International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

Smart City IoT Data Management with Proactive Middleware

Vikas K Kolekar[1], Meet Oswal[2], Yash Wankhade[3], Gayatri Shirke[4], Archana Sondur[5], Prajwal Wable[6]

[1]Department of Computer Engineering
Vishwakarma Institute of Information Technology
Pune, India
vikas.kolekar@viit.ac.in
[2]Department of Computer Engineering
Vishwakarma Institute of Information Technology
Pune, India
meet.21911162@viit.ac.in
[3]Department of Computer Engineering
Vishwakarma Institute of Information Technology
Pune, India
yash.21910843@viit.ac.in
[4]Department of Computer Engineering
Vishwakarma Institute of Information Technology
Pune, India
gayatri.21910322@viit.ac.in
5Department of Computer Engineering
Vishwakarma Institute of Information Technology
Pune, India
archana.21910678@viit.ac.in
6Department of Computer Engineering
Vishwakarma Institute of Information Technology
Pune, India
prajwal.21911184@viit.ac.in

Abstract— With the increased emergence of cloud-based services, users are frequently perplexed as to which cloud service to use and whether it will be beneficial to them. The user must compare various services, which can be a time-consuming task if the user is unsure of what they might need for their application. This paper proposes a middleware solution for storing Internet of Things (IoT) data produced by various sensors, such as traffic, air quality, temperature, and so on, on multiple cloud service providers depending on the type of data. Standard cloud computing technologies become insufficient to handle the data as the volume of data generated by smart city devices grows. The middleware was created after a comparative study of various existing middleware. The middleware uses the concept of the federal cloud for the purpose of storing data. The middleware solution described in this paper makes it easier to distribute and classify IoT data to various cloud environments based on its type. The middleware was evaluated using a series of tests, which revealed its ability to properly manage smart city data across multiple cloud environments. Overall, this research contributes to the development of middleware solutions that can improve the management of IoT data in settings such as smart cities.

Keywords- cloud computing, IoT, middleware, smart city, SLA

I. INTRODUCTION

The emergence of the Internet of Things (IoT) has led to the generation of large volumes of data from various sensors and devices in smart cities. This data is crucial for providing insights into urban infrastructure and improving city services, such as transportation, energy consumption, and public safety. However, the challenge of managing and storing this data has become increasingly complex due to factors such as cost, security, and data privacy.

Because of its low cost and scalability, cloud computing has become a popular solution for data storage and management. Cloud storage is an important service of cloud computing, which offers storage as a service, supports different database technologies, and allows data owners to store their data in the cloud[8]. Cloud data storage frees data owners from the burden of maintaining an expensive

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

on-premise storage infrastructure and offers economies of scale benefits[16]. In recent years cloud computing has shown huge success as a service-oriented computing paradigm that allows easy and on-demand network access to a shared pool of resources[7, 24].

Despite all these benefits, traditional cloud computing solutions, on the other hand, may not offer the flexibility and customization required for managing data, data security, performance, and availability[9, 10]. This is where federated cloud computing can help. Federated clouds are composed of multiple cloud providers that work together to provide a more comprehensive solution to data storage challenges[25]. By combining different cloud providers, federated clouds offer more flexibility and scalability than traditional cloud computing solutions[18,9,19,20]. This approach allows organizations to choose the most appropriate cloud providers based on their specific needs and requirements. There has been significant research in federated cloud computing in recent years. Federated cloud computing is a new paradigm in cloud computing that aims to provide a more comprehensive and cost-effective solution for data storage challenges. Federated clouds are composed of multiple cloud providers that work together to provide a more comprehensive solution to data storage challenges. The flexibility and scalability of cloud computing are two of its most significant advantages. Middleware, cloud brokerage, and service-level agreements are some of the solutions proposed by researchers for federated cloud
computing. There has been extensive research on various aspects of IoT data storage, such as data security, data privacy, and data management. Because of the unique characteristics of IoT data, such as its high volume, velocity, and variety, traditional cloud computing solutions may not be suitable for managing it. As a result, new approaches for managing and storing IoT data generated by smart cities are required.

Some researchers have proposed solutions for managing IoT data using federated clouds. For example, Dimitri et al. proposed a framework for data storage and management in federated clouds[11]. Their framework provides a data placement strategy to ensure data availability and data access time based on the user's requirements. Similarly, Zhang et al. proposed a solution for storing and managing IoT data in a federated cloud environment[21]. Their solution uses a data aggregation algorithm to compress the data and reduce storage costs.

In this research paper, we propose a middleware for the federated cloud to store IoT data generated by smart cities. Our middleware provides a customized Service Level Agreement (SLA)[30] to manage data storage based on factors such as cost, bandwidth, security, and volume preferences. By combining different cloud providers, our middleware provides a more comprehensive and cost-effective solution for data storage, improving efficiency and reducing costs.

The proposed middleware leverages the strengths of different cloud providers to address the unique challenges associated with storing and managing IoT data generated by smart cities. By providing a customizable SLA[29], our middleware enables users to choose the most appropriate cloud providers based on their specific needs, thereby enhancing the efficiency and reliability of data storage.

Our middleware leverages the strengths of different cloud providers to provide a more comprehensive and cost-effective solution for data storage. By combining different cloud providers, our middleware can provide better data storage and retrieval performance, as well as better data security and data privacy.

To evaluate the effectiveness of our proposed middleware, we conducted experiments on a smart city testbed. We measured the performance of the middleware in terms of data storage, retrieval, and security. The results of our experiments demonstrate the effectiveness of our proposed middleware in storing IoT data generated by smart cities.

The rest of the paper is organized as follows. In Section III, we provide a detailed review of related work in federated cloud computing and IoT data storage. In Section IV, we describe the proposed middleware and its components. In Section V, we present the experimental setup and results. In Section VI, we discuss the implications of our research and future work. Finally, in Section VII, we conclude the paper with a summary of our contributions and their significance.

