

# IBM z Systems Integration Guide for the Hybrid Cloud and API Economy

Richard Gamblin

Nigel Williams







International Technical Support Organization

**IBM z Systems Integration Guide for the Hybrid Cloud  
and API Economy**

February 2017

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

**Second Edition (February 2017)**

This edition applies to the current version of products at the time of publication.

© Copyright International Business Machines Corporation 2015, 2017. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
Authors .....	vii
Now you can become a published author, too! .....	viii
Comments welcome .....	ix
Stay connected to IBM Redbooks .....	ix
<b>Summary of changes</b> .....	xi
February 2017, Second Edition .....	xi
<b>Chapter 1. Introduction</b> .....	1
1.1 Evolution of integration with core systems .....	3
1.2 Business imperatives .....	4
1.3 Multi-speed IT .....	4
1.4 Mainframe connectivity landscape .....	5
1.5 Hybrid cloud connectivity challenges .....	5
<b>Chapter 2. Architecture options for service and API enablement</b> .....	7
2.1 APIs and API management .....	8
2.1.1 From services to APIs .....	10
2.1.2 API management .....	10
2.1.3 Advantages of APIs and API management .....	11
2.2 REST and JSON .....	11
2.2.1 REST .....	11
2.2.2 JSON .....	11
2.2.3 Advantages of REST and JSON .....	12
2.3 SOAP web services .....	12
2.3.1 Advantages of SOAP web services .....	13
2.4 Messaging .....	14
2.4.1 Advantages of messaging .....	14
<b>Chapter 3. Hybrid integration architecture considerations</b> .....	15
3.1 Hybrid integration architecture .....	16
3.2 System of Record .....	17
3.3 System of Engagement .....	17
3.4 Access layer .....	18
3.5 Integration layer .....	19
3.6 API gateway .....	20
3.7 Security gateway .....	21
<b>Chapter 4. IBM integration solutions</b> .....	23
4.1 IBM z/OS Connect Enterprise Edition .....	24
4.1.1 API design and workflow .....	26
4.1.2 z/OS Connect EE run time .....	28
4.1.3 When to use z/OS Connect EE .....	28
4.2 CICS .....	28
4.2.1 z/OS Connect EE with CICS .....	29

4.2.2	Java applications in a CICS Liberty JVM server . . . . .	31
4.2.3	CICS SOAP web services . . . . .	32
4.2.4	CICS JSON web services . . . . .	33
4.2.5	CICS Transaction Gateway JSON web services . . . . .	34
4.3	IMS . . . . .	35
4.3.1	z/OS Connect EE with IMS . . . . .	35
4.3.2	IMS Enterprise Suite SOAP Gateway . . . . .	37
4.4	DB2 for z/OS . . . . .	38
4.4.1	DB2 REST services . . . . .	38
4.4.2	z/OS Connect EE with DB2 REST services . . . . .	39
4.5	IBM MQ . . . . .	40
4.5.1	z/OS Connect EE with IBM MQ . . . . .	40
4.6	IBM Integration Bus . . . . .	43
4.6.1	When to use IBM Integration Bus . . . . .	44
4.7	IBM API Connect . . . . .	44
4.7.1	How IBM API Connect works with z/OS Connect EE . . . . .	45
4.8	IBM DataPower Gateway . . . . .	48
4.8.1	When to use an IBM DataPower Gateway . . . . .	49
<b>Chapter 5. Real-world scenarios . . . . .</b>		<b>51</b>
5.1	Web service enablement . . . . .	52
5.1.1	Introduction . . . . .	52
5.1.2	Key decision factors . . . . .	52
5.1.3	Solution architecture . . . . .	53
5.1.4	Next step . . . . .	54
5.2	Reusing web services . . . . .	54
5.2.1	Introduction . . . . .	54
5.2.2	Key decision factors . . . . .	55
5.2.3	Solution architecture . . . . .	56
5.2.4	Next step . . . . .	57
5.3	REST service enablement . . . . .	57
5.3.1	Introduction . . . . .	57
5.3.2	Key decision factors . . . . .	58
5.3.3	Solution architecture . . . . .	59
5.3.4	Next step . . . . .	60
5.4	API management . . . . .	60
5.4.1	Introduction . . . . .	60
5.4.2	Key decision factors . . . . .	61
5.4.3	Solution architecture . . . . .	62
5.4.4	Next step . . . . .	63
<b>Chapter 6. Summary . . . . .</b>		<b>65</b>
6.1	Integration architectures . . . . .	66
6.2	Integration solutions . . . . .	67
<b>Related publications . . . . .</b>		<b>69</b>
	IBM Redbooks . . . . .	69
	Online resources . . . . .	69
	Help from IBM . . . . .	70

# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Bluemix®	IBM API Connect™	Redpaper™
CICS®	IBM Watson™	Redbooks (logo)  ®
CICS Explorer®	IBM z Systems®	WebSphere®
CICSplex®	IMS™	z Systems®
DataPower®	RACF®	z/OS®
DB2®	Rational®	
IBM®	Redbooks®	

The following terms are trademarks of other companies:

Connect:Direct, and N logo are trademarks or registered trademarks of IBM International Group B.V., an IBM Company.

Evolution, and Inc. device are trademarks or registered trademarks of Kenexa, an IBM Company.

LoopBack, and the StrongLoop logo are trademarks of StrongLoop, Inc., an IBM Company.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.



# Preface

Today, organizations are responding to market demands and regulatory requirements faster than ever by extending their applications and data to new digital applications. This drive to deliver new functions at speed paved the way for a huge growth in cloud applications, such as those applications hosted in IBM® Bluemix®. One of the most common ways to integrate cloud applications with enterprise application logic and business data is the use of application programming interfaces (APIs).

By extending enterprise applications to form a hybrid cloud environment, you can capitalize on investments without the need to develop new solutions to support cloud services. Because over 90% of new client-facing apps require mainframes, IBM z Systems have an important role in enabling existing and new mainframe applications to be easily used by cloud applications; for example, through REST APIs.<sup>1</sup> These same APIs can be used through other channels and applications on-premises and in the cloud.

Many technologies and solutions can be used to enable hybrid cloud integration with the mainframe, including web APIs, services, connectors, messaging, and so on. The primary goal of this IBM Redpaper™ publication is to help IT architects choose between the many integration architectures and solutions and make the right choice that is based on the specific requirements of the project.

This Redpaper publication outlines some of the business imperatives and challenges. Then, it reviews the main architecture options and the key considerations for planning API-enablement of the mainframe and provides guidance for when to use specific solutions. Finally, it documents several API integration scenarios to show how these technologies are used in real-world scenarios.

## Authors

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

**Richard Gamblin** is an IBM Technical Staff Member, Redbooks® Thought Leader, and Global Leader for Hybrid Cloud & API Economy for the WW z Champions Team. As the European Technical Leader for Digital Transformation, Richard works with clients to develop solutions and capabilities with API, Mobile, Blockchain, and IBM z Systems® technologies. He has worked in several technical roles during his time in IBM, from an Integration and Connectivity Specialist to an IBM WebSphere® Architect. Before joining IBM, Richard was a researcher at the University of Leeds, from where he obtained a doctorate in the field of Bioinformatics.

**Nigel Williams** is an IT Specialist working at the IBM Client Center in Montpellier, France. Nigel specializes in IBM CICS® integration, API enablement, and mainframe security topics. He helps clients to design and test mainframe integration solutions. He is the author of many papers and IBM Redbooks publications. Previously, Nigel worked as a Developer and Tester at the IBM Hursley Software Lab.

---

<sup>1</sup> <http://www.ibmssystemsmag.com/mainframe/trends/IBM-Announcements/Analytics-Accelerator-BigInsights/>

Thanks to the following people for their contribution to this project:

Kim Clark  
Stephen Cocks  
Paul Cooper  
Rob Jones  
Matthew Leming  
Andy Lyell  
Alexander Melkonian  
Anthony Papageorgiou  
Ian Shore  
Ben Thompson  
Phil Wakelin  
**IBM UK**

Kyle Charlet  
Hernan Cunico  
Asit Dan  
James Pickel  
Bill Seubert  
**IBM US**

Aymeric Affouard  
Carl Farkas  
Eric Phan  
**IBM France**

Frank van der Wal  
**IBM Netherlands**

Charlotte Gamblin of Woodmancote School

Thanks to the following authors of the first edition, *IBM z Systems Integration Guide for the Mobile and API Economy*, published December 2015:

- ▶ Richard Gamblin
- ▶ Rob Jones
- ▶ Nigel Williams

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



# Summary of changes

This section describes the technical changes made in this edition of the paper and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes  
for IBM z Systems Integration Guide for the Hybrid Cloud and API Economy  
as created or updated on February 24, 2017.

## February 2017, Second Edition

This revision includes the following new and changed information.

### **New information**

- ▶ New Hybrid Integration Architecture
- ▶ IBM DB2® REST services
- ▶ IBM z/OS® Connect EE support for DB2
- ▶ z/OS Connect EE support for MQ
- ▶ API Connect

### **Changed information**

Support for running z/OS Connect EE inside CICS





# Introduction

Digital channels are the primary way to start consumer and business engagements by providing accessibility to a global audience and a cost-effective means to conduct business. In recent years, organizations augmented their services with third-party functions, such as Google Maps, to combine geolocation data and mapping tools that help a mobile user find the closest store or branch. More sophisticated services are emerging in these apps that allow consumers to transfer money to friends and colleagues based on their social media accounts. Indeed, the ability to enhance applications with cognitive capabilities, such as IBM Watson™, can vastly improve employee efficiency and customer experience.

To create powerful and useful mobile and web apps, the use of application programming interfaces (APIs) becomes essential. APIs can help quickly address new market opportunities by unlocking enterprise application logic and business data and combining with open (or subscription-based) APIs to bring innovative new products to market faster than ever. A particular feature of web APIs that accelerated its adoption is the ability for developers to self-discover APIs and generate application snippets that consume the APIs.

A key platform for these emerging digital services is the cloud. Many of the emerging digital applications and microservices that provide these innovative new capabilities are built to use the agility of cloud platforms, such as IBM Bluemix. However, for the applications and services to function effectively, they must harness the application logic and business data that is held within enterprise systems, such as IBM z Systems®. The ability to use z Systems as-a-Service by using APIs provides a powerful way to fuel the emerging digital applications by abstracting away the underlying IBM z/OS implementations.

Access to z Systems applications and data has long been available through multiple paths, with more being introduced to specifically satisfy hybrid cloud workloads by using APIs. The real challenge is that enterprise architect teams have so many different options that it can be difficult to understand the factors that are involved in assessing and selecting one over another. This IBM Redpaper publication describes the options and factors that need to be considered to make sound architectural decisions to integrate hybrid cloud services with z Systems applications and data.

This chapter includes the following topics:

- ▶ 1.1, “Evolution of integration with core systems” on page 3
- ▶ 1.2, “Business imperatives” on page 4
- ▶ 1.3, “Multi-speed IT” on page 4
- ▶ 1.4, “Mainframe connectivity landscape” on page 5
- ▶ 1.5, “Hybrid cloud connectivity challenges” on page 5



## 1.1 Evolution of integration with core systems

Originally, users accessed mainframe applications through a connected 3270 terminal. Although this interface is still used today, access to mainframe applications changed significantly over the past 50 years, as shown in Figure 1-1.

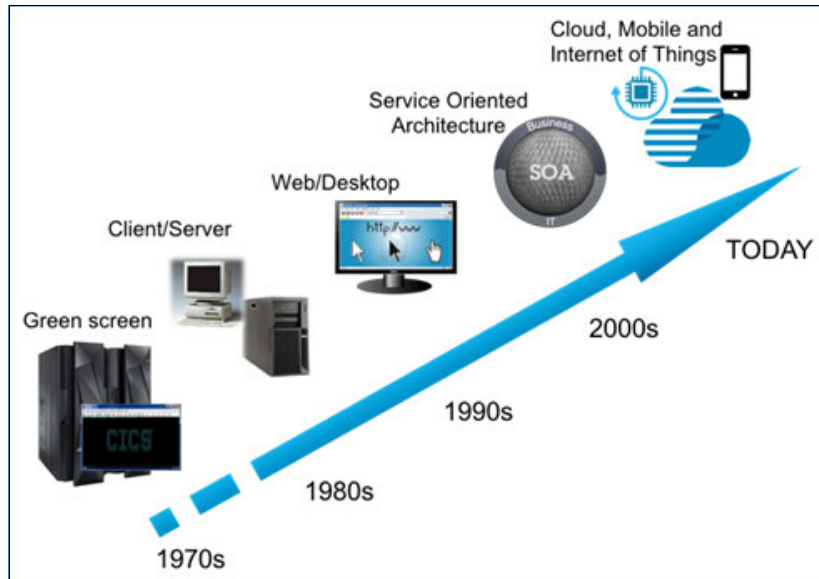


Figure 1-1 Evolution towards self-service

At each stage of the evolution, distinctive changes occurred to the types of users who access the systems and the user interface (UI). How the UI changed is listed in Table 1-1.

Table 1-1 Characteristics of evolutionary stages of access to mainframe systems

Stage	Characteristics
Green screen	Limited user community with a limited and controlled UI.
Client/server	User community remains limited; however, UI is increasingly richer.
Web	Wider user community (anyone with a browser), UI-controlled by web page (first z Systems web pages were similar to layouts that were used in green screens).
Service-oriented architecture (SOA)/Web 2.0	Much wider user community, with SOAs reusing services and making them available through decoupled integration methods. The UI is controlled by a service requester application or (in the case of Web 2.0) the user.
Cloud/Mobile or Internet of Things (IoT)	User community is anyone (often beyond the traditional enterprise boundaries) that is creating cloud applications, frequently with users interacting through mobile or IoT devices and commonly reusing services that are developed for SOA.

As different kinds of users access these systems, the following new terms emerged to distinguish between the roles of systems:

- **System of Record**

Systems of Record (SoR) are the core systems upon which organizations rely to house their mission-critical data and applications. A particular emphasis is placed on transactional integrity and data consistency. These systems are typified by IBM z Systems mainframes. SoRs tend to be hosted on-premises and often are accessed through a System of Engagement.

- **System of Engagement**

Systems of Engagement (SoE) are the systems through which people interact to access the applications and data in the SoR. Although many of these SoEs are hosted onsite, they are increasingly hosted in public or private cloud environments. SoEs, such as email clients or web applications, are increasingly accessed by mobile and IoT devices.

## 1.2 Business imperatives

Organizations are undergoing significant change to respond to market challenges. Some of these changes are in response to regulatory requirements and others are in response to the emerging requirements of consumers of their products or services. Over the last 5 - 10 years, a significant shift occurred in the marketplace regarding how consumers and employees expect to interact with businesses and government institutions. Two recent Institute of Business Value studies identified this shift away from an Enterprise-centric market to an Individual-centric marketplace. The confluence of social, mobile, and cloud technologies, along with insight that is drawn from expanding pools of data, led to this need for digital reinvention of enterprise IT to meet the demands of the Individual-centric economy.

A key part of this approach is to ensure that IT systems are optimized and aligned to support this unprecedented speed of change. Consider that green-screen applications were developed over the course of 40 years, that client/server did the same over 25 years, or that the Internet matured in 15 years. Contrast those development times with the mobile and digital age, in which consumer expectation has rocketed in a mere 5 years. Organizations must respond more rapidly than ever to meet this demand. One approach is to create a multi-speed enterprise IT.

## 1.3 Multi-speed IT

On average, the lifecycle for a mobile app is somewhere in the range of 3 - 6 weeks, after which it is typically updated in some way. In contrast, new versions of cloud-hosted applications might be delivered every 6 - 12 weeks. Major core banking applications might see updates every 3 - 6 months, all of which work to different time scales.

Core applications can handle updates on a much short lifecycle, even in days and weeks; however, these updates typically are not done because the core application forms the foundation upon which the entire organization's applications, presentation layers, and integration components are based. Instead, we see the emergence of a multi-modal, or multi-speed IT, which combines the merits of a fast-paced digital landscape with the stability and resilience of a more moderately paced enterprise landscape.

The critical part of this multi-speed IT model is its ability for the digital and enterprise systems to work together harmoniously. For example, suppose that a company identified a new service that can offer a competitive advantage in the market. If that service is prematurely introduced in the company's mobile app but critically before it is available in the enterprise application, it might effectively wipe out the advantage as competitors put in place their own alternatives in that interim period. Similarly, if the capability is available in the enterprise system but yet to be reflected in the mobile app, the return on that investment cannot be realized. In short, the two systems (the digital and enterprise) must work in tandem. It is the role of the integration layer to provide that seamless interlock.

## 1.4 Mainframe connectivity landscape

Various options are available for connecting and integrating mainframe systems across the enterprise, all of which are in use in production today. Such direct connectivity options include the following examples:

- ▶ Message-based connectivity, such as IBM MQ.
- ▶ File-based connectivity, such as Sterling Connect:Direct®.
- ▶ Standards-based connectivity (for example, web services).
- ▶ Custom connectivity options, such as IBM CICS Transaction Gateway or IBM IMS™ Connect.

In addition, the following commonly used integration technologies are available for routing, aggregating, and disseminating information to and from mainframe systems:

- ▶ Integration service technologies, such as IBM Integration Bus
- ▶ Appliance-based technologies, such as IBM DataPower® Gateway

Although these connectivity and integration options have been used for many years to support client/server, web, and SOA integration architectures, new hybrid cloud applications are emerging that combine public and privately hosted infrastructure. These hybrid applications often include business and application logic that is on mainframes. In many cases, integration solutions can be extended to support these use cases; however, there are unique aspects of hybrid cloud architectures that have an effect on the choice of integration technology.

## 1.5 Hybrid cloud connectivity challenges

Applications and data are increasingly hosted on cloud infrastructure. A recent IBM Institute of Business Value study revealed 75% of interviewed organizations adopted or implemented cloud solutions, with 90% expecting to do so within three years. To fully tap the potential of this paradigm, cloud applications, whether hosted on-premises (private) or off-premises (public), must use the rich palette of business logic and data that is held in the enterprise systems. This configuration requires a hybrid cloud model in which cloud applications can securely and efficiently be integrated with enterprise applications.

