Migrate Mainframe Batch to Cloud Microservices with Blu Age and AWS

By Alexis Henry, Chief Technology Officer, Blu Age

Batch is one of the most complex aspect of a Mainframe migration to AWS. It often dictates whether a Mainframe migration is successful or not. While transitioning to microservices, it is critical to design an AWS architecture capable of satisfying the Batch stringent performance requirements such as intensive I/Os, large datasets, and short durations. This article describes with an example how to migrate Mainframe Batch to AWS microservices using Blu Age technology.

AWS enables building new applications with microservices, managed services, and serverless computing. Microservice architecture benefits are clear although experience is still growing on such topics. Such stories focus mainly on either new application development or peeling monolith based on cloud compliant architecture. Typically, Java applications or facility software moved from on-premises to Cloud.

However, most of existing worldwide IT relies on Mainframe Monoliths. Indeed 75% of the running code is Cobol based (1). While the benefits of transforming those business-critical applications to Cloud-Native architecture are undisputed, most corporations and public agencies are planning and looking for a way to proceed.

Traditional businesses and industries are under pressure because of digital native competitors' effectiveness. Newcomers have better speed and agility while older companies suffer from their legacy Mainframes rigidity. Continuous automation, separation of concern, on demand resources, pay per use and innovation are advantaging newer companies, which increase their market shares because they have the advantage on all fields: core business optimization, better time to market, free marketing as buzz spreads out and better talent attraction. Businesses with Mainframes actively plan the digitalization of their IT but they have the feeling they are leading the path with no clear trail to follow. Indeed, few know transformation facility exists and as such most still consider their Mainframe modernization potentially too big a challenge, either technically or financially.

This article details infrastructure, software architecture and transformation technology to automate modernizing Mainframe Batch to AWS microservice applications.

Mainframe Batch

Batch processing usually involves bulk processing of data, which could not be processed in real time due to the relatively limited capabilities of transactional engines at the time of their initial design and implementation. Therefore software design made extensive use of Batch and associated constraints. High CPU power for mono thread application, locking of I/O to data storage preventing concurrent processing of Batch and transaction, higher TCO for provisioning nightly CPU pick, were the rule. Those constraints are still directly influencing each MIPS estimation, cost, and operational model.

A more efficient architecture for a cost optimized structure is now available with AWS. It can be achieved by transforming legacy Batch processes to real time microservices. Typically, Kinesis streams data, API Gateway invokes appropriate service, Lambda and serverless computing both adapt and optimize I/O and CPU power usage while significantly downsizing the overall TCO.

In the following example, we will explain how to transition a typical retail banking Mainframe application from the dual Online/Batch model toward real time microservices combining AWS services and Blu Age modernization technology.

Mainframe legacy Batch architecture example

We use here an example Mainframe Batch architecture which we will transform into a microservice architecture in later sections.

For this typical scenario, the legacy system is a Mainframe using z/OS CICS, JCL, Cobol, DB2, VSAM files, GDG files and tapes.

