ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 1, 2024

CONFIGURING THE STRUCTURE OF THE SERVERLESS SYSTEM FOR EFFICIENT DATA COLLECTION

Oleksandr Demidov, Oksana Honsor

Lviv Polytechnic National University, 12, Bandera Str., Lviv, 79013, Ukraine.
Authors' e-mails: *oleksandr.s.demidov@lpnu.ua, oksana.y.honsor@lpnu.ua*

https://doi.org/10.23939/acps2024.01.039

Submitted on 31.03.2024

Abstract: **Due to the constant development of information technology and the increasing volume of digital data, the concept of serverless systems has become relevant and promising in the field of software development. Serverless systems, also known as Serverless, are a new approach to deploying and managing applications. Developers can focus on developing functions without spending extra time managing servers and infrastructure. This approach is appropriate for various applications, including data processing, and is particularly useful for collecting and processing specialized data. However, there are numerous solutions and architectures available for data collection that cater to different data structures and requirements. The challenge is to select the most appropriate one and implement it for your specific use case.**

Index terms: **Data-collecting, Serverless Systems, AWS, Cloud, Architecture**

I. INTRODUCTION

Data collection, aggregation, and storage are essential components of most programs, projects, or services [1]. The demand for data from various sources is increasing due to the emergence of new developments, research, and expansion of existing services. In particular, machine learning requires the use of large datasets. Even relatively simple solutions require terabytes of datasets to train their models [2, 3]. In modern machine learning solutions, the primary challenge lies not in the algorithms themselves, but in the efficient collection, transformation, and aggregation of data.

Data collection can be a complex process with many steps and obstacles to consider. The specific steps vary depending on the type of data being collected and its intended use. Generally, data collection systems follow a logical sequence of steps. First of all, any collection starts with trigger collection execution. This can be initiated by various sources, ranging from manual triggers to cloud-based events or web hooks. Then, system usually updates the state of application (if it exists). Data collection applications typically have a database for storage of application status and data collection progress. The next step is execution the handling logic to obtain data collection parameters and create a folder or space on disk to store data collected from an external data source. Also, in this step, transfer the computation task to the actual workers who collect and store data, usually occurs. In case of failure, invoke a handler to check if the retry is possible and necessary at this time. As a result of its execution application should get either a new scheduled task or start execution of a new worker to retry the failed part of data collection. Finally, when all workers have finished their jobs, there should be a validation of the data integrity that was just collected. If everything worked successfully, the application state in the database should also be updated.

Start data aggregation logic parts, which are more dependent on project specifics. This may include data transformation for the corresponding model or obtaining statistics for the data. Alternatively, submit it to a machine-learning solution. So, for each of the goals mentioned above, developers would like to have some clear and systematic solution. While the data acquisition algorithm may be case-specific, the overall architecture can be reused and unified. It is worth noting that serverless systems [4] deserve special attention as they offer a new approach to application deployment and management. This allows developers to focus on feature development without spending extra time on managing servers and infrastructure [5]. This progressive direction is displacing the approach of installing physical servers, so when tackling the task of improving data collection efficiency, it is necessary to focus on implementing it in serverless systems.

The following architecture and results of its implementation can also be used in cyber-physical systems since they also frequently require data collection. Modern cyber-physical systems are usually integrated with the cloud, if it's possible, so a lot of them are using serverless-like systems to operate. For those systems, proposed architecture may bring high level of integration and better reliability, security and traceability. For example, a lot of cyber-physical real-time data devices want to fetch data and with the usage of proposed data collection architecture and real-time data streams (such as Amazon Kinesis Datastream for example) can bring fully working solution to mentioned use case.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

When designing a quality data collection system, it is important to consider two key areas: architecture and

algorithms for data processing and aggregation. Algorithms are more case-specific and cannot be generalized as they change depending on different circumstances. Therefore, it is advisable to focus on architecture and its components.