This hybrid cloud model places the following new demands on enterprise IT systems and the ways in which the two are integrated:

- ▶ Service discovery: Increasingly, cloud application developers work externally with an organization and they require ways to discover and use enterprise services.

- ▶ **Security:** For public cloud applications, there is a multi-faceted requirement for security, ranging from transport-level security, to throttling of requests, to secure application management.
- ▶ **Analytics:** With self-service mechanisms being an integral part of cloud IT, part of the integration challenge is to have an insight into who is using which versions of each service, at what volumes, and how frequently.
- ▶ **Choice with consistency:** Much of the simplicity that is associated with cloud applications is their innate ability to communicate by using simple, self-describing payloads. The ability to surface traditional applications and data using industry adopted standards is key.

The rest of this IBM Redpaper publication examines the architectural approaches and technologies that are available that address these hybrid cloud integration challenges.



## Architecture options for service and API enablement

This chapter describes the main integration architectures that can be used for mobile integration with z Systems applications.

This chapter includes the following topics:

- ▶ 2.1, “APIs and API management” on page 8
- ▶ 2.2, “REST and JSON” on page 11
- ▶ 2.3, “SOAP web services” on page 12
- ▶ 2.4, “Messaging” on page 14

## 2.1 APIs and API management

Computer programmers are familiar with the concept of an application programming interface (API). Historically, an API refers to the published details for a defined set of capabilities that an application programmer can build upon. The details of such an API might typically be published in a technical manual and distributed as a binary runtime library with an associated license.

The types of APIs that are described in this IBM Redpaper publication follow the same core principles. However, they operate in a connected world in which APIs can be easily discovered by any developer with the appropriate access. They also are self-describing to the point where application development tools can generate code to use the API, which accelerates adoption and promotes preferred practices.

Although these APIs still represent common services for application programmers, the scope in which they can operate is dramatically different. APIs today might be used across the breadth of global organizations, between companies, or by private individuals. They might be combined with other APIs from entirely unrelated providers to form innovative value propositions. With an API, developers can use the functions of computer programs in other applications.

Over the years, APIs evolved based on advances in technology (such as network speed, security, and dynamic integration). As business matured, these functions can be thought of as discreet, consumable entities.

The ways that applications intercommunicate by APIs has changed over the years. In particular, the advent of service-oriented architecture (SOA) provided an architectural model to manage consumer and provider relationships in a dynamic environment. This model paved the way for producing and making available APIs with better business enablement capabilities, including request access, entitlement, identification, authorization, management, monitoring, and analytics. Figure 2-1 shows the relationship between an API architecture and an SOA, which bridges the Systems of Engagement ecosystem with the Systems of Record ecosystem.

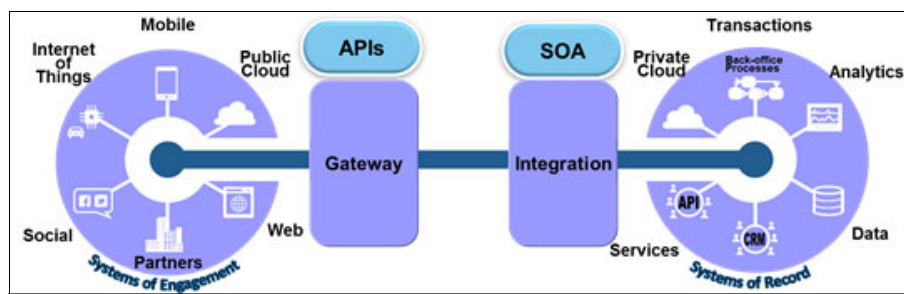


Figure 2-1 Systems of Engagement consuming APIs

Cloud application developers today can develop high-value applications by combining available business services or APIs that are often made available by various API providers and discovered independently by application developers.

Each service or API provider is unlikely to envision all of the ways in which an API might be used. However, if a mutual benefit exists, the API has a fair chance of success. Cost-effective APIs that provide rapid value to application developers and that also build a reputation for reliability can soon become the definitive method of providing a specific capability within the mobile application development community (for example, location services that use Google Maps APIs).

APIs are needed for the following reasons:

- ▶ Surfacing of services for more flexibility:
  - Supporting easier access to customers and partners
  - Extending the reach to more channels and devices
- ▶ Marketing and monetizing enterprise data and application logic

Today, progressive companies are making APIs available to allow others to use their business functions for a profit. For example, Google, Facebook, Twitter, and other companies provide APIs that are used across various applications. All of these companies built a functional platform of business capabilities and extended their business models by making APIs available so that developers can use their functionality. Google Maps is a great example. Many developers write mashups on top of Google Maps for various reasons; for example, retail store locator, traffic reports, and road conditions.

One important business reason to enable APIs in your business is to monetize your business capabilities. For example, if you are a credit reporting agency and you produce an API that establishes credit scores and facts regarding a consumer's credit history, many banks, loan companies, insurance companies, and solicitation companies can use your API for money. It also provides them with the ability to perform the API functions and avoid the need to develop and maintain their own APIs.

In addition, they can easily disconnect your API if a better one becomes available. This issue is the differentiator for APIs in an API economy: The ability to quickly subscribe to or unsubscribe to business functionality. It makes business more agile by driving a healthy competition for business function.

The term *API economy* refers to the opportunities that are associated with productizing the exposure of your business functions as APIs. Consider that your API is a consumable product and you must market and position your product correctly for maximum profit. Therefore, an API economy deals with the extra channel opportunities that are associated with the proper exposure of your usable business functions.

For potential API consumers, business resources that are on IBM z Systems with products, such as CICS, IMS, IBM DB2, and IBM WebSphere Application Server, represent a rich source of capability if only their existence can be independently discovered by application programmers and integration options are aligned with other API providers.

Potential users of enterprise APIs might be internal (inter-department, cross-function, or employee), partner (affiliated, authorized Business Partners), or public (free or by registration). In all cases, access to enterprise APIs face a common set of challenges in terms of consumability, security, auditing, measurement, billing, and lifecycle. API management aims to provide a unified approach for addressing these challenges.

## 2.1.1 From services to APIs

APIs are the externalized aspect of services. As such, they are not to be viewed as an alternative to SOA, but rather a part of a well-designed, service-oriented enterprise. However, APIs are a specific genre of services with a lifecycle that is focused on “external” usage. This externalization of enterprise services drives a focus on simplicity, security, and compatibility with standards-based external systems.

Enterprise-scale businesses most likely feature many defined web services (see 2.3, “SOAP web services” on page 12). These mission-critical transactional services and business processes often provide a rich source of content for new APIs. A collection of individual services that provide operations upon a common resource might now be represented as a collective unit (an API). Such an API can be discovered, documented, invoked, and maintained as a single entity.

Developing for internal enterprise services and external APIs enables the use of distinct content pools in which completeness of content and operations upon a specific business resource might vary according to the user. For example, an internal user in a Human Resources department might have full access to an employee record, whereas an external employee directory might redact sensitive personal information from an employee record, such as home address or salary, while ultimately accessing the same system of record asset.

An API is composed of operations, which are offered in one of the following styles:

- ▶ A REpresentational State Transfer (REST) API is structured according to the principles of REST and typically uses the JSON data format (for more information, see 2.2, “REST and JSON” on page 11).
- ▶ A SOAP API is a web service that is made available as an API.

Although a collection of independent services can be brought together under the auspices of an “API” facade, such an API might not be naturally “RESTful” if it does not intuitively reflect the create, retrieve, update, and delete operations through the set of HTTP methods.

## 2.1.2 API management

API management brings a multitude of operational capabilities and insight to bear on APIs and services, including discovery through an API marketplace, access controls, lifecycle operations, rate control (or throttling), metering, auditing, and analytics.

The combination of RESTful APIs and API management heralds a significant evolution beyond the initial service enablement patterns of SOA, and the possibility to use JSON-encoded data promises to make z Systems business assets easily usable for the rapidly expanding mobile and cloud-based application development community.

A key success metric for API-enablement for z Systems assets is discovery and ease of consumption. An API enablement technology must make z Systems APIs discoverable and easily usable on the terms of the consumer. Today, the OpenAPI Initiative, which is a Linux Foundation sponsored Open Source Initiative that is backed by several organizations (including IBM), defines a standard, language-agnostic interface for REST APIs. The implementation of this initiative, an OpenAPI (formerly Swagger)<sup>1</sup> definition document, can provide a usable unit for API consumers that provides everything they need to understand about what the API provides and how to use it.

---

<sup>1</sup> Swagger is an open specification for describing, producing, using, and visualizing RESTful web APIs. For more information, see this website:  
<http://swagger.io/specification>



Discovery of a self-describing API through a marketplace with the social capabilities, such as number of users, ratings, and lifecycle updates allow great APIs to drive rapid adoption. Direct feedback can drive the evolution and requirements gathering process or quickly identify unpopular modifications, all in one place.

### 2.1.3 Advantages of APIs and API management

The following major advantages result from implementing an API management solution:

- ▶ Extends internal enterprise services to a system of developers and new markets.
- ▶ Controls access to enterprise services.
- ▶ Provides insight into who is accessing enterprise services.

## 2.2 REST and JSON

Application developers today typically expect APIs to “talk” HTTP, to naturally use HTTP methods to represent the wanted operation, to exchange data represented in JSON, and to return resource references as fully formed URIs that are ready to flow on a subsequent request. REST and JSON are assumed to be universally available for applications that are designed for modern mobile devices, such as smartphones and tablets.

### 2.2.1 REST

REST is a defined set of architectural principles by which you can design web services that focus on resources. The REST architectural pattern uses the technologies and protocols of the World Wide Web to describe how data objects can be defined and modified.

In contrast to a request-response model, such as SOAP that focuses on procedures that are made available by the system, REST is modeled around the resources in the system.

In simple terms, REST prescribes a basic mapping from HTTP methods POST, GET, PUT, and DELETE, to the logical operations create, retrieve, update, delete. Each resource is globally identifiable through its Uniform Resource Identifier (URI), and the following HTTP methods are used:

- ▶ POST: Create a resource representation.
- ▶ GET: Read a resource representation.
- ▶ PUT: Update a resource representation.
- ▶ DELETE: Delete a resource representation.

### 2.2.2 JSON

JavaScript Object Notation (JSON) is an open standard format for data interchange. Although originally used in the JavaScript scripting language, JSON is now language-independent, with many parsers available in many languages. A JSON message is shown in Example 2-1.

*Example 2-1 JavaScript that uses a JSON-encoded array of name data*

---

```
varr employeeArray = [  
  { "firstName":"John" , "lastName":"Doe" },  
  { "firstName":"Anna" , "lastName":"Smith" },  
  { "firstName":"Peter" , "lastName":"Jones" }  
];
```

---

JSON supports two structures: Objects and arrays. Objects are an unordered collection of name-value pairs, where arrays are ordered sequences of values. JSON also supports simple types, including strings, numbers, Boolean expressions, and null values. This support enables JSON to describe any resource. JSON can be seen as readable by humans and machines. JSON is an easy language for humans to read and for machines to parse.

### 2.2.3 Advantages of REST and JSON

The use of REST services offers the following advantages:

- ▶ They are prescriptive in terms of implementation patterns and security options, which leads to a uniform approach that is intuitive for consumers and providers alike.
- ▶ The barrier of entry for mobile application programmers is set low; JavaScript application programmers can handle HTTP connections and JSON data without requiring extra specialist libraries (for example, for parsing).
- ▶ REST interfaces for z Systems assets are familiar to mobile application programmers and can be used in the same way as industry-standard APIs.
- ▶ They are independent of platform, operating system, and programming language. REST and JSON also are flexible and extensible.
- ▶ JSON can often represent data more concisely than XML. Every element in the tree has a name, and the element must be enclosed in a matching pair of tags. JSON expresses trees in a nested array format that is similar to JavaScript. This ability can enable the same data to be expressed in a relatively smaller data package than with XML, which can be a factor for mobile applications.

## 2.3 SOAP web services

SOAP web services are an implementation of a service-oriented architecture (SOA). A service is an application component that features a well-defined published interface that allows other application components to invoke operations on the service without any knowledge of how the service is implemented.

The technologies that can be used to implement a web services solution received wide acceptance as the strategic way of building distributed IT solutions that integrate heterogeneous applications over the internet and intranets.

The web service specifications are independent of programming language, operating system, and hardware to promote loose coupling between the service requester (or consumer) and service provider. The technology is based on the following open standards:

- ▶ eXtensible Markup Language (XML)
- ▶ SOAP, which is a standard protocol for exchanging XML messages
- ▶ Web Services Description Language (WSDL), which defines an XML grammar for describing web services

The use of open standards provides broad interoperability among different vendor solutions. These principles mean that companies can implement web services without having any knowledge of the service requesters. Also, service requesters do not need to know the implementation specifics of service provider applications. This use of open standards facilitates just-in-time integration and allows businesses to establish new partnerships easily and dynamically.

How a SOAP message consists of an envelope that contains zero or more headers and a body is shown in Figure 2-2.

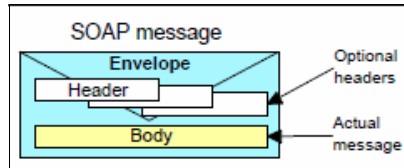


Figure 2-2 SOAP message

Application designers determine the contents of the headers. The SOAP specification does not define what headers should be used. For example, application designers might define a header that contains authentication credentials or information for transaction management. The body is where the main end-to-end information (the payload) that is conveyed in a SOAP message must be carried. This information might be parameters for calling a service for a service request, or the result of calling the service for a service response (see Figure 2-3).

```
<employees>
<employee>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
</employee>
<employee>
  <firstName>Anna</firstName>
  <lastName>Smith</lastName>
</employee>
<employee>
  <firstName>Peter</firstName>
  <lastName>Jones</lastName>
</employee>
</employees>
```

Figure 2-3 SOAP body

### 2.3.1 Advantages of SOAP web services

The use of SOAP web services offers the following advantages:

- ▶ Provides a standard for exchanging data in XML format; for example, the parameters that are used in a program call (for the inbound message) and the data that results from the call (for the outbound message).
- ▶ Is transport, platform, operating system, and programming language independent.
- ▶ Is flexible and extensible.
- ▶ Enables the use of web services standards, such as WS-Security.

SOAP supports the remote procedure call (RPC) style of web service, in addition to the document message style. Although it is transport protocol independent, HTTP is the most widely used protocol today for transporting SOAP messages.

A web service is fully defined in a WSDL file. Most major environments that host applications have development tools that use the WSDL file to generate easy-to-use proxies or adapters to send and receive SOAP messages on behalf of applications.

## 2.4 Messaging

Instead of exchanging information directly from one application to another, messages in a messaging architecture are placed in queues that store them until they are retrieved by an application. A queue manager maintains the queue and is responsible for the integrity and persistence of the message. It can also deliver messages across a network to other queue managers. The full benefits of a messaging architecture are realized when you are integrating disparate software components and you are not in control of the availability and connectivity between these components.

Enabling all applications and data sources to communicate in this way requires a messaging backbone. The messaging backbone can be thought of as a pervasive communications infrastructure. It provides reliable and secure data communication between applications on all computing platforms and in all execution environments. It also provides an inherently loosely coupled way of integrating applications.

In addition to providing connectivity, the messaging backbone should provide ensured qualities-of-transport service (for example, reliable message delivery). A messaging backbone is in many of today's application integration architectures, which are likely to be reused in mobile projects.

In addition to enterprise messaging systems, lightweight messaging transport mechanisms are well-suited to Internet of Things and mobile applications. MQTT is one such transport protocol, which offers a lightweight machine-to-machine publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required or network bandwidth is at a premium. For example, it is used in sensors that are communicating to a message broker through a satellite link, or in a range of home automation and small device scenarios.

The mechanism by which the data is moved also differs between MQTT and HTTP. MQTT follows a reliable-delivery publish/subscribe messaging pattern, rather than the try-and-retry request/response mechanism of HTTP. Again, this way is ideal for mobile applications and Internet of Things devices because it supports the publication to multiple recipients over unreliable networks.

### 2.4.1 Advantages of messaging

The use of messaging offers the following advantages:

- ▶ Provides an asynchronous communication mechanism that enables two or more applications to communicate without both having to be available at the same time.
- ▶ Enables buffering of workloads between applications, such that one application does not overload another with requests.
- ▶ Supports the single-to-many propagation of requests by using practices, such as publish/subscribe.
- ▶ MQTT protocol can be used when network bandwidth is limited, connections are fragile, and when mobile devices have limited memory or processing capabilities.

Industry standards for messaging, such as JMS, enable Java applications to be developed that can be fulfilled by any messaging provider that adheres to the standard.



## Hybrid integration architecture considerations

Integration between cloud services and mainframe applications in a hybrid environment can be seen from different perspectives, including application integration, data integration, and security integration. The goal of these approaches is to decouple systems to reduce the complexity and work effort that is involved in communicating with an enterprise application or data source, distant from the cloud application.

In this chapter, we review the key considerations when planning a hybrid cloud solution that reuses mainframe applications and data.

This chapter includes the following topics:

- ▶ 3.1, “Hybrid integration architecture” on page 16
- ▶ 3.2, “System of Record” on page 17
- ▶ 3.3, “System of Engagement” on page 17
- ▶ 3.4, “Access layer” on page 18
- ▶ 3.5, “Integration layer” on page 19
- ▶ 3.6, “API gateway” on page 20
- ▶ 3.7, “Security gateway” on page 21

## 3.1 Hybrid integration architecture

A hybrid integration architecture must offer flexibility for applications within the enterprise and users beyond it. Historically, a clear boundary separated users that were beyond and within the enterprise. Over time, more complex requirements were placed on the integration components to support newer types of consumers, such as mobile devices and Internet of Things (IoT) devices.

We are now in a world where partner applications sit beyond traditional enterprise boundaries. Some of the enterprise's own applications might be in remote locations and hosted in public cloud environments. This configuration places new requirements for a successful hybrid integration architecture in which services can be discovered and consumed by using REST APIs, web services, or message-based transports.

The major components of hybrid integration architecture are shown in Figure 3-1.