II. MOTIVATION

The rise of smart cities[27] has revolutionized urban living, making it more efficient, sustainable, and convenient for inhabitants. Smart cities leverage the Internet of Things (IoT) technology to collect and transfer data from various sensors, devices, and systems, generating vast amounts of data[26]. This data can be used to optimize city services, such as traffic management, waste disposal, energy distribution, public safety, and more. However, managing and

storing this data can be a challenging task due to its volume, velocity, variety, and veracity.

One of the key challenges of managing IoT data in smart cities is finding an efficient and cost-effective way to store and process it. Traditional data storage solutions such as on-premise servers, relational databases, and file systems are often inadequate for handling the scale and complexity of IoT data. Cloud computing has emerged as a popular solution for storing and managing IoT data, as it offers scalability, flexibility, and pay-as-you-go pricing. However, manually selecting cloud platforms for data storage can be a daunting task, as each provider has its strengths and weaknesses in terms of storage capacity, bandwidth, security, and cost.

To address these challenges, a middleware for smart city IoT data management can be developed, which can act as a bridge between the IoT devices and cloud storage platforms. The middleware can provide a unified and flexible interface for IoT devices to send data to multiple cloud providers simultaneously. This can significantly enhance the efficiency and cost-effectiveness of data storage, as the middleware can intelligently select the best cloud provider based on the user's requirements and the characteristics of the data.

The motivation to create a middleware for smart city IoT data management lies in the need for an efficient and cost-effective solution to handle the vast amounts of data generated by IoT devices. This is based on the frequent interaction and experience with industry-level Software-as-a-service providers[23] With multiple sensors collecting data at high frequency, smart cities generate enormous amounts of data that require efficient storage and management.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

Manually selecting cloud platforms for data storage can be challenging as each provider has its strengths and weaknesses regarding storage capacity, bandwidth, security, and cost. Therefore, a middleware that provides different storage resources and combines different cloud providers based on the user's requirements can significantly enhance the efficiency and cost-effectiveness of data storage.

Furthermore, the middleware can enable users to customize Service Level Agreements (SLAs) for different cloud providers, based on their specific needs and preferences. SLAs can include parameters such as cost, bandwidth, security, and volume preferences, which can be tailored to different types of data and applications. The middleware can manage storage on the cloud based on these customized SLAs, ensuring that data is stored securely and efficiently while also reducing costs.

Another advantage of using middleware for smart city IoT data management is that it can provide a unified view of the data, regardless of the underlying cloud storage platforms[28]. This can enable users to access and analyze the data seamlessly, without worrying about the technical details of the storage infrastructure. The middleware can also provide data quality and integrity checks, ensuring that the data is consistent and accurate.

The middleware can be designed to be modular and extensible, allowing for easy integration with different IoT devices and cloud providers. It can also support various data formats, protocols, and APIs, making it compatible with different IoT applications and platforms. The middleware can be deployed on-premises or on the cloud, depending on the user's requirements and preferences.

In conclusion, the motivation to create a middleware for smart city IoT data management lies in the need for an efficient, cost-effective, and reliable solution to handle the vast amounts of data generated by IoT devices in smart cities. The middleware can provide a unified and flexible interface for IoT devices to send data to multiple cloud providers simultaneously, intelligently selecting the best provider based on customized SLAs. The middleware can also provide a unified view of the data, data quality and integrity checks, and modular and extensible design. Such a middleware can enable smart cities to leverage the full potential of IoT data, making them more sustainable, efficient, and livable.

III. RELATED WORK

Creating a middleware for self-adaptive IoT service is difficult as it involves many issues which need to be addressed while creating it. The main obstacles are the domain-specific and device-specific components required for various different applications, and it becomes complex to handle them all for a hybrid application that uses various kinds of devices, components, etc. A self-adaptive IoT solution is one that can adjust its behavior and configuration automatically in response to changes in the environment or system. This type of solution can provide several benefits such as improved reliability, security, scalability, and efficiency. There have been several self-adaptive middleware frameworks introduced. The three frameworks that stand out among them the most are TOGAF, Rainbow, and OSGi (Open System Gateway infrastructure). The paper has proposed a solution by Cloudization of MAPE cycle for IoT collaboration service: the domain and device-specific components have been separated from the implementation and they provide APIs for users to easily interact with the system, Development of a convenient web-based interface: they have created an interface where a user can select some implementation from an existing list of solutions or they can even upload their own solution, Implementation, and evaluation on a real testbed. Their study [1] shows the proposed architecture can mitigate four different kinds of attacks of three different layers: machine-to-machine, network, and cloud. The paper mentions four components that can be used to extend the performance of self-adaptive IoT solutions which are: monitoring, analyzing, planning, and execution.

Fogbow [2] is a middleware that works on private IaaS clouds and is based on a membership model and plugin architecture. For any client using private IaaS the individual utilization of resources is very minimal and this can be increased by server consolidation because not all the applications (clients) will peak demand for the resources at the same time. When there is a peak load in one member then the requests can be served by another member who has lesser utilization. The federation of private IaaS will increase the utilization of resources by aggregating the resources. The utilization rate by the federation will be higher than that of using IaaS individually. A federation is created with different private IaaS as its members. Plugins here are used for interaction between the middleware and the cloud. Here the federation can be created by using the same type of private IaaS or a different one. Resources are allocated global identifiers across the system. And for authentication of a member, the middleware should perform authorization with a maximum number of identification techniques used by providers.The middleware must not interfere with the existing policies, guidelines, and structure of the private IaaS providers. The middleware consists of various private IaaS clouds which are the members and will have a membership manager each and there is one allocation manager in the federation. Any application (client) connects with the allocation manager and based on availability the allocation manager interacts with the private clouds. The clouds are inter-accessible providing instances from one cloud to be accessible from another cloud. The clients can get details about resources consumed and resources available in the federation. Each federation member has their own policies defined. And the communication between members & allocation manager, authorization, and other management is carried out by plugins by keeping the policies and security of private clouds in mind.

