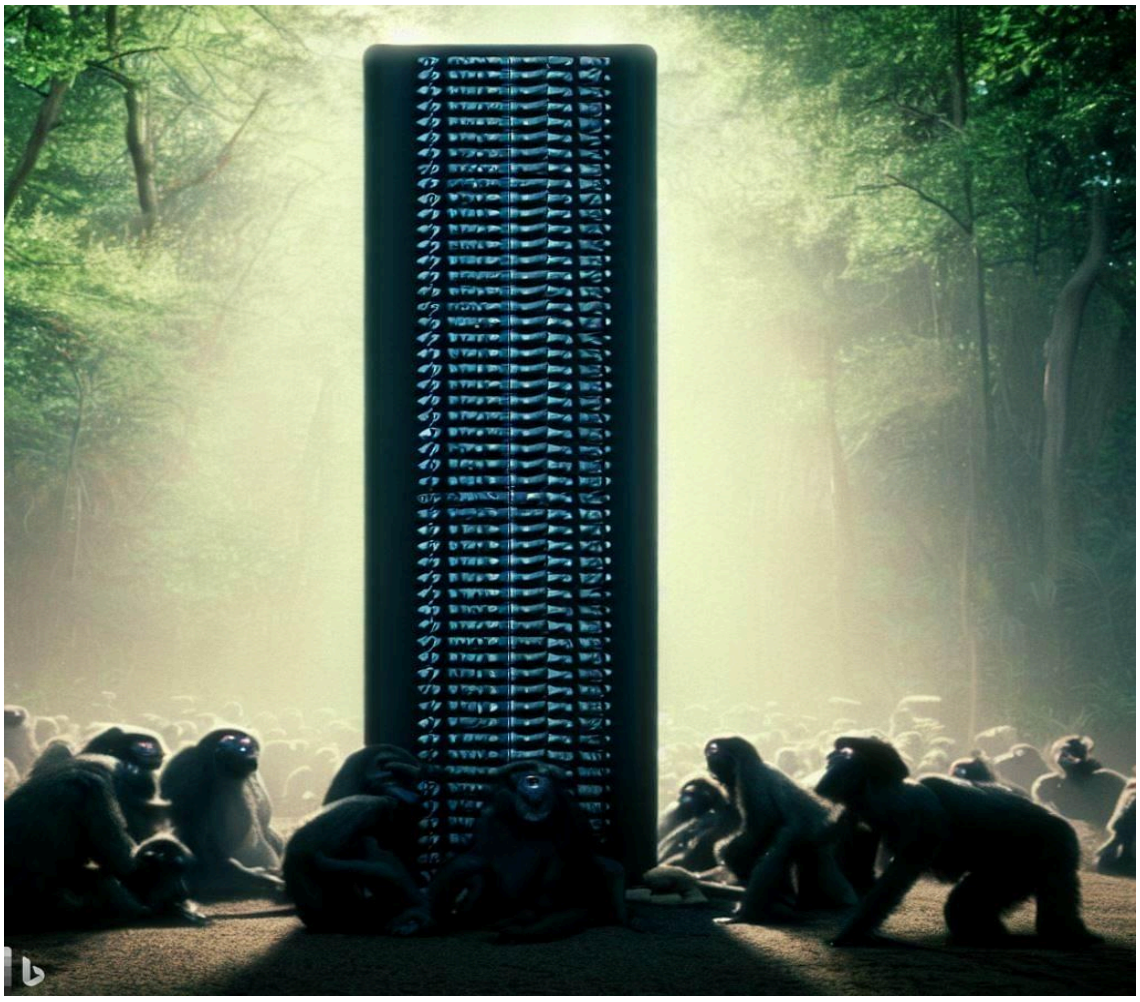
IBM Mainframe: Migración a una arquitectura open

Cómo desmontar un Mainframe y desplegar el código en Kubernetes



Introducción

¿Qué industria o sector de la economía es capaz de competir con tecnología desarrollada en los años 80's del siglo pasado?

Por distintos motivos (fuertes inversiones realizadas, pérdida de talento tecnológico y las consecuentes dificultades para ejecutar proyectos de transformación, etc.), esa base tecnológica es la realidad en gran parte del sector financiero.

La tecnología IBM Mainframe sigue teniendo una importante cuota de mercado en el sector financiero, la mayoría[1] de las grandes empresas del sector (bancos, aseguradoras, etc.) siguen manteniendo sus sistemas operacionales sobre tecnología IBM, con diseños y productos de finales del siglo pasado.

