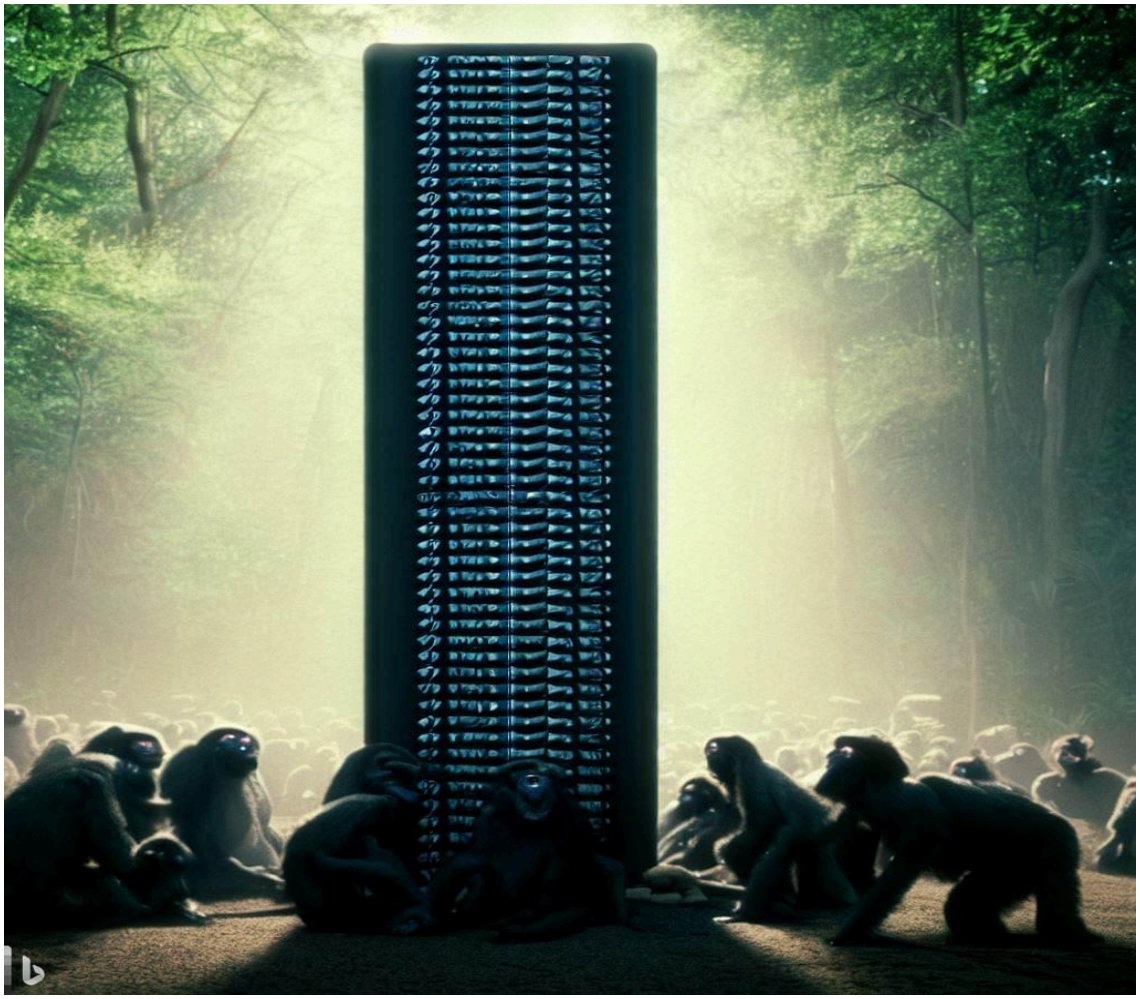
IBM Mainframe: Migrating to an Open Architecture

How to deconstruct a mainframe and deploy the code to Kubernetes



Introduction

Which industry or sector of the economy can compete with technology developed in the 1980s?

For various reasons (heavy investment, loss of technological talent and the consequent difficulties in implementing transformation projects, etc.), this technological base is the reality today in a large part of the financial sector.

IBM mainframe technology still has a significant market share in the financial sector. Many large companies[1] in the sector, including banks and insurance companies, continue to maintain their operational systems on IBM technology, with designs and products dating from the end of the last century.

