Accelerating Digital Initiatives Using Data Virtualization to API-Enable Mainframe Applications and Data

A Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

February 2018

Sponsored by

Table of Contents

1 Executive Summary

Legacy Systems Form the IT Backbone of Many Organizations – They were the catalyst of the Information Age, the IT backbone of modern business, and eventually became what we referred to today as *legacy systems*. Built before the advent of the Internet, mobile devices or cloud, they are developed with technologies that most young developers have never heard of, and if they have, they've probably never used them.

Despite that these systems may come across foreign to millennial programmers, they still support the bulk of transactional workloads that underpin numerous financial organizations, medical environments, airline companies, industrial companies, and so on. If these *backend systems* fail or stop, many business processes will grind to a halt. Airlines won't be able to sell seats, banks won't be able to transfer money, and in medical environments patient data won't be accessible anymore.

We are referring to the IBM Z platform and the mainframe applications and data that were developed many years ago with languages such as Cobol, PL1, and CICS, and with database technologies such as VSAM, IMS, IDMS, and Adabas. Although they are decades old and have a unique user experience, their functionality is still very valuable to organizations and the data they manage is crucial for organizations to continue operations.

Digital Transformation – Crowding the list of CIO imperatives today is *digital transformation*. "Digital transformation[1] is the […] transformation of business activities, processes, competencies and models to fully leverage the changes and opportunities of digital technologies and their impact across society […]." One component of digital transformation is the *customer-facing app*. External partners, such as customers, partners, agents, and suppliers want and need direct access to an organization's IT systems and data, without any human intermediate but through new systems of engagement, such as through smart phones, tablets, and the Web.

Technically this means that the functionality offered by mainframe systems and the data they store must be *externalized*. By using apps running on modern devices (such as tablets and smartphones), TV sets at homes, portals, websites, and kiosk machines in public environments (such as sport stadia, hospitals, airports), these external parties must be able to invoke this functionality and retrieve and manipulate the data.

REST-Based APIs and Secure Gateways – Over the years several technologies, such as messaging technology, enterprise services busses, and gateways, have been used with mixed results to wrap the older mainframe systems and externalize the data and functionality to make it available on today's devices. Unfortunately, most of these projects didn't have satisfying results. The problem is that they are too "heavy" and too cumbersome, for example, when running on small, mobile devices. Technology is needed that is, first, lightweight enough to be used on small devices, second, uses existing technology available on those devices and, third, is reliable, secure, and robust enough to access mainframe applications and data without denigrating quality of service. This is the domain of *REST-based API gateways*.

[1] i-Scoop, "Digital transformation: online guide to digital business transformation"; see https://www.i-scoop.eu/digital-transformation/

This Whitepaper – This whitepaper describes the importance of externalization of mainframe systems to develop customer-facing apps for mobile devices and the cloud. It also describes an API gateway solution that offers a lightweight REST-based interface that fits the needs of today's digital application development and that integrates fully with legacy mainframe applications. This solution is a combination of IBM's API gateway product called *z/OS Connect Enterprise Edition* and *IBM Data Virtualization Manager for z/OS*. It also explains the importance of the REST interface for customer-facing apps.

Although the whitepaper is somewhat technical, because it explains products and technologies, it's basically about customers and other external parties. Digital transformation and externalization of mainframe systems is about improving customer experience, customer service, customer delight, and about making our partners more efficient. It's about extending today's modern mainframe into all aspects of digital transformation, enabling external parties to integrate with our mainframe systems more directly and more freely, using any device or platform.

2 New Systems of Engagement for Digital Transformation

The Digital Transformation – There was a time when only employees were using an organization's IT systems. Access was through terminals and PCs, and none of them accessed the IT systems remotely. Everything was controlled and secured. No external parties had access.

So much has changed in the meantime. New systems and technologies for engagement and deployment have been introduced. With the introduction of the Web, mobile devices (such as tablets, smartphones, and smart watches), and the cloud, users can access IT systems from anywhere. Moreover, the group of users is not restricted to employees anymore, users can be external parties, such as customers, partners, suppliers, and agents.

For these new forms of digital engagement and new external users, apps must be developed that run on these new devices and platforms and that securely access the existing IT systems; see Figure 1. These *customer-facing apps* for external parties are a key success factor for every organization's *digital transformation*.

Examples of Customer-Facing Apps – Many examples exist where this need for developing customer-facing apps by externalizing data and functionality is clear:

Finance: Currently, many banks have developed first generation online services and some customer-facing apps for mobile devices. With these apps, customers can easily check their bank accounts and make bank transfers. For more and more customers, the easiness and functionality of these apps can help determine which bank they will use. Increasingly, the same customers expect a richer digital experience that is personalized to their unique needs.

Medical: Hospitals want to allow patients themselves to access their personal, medical data via the cloud, via the Web, and via mobile devices. For example, after giving blood and urine for a checkup, it must be possible for patients to online, while still maintaining data security. Information may need to be aggregated from multiple sources and be presented in a user-friendly and understandable way.

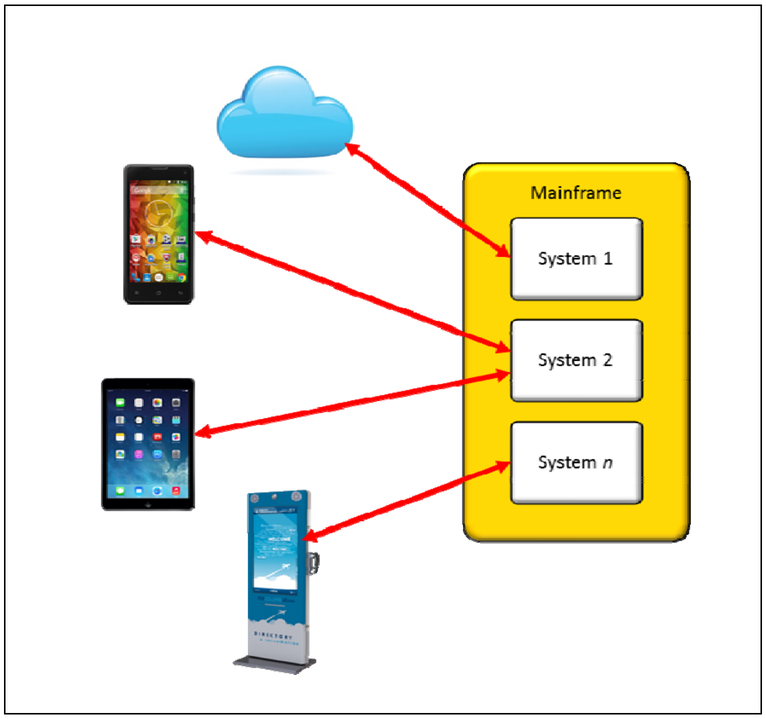

Figure 1 *Customer-facing apps running on modern devices are a key success factor for every organization's digital transformation.*

