Paper DH06

FAIRchive - Cloud-based and GxP Compliant Long-Term Archiving for Clinical Trial Data

Frank Stoinski, Entimo AG, Berlin, Germany
Alexander Lüders, Entimo AG, Berlin, Germany

ABSTRACT

Archiving clinical trial data is the final step in the clinical trial process. The data and documentation generated during the course of a clinical trial must be archived and made accessible in a secure and validated environment for the prescribed retention period. This can be particularly difficult and costly as the commonly used data formats and software products may change over time. In addition, archives reveal a particular pattern of use where data is dormant for long periods of time until it is retrieved and reviewed again. This is where the cost factor comes into play, as ideally archiving solutions should only generate low costs for operation and maintenance while the data is at rest. Today's cloud providers support serverless billing models, which could provide a particular advantage here. This paper examines how Amazon Cloud technologies can be used to build a validated GxP-compliant archive system for clinical trials, and what the advantages are compared to a self-hosted archive.

INTRODUCTION

Systems for statistical data analysis and the creation of reports are the key elements of collecting and processing clinical data to submit to regulatory authorities. In the course of their work, such systems collect hundreds of gigabytes of data that have to be persisted within the system for later tracking and tracing. But every system eventually reaches a time when it is replaced by a newer system, or is decommissioned without a replacement. Regulations require that data saved in such a system, to the extent that they are not migrated to a successor system, must be available and retrievable even beyond the life-cycle of the old system. Furthermore, it must be possible for the data of an older system to be made available for further use. These requirements are described by the FAIR principle [1], which states that such archived data should be findable, accessible, interoperable and reusable.

This paper describes an archival solution, under the name FAIRchive, for an existing data analysis and reporting environment where the archival solution is implemented in AWS, a comprehensive cloud computing platform that provides on-demand IT resources over the internet. The paper discusses the advantages and disadvantages of such a solution compared with a self-hosted solution, as well as the various pitfalls that have shown up in the step-by-step development of the completed solution. An evaluation of the risks that are tied to particular decisions taken during the implementation process is also presented. Finally, we describe why this particular approach also works with large amounts of data and can be applied to other systems that are sources of clinical data.

THE CHALLENGE

In order to discuss an archive, it is first necessary to consider the important aspects of the systems that are the sources of the data. In our case, it is a GxP-compliant system that was used for more than 10 years in various use contexts with as many as 600 simultaneous users. The system is a classic client-server solution and implements functionalities for data analysis and reporting in connection with a sandbox architecture that allows an individual user to run programs in R and SAS controlled and isolated from other users. For the system's data management there are, in addition to a standard file system, a relational database and a version control system to retain historical versions. For the archiving it is thus necessary to manage three pools of data:

- the relevant data that is saved in the relational database,
- the physical file system that reflects the current state of the clinical data, programs, outputs and other documents, and
- the historical versions of files from the version control system.

In this case, the old system will be replaced by a modern cloud-based system. During the replacement, only a small share of the data from the old system will be migrated to the new system. In this instance, that will be the data that the users are actively working with at the time that the new system is brought into service. Data whose use is already completed at this time will be stored in the archive, which will be set up as a part of the migration. That means that for the planned archive we have to work with data from the three pools listed above. While the physical files and

their historical versions are important information that have to be saved in the archive, it is mainly the data from the relational database that will be regularly called up from the archive. To get closer to a solution for archiving this data, the data in the database were divided into logical domains. From our perspective, the domains do not belong specifically to the old system; instead, they can be found in similar or related forms in many other clinical analysis and reporting systems. The domains are:

- Object Search Domain – information about objects in the system and their changes, made by both users and automated processes, over time,
- Permissions Domain – information about users and their access rights within the system,
- Audit Trail Domain – audit trail data that is generated by use of the system,
- Action Log Domain – user interactions with the system, and
- Execution Trace Domain – information about program executions within the system.

These data domains that are important for the archive are one part of what are called “essential documents” [2]. In addition to the data domains listed above, a plethora of additional information in the source system exists, but which does not necessarily have to migrate to the archive. These are called “non-essential documents”. Examples of this category include configuration information such as workflows, descriptions of forms, or options for the management of the execution of R and SAS programs.