¿Cómo afecta el uso de este tipo de tecnologías a las organizaciones de IT de los bancos?

En estas compañías, los departamentos de IT suelen ser ineficientes, con unos elevados costes respecto al resto de industrias, un ciclo de desarrollo muy largo en la entrega de nuevas soluciones al negocio y una escasa capacidad para atraer o retener talento.

Sin embargo es posible seguir estrategias que permitan:

- Reducir los costes de operación, liberando recursos que pueden ser usados en otros negocios
- Aumentar significativamente la productividad de los equipos de desarrollo y en consecuencia incrementar la capacidad para entregar nuevos productos y servicios
- Atraer nuevo talento a la organización

Estas estrategias se apoyan en capacidades que permiten la migración del código desarrollado en la plataforma IBM Mainframe a una arquitectura open híbrida basada en Kubernetes.

Resumen ejecutivo

La tecnología IBM Mainframe fue determinante en el siglo pasado para la automatización de las operaciones del sector financiero, IBM dominó el sector hasta casi monopolizarlo, sin embargo

[1] De acuerdo a IBM, 44 de los 50 mayores bancos del mundo utilizan tecnología Mainframe (zSystems)

esa ventaja ahora se ha transformado en un lastre al impedir la innovación e introducción de nuevas soluciones.

IBM necesitaría hacer cambios disruptivos en su plataforma para poder ponerse al nivel de los principales players digitales. Eso supondría que millones de programas, escritos hace más de 20 años y que son la base del negocio bancario, dejaran de funcionar. Estamos por tanto, ante una plataforma de nicho, cada vez más alejada de los estándares y mejores prácticas del mercado.

¿Cómo eliminar la plataforma Mainframe, salvaguardando los principales activos de valor de las entidades (código COBOL y datos)?

Hasta hace poco, las estrategias para eliminar el Mainframe pasaban por la conversión del código COBOL o la emulación de la plataforma Mainframe en una arquitectura técnica distinta.

Sin embargo con la tecnología open source disponible actualmente el enfoque, creemos puede ser distinto;

- Compilar el código COBOL sobre plataforma Linux
- Las transacciones CICS/IMS y los procesos Batch se despliegan como microservicios sobre una plataforma Kubernetes
- Los microservicios COBOL serán interoperables con los que sean desarrollados en otros lenguajes.
- El código se gestiona mediante un pipeline automatizado (CI/CD)

Kubernetes es una plataforma ampliamente soportada y evolucionada por los principales players digitales, los programas COBOL de la plataforma Mainframe, migrados a esta arquitectura de ejecución, se comportan como cualquier otro microservicio, beneficiándose desde el primer momento de las ventajas de la plataforma (seguridad, monitorización, operación automatizada, despliegues, portabilidad, etc.)

Las entidades financieras pueden decidir donde ejecutar sus cargas de trabajo, On-prem (instalando Kubernetes en su DC), On-cloud (en cualquiera de los hyperscalers, Google, AWS, Azure, etc.) o en una arquitectura híbrida sin necesidad de realizar cambios en el código y sin ataduras a un entorno de ejecución propietario.

Los beneficios esperables al adoptar esta aproximación son:

- Reducción en el coste de operación. Este coste puede variar mucho entre entidades,

pero haciendo un ejercicio sencillo que incluya el coste del Hardware, las licencias de Software, el equipo de sistemas/mantenimiento de la plataforma y el coste del DC puede suponer ***más de 100M€ año*** en una entidad de mediano/gran tamaño

- Productividad de los equipos de desarrollo. En este caso, la promesa es ***duplicar la productividad*** de los equipos, acelerando el "time to market", implementando pipelines automatizados de despliegue de microservicios en Kubernetes, así como la posibilidad de desarrollar y testar COBOL de manera autónoma.
- Atracción de talento, unificando la plataforma técnica sobre la tecnología más demandada del mercado y permitiendo la introducción de nuevos frameworks (spark, AI, etc). Ya no existen equipos "legacy" especializados en tecnologías Mainframe y equipos especializados en tecnologías "modernas", todos comparten las mismas herramientas y plataforma, la única diferencia es el lenguaje de programación usado (COBOL, Java, Python, Go, etc.)