To improve data collection, it is recommended to start by considering a flawed solution. Using basic and non-adaptive examples can help identify the problems that need to be addressed. For instance, let's consider a scenario where a user needs to collect data from various Internet resources using a public API and their own set of servers. In this case, data is collected using the default HTTP client and stored randomly in chunks named with a timestamp when the data is downloaded. The data is then saved on the hard drives of the servers. Assuming that the data collected must be transferred to another system in a structured manner for the machine learning process. It is important to consider potential issues that may arise with this approach. Therefore, it is worth considering them in more detail and exploring possible solutions through a better-structured architecture.

Scalability – almost each architecture solution should worry about it. Without scalability, sooner or later architecture becomes irrelevant, especially in field of modern data collection systems. As mentioned in the previous section, serverless technologies are replacing the traditional on-premise approach. This enables flexible adaptation to the required data volume. Physical servers cannot provide rapid deployment of new computing power [4] when needed. Additionally, if these servers are not required for a certain period of time, they still require maintenance, which can result in additional costs depending on the scale [6].

Efficiency or, in this case, adaptability to demands. When using local servers, it is impossible to adjust both horizontal and vertical settings. In case there is a need for more CPU or RAM, an immediate increase of these parameters is impossible [7]. Some data collection, aggregation, or transformation tasks require more free space, while others require multithreading for better performance. The ability to identify and adapt to these needs will greatly improve the overall performance of the architectural solution [6].

Tracking and testing, having it onboard, makes products more stable and predictable. Applying the right software components and effort allows one to track almost all indicators on the server. This enables the collection of statistics on time, memory, load, traffic, number of errors during data collection, missing data and so on. However, collecting, summarizing and analyzing this data requires a lot of work. Cloud solutions provide these indicators with the help of unified server agents [8], which can even analyze data and identify bottlenecks in the system used. In addition, because they return data in the same structure for many applications worldwide, the user has access to numerous third-party open-source solutions for processing this data.

Reliability, not the most important part particularly in data collection use case, but generally required for good architecture solutions. During the actual storage of data on only one device, there is a risk of losing the data that is being collected at the moment and the data that was previously collected, structured and prepared for transmission. Therefore, a backup copy for the storage is required. Cloud solutions eliminate this problem by duplicating servers that store data [8], providing protection against events such as power outages and natural disasters.

Security. This one is truly needed, especially when collecting data, as some of it can be private or even confidential. Ensuring data security during collection and processing, as well as protecting servers and the system as a whole, is a crucial issue. Physical servers are more secure due to their isolation. However, tasks related to data collection require interaction with external resources, which negates this advantage as a network gateway is used anyway [9]. Cloud technologies offer enhanced security by enabling the storage of private variables, such as credentials and API keys, in secure cloud systems. These variables can be extracted from the system without going through an internet gateway. In addition, roles can be assigned to each part of the deployment architecture to restrict access to specific resources, and access to infrastructure resources can be controlled. Furthermore, access to infrastructure resources can be blacklisted or whitelisted through the implementation of network rules. Security analysis tools are available to detect security issues and suggest solutions to problems in a particular architecture [9].

Process orchestration. By taking care of this requirement, one can provide a lot of scalability and reliability advantages to architecture solutions. In the example of the bad decision described above, no plan for data collection orchestration was proposed. Therefore, if specific data needs to be collected, the action must be run manually, and any unsuccessful actions must be repeated if necessary. This may occur if the data fails to load or if an inconsistency is detected. Furthermore, if this process is not properly orchestrated, it can result in a significant performance loss. Unintentionally causing a DDOS attack on a server can trigger a threshold of API responses, leading to the blocking of each request until the server recognizes your eligibility for the data. To prevent this, patterns such as Circuit Breaker [10] can be useful. However, a well-organized infrastructure system is necessary for their application. This includes not only data collection but also every other part of the system. Good orchestration can be achieved by triggering actions based on events or webhooks that are called according to the user application's state stored in a database.