The starting point for the additional observations in this paper is a medium-sized source system that was used for more than 10 years. The number of rows in the relational database that contains data domains listed above and which must be transferred to the archive, can reach up to 60 million, depending on the domain. The physical file system at the time of archiving was expected to have a size in the range of 7 to 8 Tbyte. The file system would then have approximately 1 million files. Archive users will pose queries to the archive using the familiar paths from the old system to call up information like the following:

- How, by whom, and with which processes was an object from the old system changed over time?
- Which access rights did a particular user have at a given time in relation to an object or a hierarchical structure of objects?
- Which programs ran at which time in the old system, and which input and output objects were touched by a particular program execution?
- Find a file with particular characteristics and export it so that it can be imported to another system.
- Find files that belong together and have been supplied with a common attribute (a “tag”) and export this set of files to import into another system.

REQUIREMENT ON THE TARGET SOLUTION

At the beginning of considerations for a solution is the question of how the system will be used. From our perspective, an archival system will in fact be used regularly, but with a noticeably reduced intensity compared to the source system. There will be a small group of users who use the archive for regulatory queries and a larger group of users who consult the archive to evaluate files and export them for further use. It can thus be expected that the data in the archive must mostly be simply available, and that the access to a particular file will be relatively uncommon.

As with any new system being developed, the expected operating costs are a significant factor. Because of the archive’s reduced activity compared to the source system, it is desirable for the archive to generate only limited costs for just saving the data and keeping it available. Considering in advance the patterns of how the archive will be accessed is particularly important because different guaranteed access times for data in cloud environments is tied to varying costs for cloud users. In addition, the costs for the archive when it is not being used should be kept to a minimum. Systems that have been brought into service must be kept up to date over the course of many years for reasons of security and technological evolution. In this area as well, it is desirable for the system’s architecture to be put together in a way so that the owners only have to take a minimal number of components into their maintenance plan.

The data in the archive has to be kept over a span of decades. Data formats change, over the course of the years they are replaced by new formats, and the knowledge about older formats can be lost. This fact has to be kept in mind when a target format for the archival data is chosen. Ideally, the target format is sufficiently simple and self-explanatory that even decades from now it can be understood and processed. This is the case with open data formats, which ideally are still human-readable. On the other hand, it must be ensured that the selected data format allows the request to the archive to be processed efficiently. Here a good compromise must be found. Obviously, the format can only be based on technologies that are established and well understood at the time the archive is set up.

Despite the expected reduction in user activity, the archive system must still be GxP-compliant. That means that in addition to the challenge of transferring the data itself into a new representation and a new query system, it is necessary to ensure that the future archive has a GxP-compliant foundation that guarantees the integrity of the data and only allows access to the data by authorized users.

THE NAIVE APPROACH

The source system was used for many years as an on-premises system. Corresponding costs were incurred not only when the system was set up, such as the purchase of the necessary hardware, but also during operations. Archive systems typically operate for many years, sometimes many decades. Keeping such systems running, accessible and secure is a major challenge. If such a system is kept on-premises, crucial decisions must be made at the very outset regarding the scale and future maintenance of the hardware. Once made, decisions in this area can only be corrected at great cost. Modern cloud providers promise remedies, in that they limit the barriers to entry for execution platforms on an almost arbitrary scale and take over the operation of the necessary hardware resources. In particular, they offer various computing configurations that also support applications for which computing capacity is only occasionally required. The planned archive system is one of this type, given that the data are only occasionally accessed. In the time between requests, the system should be available but not necessarily using computing resources and thus generating costs. Furthermore, cloud providers support their customers by offering up-to-date execution environments that have all of the recommended security updates.

The choice for the realization of the archive went to Amazon and AWS. AWS offers a diverse array of possibilities for the configuration of and invoicing for the computing resources that are kept available for a software solution. In particular, the decision to do without dedicated machines, also known as “serverless computing”, led to decisive price advantages. Similarly, the various storage classes for objects in AWS offers opportunities to store data cost-effectively according to the patterns of how often they are accessed.