Antecedentes

No es objeto del presente documento ofrecer una descripción detallada de la arquitectura técnica de la plataforma de IBM Z Mainframe, sin embargo es importante destacar alguna de sus características más relevantes;

- Las raíces de la plataforma actual tienen su origen en los modelos S/360 de mediados de los 60 y posteriores evoluciones S/370 y S/390. IBM desarrolla en esos años sus propios estándares "propietarios" (EBCDIC, SNA, etc.) que siguen presentes en mayor o menor medida en los modelos actuales
- Las limitaciones de computación (procesador, memoria, acceso a dispositivos, terminales gráficos, etc.) en aquellos años tienen un reflejo directo en las capacidades ofrecidas por el Sistema Operativo, el lenguaje COBOL, el subsistema de almacenamiento, etc
- El altísimo coste de computación de estos primeros modelos llevó a IBM a establecer una estrategia muy estricta de retrocompatibilidad para proteger los activos software de los clientes, a día de hoy sigue siendo posible ejecutar programas desarrollados en los 70 en un Mainframe actual

Esto llevó a IBM a dominar el mercado en el sector financiero del siglo pasado. Sin embargo, todas estas ventajas se han transformado en un lastre para las instituciones financieras actuales, atándolas a una tecnología obsoleta que impide la evolución tecnológica.

La revolución tecnológica del siglo XXI (auge de internet, el móvil como herramienta de conectividad universal, el software open source, cloud computing, big data, IA, etc.) ofrece a las corporaciones unas capacidades de innovación sin apenas precedentes y por tanto enormes oportunidades de incrementar su relevancia en el mercado. Sin embargo y salvo excepciones, los grandes bancos han sido incapaces de aprovechar esta revolución, en comparación con otras industrias, para construir nuevas soluciones que incrementen su base de clientes y maximicen su rentabilidad.

Mientras nuevas compañías nacen cada día desarrollando productos sobre tecnologías que no existían hace 5-10 años, el sector financiero sigue invirtiendo enormes cantidades de recursos financieros y humanos en seguir evolucionando sus sistemas tradicionales basados en tecnología Mainframe, sin ser capaces de ofrecer a sus clientes experiencias similares a las que encuentran cuando consumen servicios digitales diseñados en los últimos 5 años.

¿Pero qué es lo que dice IBM sobre su plataforma Mainframe?

"*Los sistemas Z tienen un coste por debajo del resto de plataformas del mercado, son extremadamente robustos, seguros y permiten acelerar la implantación de soluciones de negocio*".

Esta aseveración es cuestionable si atendemos a la escasa o incluso nula presencia de tecnología IBM Mainframe en sectores clave distintos al financiero.

La tecnología Mainframe (zSystems) tiene un bajo TCO, por debajo de la competencia.

Vamos a descomponer este coste, para intentar entender qué hay detrás de esta afirmación.

Hardware, IBM ha ligado su sistema Z a una plataforma Hardware propietaria. Su estrategia en las últimas décadas no pasa por permitir la ejecución de cargas de trabajo Mainframe en arquitecturas de terceros (por ejemplo intel x86[2]), en su lugar ha llegado a acuerdos o desarrollado tecnología propia para permitir la ejecución de sistemas Linux sobre sus procesadores/arquitectura propietaria.

[2] IBM dispone de la tecnología necesaria para ejecutar una imagen de su sistema operativo (z/OS) sobre plataforma intel, sin embargo no permite su uso en entornos productivos

A pesar de seguir apostando por su división de Hardware, IBM ha sido incapaz de seguir el ritmo de la competencia (i.e. intel) en el desarrollo y evolución de sus procesadores y no cuenta con la economía de escala necesaria para competir en precio.

Los últimos movimientos de la industria (hyperscalers, Apple, etc.) hacia tecnología ARM y el desarrollo de la AI, van a seguir acentuando estas diferencias.

Software, independiente del coste total de las licencias de software, IBM ha empujado a sus clientes a un modelo de licenciamiento que prima los acuerdos a largo plazo y a garantizar un "consumo mínimo" para ofrecer un descuento sobre los precios de lista. En la práctica, supone eliminar cualquier incentivo por parte de los clientes para sustituir la tecnología de IBM.

Es frecuente encontrar en numerosas instituciones financieras, productos de IBM que tienen un peor desempeño al de la competencia (servidores BBDD, servidores J2EE, etc.) simplemente porque "no tienen coste", cuando en realidad dicho coste está incluido en un contrato a largo plazo muy difícil de renegociar. Para casi cualquier producto del portfolio de IBM, existe una alternativa en el mercado (software con licencia, opensource, gestionado por un hyperscaler, etc.), con un rendimiento superior o un coste de mantenimiento inferior.

