P<TimeStamp>[^\]]+)\] "(?P<AccessMethod>[A-Z]+)
(?P<AccessRequest>.+)? HTTP/[0-9.]+" (?P<ResultStatus>[0-9]{3})
(?P<SizeBytes>[0-9]+|-) "(?P<ReferrerURL>[^"]*)"
"(?P<UserAgent>[^"]*)'
columns = ['clientip', 'remotelogname', 'authusername', 'timestamp',
'accessmethod', 'accessrequest', 'resultstatus', 'sizebytes', 're-
ferrerurl', 'useragent']
```

FIGURE 3. Regex parser to extract fields and contents from log files.

The blobTrigger file is initially unformatted, therefore it must be decoded in UTF-8 format in order to be parsed line by line during the transformation process. First we read the bytes triggered by the blob until the EOF is reached, then we decode the UTF-8 bytes read and assign them to a string object. Python bytes decode() function is used to convert bytes to string object. Once we have the string object, by using the python splitlines() function, we can now split the string into lines to check each line if it matches the regex format (Figure 3).

The lines that meet the transformation criteria are passed into a Panda DataFrame to which we subsequently apply the parquet() transformation function: df_log.to_parquet(buffer). The python to_parquet() function performs the transformation of the DataFrame into a binary Parquet format, which will be allocated to the io.BytesIO buffer in order to be pushed

into the first output blob *myblobout* and uploaded to Cloud DL in the *enriched/* folder.

The lines that do not follow the formatting in the regex are ultimately saved in an ASCII encoded.txt file and transferred into the second output blob *errmyblobout*. Thus, *myblobout* loads the new Parquet file into the *enriched/* folder, and *errmyblobout* loads the.txt file containing the error lines into *enriched/logerrparsing/* folder in DL.

IV. EXPERIMENTAL RESULTS

Once the trigger function is created, one can proceed to run it locally, and then upload a new log file to the Azure DL in the folder monitored by the trigger. In this research, the data subjected to ELT processes were downloaded from Harvard University Datasets ''Harvard Dataverse'', ''Online Shopping Store - Web Server Logs'' [16], coming from an ecommerce website. A log file of about 3.42 GB was used for testing. We split this file into smaller chunks of 1.0G in order to reduce the waiting time in the transformation process. This resizing step will not be necessary in a real environment. Loading files from the server into DL will be a batch process that will start automatically at regular intervals when a log file reaches a certain size. Upload times were relatively long for a log file of 1.00 GB, from 10 min up to 30 min. In practice, these loading times may vary depending on the ingestion technology used, the type of Azure account chosen, the communication speed in the WAN, etc.

Once the upload to the server is successfully completed, a sequence of messages appears in the VS Code terminal confirming that the trigger has detected a new file that has successfully uploaded to the DL, which triggers the execution of the function that performs the transformation process. Figure 4 shows the results obtained. We can see that after loading the file access-weblogs1.log in *raw/* DL, the logging info of the main transformation function reports that a blob of 1048576000 bytes = 1GB is to be transformed. The log file underwent a line-by-line read process for a regex check on each line. In total, 3,170,024 processed log lines were identified. Following the transformation process from a file log of 1000 MiB (1048576000 bytes) we obtained a parquet file of only 108.2 MiB (113455923.2 bytes).

The conversion time varies greatly depending on the machine on which the process is performed. In this case, for a 1.0 GB log file, the transformation time was about 76s from the time the trigger found a load at the selected directory level. In the transformation process, a delay time of about 8 ÷ 10 min can be encountered from the moment the upload finishes until the trigger confirms that a successfully uploaded file has been detected, because trigger functions rely on logs to scan for new/changed blobs; this can cause delays, which can be improved, according to [17]. It is important to emphasize that this research was performed using a student account in Azure; therefore, we had limited resources. After the transformation process, a 108.39 MiB access-weblog1.parquet Parquet file and a 102.02 KiB error access-weblog1.txt text file was obtained from the 1.0 GB log file. The file was reduced from 1.0 GB to 108.39 MiB, that is, 10.8 % of its initial size. Following the tests carried out, in Azure Portal (Figure 5), we obtained in each of the two areas, raw and enriched, the four files corresponding to each area. Based on different laboratory tests, we obtained the dataset below (Table 2).

The Parquet files containing the data transformed and saved in columnar format were greatly reduced in size compared with the initial size. In the text files, we captured log lines that could not be transformed because they did not respect the constraints imposed by the regex.

The nature of the errors should be evaluated, and the regex should be adapted in order to obtain as few error lines as possible. At a first analysis of the lines returned as errors by the regex process, they appear to be lines that, from an analytics point of view, are irrelevant and can easily be ignored. If, however, they are still of interest and must exist in the Parquet files for further analysis, then the regex in the main function must be adjusted. Thus, in this paper, the results of the Blob Trigger Function and the related results of the DL transformation are presented.