Place of data storage. Depending on the goals, data can be stored on different types of equipment. Certain data should be archived and stored on simple hard drives as a backup, as it does not require high efficiency to function. Other types of data may need to be stored on faster SSDs or even in RAM [11] because they are required frequently to fill in other pieces of data. The user may not immediately know where to store the data. With the use of cloud solutions, data can be stored in the most appropriate space,

providing the necessary level of flexibility. Additionally, cloud solutions like EFS (Elastic File System) or EBS (Elastic Block Store) can facilitate data sharing among multiple servers and processes [11], leading to improved parallelization for data reading and storage. In this case, correct multi-source writing is important, but it can significantly improve system performance [12].

Other improvements. An important advantage of utilizing cloud solutions is their cost-effectiveness while maintaining high work efficiency. Additionally, a variety of other services can be provided within a single cloud. These services can be used for real-time data collection, data aggregation from large files, and customizing machine learning scripts for processing newly added data.

After discussing common issues that may arise when using standard data collection methods, it is important to note that these problems are unique and, in some cases, standard solutions may be effective. However, creating flexible solutions requires a different approach that can address the aforementioned problems. The following section proposes solutions to these issues. This fragment represents a fundamental approach to data collection architecture.

III. SCOPE OF WORK AND OBJECTIVES

The aim of this research is to develop and compare serverless architectures, solutions, and data collection methods using cloud computing. The subsequent step is to generalize the findings to select the appropriate architecture for various cases based on variables such as data, scale, format, and cost. The design and architecture of potential data collection solutions were created using the Amazon Web Services cloud platform. Different AWS services were utilized for various aspects of data collection, such as Lambda, ECS, and EC2 for computing power, MongoDB, Athena, and S3 for data aggregation, and IAM and AWS System Manager for security concerns. Additionally, CloudFront, Route 53, API Gateway, and Event Bridge were employed. The primary objective was to evaluate the outcomes of various architecture and design combinations to determine the optimal solution for specific data collection scenarios. The proposed architectural solutions analyze issues such as efficiency, data overflow, security, resilience to server errors, and data inconsistencies, and provide ways to resolve them.

The purpose of data collection may differ depending on different purposes and specifics, but in most cases, clear and structured data are necessary. It is important to understand their structure, or at least to have an algorithm that can structure them in the desired way. Most modern data sources and APIs provide this structure. When collecting and aggregating data for user needs, it is important to consider not only the methods of downloading and storing it but also selecting an environment that enables efficient data aggregation. Additionally, when working with large amounts of data, prioritizing the performance of the data aggregation environment is crucial.

IV. RESULTS AND DISCUSSION

A cloud-based architecture was developed to address common issues encountered by data collection systems. The first step was selecting the cloud platform for this solution, and the AWS cloud was chosen due to its popularity, large scale, and cost efficiency [13]. Additionally, it boasts the widest network and the greatest number of data centres worldwide, ensuring high availability and elasticity for both the user and the system as a whole.

A. GENERAL ARCHITECTURE AND SERVICE CONNECTIONS

AWS is a collection of services that can be used for various purposes, including creating short-term functions with computing power, data storage design, and tracing. With over 200 services available, AWS can provide significant assistance in any area of cloud computing. Only about 10 services are needed for a data acquisition system of moderate complexity. The remaining 10 services will be used under the hood.

When considering the communication between services, it is important to also consider their respective responsibilities. The next section will provide the working algorithms for this infrastructure, but here is a brief overview of the general architecture. It is crucial to configure the architecture for most data acquisition systems, as the algorithms may change. The structure of AWS is shown in Fig. 1

1) AWS VPC: The AWS infrastructure is encapsulated by the wrapper. It defines a virtual private cloud for all infrastructure, created within data-collection systems [14]. The only resources that are not included in this encapsulation are those for monitoring and the database.

2) MongoDB (database). MongoDB was selected as one option, but it can be substituted with any other database for this architecture. Alternatively, it can be a database hosted on AWS. However, to demonstrate the feasibility of this solution, it is recommended to use a database external to AWS.

3) Event Bridge. This service triggers events for various AWS services, such as SQS and Lambda. It can also trigger events based on a cron expression, which acts as a timer. This is the starting point for all data collection processes.