How does the use of such technologies affect banks' IT organisations?

In these companies, IT departments are often inefficient, with high costs relative to other industries, a very long development cycle in delivering new solutions to the business, and a poor ability to attract or retain talent.

It is, however, possible to implement strategies that will allow you to:

- Reduce operating costs, resources can be freed up for use in other businesses
- Significantly enhance the productivity of development teams, thereby increasing the capacity to deliver new products and services
- Attracting new talent to the organisation

These strategies are supported by capabilities that enable the migration of code developed on the IBM Mainframe platform to an open hybrid architecture based on Kubernetes.

[1] According to IBM, 44 of the world's 50 largest banks use mainframe technology (zSystems)

Executive summary

IBM Mainframe technology played a pivotal role in the last century, automating the operations of the financial sector. IBM's dominance in this sector was such that it came close to monopolising it. However, this advantage has now become a liability, as it is preventing innovation and the introduction of new solutions.

IBM would need to implement disruptive changes to its platform in order to remain competitive with the major digital players. This would result in the non-functioning of millions of programs, written more than 20 years ago and which form the basis of the banking business. This leaves us with a niche platform that is no longer aligned with market standards and best practices.

How can the mainframe platform be eliminated while safeguarding the entities' main assets of value (COBOL code and data)?

Previously, strategies for eliminating the mainframe involved converting COBOL code or emulating the mainframe platform on a different technical architecture.

However, with the open source technology available today, we believe there is an alternative approach.

- Compiling COBOL code on Linux platform
- Deploying CICS/IMS transactions and batch processes as microservices on a Kubernetes platform
- COBOL microservices will be interoperable with those developed in other languages
- The code is managed through an automated pipeline (CI/CD) to ensure efficient and streamlined development processes

Kubernetes is a platform that has been widely adopted and developed by the leading digital players. COBOL programs on the mainframe platform can be migrated to this execution architecture, which allows them to behave like any other microservice. This results in immediate benefits, including enhanced security, monitoring, automated operation, deployments, portability, and more.

Financial institutions have the flexibility to choose where to run their workloads, including

on-premises (by installing Kubernetes on their DC), on-cloud (on any of the hyperscalers, Google, AWS, Azure, etc.), or in a hybrid architecture. This eliminates the need for code changes and ensures that they remain agile and flexible.

The anticipated advantages of adopting this approach are as follows:

- A reduction in the cost of operation is a key objective. The cost of operation can vary significantly between entities. A simple exercise that includes the cost of hardware, software licences, z/System maintenance team and the cost of the Data Center can amount to more than 100M€ per year in a medium/large entity.
- The productivity of development teams. The objective is to double the productivity of the teams, accelerate time to market, implement automated microservices deployment pipelines in Kubernetes, and enable the development and testing of COBOL autonomously.
- The solution attracts talent, unifies the technical platform on the most demanded technology in the market, and allows the introduction of new frameworks (Spark, AI, etc.). There are no longer "legacy" teams specialising in mainframe technologies and teams specialising in "modern" technologies. All teams share the same tools and platform, with the only difference being the programming language used (COBOL, Java, Python, Go, etc.).

Background

This document is not intended to provide a comprehensive technical architecture of the IBM Z mainframe platform. However, it is important to highlight some of its most relevant features.

- The origins of the current platform can be traced back to the S/360 models of the mid-1960s and subsequent S/370 and S/390 iterations. IBM developed its own "proprietary" standards in those years (EBCDIC, SNA, etc.), which are still present to a greater or lesser extent in the current models.
- The limitations of computing in those years directly impacted the capabilities offered by the operating system, COBOL language, storage subsystem, and other components.
- The high cost of computing these early models led IBM to implement a strict backward compatibility strategy to protect customers' software assets. This strategy has remained in place to this day, allowing customers to run programs developed in the 1970s on a current mainframe.

This resulted in IBM's market dominance in the financial sector during the last century. However, these advantages have become a burden for today's financial institutions, as they are tied to an obsolete technology that impedes technological evolution.