The idea of simply moving the source system’s relational database to the Amazon cloud and operating it there was rejected for cost reasons. On one hand, a dedicated machine with a running instance of the database on it would have had to be kept available, which entailed the corresponding usage-independent costs and a frequent monitoring for updates. On the other, this database required a connection to a fast file system of sufficient sizes. In AWS, that can be realized either with EBS Volumes or with storage maintained by AWS within the framework of the Amazon product “Amazon Aurora” [5]. After an initial estimate of the expected costs for both solutions, we came to the conclusion that we would need to seek other options for implementing the archive in AWS.

Because the direct transformation of the data from the source relational database into a different target relational database was rejected, the first task was to find a suitable interim format for the source data. A generally recognized, well understood and accepted standard in this area is JSON. JSON is an open and ASCII-based format that can be interpreted by many different tools. Furthermore, it can be expected that this format will still be understood and supported many years from now. The first step was thus to convert the relevant tables in the relational database to JSON. The structure of these JSON files corresponded as far as possible to the table structure of the original tables in the source database. The plan was to then convert the JSON data into an archive system that had yet to be defined. The identified relational tables in the source system were converted, via database jobs, into JSON and then written there.

As part of the search for a technical solution for the archive, additional services offered by Amazon were examined for the suitability for an archive system. One such product is Amazon Athena [6]. This service enables JSON files or files in Parquet format to be queried via SQL. The JSON files needed for that had already, in principle, been made available by the extraction of the relevant data from the source system. Building on that approach, an initial idea for an archive in AWS was developed. The JSON files were imported to Amazon S3 with an import script. Amazon S3 is a storage service from Amazon that allows storage of data of an arbitrary size with access to this data via a URL. Athena then offers the possibility of starting queries against JSON files in Amazon S3 via conventional SQL queries.

We followed this approach, exported the original tables from the source database into JSON and imported the resulting files into S3. For example, two tables were named that in the source system contained information about the objects and their corresponding logical versions. The table with the objects, which contained all of the entities in the system with which a user can interact, has approximately one million entries. The table with the corresponding versions contains 10 million entries. The JSON file that results from exporting the table of objects is approximately 500 Mbyte, and the JSON file for the table of object versions is 3.5 Gbyte. Both files were divided into line-based chunks and uploaded to Amazon S3.

For archiving both the physical files and their revisions, Amazon S3 again seemed like a possibility. Here the idea was that we could profit from the various storage classes in Amazon AWS with their correspondingly graduated costs [3]. The storage classes are used to define the requirements for the response time of stored data in answer to a query. The more seldom access to the data is needed and the more time that Amazon can allow, for example to retrieve the data from cold storage, the less costly the storage. Here it is less a matter of optimizing the queries and more a matter of making the expected pattern of access to this data concrete in order to derive the appropriate storage class. In our case, we presumed that the historical revisions of a file would seldom be queried, and that for such a query the user would be prepared to wait a certain amount of time for the data. As a result, the less expensive storage classes, such as S3 Glacier Flexible Retrieval or Amazon S3 Glacier Deep Archive, lent themselves to this approach. Access to the most recent state of the physical files would, by contrast, happen more often. For this use, it would be possible to select a more dynamic class of storage, such as Amazon S3 Standard – Infrequent Access. Because objects in S3 can be addressed with a URL, this approach offers the natural opportunity to enhance the object and version information with a link to the physical files and versions. By contrast the query tables themselves, which are also kept in S3 and thus require a storage class, will be used very often and must be immediately available. For these reasons, it is necessary to select the most expensive storage class, S3 Standard.