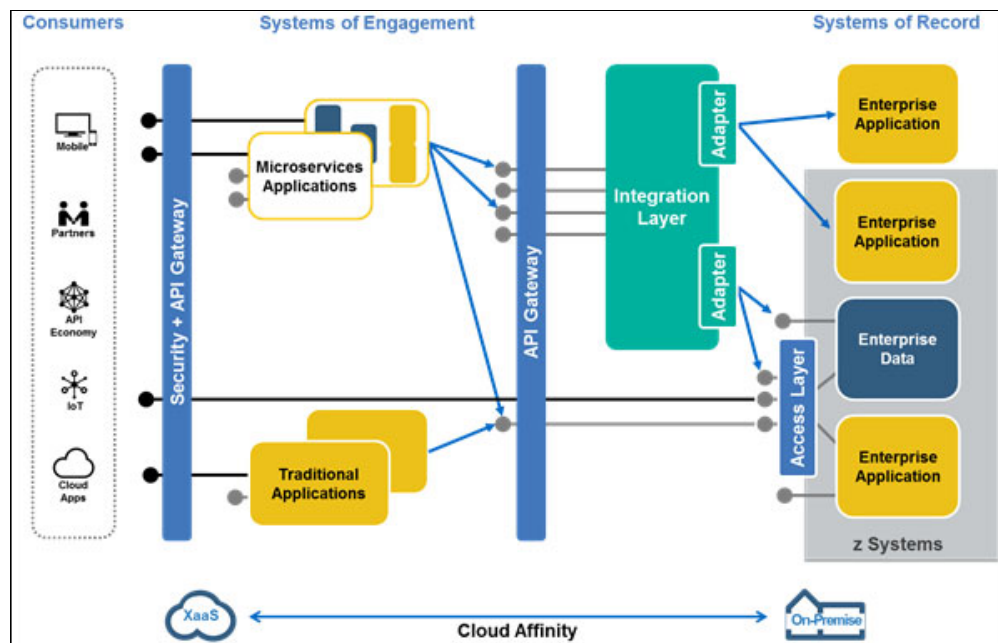


Figure 3-1 Hybrid integration architecture

The components of the hybrid integration architecture are described next.

**Note:** Not all solutions require all of the components that are shown in Figure 3-1. For more information about hybrid integration architectures, see *An Architectural and Practical Guide to IBM Hybrid Integration Platform*, SG24-8351.

## 3.2 System of Record

When contemporary digital applications present services to consumers, it is vital that they can access current and accurate data. In a banking scenario, a cloud-hosted account information service aggregator must provide the same result as queries that are made by using traditional ATM or web channels. The integrity and validity of the account data is maintained by the System of Record (SoR), which can be hosted on different systems, including IBM z Systems mainframe.

How a specific SoR is accessed depends on the type of SoR that is used (for example, CICS, IMS, and DB2), the software version of the SoR, and the type of application interfaces that are made available by the SoR.

## 3.3 System of Engagement

System of Engagement (SoE) applications combine channel-specific logic with custom business logic to provide presentation tier services. They support a diverse set of consumers, ranging from human-driven interfaces to web and mobile applications, to business-to-business applications, and machine appliances, such as IoT devices.

The following main architectural styles are used for building SoE applications:

- ▶ Traditional enterprise applications

These applications are monolithic applications that are developed to a set of standards (often in a single language, such as Java or .NET) that perform multiple related functions. These functions can be in-house developed applications or application packages that supporting business process management functions, for example. The communication mechanisms for these applications tend to be diverse and support messaging, SOAP web services, and increasingly REST APIs.

- ▶ Microservices applications

By contract, these applications are a collective of multiple, loosely coupled components (as shown in yellow and blue boxes in Figure 3-1 on page 16). Each component performs a single function, can be developed in a language that is best-suited to the function, and can be scaled independently of another. The communications models for these applications are almost always based on a combination of messaging and REST APIs.

Multiple deployment options are available for SoE applications. Traditionally, these deployments were within the enterprise, but increasingly are now within private- or public-cloud environments and potentially across a number of these environments. These applications require access to logic and data from the SoR applications, which drive the requirement for hybrid integration options with SoRs, such as z Systems.

## 3.4 Access layer

SoRs offer several mechanisms to access the functional units of an application by providing an access layer of interfaces that are well-defined and implementation independent. This access layer can be implemented by the enterprise applications or enterprise data systems, or by using gateway components within the SoR.

Consider the following questions when deciding how SoE applications should access the SoR:

- Does a service interface exist?

In some cases, the SoR is not service-enabled, and requires an integration component to convert an API request (typically REST/JSON) into the data format and transport protocol that is supported by the SoR application. This integration component might be in the integration layer and natively invoke functions in the SoR.

Alternatively, the SoR might already have been service-enabled by using SOAP web services. Although some SoE applications might consume these services directly, an increasing trend is to exploit RESTful APIs. In this case, an integration component (which might be in the API gateway or the integration layer) is required to map the API onto the web service.

However, making a REST service available directly might be more convenient and streamlined. For this reason, traditional mainframe SoR products now provide options for accessing applications that use JSON over HTTP. In addition, some gateway components of the access layer can convert REST APIs and invoke SoR applications natively.

- What service granularity is made available by the SoR application?

An important characteristic of the service interface is the service granularity (course-grained or fine-grained). It is possible, even likely, that the granularity of the interface must be adapted. For example, of the thousands of underlying functions that are performed by an SoR application, some can be surfaced as a widely useful service, such as a balance inquiry. However, in other cases, it might be necessary to combine multiple fine-grained services to make available a more useful, course-grained API. This kind of aggregation often is deployed in the integration layer or API gateway.

- How will services be discovered?

The lifecycle of a typical SoE application is much shorter than the traditional mainframe application. To facilitate fast and agile development of SoE applications, it is necessary to allow developers to quickly find what services are available. Ideally, a developer should be able to search a catalog of services and retrieve the associated Swagger document that is required to invoke each API.



## 3.5 Integration layer

An integration layer provides a means to connect service requesters and providers. Each invocation from a service requester (for example, a SoE application) results in one or many invocations out to a service provider (for example, a mainframe SoR). The integration layer often handles API composition, event handling, data synchronization, and adapter-based technology integration.

SoE service invocation messages should be lightweight and consist of a limited number of primitive data types and operations. The integration layer can handle the necessary routing, mediation of service interface differences, data transformations, protocol transformations, caching, and orchestrations, retry logic, exception handling, and so on.

Consider the following questions when deciding what role an integration should play in a hybrid integration architecture:

- Is an integration layer used in the application integration architecture today?

Probably the most compelling reason for the use of an integration layer is if such a component is being used today to enable integration with service requesters. The integration layer often has a pivotal governance role in applying policies, such as authentication, audit, logging, and service versioning. In this case, it is likely that SoE application requests are subject to the same governance policies.

It is now the norm for an integration layer to support the request protocols and data formats that are associated with RESTful APIs and cloud services.

- How many different types of service requester and service provider must be supported by the application integration architecture?

The value of an integration layer is partly determined by the range of different service requesters and providers that must be integrated. When a significant and growing number of endpoints require support for different protocols and data formats, the integration layer plays a crucial role in avoiding a spider web of point-to-point connections. An integration layer also makes it easier to introduce systems into the application integration architecture if business mergers and acquisitions occur.

- Does the SoE application need to support asynchronous requests?

For asynchronous requests, the response of a service provider must be correlated to the original request from the service requester. The requester must determine which request a response answers. An integration layer can provide correlation for asynchronous invocations by using a messaging engine.

## 3.6 API gateway

An API gateway simplifies SoE integration by enabling:

- ▶ API developers to create, secure, control, deploy, analyze, and manage SOAP and REST APIs.
- ▶ API business owners to advertise, market, socialize, internally for cross-charging purposes, or externally to sell APIs as products.
- ▶ Application developers to easily find, understand, and use APIs.
- ▶ IT operations staff to manage and upgrade the API environment to use IT investments.

The API gateway provides access to APIs and essential services, such as security, governance, monitoring, and analytics. For example, metering API invocations enable rate limiting to be applied and API use to be charged.

The API gateway can also provide the underlying technology to support message-format translation and version and change management. It can be deployed in the same physical or virtual server as the security gateway, depending on if it makes services available internally within an organization or beyond.

API gateways can be implemented to address different requirements. As shown in Figure 3-1 on page 16, two logical API Gateways are used: One to support requests from external sources and one to support internal requests. The ability to deploy multiple API Gateways has many advantages that are related to the different non-functional requirements that are expected of external versus internal users of API services.

Consider the following questions when deciding what role an API gateway should play in a hybrid integration architecture:

- ▶ Do you need to make your business services more usable?

Making APIs usable means more than just providing key technical information about how to invoke the APIs; that is, the interface description. APIs should be simple to look up from an easily browsable and searchable catalog.

- ▶ Do you want to reach new markets, customers, and partners?

By making core business functions available as APIs to Business Partners, a business can deliver more comprehensive services and reach more customers.

- ▶ Do you need more control over who uses your business services?

The API gateway can check the entitlement for the invoking application, control workload, and generate audit data on each invocation. The collected audit data can then be analyzed and presented as a report for gaining insight into API invocation; for example, which APIs are invoked, how often, and by which applications.

- ▶ Do you need to charge consumers for accessing your business services?

The audit data that is collected by the API gateway can be used for charging.

## 3.7 Security gateway

The most effective mechanism to ensure that proper access management policies are enforced is by using a centralized security gateway. The gateway serves as an entry point for requests from external application traffic into the enterprise. It delivers a configurable set of capabilities to protect and enhance SoE interactions from beyond the enterprise.

The security gateway acts as the policy enforcement point (PEP) for all authentication and authorization decisions that are related to in-bound and out-bound traffic. This architecture allows enterprises to decouple the enforcement of security policy from the underlying application. It also provides functional offloading of security capabilities to allow the SoR applications and resources to more efficiently scale to meet the high volume demands that inevitably occur with mobile and cloud-initiated traffic.

For the most sensitive applications, a security gateway can be deployed as a physical appliance with tamper proof protection.

Consider the following questions when deciding the role that a security gateway should play in a hybrid integration architecture:

- ▶ Is a security gateway used in the application integration architecture today?

The security challenges of hybrid cloud solutions overlap with the challenges of other integration solutions; for example, protection against denial of service and other threats and rate limitation and centralized security policy control. If these types of security policies are implemented in a security gateway today, it makes sense to extend the gateway functionality to control access from applications and services that are hosted beyond the enterprise boundary.
- ▶ What are the specific security requirements of the hybrid application?

Different applications have different security requirements that are influenced by the following factors:

  - Type of user (employee, client, partner, and so on)
  - Type of device (cloud, mobile, IoT, and so on)
  - Type of SoR services that are required by the service requester
  - Authentication, authorization, and audit requirements
  - Confidentiality and data integrity requirements
  - Company and industry standards that must be respected

A security gateway can play an important role in any hybrid integration architecture. However, it will probably not be the only security policy enforcement point. An end-to-end view is required in which access control is enabled in each component of the architecture. For example, some solutions require that the user's identity flows securely with the request message and passes through different layers of the application architecture until it arrives in the mainframe SoR.





## IBM integration solutions

This chapter describes the main solutions and products that can be used for REST API enablement of z Systems applications. We also provide an overview of each solution and guidance on when to use each one.

This chapter includes the following topics:

- ▶ 4.1, “IBM z/OS Connect Enterprise Edition” on page 24
- ▶ 4.2, “CICS” on page 28
- ▶ 4.3, “IMS” on page 35
- ▶ 4.4, “DB2 for z/OS” on page 38
- ▶ 4.5, “IBM MQ” on page 40
- ▶ 4.6, “IBM Integration Bus” on page 43
- ▶ 4.7, “IBM API Connect” on page 44
- ▶ 4.8, “IBM DataPower Gateway” on page 48

## 4.1 IBM z/OS Connect Enterprise Edition

z/OS Connect Enterprise Edition (z/OS Connect EE) provides a single common gateway for REST HTTP calls to reach business assets and data on z/OS operating systems. Where these assets run is specified in the z/OS Connect configuration, which relieves client applications in the cloud, mobile, and web worlds of the need to understand the details about how to reach them and how to convert payloads to and from the formats that the applications require. Services can be enabled without writing code and tooling is provided for creating the data transformation artifacts.

With z/OS Connect EE, mobile and cloud application developers can incorporate z/OS data and transactions into their applications whether they work inside or outside the enterprise without needing to understand z/OS subsystems. The z/OS resources appear as any other REST API.

An overview of z/OS Connect EE is shown in Figure 4-1.

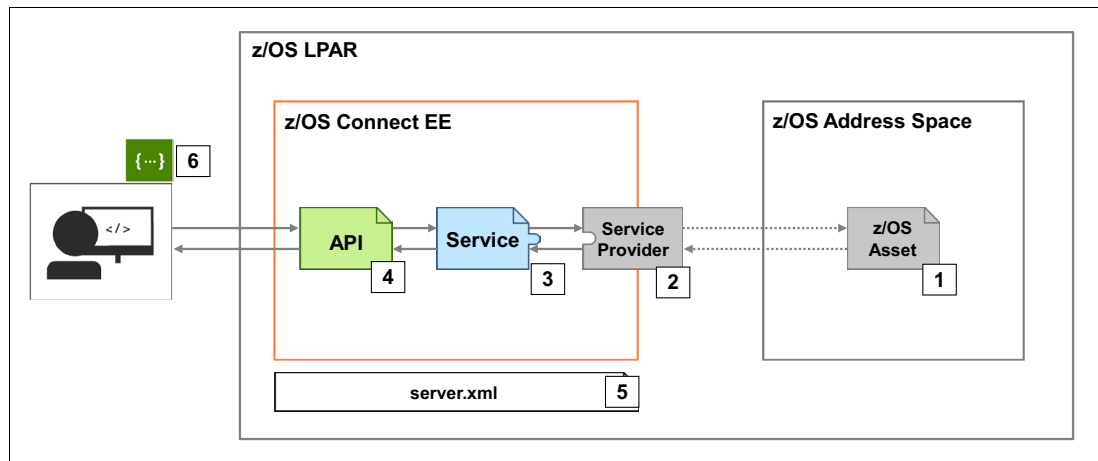


Figure 4-1 z/OS Connect EE

**Statement of Direction:** IBM intends that a future release of z/OS Connect EE will provide the ability for IBM z/OS subsystems to become consumers of REST/JSON APIs.

The following components are shown in Figure 4-1:

### 1. z/OS asset

z/OS Connect EE provides a framework that enables z/OS assets (programs and data) to be enabled as APIs so that they can be more easily used by mobile and cloud applications. z/OS connect EE supports the following types of z/OS assets:

- CICS applications
- IMS applications
- DB2 data
- IBM MQ queues and topics
- Custom long-running tasks

## 2. Service providers

A z/OS Connect EE service provider forwards requests to a System of Record (SoR). You can use one of the following IBM supplied service providers:

- A WebSphere Optimized Local Adapter (WOLA) service provider for connecting to CICS and custom long-running tasks.
- An IMS service provider for connecting to IMS using IMS Connect.
- A REST Client service provider for connecting to a REST service (HTTP/JSON endpoint), for example, a DB2 REST service, or a REST service running in WebSphere Application Server for z/OS.
- An IBM MQ service provider for putting or getting messages from an IBM MQ queue.

You also can write your own service provider that implements the z/OS Connect EE Service Provider Interface (SPI) `com.ibm.zosconnect.spi.Service`.

## 3. Services

Before you create an API, you must create and configure services that provide information about the z/OS asset, including its expected request and response JSON schemas and information about how to connect to the service. The way that you create the service depends on the type of z/OS asset that is being API enabled.

z/OS Connect EE services can be invoked directly by using a basic “remote procedure call” model of REST where typically an HTTP POST is used with the required JSON request and response messages. However, the full value of z/OS Connect EE is achieved when an API layer is built on top of the JSON services.

## 4. APIs

The API defines the REST interface that you want to enable for the z/OS asset, including what HTTP verbs are used, the format of the URIs, and the specific services on which the different paths of the API are implemented.

The API mapping model provides more fine-grained control of the format of the JSON request and response messages, and the use of URI query parameters, path parameters, and HTTP headers in the design of the API. It adds a powerful abstraction layer between the API consumer and the underlying z/OS assets. The mapping model allows inline manipulation of requests, such as mapping HTTP headers, pass-through, redaction, or defaulting of JSON fields.

Behind each API, the JSON request-response schema that is associated with a specific HTTP method (GET, POST, PUT, and DELETE) is mapped to an associated service.

## 5. server.xml

z/OS Connect EE is based on Liberty server technology; therefore, services and APIs are configured in the Liberty `server.xml` file.

## 6. Swagger document

The z/OS Connect EE API Editor generates a Swagger document that is used by the client application developer to generate code that invokes the API.

**Note:** z/OS Connect EE is a separately orderable IBM product. It builds on the capabilities of z/OS Connect V1.0, a feature of WebSphere Liberty Profile z/OS that included support for JSON services only. z/OS Connect V1 functionality is stabilized.

### 4.1.1 API design and workflow

How z/OS Connect EE enables the creation, deployment, discovery, and invocation of an API is shown in Figure 4-2.