The technological revolution of the 21st century (the rise of the internet, the mobile phone as a universal connectivity tool, open source software, cloud computing, big data, AI, etc.) offers corporations unprecedented innovation capabilities and therefore enormous opportunities to increase their relevance in the market. However, with some exceptions, large banks have been unable to leverage this revolution to build new solutions that expand their customer base and maximise their profitability compared to other industries.

While new companies are being established on a daily basis to develop products based on technologies that did not exist 5-10 years ago, the financial sector continues to invest significant financial and human resources in further evolving its traditional mainframe-based systems, without being able to offer its customers experiences similar to those they encounter when consuming digital services designed in the last 5 years.

However, what does IBM say about its mainframe platform?

"Z systems cost less than other platforms on the market, are extremely robust, secure and allow for faster implementation of business solutions."

This assertion is questionable given the limited or non-existent presence of IBM mainframe technology in key sectors other than finance.

The mainframe technology (zSystems) has a low total cost of ownership (TCO), below competitor products.

To gain a deeper understanding of this statement, let's break down the cost.

The **hardware** is tied to a proprietary IBM platform. IBM's strategy over the last decades has not been to allow the execution of mainframe workloads on third-party architectures (e.g. Intel x86[2]). Instead, it has entered into agreements or developed proprietary technology to allow the execution of Linux systems on its proprietary processors/architecture.

Despite continuing to invest in its hardware division, IBM has been unable to keep pace with the competition (i.e. Intel) in the development and evolution of its processors and lacks the economy of scale to compete on price.

The recent industry moves towards ARM technology and the development of AI, including those by hyperscalers and Apple, will further accentuate these differences.

Software, regardless of the total cost of software licences, IBM has encouraged its customers to enter into long-term agreements and to guarantee a minimum consumption in order to qualify for a discount on list prices. In practice, this removes any incentive for customers to replace IBM technology.

It is not uncommon for financial institutions to find that IBM products perform worse than the competition (DB servers, J2EE servers, etc.) simply because they are perceived to be "cost-free". However, this cost is included in a long-term contract that is very difficult to renegotiate. For almost any product in IBM's portfolio, there is an alternative on the market offering enhanced performance or reduced maintenance costs.

The cost of operating a platform typically excludes the expense of **maintaining or developing**

[2] IBM has the technology to run an image of its operating system (z/OS) on an Intel platform, but does not allow its use in production environments.

new business applications, which often requires collaboration across teams. This cost is typically the most significant item in a technology department budget and is directly correlated with the platform's productivity.

It is rare for a developer to be productive on a mainframe platform. There are a number of reasons for this, including:

- The limitations inherent to the traditional programming languages used (COBOL, PL/I) present a challenge
- The difficulty in the initial phases of development (coding, unit testing, integrated testing, etc.) is often encountered when developing on a mainframe platform
- The lack of a modern CI/CD process with tools for agile and collaborative development hinders efficiency
- The radical separation between Dev and Ops teams, with different teams, tools and processes, objectives and cultures, makes it difficult to streamline processes and foster collaboration

Mainframe technology (zSystems) is robust and secure.

This capability is not exclusive to this type of technology.

Applications developed on mainframe architectures are "monolithic" systems due to the high cost of the hardware and the limited number of processors and memory available at the end of the 20th century.

In a z/System, processes share memory as well as access to the storage subsystem. The most efficient method of communication[3] between these processes is through the use of shared memory, which results in a high level of integration between the various products and systems developed.

There are no application programming interfaces (APIs) available to expose business functionality, access a database, and so forth. Instead, application programs exchange memory addresses (pointers) with pre-established data structures.

The only way to expand such architectures is to add more resources (processors, memory, etc.) to the existing hardware[4]. It is therefore of the utmost importance to ensure a very low error/failure rate on the hardware platform.

IBM's claims of reliability and security are based on its purported minimal error rate on the hardware platform. However, it is important to note that no system is immune to human error or a poor technical decision.

In modern architectures, such as those based on microservices/kubernetes, hardware reliability is no longer a critical factor. Instead, growth is achieved horizontally, through the deployment of new instances of components over a distributed network, storage and compute infrastructure.

In the event of an irrecoverable error on a mainframe, the most common contingency mechanism is to have an equivalent hardware configuration on an alternative data centre. The strategy is to replicate data over an alternative data centre using IBM products.