Desarrollo/mantenimiento de aplicaciones, generalmente en el cálculo del coste de operación de una plataforma se excluye el coste asociado al equipo de trabajo necesario para el mantenimiento o desarrollo de nuevas aplicaciones de negocio. Dicho coste suele ser la partida más importante del presupuesto de un área de tecnología y está directamente asociado a la productividad de la plataforma.

Salvo excepciones, la productividad de un desarrollador sobre plataforma Mainframe es extremadamente baja por los siguientes motivos;

- Limitaciones propias de los lenguajes de programación tradicionales utilizados (COBOL, PL/I)
- Dificultad en las fases iniciales del desarrollo (codificación, prueba unitaria, prueba integrada, etc.), generalmente realizadas sobre plataforma Mainframe
- Ausencia de un proceso CI/CD moderno con herramientas que permitan un desarrollo ágil y colaborativo
- Separación radical entre los equipos Dev y Ops, distintos equipos de trabajo, distintas herramientas y procesos, distintos objetivos, distintas culturas, etc.

La tecnología Mainframe (zSystems) es robusta y segura.

Esta capacidad no es privativa de este tipo de tecnología.

Los sistemas desarrollados sobre arquitecturas Mainframe, son sistemas “monolíticos”, debido al elevado coste del Hardware y el limitado número de procesadores y memoria disponibles a finales del SXX.

Los procesos que se ejecutan sobre un sistema Z comparten memoria, así como el acceso al subsistema de almacenamiento. La manera más eficiente de comunicación[3] entre dichos procesos es mediante la utilización de memoria compartida, esto lleva a un acoplamiento muy fuerte entre los distintos productos y sistemas desarrollados.

No existen APIs para exponer la funcionalidad de negocio, acceder a una base de datos, etc. en su lugar los programas de aplicación intercambian direcciones de memoria (punteros) con estructuras de datos pre-establecidas.

Este tipo de arquitecturas de proceso, sólo pueden crecer verticalmente, añadiendo más recursos (procesadores, memoria, etc) al Hardware[4] existente. Es por tanto crítico, asegurar una tasa de errores/fallas muy baja en la plataforma Hardware.

La fiabilidad y seguridad de la que habla IBM hace *referencia a su supuesta mínima tasa de errores en la plataforma Hardware*. Sin embargo, nadie está a salvo de un error humano o una mala decisión técnica.

En las arquitecturas modernas, por ejemplo basadas en microservicios/kubernetes, la fiabilidad del Hardware deja de ser crítica, el crecimiento se realiza de manera horizontal, desplegando nuevas instancias de los componentes sobre una infraestructura de red, almacenamiento y cómputo distribuida.

Por otro lado y en caso de un error irrecuperable en un Mainframe, el mecanismo de contingencia más común es disponer de una configuración Hardware equivalente en un data-center alternativo. La estrategia pasa por realizar, mediante productos de IBM, una réplica

---

[3] Don't communicate by sharing memory; share memory by communicating

[4] En una arquitectura Sysplex se puede configurar un cluster de máquinas que comparten memoria, pasando de una arquitectura monolítica a un monolito distribuido aún más complejo de mantener

de los datos existentes en el data-center principal, sobre el data-center alternativo.

Una arquitectura moderna (stateless) permite descomponer los sistemas en distintos servicios independientes e implementar total o parcialmente estrategias de recuperación activo-activo[5]. No es necesario invertir en data-centers alternativos pudiendo usar las capacidades de los hyperscalers para asegurar la fiabilidad y robustez (despliegue híbrido on-prem / on-cloud multi AZ en una única región, despliegue multi región, despliegue en distintos proveedores cloud, etc.).

La tecnología Mainframe (zSystems) permite acelerar la implantación de soluciones de negocio

Realmente el proceso y herramientas utilizadas por un desarrollador COBOL apenas ha cambiado en los últimos 30 años.

IBM no ha empleado tiempo y recursos en mejorar las capacidades de los productos tradicionales (COBOL, CICS/IMS, Batch, etc.) sobre las que se han construido la mayoría de las funcionalidades críticas bancarias, en su lugar se ha dedicado a añadir capacidades “open” (Linux) a su plataforma Mainframe.