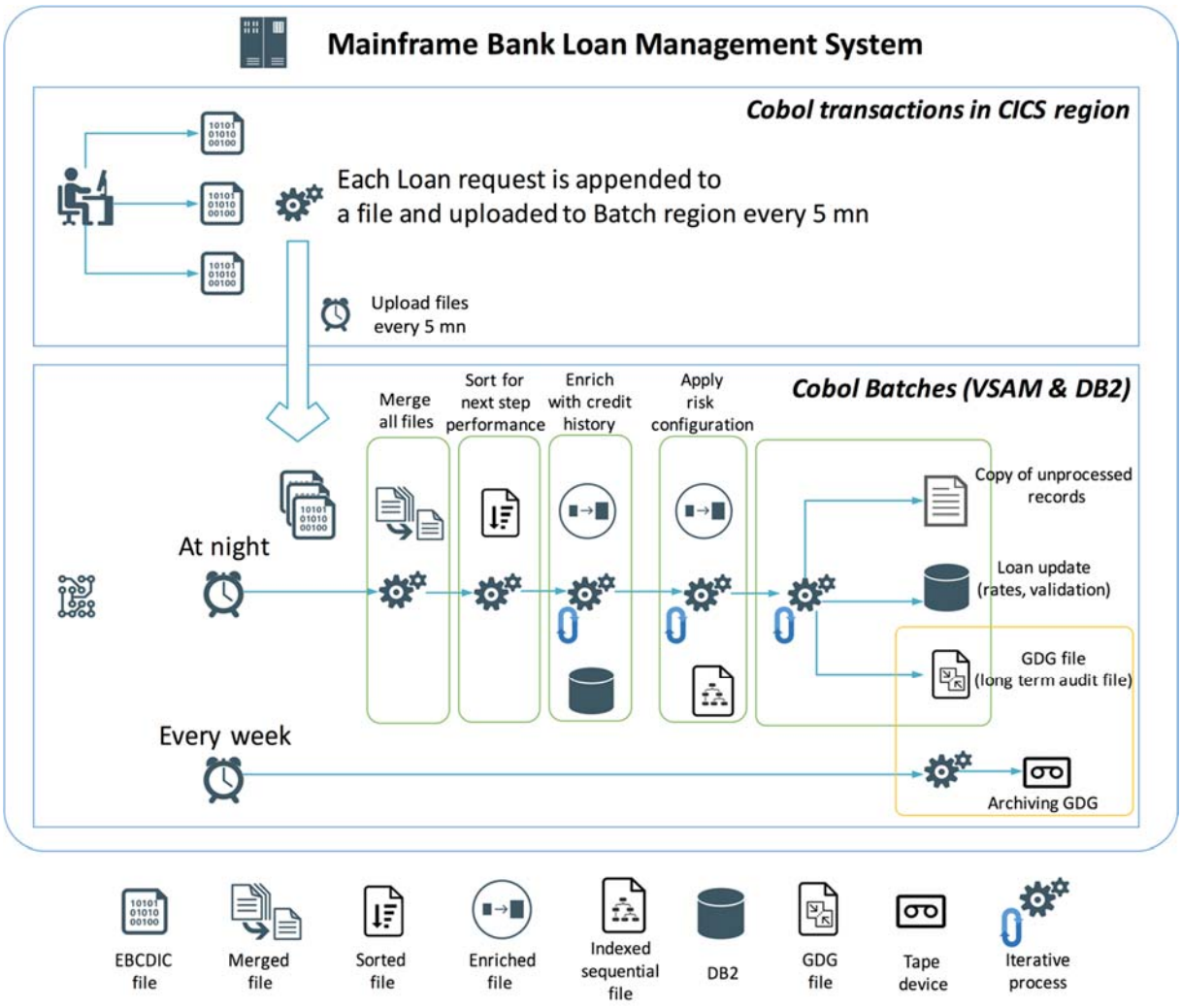

Figure 1 Mainframe legacy Batch architecture example

The Batch programs have been designed to avoid multiple user locking and waiting for transaction responses. During the day, loan request transactions append transaction data to a temporary file, with one file per physical office. At night CICS transactions, sharing data with Batches are switched off to avoid concurrency, locks and consistency issues. There are three Batch programs.

Every 5 minutes, the Upload Batch program sends all temporary files to the Batch region via message queuing.

Every night, the Loan Bath program is triggered by the scheduler. It executes the following logic:

1. All files are merged into one
2. The merged file is then sorted to improve performance and to prepare next processing steps
3. Each record in the sorted file are processed for enrichment (personal information about credit history and other loans are collected from DB2 and injected into the enriched record). Output consists of a new enriched records file
4. Each record is enriched a second time by injecting risk assessment information from a VSAM file. Output is then an enriched file with all information required to perform risk analysis and grant or reject loan request
5. Each record in previous output file is processed, and eventually Cobol programs create 3 outputs: A copy of unprocessed/rejected records which will need further processing (parsing error, missing elements, fails …). An update of records in DB2 table for each customer requesting a loan with current status (rejected, approved, pending) and loan proposal only for approved requests (rates, duration etc …). Audit information (who, what, when, where …) are traced into Mainframe Generation Data Groups (GDG) files

Every week, the Archiving Batch is triggered to save some GDG data to tape devices (for legal reasons), and to prune GDG (removal of files sent to tapes).