Insurance: Over the years numerous insurance companies have developed separate systems for supporting different insurances products. Customers don't want to be confronted with siloed disconnected insurance data. On their mobile phones they want to see an integrated, 360 degrees overview of the status of their entire insurance portfolio with their insurance company, including for example their property insurance, car insurance, life insurance, income protection insurance, and health insurance. Today's more robust mobile apps can allow customers to make a wide range of simple changes to the policies and include many value-added features to serve customers and ultimately sell more products.

Transport: Transport companies were early in noticing the benefits of allowing employees and customers the facility of mobile enablement. Today, some of these applications may look somewhat simplistic, because they only show whether the parcel has been shipped, has arrived in the country, and so on. With the technology of today, the tracking information displayed can be a lot more precise. It's possible to show to the customer a map that indicates live where the truck, train, ship, or airplane with the customer's parcel is located at that specific moment and how long it will take before the parcel arrives.

Cross-Organizations Apps – The most exiting examples of customer-facing apps are those that combine data and functionality coming from multiple organizations. For example, music collectors would be helped with an app that shows on a map that there are record stores in the neighborhood that have a record or CD in store that is on their wanted list. Such a cross-organizational customer facing app would need access to a GPS map service, a list of record stores and their locations, and access to the inventory management systems of as many stores as possible, and to a system where collectors can enter their wanted list of records.

Cross-organization app are a reality today, with many mobile and Web apps are already available that do exactly this, such as Booking.com and Expedia.

In industries such as banking, directives and regulations are defined to help developing such apps. For example, in the EU the *Payment Services Directive*[2] (PSD2) has become active. PSD2 is a data and technology-driven directive that aims to improve competition, innovation, and transparency across the European payments market. At the core of PSD2 is the requirement for banks to grant third-party providers access to a customer's online account/payment services. This will lead to new players on the financial market offering services via apps that access the data and functionality deployed in the banks' IT systems. The effect will be that banks have to decide on whether they want to become a banking "utility" or an "everyday bank" playing a central role in customers' daily lives.

Agents and other intermediates in the insurance branch have similar needs. They may want to offer to their customers the use of a mobile app or web portal that allows them the see their entire insurance portfolio regardless of with which insurance company the customer has signed a contract.

Externalization as a Business Driver – Allowing external parties to access older IT systems through customer-facing apps is not a fad. It can be a business driver, because it influences the competitiveness of an organization, its productivity and efficiency, and its customer friendliness. It has become a matter of importance. For example, the easiness with which customers can check their financial situation via their smart phones, track orders via the web, change contracts, order products, will increasingly determine which bank, which parcel service, which retailer, and which airline they'll use. It has become a key distinguishing factor. As indicated, the customer-facing app plays a key role in an organization's journey to digital transformation.

3 Customer-Facing Apps and the Mainframe

Externalizing Data and Functionality – Most often, the data presented by customer-facing apps resides in databases running on server machines; see Figure 2. Examples of data elements are customer addresses, invoices, stock levels, insurance agreements, and orders. Likewise, the functionality that apps invoke are modules residing in current IT systems. Examples of such modules are transfer money, book flight, calculate stock level, and predict a parcel's estimated delivery time. Functional modules are also called *business logic*.

To make this data and functionality available to apps, it must be *externalized*. Technically, interfaces must be developed that make it possible to invoke modules within the IT systems and to retrieve and manipulate data in the databases; see Figure 3.

Externalizing Mainframe Data and Business Logic – When databases and IT are developed on modern platforms with structured development languages and modular internal structures, it's not that difficult to create technical interfaces that can be invoked by apps to get access to the data and business logic. But for many organizations, who started to develop their IT systems many years ago on mainframe systems, externalizing their data and business logic is a technological challenge. These databases are not popular SQL-based databases. Data is not stored in flat, relational data structures and there are no high-level languages to easily query the data. In addition, the business logic can be deeply hidden in the code of old applications with no clear modular structure.

[2] Wikipedia, Payment Services Directive; see https://en.wikipedia.org/wiki/Payment_Services_Directive

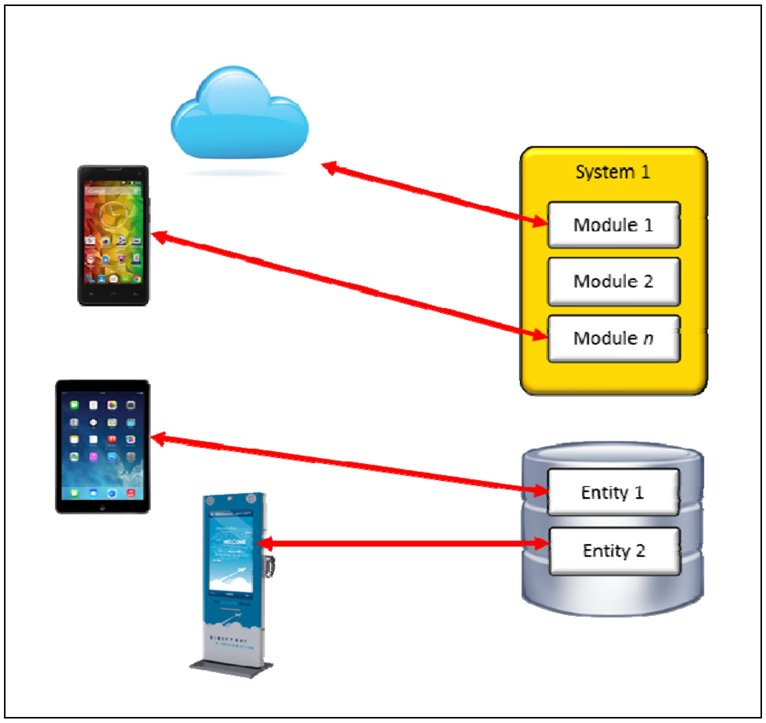

Figure 2 *The data presented by customer-facing apps and the functionality they invoke resides on server machines.*