El resultado de esta estrategia es bastante cuestionable, las soluciones tradicionales siguen presentando enormes carencias y no son aptas para soportar nuevos productos y servicios con agresivos “time to market”. Por otro lado, las soluciones “open” no tienen sentido comparándolas con las posibilidades que ofrecen otras compañías o los hyperscalers, la capacidad de las organizaciones para mejorar su ciclo de desarrollo se ve limitada a las opciones que IBM considera válidas: las decisiones tecnológicas estratégicas y las rutas de innovación dejan de estar en manos de las organizaciones y dependen exclusivamente de la estrategia comercial de IBM.

Adicionalmente, IBM lleva tiempo invirtiendo la caja (Cash Flow Operativo) que obtiene con esta tecnología en distintas iniciativas de futuro, pero parece que en todas ellas ha llegado tarde o con un enfoque insuficiente (cloud, AI, etc.). La reciente compra de Redhat parece la última oportunidad de IBM para evitar convertirse en una compañía irrelevante en los próximos años.

---

[5] La configuración activo-activo de las bases de datos está condicionada por la tecnología utilizada

Solución propuesta

Si bien no es posible cambiar “en un abrir y cerrar de ojos” la tecnología y paradigmas arquitecturales en los que se basa en buena medida las operaciones de la industria, existe un camino bien estructurado para la evolución estratégica de las plataformas Mainframe hacia propuestas similares a aquellas que fundamentan el éxito de los grandes players digitales.

Problemas en el pasado

Para entender este cambio de enfoque, intentaremos explicar brevemente cómo se han abordado tradicionalmente este tipo de iniciativas y por qué no han sido exitosas en una institución financiera de mediano/gran tamaño.

Conversión del código a un lenguaje de programación moderno

Bajo este enfoque el código COBOL, se “parsea” y convierte de manera automática (o semiautomática) a otro lenguaje de programación, por ejemplo java.

- Los sistemas transaccionales, CICS/IMS, COBOL, DB2 se migran a una entorno J2EE, java, base de datos relacional (Oracle, PostgreSQL, etc.)
- Para emular las sentencias de los monitores transaccionales (especialmente CICS), deben crearse “utilidades” que repliquen dicho comportamiento[6]. La complejidad depende del número y tipo de sentencias utilizadas en los programas de aplicación
- Las sentencias de acceso de la base de datos DB2 son estáticas, deben pre-procesarse para su adaptación a la base de datos destino (Oracle, PostgreSQL, etc.)
- El Batch presenta particularidades adicionales. Para simular el funcionamiento de los JCLs/JES, los JCLs generalmente se transforman en bash y los procesos java se ejecutan sobre servidores de aplicaciones J2EE

Este enfoque presenta una serie de inconvenientes, que describimos a continuación.

Como hemos mencionado anteriormente, los sistemas desarrollados sobre plataforma ZSystems están altamente acoplados (arquitectura monolítica), esto es especialmente crítico en determinados módulos sobre los que descansa lógica aplicativa común.

Las posibles alternativas para enfrentar este problema, son siempre complicadas de gestionar;

[6] Envío/recepción de mensajes, formateo de fechas, uso de TS QUEUE, gestión cabeceras EIB, etc.

- Crear una arquitectura de convivencia que nos permita hacer un migración progresiva del código[7]. La propuesta de IBM pasa por desarrollar en java sobre plataforma Mainframe y permitir la comunicación entre los programas COBOL/Java mediante JNI, esta opción carece de sentido excepto para el departamento comercial de IBM
- Ir a una estrategia de Big-Bang en la que no sea necesario implementar convivencias, si bien este enfoque puede ser válido en aplicaciones autocontenidas, de bajo volumen/criticidad, es inviable en instituciones financieras de mediano/gran tamaño por el enorme riesgo/coste que implica
- Mantenimiento duplicado de los sistemas comunes, tendríamos hasta el fin del proyecto dos aplicativos (uno en COBOL y otro en java). Esta opción podría ser válida siempre que el número de sistemas a duplicar sea acotado y el mantenimiento de los mismos muy limitado

Por otro lado, es importante entender el resultado final del proyecto para analizar el impacto sobre la organización de tecnología y especialmente sobre el equipo de desarrollo.