Transformation of Batch logic to microservices with Blu Age Velocity

Blu Age technology accelerates legacy applications modernization with automation for both reverse-engineering of the legacy procedural Mainframe applications (code + data) as well as forward-engineering to new microservice-ready object-oriented applications.

When modernizing from Mainframe monoliths toward AWS, both the transformation and the definition of the target architecture are automated and standardized for AWS by Blu Age Velocity transformation technology. The prerequisite discovery and transformation setup is facilitated by BluAge Analyzer. Blu Age also offer an off-the-shelf Transformation Factory and Continuous Integration with Blu Genius.

Blu Age Velocity execution environment is available off the shelf and relies upon two sets of components:

- Blu Age Velocity Framework brings all utilities and services to get rid of former system specificities and anti pattern: Go To removal, data memory model, execution model, data access model, sort and file management utilities, etc.
- BluSam Server can be seen as a full stack microservice container. Any number of container may be deployed, with each being the execution unit for locally deployed services and data access service. Each former Batch program becomes a Spring Boot autonomous executable. Microservice containers are distributed. Programs are freely deployed. Data is freely deployed. In-memory read/write cache may be enable on demand or at start-up. All services are available as REST API. All services are registered into a service directory automatically.

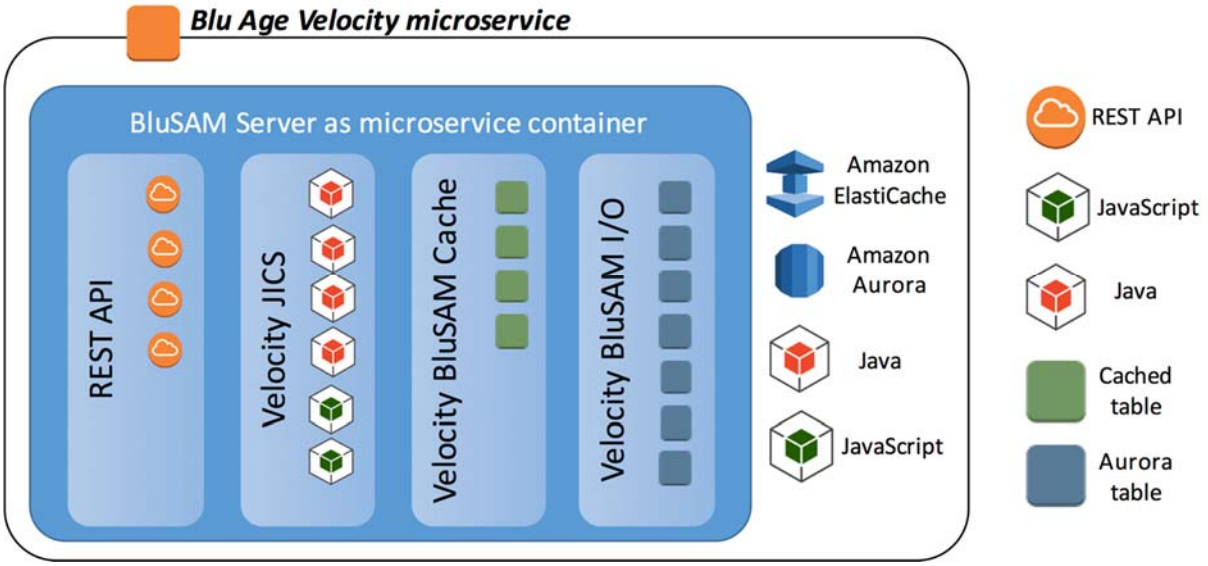

Figure 2 Blu Age Velocity microservice design

Blu Age recommendation for a successful Mainframe to microservices project is to separate the technical stack transformation phase from the business split transformation phase in order to keep the microservices transformation complexity manageable for each phase and minimize project risks. With such an approach, the first technical stack transformation phase focuses on the application code and data isofunctional migration with mostly infrastructure teams from both the Mainframe and the AWS sides. The later business split transformation phase will not involve any Mainframe team anymore but requires participants from the Line Of Business with an understanding of the business functions and processes.

For the technical stack microservice transformation phase, the Mainframe Batch architecture is automatically refactored with Blu Age Velocity in the following way:

- *Rest APIs*: Each service has its REST APIs available and deployed. This enables remote call capability for both business logic services and data access services. Typical integration strategy is made with Kinesis, Lambda, and API Gateway.
- *Java programs*: All former programs and scripts (Cobol programs, JCLs, CICS transactions, BMS maps) are transformed into single executables. BMS maps are transformed into Angular single page application while server-side services are transformed into Java Spring Boot applications. Each may be freely deployed to the BluSam server of your choice. They show in the Java Information Control System (JICS) layer of the above picture.