4) SQS (Simple queue service). SQS is a crucial component of our architecture, providing significant efficiency and parallelization. It stores messages containing payloads, such as query parameters for data collection. SQS then waits for available computation power, specifically our Lambda function, to process the message. As an orchestrator of computation processes in our system, SQS ensures a logical flow of information with causal connections between statements.

5)Lambda Function**.** The choice of service used to obtain the necessary computing power is crucial in development. Its choice depends on the requirements of a specific task [15]. Lambda function, for example, offers fast and efficient execution of cloud functions with minimal memory usage. Its availability and speed allow it to invoke hundreds of thousands of containers while loading

data, achieving high levels of parallelism. The main function of a Lambda is to collect and load data into storage.

6) STS and Secret Manager. Lambda is supported by STS and Secret Manager services [15]. STS provides roles for the Lambda function, allowing it to write files to the storage or get and delete messages from SQS. Secret Manager allows you to securely store API keys or passwords for use by Lambda at runtime.

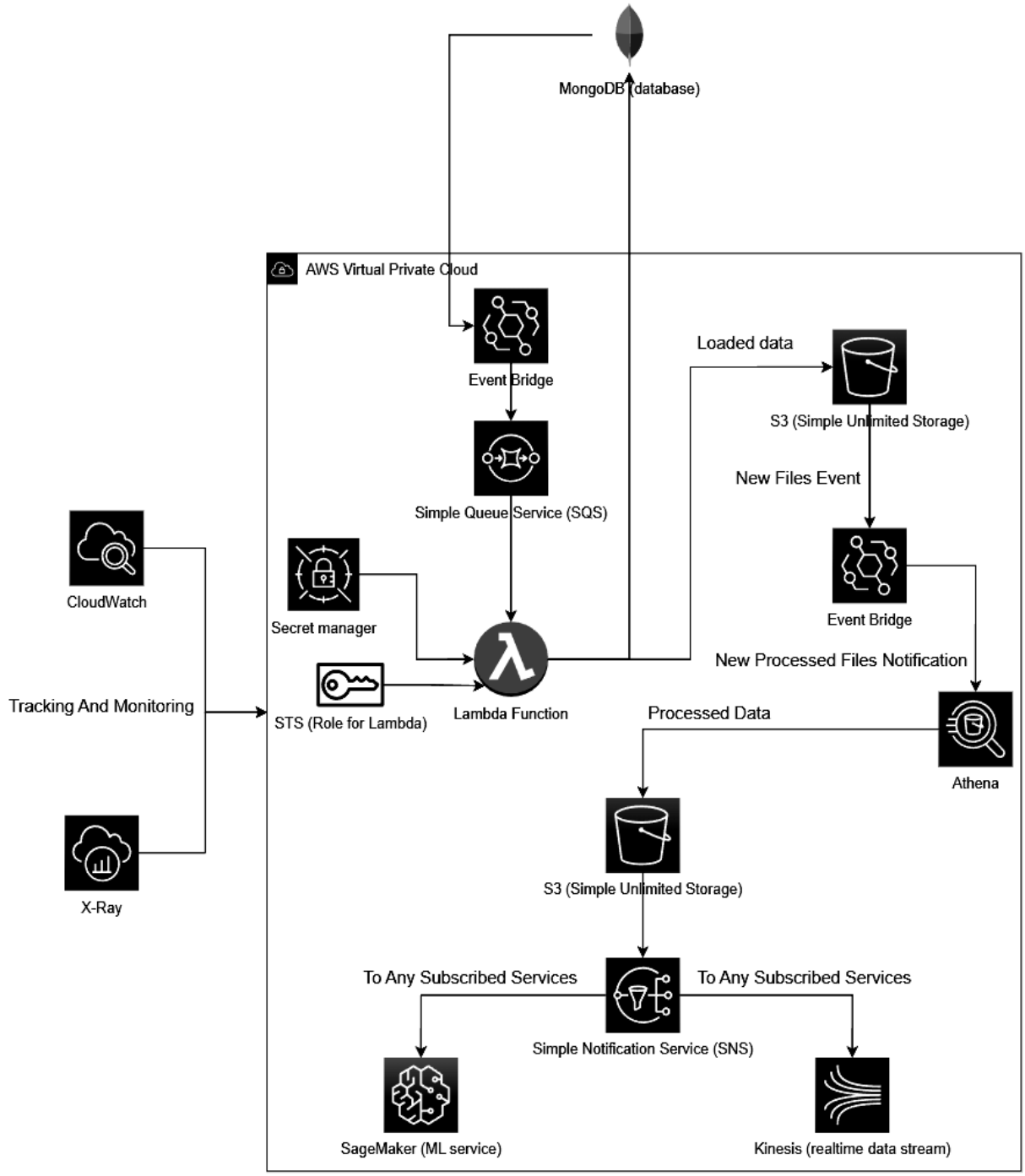


Fig. 1. AWS architecture of data collection system

7) S3. It's a large storage of files that allows for the storage of an unlimited amount of data with different access performance and pricing policies [15]. All collected data files will be stored there after they have been transformed. While other services such as EBS or EFS may offer better performance, S3 is sufficient for most cases.

8) Athena. This service enables SQL queries and data aggregation from previously collected data stored on S3. It has the capability to query almost any file type [15].

9) SNS (Simple Notification Service). This service is comparable to SQS, but it allows for multiple consumers of messages instead of just one. It operates in a Publisher/Subscriber model, meaning that the number of lis-

teners does not affect its functionality. Each subscribed service receives a notification.

10) Other consumers, Sagemaker, Kinesis. These two services, an ML solutions service in AWS and a real-time data stream service are used as examples of end users of the data provided by our system. They receive the aggregated data to proceed with other parts of their business logic.

B. WORKING SOLUTION BASED ON ARCHITECTURE

After presenting the basic architecture of the cloud and its connections, the next step is to examine its overall functionality. The following section describes how the aforementioned services are utilized to complete the entire data collection cycle within the system.