El código COBOL desarrollado en los últimos 30 años, normalmente presenta las siguientes características;

- Insuficiente o nula documentación[8]
- Dificultad para comprender el código. Para optimizar el uso de memoria, los programas están limitados a 8 caracteres, el nombre de las tablas DB2 suele tener las mismas limitaciones, las variables utilizadas en los programas no son significativas (en lugar de utilizar como estado civil, soltero, casado, etc. se usa una variable numérica de longitud 1), lo mismo sucede con los códigos de retorno, errores, etc.
- El modelo de integridad referencial no suele estar implementado sobre la base de datos DB2[9], debe deducirse por medio del código de aplicación
- Restricciones del lenguaje COBOL para optimizar el uso del procesador. Variables estáticas, alfanuméricas de longitud fija, numéricas con/sin signo de longitud fija con máscara decimal, decimales empaquetados, etc.
- Lenguaje procedural. El código se empaqueta en distintas “subrutinas” que son llamadas

[7] ¿Cómo conectar un programa COBOL ejecutado en un Mainframe con una clase java ejecutada en un servidor J2EE sobre plataforma intel?

[8] En la mayoría de los casos la única documentación es el propio código COBOL

[9] Debido al elevado coste de implementarlo en las primeras versiones del producto

de manera dinámica o estática mediante el intercambio de estructuras de datos (Copybooks)

Este código es transformado mediante un proceso automático en una clase java. Como resultado de este proceso;

- El equipo de desarrolladores COBOL Mainframe necesita un proceso de adaptación no menor para volver a ser capaz de mantener el código de manera eficiente. Dependiendo de la estrategia de conversión de código usada el resultado puede llegar a imposibilitar dicho mantenimiento
- Un equipo nuevo de desarrollo con “skills” avanzados en java tendrá muy complicado comprender la lógica y la estructura del código generado para poder mantenerlo en el futuro

Emulación Mainframe en una arquitectura Hardware distinta

En este caso el código COBOL se compila en una nueva plataforma Hardware, emulando las funcionalidades de los productos de IBM necesarios para la ejecución del mismo (sentencias CICS/IMS, JCLs, acceso a la Base de Datos DB2, etc).

Ejemplos de este enfoque son las iniciativas de Oracle para migración del código Mainframe a Tuxedo/Oracle DB o la suite de productos de Opentext (Microfocus).

Los principales inconvenientes de este enfoque son los siguientes;

- Todos los problemas inherentes a la plataforma Mainframe se arrastran a la nueva arquitectura, sistemas monolíticos, alto acoplamiento, dificultad en el desarrollo, etc.
- Se sustituye una plataforma técnica (IBM Mainframe) sin apenas evolución en los últimos años, por una plataforma propietaria que difícilmente tendrá mejor evolución/soporte que la original
- Como en el caso anterior, es necesario proveer una arquitectura de convivencia para permitir una migración progresiva del código evitando estrategías de Big-Bang

Salvo en el caso de entidades con un uso muy marginal de tecnología IBM Mainframe, el resultado final de este tipo de iniciativas es un proyecto muy largo y costoso que deja a las entidades en una situación igual o incluso peor tecnológicamente.

La promesa de reducción de costes de operación puede verse rápidamente comprometida por el coste/plazo del proyecto. El resultado final es un callejón sin salida que irremediablemente

implicará a medio plazo un nuevo proyecto de transformación tecnológica.