- *Cache*: Persisted data may be loaded into the in-memory cache for optimization of performance. Cache relies on Amazon ElastiCache with an enhancement to support write-behind. As such not only application benefit from increased read performance but for write performance in bulk mode as well. Native write-through is designed for reading access but cause delays when refreshing data into the database, while write-behind allows optimal performance in all scenario.
- *Persistence data layer*: Persisted data is managed by BluSam I/O. Any former data storage (VSAM, GDG, DB2 z/OS tables, etc) is now stored in a persistence data store. Any prior data access mode (sequential, indexed sequential, hierarchical, relational, etc) is refactored to fit with a new database.

- *Persistence data store*: Typically, Amazon Aurora is the relational data store of choice for data persistence. Now each BluSam Server has the flexibility to operate its own database choice (RDBMS, Key Value store, No SQL, Graph database). As such most AWS compliant database technologies apply to BluSam Server.
- *Service directory*: All deployed services are published into a central directory for location lookup and integration across microservices.

For the business split transformation phase, a Domain-Driven Design approach is recommended to identify each microservice scope with a Bounded Context. Blu Age Analyzer automates domain discovery by analyzind data and call dependencies between legacy programs. Domain decomposition refactoring using functional input is supported as well by Blu Age Analyzer. Decompostion strategy produced by Blu Age Anlyzer is then driving the modernization transformations. For a description of the approach see the following (https://www.bluage.com/products/blu-age-analyzer, https://martinfowler.com/tags/domain%20driven%20design.html, https://martinfowler.com/bliki/BoundedContext.html, https://en.wikipedia.org/wiki/Domain-driven_design)

Once the microservices scope and Bounded Context have been defined, Blu Age automation can quickly re-factor the application code to separate and create the new microservices application packages.

Resulting real time microservices architecture example

Getting back to our Mainframe legacy Batch example, application owners decide to modernize the Mainframe with two main goals in mind:

1. Enhance customer experience and satisfaction with answers in minutes rather than the following day once the nightly Batch is complete. Enable self-service and notifications to mobile application users. This is achieved with a real-time microservices architecture.
2. Introduce agility and the ability to support change requests with a better time to market by refactoring all business logic.

For this purpose, executives decide to transform their Mainframe Batch leveraging Blu Age technology as described in the preceding section. We now detail the resulting real time microservices architecture on AWS.