In the Figure 6, we can see by how much the data storage space can be reduced by transforming log files into Parquet files. The graph was obtained based on the values derived from the various tests, which are also reported in Table 2.

Once the log files from the raw area have been converted to Parquet files, one can then move the log files to a cooler tier for long-term archiving and for significantly reducing storage costs.

During the creation phase of a standard Azure account, one can choose between the three different types of tiers in order to minimize the costs applied to data storage, processing, and consumption. Hot tier is optimised for storing data that are accessed or changed frequently. It has the highest storage costs but the lowest access costs. Cool tier is optimised for storing data that are accessed or changed infrequently. Structuring implies lower storage costs and higher access costs than hot tier. Data in cool tier should be stored for at least 30 days. Archive tier is optimised for storing data that are rarely accessed. Data in archive tier should be stored for at least 180 days. This tier type cannot be selected as the default storage type for an Azure account. Archive tier data cannot be read unless they are first rehydrated, which is a time-consuming and potentially costly process in data management.

DL datasets have different lifecycles. For example, at the beginning of the data lifecycle, the data are accessed frequently, but the need for access often decreases drastically as the data age. Some data are accessed frequently (these are usually the most recently saved data that are at the beginning of their lifecycle), while other data remain inactive in the cloud and are rarely accessed. Some datasets expire within days or months of creation, while other datasets are actively read and modified throughout their lifetime.

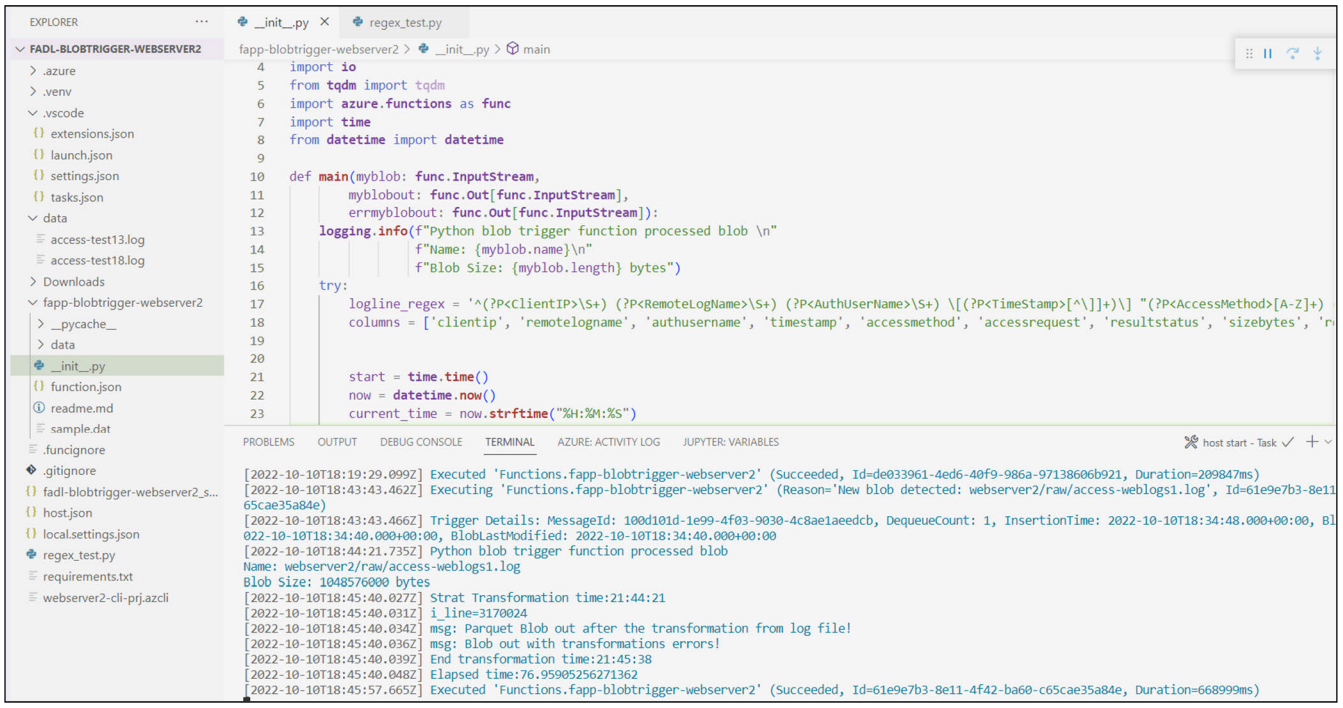


FIGURE 4. Serverless blob trigger function running in vs code.