After an upload of the files, the first queries in Amazon Athena against the imported JSON archive data could be started. Initial tests showed that the system's reply behavior was not satisfactory. The reason is that Amazon Athena is not a database engine in the conventional sense, but only has an understanding about the structure of the queried JSON files and can answer SQL queries with the knowledge of this structure. Beyond that, there is no additional support or optimization of queries, such as indices. Instead, with every query Amazon Athena parses the corresponding JSON file from the beginning until it reaches an answer that matches the query. With Amazon Athena, the commonly dreaded full-table scan of a relational database is often more the standard than the exception. On one hand, queries against large JSON file could quickly exceed a period of 30 minutes, after which Athena simply stops the query. On the other hand, the amount of data scanned corresponds directly to the amount of costs incurred with Amazon. Queries that scan complete JSON files are thus considerably more expensive than those that can find an answer after just a few lines. Countermeasures are necessary, and our next goal was thus to improve the speed of queries in Amazon Athena against the archived data.

APPROACH RECONSIDERED

An initial solution that lay close at hand was to convert the JSON data into Parquet format, which Athena also supports queries against [4]. In this format, data is saved by columns, and thus optimized for many queries in Athena. If a query targets only a few columns of the table the Parquet format helps as the required data can be collected from the corresponding column data pool.

ID	Name	Age
1	John	24
2	Sue	45
3	Anne	33

ID	1	2	3
Name	John	Sue	Anne
Age	24	45	33

Figure 1: The Parquet format (simplified)

The conversion could take place directly in Amazon Athena using the JSON tables that are already at hand in S3. The existing JSON files for the various data domains were converted to Parquet format. This first optimization already showed improvement in the answering behavior. An additional positive effect was that the amount of data scanned per query decreased, which showed up directly in the costs for the Archive. Nevertheless, we looked for additional means to query the archive's large amount of data more efficiently. The only way to do that, it seemed, was to change the structure of the data in order to enable more efficient processing by the Athena engine.

Thus we had to transform the JSON tables that had been exported from the source data and imported into S3. The tables also included information that was not relevant for the archiving. Further, the data required multiple joins across multiple tables to deliver a result. Cleaning the data and transforming it into a new JSON structure promised additional improvements. We decided to preserve the unchanged JSON tables as the original state of the source system and to derive an optimized JSON structure based on the original data, which would then be used for queries. On one hand, this approach ensured that we could revise the transformation specifications at any time to create new optimized tables. On the other hand, it is possible that over the course of time new user requirements would arise

that would require making additional data from the original tables available for optimized queries.

The tables identified in the source system were exported for a second time, this time with Java-based transformers and PL/SQL scripts that converted the original data from the database into optimized JSON structures, each for the corresponding domain. As part of this process, various irrelevant attributes from the original data were discarded and information from neighboring tables was aggregated in order to reduce the need for joins across multiple tables during queries from Athena. Furthermore, this transformation made it possible to store references to the physical files and revisions in S3 in the new tables. The diagram below illustrates the process of extraction from the source system.

