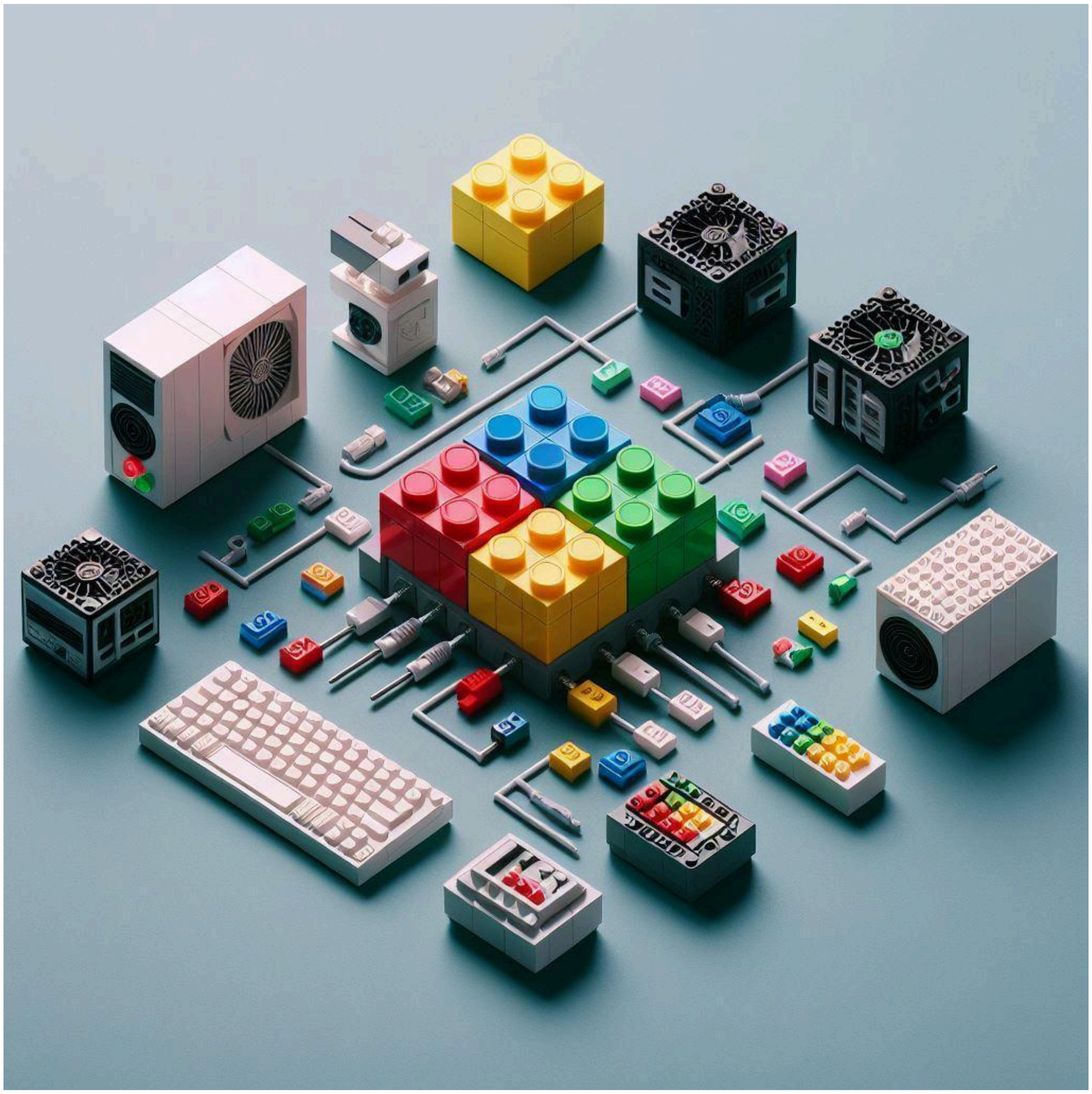
d8parti: IBM Mainframe Batch

How to run Batch processes in an open architecture



Introduction

The IBM mainframe architecture has its origins in a time when the main type of processing was through the execution of Batch chains.