A successful cloud database management system should be able to achieve certain goals such as availability, multi-tenancy, security, load balancing, and fault tolerance. Bashir Alam et al. proposes a 5-layered architecture of a Cloud Database Management [3] System typically includes the following layers:

- Client Layer: This layer is the outermost layer with which the user interacts. It is responsible for verifying a user. It contains the APIs to interact with the system. This layer generates the output a user wants for a particular issue. Security and transparency are important at this layer of the system.
- Middleware Layer: It acts as a layer between the client layer and the database layer. It is responsible for the authentication and

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

authorization of the user. It can be used for scalability and improvement of the system

- Database layer: This layer includes the database management software and the databases themselves. This layer is responsible for managing the data, ensuring data consistency, and providing data access services. This layer consists of programming techniques, query optimization, and processing, and security. The problem with this layer is finding a suitable database solution and query language for the application.
- Physical middleware Layer: This layer contains the operating system and it should ensure that the application is running smoothly and the way it is intended to work on all machines.
- Physical Layer: This layer includes the physical hardware and network infrastructure that support the cloud database. This layer provides the resources that are needed to run the cloud database, such as computing power, storage, and network connectivity. It is responsible for monitoring, scalability, and resource allocation.

Persist [4] is a middleware that allows multi-tenant SaaS applications to use more than one database or storage system according to their requirements. It combines public, private, and on-premise cloud storage to form a single platform for federated cloud storage. This system reduces the complexity of cloud storage by externalizing it. The storage management is done based on the SLA files of every tenant. Additional configuration files are provided by tenants to store the data dynamically based on their preferences. It allows tenants to design or reconfigure policies at runtime based on their confidentiality, privacy, and other needs. This system gives a performance overhead over other federated cloud storages and implements complex policies which are cost and performance efficient. Various SaaS applications have multiple storage needs based on the data. Configurations/policies provided by the tenant are overridden by the one pre-defined by the system/application. Data migration or switching between providers is done based on availability, performance, cost, and other factors. API abstraction is provided to hide the complexity of architecture and show only the required information. The future scope of this system is to support more dynamic properties at runtime and also provide static policies based on different providers.

The research paper [5] proposes a policy-driven data management middleware for multi-cloud storage in multi-tenant Software-as-a-Service (SaaS) applications. In such applications, data is distributed and replicated over multiple cloud storage systems, which differ in terms of supported data models, development APIs, performance, scalability, availability, and durability. The proposed middleware makes abstraction of multiple cloud storage technologies and providers follow a policy-driven approach for making data placement decisions. It also provides tenant customization support, allowing tenants to define storage configurations and data storage policies. The prototype implementation of the middleware is validated and evaluated in the context of a realistic multi-cloud SaaS application, with performance benchmark results indicating that the benefits of the proposed middleware can be achieved with acceptable overhead. The paper highlights the challenges of managing a multi-cloud storage architecture in practice and explains how the proposed middleware addresses these challenges. The paper also discusses the benefits of using a combination of different cloud storage technologies and providers, in contrast to relying on a single provider, which comes with the risks of technology, provider or vendor lock-in, and concerns about provider reliability, availability, scalability, and performance guarantees. The proposed middleware can enable cloud providers to leverage the benefits of a multi-cloud setup while addressing the complexity of configuring and operating it, by making abstraction of different cloud storage technologies and providers and allowing for tenant customization.

Nebula [6], a middleware can query several relational databases. Reasons that nebula's approach is faster and less expensive. The creation of Nebula was inspired by the historical query decomposition technique and work on multi-objective query optimization. Their quoting procedure starts by decomposing a multi-cloud query Q into a directed acyclic graph (DAG) GQ = <V, E>: each vertex v ∈ V models a sub-query involving a maximal amount of clauses for a given combination of providers; each directed edge e ∈ E represents dependencies between sub-queries. The paper states that to overcome the inherent limits of the exhaustive search, moving towards integration of reinforcement learning techniques to solve the Join Order Problem could be an inspiration. Transcending the relational model to offer support for heterogeneous data sources, in order to push the polystore systems in a multi-cloud environment, is an exciting perspective at a time when diversity is the rule for public data.

Densely populated cities increase energy loads, water, buildings, public places, traffic, and more things. Smartphones, sensors, and RFIDs are used as real-world user interfaces in smart city technologies, which support cloud- and Internet of Things (IoT)-based applications. The intelligence of a city describes its ability to gather all its resources, achieve its goals, and accomplish them efficiently and smoothly. The paper [12] addresses the convergent domain of cloud computing and IoT for any smart city application deployment. The use of an IoT-based framework for the healthcare system is discussed. IoT and cloud computing together can help in digitizing patient information, which can then be accessible to doctors or healthcare personnel anywhere in the world. The framework helps in minimizing costs and optimize the management process. The data transfer from one place to another will become easy as the data will be stored on the cloud. The main challenge lies in standardizing a large amount of data along with its management, handling, and distribution.

The paper [13] presents the concept of using cloud-based intelligent car parking services in smart cities as an important application of the Internet of Things (IoT) paradigm. The various issues faced by car drivers while parking their cars are evident as the existing car parking systems do not provide efficient services. The application is spread across various layers such as the sensor layer which consists of various sensors used in the system, the communication layer, and the application layer. The various car parking areas are identified. A middleware is developed for university campuses and describes various software solutions to provide the 'best' car parking service experience to their users.

For end-to-end cloud-fog communications involving smart devices and cloud-hosted apps, the paper introduces flexible IoT security middleware. The middleware [14] is made to function with devices that have limited computational, memory, energy, and network bandwidth as well as intermittent network access. The "Optimal

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

Scheme Decider" algorithm allows the middleware to choose the best end-to-end security scheme option that fits with a given set of device constraints, and the "Session Resumption" algorithm is used to reuse encrypted sessions from the recent past. By utilizing static pre-shared keys (PSKs), the middleware implementation also offers quick and resource-conscious security for a variety of IoT-based application requirements that require a choice between better security and faster data transfer rates.