This example microservices Bounded Contexts split is as follow:

- Retail banking SPA Portal microservice: This is a distributed UI system which is localized per region/country (languages, legal specifics)
- Loan Risk Assessment microservice: This service is in charge of assessing risk of granting loans and sending a rate and duration proposal based on customer profile, credit history, and risk assessment rules
- Transactional retail microservice: This service handles checks, credit card and all former desk facing simple operations
- Long-term data storage microservice: This becomes a service of its own, which other microservices do not have to be aware of (i.e. do not have to trigger or do service composition with)

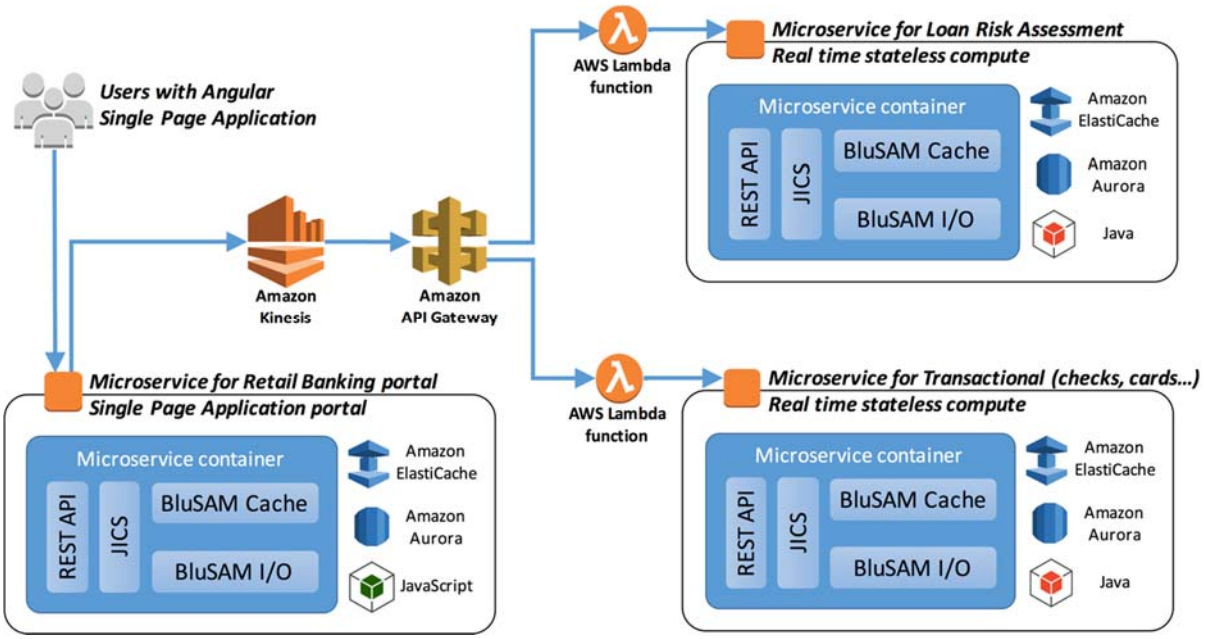

Figure 3 Real time microservices architecture with Blu Age Velocity and AWS

We now describe components of this real time microservices architecture.

Angular front end

Angular enriches user experience while ensuring that users of the legacy Mainframe application do not require specific training for using the modernized system. All surface behaviors of the former Mainframe system are mimicked. However, only user experience related processes and screens have been transformed into a portal in charge of centralizing retail banking activities. This architecture distributes automatically users' connections to different containers and regions for a better availability and reliability. It distributing the local versions of retail Application in various countries and offices as well (because of various languages and legislation). As soon as a user submits a request, the Angular single page application emits data into Kinesis to route and process this request in real-time.

Amazon Kinesis

It is the unified data hub for the real-time system also called the data transport layer. Requests are pushed into Kinesis, which acts as a central, fast and robust real-time streaming hub. Beyond basic message queuing, Kinesis can continuously send data in the stream if need be. It also allows data replay and data broadcasting. Kinesis is fully managed, which means that you do not have to manage the infrastructure nor the configuration to adapt to burst or capacity variations. Even though data will be processed on the fly, Kinesis provides data retention capability for data remaining accessible for a longer time if required. This is beneficial for Mainframe Batch when there is a need to replay or reject prior data processing. Furthermore, Kinesis is used to support the Database per Service design pattern (https://martinfowler.com/articles/microservices.html, http://microservices.io/patterns/data/database-per-service.html).