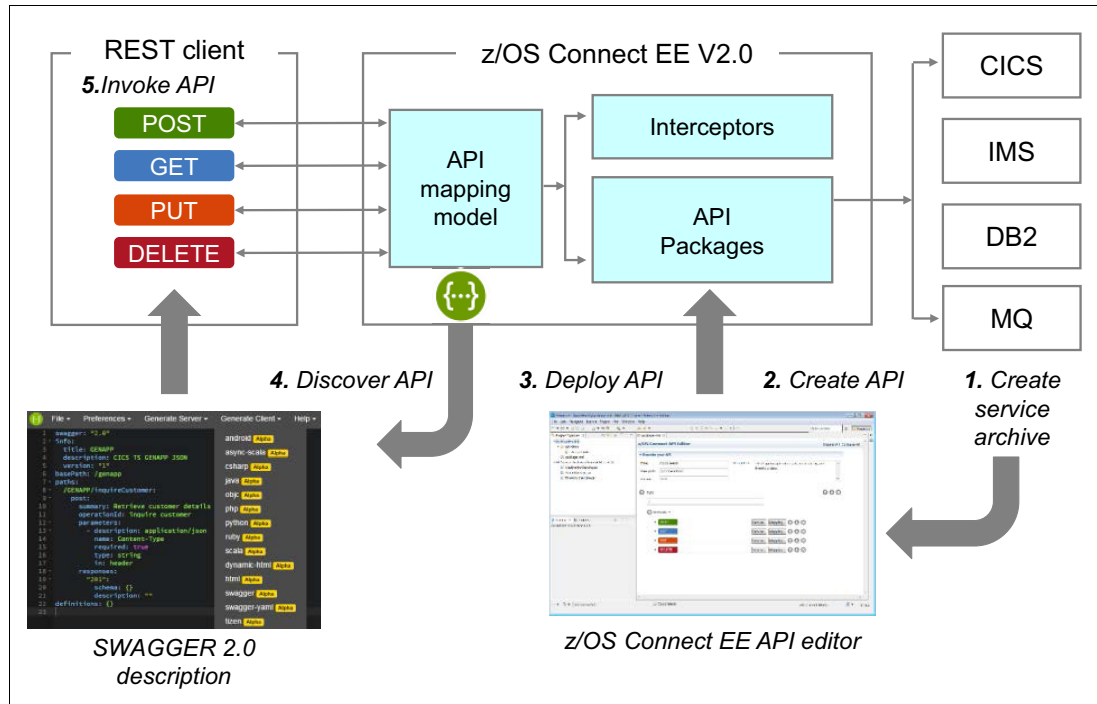


Figure 4-2 z/OS Connect EE API deployment flow

The following steps in the z/OS Connect EE API deployment flow are shown in Figure 4-2:

#### 1. Create a service archive.

A service archive (SAR file) contains the information that is needed by a z/OS Connect EE service provider to install and provide the service, and to enable the service as a JSON asset. Service archives are required for service interface mapping with the z/OS Connect EE API Editor.

An overview of a service archive is shown in Figure 4-3.

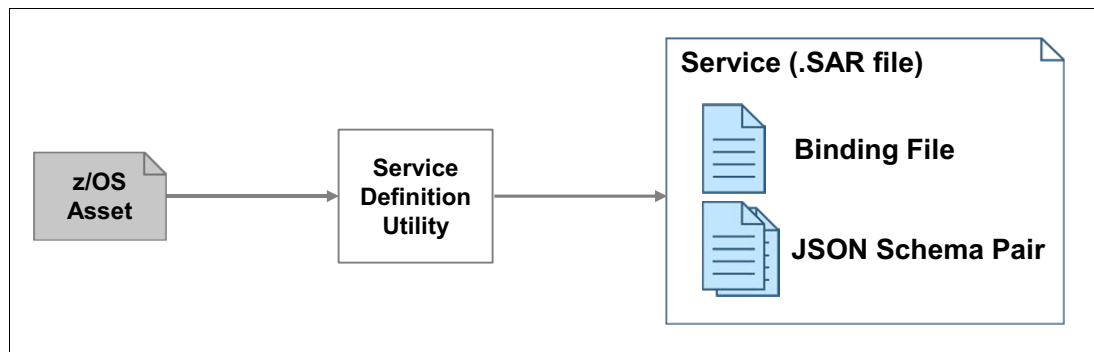


Figure 4-3 z/OS Connect EE service archive



Depending on the z/OS subsystem, you use one of the following tools to generate the service archive:

- For IMS, use IMS Enterprise Suite Explorer for Development.
- For CICS, use the z/OS Connect EE utilities (BAQLS2JS and BAQJS2LS).
- For DB2, use the z/OS Connect EE Build Toolkit.

**Note:** Currently, the IBM MQ service provider supports only the service interface of z/OS Connect EE. You cannot create a service archive for an IBM MQ service or use the z/OS Connect EE Editor to customize the REST interface.

2. Design and create the API by using the eclipse-based z/OS Connect EE API Editor.

The user interface of the API Editor is intuitive for a z/OS developer who is familiar with REST API concepts. Behind each API definition, the format of a REST API request can be mapped to an associated z/OS Connect EE service. In combination with values from the HTTP headers and URI path and query parameters, JSON fields are mapped to the JSON request and response schema that are associated with the service interface of the z/OS asset.

An example API mapping model is shown in Figure 4-4.

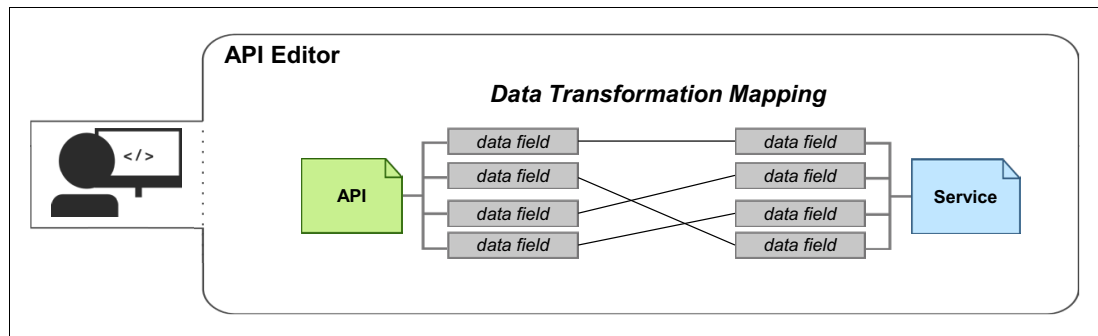


Figure 4-4 z/OS Connect EE API mapping model

JSON fields can be passed-through, moved, omitted, or set to a default value. A Swagger document is generated, along with the HTTP-to-JSON mapping and other API and service-related information. If necessary, you can edit the generated Swagger document in the editor.

3. Deploy the API to a connected server.

You can deploy the API directly from within the editor by right-clicking the API project and selecting the corresponding menu item.

You can also deploy the API by using the **apideploy -deploy** command. By using this approach, you export the API project as an API archive (AAR) file.

4. Examine and test your API in the editor by using the embedded Swagger UI.

5. Generate client code or import the Swagger document into an API management system, such as IBM API Connect™. For more information, see 4.7.1, “How IBM API Connect works with z/OS Connect EE” on page 45.

### 4.1.2 z/OS Connect EE run time

z/OS Connect EE is based on Liberty server technology and is lightweight and easily configurable. z/OS Connect EE benefits from unique z/OS capabilities, such as SAF security integration, z/OS Workload Manager (WLM), and audit logging to SMF. SAF integration means that z/OS Connect EE supports z/OS Identity Propagation that can be used to map a distributed user ID to an IBM RACF® user ID and then pass the mapped RACF user ID onto the SoR (for example, CICS or IMS). WLM integration means different URLs can have varying levels of priority and performance criteria.

z/OS Connect EE provides a framework that enables *interceptors* to work with operations, such as service invoke, status, start, or stop. z/OS Connect EE provides interceptors to perform tasks, such as System Authorization Facility (SAF) authorization, SMF activity recording, and logging JSON payloads. You can also write your own interceptors that implement the z/OS Connect EE `com.ibm.zosconnect.spi.Interceptor SPI`.

### 4.1.3 When to use z/OS Connect EE

Consider the use of z/OS Connect EE V2.0 for REST API enablement when you want to perform the following tasks:

- ▶ Provide intuitive, workstation-based tooling that enables a developer to create REST APIs from traditional z/OS based assets.
- ▶ Simplify the REST API development process by making the mainframe application owner responsible for creating APIs.
- ▶ Support the discovery of defined APIs by using the OpenAPI standard to share API definitions as Swagger documents.
- ▶ Enable interoperability between z/OS Connect EE and API management solutions.
- ▶ Manage API access control using SAF and audit access to SMF.
- ▶ Minimize the required changes to SoRs.
- ▶ Use Java based message transformation that can be offloaded to zIIP speciality engines.

For more information about z/OS Connect EE, see this website:

<https://ibm.biz/zosconnectdc>

## 4.2 CICS

CICS is used extensively for high-volume transaction processing. In this section, we introduce the following main integration solutions that can be used for API enablement:

- ▶ z/OS Connect EE
- ▶ Java applications in a CICS Liberty JVM server
- ▶ CICS SOAP web services
- ▶ CICS JSON web services
- ▶ CICS Transaction Gateway JSON web services

## 4.2.1 z/OS Connect EE with CICS

z/OS Connect EE enables the creation and deployment of APIs that reuse CICS applications. z/OS Connect EE can be configured to connect to CICS using the WOLA service provider (z/OS Connect runs in a separate address space) or deployed inside CICS.

### z/OS Connect EE WOLA service provider with CICS

An overview of the use of the WOLA service provider with CICS is shown in Figure 4-5.

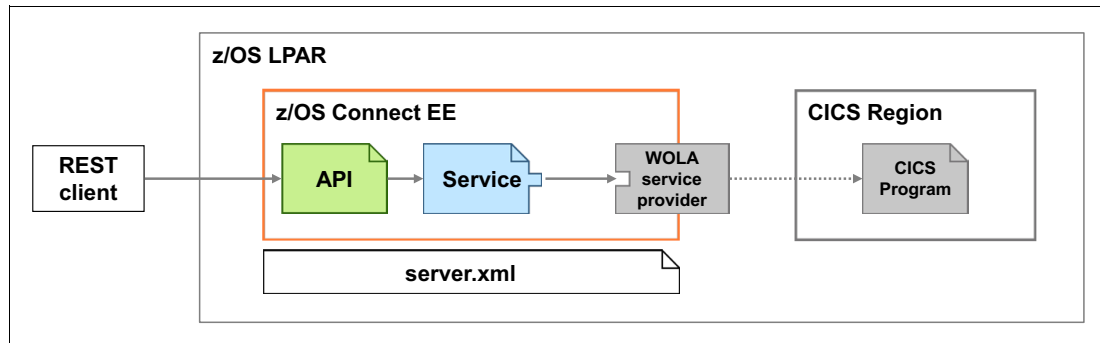


Figure 4-5 Using the z/OS Connect EE WOLA service provider with CICS

As shown in Figure 4-5, the REST client invokes an API using the interface that is shared in a Swagger document. The API mapping model of z/OS Connect EE interprets the request by inspecting the URI, HTTP headers, and JSON body, and then maps the request to a service. The service definition provides information about the CICS program, including a JSON schema representation of its expected COMMAREA or CHANNEL interface. The request message is converted from JSON to a byte array and the CICS program is invoked by using the WOLA service provider.

The WOLA service provider is included with z/OS Connect EE and uses the WOLA function that is provided with the Liberty profile. WOLA provides an efficient cross-memory mechanism and can exchange messages at high-volumes.

### When to use the z/OS Connect EE WOLA service provider with CICS

In this configuration, the same z/OS Connect EE instance can be used for API enablement of different SoRs (CICS, IMS, DB2, and so on). For more information about the advantages of using z/OS Connect EE for creating and deploying APIs, see 4.1.3, “When to use z/OS Connect EE” on page 28.

**Statement of Direction:** IBM intends that a future release of z/OS Connect EE will include an alternative service provider component for CICS integration, based on the CICS IP interconnectivity (IPIC) capability.

## z/OS Connect EE inside CICS

An overview of running z/OS Connect EE inside CICS is shown in Figure 4-6.

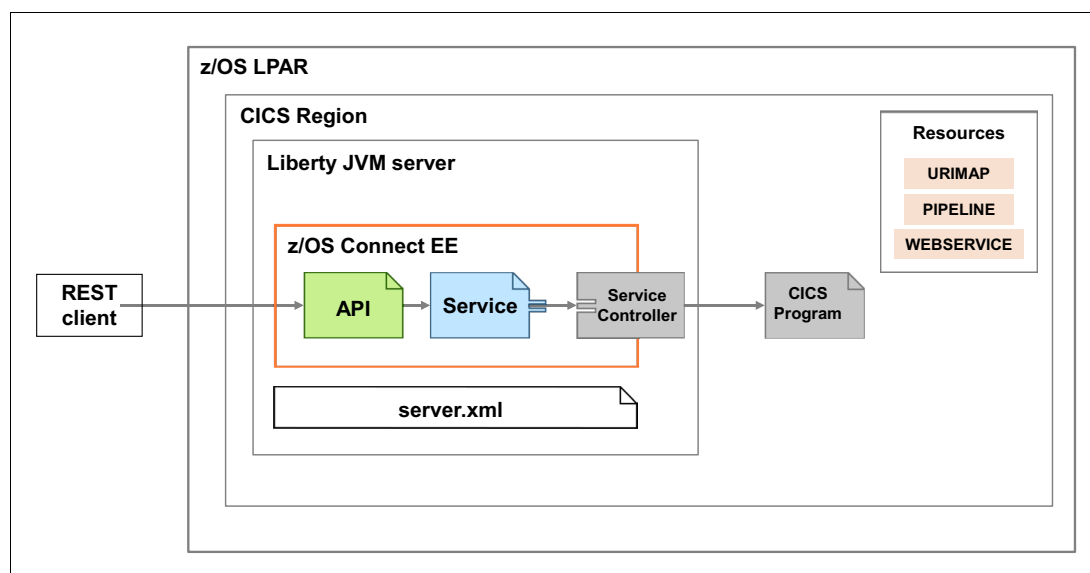


Figure 4-6 z/OS Connect EE inside CICS

In this configuration, the Liberty JVM server is configured inside CICS. This configuration enables an optimized, high-performance solution for REST API enablement of CICS applications.

When z/OS Connect EE receives an API request, it maps the request to a service, runs any configured interceptors, and calls the CICS Service Controller.

The Service Controller checks for any CICS JSON web service resources that are defined to control request processing; for example, a URIMAP can specify the transaction ID to be used for the request. It then uses a local CICS link to call the target CICS program. These programs might then access DB2 or VSAM data, or other external subsystems.

This configuration offers the choice to parse JSON request messages by using Java in the Liberty JVM server (the default), or to use a non-Java JSON parser. In some cases, the non-Java JSON parser can provide a reduction in overall CPU usage per request, but less processing is offloadable to zIIP speciality engines.

### When to use z/OS Connect EE inside CICS

Consider the use of z/OS Connect EE for CICS for API and service enablement when you want to perform the following tasks:

- ▶ Deploy, manage, and monitor CICS REST APIs using CICS procedures and tooling (for example, IBM CICSplex® SM and IBM CICS Explorer®).
- ▶ Reuse a CICS JSON web services configuration.
- ▶ Automatically scan for CICS JSON web service resource definitions, which avoids the need to manually define each service in `server.xml`.
- ▶ Use non-Java JSON data transformation (CICS TS V5.3 only).

For more information about other advantages of the use of z/OS Connect EE for creating and deploying APIs, see 4.1.3, “When to use z/OS Connect EE” on page 28.

**Statement of Direction:** IBM intends that each service provider component that is supplied with a future release of z/OS Connect EE will be supported when deployed in a CICS Liberty JVM server run time.

## 4.2.2 Java applications in a CICS Liberty JVM server

The WebSphere Liberty profile is a dynamic, easy to use Java EE application server. It is certified for the Java EE 7 full platform and includes support for web applications, Enterprise JavaBeans, and Java web services. The Liberty server runtime is provided and licensed with CICS and requires no extra installation steps.

To develop a RESTful service, your program can use the Java API for RESTful Web Services (JAX-RS). To develop a SOAP-based service, your program can use the Java API for XML Web Services (JAX-WS). The Java program can link to other CICS programs and access CICS data and queues using the JCICS API.

An overview of this scenario is shown in Figure 4-7.

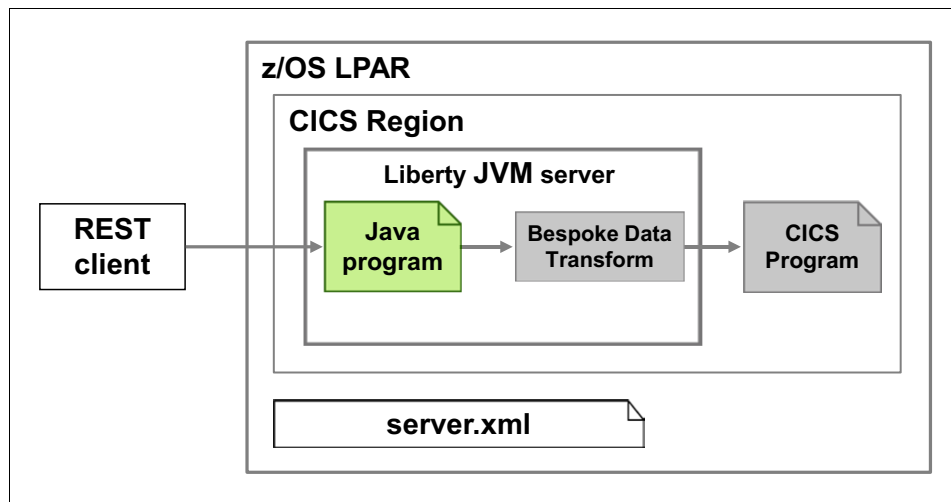


Figure 4-7 Java application in CICS Liberty JVM server

The Java program also can access local and remote relational databases using JDBC and use JMS to get and put messages from messaging run times, such as IBM MQ.

The JZOS batch toolkit for z/OS supplies a set of handy tools for Java on z/OS. Together with these tools, JZOS supplies a Java record generator for COBOL and assembler language structures. This utility takes a COBOL record structure as input and generates JavaBeans with getter and setter methods to provide access to the fields within the record. Alternatively, the IBM Rational® J2C tooling that is supplied with IBM Developer for z Systems Enterprise Edition also can be used for the same purpose.

The Link to Liberty capability of CICS TS V5.3 enables a CICS program to invoke a Java EE application that is running in a Liberty JVM server. This feature can be useful if a CICS application needs to invoke an external REST API. The CICS application can link to a Java program that is running in a CICS Liberty JVM, which can then use JAX-RS to invoke the external API.

### When to use Java applications in a CICS Liberty JVM server

Consider the use of custom Java applications in a CICS Liberty JVM server when you want to perform the following tasks:

- ▶ Develop Java integration logic that reuses CICS programs; for example, a Java application that links to several COBOL programs and returns a single JSON response.
- ▶ Develop new Java based business services in CICS.
- ▶ Have complete control over the application interface (REST or SOAP).
- ▶ Use Java frameworks to handle complex data transformations.
- ▶ Call external APIs from CICS.
- ▶ Use Java based message transformation that can be offloaded to zIIP speciality engines.