The solution [15] for Big Data management in PaaS Cloud federation allows providers to benefit from SQL-like advantages and develop efficient services without divesting their legacy systems. It requires a two-layer hierarchy of storage, with each Cloud storing data in its own local SQL-like DB and a Global XML-like NO-SQL distributed database spread over all the Cloud providers. The proposed two-layer architecture improves the scalability of the data access service and reduces the amount of data available in the federation. To enable the PaaS federation, each Cloud system must include five components: a Middleware, a Sensor Observation Service (SOS) Agent, a local DB, and a Global DB. The two-layer architecture for IoT in Cloud federation enables clients to gather and deliver sensed data according to two approaches: on-demand interaction and event-based interactions. The event-based interaction is supported by the SAS Agent and provides sensed data to the client according to a publish-subscribe model. Data can be classified in Subscription Offerings according to different features, such as type of observation, covered area, monitoring device, and so on. The main concept of Offering ID is introduced by the SWE specifications and the management of such identifiers is out of the scope of this paper. The Global DB and MOM allow federated Clouds to build up a communication system based on message exchange.

Cloud storage services for cloud data storage are generally priced on two factors: how much data is to be stored and for how long. Cloud service providers are vulnerable to various threats, such as Denial of Service attacks and single point of failure. Availability of data is also affected if the cloud service provider runs out of business. The three clients (C1, C2, and C3) who saved their data on three distinct service providers (CSP1, CSP2, and CSP3) are the most crucial information in this article [17]. If a failure occurs at CSP1, all C1's data which was stored on CSP1's servers will be lost and cannot be retrieved. To ensure better availability of their data, the user will seek to store their data at multiple service providers to ensure better availability. Colluding cloud service providers are also a threat, as they may collude together to reconstruct and access the user-stored data. The authors provide the idea of distributing the data among two storage clouds such that an adversary cannot retrieve the contents of the data without having access to both storage clouds. This scenario is passive, as the cloud user cannot detect that their information has been collectively retrieved from the service providers without their consent.

CYCLONE [22] is an open-source middleware stack that simplifies the deployment and administration of cloud-based applications across multiple cloud platforms. It consists of a deployment manager, functional identity federation, and a network manager to manage software-defined networks, application deployment and management, and authentication and authorization based on federated identities. The middleware aims to simplify access for institutional users and assist DevOps teams in resolving issues in multi-cloud environments. The bioinformatics use case demonstrates the potential benefits of using CYCLONE from a single virtual machine installation to a multi-cloud infrastructure for cutting-edge genomic resources.

IV. PROPOSED MODEL

Dynamically adapting the system behavior requires an architecture that provides active cloud SLA processing capabilities and supports SLA updates at run-time. Figure 1 presents the overall architecture which consists of i) the User Layer, ii) the IoT system Layer iii) Frontend Application Layer, iv) Middleware Layer, and v) Cloud Layer. The User Layer represents different departments of the smart city. The IoT system layer represents the IoT systems that are in the Smart City. These IoT systems collect data and send it to the middleware using API.

The Frontend Application Layer takes the User SLA configuration and sends it to the middleware for further operations.

The Cloud Layer provides a uniform API which underneath consists of several database-specific drivers for different backend storage systems operating at different cloud storage providers. However, the core of the middleware and focus of this is the Middleware Layer which is described in detail in the rest of this section. We mainly focus on the roles of different components of the Middleware layer and how they efficiently manage the middleware operations.

The Middleware layer provides cloud selection and adaptation capabilities for responding to changes at run time and meeting different SLA requirements specified by each user. The layer comprises seven components: i) Data Access Component, ii) User SLA Management, iii) Cloud Management, iv) Cloud SLA Management, v) Deploy Monitoring, vi) Update Management, and vii) Deploy Scripts. This section further discusses the role of each component of the middleware layer with respect to the scenarios discussed below.

The Data Access Component (DAC) is responsible for getting the data files that are provided by the IOT system. The API through which the data is transferred to the DAC also carries the SLA ID. Through this ID Data Access Component can get the SLA details from the *User SLA Management* Component. These details contain the Cloud ID to which the data is to be stored. After getting the Cloud ID, the DAC gets the Cloud details from *Cloud SLA Management components* like cloud domain (e.g. AWS, IPFS) and cloud name (e.g. S3 standard). The DAC sends the Cloud and data file information to the *Upload Monitoring* Component. After the data is stored in the cloud the URL to the data is returned to DAC. This URL is saved in user logs with the date of data storage and the data file's original name. This is done so that in case the user needs to access the files at any given point, it can be fetched and provided to the user in the minimum time possible.

The User SLA Management Component (USM) is responsible to collect the user specifications. This contains the features the user requires and the minimum quality of the features (e.g. let's say a user requires high security for his data, but as the data quantity is little the need for storage capacity is very low). Table 2 shows an example of user requirements for security, storage capacity needed, and cloud Access needed. After getting the user requirements the information is

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

sent to the Cloud *Filtering* component where users can select the cloud which satisfies them in features as well the service cost. After selection, the selected cloud's cloud id and the cost the user will need to pay to the CSP are stored in the user SLA agreement. These details are stored in USM. Furthermore, users can change the requirements in SLA at any point in time and also change the CSP where the data will be stored.

Table 1. Overview of multi-cloud scenarios and their expected adaptation actions.

#	Scenarios	Actions
1	User Creates an SLA with specifications of minimum security, Access, and storage size	Middleware should filter out clouds that match the minimum criteria and show it to the user in sorted order with respect to the cloud service cost.
2	The user makes some changes to the SLA specification or changes the cloud provider from CP1 to CP2 for the SLA-specific data storage	Middleware should change the setting for the future data storage request and migrate the existing data from CP1 to CP2.
3	Cloud Provider CP1 changes some features in the storage service or changes the cost of the service	1. Middleware should check if the changes made in the cloud service are above the user SLA specifications. If they do not satisfy user SLA requirements, then middleware should inform the user about the changes to the cloud. 2. If the cost of the service of CP1 increases the user should be provided with the new cost of service.