It is therefore a fundamental component in this type of architecture. Although in recent years/decades the focus of development in most financial institutions has been on implementing online processes (payment systems, internet banking, mobile banking, etc.), there are still a large number of business-critical batch processes.

Many institutions are now exploring the potential of new processing models, such as asynchronous event-driven architectures. However, it is important to recognise that batch processing remains an extremely efficient model that should not be discarded. It forms the core of accounting processes in most medium/large financial institutions.

Do financial institutions really process information in real time?

To gain insight into the significance of Batch, it would be helpful to take a step back and examine the circumstances surrounding the initial adoption of this technology by financial institutions. Up until the end of the 20th century, it was possible to divide workloads in a mainframe environment into two types, with a separate execution window for each.

- The online window is open during office opening hours. In which the machine's resources are allocated in priority to transactional monitors (CICS/IMS) and the execution of batch processes is restricted.
- The Batch window. As offices close, capacity is allocated to daily batch processes.

It is often the case that the closure of offices is accompanied by a change of accounting date/session and the launch of batch processes with extensive access to data (read/update). This model allows both types of process to be shared on the same hardware infrastructure, which avoids the potential issues that can arise from online batch concurrency. These include excessive consumption of hardware resources, data blockages due to massive updates, inconsistencies in data access, and so on.

The basic product servicing operations are carried out from the offices and this information is subsequently consolidated by means of batch processes (risks, financial/analytical accounting, reporting, etc.).

It is vital that these processes are completed successfully during the Batch window, as failure to do so will result in the offices not opening as scheduled the following day.

What are the problems with this model?

This model has significant issues.

Mainframe technology has been key to the success of most entities in the financial sector. However, with the turn of the century and the widespread use of the internet, the previous processing model (two independent online and batch windows) is becoming obsolete.

The digitalisation of the sector is accelerating exponentially, with initiatives such as mobile banking, process automation, document management and data analytics. This will lead to the disappearance of the batch window:

- The online window must no longer be limited to office opening hours. The customer demands 24x7 opening hours
- Batch processes must be reviewed to avoid contentions that impede the correct functioning of the online processes (downloading and processing from sequential files, frequency of commit to the database, etc.)
- Data is offloaded or replicated to other analytical platforms, but the bulk of the accounting processes still reside on the mainframe server
- It is essential that system resources (CPU, memory, etc.) are shared as all batch and online processes run on the same hardware platform

Consequently, the complexity and duration of batch processes (elapsed time) will inevitably increase, putting the daily operations of the institutions at risk.

Entities face a vicious cycle of mainframe infrastructure growth as they are unable to separate and isolate workloads on dedicated servers..

How to improve the performance of Batch processes in a highly distributed platform based on

Kubernetes?

Batch processes are no different. Their performance is determined by the system resources they use, which are mainly:

- CPU cycles
- Memory
- I/O
- Network

CPU and memory

NASA used IBM Mainframe technology last century, but the batch processes executed in a financial institution are far from the complexity needed in the space race. They are mostly sorting processes (SORT), string manipulation, report/interface generation and simple calculations.

Any CPU can execute such instructions. However, in a monolithic IBM Mainframe architecture, CPU and memory must be shared among all processes running on the machine. This includes operating system tasks, CICS/IMS transactions, batch processes, DB2 database, and so on.

By isolating each step of a batch process in an independent container and executing it on a distributed architecture, it is possible to allocate specific resources (CPU and memory) to the execution of the process, avoiding any kind of resource contention.

This type of architecture allows for transparent and infinite horizontal growth by adding more hardware (nodes) to Kubernetes clusters.

I/O

Without a doubt, the most critical resource in a Batch process is the I/O. The Batch Mainframe is usually made up of sequential processes of low complexity, but very I/O intensive.