For more information about running Java applications in a CICS Liberty JVM server, see the following CICS Developer Center website:

<https://developer.ibm.com/cics/>

## 4.2.3 CICS SOAP web services

Application programs that are running in CICS can participate in a heterogeneous web services environment as service requesters, service providers, or both.

An outline of how CICS processes SOAP XML requests by using a SOAP provider pipeline is shown in Figure 4-8.

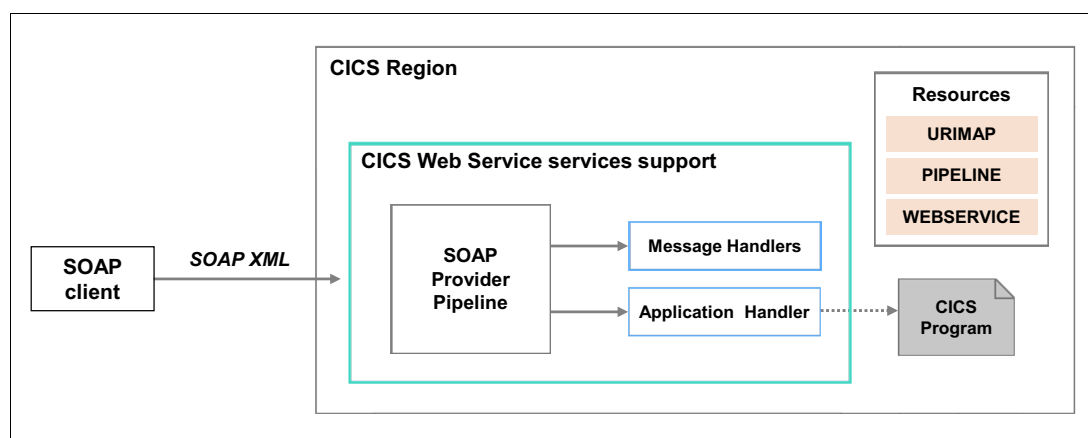


Figure 4-8 CICS SOAP web services

The pipeline defines a set of message handlers that act on a service request and response. A message handler is a program in which you can perform your own processing of web service requests and responses; for example, security processing.

A CICS provided application handler is responsible for processing the body of a SOAP request and for generating a response using the returned data. It maps XML into a byte array and links to the target CICS program.

CICS web services support is configured by using a set of resource definitions, including URIMAP, PIPELINE, and WEBSERVICE definitions. These resources control the processing that CICS performs; for example, the WEBSERVICE points to the web services binding file that is used to parse the XML body of the SOAP message. The binding file is created by using the web services assistant.

CICS support for web services conforms to open standards, including SOAP, WSDL, and several web services standards, such as WS-Security. CICS also supports sending SOAP messages to external service providers.

**Note:** CICS web services support is a mature solution that was widely adopted.

### When to use CICS SOAP web services

Consider the use of CICS web services for REST API enablement when you want to perform the following tasks:

- Reuse a CICS web services infrastructure. In this approach, the REST API call is handled in an intermediary gateway, and the CICS application is then invoked using a SOAP web service.

**Note:** This solution is a tactical solution for REST API enablement. However, as the number of REST client applications grows, it might be more efficient over time to enable a REST JSON interface directly with the CICS application.

- Implement a security model using WS-Security.

## 4.2.4 CICS JSON web services

The CICS web services infrastructure also supports the processing of JSON requests. This support is similar to the service interface that is provided by z/OS Connect EE and it shares the data transformation tools and deployment artifacts.

**Note:** For Java parsing of JSON messages, z/OS Connect EE is recommended because it performs better than the CICS JSON web service support, which is performed within an Axis2 JVM server.

In CICS TS V5.3, support was added for a non-Java service provider pipeline. This addition is appropriate when a non-Java solution is required; for example, on machines in which no zIIP speciality engines are configured. Some workloads might also realize performance and throughput benefits; however, none of the processing is eligible for offloading to zIIP speciality engines.

An overview of the non-Java service provider pipeline is shown in Figure 4-9.

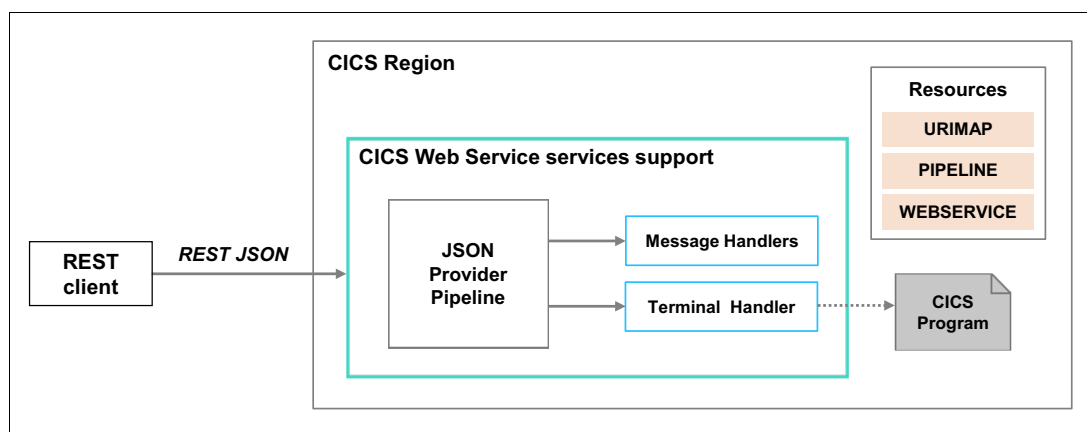


Figure 4-9 CICS JSON web services (non-Java service provider)

As shown in Figure 4-9, the client invokes a CICS JSON web service. The service is defined to be processed in a non-Java JSON service provider pipeline. Optional message handlers can be defined which, for example, can change the contents of the JSON request or response messages. CICS supplies a Terminal Handler program that enables non-Java processing of JSON messages. The request message is converted from JSON to a byte array and the CICS program is invoked.

CICS does not provide built-in support for requester mode JSON web services. If you want to call an external REST API from a CICS application, you can use the **EXEC CICS TRANSFORM DATATOJSON** and **JSONTODATA** commands to transform JSON messages and the **EXEC CICS WEB API** commands to process an HTTP request.

### When to use CICS JSON web services

Consider the use of CICS JSON web services when you want to perform the following tasks:

- ▶ Implement a solution that does not require a CICS JVM.
- ▶ Develop a CICS service requester application that invokes an external REST API.

For more information about CICS JSON web services, see the following IBM Knowledge Center website:

<https://ibm.biz/BdsGH8>

**Note:** In earlier versions of CICS (before CICS TS V5.2), CICS JSON support was provided with the CICS TS Feature Pack for Mobile Extensions.

## 4.2.5 CICS Transaction Gateway JSON web services

CICS JSON web services can also be enabled by using the CICS Transaction Gateway (CICS TG). The JSON transformation is performed within the CICS TG daemon, and a channel or COMMAREA payload is then passed to the CICS program.



An overview of the CICS TG support for JSON web services is shown in Figure 4-10.

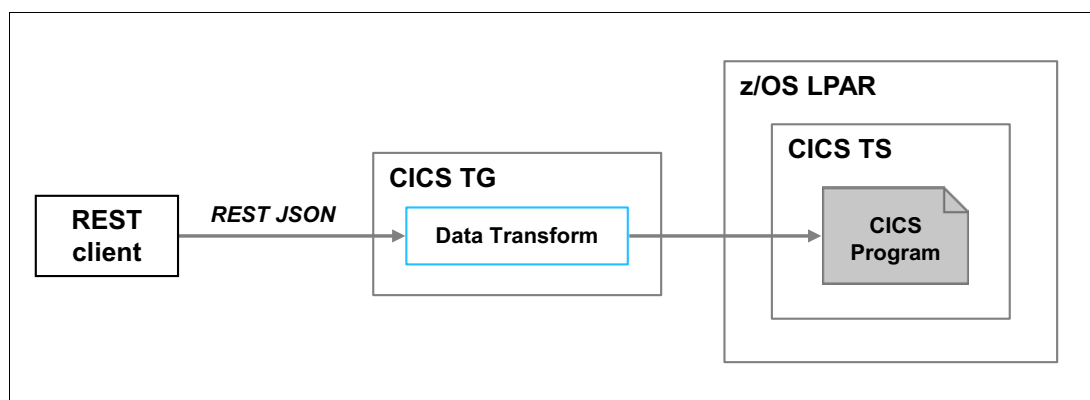


Figure 4-10 CICS Transaction Gateway JSON web services

The CICS TG daemon can run on z/OS or one of many supported distributed platforms.

### When to use CICS Transaction Gateway JSON web services

Consider the use of CICS Transaction Gateway JSON web services when you want to perform the following tasks:

- ▶ Support REST services with older versions of CICS.
- ▶ Reuse a CICS TG infrastructure; for example, a set of cloned CICS TG daemons that enable high availability.

For more information about CICS TG, see this website:

<http://www.ibm.com/software/products/en/cics-ctg>

## 4.3 IMS

IMS provides a high-performance application and data server environment for core business transaction execution and database access.

In this section, we introduce the following main integration solutions that can be used for IMS API enablement:

- ▶ z/OS Connect EE
- ▶ IMS Enterprise Suite SOAP Gateway

### 4.3.1 z/OS Connect EE with IMS

z/OS Connect EE enables the creation and deployment of APIs that reuse IMS applications. z/OS Connect EE can be configured to connect to IMS by using IMS Connect.

An overview of the use of z/OS Connect EE with the IMS service provider is shown in Figure 4-11 on page 36.

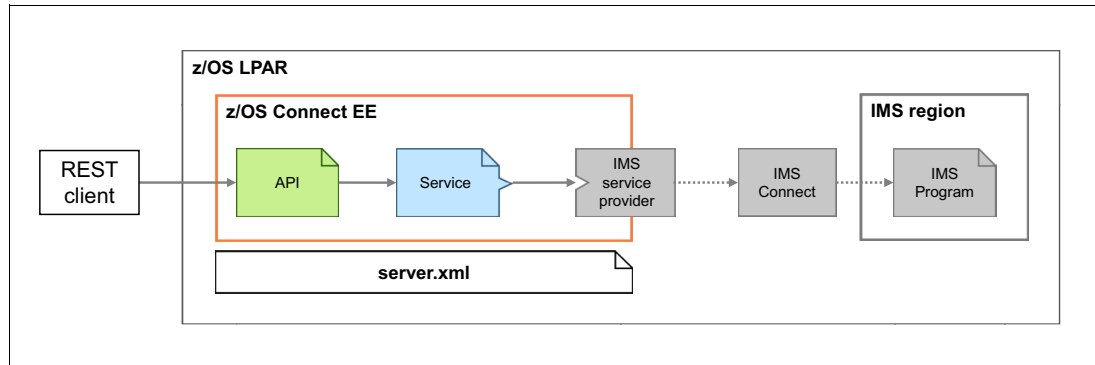


Figure 4-11 z/OS Connect EE with IMS service provider

The REST client invokes an API using the interface that is shared in a Swagger document, as shown in Figure 4-11. The API mapping model of z/OS Connect EE interprets the request by inspecting the URI, HTTP headers, and JSON body, and then maps the request to a service. The service definition provides information about the IMS program, including a JSON schema representation of its expected message segment interface. The request message is converted from JSON to a byte array and the IMS program is invoked using the IMS service provider. The request then goes through IMS Connect to access the IMS program, which might then access IMS DB, DB2, or other subsystems.

This scenario involves the use of the IMS Mobile feature (imsmobile-2.0), which serves as the IMS service provider and is now included with z/OS Connect EE.

**Note:** The IMS Mobile feature previously was included in the IMS Enterprise Suite as the IMS Mobile Feature Pack for z/OS Connect EE.

The IMS Mobile feature includes the following functions:

- ▶ A data transformation module that converts request messages from the JSON format to the native representation of the input message and then converts the response messages to JSON.
- ▶ An IMS Connect adapter module that interacts with IMS Connect for IMS Transaction Manager access.
- ▶ A service management and administration interface that is used by IMS Explorer for Development.
- ▶ A sample ping service to use as an installation verification program (IVP).

The IMS Explorer for Development (an Eclipse-based tool) is used for creating, testing, and publishing services that are based on data structures in IMS COBOL copybooks or PL/I includes. The z/OS Connect EE API Editor then is used to define RESTful APIs that customize the interface of the IMS services.

### When to use z/OS Connect EE with the IMS service provider

Consider the use of z/OS Connect EE V2.0 with the IMS service provider when you want to provide intuitive, workstation-based tooling that enables a developer to create REST APIs from IMS applications. For more information about other advantages of the use of z/OS Connect EE for the creation and deployment of APIs, see 4.1.3, “When to use z/OS Connect EE” on page 28.

**Note:** Unlike with CICS, IMS does not provide direct support for REST JSON services.

For more information about the use of the IMS service provider, see the following Developer Center website:

<https://ibm.biz/zosconnectdc>

### 4.3.2 IMS Enterprise Suite SOAP Gateway

Application programs that are running in IMS can participate in a heterogeneous web services environment as service requesters or service providers that use the IMS SOAP Gateway.

An overview of the SOAP Gateway is shown in Figure 4-12.

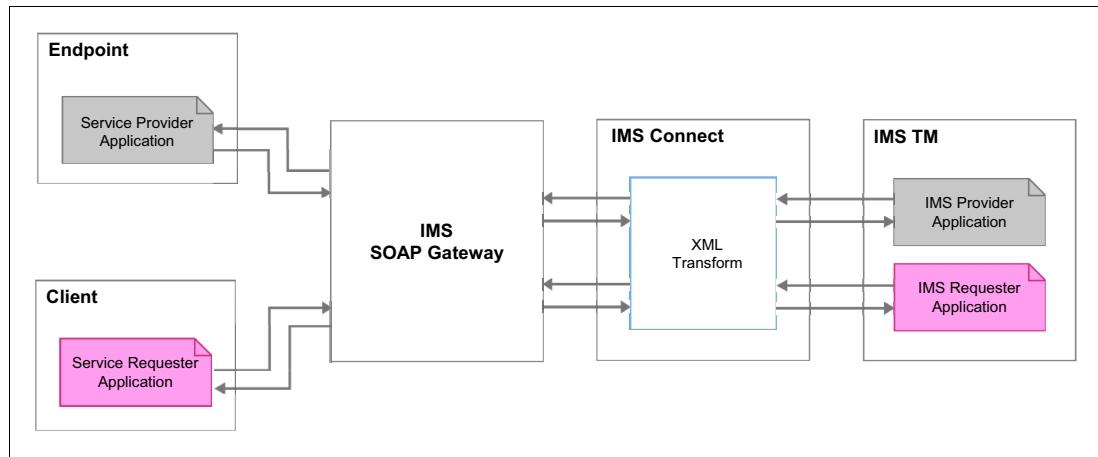


Figure 4-12 IMS SOAP Gateway

The SOAP Gateway acts as the gateway between external web services and IMS applications. The SOAP Gateway communicates with IMS through IMS Connect, which is the TCP/IP gateway for IMS. Messages between the SOAP Gateway and IMS Connect are transmitted in XML format. IMS Connect converts the XML data into bytes and passes the request to the IMS application.

Web service consumer applications (outbound requests from IMS) also are supported. The SOAP Gateway enables IMS applications to make synchronous or asynchronous callout requests to external web services.

The SOAP Gateway can run on z/OS, Linux on z, or a Windows platform. It conforms to open standards, including SOAP, WSDL, and WS-Security.

#### When to use the IMS SOAP Gateway

Consider the use of the IMS SOAP Gateway for REST API enablement when you want to perform the following tasks:

- Reuse an IMS web services infrastructure. In this approach, the REST API call is handled in an intermediary gateway and the IMS application is then invoked using a SOAP web service.

**Note:** This solution is a tactical solution for REST API enablement. However, as the number of REST client applications grows over time, it might be more efficient to enable a REST JSON interface directly with the IMS application.

- Implement a security model using WS-Security.

For more information about the IMS SOAP Gateway, see the following website:

<http://www.ibm.com/software/products/en/imsentesuitsoapgate>

## 4.4 DB2 for z/OS

A large amount of z/OS data is stored in DB2 and there are many benefits to accessing this data as REST APIs. This section describes the following main integration solutions that can be used for DB2 API enablement:

- DB2 REST services
- z/OS Connect EE with DB2 REST services

### 4.4.1 DB2 REST services

As a REST service provider, DB2 enables web, mobile, and cloud applications to interact with DB2 data through a set of REST services. These services are integrated in the DB2 distributed data facility (DDF). You create, discover, run, and manage user-defined services in DB2.

DB2 defines a REST service as a package. Each package contains a single static SQL statement and is stored in a user-defined catalog table. When a service is created, a new row is added to the table that associates the service with its corresponding package. After the package is bound, it can be executed only as a service.

All DB2 REST services are managed as native services. This DB2 native REST service solution uses the DDF capabilities for authorization, authentication, client information management, service classification, system profiling, and service monitoring and display.

DB2 provides a set of system defined APIs that can be used create and discover REST services. An overview of how a DB2 REST service is created and then called from a REST client is shown in Figure 4-13.

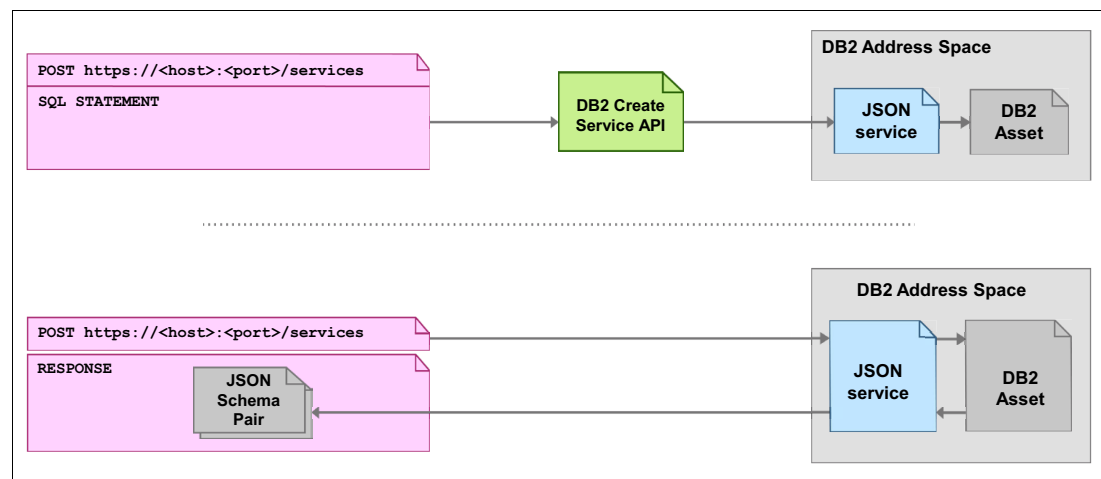


Figure 4-13 DB2 REST services

**Note:** A DB2 stored procedure call can also be specified as input to the API used for creating a DB2 REST service.

An authorized user can discover and invoke the service through a REST HTTP client. DB2 accepts the HTTP POST request, processes the JSON request body, runs the bound SQL statement, and returns any output in JSON.