The Cloud SLA Management Component (CSM) is responsible for storing the Cloud SLAs. This contains the features and their quality the CSP is providing with the cost for this service. Table 3 shows an example of how a CSP can show its cloud features and the cost of the service it provides. The Cloud has access to change the feature they provide or the cost of its cloud during runtime. For this paper, we are going to use 2 CSPs: AWS and IPFS. CSM is also responsible to notify users of any change in the cloud SLAs. Any change in the Cloud SLA which may affect the user will be notified (e.g. the CSP1 can fulfill the user requirements at less cost than the user's current CSP).

Table 2. Example of User Requirements.

Feature	Requirement
Security	High
Storage Capacity	High (200 GB)
Access	Medium (hourly bases)

Update Management Component is responsible to transfer the user data from the original cloud to the new cloud the user has changed to. It will fetch the data from the URL of the original CSP and transfer this data with the cloud details like cloud domain (e.g. AWS) and cloud name to the *Deploy Monitoring* component. After saving the data to the cloud the new URL will be replaced with the previous URL from the user logs.

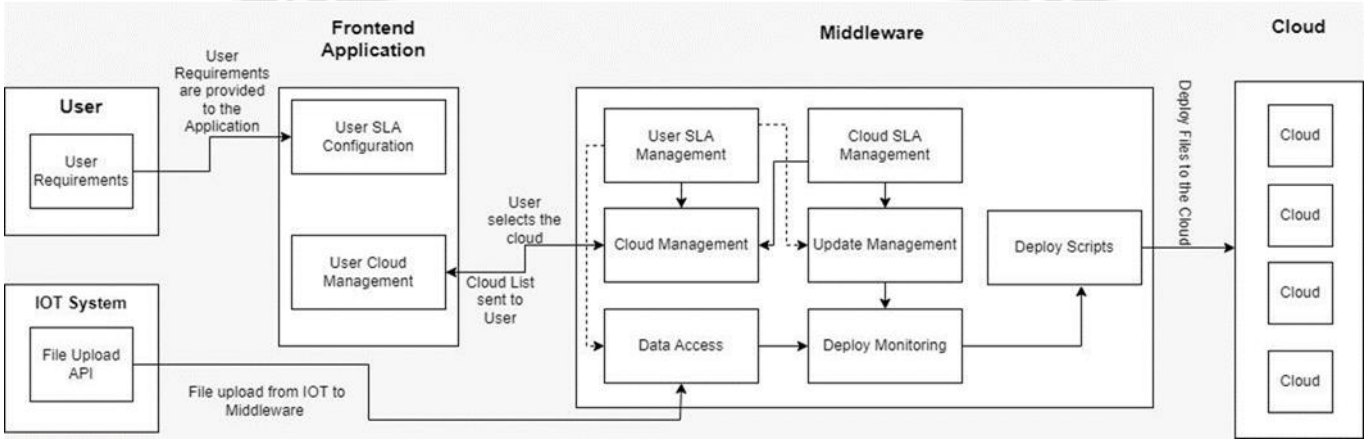

Fig. 1. Middleware Architecture

Cloud Management Component (CM) is responsible to filter clouds based on the user SLA provided by the *User SLA management* component. The CM takes user requirements and gets all the cloud SLA which can provide features required by the user or even better. After getting all these cloud SLAs, they are sorted with respect to the cost they charge and given to the user to select from. After the user selects the cloud, they are most satisfied with it. The cloud SLA Id is passed to the *User SLA Management* to store. If any updates occur (e.g.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
Article Received: 05 March 2023 Revised: 24 April 2023 Accepted: 08 May 2023

User decides to change SLA requirements or change CSP) the *Cloud Management* component once again repeats the process of cloud selection.

Table 3. Example of features provided by the cloud

Cloud Service Provider	Security	Storage Capacity	Access (Read & Write)
AWS - S3 Standard	High	0.02 $/GB	High (frequent read and write)
IPFS	Content is public	0.08 $ / GB	Medium (frequent read, rear write)
Atlas	High	0.25 $/ GB	High (frequent read and write)
Microsoft Azure	High	0.018 $/ GB	High (frequent read and write)
Google Cloud - Standard	High	0.02 $/GB	High (frequent read and write)

Deploy Monitoring Component is responsible to get information about the cloud and the data which will be stored in the said cloud. With these details, *the Deploy Monitoring* component will fetch the cloud keys (e.g. API_KEY for IPFS storage), and pass it to the Deploy Scripts component with the files to store in the cloud. After saving the file the URL will be fetched from the Deploy script and transferred to the *Data Access* component or the *Update Management* component depending upon which had sent the request to store the data

Deploy Scripts Component is responsible for storing the files in the cloud. It uses the Keys provided by the *Deploy Monitoring* component to access the cloud and store the files in the cloud with their respective APIs. After uploading the files to the cloud. The URL received from the cloud is returned to the Deploy Monitoring component.

The middleware has the following application flow for creating new SLA: (see Table. 1 # 1)

- The web Frontend takes user inputs in an HTML form(name, cloud requirements, etc) and sends them to MongoDB via NodeJS. A new document is created in the database and the user is provided with an API to connect to the IOT system.
- User selects a cloud from the recommended CSPs provided based on the user requirements.
- The IOT system can use the API to upload files to the middleware. The API contains the SLA ID, through which middleware gets the details about the cloud to which the files will be stored.
- After the cloud details are fetched, the files are provided to the deploy script of the particular CSP with the required API / credentials.
- The files are uploaded to the CSP, and the returned link is saved in a map stored in the user database in addition to the date and time of upload.

The middleware has the following application flow for fetching the files from the cloud:

- The web frontend fetched the user details which contains a map data structure that holds links to user files uploaded to the cloud.
- Users can filter out the files based on the file name, or upload date.
- The backend fetches the files from the cloud using CSP credentials(in case the file is protected) and passes it to the web frontend.
- Users can download/view the file directly using the web frontend.