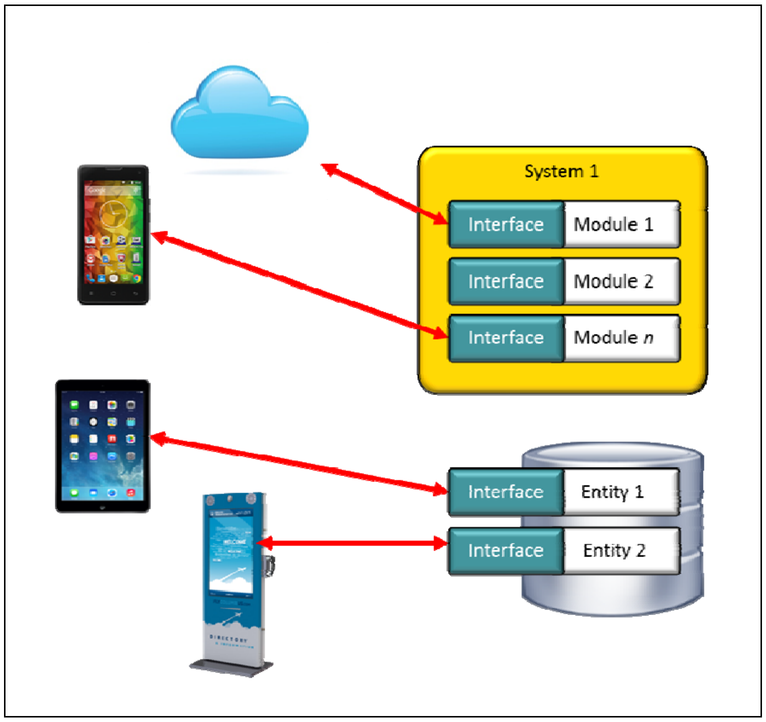

Figure 3 *For externalizing data and functionality special interfaces are required.*

The fact that these mainframe-based IT systems have been around for so long doesn't mean that the data they store and the business logic they encompass have become obsolete. On the contrary. The data is still indispensable to organizations. Bank transfers are still stored in these mainframe systems, but also parcel tracking data, flight reservations, patient statuses, insurance policies, and so on, are still being stored there. Additionally, most of an organization's key business logic may still run on a mainframe. Examples of business logic still operating on mainframes are: checking the correctness of a new delivery address, safely and securely processing a bank transfer, copying and transforming data from an operational database to the data warehouse for reporting and analytics, and calculating the amount of dead stock. Business logic also contains all the rules for governance and auditing.

The bottom line is that all this data and business logic residing on the mainframe systems form the IT backbone of an organization. Regardless of the technological complexity, it has to be unlocked. It must be externalized to support customer-facing apps, to help an organization with their digital transformation.

4 Externalizing Data and Business Logic the Old Way

Technologies for Externalizing Data and Business Logic – This need to externalize data and business logic and to link applications together has existed for many years now. Therefore, many mature technologies and solutions already exist that can be used to externalize mainframe applications. But most of these "older" solutions are too heavy-weight for the new customer-facing apps. On the client side it's important that the solution can operate on devices with a very small footprint, such as smartphones and watches. And even on the Web the solution should not take minutes to download before the user can invoke the app.

Most of the old solutions are also proprietary solutions, often based on technologies developed by a single vendor. So, all the connections must be made with one and the same tool. This means that if a bank wants to externalize some of their data, every user on every device must install that solution as part of the app. Imagine that every financial organization, retailer, and airline company selects a different solution, the number of solutions users must install on their devices will be phenomenal. Additionally, for a cross-organization app to extract data from multiple organizations, it must deal with all the different technical solutions.

Custom-Coded Solution – Some organizations have tried to develop their own home-made solution. Many of these projects failed because of lack of sufficient technology skills. On PowerPoint such a solution looks simple, but developing a scalable, high-performance, robust, and secure solution is complex. Because a custom-coded solution is a proprietary solution, it's hard to make it work together with applications and systems not controlled by the organization itself. Keeping this solution up to date in the future is not straightforward either.

Messaging Technology – Messaging technologies are proven solutions to send messages back and forth between applications on different machines. Lately, messaging technology has become hot again because of the concept of *fast data* in general and the *Internet-of-Things* in particular. Products such as Apache Kafka and RabbitMQ are very powerful products and can connect remote client apps to server environments such as mainframes. But unfortunately, such technologies require the installation of software modules on client machines. In other words, they can't be considered lightweight solutions.

Enterprise Service Bus – An Enterprise Service Bus or ESB can be used to implement Service Oriented Architectures. They can be used to connect clients to servers. But, as with messaging, this is not a lightweight solution on the client machines. In fact, for developing customer-facing apps, ESBs have the same drawbacks as messaging technologies.

XML – Many of the products used to send messages have used XML as messaging format. This has several benefits, including the capture of and meta data inside the messages. XML, for the purpose of developing customer-facing apps, has two drawbacks. First, because XML was originally meant to be a markup language, it offers a lot of functionality not used by customer-facing apps, and making XML messages quite verbose. Second, processing XML messages is not very fast.

Most of the above solutions were developed to connect big systems with each other, not tiny systems with big systems. They were developed to be scalable, but not for thousands of users concurrently. Their lack of support of stateless solutions limit the scalability.

5 REST as API for Accessing Applications and Data

REST is the Lingua Franca for Customer-Facing Apps – What is needed is a simple, fast, flexible, lightweight, and popular technology for allowing customer-facing apps to access data and business logic. Currently, the most popular API that meets these requirements is *REST* (Representational State Transfer). REST has become the lingua franca of the cloud and the mobile world. Well-known websites, such as Google and Facebook, use REST to let their web interfaces communicate with their own systems. Most mobile apps use REST as well.

The benefits of REST are:

- **Simplicity:** The commands to invoke business logic or retrieve data are straightforward HTTP calls. If they want to invoke a remote service, they only have to specify the URL location and the parameters. For example, the following REST call may be sufficient to retrieve the details of a particular product: `GET api.company.com/productdetails/productId`. Developers don't have to learn a new language or tool.