The initial step of any data collection process is the trigger phase. Depending on needs of each individual user it can be developed in various ways. But to build independent abstract architecture, we will need to support any possible implementation of this step. In the proposed architecture there is no difference between user-created data collection requests, timer event triggers, or even most of the other events which can be integrated with external services, such as IoT device sensors, and data scrapers, which check for new information that appeared on various 3rd party resources.

They all are going to generate simple SQS messages with some payload. Payload is a vital part of this process, it should specify all the details, parameters and data types for data source, which user wants to collect data from. At this stage, we got to 1st Lambda function (orchestration one).

It should be noted that at this stage orchestration lambda can return either new formatted message for worker lambda (through SQS service) or even invoke another orchestration lambda to continue the orchestration process (Fig 2.).

Next, we're coming to the actual core of the data collection process. It's an interaction between orchestration labmda, woker lambda an s3, which is our data storage itself.

It's also orchestrated by SQS, as in all other cases when we need to pass some sort of data or callback between AWS services.

When it comes to worker lambda, it's not as simple as one might think. It requires high level of abstraction to adapt to different cases. First of all, we need to receive and process SQS messages and extract information from them. It's usually done in *vendor* section of lambda controller. In this stage we should also pass expected data interface to match all received values, for instance, transform data types, merge some fields or events make HTTP calls for some required information if needed.

In *controller* section of lambda, we have an enum of possible data collectors. By analyzing payload we distinguish which one we should use. In this step, we also have circuit breaker [16]. It's a pattern that allows us to handle errors in various ways and more importantly – deal with server giving us timeout due to too many requests sent in some period of time. Most API don't specify max amount of calls to be made within some period of time, so we will frequently encounter the mentioned issue. In a few words, circuit breaker allows us to open and close flow of requests with fallback system to reschedule http calls sequences. This process is displayed on the individual requests example (Fig 3).