In a distributed architecture, you can parallelise this type of process (i.e. Spark) to reduce the elapsed time transparently to the application programmer. The main process is divided into different execution processes that access the data independently (shared-nothing vs. shared-everything architecture).

It is imperative to continue executing traditional COBOL PL/I processes with file access (read/write) sequentially. However, their performance can be dramatically improved by using state-of-the-art storage devices that dispense with rotational disks.

Network

The IBM mainframe is a standalone system. It does not require the use of network resources, as all process components run on the same hardware.

This difference is crucial when accessing database servers. The use of static SQL, together with the execution of processes and the database server in the same instance of the operating system, means that it is common to find massive data reading processes (OPEN CURSOR) to subsequently perform simple calculations in the COBOL or PL/I application program (SUM, AVG, etc.).

In a distributed architecture, these accesses are performed by dynamic SQL and involve a network hop (from the application container to the database server), which makes the process inefficient. The solution is to resolve these calculations in the database server, retrieve the information in blocks of multiple rows and increase the parallelism of the process.

Proposed solution

The proposed strategy for batch processing is based on a dual approach. It allows us to attack the "flow" of new projects and the migration of the "stock" of processes built on mainframe technology to the same open technical architecture based on Kubernetes.

The client has the option to;

- Migrate existing functionality by compiling COBOL PL/I programs on a Linux platform.
- Build new functionality on a distributed Spark architecture with access to the main Mainframe data sources (DB2 and sequential files).

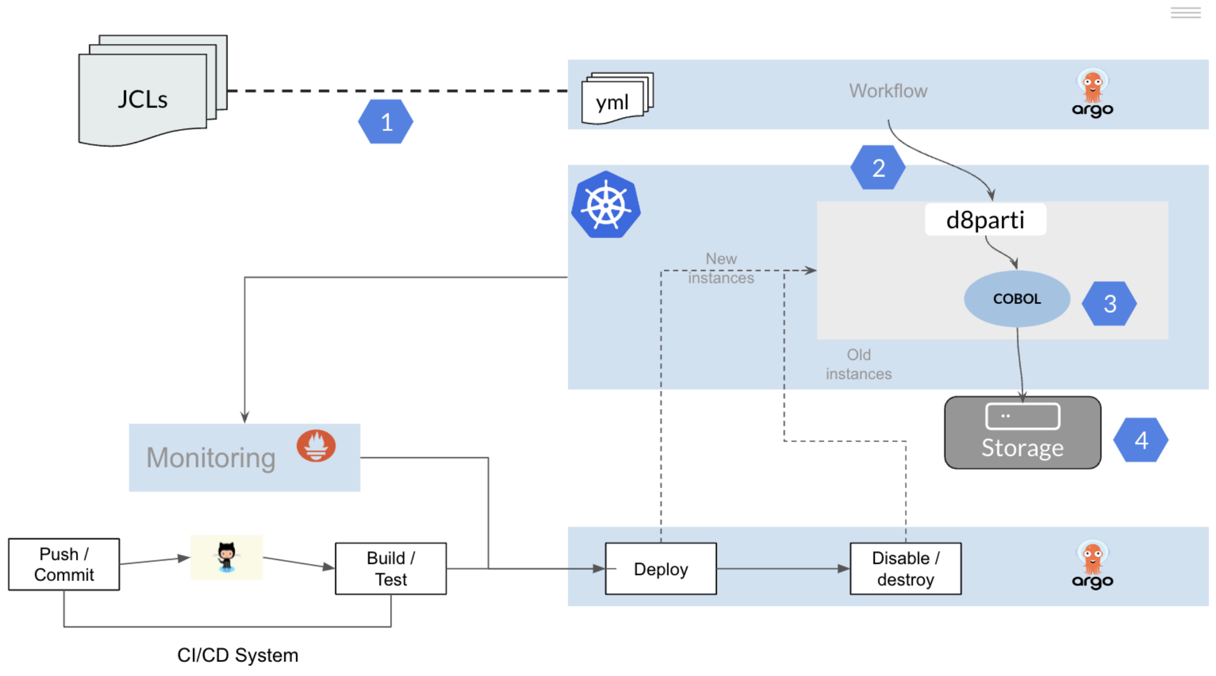
The two strategies are compatible, and you can combine COBOL and Spark programs in different steps of the same JOB. For example, STEP1 can execute a COBOL program and STEP2 a Spark program.