- **Lightweight:** REST doesn't require the installation of complex software on the client devices. REST makes use of the networking infrastructure that is available on every device and machine nowadays. It uses HTTP to send messages back and forth.

- **De-facto standard:** Usage of REST is extremely widespread. Because it's already used by so many popular websites and countless apps, it is considered a *de-facto standard*. Making an investment in this technology is very safe. REST is not managed or ruled by a specific vendor.

- **Stateless:** REST is a *stateless* interface. This means that when, for example, a mobile app invokes business logic on a server through REST, the application returns an answer and no state remains on the server. Simply put, with REST there is no such thing as "Give me the next …". The server doesn't have to keep track of the state of every user. Especially in environments with potentially hundreds and thousands of users this would seriously hamper the system. This stateless interface is one of the technical reasons why REST is a lightweight interface and why it allows development of highly scalable systems.

- **Self-descriptive:** REST uses the JMS messaging standard. Beside the data itself, JMS messages contain descriptive meta data as well. JMS is less verbose than XML.

REST and Microservices – REST also plays an important role in a new approach for application development called *Microservices* that leads highly modular and maintainable systems. Wikipedia[3] indicates that "services in a microservices architecture are processes that communicate with each other over a network in order to fulfill a goal." Services must have a small granularity and the protocols must be lightweight. So, instead of developing one large monolithic system internally consisting of objects, modules, and components, the microservices are all independent little apps that together form a system. They communicate with each other through a lightweight protocol. In most cases, REST is uses as that protocol. The increasing popularity of microservices is an extra boost for REST's popularity. Companies such as Amazon, eBay, and Netflix have adopted a microservices architecture.

6 Externalizing Data and Business Logic with an API Gateway

Systems that have to be externalized commonly do not support REST but their own technical interfaces, such as home-made interfaces or interfaces based on international standards (e.g. SOAP/RPC, JDBC/SQL, IMS DLI, or CICS). These native interfaces must be converted to REST interfaces. Developing such interfaces and guaranteeing that they are secure, robust, and scalable is time-consuming and complex. A more popular and recommended approach is to deploy an *API gateway*. An API gateway is a façade that converts one API into another and hides the complex, native API of the underlying application, database, or system; see Figure 4.

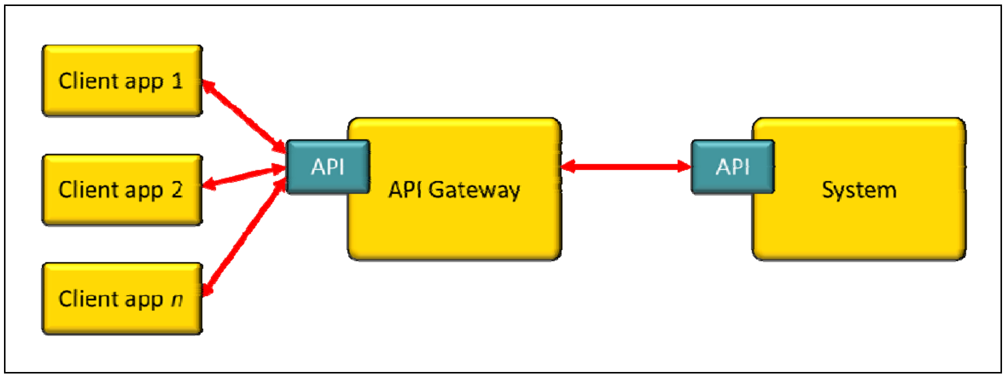

Figure 4 *An API gateway is a façade that converts one API into another.*

In general, an API gateway decouples the interface that client apps see from the underlying implementation. For example, an API gateway may expose data from enterprise systems, such as Billing, CRM, and Subscriptions, to any other application through a simple API. Most commercially available API gateways offer REST as interface, and can convert the native interfaces of all kinds of applications and databases to the REST interface. Additionally, an API gateway performs several additional tasks such as API limit throttling, logging, and security.

[3] Wikipedia, Microservices; see https://en.wikipedia.org/wiki/Microservices

7 Overview of IBM's Mainframe API Gateway: z/OS Connect Enterprise Edition

IBM's Mainframe API Gateway – Although several API gateway products are available, not that many exist that can access legacy systems on mainframes through a REST interface. IBM offers *IBM z/OS Connect Enterprise Edition* as the preferred API gateway for this purpose. It's designed specifically to develop REST interfaces on applications and databases running on mainframes. Because the tool has a deep understanding of and is fully integrated with typical mainframe systems, such as IMS and CICS, development of secure, robust, and fast REST interfaces is relatively easy. Moreover, the product comes with an easy-to-use development environment for REST interfaces on mainframe systems. Figure 5 presents a high-level overview of IBM z/OS Connect EE.

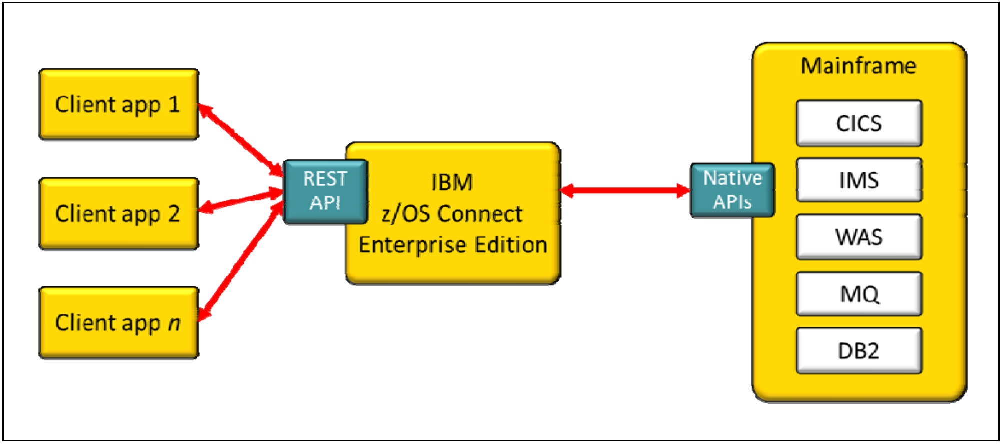

Figure 5 *IBM z/OS Connect EE is an API gateway designed specifically to develop REST interfaces on applications and databases running on mainframes.*

IBM z/OS Connect EE supports a long list of IBM subsystems, including CICS, IMS, WebSphere Application Server, WebSphere MQ, and Db2. With IBM z/OS Connect EE retrieval-like REST APIs can be developed to retrieve data, but also transactional APIs for doing updates, inserts, and deletes. Through integration with IBM z/OS System Management Facility requests from the Cloud, mobile, and Web environments can be tracked for auditing and governance purposes. Authorization rules can be specified with SAF Security to indicate which users are allowed to invoke which APIs.