Amazon API Gateway

It identifies which API or service map to the incoming Kinesis data, it is the perfect fit as the central Hub to access all your microservices whatever their underlying technology and locations. Moreover, API Gateway serves as a service locator to call appropriate microservices, which is key when distributing services. API Gateway also enables service versioning (deploying new version and linking requests to newly deployed services) and Canary deployment strategy (routing a set of user to a new version while routing remaining users to the former version of the same service). Canary release

strategy allows reducing the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody. Another reason for using API Gateway as Canary strategy is when different business versions exist which is typical in Banking because large banks usually operate in many countries with different regulations. Using microservices over API Gateway both solve multichannel and multi regulation issues.

AWS Lambda

API Gateway uses Lambda as a proxy to call microservices. Lambda will initiate context and parameters value to inject into remote service. In the legacy system JCL receives parameter values from a scheduler or programmatically define variables. In such context, those parameters are used to setup the Batch runtime environment (name of the dataset, version of deployed programs, execute Batch in production or test partition, etc). In the new paradigm, Lambda is used to keep this runtime parameter capability. In order to preserve logic into endpoints, parameters are not managed in the data transport layer. Lambda then triggers the appropriate Groovy script (which replaces z/OS JCL after being transformed with Blu Age). Groovy is preferred to Java to be modified without compilation while sharing the same JVM as the Java classes to be run. Therefore, Mainframe Batch job Steps or run units may be reproduced, and modification can be done to Batch setup without compilation. Groovy and Java classes are called via REST and may be either synchronous or asynchronous. Asynchronous is preferred in case of long processing time. Indeed Lambda lifetime must be kept below 500 seconds, and in such case detached services is the right pattern.

Amazon Elasticache

One specificity of Mainframe system is I/O capabilities provided both by the underlying file system and non-relational databases built on top of it. Among those, VSAM relies on indexed sequential data store for which modern databases (RDBMS, graph database, column database, etc) do not provide equivalent indexing, at least not preserving performance for all features. Blu Age BluSam uses Elasticache (Redis implementation) in order to bring equivalent performance while supporting necessary I/O capabilities features:

- **In memory indexes:** VSAM indexes are stored in Elasticache to support fast and full featured indexed sequential logic.
- **Index persistence:** Indexes are saved in real time to an underlying database to provide the required availability requirements. The default configuration stores into Aurora. BluSam allows using any RDBMS or Key/Value store as well.
- **Record caching:** Mainframe datasets records may as well be uploaded to cache, either at BluSam Server startup using bulk cache insert or on the fly as requests hit the cache.
- **Write-behind:** In addition to ***write-through*** BluSam adds persistence specific services to support ***write-behind***. Indeed write-through could falls-short for bulk data change, specifically in Mainframe Batch mode. Write-through induces a delay (database acknowledge), which may cause a performance slow down when doing the bulk processing. For this reason, write-behind has been added to manage transactions only at the cache level. The cache manages persistence to the underlying database on its own with a first of strategy (first of N-record changed and elapsed time since the last cache saving; the default configuration is 10 000 records and 1 second which can be adjusted). This write-behind feature is available for both indexes and individual records.
- **Managed service:** Elasticache is a fully managed service that enables transparent scaling. With such feature, even large Mainframe Batch systems requiring processing of terabytes of business records per day are handled by ElastiCache with no need for a system administrator.

Data may be uploaded into the cache in burst mode to process reliably any bulk data (warming cache and processing in memory is typically a good strategy for bulk processing).