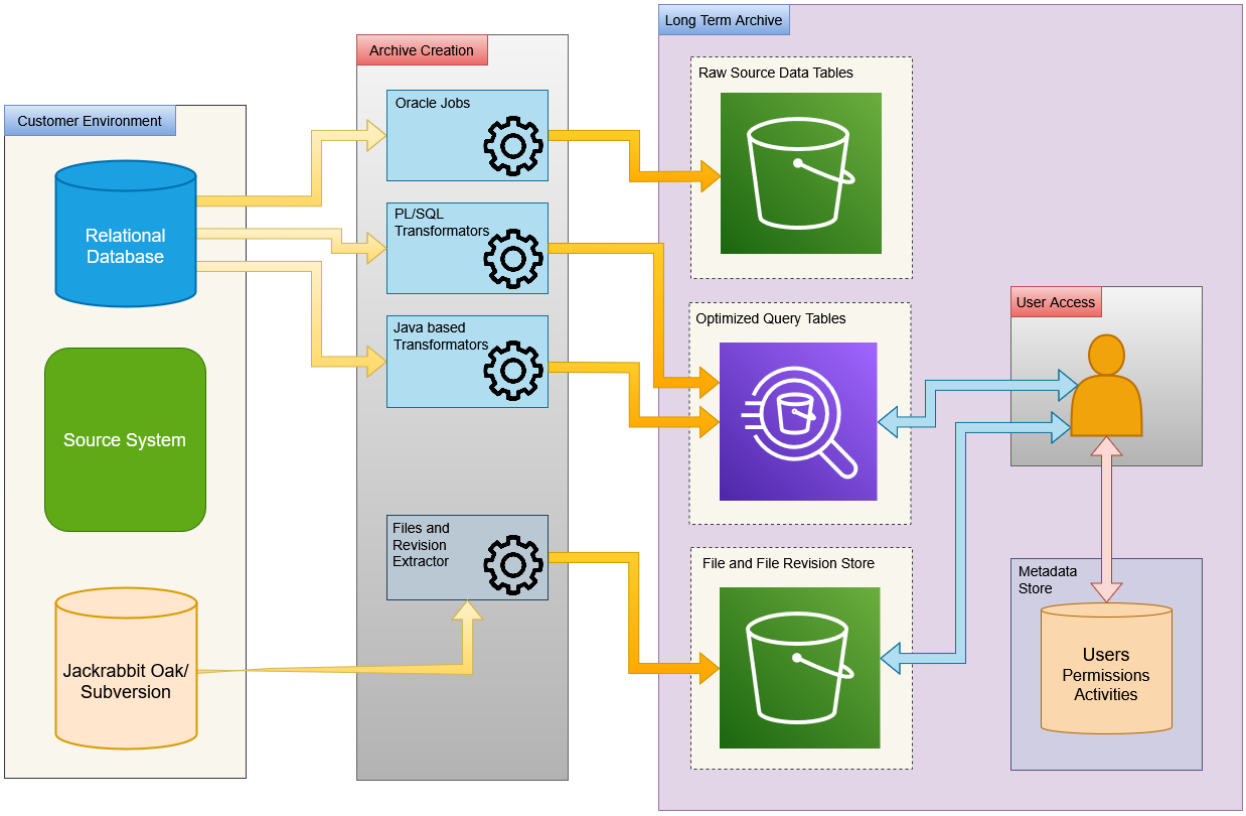

Figure 2: The Archive Creation Process

The optimized JSON tables produced by the extraction processes were then further converted into Parquet format. The new table structure, the avoidance of joins in the optimized query tables, and the use of Parquet format improved the archive system's answering performance for typical queries.

For the storage of the physical files and revisions in S3, in addition to the choice of storage class, the amount of stored data is decisive. To reduce the amount of data that has to be kept in the archive, before the upload to S3 all of the files were compressed to save storage space and thus costs.

The diagram above also shows part of the archive that has not yet been discussed: the metadata store. This constitutes the structures and processes in the archive system that work together to implement a GxP-compliant system. Via user identities, policies and logging tools, Amazon furnishes all of the tools that are necessary for a GxP-compliant system. Access to the archive's data can be regulated by Amazon user identities and assigned policies. Because access control in Amazon S3 functions via policies assigned to the S3 buckets in use, there is no way for users to access the archive data without being authorized by their Amazon user account. If users are signed in to Amazon and authorized to use the archive, the logging tools enable recording of all interactions with the archive, up to and including saving every query and the results that are returned. The regulations for retaining these protocols can be set up in Amazon such that the retention period for this information is unlimited. This thus enables the implementation of the requirements for a GxP-compliant system.

MORE TUNING

Next Amazon Athena was checked for additional means of optimization. The fundamental question was whether it is possible to further reduce the amount of data that needed to be scanned simply on the basis of the query. If that

were possible, then we could use Amazon Athena's concept of partitions, under which data can be partitioned according to pre-defined characteristics. Ideally, when queried Athena then only needs to address a subset of the partitions in order to answer the query. A new analysis of the structure of the data being queried was undertaken with this goal. It revealed that a large share of the data is naturally partitionable. Common attributes for partition include paths and time specifications. In addition, there are possibilities for partition based on user IDs that are tied to particular data records within the archive.

With partitioning based on the path information that is in the archive data, it is possible, for example, to limit the number of scanned partitions by entering the first components of the path. By entering additional path components, it is possible to further reduce the number of partitions to be scanned. Because in our case the user making a query often knows exactly in which area or which path within the archived data should be sought, this approach can be used in all cases where entering a path is a primary criterium in the query. The roughly 1 million objects in the source system that need to be archived, yield about 1100 partitions. Athena then divides the data into 4612 files. A further option is partitioning according to time specifications. Many pieces of data within an archive carry a time stamp that details, for example, when a particular record was generated. These time specifications can then also be used for partitioning. In an archive that should contain archived data spanning multiple years, partition by year and month would be sensible.