TABLE 2. Log file transformation process based on blob trigger function.

Log File (Name)	LOG (BYTES)	LOG LINES (NUMBER OF LINES)	Transformation Time (s)	DL log (MiB)	DL parquet (MiB)	DL err txt (KiB)
access-weblog1.log	1048576000	3170024	76.95	1000.00	108.39	102.02
access-weblog2.log	1048576000	3037405	75.51	1000.00	111.2	21.99
access-weblog3.log	1048576000	3203638	107.60	1000.00	116.06	18.78
access-weblog4.log	356712823	954088	17.86	340.19	31.26	8.35
Total	3502440823	10365155	277.92	3340.19	366.91	151.14

ADLS Gen2 is a cost-effective storage environment with no upfront costs. It is a pay-per-use service, allowing users to pay only for the number of gigabytes stored and the number of transactions (reads and writes) that take place on the data. The costs differ according to the type of subscription, the type of data redundancy LRS, ZRS, GRS, RA-GRS, the selected region. Azure also provides an online calculator for the price estimation. As a guideline, for a standard account with active hierarchical and redundant LRS storage costs can start from €0.01699/GB hot tier, €0.00924/GB cool tier, €0.00092/GB archive tier for the first 50 terabytes (TB) / month. As costs per transaction for write operations (every 4MB, per 10,000): hot tier €0.06002, cool tier €0.12003, archive tier €0.12003 and read operations hot tier - €0.00481, cool tier - €0.01201, archive tier €6.00130. More about these costs can be found in [18]. Data lifecycle management in the cloud is an important process that can be achieved using Data Lifecycle Management (DLM) policies. Azure storage offers rules-based management policies that can be used to manage the right types of data access.

In [19] and [20], more information regarding DLM-based rules and how costs can be optimized through automated data lifecycle management, respectively, can be found.

The overall objective of this scientific paper is to conduct research on DL, with a focus on the level of WSAL ingestion and storage. In order to achieve this objective, the following specific objectives have been achieved:

- Deepen the current state of research on DL architectures, this new concept emerging due to technological evolution, the speed at which data is generated and its variety.
- After identifying different implementation methodologies by conducting a comparative study in terms of costs and resources required to implement a storage system, a solution for implementing a DL architecture in the Cloud was identified, which comes with a number of advantages that were specifically highlighted during this work.