The middleware has the following application flow in case there are updates in the CSP features.(see Table. 1 # 3)

- The web frontend gets the update and passes it to the database. The database(CSP SLA) is updated with the new information.
- The updates are compared with the user SLA requirements connected to the CSP.
- In case a user's SLA requirements are compromised after the update. The user is notified to change their CSP through email. Otherwise, the user is just notified of the CSP updates.
- After the user changes the CSP. The data is fetched from the original CSP and uploaded to the new CSP. The old cloud link is replaced with the new one in the database. This process is carried out by the middleware without any human intervention.

The middleware has the following application flow in case the user wants to change the CSP or update the SLA. (see Table. 1 # 2)

- The web frontend gets the update and passes it to the database. The database(user SLA) is updated with the new information.
- In case of an SLA update. The updates are compared with the user CSP SLA. In case a user's SLA requirements are compromised after the update. The user is notified to change their CSP through email.
- If the user changes the CSP, The data is fetched from the original CSP and uploaded to the new CSP. The old cloud link is replaced with the new one in the database.

V. IMPLEMENTATION AND RESULTS

The machine used in running the middleware had a processor of Intel i3 10th gen with 3.60GHz. We implemented the above model using the NodeJS framework. We used MongoDB to store user details, and details of user SLAs and cloud SLAs, and to connect NodeJS and MongoDB servers we used the Mongoose package. Other than that, sending notifications to users about any change in cloud SLAs was done using mail. We used the node js package nodemailer to send automatic emails to users. This middleware was specifically designed to take input from IoT systems, especially from IOT clusters like Smart City, hence it can be used by multiple users of different city

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

departments to input their data to the cloud using a single system (e.g. traffic data, AQI data, and Weather report data.)

Cloud SLA Schema

```
cloudName: String,          => CSP Name
security : Number,          => Feature provided by CSP
storageCapacity: Number,    => Feature provided by CSP
bandwidth: Number,          => Feature provided by CSP
price: Number,              => Cost of Storage
cloud_id: String            => Unique Id of Cloud SLA
```

We created four database schemas for storing user details, user SLAs, cloud SLAs, and user logs respectively. We made it so that a single user can create multiple SLAs. User details contained their login and password details. When a user creates an SLA, it needs to be connected to a cloud so that the middleware can take data files as input. When the clouds are filtered out based on user requirements the final provisional cost is given to the user based on how much storage he needs and at what rate the CSP provides that much storage capacity (e.g. S3 Standard can give 200 GB of storage space for 4$). After the user selects a cloud from the recommended clouds, it will generate an API for that specific SLA. Users can use this API in their IOT system to directly send files generated by the IOT system to the middleware without any human interaction. The middleware can take multiple files with a single API call in the format of form data and these files can be of the same type or different (e.g., 5 CSV files uploaded together or 4 CSV and 1 JSON file together). After processing by the Data Access component, the files will be saved to the cloud one after the other. As a result, we will get the same number of URLs as the files uploaded. These URLs will be saved to the user log database with the file's original name and the date and time at which they were uploaded for easy access and retrieval.

User SLA Schema

```
dataDescription: String,    => For user
security : Number,          => User Requirement
storageCapacity: Number,    => User Requirement
bandwidth: Number,          => User Requirement
itemsStored : Number,       => details of data stored on cloud
user_id : String,           => Unique Id for user SLA
cloudSLA: String            => unique Id of cloud the user
selected
cloudPrice: Number          => Cost of storage
```

The whole process of accessing files from IOT to upload to the cloud is very quick as it does not require any complex computation which may increase the final time. We created a fronted application so that users can access their accounts, deploy SLA, and upload Logs. If required, users can make changes to their SLAs using the fronted application. In case of a change in CSP, the update manager gets all the URLs from the user Log which are related to the SLA for which the CSP was changed. After which the update manager will fetch these files and send them to deploy scripts via deploy monitoring to be uploaded to the new CSP. The URLs of the new CSP will be replaced in the user logs without any change in date or the file name so as to maintain the user record. When a new cloud is connected to the middleware, first its API to upload docs to the cloud is added to the deploy scripts then the Deploy monitoring component is updated with the cloud domain and cloud name (e.g., the domain can be AWS and the cloud name can be the S3 Standard / S3 Glacier of the AWS S3 instance). After which the SLA provided by the CSP is added to the database. In case the newly added SLA can give better cost for performance to some users, these users are notified via mail with the new SLA details. If the user decides to shift to the new CSP, he/she can update the user SLA's settings using the frontend application.

User Logs Schema

```
Log Id: String,       => Unique Id of Log
User Id: String,      => User Id of Log Owner
Entries: Array        => All entries of file uploads to cloud
        {
        file name: String,      => File Name
        URL: String,            => URL of uploaded file
        date: Date,             => Date to upload
        SLA Id: String          => SLA id through which file was
                                   uploaded
}
```

For this paper, we used mainly 5 types of clouds: Amazon Web Service, IPFS decentralized Storage, Atlas Cloud, Microsoft Azure, and Google Cloud. For testing the middleware, we uploaded 5 CSV documents to each cloud and tested the time taken by each cloud to store the data on the cloud.

Figure 2 shows an example of CSV upload to the AWS S3 bucket, Figure 3 shows the average time to fetch URL from the cloud. Fetching files from IoT System API took approx. 20ms and storing cloud URLs to the database and some database write operations took approx. 38ms. When trying to upload to AWS the total time from accessing data to storing it on the cloud took approx. 2.5 seconds (Figure 4 shows the time needed to upload 5 CSV to Different Clouds). For AWS we used aws-sdk[31] npm package to access the cloud. Fetching the file from the AWS cloud URL took an average of 1 second.