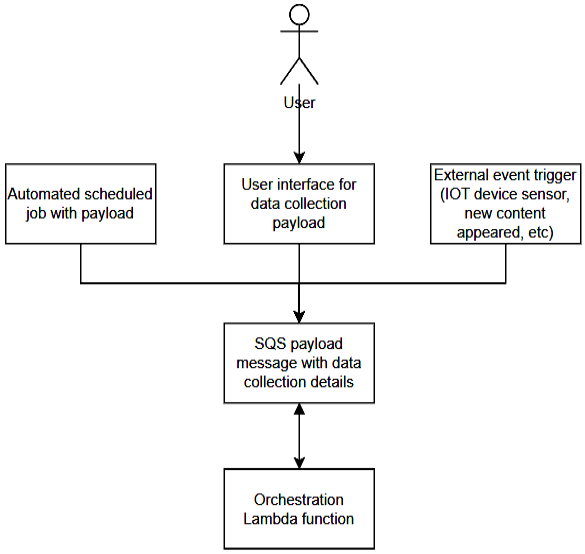


Fig 2. Proces trigger sources

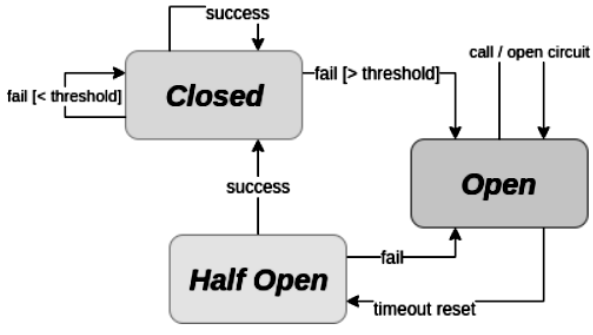


Fig 3. Circuit breaker process

Then after controller we have *abstract.collector* which specifies abstract class for collecting process from certain data collection type from enum. In this stage we reach out for all required credentials, initialize requesting client and do all needed steps to prepare for data collection. It also should be reusable for every data collection option, so there should be nothing specific about some data collection method itself.

Another abstract.repository entity is for data storing process. Just as abstract collector it should do all preparations for storing collected data in its implementations. Here, for example, we specify methods to identicate

where we should store collected data, filename, position, data type and make some trasformation to uploading data if needed.

On top of all mentioned entities [17], we also have folder for each data collection method from enum. It contains 2 files with actual exact implementations of data collection and storage logic parts. They are called *<data-collection-type>.collector* and *<data-collection-type>.repository*. As you can expect here we work directly with API's to download, transform and store data. In Fig 4, you can see detailed representation of data structure and communications for controller architecture.

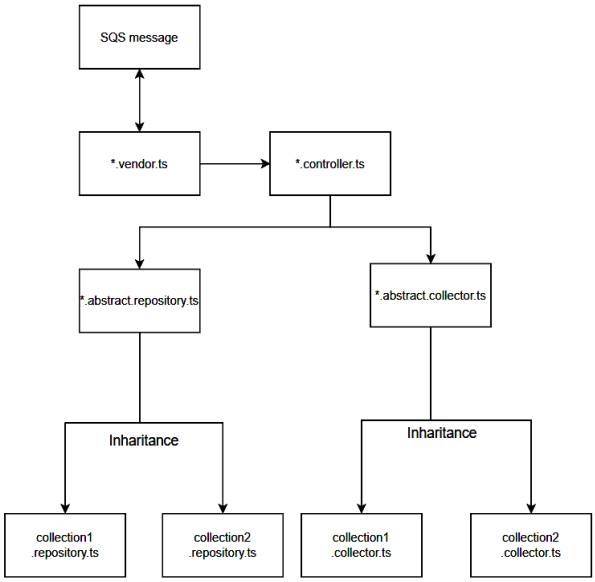


Fig 4. Controller architecture

Now, when controller architecture is fully covered, we will step forward to the moment when it's already executed and needed data is written to s3 bucket.

In this stage, there are various other methods to structure, archive and analyze data. For example, let's imagine that we want to aggregate newly collected data and extract some statistics data from it. For that, we can use Athena or some other data aggregator. Needed params can be passed into SQS with specific step, called "post-processing". They make any sort of query requests to collected data.