- We have designed and implemented a DL architecture in ADLS Gen2 for long-term WSAL file storage.

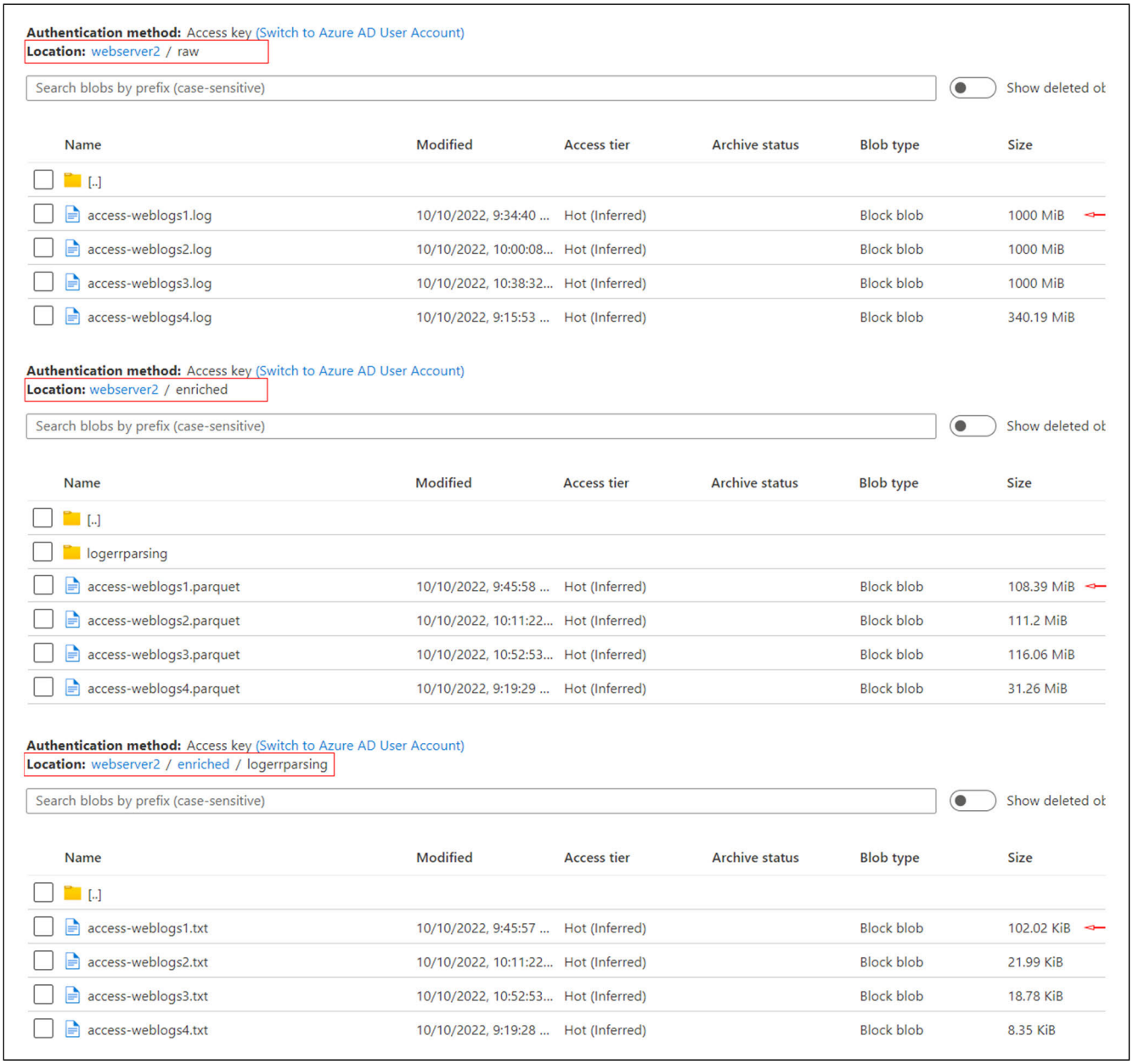
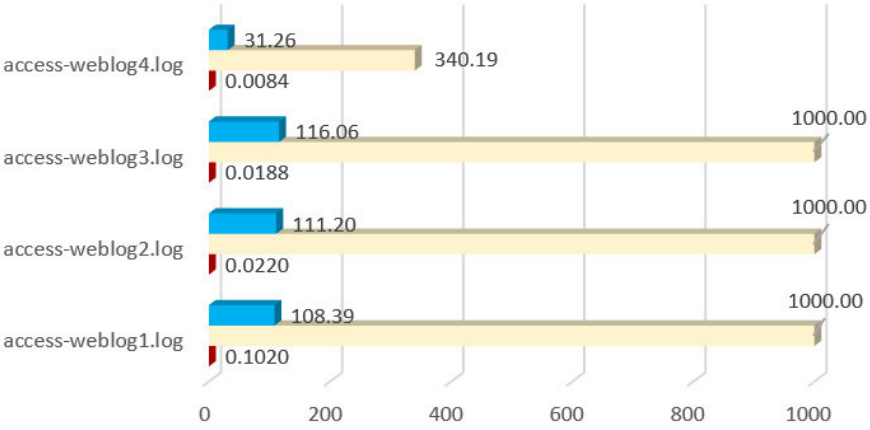


FIGURE 5. Experimental data based on blob trigger function in azure portal.

- We proposed a hierarchical storage model to serve the different levels within the DL architecture. Thus, the raw data area, following the ingestion process, stores the data in its raw form. The enriched data area stores data from the raw area that has undergone an automatic transformation and standardization process, whereby the raw unstructured data is transformed into structured data and stored in parquet files. In this area are also stored in a separate directory called logerrparsing the log lines that could not be transformed according to the required criteria. The curated data area stores preprocessed data ready for consumption according to purpose. On the last level we have the workspace data area with fully transformed and aggregated data that can be accessed by the various work teams who understand the needs of the organization.
- Following the ingestion process the data is stored in its raw form without transformation. Since WSAL in its raw form is data that is difficult to analyze and interpret, it was proposed to transform it into parquet files.
- To automatically transform data from log files to parquet files, we made an Azure Blob Trigger function written in Python. This trigger function monitors the raw area of the data lake. When it detects that a new log file



	access-weblog1.log	access-weblog2.log	access-weblog3.log	access-weblog4.log
DL curated parquet (MB)	108.39	111.20	116.06	31.26
DL raw log (MB)	1000.00	1000.00	1000.00	340.19
DL logerr txt (MB)	0.1020	0.0220	0.0188	0.0084

FIGURE 6. Storage space reduction achieved by transforming log files into parquet files.

has been successfully loaded, the trigger launches the transform function into execution. Various tests have been performed on loading the raw data into the DL with the aim of adjusting the transformation criteria so that we have as few non-conforming log lines as possible ending up in the logerrparsing directory. After transforming the log files to parquet files, a reduction in storage space of about 90% from the original size was observed. This transformation thus comes with 2 major advantages: a significant reduction in storage costs and the transformation of the data into structured data that is easy to retrieve.