In IBM z/OS Connect EE one REST call is limited to invoke one data source. When a client app needs to integrate data from, for example CICS and Db2, the integration can't be executed by IBM z/OS Connect EE. In other words, the product does not support data federation. This can be solved if one of the systems supports data federation functionality. For example, IBM InfoSphere Federation Server can be used to let Db2 access another data source. In this case, IBM z/OS Connect EE invoke a REST interface to Db2, Db2 asks Federation Server to do the integration with the other source.

Some client apps need to update two independent systems on the mainframe and the two updates must be processed as one atomic transaction. IBM z/OS Connect EE transactions doesn't support distributed transactions. A REST API invokes one system and not two or more.

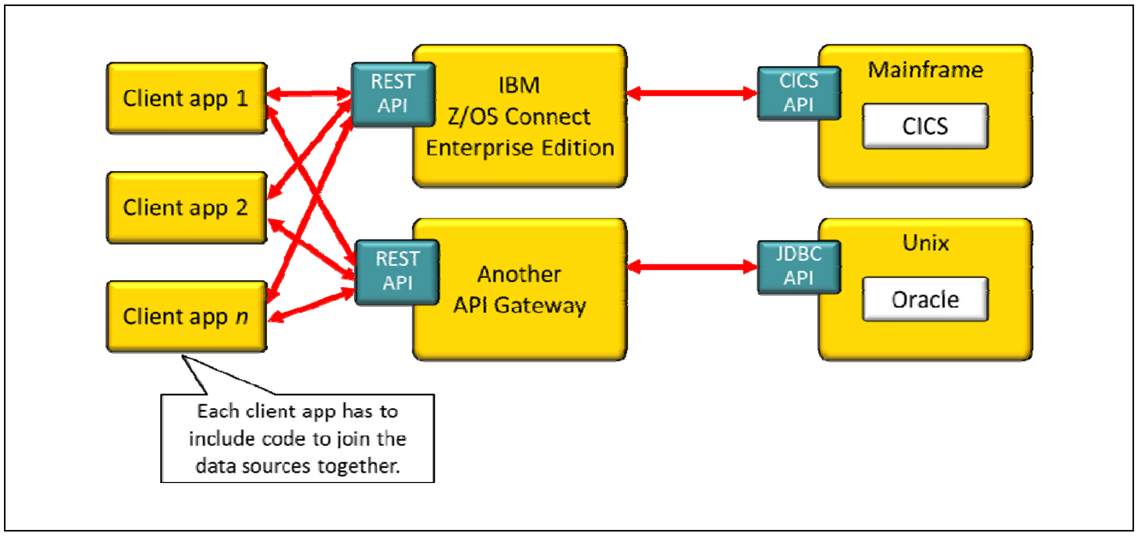

Figure 6 *Each client app must include code to join the results of the two REST APIs.*

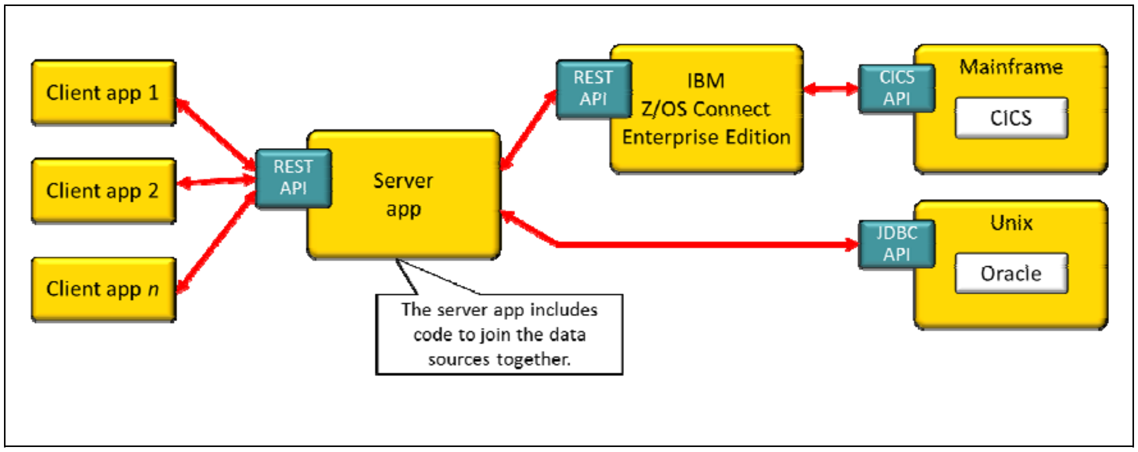

Figure 7 *The server app includes code to join the result of the REST API with that of the result coming from Oracle.*

8 Data Virtualization Servers as API Gateways

Introduction to Data Virtualization – Besides an API gateway another technology can be used to externalize older systems through a REST interface, namely *data virtualization*. This section briefly introduces data virtualization. For more information see the whitepapers "Making Mainframe Data Available for the Entire Organization[4]" and "Enriching Big Data Using Data Virtualization[5]."

Data virtualization servers allow the decoupling of data consumers from data stores; see Figure 8. A data virtualization server presents all the data stored in a heterogeneous set of data stores as a single logical database. Data consumers don't have to be aware of where and how the data is stored; all of the details of data storage are hidden for the data consumers. They don't have to know or care about whether the

[4] Rick F. van der Lans, "Making Mainframe Data Available for the Entire Organization", July 2014; see http://info.rocketsoftware.com/Making-Mainframe-Data-Available-for-the-Entire-Organization.html

[5] Rick F. van der Lans, "Enriching Big Data Using Data Virtualization", July 2014; see http://info.rocketsoftware.com/enriching-big-data-using-data-virtualization.html

data they're using is coming from a SQL database, an IMS database, or plain VSAM files. They don't have to be aware that data from multiple data stores have to be joined, nor do they have to know whether they are accessing a SQL database, a Hadoop cluster, a NoSQL database, a web service, or simply one or more flat files. The structure of the data stores is hidden as well; data consumers only see the data in the way that's convenient for them, and they only see data that is relevant to their task. This is all achieved by decoupling data consumers from data stores.