There are a lot of variations for this architecture, but the core problem solutions should be resolved. So, for mentioned case, we can just combine some parts of architecture with solutions that match our needs. Also, in some specific cases, some architecture steps may be redundant, thus removed to have less complexity. Such examples can be data aggregation if it's not needed, or even circuit breaking, in case user knows that he can't face cases of API blocking incoming requests.

The architectural solution developed was tested and demonstrated great potential as a common base for small and medium-sized data collection systems. Most of the issues listed in this article have been resolved in various ways. The architecture has achieved better performance in many key areas important to data collection systems with cloud computing and serverless technologies.

The suggested architectural solution provides a general approach to addressing common issues. It is recommended to tailor the structure to each specific system. Depending on the data being collected, the APIs used to load the data, and the method of data transformation, various architectural components may require modification. These components may include the source of computing power, the work coordinator, and data storage. Additionally, minor adjustments may be made depending on the end user of this data, and some parts of the architecture may be modified for services outside of the AWS cloud.

Due to high level of abstraction, almost any part of the process can be extended, but not modified. Taking into account the load system can face with such kind of architecture solution, it can be argued that this is a great working solution or starting point for most applications' needs. The only possible downside of the proposed architecture is its incompatibility with real-time data collection. This would reduce some other advantages of the proposed solutions, thus the decision to ignore real-time data collection was made.

This research demonstrates the advantages of a general architectural view for data collection. It also highlights common issues and pitfalls when designing architecture for a data collection system. It also proposes a solution using a cloud-based serverless architecture.

V. CONCLUSION

This paper presents a novel architectural solution for serverless data collection systems. The proposed architecture addresses common issues and provides benefits in crucial areas of data collection. It demonstrates excellent scalability and efficiency by leveraging cloud computing (AWS), which offers modern cloud features such as storage, computing power, orchestration, and more.

Another important aspect to consider is tracking and security. Tracking allows for precise testing of systems under different loads while working with various data sources and APIs. This can be a significant advantage for further investigations into improving cloud-based architectures. Additionally, it enables detailed reports on possible failure points and suboptimal usage of hardware.

A limitation of this investigation is the inability to work with vast amounts of data resources and external consumer systems. Gathering reliable data for numerous scenarios requires significant effort. However, the tested data volume is sufficient for the initial stages of this investigation. It provides enough insight to identify general bottlenecks and problems, as well as alternative solutions.

Further research entails collecting additional data on working with diverse data sources and aggregating data for various external consumers. This will provide a more profound comprehension of developing either a more generalized solution with multiple variables depending on the end goal or creating different types of cloud serverless solutions adapted to different data collection systems and their requirements.

References

[1]. Müller, I., Marroquín, R., Koutsoukos, D., Wawrzoniak, M., Akhadov, S., Alonso, G. (2020) The collection Virtual Machine: An abstraction for multi-frontend multi-backend data analysis. *Proceedings of the 16th International Workshop on Data Management on New Hardware, DaMoN '20*. DOI: 10.1145/3399666.3399911

[2] Yuji Roh, Geon Heo, Steven Euijong Whang (2019). A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering,* Vol 33, pp. 1328 – 1347. DOI: https://doi.org/10.1109/TKDE.2019.2946162

[3] MN Sarkies, KA Bowles, EH Skinner (2016). Data collection methods in health services research. *Applied Clinical Informatics. 6(1).* pp. 96-109. DOI: https://doi.org/10.4338/ACI-2014-10-RA-0097

[4] Sam Newman, Building Microservices: Designing Fine-Grained Systems. 1st Edition / O'Reilly, 2015, 250 p.

[5] R. Arokia Paul Rajan (2018). Serverless architecture – a revolution in cloud computing. *Tenth International Conference on Advanced Computing (ICoAC). 23 December 2019.* DOI: https://doi.org/10.1109/ICoAC44903.2018.8939081

[6] Sebastian Lehrig, Hendrik Eikerling, Steffen Becker (2015). Scalability, elasticity, and efficiency in cloud computing: a systematic literature review of definitions and metrics. *QoSA '15: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. 4 May 2015.* DOI: https://doi.org/10.1145/2737182.2737185

[7] H.M. Khan, A. Khan, F. Jabeen, A. Anjum, G. Jeon (2021). Fog-enabled secure multiparty computation-based aggregation scheme in smart grid. *Computers & Electrical Engineering, Vol. 94, 107358*. DOI: https://doi.org/10.1016/j.compeleceng.2021.107358.