DB2 REST services do not support different HTTP verbs, API mapping, or discovery using a Swagger document. However, these capabilities can be achieved by creating an API layer in front of the DB2 REST service using z/OS Connect EE. For more information, see 4.4.2, “z/OS Connect EE with DB2 REST services” on page 39.

**Note:** DB2 REST services are available with DB2 V11 or later.

### When to use DB2 REST services

Consider the use of the DB2 REST services when you want to perform the following tasks:

- ▶ Simplify the deployment of mobile or cloud-based applications that require access to DB2 assets.
- ▶ Simplify the REST service development process by making the DB2 data owner responsible for creating service artifacts.
- ▶ Provide a basic DB2 REST service discovery capability.

For more information about the use of DB2 REST services, see the following IBM Knowledge Center website:

<https://ibm.biz/zos-connect-db2-rest-services>

## 4.4.2 z/OS Connect EE with DB2 REST services

z/OS Connect EE enables the creation and deployment of APIs that reuse DB2 REST services. z/OS Connect EE can be configured to connect to DB2 using the REST service provider.

An overview of using z/OS Connect EE with DB2 REST services is shown in Figure 4-14.

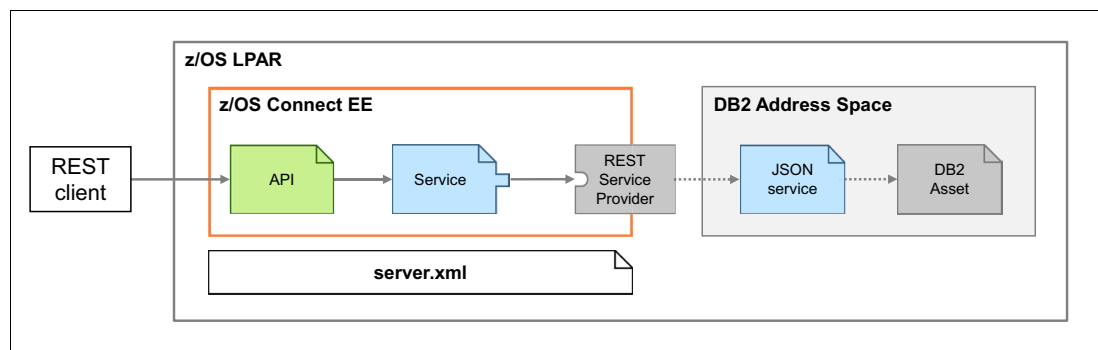


Figure 4-14 Using z/OS Connect EE with DB2 REST services

The REST client invokes an API using the interface that is shared in a Swagger document, as shown in Figure 4-14. The API mapping model of z/OS Connect EE interprets the request by inspecting the URI, HTTP headers, and JSON body, and then maps the request to a service.

The service definition provides information about the location of the REST service. The JSON request message is forwarded to DB2 using the REST Client service provider.

The z/OS Connect EE Build Toolkit is used to create the service archive for the DB2 REST service. The service archive is then imported into the z/OS Connect EE API Editor to create and deploy the API.

**Note:** The support of DB2 REST services with z/OS Connect EE supersedes the DB2 Adapter for z/OS Connect.

### When to use z/OS Connect EE with DB2 REST services

Consider the use of z/OS Connect EE with DB2 REST services when you want to perform the following tasks:

- ▶ Provide intuitive, workstation-based tooling that enables a developer to create REST APIs from DB2 data.
- ▶ Support the discovery of defined APIs by using the OpenAPI standard to share API definitions as Swagger documents.
- ▶ Add a more RESTful interface (for example, use of different HTTP verbs and intuitive URIs) on top of DB2 REST services.
- ▶ Manage API authentication by using the security capabilities of the Liberty server.
- ▶ Manage API access control and audit by using the z/OS Connect interceptor framework.

## 4.5 IBM MQ

IBM MQ remains one of the most widely adopted connectivity patterns for messaging in the enterprise today. Its characteristics of assured, once-and-once only delivery of messages is well-suited to the business critical functions that are provided by mainframe applications and data.

SoE applications can use IBM MQ-based mainframe applications as REST APIs in the following ways:

- ▶ The IBM MQ Service Provider for z/OS Connect allows REST aware applications to interact with z/OS assets that are accessed using IBM MQ queues. You can achieve this configuration without being concerned with the coding that is required to use asynchronous messaging. For more information, see 4.5.1, “z/OS Connect EE with IBM MQ”.
- ▶ IBM Integration Bus provides HTTP services with a JSON parser that enables it to be used as a RESTful façade to IBM MQ-based mainframe applications. For more information, see 4.6, “IBM Integration Bus” on page 43.

### 4.5.1 z/OS Connect EE with IBM MQ

IBM z/OS Connect EE enables the creation and deployment of REST services that reuse IBM MQ-based applications. The IBM MQ Service Provider is delivered as a component of IBM MQ for z/OS.

**Note:** The IBM MQ Service Provider is made available with IBM MQ V9.0.1, but is supported for use with IBM MQ V8 or later.

The IBM MQ Service Provider supports HTTP GET, HTTP DELETE, and HTTP POST verbs that permit basic interaction with IBM MQ. Composing several of these verbs allows more complicated functions to be made available.

The IBM MQ Service Provider also supports the following distinct types of service:

- ▶ A one-way service provides IBM MQ Put and IBM MQ Get support to a single destination.
- ▶ A two-way service provides a request-reply capability in which some form of response is expected as a result of the initial request.

**Note:** The IBM MQ Service Provider supports z/OS Connect services only. API mapping and discovery using a Swagger document are *not* supported.

## One-way services

In the one-way service, REST clients can put or retrieve a message from a queue. To distinguish between different ways of interacting with a queue, REST clients can issue the following verbs to indicate different functions:

- ▶ An HTTP POST puts a message to a queue or topic.
- ▶ An HTTP GET browses a message on a queue.
- ▶ An HTTP DELETE destructively gets a message from a queue.

For example, when a REST client issues an HTTP POST with a JSON payload to a one-way service, z/OS Connect EE puts the message on the target queue or topic (see Figure 4-15).

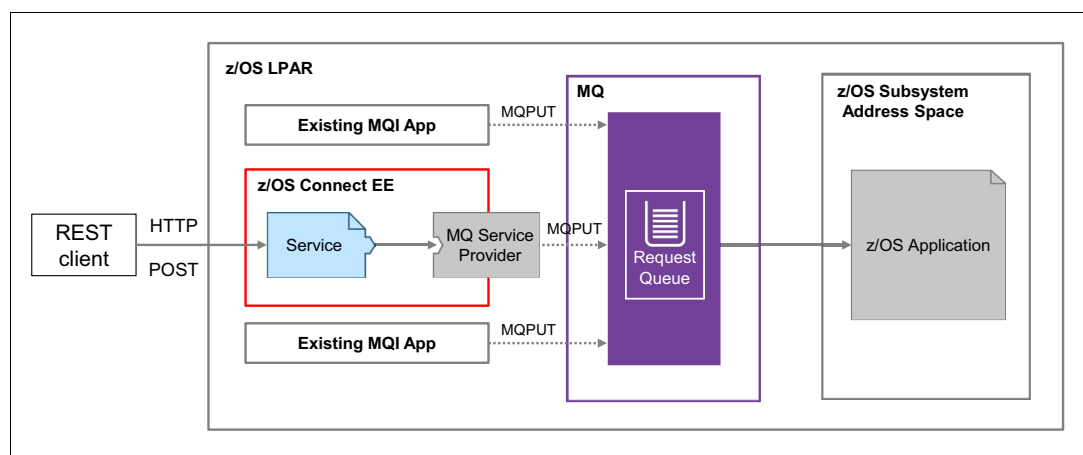


Figure 4-15 Using the z/OS Connect EE IBM MQ service provider with a one-way service

A one-way service is used to provide a REST API on top of a single IBM MQ queue or topic, as shown in Figure 4-15. REST clients can issue an HTTP POST with a JSON payload to a one-way service and the service takes the payload and send a message (with the payload as the message body) to the target queue or topic. When the REST client receives an HTTP 200 response, this response is confirmation that the message was successfully placed on the queue.

**Note:** IBM MQ topics are supported only with an HTTP POST to a one-way service.

z/OS Connect can also be used to convert the JSON payload into an appropriate format; for example, a COBOL copybook. Applications can then get that message from the queue or topic and process it as they do with any other message. They are not aware of the fact that a REST client sent it.

One-way services also allow HTTP DELETE and GET requests to be issued against IBM MQ queues. An HTTP DELETE results in a destructive get of any available message from the queue. An HTTP GET results in a browse of the first available message from the queue. The body of the message is returned to the REST client in the form of JSON. If the payload is not JSON, z/OS Connect can be configured to convert it to JSON.

## Two-way services

A two-way service allows a REST client to perform request-reply messaging against a pair of queues by using z/OS Connect EE, as shown in Figure 4-16.

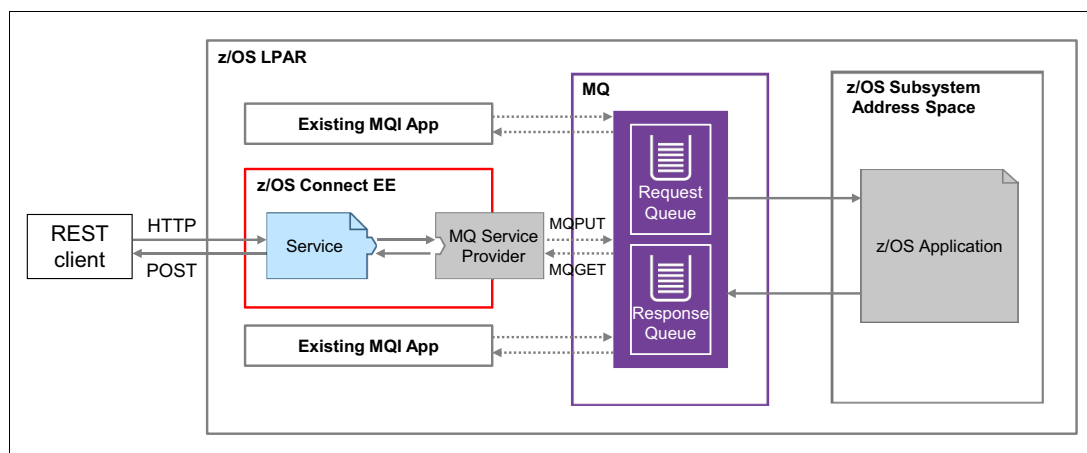


Figure 4-16 Using the z/OS Connect EE IBM MQ service provider with a two-way service

The client issues an HTTP POST request that specifies a JSON payload, as shown in Figure 4-16. z/OS Connect EE processing is the same as in the one-way service, except z/OS Connect EE waits to get the response from the Reply queue.

An IBM MQ-based application, such as CICS or IMS, uses the message, processes it, and generates a response that is placed on the Reply queue. The IBM MQ service provider locates this message by using a correlation identifier, takes its payload, optionally converts it to JSON, and returns it as the response body of the HTTP POST.

**Note:** The IBM MQ service provider for z/OS Connect is available for z/OS Connect V1 and z/OS Connect Enterprise Edition V2.

## When to use IBM MQ with z/OS Connect EE

Consider the use of IBM MQ with z/OS Connect EE when you want to perform the following tasks:

- ▶ Create a REST interface to an IBM MQ-based mainframe application.
- ▶ Decouple REST clients from offline processing systems (such as batch processing applications) in a “fire-and-forget” messaging model.
- ▶ Buffer REST requests to mainframe applications by using the pull-based consumption model of an asynchronous system rather than the push-based model of a synchronous system.

For more information about the use of IBM MQ with z/OS Connect, see the following website:

<https://ibm.biz/BdsGHu>



## 4.6 IBM Integration Bus

IBM Integration Bus provides flexible integration services that offer universal connectivity between any two or more end points. It also provides a comprehensive run time for interpreting, transforming, and routing various message formats.

It fully supports message transports, such as JMS, HTTP, and IBM MQ. It includes libraries for all major message formats, such as JSON, XML, SOAP, fixed length, variable length, tagged, SWIFT, and EDI. In addition to a base set of over 50 built-in visual nodes, various transformation languages are supported, such as Java, extended SQL, XSLT, and WebSphere Transformation Extender.

IBM Integration Bus evolved to address many of the issues that are associated with hybrid integration architectures. It also has recently been enhanced to provide integration services wherever needed. For example, IBM Integration Bus can be deployed as a light-weight integration engine that is deployed within a virtual container to manage digital integration in a cloud environment. In contrast, IBM Integration Bus can also perform the complex integration orchestrations and functions that are associated with a traditional centralized ESB.

The use of a common toolset, powerful transformation techniques, and adapter technologies provides a flexible integration engine that can be deployed anywhere with integration services that are developed by using a common skill set.

Some of the principal integration scenarios that are enabled by IBM Integration Bus are shown in Figure 4-17.

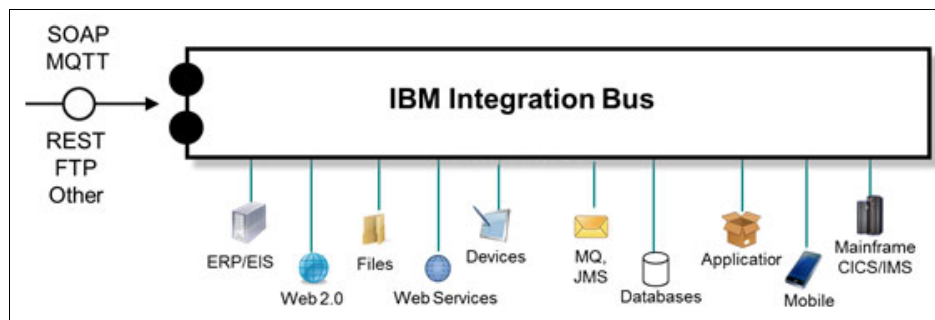


Figure 4-17 IBM Integration Bus

IBM Integration Bus provides secure, scalable access to SoR applications and data from requests that originate in cloud and SoE applications. These SoE applications can start IBM Integration Bus services that can be used for aggregation of endpoints, business rules-based behavior, complex message mapping, transactionality across multiple resources, content-based routing, and much more.

The following REST API integration capabilities are provided by IBM Integration Bus:

- ▶ HTTP services with a JSON parser that provides RESTful façades to back-end systems, and the ability to invoke RESTful endpoints.
- ▶ Dedicated integration services for REST APIs, which are implemented by using REST Request and Response nodes. These nodes allow quick and simple access to APIs that are based on Swagger documents.
- ▶ The ability to generate a Swagger document that then can be pushed to API management tools, such as IBM API Connect.

- ▶ Multiple connectivity options for CICS, IMS, and DB2.
- ▶ The IBM Integration Toolkit, which can be used to import a Swagger document.

#### 4.6.1 When to use IBM Integration Bus

Consider the use of IBM Integration Bus for hybrid cloud and API integration with z Systems when you want to perform the following tasks:

- ▶ Reuse an application integration service that is based on IBM Integration Bus for speed or cost efficient delivery.
- ▶ Implement an integration solution with complex orchestrations or diverse data transformation and protocol requirements.
- ▶ Invoke mainframe applications and data service calls for which a custom interface is required, such as IBM MQ or CICS IPIC.

For more information about IBM Integration Bus, see the following website:

<http://www.ibm.com/software/products/en/ibm-integration-bus>

### 4.7 IBM API Connect

IBM API Connect is a comprehensive platform from IBM for API management. The platform provides a way to create, run, manage, and secure APIs and microservices. It offers analytics, Node.js, and Java support, a system of governing APIs, capabilities for customizing and publishing APIs, security, and more.

IBM API Connect allows developers, small and large businesses, Business Partners, and other stakeholders to work together to develop and manage the entire API lifecycle from one foundational platform. The IBM API Connect offering provides businesses with the tools to reuse their existing assets to create new applications and open new revenue streams.

The following components of IBM API Connect that support the complete lifecycle of an API are shown in Figure 4-18 on page 45:

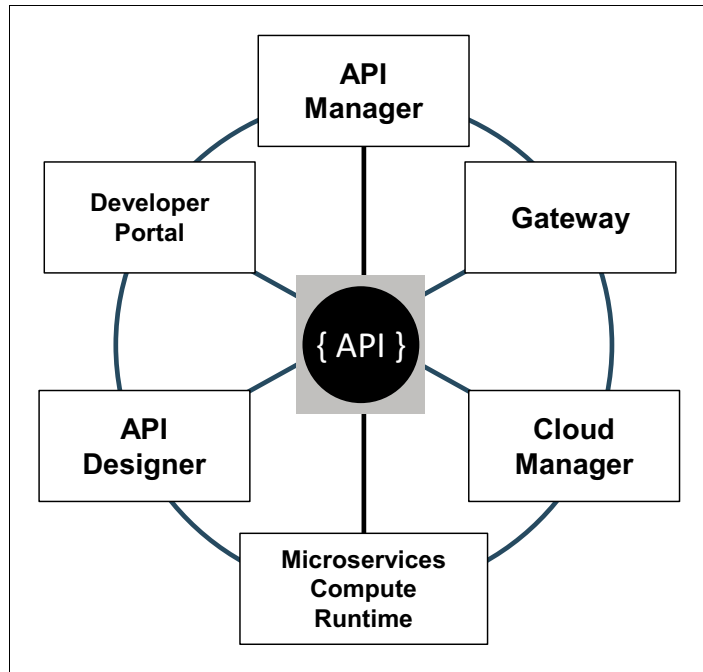


Figure 4-18 IBM API Connect components

- ▶ **API Designer**  
An API developer toolkit that is used to create Microservices and APIs rapidly by using the Node.js LoopBack® and Express frameworks.
- ▶ **Developer Portal**  
A portal for application developers to register their applications, subscribe to use APIs, rate APIs, and post comments.
- ▶ **API Manager**  
An interface to manage and secure APIs. This interface also shows API subscriptions and analytics reports on API usage.
- ▶ **API Gateway**  
A gateway that manages and controls access to APIs. IBM API Connect supports two types of gateway: A DataPower gateway that runs in an appliance and a Microgateway that runs in a Liberty collective.
- ▶ **Cloud Manager**  
An administrator interface that is used for managing servers, including DataPower appliances and Liberty collectives.
- ▶ **Microservices Compute Runtime**  
A run time for running APIs and microservices in Node and Java.