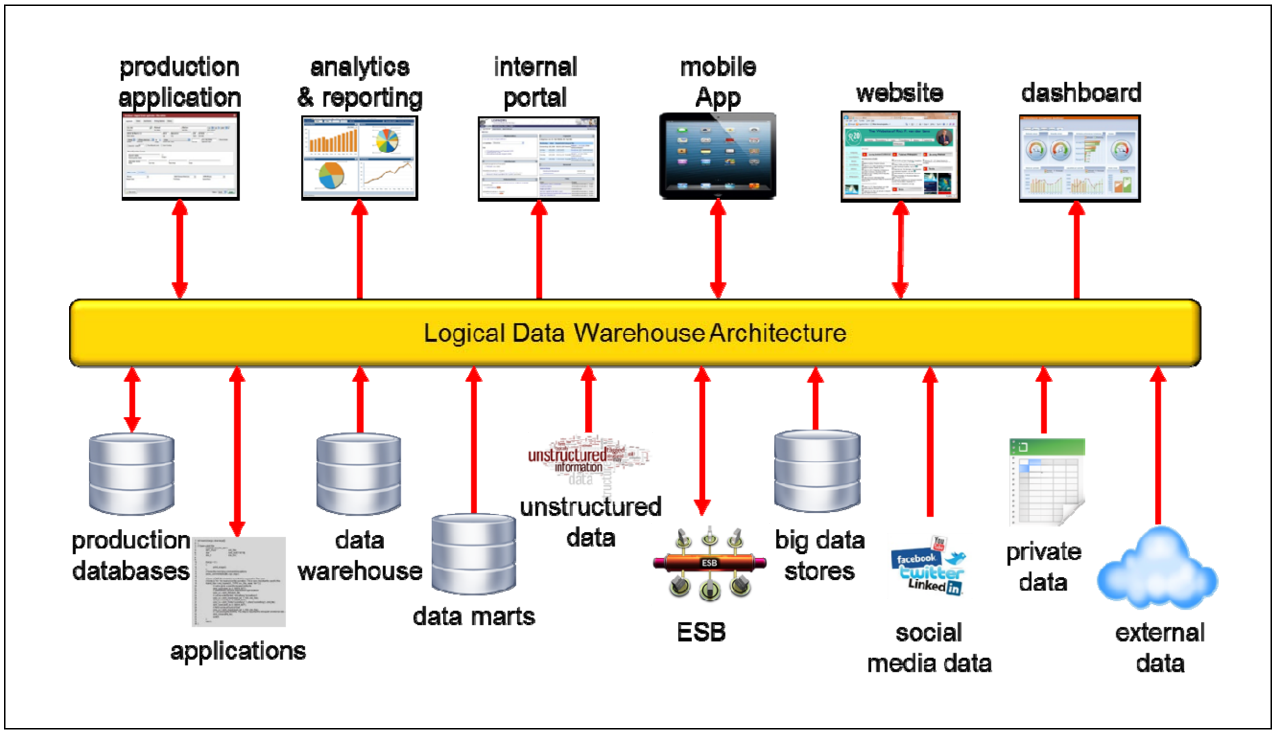

Figure 8 *With data virtualization, data consumers are decoupled from the data stores.*

The primary goal of this decoupling is to get a higher level of flexibility. For example, changes made to the data stores don't automatically mean that reports must be changed as well, and vice versa. Or, replacing one data store technology by another is easier when that data store is "hidden" behind the logical data warehouse architecture. With data virtualization, adopting big data is relatively easy, access to real-time data is less complex for data consumers, and dealing with cloud-based data becomes simple.

In a nutshell, the key advantages of using data virtualization are agility and simplicity.

Data Virtualization and REST – Besides being able to access all kinds of data stores, a data virtualization server also supports a wide range of APIs. For example, applications and tools can access the data through all kinds of APIs, including JDBC/SQL, SOAP, and REST.

Developing a REST interface with a data virtualization server is relatively straightforward. When a so-called *virtual table* has been defined, developers only have to indicate that a REST interface is required. Defining a REST interface involves selecting an input parameter and determining the structure of the resulting JSON data structure.

Data Virtualization and API Gateway – Data virtualization servers offer all the functionality API gateways do. But besides transforming a native API into another API, the former group offer more features for transforming, integrating, filtering, aggregating, and cleansing the data before it's passed on to the calling application.

Use Cases of Data Virtualization – Data virtualization technology supports a larger set of use cases then API gateways. Here are some of the popular use cases of data virtualization:

- Development of a *logical data warehouse* to offer business users a more agile business intelligence environment
- Development of a *data lake* for data scientists and other investigative business users
- Simplifying access to *big data* storage technologies, such as Hadoop and NoSQL
- *Democratizing data* to make the entire data asset of an organization available to a wide range of business users

The new use case for data virtualization is externalization of the internal systems; see Figure 9. Data virtualization is used for this use case because it simplifies development of REST interfaces on legacy systems. This new use case of data virtualization is sometimes referred to as *data as a service*.

9 Introduction to IBM Data Virtualization Manager for z/OS

Introduction to IBM Data Virtualization Manager for z/OS – *IBM Data Virtualization Server manager for z/OS* (IBM DVM) is designed specifically to integrate data sources on the mainframe data and to integrate mainframe data with off-mainframe data. IBM DVM can efficiently access all the well-known mainframe systems, such as CA IDMS, IBM CICS, Db2 z/OS, IMS TM & DB, Software AG Adabas and Natural. It can efficiently and smartly transform all these non-relational data structures into flat, relational data structures. In addition, typical mainframe file systems, such as sequential files and VSAM files, can be accessed. The product has taken advantage of IBM's data integration standard DRDA (Distributed Relational Database Architecture) by which it can access many SQL database servers, including IBM Db2 LUW and PureData for Analytics (formerly called Netezza), Oracle, and Microsoft SQL Server.