[8] Maricela-Georgiana Avram (2014). Advantages and challenges of adopting *Cloud Computing from an Enterprise Perspective. 15 January 2014*. DOI: https://doi.org/10.1016/j.protcy.2013.12.525

[9] Mazhar Ali, Samee U. Khan, Athanasios V. Vasilakos (2015). Security in cloud computing: Opportunities and challenges. *Information Sciences, 305.* pp. 357-383. DOI: https://doi.org/10.1016/j.ins.2015.01.025

[10] Gibeon Aquino, Rafael Queiroz, Geoff Merrett & Bashir Al-Hashimi (2019). The circuit breaker pattern targeted to future IoT applications. *Part of the Lecture Notes in Computer Science book series (LNPSE, volume 11895).* pp 390–396. 2019. DOI: https://doi.org/10.1007/978-3-030-33702-5_30

[11] Binbin Song, Yao Yu, Yu Zhou, Ziqiang Wang & Sidan Du (2018). Host load prediction with long short-term memory in cloud computing. *The Journal of Supercomputing, Vol. 74(12),* pp. 6554–6568. DOI: https://doi.org/10.1007/s11227-017-2044-4

[12] Joel Gibson, Robin Rondeau, Darren Eveleigh, Qing Tan (2016). Benefits and challenges of three cloud computing service models. *Fourth International Conference on Computational Aspects of Social Networks (CASoN). 15 January 2016.* DOI: https://doi.org/10.1109/CASoN.2012.6412402

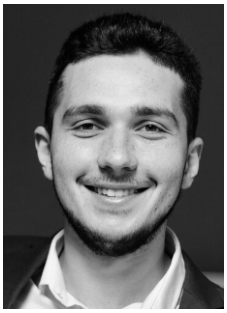
[13] Sourav Mukherjee (2019). Benefits of AWS in modern cloud. 7 March 2019. DOI: https://doi.org/10.48550/arXiv.1903.03219

[14] Andreas Wittig, Michael Wittig (2022). Amazon Web Services in Action, Third Edition: An In-depth Guide to AWS. May 2022. ISBN: 163343916X

[15] Sajee Mathew (2014). Overview of Amazon Web Services. Whitepaper. Pp. 25-58. November 2014. Available from: https://www.sysfore.com/Assets/PDF/aws-overview.pdf

[16] Fabrizio Montesi, Janine Weber (2016). Circuit Breakers, Discovery, and API Gateways in Microservices. DOI: https://doi.org/10.48550/arXiv.1609.05830

[17] Duc Minh Le, Duc-Hanh Dang, Viet-Ha Nguyen (2018). On domain driven design using annotation-based domain specific language. *Computer Languages, Systems & Structures, Vol. 54*. pp. 199-235. DOI: https://doi.org/10.1016/j.cl.2018.05.001



Oleksandr Demidov received a Master's degree in the Department of Specialized Computer Systems at Lviv Polytechnic National University. Since 2019 till now he has been a Software Engineer at a Software Developing company for web interfaces and serverless backend solutions. Currently, he is a PhD degree student of Computer Engineering at Lviv Polytechnic National University. His research interests include cloud-based architecture and serverless solutions.

ORCID ID: https://orcid.org/0009-0004-3464-1655

Oksana Honsor PhD, Assoc. Professor at the Department of Specialized Computer Systems, Institute of Computer Technology, Automation and Metrology of Lviv Polytechnic National University. Scientific interests include metrological support in IoT systems and sensor networks, ensuring traceability, processing of large data sets for reproduction and comparison of the results of measurements of physical quantities in remote mode. She is the author and co-author of 40 scientific and conference papers, co-author of 15 educational and methodological instructions and 3 electronic educational and methodological complexes used in the educational process.

ORCID ID: https://orcid.org/0000-0003-0895-5859