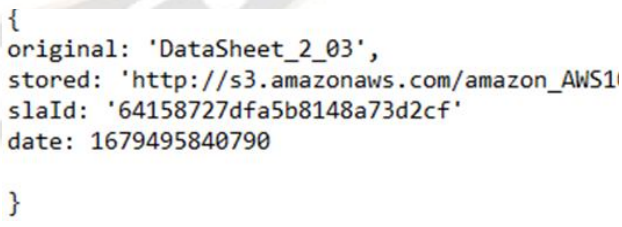

Fig. 2. Upload CSV to AWS S3 Standard

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

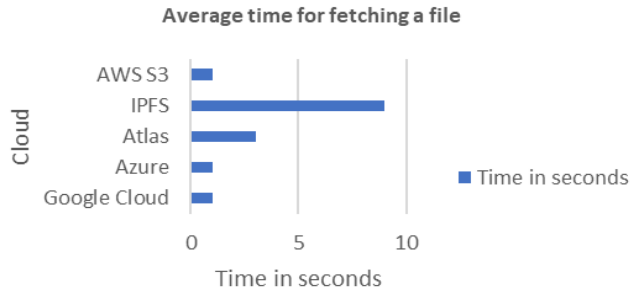

Fig. 3. Average time to fetch a file from the cloud

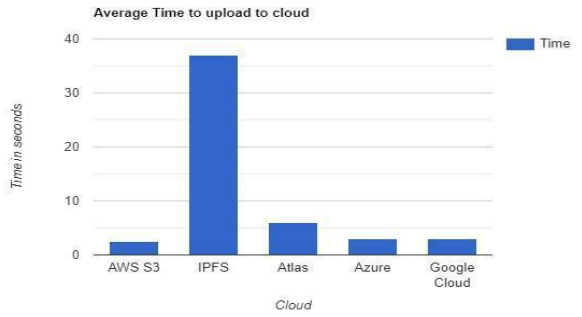

Fig. 4. Time needed to upload CSV

IPFS took pprox.. 38 seconds total time to upload 5 files to IPFS storage. For IPFS we used nft.storage gateway[32] to access the decentralized cloud. Fetching the file from the IPFS cloud URL took an average of 9 seconds.

Atlas Cloud took pprox.. 7 seconds total time to upload 5 files to Atlas. For Atlas, we used MongoDB GridFS[33] to store files to the cloud. Fetching the file from the Atlas cloud URL took an average of 2.5 seconds.

Microsoft Azure Cloud took pprox.. 3 seconds total time to upload 5 files to Atlas. For Azure, we used the azure -storage-blob[34] npm package to store files in the cloud. Fetching the file from the Azure cloud URL took an average of 1 second.

Google Cloud took pprox.. 3 seconds total time to upload 5 files to GC. For Google, we used the @google-cloud/storage[35] npm package to store files in the cloud. Fetching the file from the Google Cloud URL took an average of 1 second.

Table 4 compares the time needed to store 5 CSV in the cloud. The retrieval time for clouds was pprox.. same which was in milliseconds. For security, AWS has options to allow reading or writing on the file. But on the IPFS cloud anybody who has the CID of the file can access the file without any permissions, but writing to the file is impossible.

Table 4: Comparison of storage time on Different Clouds

Cloud Service Provider	Data Access time from Cloud	Upload To Cloud (5 CSVs)
AWS S3 Standard	~ 1 sec	~2.5 sec
IPFS (nft.storage)	~ 9 sec	~37 sec
Atlas	~ 2.5 sec	~6 sec
Microsoft Azure	~ 1 sec	~ 3 sec
Google Cloud - Standard	~ 1 sec	~ 3 sec

VI. CONCLUSION

This study looked at how the IoT industry might use government clouds, with a focus on the smart city sector. The study shed light on the capabilities of this technology by investigating the benefits of using federal clouds and how they can be used to improve operational efficiency and decision-making procedures. The proposed middleware architecture has been discussed along with its seven components: Data Access Component, User SLA Management, Cloud Management, Cloud SLA Management, Deploy Monitoring, Update Management, and Deploy Scripts. A comparative study of the capabilities of various cloud service providers has been undertaken which determines the right CSP for the user. We have analyzed the cloud service providers based on data access time and upload time of files. The cloud service providers used were AWS S3 standard, IPFS, MongoDB Atlas, Microsoft Azure, and Google Cloud. As per our observations, IPFS takes the longest time both for upload and retrieval of files whereas AWS S3 takes the shortest time for upload and AWS S3, Azure and Google Cloud take nearly the same amount of time for retrieval of files.

Furthermore, the proposed application eliminates the need for technical expertise, making it accessible to a broader range of stakeholders and simplifying the cloud decision process for non-technical users. The results have shown the potential for a symbiotic system that improves the capabilities of both technologies by connecting federal clouds and IoT technology.

VII. FUTURE SCOPE

The current system uses only a specific number of cloud providers. In the future, more cloud service providers can be considered to provide more versatile services to the user. Though now it only targets IoT devices, in the future other kinds of applications can be also considered, and the system can be expanded to include other systems. Security is one of the most critical concerns for organizations that use cloud computing. Federal Cloud Middleware has the potential to improve security by providing a unified interface for managing multiple cloud providers. Federal Cloud Middleware can help organizations to scale their data storage requirements based on demand. It can dynamically allocate resources to different clouds based on the user's SLA, ensuring that data is always available and accessible. AI can be used to analyze data stored in different clouds and provide insights that can help organizations make better decisions. Federal Cloud Middleware can enable multi-cloud analytics, allowing organizations to analyze data stored in different clouds using a single interface.

REFERENCES

[1] Soojin Park, Sungyong Park, (2019). 'A cloud-based middleware for self-adaptive IoT-collaboration services'. Sensors 2019, 19(20), Published: 20 October 2019

[2] Abmar Barros, Francisco Brasileiro, Giovanni Farias, Francisco Germano, Marcos Rios Nobrega, Ana C. Ribeiro,

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

Igor Silva, Leticia Teixeira, 'Using Fogbow to federate private clouds.', January 2015.