An additional aspect of partitioning should be mentioned here as well. Sometimes it is not possible for a query to specify in advance which partitions should be scanned because the data might possibly be divided among all partitions. However, partitions can help to make querying a large archive possible here as well by starting multiple queries sequentially on the various partitions and then aggregating the results. So, for example, partitioning according to time can be used to give a user the option to query data incrementally by having successive queries look increasingly further back within the archive. The users can thus decide on their own whether the data the query returns are sufficient or whether, for example, it should look another year further back in the past. Each additional query will be addressed to another subset of the partitions. Here as well, the idea is to keep costs under control by limiting the amount of data that needs to be scanned.

Since the partitioning approach worked so well, this solution was tested on a much larger customer system. In this system, up to 1 billion table rows had to be archived, depending on the domain. The test was successful, and we were able to prove that the solution can also be used for systems of this size in the future.

In summary, it is possible to say that partitions are the key to efficient and cost-effective queries on a large archive. For this purpose, it is important to examine the data itself and later queries to determine what would be a suitable partition. Generic queries that are intended to collect data from the complete life-span of the source system will, in the end, have to address all available partitions. In this circumstance, partitions still offer an opportunity to carry out such queries step by step and to aggregate the query's results.

ACCESSING THE ARCHIVE

At this stage, the archived data are now on Amazon S3 and can be queried via Amazon Athena. For queries to JSON or Parquet files in S3, Amazon offers an integrated Athena query console. With this console, queries against the data can be submitted with SQL. The results of these queries are also then saved by Athena in S3. That is sufficient for initial testing and evaluation. For productive use of the archive, however, a more comfortable means of querying would be desirable. Similarly, a solution was sought that would keep costs as low as possible. In the Amazon universe, low-cost often means serverless. That means that Amazon itself organizes the division of computing tasks among various nodes in the cloud, and the customer is not required to rent any machines for themselves. Athena is what's known as a "serverless service." For queries to data in Athena, Jupyter notebooks that combine query code and data visualization can now be used. With Jupyter notebooks simple query masks can be defined that keep the search parameters in the foreground with the corresponding SQL query saved in the code.

Jupyter notebooks in the context of Athena consist of Python code whose Athena-related computational commands are executed in Amazon AWS. Furthermore, Python is a suitable candidate for implementing integration with other systems. With the use of notebooks for archive queries, Amazon takes care of the availability of the required computing capacity in the cloud. In addition we can decide for an alternative cost model where the execution of queries in Athena will be paid according to the number of so called Data Processing Units (DPU) that have been involved in answering a query. The approaches to optimization that have been previously discussed work exactly the same for optimization geared toward the computing resources used for a query. Optimized query tables, Parquet format and partitions help for the DPU cost model as well, in order to minimize the computing capacity that Amazon needs to provide to answer queries.

Figure 3: Search for Objects in Archive

The figure above shows the archive's interface for querying about objects. This interface can be understood as an initial interface because in most cases users will first use this search mask to find the desired objects in the archive. In most cases, access to an object then follows via a path known from the source system. It is possible to imagine additional search parameters in the final search mask such as the object's name, extension and type that could be used to further narrow the search. When the relevant objects are found, additional details about the objects such as their historical development along with the metadata that is linked to the historical versions can be called up. The query for these details is shown in the figure below.