[3] Don't communicate by sharing memory; share memory by communicating
[4] In a Sysplex architecture, a cluster of machines sharing memory can be configured, moving from a monolithic architecture to an even more complex distributed monolith, which requires even greater maintenance

A modern, stateless architecture allows for the decomposition of systems into different, independent services, as well as the implementation of fully or partially active-active[5] recovery strategies. There is no need to invest in alternative data centres. The capabilities of hyperscalers can be used to ensure reliability and robustness. This can be achieved through a hybrid on-premise/on-cloud multi-AZ deployment in a single region, multi-region deployment, deployment in different cloud providers, and so on.

The use of mainframe technology (zSystems) enables the rapid implementation of business solutions.

In fact, the process and tools used by a COBOL developer have remained largely unchanged over the past 30 years.

IBM has not invested in enhancing the capabilities of traditional products (COBOL, CICS/IMS, Batch, etc.) on which most of the critical banking functionality has been built. Instead, it has focused on adding "open" (Linux) capabilities to its mainframe platform.

The result of this strategy is open to question. Traditional solutions still have significant shortcomings and are unable to support new products and services with the necessary agility. Conversely, "open" solutions are not a viable option in comparison to the possibilities offered by other companies or hyperscalers. Organisations are limited in their ability to improve their development cycle to the options that IBM considers valid. Strategic technology decisions and innovation paths are no longer in the hands of the organisations and depend exclusively on IBM's business strategy.

Furthermore, IBM has been investing the cash (operating cash flow) it obtains with this technology in various future initiatives, but it appears that in all of them it has arrived late or with an insufficient focus (cloud, AI, etc.). The recent acquisition of Red Hat represents IBM's last opportunity to avoid becoming a mere also-ran in the years ahead.

[5] The active-active configuration of databases is conditioned by the technology used

Proposed solution

While it is not feasible to alter the underlying technology and architectural paradigms that underpin much of the industry's operations immediately, there is a well-structured path for the strategic evolution of mainframe platforms towards approaches that have proven successful for the major digital players.

Problems in the past

To understand this change, we will explain how similar initiatives have been approached and why they have not worked in large financial institutions.

Converting the code to a modern programming language

Under this approach, COBOL code is automatically (or semi-automatically) "parsed" and converted to another programming language, such as Java.

- Transactional systems, CICS/IMS, COBOL and DB2 are moved to a J2EE, Java and relational database environment (Oracle, PostgreSQL, etc.).
- To copy the way transactional monitors (especially CICS) work, you need to create "utilities" to do this. The complexity depends on the number and type of CICS statements[6] used in the application programs.
- DB2 database access statements are static and must be adapted to the target database (Oracle, PostgreSQL, etc.).
- Batch has special characteristics. To simulate JCLs/JES, JCLs are usually transformed into bash and Java processes are run on J2EE application servers.

This approach has a number of drawbacks, which are described below.

[6] Sending/receiving messages, formatting dates, use of TS QUEUE, management of EIB headers, etc.

As mentioned above, systems developed on the ZSystems platform are highly coupled (monolithic architecture). This is especially critical in certain modules on which common application logic rests.

The possible alternatives for dealing with this problem are always complicated to manage:

- We must create a coexistence architecture that allows us to progressively migrate the code[7]. IBM's proposal is to develop in Java on the Mainframe platform and allow communication between COBOL/Java programs by means of JNI. This option makes no sense except for IBM's commercial department.
- A big-bang strategy is not a viable approach for medium and large financial institutions. While this may be valid for self-contained, low-volume/criticality applications, the enormous risk and cost involved make it unfeasible in the majority of cases.
- Duplicate maintenance of the common systems will result in two applications (one in COBOL and the other in Java) until the end of the project. This option is only valid if the number of systems to be duplicated is limited and their maintenance is minimal.

It is crucial to grasp the project's final outcome to assess its impact on the technology organisation, particularly the development team.