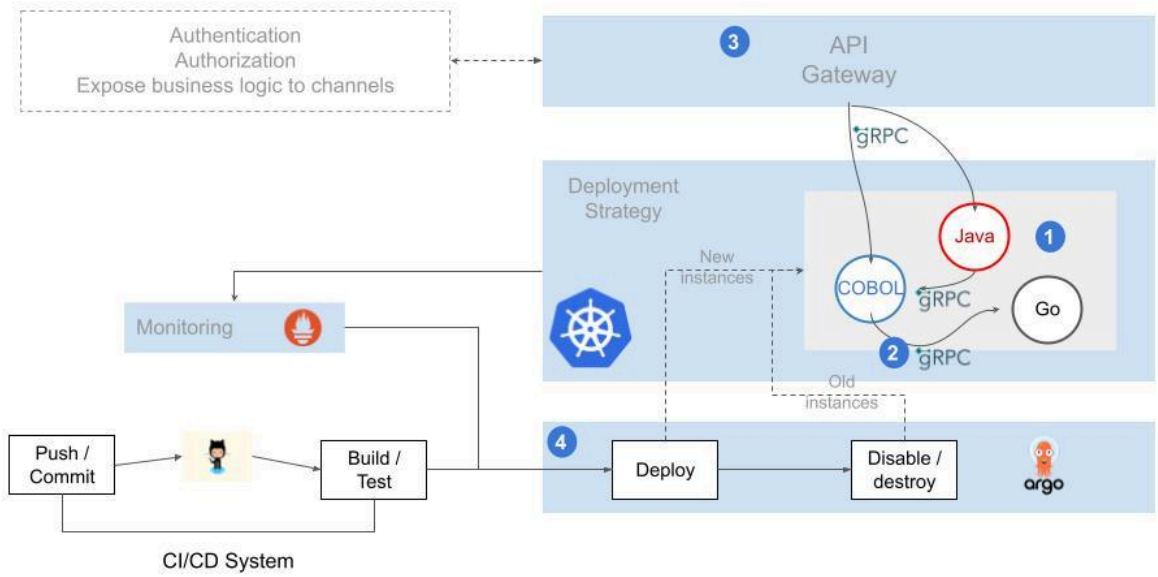
Solución Driver8 Software

La solución de Driver8 se basa en las siguientes premisas;

- *Enfoque dual*. Separando el flujo de nuevos proyectos, del stock de aplicaciones Mainframe
    - Flujo. Un modelo de arquitectura de microservicios que permite interceptar las nuevas iniciativas de negocio para su despliegue en una infraestructura Next-gen basada en Kubernetes
    - Stock. La migración de manera progresiva y minimizando el riesgo, de las cargas de trabajo COBOL (Online/Batch) Mainframe a dicha infraestructura Next-gen
- *Uso de proyectos Open Source*. La arquitectura destino se basa en productos y estándares soportados por las principales compañías tecnológicas. Las aplicaciones podrán desplegarse sobre infraestructura On-prem (en los DC ya existentes), On-Cloud (Google, AWS, Azure disponen de soluciones válidas en distintas geografías) o en una arquitectura híbrida
- *Plataforma integrada*. Sobre la misma infraestructura y arquitectura técnica se podrá desplegar nueva funcionalidad (microservicios construidos en Java, Python, Go, etc.) y el código COBOL del Mainframe. Los desarrolladores comparten las mismas herramientas de CI/CD (repo, pipelines, etc.), siendo la única diferencia el lenguaje en el que se construyen los microservicios
    - Las transacciones Mainframe (Online) se despliegan como microservicios COBOL/gRPC en un cluster Kubernetes
    - En el caso de los procesos Mainframe Batch, los JCLs se convierten a ficheros YAML/JSON y son interpretados por un Scheduler Batch de Kubernetes especializado en la ejecución de procesos masivos (cómputo y datos)
- *Mecanismos de convivencia*. Se definen dos niveles de convivencia, para garantizar una migración progresiva y sin riesgos del código Mainframe. Cada entidad podrá definir su plan y establecer los plazos para acometer el proyecto de migración;
    - Convivencia entre plataforma Next-gen y Mainframe. Permite el acceso en lectura/escritura a la plataforma Mainframe desde los microservicios desplegados en la plataforma Next-gen (Kubernetes), mediante dos mecanismos básicos de conexión, acceso SQL a la Base de Datos DB2 (proxy sobre el driver odbc DB2) y

ejecución de transacciones CICS/IMS (conexión sockets TCP/IP con los gestores transaccionales)

- Convivencia entre microservicios, independientemente del lenguaje de programación utilizado (COBOL, Java, Go, etc.), exponen una interfaz siguiendo el estándar gRPC que permite la integración transparente de los mismos (las Copybooks COBOL se exponen como protos gRPC)



(1) La solución permite la codificación de servicios (implementación de operativa bancaria) en lenguajes modernos y atractivos para los desarrolladores. A su vez permite el aprovechamiento de piezas codificadas en lenguajes "legacy" cuya recodificación resultaría en un gasto de recursos innecesario

(2) La comunicación entre los diferentes servicios, comunicación interna, se implementa mediante un protocolo ligero y eficiente

(3) Los servicios son invocados desde frontales o sistemas de terceros (plataformas de pagos de terceros, software de partners u otras entidades, etc.) a través de un

mecanismo de exposición securizado, resiliente y fácilmente escalable

(4) *La operación está soportada por pipelines para la automatización de los despliegues y capacidades avanzadas de observabilidad que permiten una visión integrada y consistente de todo el flujo aplicativo y del estado de salud de los elementos involucrados*