#### 4.7.1 How IBM API Connect works with z/OS Connect EE

IBM API Connect can be used with z/OS Connect EE in the following scenarios:

- ▶ Securing and managing z/OS Connect EE APIs
- ▶ Creating and running APIs

## Securing and managing z/OS Connect EE APIs

In this scenario, a z/OS Connect EE API is available as a proxy in the API gateway so that access to the API can be controlled, as shown in Figure 4-19.

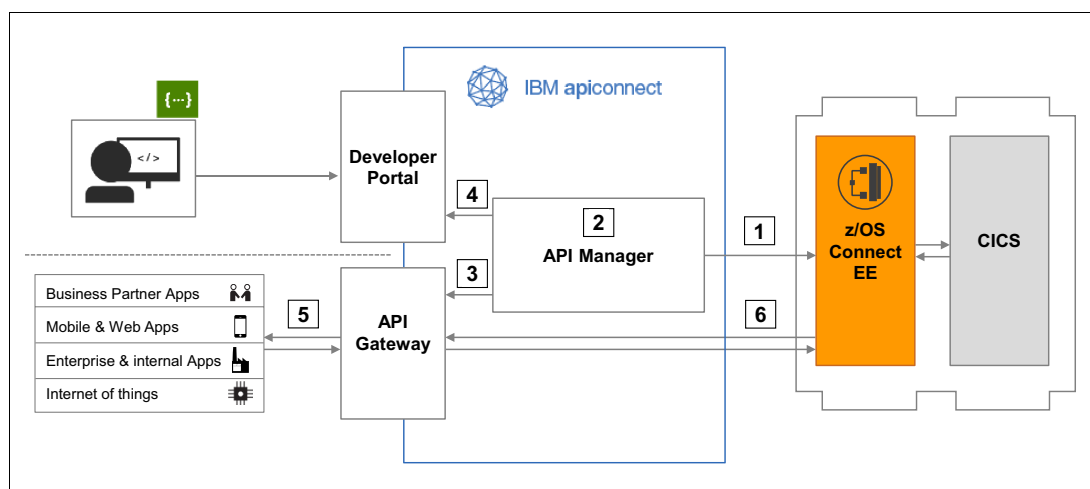


Figure 4-19 Using IBM API Connect to secure and manage APIs

The following steps are shown in Figure 4-19:

1. The API is retrieved by importing the Swagger document. This file is parsed to re-create the operations of the API.
2. An assemble flow is created and a security policy is defined for the API. Optionally, you can add some pre-request and post-request processing; for example, to modify JSON request and response messages.  
  
You include an API in a Plan that is contained in a Product. Application developers access APIs by registering applications to access Plans. You can specify policy settings to limit the use of the APIs that are exposed by the Plan. You can also define a single quota policy that applies to all the API resources that are accessed through the Plan, or separate quota policies for specific API resources.
3. During the publishing process, the API Manager sends the Product configuration to the API Gateway.
4. Also, during the publishing process, the API Manager sends the Product information to the Developer Portal to make it available to communities of application developers.
5. After it is published, the *managed API* can be invoked by authorized applications, including Business Partner applications, mobile and web applications, enterprise applications, and Internet of Things (IoT) devices. The API provides runtime policy enforcement for security, rate limitation, and general governance.
6. The API Gateway invokes the *system API* that is hosted by z/OS Connect EE.

## Creating and running APIs

In this scenario, a LoopBack application is created that aggregates data from different sources, including a z/OS Connect EE API, as shown in Figure 4-20.

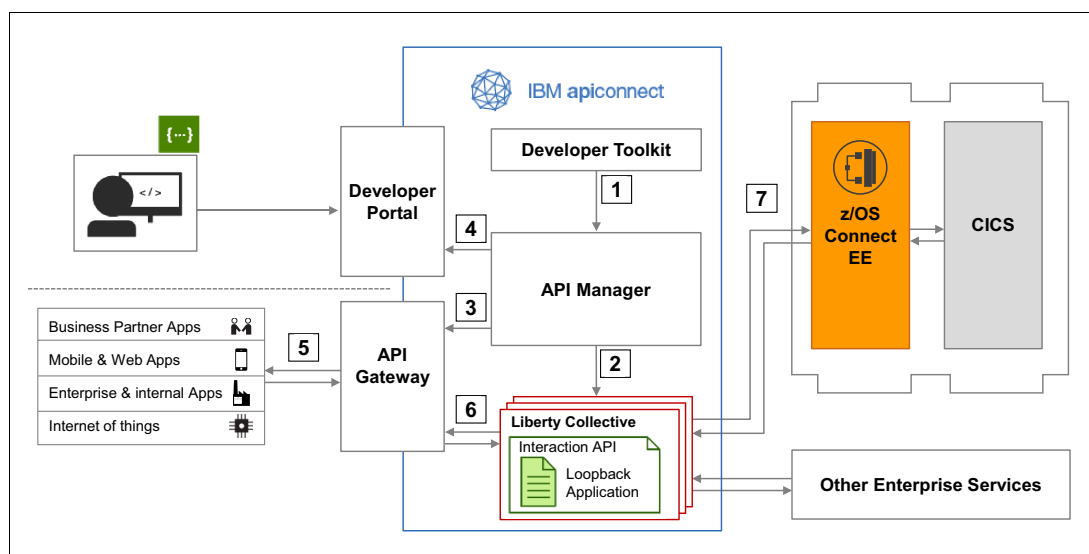


Figure 4-20 Creating and running APIs with IBM API Connect

**Note:** The LoopBack model uses a JavaScript object that represents application data and includes validation rules, data access capabilities, and business logic. LoopBack models provide a REST API by default and connect to data sources for access to back-end data.

The following steps are shown in Figure 4-20:

1. Using the Developer Toolkit, a LoopBack application is created that interacts between the z/OS Connect EE API and other resources; for example, a NoSQL database. An assemble flow is then created for the LoopBack application and a security policy is defined for the API.
2. The application and catalog is defined in the API Manager and when published the API Manager sends the archive to the chosen Liberty Collective for the LoopBack application.
3. During the publishing process, the API Manager sends the Product configuration to the API Gateway.
4. The API Manager also sends the Product information to the Developer Portal during the publishing process to make it available to communities of application developers.
5. After it is published, the managed API can be invoked by authorized applications, including Business Partner applications, mobile and web applications, enterprise applications, and IoT devices. The API provides runtime policy enforcement for security, rate limitation, and general governance.
6. The API Gateway invokes the interaction API.
7. The interaction API invokes the system APIs, including the API that is hosted by z/OS Connect EE.

## When to use IBM API Connect

Consider the use of IBM API Connect in a hybrid integration architecture when you want to perform the following tasks:

- ▶ Extend the value of your mainframe assets by socializing SOAP or REST APIs to developers, and provide controlled access to third parties.
- ▶ Streamline the development of new REST APIs through service discovery.
- ▶ Secure, govern, and monitor access to REST APIs.
- ▶ Extend the value of your mainframe assets by rapidly creating APIs that are based on an enterprise-grade Node.js and LoopBack framework.
- ▶ Augment and enrich mainframe services with other endpoints to aggregate multiple services into a single API.

**Important:** The combination of IBM API Connect and z/OS Connect EE is a powerful solution for simplifying the reuse of mainframe assets by mobile, web, and cloud-based clients.

For more information about IBM API Connect, see the following Developer Center website:

<http://developer.ibm.com/apiconnect/>

## 4.8 IBM DataPower Gateway

IBM DataPower Gateway appliances help quickly secure, integrate, control, and optimize access to various workloads through a single, extensible, DMZ-ready gateway. These appliances act as security and integration gateways for a full range of mobile, cloud, API, web, SOA, and B2B workloads.

The principal roles of a DataPower Gateway are shown in Figure 4-21.

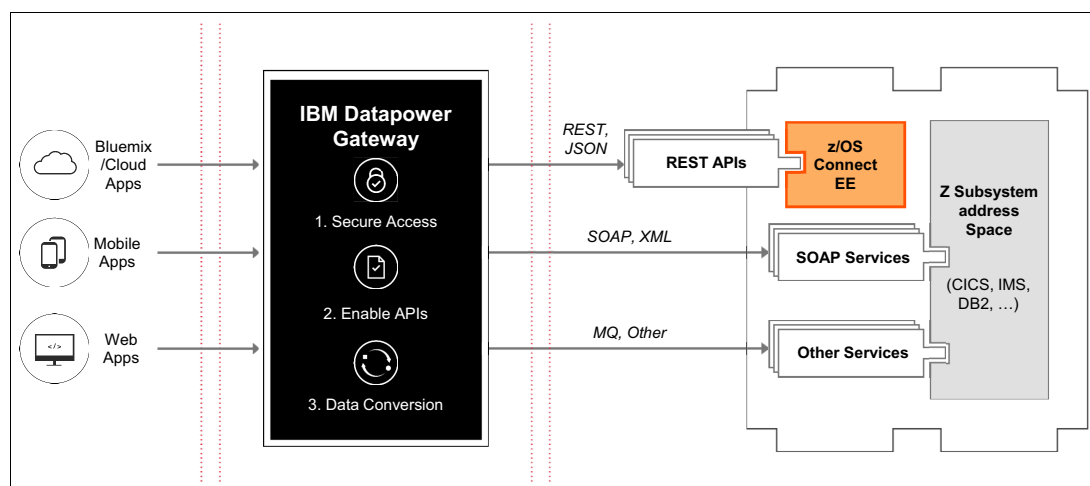


Figure 4-21 IBM DataPower Gateway

IBM DataPower can play the following roles in a hybrid integration architecture:

- ▶ As a security gateway, IBM DataPower can securely make available corporate data, application, and services to cloud applications while optimizing delivery of the workload. It can provide the following security capabilities:
  - Enforcement point for centralized security policies.
  - Authentication, Authorization, SAML, OAuth 2.0, Audit.
  - Threat protection for XML and JSON.
  - Message validation and filtering.
  - Centralized management and monitoring point.
  - Traffic control and rate limiting.
  - Establishment of a secure connection from the enterprise to IBM Bluemix using the IBM Bluemix Secure Gateway service.
- ▶ Data transformation; for example, JSON to XML.

IBM DataPower can also act as a gateway for API access and traffic management. It processes and manages security protocols and stores relevant user and appliance authentication data. The Gateway server also provides assembly functions that enable APIs to integrate with various endpoints, such as databases or HTTP-based endpoints.

If you have an IBM DataPower appliance, you can use this appliance as your gateway for an API management solution. You might want to use physical IBM DataPower appliances for your production Gateway servers. The physical Gateway servers provide improved performance throughput when compared with virtual Gateway servers.

#### 4.8.1 When to use an IBM DataPower Gateway

Consider using the IBM DataPower Gateway in a hybrid integration architecture when you want to perform the following tasks:

- ▶ Protect SoR applications against security attacks.
- ▶ Implement authentication and authorization standards that are not supported natively on the mainframe.
- ▶ Implement a secure API gateway as part an API enablement solution.

For more information about the IBM DataPower Gateway, see the following website:

<http://www.ibm.com/support/knowledgecenter/SS9H2Y>







## Real-world scenarios

In this chapter, we summarize several integration scenarios that integrate cloud, mobile, and digital applications with mainframe systems. For each scenario, we provide the project context, including business drivers and solution goals. We then describe the key decision factors that were used for deciding on the most appropriate mainframe integration solution. We conclude with an outline of the solution architecture.

This chapter includes the following topics:

- ▶ 5.1, “Web service enablement” on page 52
- ▶ 5.2, “Reusing web services” on page 54
- ▶ 5.3, “REST service enablement” on page 57
- ▶ 5.4, “API management” on page 60

## 5.1 Web service enablement

This scenario focuses on the enablement of CICS web services, which are used (indirectly) by mobile and other clients.

### 5.1.1 Introduction

Many banks embarked on a core banking transformation strategy to gain flexibility and reduce cost. A service-oriented architecture (SOA) facilitates reusing assets. After services are created, Systems of Engagement (SoE) application developers can more easily access the core banking services.

Bank A uses a mainframe-based, multi-channel core banking package that provides functions, such as transaction accounts, loans, mortgages, and payments. In the past, these transactions were tightly integrated with other business applications, where the transactions were invoked directly by using connector and messaging technologies.

With new business drivers, such as enabling mobile applications for better engagement with customers and API economy for enabling partners to reach new markets and customers, Bank A wants to use its core banking application to create a set of APIs. The Bank A business team knows that an increased mobile and device application presence enhances their brand image and increases customer satisfaction. The goal is to be seen as an innovative bank that engages with its customers and partners in new ways; for example, through “hackathons”.

The initial step for Bank A is to create a more flexible interface to the CICS core banking applications. The following key questions are to be addressed in this project:

- ▶ Which service enablement architecture should be used (REST or SOAP/XML-based web services)?
- ▶ Which CICS service enablement solution should be implemented?

### 5.1.2 Key decision factors

Bank A considered several factors when deciding how to service enable the CICS core banking application. However, the primary factors were solution maturity and solution adoption. The key decision factors are reviewed next.

#### **Mainframe application interface**

The core banking functions make available a COMMAREA interface that is based on the package proprietary data protocol. With the CICS integration solutions, knowledge of this interface is required by the developers that are developing channel enablement applications.

The new solution should provide tooling for creating services and a way to define and exchange service definitions.

#### **Application integration infrastructure**

The core banking application runs on CICS TS and DB2. The application integration architecture includes an application server that is used for channel applications and an Enterprise Service Bus (ESB), which is used for service composition.

The target solution must be compatible with the existing integration architecture.

## Hybrid integration requirements

Emerging digital applications (such as mobile) that are used by Bank A clients use a REST interface and JSON payload to access core banking functions. Other applications also use a REST interface, but many of the vendor packages that are used by the bank support a web service (SOAP) interface only.

An eventual goal is to develop a services catalog (REST and SOAP) and developer portal that supports Bank A developers and third parties.

## Non-functional requirements

The project features the following main non-functional requirements:

- Minimize risks

The core banking application is a critical application that is used at high volume (peak transaction rate of 500 tps). An essential project requirement is to implement new SoE applications in a non-disruptive way.

The business-critical core banking services must be available always. The infrastructure must support continuous service availability across planned and unplanned outages.

- Optimize performance and scalability

The solution must be optimized and must meet the performance and scalability expectations of Bank A.

- Security

The solution must offer a range of security solutions that are based on open standards and protected against Denial of Service attacks.

Bank A also wants to implement a service enablement solution that has been widely adopted in the banking industry.

### 5.1.3 Solution architecture

Bank A chose to implement a solution that is based on SOAP web services (see Figure 5-1) because CICS web services is a mature solution that is widely deployed in the finance industry.

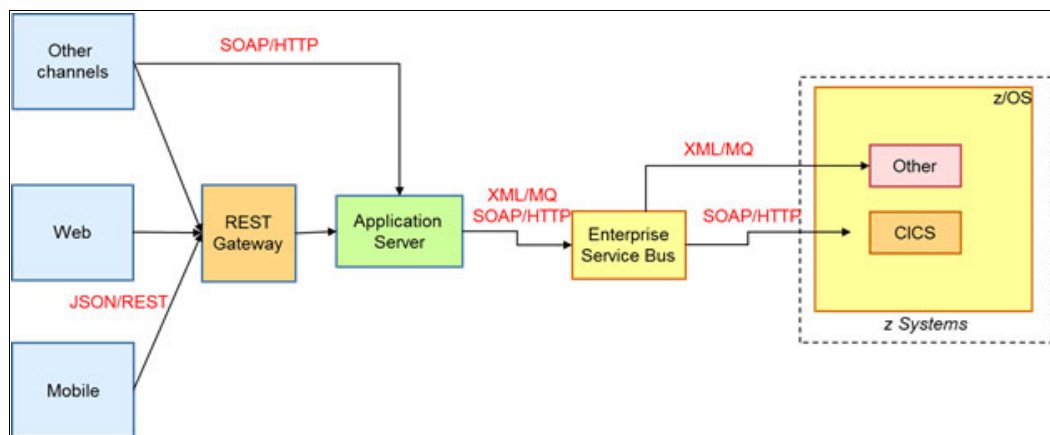


Figure 5-1 Web service enablement

The solution features the following main components:

- ▶ CICS web services

Core banking functions are made available as web services for the following reasons:

- The web services specification offers a mature standard for service definition and a range of other specifications, such as WS-Security.
- The CICS native web services support is a mature solution with a large deployment base.
- CICS web are supported by the customer's current version of CICS TS and require only traditional CICS systems programming skills to implement.

- ▶ Enterprise Service Bus

IBM Integration Bus is used as an ESB for routing and protocol conversion.

- ▶ Application Server

The channel applications are made available as XML-based services.

- ▶ REST Gateway

The home grown REST gateway converts REST and JSON requests from mobile and web clients to XML and IBM MQ or SOAP and HTTP.

## 5.1.4 Next step

Bank A is rolling out the first wave of CICS web services. IBM DataPower has been recently implemented for its additional security capabilities, and a project has started to replace the home-grown REST gateway with IBM API Connect to improve service management and governance.

## 5.2 Reusing web services

This scenario focuses on the RESTful service enablement of a CICS/ DB2 application to improve integration efficiency between the organization's digital channels services and their core customer system on z/OS. In this scenario, the existing connectivity and integration architecture is reused, which is based on IBM MQ, CICS web services, IBM Integration Bus, and DataPower.