Stock migration

To migrate the processes built on Mainframe technology it is necessary to replicate the functionality described above on a Kubernetes cluster.

It is therefore essential;

1. Convert the JCLs (JOBs) to a tool or framework that allows the execution of workflows on a Kubernetes platform
2. Replicate JES functionality to enable scheduling and execution of COBOL PL/I programs on the Kubernetes cluster
3. Recompile the application programs
4. Allow access to data (files and databases)



Build new functionality

It may be beneficial to consider a distributed process architecture model based on Apache Spark for the construction of new Batch functionality.

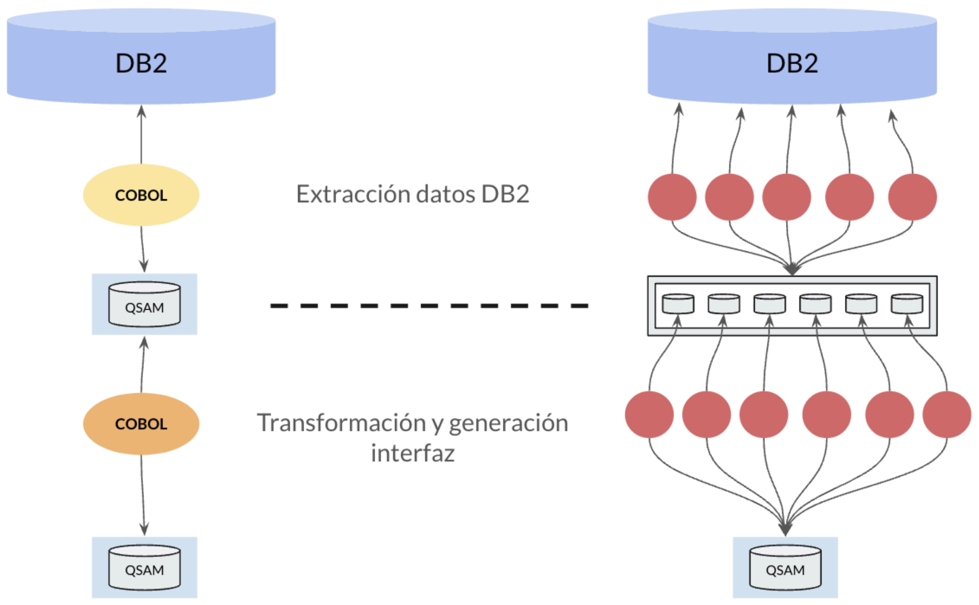
Spark processes share with COBOL processes the technical architecture defined above;

- The workflow definition tool, Argo, can be used to create jobs that mix steps in both technologies
- The Kubernetes platform
- Monitoring and logging tools
- CI/CD pipeline for container deployment on the platform
- Access to databases (DB2 / Oracle) and files.
- Etc.

Spark supports multiple data sources, including jdbc, which can be used to easily access DB2 on the IBM Mainframe platform.

You can access and transform mainframe files into different formats, including text, parquet, CSV, JSON, and more.

This is an example of how to convert a sequential process to a distributed Spark process.



(1) The DB2 database read process can be parallelised in contrast to the mainframe COBOL process. This can be achieved by raising a Spark pod (executor) for each of the table partitions (partitioned tablespaces).

(2) The retrieved information is loaded into the Kubernetes cluster memory or can be written to an S3-like intermediate storage (object storage) in a columnar format (parquet) without losing the original DB2 table structure or names.

(3) The information transformation process can also be parallelised. Each Spark executor's final result is consolidated and written to an output file.