Amazon Aurora

It is the preferred target AWS database for Mainframe modernization because of the following characteristics:

- **Aurora performance and equivalence**. Indeed legacy Mainframe applications rely mostly on VSAM and SQL I/O capabilities. Replacement from I/O access type to another type like put/get of a document would make all application logic disconnected from data access API's provided by the data storage. As a result, the code base would require full rewrite which is the riskiest and longest way to transition from Mainframe to Cloud and microservices because the complexity and quantity of refactoring involved. However, using Aurora with BluSam and AWS Elasticache allows preserving transformation automation and performance with no need for refactoring for both VSAM's like indexes (permanent storage in Aurora; live indexes in Elasticache), and native SQL support.
- **Aurora managed service.** Scalability is important when modernizing Mainframe because legacy application usage varies over time (Payroll system, tax payment system have a peak of activity every month/quarter. Banking systems have different CPU consumption needs depending on customer transaction vs fraud detection system). Aurora is a managed database which storage can automatically grow to 64 TB per instance.
- **Aurora native Lambda integration.** One challenge with microservices is the Database per Service design pattern. This pattern creates a need for data synchronization because data is distributed (https://martinfowler.com/articles/microservices.html, https://www.nginx.com/blog/event-driven-data-management-microservices/, https://en.wikipedia.org/wiki/CAP_theorem). While specific patterns exist to do the trade-off between eventual consistency, rollback mechanism, and transactional delay, they were all designed for online transactions. However, they all suffer from network and transactional delay which do not fit with the Batch latency requirements (commit time must be within 1 millisecond while patterns such as Saga introduces up to 100 milliseconds delay (http://microservices.io/patterns/data/saga.html, http://microservices.io/patterns/data/api-composition.html). Aurora brings a simple yet effective capability for data synchronization with native Lambda integration. Whenever a record is modified, then a Lambda is triggered. The Lambda is used to stream into Kinesis which delivers to AWS API Gateway. Kinesis allows having multiple subscribers to propagate the data change to their local data store, while API Gateway allows doing API management for each domain. Then all domains are synchronized simultaneously while each implements its private synchronization APIs based on its private choice of languages and databases.
- **Aurora Serverless**. It opens new strategies to achieve high performance. Aurora Serverless behaves like the regular Aurora service but automatically scales up or down capacity based on your application's needs while preserving ACID transactions. Aurora Serverless is a cost-effective solution for Batch, burst and high-performance jobs such as high level of I/O burst, massive upload of data in memory, low latency write-behind for massive data update, shorter elapse time for long processes such as payroll Batches, data consolidation, etc.

Data Storage microservice

In the Mainframe system, long-term data storage was handled at the application level, with several Batch jobs being responsible for archiving, back up and pruning. Furthermore, the Mainframe long

term data storage technology is costly and complex using GDG files on IBM Mainframe and then tapes shipped to off-site storage.

With AWS, the long term data storage microservice is built with a single Lambda function leveraging Amazon S3 and Amazon Glacier.