(5) *La seguridad está embebida en la plataforma con componentes estándar que implementan mecanismos avanzados para proporcionar las capacidades necesarias de acuerdo a los requerimientos de la industria financiera*

Glosario de términos

AI: Inteligencia artificial

ARM: Arquitectura de procesadores de 32 y 64 bits con un con un conjunto de instrucciones reducido (Advanced [*RISC]* Machine) que ha resultado óptimo inicialmente para un escenario de computación como el de los teléfonos móviles. El conjunto de instrucciones optimizado conlleva a su vez un conjunto menor de transistores lo que a su vez impacta en un menor consumo de energía y menores costes de producción. El éxito y ubicuidad de los que esta arquitectura goza en la actualidad está planteando la posibilidad de incluso incorporarla en servidores corporativos donde aspectos como el menor consumo de energía resulta de gran relevancia.

AZ:(Availability Zone) Despliegue de infraestructura (cómputo, networking, seguridad, almacenamiento, operaciones) que es capaz de ofrecer servicio de forma autocontenida e independiente . Los cloud providers estructuran en diferentes AZ's, separadas físicamente incluso por decenas de kilómetros de distancia, su "offering" en una "región geográfica" para asegurar la disponibilidad de su servicio.

Batch: Esquema tradicional de procesamiento en lotes, inicialmente asociado a la lectura/escritura masiva de ficheros. Una gran parte de los procesos en la industria bancaria se siguen ejecutando de esta forma.

CI/CD: (Continuous Integration / Continuous Deployment). Procesos y mejores prácticas técnicos, de seguridad y de gobierno para soportar a los equipos de desarrollo y operaciones (devops) durante el ciclo productivo de desarrollo de software que se extiende desde la compilación del código desarrollado hasta su despliegue en la plataforma elegida pasando por la realización de pruebas automatizadas (unitarias, de integración), validaciones de seguridad (descubrimiento automatizado de vulnerabilidades en el código, certificación de la validez de la cadena de suministro del software), gestión de la configuración y gestión de "rollback" para el restablecimiento rápido de una situación estable ante problemas inadvertidos en las nuevas versiones. Estos procesos y prácticas descansan en un "toolbox" bien establecido que permite rápidamente el setup de las capacidades técnicas para que los equipos puedan centrarse rápidamente en el desarrollo y release de productos.

COBOL: Lenguaje de programación procedural ampliamente usado en las instalaciones IBM Mainframe, tiene sus orígenes en los estándares CODASYL de los años 60.

CICS: Monitor o gestor transaccional. Los Mainframes, así como el lenguaje COBOL fueron creados en una época en la que no existían terminales interactivos y todo el procesamiento se realizaba en Batch. Tanto CICS, como IMS, sirven para poder conectar una red de terminales (originalmente SNA) a un Mainframe, asegurando la integridad de las operaciones de escritura.

DB2: Base de datos relacional, sustituye en los 80s las anteriores bases de datos jerárquicas/red de IBM y los sistemas de ficheros

EBCDIC: Sistema de codificación propietario de IBM

gRPC: Framework para la comunicación eficiente RPC (Remote proceduce call) entre servicios distribuidos, construidos en distintos lenguajes de programación

JCL: Lenguaje para definición de cadenas de trabajos Batch, heredero de los antiguos sistemas basados en tarjetas perforadas

JES: (Job entry subsystem), subsistema del Mainframe encargado de ejecutar y planificar los procesos por lotes (Batch)

JNI: (Java Native Interface), Estándar Java para la comunicación con programas “nativos” escritos en lenguaje C

IMS: Ver definición CICS

J2EE: (Java 2 platform, Enterprise Edition), estándar para el desarrollo de código Java en servidores de aplicación

Kubernetes: Plataforma open source para la gestión de contenedores inicialmente desarrollada por Google

PL/I: Lenguaje de programación propietario de IBM para sus sistemas Mainframe

SNA: (System network architecture), arquitectura de red propietaria creada por IBM en los años 70. La implementación en la plataforma Mainframe se realiza mediante el producto VTAM

Spark: Motor para el procesamiento de datos distribuidos a gran escala, sus capacidades multi-lenguaje (Java) y SQL permiten la conexión a multitud de gestores de bases de datos, incluido DB2 Mainframe.