To streamline integration with big data, IBM DVM includes a capability for MongoDB's NoSQL database that transforms mainframe data into a binary form of JavaScript Object Notation (JSON) referred to as BSON. MongoDB uses JSON documents in order to store records, similar to how tables and rows store records in SQL database. MongoDB represents these JSON documents in a binary-encoded format called BSON[6]. BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages.

The Architecture of IBM DVM – To efficiently access the mainframe data sources, IBM DVM runs on the *zIIP processor*[7] (System Z Integrated Information Processor). These processors are designed to handle specialized workloads, such as large queries on Db2, Java, and Linux, and divert processing away from the mainframe's central processors. The importance is that by running on zIIP processors, IBM DVM doesn't

[6] MongoDB Architecture, see http://www.mongodb.com/json-and-bson
[7] See http://www.ibmsystemsmag.com/mainframe/administrator/performance/add_some_zIIP_to_your_mainframe/

interfere with mission critical enterprise applications running on the general purpose central processors and thus effectively reduces mainframe processing usage and reduces costs.

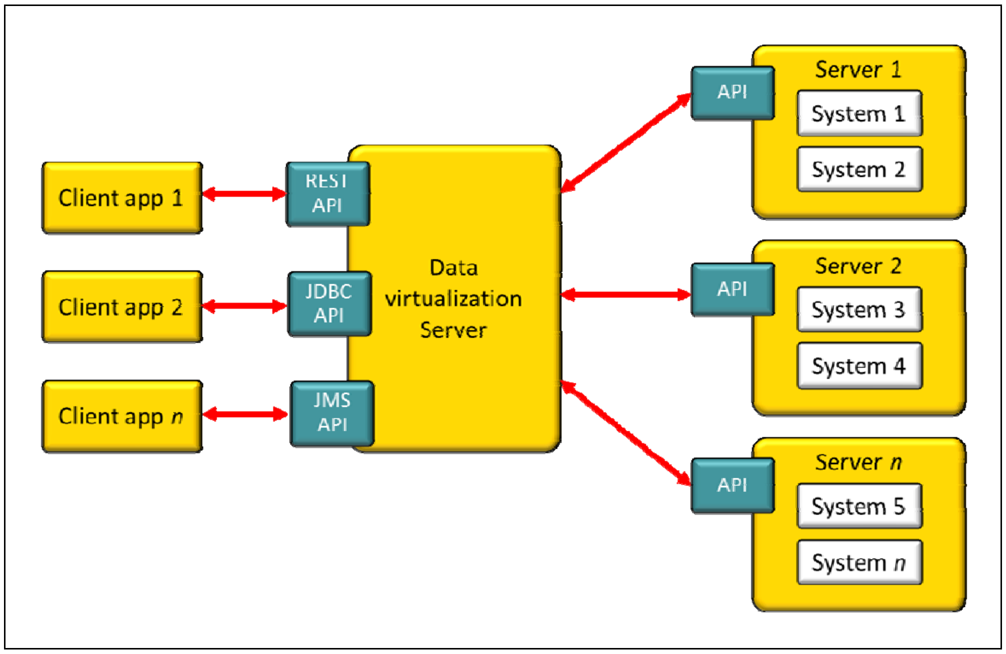

Figure 9 *Data Virtualization can be used to offer REST interfaces to client apps and act as a super API gateway.*

The strong points of IBM DVM are:

- Efficient database access to a wide range of popular mainframe data sources.
- High performance data architecture utilizing parallelism (I/O and MapReduce).
- Highly secure - integration of the data sources takes place on the mainframe itself and existing data security systems are not bypassed.

10 The Synergistic Effect of Combining IBM DVM and IBM z/OS Connect EE

Combining IBM DVM with IBM z/OS Connect EE – Many of the limitations of IBM z/OS Connect EE listed in Section 7 can be avoided by combining IBM DVM and IBM z/OS Connect EE to create a powerful API gateway for developing REST interfaces on mainframe and non-mainframe sources. This combined architecture is presented in Figure 10. The two products are seamlessly integrated. Developers familiar with working with IBM z/OS Connect EE don't even have to see that they're using IBM DVM.

Benefits of Combining IBM DVM with IBM z/OS Connect EE – In this combined architecture, IBM z/OS Connect EE brings to the table an easy to use environment (Studio) for developing REST interfaces, efficient and secure access of mainframe systems, robust access, and a scalable solution. IBM DVM extends this with the following features:

Access to Mainframe Sources: Where IBM z/OS Connect provides a RESTful interface to mainframe *applications*, IBM DVM allows for the development of REST interfaces that access mainframe *data* sources residing on mainframes, including Adabas, IMS DB, IDMS, System Management Facility (SMF), VSAM and physical sequential files.