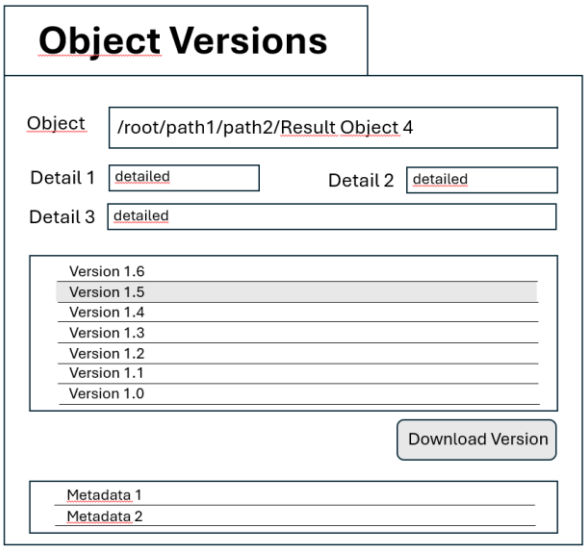

Figure 4: Show Object Versions of Object in Archive

In both dialogs the user has the option to download the physical file which has been identified by the object search. In case of the first search, it is the file itself that is going to be downloaded. In the second dialog the selected historical version of the file will be downloaded.

Several variants of a user interface that could be provided for the archive are conceivable. For one, Amazon Amplify can be used to implement a serverless UI [7]. With this UI we have a considerable degree of freedom and can, for example, implement the interface with Angular. Implementing the UI logic is a bit more challenging in this case because Amplify will simply activate a Lambda function, which is then processed asynchronously with the query. The Amplify UI is then using poll mechanisms to check when Athena returns the answer, at which point it presents the results to the user.

If you are prepared to drop the serverless approach and rent computing resources within Amazon, many additional promising ways open up to implement your own service. This service could, for example, provide an API gateway which in turn can be accessed by an Amplify frontend. The disadvantage to this solution is that it requires the provision of an appropriate machine within AWS for the service to run on. These additional costs, however, also buy the freedom of many more possibilities for a UI. The service itself can access data in Athena by using the Amazon API. If the service is implemented as a REST service, it can serve as the backend for a variety of frontend solutions. The GxP-relevant system characteristics can even be outsourced to this service because user authentication, queries and query results will all be processed by this service, and the controls and access regulation that are part of GxP can all be implemented in the service itself. Access to the GxP-relevant system protocol data and the access to the corresponding reports can be seamlessly built into the frontend as it is developed. Furthermore, a REST service enables seamless integration of the archive with other systems.

IS IT FAIR?

In the context of archives, the term FAIR was developed primarily as a description of demands that are made on academic archives. In this context, it is a matter of the characteristics of the information in the archive, that it should be:

- Findable,
- Accessible,
- Interoperable, and
- Reusable.

The data in the source system, which will be archived, can also be considered from these perspectives. The physical data and revisions are the artifacts that are stored in the archive for later use and which are intended to be made accessible by means of download from Amazon S3. After the download, individual artifacts can be made available to other systems. The information from the source system that is linked to these artifacts - such as the historical development of and controlled changes to a file, the attached metadata including tags, and users who had access to the file under the control of the access rights system - can be used as search parameters in the archive. That makes the file findable. The possibility of controlled downloads of files from the archive ensures the reusability of the archived artifacts. All information about the physical artifacts from the source system are deposited in the archive in JSON format. This format is understandable by both humans and machines, which provides for interoperability.

VALIDATION

Validation of the archive is accomplished primarily by validating the tools and processes used to set it up. The transformations that carry out the extraction of the data from the source system and transform them into optimized formats for archive queries are validated by standard processes such as function, performance, and security tests carried out using meaningful test data. During a transformation the transformers generate checksums for each line in the corresponding JSON files that are created. At the same time, these checksums are also pulled from the source system with database scripts. The lines' individual checksums are aggregated into a common checksum for the table, which then enables a conclusion about the contents of the table's data. Only if the table checksums from the source system and the archive agree is the first part of the validation successful.

Not all transformations can be validated with this process. For some transformations, a single line in the source system is, in certain circumstances, transformed into multiple lines in the archive system. In this case, it is necessary to take the results of the transformation mapping into account when calculating the table checksums. When one line in the source system turns into more than one line in the target system, it is also necessary to calculate multiple line checksums in the source system. The resulting line checksums are then brought together again in a table checksum. For files that will be moved from the source system to the archive, checksums are also used to ensure that every file arrives in full and unchanged.