By achieving the objectives, we have demonstrated the capability and performance of a DL [21], which in the near future will become one of the most widely used storage media for diverse data types. DLs can store all types of data within an organization, in one central location, without the need to impose a schema upfront as with Data Warehouses. Unlike most databases and data warehouses, DLs can process all types of data that are essential for advanced analysis processes [22], [23]. The main purpose of a DL is to make all organizational data, from different sources, accessible to end users: Business Analysts, Data Engineers, Data Scientists, Product Managers, Executives, etc. Their access to these varied sources of data can maximize the information gained from them in a cost-effective way in order to improve an organization's performance [24], [25].

V. RELATED WORKS

Log files provide relevant information about events related to client activities, server activities, etc. In the paper [26], algorithms are proposed to analyze the content of the information hidden in log files and to discover patterns of users identified together with their browsing behaviors, followed by grouping similar users based on different interesting contents of log files for many websites hosted on the web server. Finding statistics for each part of the log file command line that are not present in many log file analysis tools is supported here, with the final aim of discovering frequent websites, users, and user activities on those websites. The process of discovering hidden information in a WSAL file is called web mining. Its purpose is to gain insight into browsing behavior and retrieve useful information from very large raw datasets, which can be represented by several million event records in the log file. WSAL data contain different types of information, including web documents, web structure, and user profiles. Web mining can be classified into three categories depending on which part of the web needs to be extracted.

In DLs, typical challenges with Big Data, such as the variety, velocity, and volume of data, need to be solved. Since there are atypical challenges for this type of project and the processed data, current research suggests that it is normal to define several levels in the design stage of a DL. For example, in [27], the authors introduce a DL composed of three layers, i.e., assimilation, maintenance, and query. However, we noticed in this architecture a lack of consideration for access management control, which is important when dealing with heterogeneous data sources and especially with data that have sensitive information. Furthermore, we noted in the literature that it is common practice to define different areas for data storage in a DL [28].

One of the most effective ways to efficiently manage the growing edge cloud is to store each of the distributed edge cloud data in the cloud. The DL is a single domain model that can represent the most important data to the

entire business by accurately representing the data; this is performed by converting the information into enterprise data by efficiently processing large numbers of data. Typical DL models provide data capture, processing, analysis, and consumption systems for users or data. In the paper [29], the authors successfully propose a new concept of a connected data system. The authors implemented a connected DL system based on distributed micro cloud storage. The DL implementation delivers data stored in multiple edge clouds to be stored in the micro cloud. Additionally, the proposed design facilitates real-time error detection, i.e., the transferred data can be restored when an error occurs in the transmission process.

In the paper [30], the authors present the design and implementation of a production DL in Hadoop consisting of front-end production data; they also explain how the data are used to provide business values through advanced analytics. Thus, a DL is characterized by three key attributes, i.e., ''collect everything'': a DL contains all the data, both raw sources over long periods of time and any processed data; ''dive in anywhere'': a DL allows users across multiple business units to perfect, explore, and enrich the data on their terms; ''flexible access'': a DL enables multiple data access models in a shared infrastructure (batch, interactive, online, search, in-memory, and other processing engines). While storing and processing data at low costs on Hadoop enables large numbers of data to be collected, the use of these data requires the development of appropriate data formats, schemas, and query engines that allow semiconductor manufacturers to take advantage of them. Big Data have reached a crossroads that is expected to give companies more intelligence to make the right decisions at the right time.

The experiments presented in [31] illustrate the usefulness of implementing a DL as a unified data management and analysis platform. The main idea of a DL is to ingest raw data without processing and process the data upon usage. The proposed solution supports multi-level analysis involving the ingestion, transformation, and storage of Big Data in different formats from a variety of sources. The basic idea of a DL is simple: all data collected by an organization are stored in a single data warehouse in the lake in their original format. As such, the complex pre-processing and transformation of the loaded data into a warehouse are eliminated. In addition, the upfront costs of data ingestion are reduced. The major benefit of a DL is the centralization of content from disparate sources. A DL can have tens of thousands of tables or files and billions of records, requiring scalable data storage, management, and analysis [32], [33]. Once gathered in the DL, data from multiple sources can be correlated, integrated, and processed using state-of-the-art Big Data search and analysis techniques that would otherwise be impossible. A DL often contains proprietary or sensitive information that requires appropriate security measures. Security measures in a DL can be implemented to only allow individuals to have partial access to selected information and anonymize or encrypt data for users in a variety of roles who do not have access to the original data [34]. The lake should also allow one to have ubiquitous access to data from anywhere at any time. This feature could increase data reuse and help organizations to more easily collect, and process data as needed to drive business decisions [35].