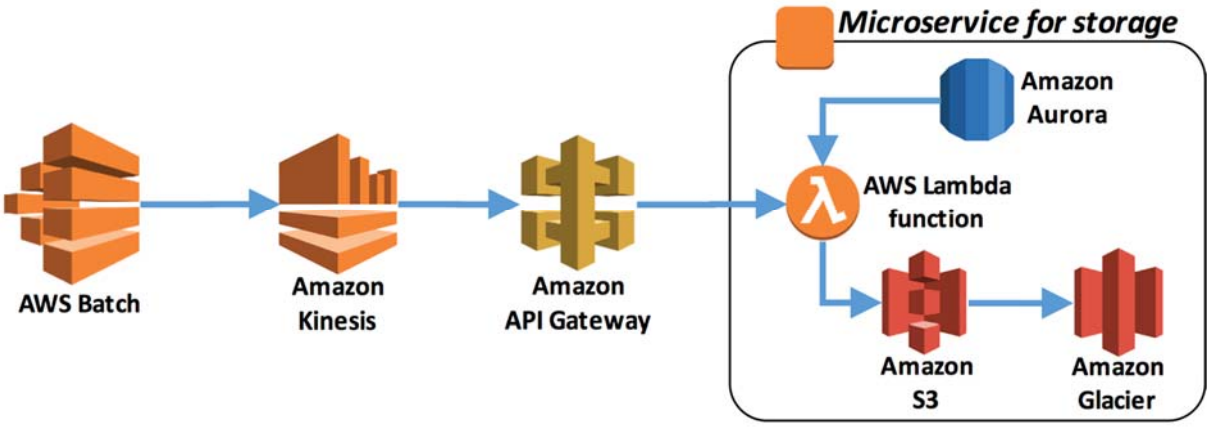

Figure 4 Long term data storage microservice

To preserve the global architecture Kinesis and API Gateway stay the central transportation layer. AWS Batch is used to schedule data storage actions through a Lambda function. The Lambda function copies Aurora data into S3 with a command similar to the following:

SELECT INTO OUTFILE S3 ***<sql statement>*** , where ***sql statement*** selects data to be stored externally to the application database.

Amazon S3 functionalities replace z/OS GDG features, local copies of data, audit trails, medium term archiving and pruning of data, extra backups, etc. For long term data archival and possibly satisfying regulatory requirements, the data is later moved from S3 into Glacier based on a Lifecycle Rule.

Customer benefits

Blu Age automation accelerates the migration from the Mainframe Batch to the target AWS microservices minimizing project risks and costs. There are also savings coming from transitioning from a Mainframe MIPS cost structure to AWS managed services budget structure. Such savings typically finance the modernization project in short time period allowing a quicker Return On Investment.

The target architecture uses AWS managed services and serverless technology. As such each microservice is elastic minimizing system administrator tasks. It adapts to the client demand automatically ensuring availability and performance of services while consuming only the necessary resources.

From a design perspective Mainframe Batch applications are migrated to real time, which allows changing the user and customer experience. This of course drives a better service quality and it promotes the feeling that you are consuming innovative services which is another buy in factor.

Batch application are transformed into microservices which benefit from more flexibility, increased agility to satisfy change requests, independent business domains, deployment automation, safe deployment strategy (Canary and Blue/Green deployment). In short: better, faster, safer capability to deliver and implement changes.

Learn More About Blu Age Velocity

Blu Age Velocity can be used by any customer in any industry, for any mainframe executing languages such as COBOL (including most of its various flavors), JCL, and subsystems such as CICS, IMS, and VSAM.

Customers engaging in complex Mainframe Batch migration require proven technology and methodology. Blu Age Velocity secures and accelerates the code automated modernization and refactoring as well as the target architecture definition. Blu Age also facilitates the necessary activities from legacy code base inventory and analysis to control of like-for-like business logic testing and compliance with the latest development standards. Blu Age recommends performing a Proof of Concept with the most complex Batch jobs. This proves the technology robustness and minimize risks for the other jobs or programs.

Blu Age Velocity is a facility ready to use. Its full automation allows quick projects, minimized risks and optimized budgets. Financial savings due to transitioning from a Mainframe MIPS cost structure to a pay-as-you-go model can be significant and typically finance the modernization project in a short time frame. Essentially Blu Age modernization effectiveness allows a quick Return On Investment.

For easy access, Blu Age Velocity is embedded within our Blu Genius portal:

https://www.blugenius.com

For more information about Blu Age capabilities, visit https://www.bluage.com/ .

For Blu Age references, visit https://www.bluage.com/overview/our-projects

Feel free to contact us at https://www.bluage.com/contact

Sources:

1: network world 2003, The Guardian 2008, https://blog.codinghorror.com/cobol-everywhere-and-nowhere/, Quora https://www.quora.com/How-many-COBOL-programmers-are-left-in-the-US, Gartner group, Microfocus https://blog.microfocus.com/cobol-still-standing-the-test-of-time/