[3] Bashir Alam, M.N. Doja, Mansaf Alam, Shweta Mongia, '5-Layered architecture of cloud database management system', 2013 AASRI Conference

[4] Ansar Rafique, Dimitri Van Landuyt, Wouter Joosen, 'PERSIST: Policy-Based data management middleware for multi-tenant SaaS leveraging federated cloud storage', 6 February 2018

[5] Ansar Rafique, Dimitri Van Landuyt, Bert Lagaisse, and Wouter Joosen, 'Policy-Driven data management middleware for multi-cloud storage in multi-tenant SaaS', 2015 IEEE/ACM 2nd International Symposium on Big Data Computing

[6] Damien T. Wojtowicz; Shaoyi Yin; Franck Morvan et al. 'Cost-Effective dynamic optimization for multi-cloud queries' 2021

[7] Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A., 'A review of auto-scaling techniques for elastic applications in cloud environments.' J. Grid Comput. 12(4), 559–592 (2014)

[8] K. Yang and X. Jia. An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE Transactions on Parallel and Distributed Systems, 24(9):1717–1726, Sept 2013

[9] Dan Dobre, Paolo Viotti, and Marko Vukolic. Hybris: Robust ´ hybrid cloud storage. In Proceedings of the ACM Symposium

[10] Y. Singh, F. Kandah, and Weiyi Zhang. A secured costeffective multi-cloud storage in cloud computing. In 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 619–624, April 2011.

[11] "Towards an Adaptive Middleware for Efficient Multi-Cloud Data Storage", Ansar Rafique, Dimitri Van Landuyt, Vincent Reniers, Wouter Joosen.

[12] Maninder Jeet Kaur and Piyush Maheshwari, 'Building smart cities applications using IoT and cloud-based architectures', Department of Engineering, Amity University Dubai.

[13] Zhanlin Ji, Ivan Ganchev, Máirtín O'Droma, Li Zhao and Xueji Zhang, 'A cloud-based car parking middleware for IoT-based smart cities: design and implementation', 25 November 2014

[14] Bidyut Mukherjeea, Songjie Wangb, Wenyi Lua, Roshan Lal Neupanea, Daniel Dunna, Yijie Rena, Qi Sua, Prasad Calyamb, 'Flexible IoT security middleware for end-to-end cloud-fog communication', Department of Electrical Engineering and Computer Science, University of Missouri-Columbia, MO, USA

[15] Maria Fazio, Antonio Celesti, Massimo Villari and Antonio Puliafito, 'The need of a hybrid storage approach for IoT in PaaS cloud federation' 2014

[16] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13, pages 247–258, New York, NY, USA, 2013. ACM.

[17] Yashaswi Singh, Farah Kandah, Weiyi Zhang, 'A secured cost-effective multi-cloud storage in cloud computing' IEEE INFOCOM 2011

[18] lysson Bessani, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa. Depsky: Dependable and secure ´ storage in a cloud-of-clouds. Trans. Storage, 9(4):12:1–12:33, November.

[19] Thanasis G. Papaioannou, Nicolas Bonvin, and Karl Aberer. Scalia: An adaptive scheme for efficient multi-cloud storage. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pages 20:1–20:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[20] Ansar Rafique, Dimitri Van Landuyt, Bert Lagaisse, and Wouter Joosen. Policy-driven data management middleware for multi-cloud storage in multi-tenant saas. In 2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC), pages 78–84. IEEE, 2015.

[21] "Data Quality Management in the Internet of Things", by Lina Zhang ,Dongwon Jeong, and Sukhoon Lee

[22] Mathias Slawik, Yuri Demchenko, Fatih Turkmen, Alexy Ilyushkin, Cees de Laat, Christophe Blanchet, Charles Loomis, 'CYCLONE: The multi-cloud middleware stack for application deployment and management', 2017 IEEE 9th International Conference on Cloud Computing Technology and Science

[23] SeClosed. Secure, cloud-based storage and processing of sensitive documents. http://www.iminds.be/en/projects/SeClosed, 2016. [Last visited on November 23, 2016].

[24] Mell, P., Grance, T., "The NIST Definition of Cloud Computing". February 18, 2016

[25] Bermbach, D., Klems, M., Tai, S., Michael, M. "Meta Storage: A federated cloud storage system to manage consistency-latency tradeoffs". In: IEEE International Conference on Cloud Computing (CLOUD), 2011, pp. 452– 459. IEEE (2011)

[26] M. Fazio, M. Paone, A. Puliafito, and M. Villari, "Huge amount of heterogeneous sensed data needs the cloud," in International MultiConference on Systems, Signals and Devices (SSD 2012), (Chemnitz, Germany), March, 20-23 2012.

[27] S. Dey, A. Chakraborty, S. Naskar, and P. Misra, "Smart city surveillance: Leveraging benefits of cloud data stores," in IEEE 37th Conference on Local Computer Networks Workshops (LCN Workshops), pp. 868–876, 2012.

[28] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," Future Gener. Comput. Syst., vol. 28, no. 2, pp. 358–367, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.07.003

[29] A. Amato and S. Venticinque, "Multi-objective decision support for brokering of cloud sla," in The 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013). Barcelona, Spain: IEEE Computer Society, March 25-28 2013

[30] A. Amato, B. D. Martino, and S. Venticinque, "Evaluation and brokering of service level agreements for negotiation of cloud infrastructures," in ICITST, 2012, pp. 144–149

[31] "AWS SDK for JavaScript", Available : https://www.npmjs.com/package/aws-sdk

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6754
***Article Received**: 05 March 2023 **Revised**: 24 April 2023 **Accepted**: 08 May 2023*

[32] "nft.storage", Available :https://www.npmjs.com/package/nft.storage
[33] "MongoDB GirdFS", Available: https://www.mongodb.com/docs/manual/core/gridfs/
[34] "Azure Storage Blob client library for JavaScript", Available: https://www.npmjs.com/package/@azure/storage-blob
[35] "Google Cloud Storage Node JS Client", Available : https://www.npmjs.com/package/@google-cloud/storage