An increase in IoT (Internet of Things) devices is a factor that is explosively accelerating data growth. In other words, the data generated by the IoT [36] and various devices are growing by the day, and edge clouds for effectively managing these data are increasing. This trend has increased the need to reliably and flexibly manage massive data stored in edge cloud storage. There is a growing interest in integrating the edge cloud and the cloud to efficiently manage growing data in the edge cloud. However, it is difficult to efficiently integrate large amounts of data between the edge cloud and the cloud. To solve this problem, research is needed for scalability, high availability, fault tolerance, high throughput, low latency, fast recovery, rescaling, end-to-end consistency, and transactional integration. One concept of connected data architectures is an interconnected data pool that can connect to pools in cloud data centers. All data are connected and flowing all the time. The connected DL in this paper limits the data connection between the edge cloud and micro cloud storage. The micro cloud storage in this paper is private cloud storage.

The paper [32] suggests a DL approach built based on Big Data technologies to gather all the data for further analysis. The platform described in this paper allows data collection, storage, integration, and analysis, and subsequent visualization of the results to be performed. Data generated from different sources, such as environmental sensors, social media platforms, and traffic counters, are leveraged to achieve these end goals. However, collecting, integrating, and analyzing all available heterogeneous data sources from cities is challenging. DLs, as opposed to data warehouses, are databases that contain data from different sources in structured, unstructured, and semi-structured formats, along with batch and real-time flow management capabilities [37], [38]. DLs also have some limitations. For example, implementing a DL requires a lot of technical effort, regardless of the availability of different frameworks that address the overall requirements. Integrating different data sources imposes requirements for metadata management [39], [40].

VI. CONCLUSION AND FUTURE WORK

This research project demonstrates the importance of using the new DL technology in the cloud to store large volumes of unstructured data. The new trend is to move and deploy a DL in the cloud, which offers comprehensive services for the entire process of data ingestion, storage, processing, and analysis with a high level of security—services that are being constantly improved by cloud service providers.

In this paper, we present a detailed theoretical description of the new cloud technology—ADLS Gen2—and at the same time, we propose and report the practical implementation of

an economical and reliable cloud DL architecture in order to provide an optimal implementation for storing WSAL data that can then be subjected to various advanced analysis processes. A major advantage of storing in ADLS Gen2 is that it enables hierarchical data storage. ADLS Gen2, through its hierarchical namespace system, supports atomic operations which eliminates the risk of data loss encountered with flat storage. There are a number of important differences between flat storage and hierarchical storage in terms of performance and security, among which we can list:

- query performance, with a hierarchical file system it is possible to scan only certain partitions to obtain the data searched that improves the data query processes.
- performance moving data, in a hierarchical system renaming or moving files from one directory to another is done almost instantaneously.
- data consistency and atomic operations, ADLS Gen2, through its hierarchical system, supports atomic operations which implicitly eliminates the risk of data loss if an error occurs during a move or rename operation, as with object-based storage.
- granular directory and/or file level security, directories and files can be given granular access permissions, which provides more flexibility in assigning and managing security on data.

Based on the studies and practical work performed so far, we intend to improve the transformation process to eliminate the errors produced by the transformation process as much as possible and to deepen various data enrichment and analysis processes in the cleansed data area. The goal is to demonstrate the ease with which data stored in a Cloud DL environment can be queried using the built-in services provided by Microsoft. Analysis techniques can be greatly varied, from the simplest to the most complex, based on built-in technologies or technologies built ad hoc depending on the final purpose.

REFERENCES

[1] L. Korzeniowski and K. Goczyla, "Landscape of automated log analysis: A systematic literature review and mapping study," *IEEE Access*, vol. 10, pp. 21892–21913, 2022, doi: 10.1109/ACCESS.2022.3152549.

[2] S. Hernandez, P. Alvarez, J. Fabra, and J. Ezpeleta, "Analysis of users behavior in structured e-commerce websites," *IEEE Access*, vol. 5, pp. 11941–11958, 2017, doi: 10.1109/ACCESS.2017.2707600.

[3] E. Zagan and M. Danubianu, "ADLS Gen 2 for web server log data analysis," in *Proc. Int. Conf. Develop. Appl. Syst. (DAS)*, Suceava, Romania, 2022, pp. 161–166, doi: 10.1109/DAS54948.2022.9786071.

[4] J. M. P. Jeba, M. S. Bhuvaneswari, and K. Muneeswaran, "Extracting usage patterns from web server log," in *Proc. 2nd Int. Conf. Green High Perform. Comput. (ICGHPC)*, Nagercoil, India, Feb. 2016, pp. 1–7, doi: 10.1109/ICGHPC.2016.7508074.

[5] P. Ghavare and P. Ahire, "Big data classification of users navigation and behavior using web server logs," in *Proc. 4th Int. Conf. Comput. Commun. Control Autom. (ICCUBEA)*, Pune, India, Aug. 2018, pp. 1–6, doi: 10.1109/ICCUBEA.2018.8697606.