After the general data integrity during the transformation process was confirmed via checksums, a second validation step can compare information from the archive with information from the source system. For this purpose, known and usual queries that were possible in the source system can be undertaken in the archive. The results of the queries to both systems will then be compared afterward. Only if both results match is the validation successful. The number of samples has to be determined on a risk-based approach. In this process, scenarios from all of the archive's domains must be taken into account. For example, consider the Object Search domain. In the source system, the users can access individual objects and for each object have its history table displayed. This table lists the logical versions of the object. Logical versions are created, for example, by workflows that are defined by quality-relevant processes and which change the object's metadata in a way that is relevant for its validation state. A typical test case would therefore open the object version table of an object in the source system with the path *mypath* and save this

data for the subsequent verification process. Afterward the archive would be queried with the same path *mypath* and the data that were returned compared with the saved data from the source system. If the two results, each of which was pulled from the corresponding system via the user interface, agree then the test case is considered to have succeeded. In a similar way, test cases for the other domains in the source system can be defined and performed. If all of the test cases of the risk-based test suite are successfully completed, the system is validated.

CONCLUSION

The approach we developed to building a GxP-compliant long-term archive in the Amazon cloud works and is at present being rolled out for an initial customer. The developed solution focuses on optimizing archive costs while taking functional and regulatory requirements into account. In this respect, we weighed user friendliness on the one side and expected costs for data storage, query processing and maintenance on the other. With the decisions taken, we obtained an archive solution that is distinguished particularly by the low number of infrastructure components that will require maintenance. That reduces the amount of work needed and thus the costs for availability and upkeep of infrastructure components compared to a self-hosted archive. Furthermore, the cloud solution secures an upgrade path to more capable machines in the future without having to keep an eye on purchasing new hardware. The various storage classes in the Amazon cloud help implement a long-term archive because they relieve us of the task of developing our own local solutions for large file systems that can store data and make it available over the course of decades.

It can also be expected that infrastructure and data format standards will experience changes over the course of years. The decision in favor of JSON file as the primary data basis in the cloud ensures on the one hand the future understandability and interpretability of the data and on the other the ability to undertake a migration from the proposed cloud solution to another solution that might become necessary in the future. Even the path from JSON files back to a self-hosted relational database is not closed off with this approach.

If one is prepared to invest more money in an archive solution, more comfortable forms of querying the archive can be implemented. Such solutions can be put into practice via the implementation of a REST service with the accompanying frontend in the Amazon cloud. This service can also take on the audit trail that is necessary for a GxP-compliant system. Furthermore, the REST interface enables machine use of the archive, which can then also be used for integration with other systems.

REFERENCES

[1] The FAIR Guiding Principles for scientific data management and stewardship, Scientific Data, 2016, https://www.nature.com/articles/sdata201618
[2] The EUCROF Archiving Paper, European CRO Federation, 2024, https://eucrof.eu/02-03-archiving-position-paper/
[3] Amazon S3 Storage Classes, https://aws.amazon.com/de/s3/storage-classes/
[4] Apache Parquet Format, The Apache Software Foundation, https://parquet.apache.org/
[5] Amazon Aurora, https://aws.amazon.com/de/rds/aurora/
[6] Amazon Athena, https://aws.amazon.com/de/athena/
[7] Amazon Amplify, https://aws.amazon.com/de/amplify

ACKNOWLEDGMENTS

We would like to thank Stephan Schröder, who sharpened our focus on the solution in countless discussions and researched many fundamental aspects of archiving in Amazon Athena. We would also like to thank Jacob Thauer, who proved that this solution is a viable option with his prototype implementation of Jupyter Notebooks for the archive.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author Name: Frank Stoinski
Company: Entimo AG
Address: Stralauer Platz 33-34
Work Phone: +49 30 520024 123
Email: sto@entimo.de
Website: https://www.entimo.com/

Author Name: Alexander Lüders
Company: Entimo AG
Address: Stralauer Platz 33-34
Work Phone: +49 30 520024 131
Email: alu@entimo.de
Website: https://www.entimo.com/

Brand and product names are trademarks of their respective companies.