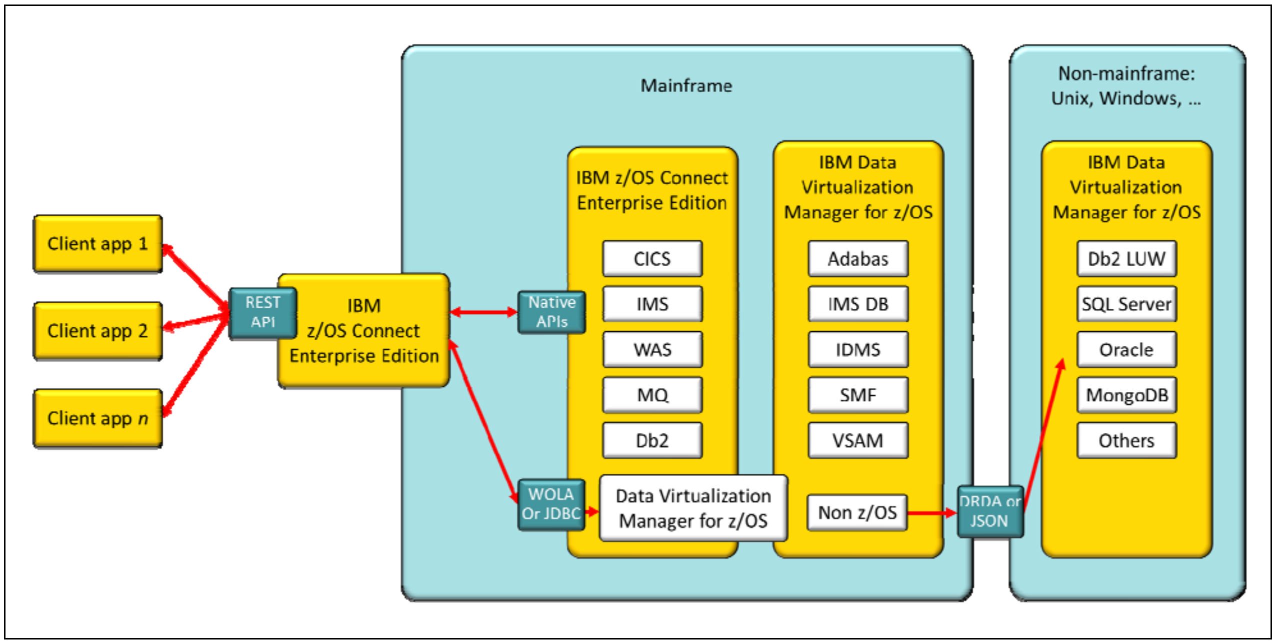
Mainframe
Non-mainframe: Unix, Windows, ...
Client app 1
Client app 2
Client app n
REST API
IBM z/OS Connect Enterprise Edition
Native APIs
IBM z/OS Connect Enterprise Edition
CICS
IMS
WAS
MQ
Db2
WOLA Or JDBC
Data Virtualization Manager for z/OS
IBM Data Virtualization Manager for z/OS
Adabas
IMS DB
IDMS
SMF
VSAM
Non z/OS
DRDA or JSON
IBM Data Virtualization Manager for z/OS
Db2 LUW
SQL Server
Oracle
MongoDB
Others

Figure 10 *IBM DVM and IBM z/OS Connect EE can work together as one integrated API gateway that offers access to a large set of data sources, including those running on and outside IBM mainframes.*

Access to Non-Mainframe Sources: IBM DVM supports access to data sources not residing on mainframes, including SQL database servers, NoSQL databases such as MongoDB, and big data technologies such as Spark and Hadoop. This is important because many large organizations, besides having a large installed base of applications on the mainframe, use more and more applications not residing on mainframes.

Multi-Source REST Calls: REST interfaces can be developed with IBM DVM that combine multiple data sources through the product's "standard" features such as the federated join. Two tables can be joined together inside one virtual table definition. This joined virtual table can then be used as the source for a REST API. For example, in IBM DVM a virtual table can be defined that joins data from IMS on a mainframe with Oracle on a Unix machine. Next, a REST interface can be developed for this virtual table. Developers of the customer-facing app accessing this REST interface won't see that data comes from two systems.

Multi-Source Transactions: IBM DVM supports transactions that span multiple sources. For example, one transaction doing inserts in Db2 on the mainframe and deletes on a Microsoft SQL Server running on a Windows machine can be treated as one atomic transaction. A REST interface can be developed for such a transaction. Every client application invoking this API feels as if it's working with one source system.

Transformation Features: As befits a data virtualization server, IBM DVM offers many features to transform data from the source system before it's send back to the client application. Data can be aggregated, filtered, joined, concatenated, unioned, etc. The full power offered by the SQL query language can be deployed to turn the data into the right structure for the API. No programming in a low-language is required.

Minimal Interference: As indicated in Section 9, because IBM DVM runs on zIIP processor there is minimal interference on the workload of operational systems.

11 Closing Remarks

Digital transformation is on the agenda of countless organizations; commercial and non-commercial. For some it will be a long journey. For most it entails development of customer-facing apps and externalizing legacy systems through APIs allowing third party apps to retrieve data and business logic.

For those organizations with a massive investment in databases and systems running on mainframes, implementing REST interfaces is a technological challenge. In general, API gateways simplify the development of REST APIs. IBM z/OS Connect EE is an API gateway designed specifically to develop fast, robust, and secure REST APIs for subsystems such as CICS, IMS, WebSphere Application Server, WebSphere MQ, and DB2. The easy-to-use development environment allows for fast development, but does not provide access to non-IBM sources, no access to non-mainframe sources, single-source REST calls, and single-source transactions.

Extending IBM z/OS Connect EE with IBM DVM overcomes these limitations. IBM DVM integrates seamlessly with IBM z/OS Connect EE, providing developers with a single solution to build REST APIs for a larger set of sources, including non-IBM sources on the IBM mainframe and sources running on other platforms than the mainframe. Additionally, IBM DVM can integrate data from sources allowing APIs to be developed to show data from multiple systems in an integrated fashion.

Besides allowing the development of REST APIs, because it's a full-blown data virtualization server, IBM DVM can also be used for other use cases, such as the development of logical data warehouses and data lake for data scientists, simplifying access to big data storage technologies, and for democratizing data. This will be a reuse of the investment made in IBM DVM and will help organizations with other aspects of their digital transformation.

About the Author

Rick F. van der Lans is an independent analyst, consultant, author, and lecturer specializing in data warehousing, business intelligence, big data, database technology, and data virtualization. He works for R20/Consultancy (www.r20.nl), which he founded in 1987.

Rick is chairman of the annual European Business Intelligence and Analytics Conference (organized annually in London). He writes for Techtarget.com[8], B-eye-Network.com[9] and other websites. He introduced the business intelligence architecture called the *Data Delivery Platform* in 2009 in a number of articles[10] all published at B-eye-Network.com. The Data Delivery Platform is an architecture very similar to the logical data warehouse.

He has written several books. His latest book is *Data Virtualization for Business Intelligence Systems*[11]. Published in 1987, his popular *Introduction to SQL*[12] was the first English book on the market devoted entirely to SQL. After almost thirty years, this book is still being sold, and has been translated into several languages, including Chinese, German, Italian, and Dutch.

For more information please visit www.r20.nl, or send an email to rick@r20.nl. You can also get in touch with him via LinkedIn and Twitter (@Rick_vanderlans).

[8] See http://www.techtarget.com/contributor/Rick-Van-Der-Lans
[9] See http://www.b-eye-network.com/channels/5087/articles/
[10] See http://www.b-eye-network.com/channels/5087/view/12495
[11] R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012.
[12] R.F. van der Lans, *Introduction to SQL; Mastering the Relational Database Language*, fourth edition, Addison-Wesley, 2007.