COBOL code developed over the last 30 years is typically characterised by the following:

- Insufficient or no documentation[8]
- Difficulty in understanding the code. To optimise the use of memory, programs are limited to 8 characters, the name of DB2 tables usually has the same limitations. Variables used in programs are not significant, instead of using marital status, single, married, etc., a numeric variable of length 1 is used, the same happens with return codes, errors, etc.
- The referential integrity model is not typically implemented on top of the DB2[9] database. It must be deduced from the application code.
- The COBOL language is designed to optimise processor usage. Static variables, fixed-length alphanumeric, fixed-length signed/unsigned numeric with decimal mask, packed decimals, etc.

[7] How to connect a COBOL program running on a mainframe with a Java class running on a J2EE server on an Intel platform?
[8] In most cases the only documentation is the COBOL code itself
[9] Due to the high cost of implementing it in the early versions of the product

- Procedural language. The code is packaged into distinct "subroutines" that are called dynamically or statically through the exchange of data structures (copybooks)

This code is transformed by an automatic process into a Java class:

- This strategy will require a significant adaptation from the COBOL Mainframe developers to maintain the code efficiently. Depending on the code conversion strategy used, the result may be to make the code unmaintainable
- A new development team with advanced Java skills will find it challenging to understand the logic and structure of the generated code, making it difficult to maintain in the future

Mainframe emulation on a different hardware architecture

In this case, the COBOL code is compiled on a new hardware platform, emulating the functionalities of the IBM products necessary for its execution (CICS/IMS sentences, JCLs, access to the DB2 database, etc.).

This approach is exemplified by Oracle's initiatives to migrate mainframe code to Tuxedo/Oracle DB and Opentext's suite of products (Microfocus).

This approach has several significant drawbacks.

- Firstly, all the inherent problems of the mainframe platform are carried over to the new architecture, including monolithic systems, high coupling, and difficult development.
- The replacement of a technical platform (IBM Mainframe) with hardly any evolution in the last years by a proprietary platform that will hardly have a better evolution/support than the original one is a poor decision.
- As in the previous case, a coexistence architecture must be provided to allow a progressive migration of the code, avoiding big-bang strategies.

In the vast majority of cases, this type of initiative leads to a very long and costly project that leaves the organisation in an equal or even worse technological situation.

The promise of reduced operating costs is quickly compromised by the cost/length of the project. The end result is a dead end that will irremediably involve a new technological transformation project in the medium term.