[6] A. Nambiar and D. Mundra, "An overview of data warehouse and data lake in modern enterprise data management," *Big Data Cognit. Comput.*, vol. 6, no. 4, p. 132, Nov. 2022, doi: 10.3390/bdcc6040132.

[7] G. Turkington and G. Modena, "Big data con Hadoop," Apogeo, 2015. [Online]. Available: https://www.unilibro.it/libro/turkington-garry-modena-gabriele/big-data-con-hadoop/9788850333431

[8] *Load Data Into Azure Data Lake Storage Gen2 With Azure Data Factory*. Accessed: Jan. 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/data-factory/load-azure-data-lake-storage-gen2

[9] D. Sarramia, A. Claude, F. Ogereau, J. Mezhoud, and G. Mailhot, "CEBA: A data lake for data sharing and environmental monitoring," *Sensors*, vol. 22, no. 7, p. 2733, Apr. 2022, doi: 10.3390/s22072733.

[10] S. Kundu and L. Garg, "Web log analyzer tools: A comparative study to analyze user behavior," in *Proc. 7th Int. Conf. Cloud Comput., Data Sci. Eng.*, Jan. 2017, pp. 17–24, doi: 10.1109/CONFLUENCE.2017.7943117.

[11] B. Plejic, B. Vujnovic, and R. Penco, "Transforming unstructured data from scattered sources into knowledge," in *Proc. IEEE Int. Symp. Knowl. Acquisition Model. Workshop*, Wuhan, China, Dec. 2008, pp. 924–927, doi: 10.1109/KAMW.2008.4810643.

[12] *Hadoop is Data's Darling for a Reason*. Accessed: Nov. 2022. [Online]. Available: https://www.forrester.com/blogs/hadoop-is-datas-darling-for-a-reason/

[13] *The Unseen Data Conundrum*. Accessed: Nov. 2022. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2022/02/03/the-unseen-data-conundrum/?sh=4a07b7ac7fcc

[14] J. Sun, G. Gui, H. Sari, H. Gacanin, and F. Adachi, "Aviation data lake: Using side information to enhance future air-ground vehicle networks," *IEEE Veh. Technol. Mag.*, vol. 16, no. 1, pp. 40–48, Mar. 2021, doi: 10.1109/MVT.2020.3014598.

[15] *Ingestion and Processing Layers in Azure Data Lakehouse*. Accessed: Dec. 2021. [Online]. Available: https://www.mssqltips.com/sqlservertip/7037/azure-data-lakehouse-ingestion-processing-options/

[16] Z. Farzin. (2019). *Online Shopping Store—Web Server Logs*. Accessed: Sep. 2021. [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/3QBYB5

[17] *Azure Blob Storage Bindings for Azure Functions Overview*. Accessed: Jan. 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-blob?tabs=in-process%2Cfunctionsv2%2Cextensionv3&pivots=programming-language-python#trigger—polling

[18] *Azure Data Lake Storage Pricing*. Accessed: Feb. 2023. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/storage/data-lake/

[19] *Configure a Lifecycle Management Policy*Accessed: Oct. 2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/lifecycle-management-policy-configure?tabs=azure-portal

[20] *Optimize Costs by Automatically Managing the Data Lifecycle*. Accessed: Oct. 2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/lifecycle-management-overview

[21] S. Villarroya, J. R. R. Viqueira, J. M. Cotos, and J. A. Taboada, "Enabling efficient distributed spatial join on large scale vector-raster data lakes," *IEEE Access*, vol. 10, pp. 29406–29418, 2022, doi: 10.1109/ACCESS.2022.3157405.

[22] A. Cuzzocrea, "Big data lakes: Models, frameworks, and techniques," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Jeju Island, Korea (South), Jan. 2021, pp. 1–4, doi: 10.1109/BigComp51126.2021.00010.

[23] H. Fang, "Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem," in *Proc. IEEE Int. Conf. Cyber Technol. Autom., Control, Intell. Syst. (CYBER)*, Shenyang, China, Jun. 2015, pp. 820–824, doi: 10.1109/CYBER.2015.7288049.

[24] J. C. Couto and D. D. Ruiz, "An overview about data integration in data lakes," in *Proc. 17th Iberian Conf. Inf. Syst. Technol. (CISTI)*, Jun. 2022, pp. 1–7, doi: 10.23919/CISTI54924.2022.9820576.

[25] F. Nargesian, K. Pu, B. Ghadiri-Bashardoost, E. Zhu, and R. J. Miller, "Data lake organization," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 237–250, Jan. 2023, doi: 10.1109/TKDE.2021.3091101.