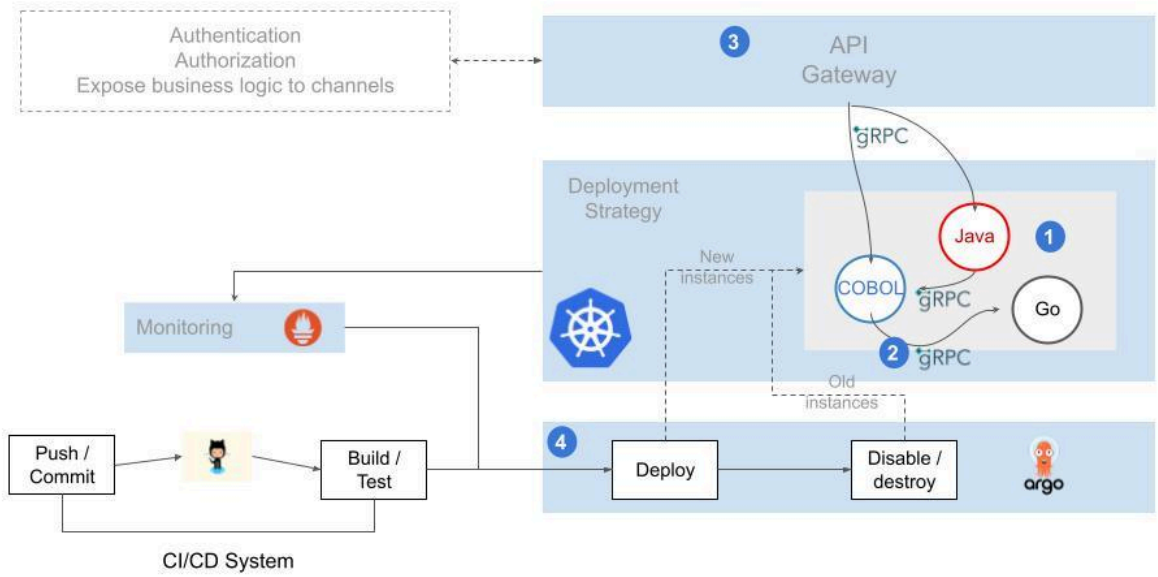
driver8 solution

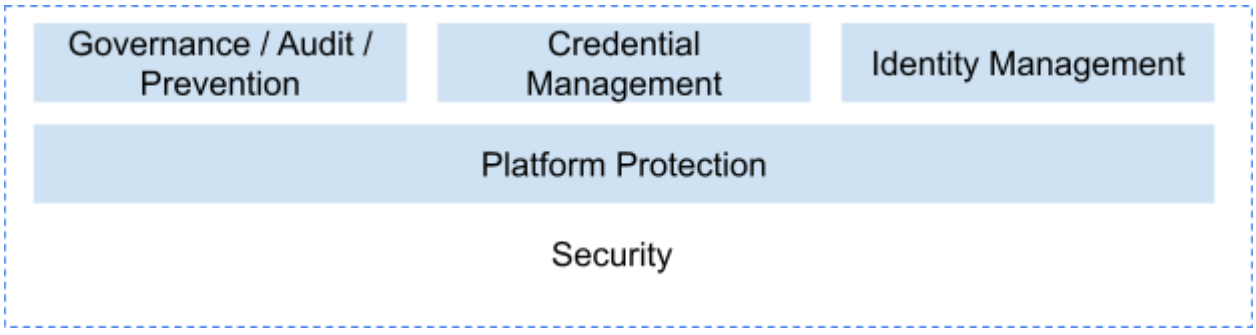
Driver8's solution is based on the following premises;

- *Dual approach*: Separating the flow of new projects from the stock of mainframe applications.
    - Flow: A microservices architecture model to intercept new business initiatives for deployment on a Next-gen infrastructure based on Kubernetes.
    - Stock: The progressive and risk-minimised migration of COBOL (Online/Batch) Mainframe workloads to this Next-gen infrastructure.
- Use of *Open Source* projects. The target architecture is based on products and standards supported by the main technology companies. Applications can be deployed on on-prem infrastructure (on existing DCs), on the cloud (Google, AWS, Azure have solutions valid in different geographies) or in a hybrid architecture
- *Integrated platform*. New functionality (microservices built in Java, Python, Go, etc.) and mainframe COBOL code can be deployed on the same infrastructure and technical architecture. Developers use the same CI/CD tools (repo, pipelines, etc.), with the only difference being the language in which the microservices are built.
    - Mainframe (Online) transactions are deployed as COBOL/gRPC microservices in a Kubernetes cluster
    - In the case of Mainframe Batch processes, the JCLs are converted to YAML/JSON files and interpreted by a Kubernetes Batch Scheduler specialised in the execution of massive processes (computation and data)
- *Coexistence mechanisms*. Two levels of coexistence are defined to ensure a progressive and risk-free migration of Mainframe code. Each entity will be able to define its plan and establish the deadlines for undertaking the migration project.
    - Coexistence between next-gen platform and Mainframe. The solution allows read/write access to the Mainframe platform from the microservices deployed on the Next-gen platform (Kubernetes) by means of two basic connection mechanisms: SQL access to the DB2 Database (proxy on the DB2 odbc driver) and

execution of CICS/IMS transactions. TCP/IP socket connection with transaction handlers

- Coexistence between microservices, regardless of the programming language used (COBOL, Java, Go, etc.). They expose an interface following the gRPC standard that allows their transparent integration (COBOL Copybooks are exposed as gRPC proto messages)





1. The solution allows developers to code services in modern and attractive languages, while also allowing the use of parts coded in "legacy" languages. This avoids the unnecessary waste of resources that would result from recoding
2. Communication between the different services, internal communication, is implemented by means of a light and efficient protocol
3. Services are invoked from front-ends or third-party systems (third-party payment platforms, partner software or other entities, etc.) via a secure, resilient and easily scalable mechanism
4. The operation is supported by deployment automation pipelines and advanced observability capabilities that allow an integrated and consistent view of the entire application flow and the health status of the elements involved
5. Security is embedded in the platform with standard components that implement advanced mechanisms to provide the necessary capabilities according to the requirements of the financial industry.