[26] T. A. Al-Asadi and A. J. Obaid, "Discovering similar user navigation behavior in web log data," *Int. J. Appl. Eng. Res.*, vol. 11, no. 16, pp. 8797–8805, 2016.

[27] R. Hai, S. Geisler, and C. Quix, "Constance: An intelligent data lake system," in *Proc. Int. Conf. Manag. Data*, Jun. 2016, pp. 2097–2100.

[28] A. Gorelik, *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science*. Sebastopol, CA, USA: O'Reilly, 2019.

[29] S. Park, B. Cha, and J. Kim, "Design and implementation of connected DataLake system for reliable data transmission," in *Proc. 23rd Int. Comput. Sci. Eng. Conf. (ICSEC)*, Oct. 2019, pp. 141–144, doi: 10.1109/ICSEC47112.2019.8974823.

[30] S. Munirathinam, S. Sun, J. Rosin, H. Sirigibathina, and A. Chinthakindi, "Design and implementation of manufacturing data lake in Hadoop," in *Proc. IEEE Int. Conf. Smart Manuf., Ind. Logistics Eng. (SMILE)*, Apr. 2019, pp. 19–23, doi: 10.1109/SMILE45626.2019.8965302.

[31] R. Liu, H. Isah, and F. Zulkernine, "A big data lake for multilevel streaming analytics," in *Proc. 1st Int. Conf. Big Data Anal. Practices (IBDAP)*, Sep. 2020, pp. 1–6, doi: 10.1109/IBDAP50342.2020.9245460.

[32] H. Mehmood, E. Gilman, M. Cortes, P. Kostakos, A. Byrne, K. Valta, S. Tekes, and J. Riekki, "Implementing big data lake for heterogeneous data sources," in *Proc. IEEE 35th Int. Conf. Data Eng. Workshops (ICDEW)*, Apr. 2019, pp. 37–44, doi: 10.1109/ICDEW.2019.00-37.

[33] T. Hlupic, D. Orescanin, D. Ruzak, and M. Baranovic, "An overview of current data lake architecture models," in *Proc. 45th Jubilee Int. Conv. Inf., Commun. Electron. Technol. (MIPRO)*, Opatija, Croatia, May 2022, pp. 1082–1087, doi: 10.23919/MIPRO55190.2022.9803717.

[34] M. Pingos, P. Christodoulou, and A. Andreou, "DLMetaChain: An IoT data lake architecture based on the blockchain," in *Proc. 13th Int. Conf. Inf., Intell., Syst. Appl. (IISA)*, Corfu, Greece, Jul. 2022, pp. 1–8, doi: 10.1109/IISA56318.2022.9904404.

[35] A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Dallas, TX, USA, Apr. 2020, pp. 709–720, doi: 10.1109/ICDE48307.2020.00067.

[36] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big IoT data analytics: Architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017, doi: 10.1109/ACCESS.2017.2689040.

[37] S. Ramchand and T. Mahmood, "Big data architectures for data lakes: A systematic literature review," in *Proc. IEEE 46th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Los Alamitos, CA, USA, Jun. 2022, pp. 1141–1146, doi: 10.1109/COMPSAC54236.2022.00179.

[38] A. A. Munshi and Y. A.-R.-I. Mohamed, "Data lake lambda architecture for smart grids big data analytics," *IEEE Access*, vol. 6, pp. 40463–40471, 2018, doi: 10.1109/ACCESS.2018.2858256.

[39] J. Singh, G. Singh, and B. S. Bhati, "The implication of data lake in enterprises: A deeper analytics," in *Proc. 8th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Coimbatore, India, Mar. 2022, pp. 530–534, doi: 10.1109/ICACCS54159.2022.9784986.

[40] B. Malysiak-Mrozek, A. Lipinska, and D. Mrozek, "Fuzzy join for flexible combining big data lakes in cyber-physical systems," *IEEE Access*, vol. 6, pp. 69545–69558, 2018, doi: 10.1109/ACCESS.2018.2879829.

ELISABETA ZAGAN (Member, IEEE) received the M.Sc. degree in automation and applied informatics from the Ştefan cel Mare University of Suceava, Suceava, Romania, in 2005. She is currently pursuing the Ph.D. degree with the Department of Computer Science, Ştefan cel Mare University of Suceava. Her research interests include databases, big data, data lake, and data warehouse.

MIRELA DANUBIANU (Member, IEEE) received the M.Sc. degree from the Department of Automation and Computers, Faculty of Electrotechnics Craiova, in 1985, and the Ph.D. degree from the Department of Computer Science, Ştefan cel Mare University of Suceava Romania, in 2006. She is currently an Associate Professor with the Ştefan cel Mare University of Suceava, with more than ten years of teaching and research experience.