### 5.2.1 Introduction

Bank B is a large national retail and commercial bank that offers a comprehensive set of banking, credit, and insurance services. Their core customer system is a CICS/DB2 z/OS application that was largely developed in COBOL. This system is accessed through multiple routes, including IBM MQ, web services, and file transfer.

There are several integration layers between the bank's WebSphere Application Serving mid-tier and the customer system. These layers include IBM Integration Bus, DataPower, and CICS Transaction Gateway. These layers support the traditional branch and call center applications, and the bank's new digital platforms.

To increase agility, improve efficiency, and simplify the layers of integration, Bank B plans to rationalize the number of access points into CICS/DB2 from slightly less than a thousand discrete services today to a few dozen over the next 18 months. To achieve this goal, Bank B intends to establish a set of REST APIs through which all core services can be invoked.

The bank is interested in the use of JSON payloads because the payloads avoid some of the more rigorous data coupling characteristics that are associated with more strongly typed data payloads, such as XML. For example, if a field is added to an XML payload by a service provider, all consumers must then adhere to the news scheme. By contrast, the equivalent service provider change to a JSON payload is ignored by the applications. This decoupling of data payloads is attractive to Bank B.

Bank B uses several technologies in production, which satisfy their requirement to make the customer system available as a set of REST APIs. The question to be addressed by this project is: Of the following options, which is the most appropriate to REST-enable their CICS/DB2 customer system:

- ▶ IBM Integration Bus
- ▶ IBM DataPower Gateway
- ▶ z/OS Connect Enterprise Edition

## 5.2.2 Key decision factors

Bank B considered several factors when deciding how to REST-enable the customer system; however, their primary motivations were to reduce the number of interfaces and to exploit REST API interfaces with JSON data payloads. The key decision factors are reviewed next.

### Mainframe application interface

Today, the customer system is made available through multiple connectivity options, including IBM MQ, CICS web services, and CICS Transaction Gateway. The decision for which connectivity option is adopted is largely dependent on which channel or business silo is undertaking the development.

The new solution must enable the customer system to be invoked in a standard way, regardless of the application language, location (on-premises or cloud-based application), or data model employed by the calling application.

### Application integration infrastructure

The customer system is based on CICS TS and DB2 z/OS. The target solution must be flexible enough to support multiple software versions; however, Bank B wants to use new functions as early as possible as they move through their upgrade cycle.

The application integration architecture uses IBM MQ, IBM Integration Bus, CICS Transaction Gateway, and IBM DataPower Gateway appliances. The target solution should reuse (where possible) the investments that have been made in technology and architecture designs.

### Hybrid integration requirements

Bank B services an extensive digital platform of mobile and web apps (cloud will be used in the future). The strategy for the digital platform is to move entirely to REST and JSON services for synchronous calls, which will be achieved by using outward-facing DataPower security gateways and mid-tier WebSphere Application Servers. The mid-tier invokes the IBM Integration Bus using IBM MQ. Finally, IBM Integration Bus makes one or more calls to the customer system, aggregating the response for the originating WebSphere Application System application.

The digital platform continues to access the customer system in a similar manner by using a subset of the interfaces and (where possible) with a shorter path length to the underlying CICS subsystem.

## Non-functional requirements

The primary non-functional requirements in this project are all related in some way to simplification, and are in the following categories:

- Simplification through fewer interfaces

A critical objective is to reduce the thousand interfaces down to a few dozen. This goal is achieved in part by retiring duplicated interfaces and non-shared interfaces across multiple business silos.

- Performance and Scale

Today, multiple connectivity and integration layers are in place to link the Bank's digital platforms to the customer systems, which result in high latency and poor performance for mobile users.

- Security and Management

The Bank's stringent security requirements mandate that any changes to integration designs must not reduce the security protocols that are associated with access to the customer system. A by-product of the reduced number of interfaces and simplification of integration layers is that the management and controls that are necessary to secure the customer system are significantly reduced. Similarly, this issue is true for fault identification and resolution of the end-to-end service for mobile-to-customer system.

## 5.2.3 Solution architecture

Bank B chose to adopt a solution that is based on reusing the DataPower appliances to provide a RESTful API interface across all channel application teams within the Bank. Acting as a gateway between the WebSphere Application Server mid-tier and the mainframe, the DataPower appliances convert the REST APIs and invoke the customer system by using the CICS web service interface. This configuration was considered to be the fastest way to make REST APIs available and meet the non-functional requirements. The solution is shown in Figure 5-2.

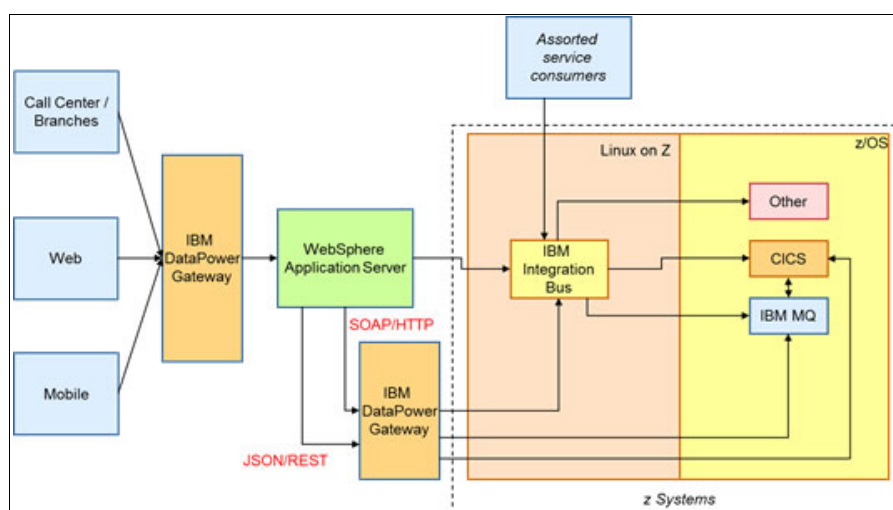


Figure 5-2 Reusing web services

The solution features the following components:

- ▶ **IBM DataPower Gateway**

In this scenario, DataPower is positioned in two places: At the edge of the network acting as a security appliance, and internally within the network acting as a security enforcement point and integration appliance. It is the latter appliance that is relevant here because it performs the following functions:

- Data conversion between the inbound JSON to XML and SOAP payloads
- Request mapping from the inbound REST call to the appropriate CICS web service

- ▶ **WebSphere Application Server**

WebSphere Application Server provides Java hosting for presentation and business logic that is tailored for each in-bound channel. It accesses data from the customer system through DataPower and IBM Integration Bus.

- ▶ **IBM Integration Bus**

Where appropriate for the calling service, IBM Integration Bus provides extra levels of service aggregation and orchestration. For the purposes of this project, direct access to single, CICS based services was largely sufficient to satisfy the requirements of each of the channels platform team.

## 5.2.4 Next step

Bank B is creating the REST and JSON service mappings to the CICS web services in the customer system in DataPower. This mapping gives them the speed and flexibility to establish a foundation that the Bank can settle on as their reduced set of core services. In parallel to this decision, the customer system team decided to further explore the use of z/OS Connect EE as a mechanism to publicize and directly REST-enable the CICS application. They see RESTful APIs as a necessary interface for any major system within the Bank.

## 5.3 REST service enablement

This scenario focuses on the REST service enablement of IMS applications using z/OS Connect EE to support emerging mobile and digital channels.

### 5.3.1 Introduction

Bank C is a mid-sized retail bank that offers financial services to individuals and corporate clients. Their core banking systems are home-grown applications that were developed in COBOL and Java and hosted mostly in IMS, DB2 z/OS, and WebSphere Application Server z/OS. In addition, there are other applications that are hosted within CICS environments.

All of these banking systems are accessed using IBM MQ, which provides a reliable, request and response messaging model. This access method supports all channels across the Bank, including branch, call center, internet and (more recently) their smartphone and tablet banking apps.

With the increasing need to respond quickly to regulatory changes, coupled with the expectation from lines of business that new offerings be delivered in weeks not months, there is great interest in the promise of the API economy to provide this level of agility.

It is hoped that an approach that is based on REST service enablement will have the following advantages:

- ▶ Self-service and discovery of services
- ▶ A standardized interface for mainframe applications
- ▶ JSON payloads for flexible data representation

Longer term, Bank C wants to develop an enterprise-wide API framework to make core services and data available internally and with partner organizations. They want to unburden the development and partner communities from the need to use custom protocols and the need for awareness of the underlying application implementations.

The initial step for Bank C is to determine how best to surface their IMS core banking systems through a REST interface. This project addressed the key question of which REST service enablement technology should be adopted: IBM DataPower Gateway or z/OS Connect EE?

### 5.3.2 Key decision factors

Bank C considered several factors when deciding how to REST-enable the IMS core banking application. The primary factor was to make available a common solution for all z subsystems that enable them to participate in an API framework. The key decision factors are reviewed next.

#### Mainframe application interface

The core banking functions are made available through a set of IBM MQ queues. Developers that are creating applications must know the COBOL COPYBOOK data structure and details of the target IBM MQ queues.

The new solution should enable the service to be invoked as a REST API and, where relevant, allow the developer to discover all of the necessary details of the service to successfully invoke the IMS application.

#### Application integration infrastructure

The core banking application is based on IMS TS and DB2 z/OS. The target solution must offer the flexibility of supporting current and future versions of IMS and DB2.

The application integration architecture uses IBM MQ to provide a reliable and scalable pseudo-synchronous messaging service. However, the target solution does not need to be reliant on the messaging backbone.

#### Hybrid integration requirements

Today, Bank C's web, mobile, and tablet banking apps use a REST/JSON to access the Bank's core systems. All requests are directed to a set of dedicated, channel-specific WebSphere Application Servers, which perform a presentation-layer function before invoking the IMS core banking system using IBM MQ.

In future, the presentation tier will be hosted in a public cloud environment and will access enterprise-wide services through an API framework. Application developers have the option of using the new REST APIs or reusing the IBM MQ backbone.

#### Non-functional requirements

The following non-functional requirements must be met:

- ▶ Improve agility

Bank C is adopting REST-based JSON APIs to deliver services more quickly. They see this requirement as a way to enable developers and partners to self-discover and use



services. Moreover, adopting the use of JSON payloads enables a de-coupling of the data structures between requester and provider applications.

- ▶ Adopt a common integration architecture

As the Bank proceeds with its enterprise-wide API framework, they want to use a standardized set of APIs, regardless of the underlying service provider. As such, they require mainframe services to be used through the same REST API model. In addition, there is a longer-term aspiration to surface all mainframe applications through a common API mechanism to reuse tools and skills.

- ▶ Provide security

A by-product of offering discoverable services is that, if left uncontrolled, any individual with access to the IP address can identify and invoke these services. A key non-functional requirement is to control which groups and individuals have access to different sets of services. This control includes the type of access; for example, some individuals might have discover-only access versus other users who can invoke the service.

- ▶ Offer performance and scale

The solution must offer a low-latency, high-throughput connectivity service to mainframe applications. At no time should the solution introduce bottlenecks or delays, particularly during the peak banking hours.

### 5.3.3 Solution architecture

Bank C chose to implement a solution that is based on z/OS Connect EE (see Figure 5-3) because this solution provides a unified REST/JSON interface.

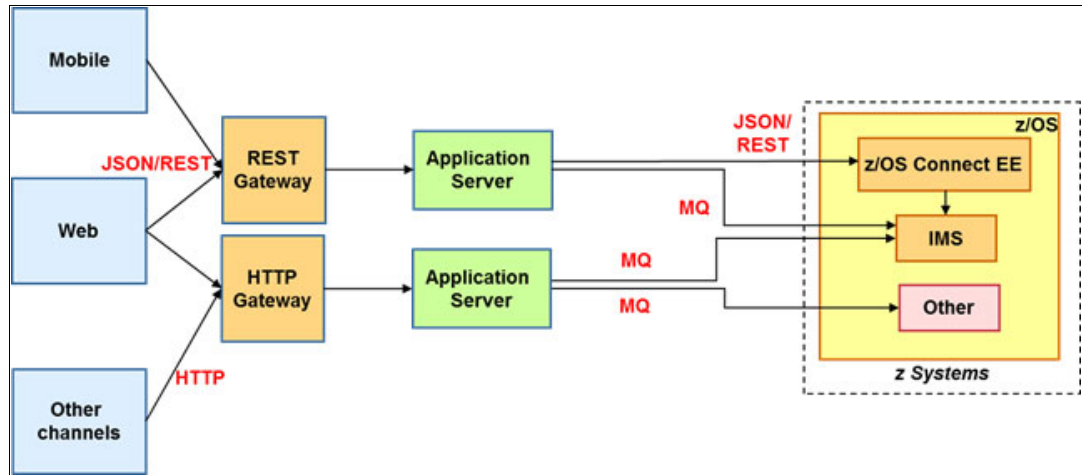


Figure 5-3 Enabling REST

The solution features the following major components:

- ▶ z/OS Connect EE

Core banking services are made available through z/OS Connect EE for the following reasons:

- Application developers can discover services.
- It provides a common solution for IMS today and in future, the WebSphere Application Server for z/OS, DB2 z/OS, and CICS subsystems.

- Tooling for data transformation is available with the IMS Explorer.
- Security capabilities including authentication, authorization, and audit are supported.
- ▶ Application server
 

The channel applications are made available as IBM MQ and REST-based services.
- ▶ HTTP and REST Gateways
 

The in-bound HTTP and REST gateways act as channel aggregation points for each of the different channels that are entering the Bank's network.

### 5.3.4 Next step

The first REST services are being rolled out in a pilot phase. This process fulfills balance-enquiry and amendment functions from the IMS subsystem. Soon, this rollout will be extended to include other capabilities that support account transfers and payments. Longer-term, REST APIs will be made available and used as part of their enterprise-wide API Management framework to support their aspirations as a “cloud-first” enterprise.

## 5.4 API management

This scenario focuses on establishing a managed API framework that allows internal and external developers to discover and securely consume business services that are available across the organization.

### 5.4.1 Introduction

Bank D is a boutique retail bank that offers a custom suite of retail and business banking and credit card services. Their modern core banking systems are based on CICS/DB2 applications that were developed in COBOL and Java. There are multiple access points to these systems; for example, the use of IBM MQ and web services. However, the predominant access method for new applications is by using REST services.

In recent years, the Bank made significant investment in its digital channels and now supports new web and mobile application services. Bank D placed customer service as its top priority to encourage growth and invested in new tablet applications for staff. These mobile services need to access the core banking systems, a Master Data Management (MDM) system, and a business rules engine.

An enhanced mobile application is planned that will enable the bank's small business advisors to offer tailored financial products to customers in real time. The mobile app will allow an advisor to meet with clients at their place of business, retrieve a customer's profile and financial data, and be advised on what financial products to offer the client.

Bank D is also under pressure to respond to regulatory change and new business requirements. Under a new regulatory directive, the Bank will need to offer secure access to its payments systems to third-party payments providers.

Based on the needs to deliver new mobile services quickly and to comply with new regulations, Bank D wants to enable a set of enterprise-wide APIs. The intention is to improve agility and speed to market of new financial products by empowering the Bank's own development community. The Bank also wants to offer a subset of APIs to their partner community to extend into a wider array of markets.

The key decision to address in this project is how to enable an enterprise-wide API framework that supports the following functions:

- ▶ Developing and creating APIs
- ▶ Self-discovery of APIs
- ▶ Robust security and traffic shaping
- ▶ Reusing internal services
- ▶ Visibility and monitoring of APIs

## 5.4.2 Key decision factors

Bank D considered several factors when deciding how to deploy and manage APIs. The primary factors were to simplify the reuse of mainframe applications and data from internal and external clients, and the governance (access control, management, and monitoring) of the published APIs.

### Mainframe application interface

Currently, Bank D uses many different interfaces into z/OS based applications. The requirement is to define a set of reusable APIs to speed up mobile app development, improve integration with cloud-based applications, and interoperability with third parties.

### Application integration infrastructure

The core banking systems, including the Accounts and Customer applications, run on CICS TS and DB2. IBM's Master Data Management (MDM) solution provides a comprehensive and searchable view of customer data. Operational Decision Management (ODM) is used for processing business rules (for example, to determine the level of risk in extending a loan at a specific interest rate).

Most core systems are service-enabled as SOAP or REST services. In some cases, such as the Cards Management System (CMS), applications are accessed as REST services through a Java JAX-RS layer that is deployed in a WebSphere Liberty server.

The IBM DataPower Gateway is used today as a security gateway; for example, to protect against denial of service attacks and to authenticate client requests.

### Hybrid integration requirements

The new mobile app is a native iOS app that is provided to bank advisors on company-owned iPads. The app must display real-time client financial data, compare the financial information against other similar businesses that are in the same area, and allow the bank advisor to apply for a line of credit while at the client's workplace.

The app must securely integrate with several mainframe applications through a set of APIs, as shown in the following examples:

- ▶ `listClients`: Retrieve information about clients based on a set of filters, including type of business and geographic location.
- ▶ `viewClientProfile`: View the client profile, including address, accounts, credit cards, out-standing loans, and cash flow.
- ▶ `listTransactions`: List the transaction history of the client.
- ▶ `getCreditRating`: Get a credit rating.

## Non-functional requirements

The following main non-functional requirements must be met:

- ▶ Speed of deployment

It is imperative that the mobile solution is deployed quickly to gain a competitive advantage over other banks that offer similar small business financial services.
- ▶ Security

All personal and financial data must be stored on the mainframe and encrypted in transit.
- ▶ Always available

As the new primary way to engage small businesses, the new mobile solution must be available 24/7.
- ▶ API lifecycle

API creation, deployment, and discovery must support versioning and be simple to implement.
- ▶ Management and monitoring

The solution must provide the operational metrics and analytics capability to be able to monitor API usage and manage the traffic demand when APIs are experiencing peak loads.

### 5.4.3 Solution architecture

Bank D chose to adopt a solution that is based on IBM API Connect and z/OS Connect EE. This solution supports the set of REST APIs that are used by the bank advisor mobile app and will be used in the future for hybrid cloud and partner integration. The solution that is shown in Figure 5-4 is an example set of deployed APIs.

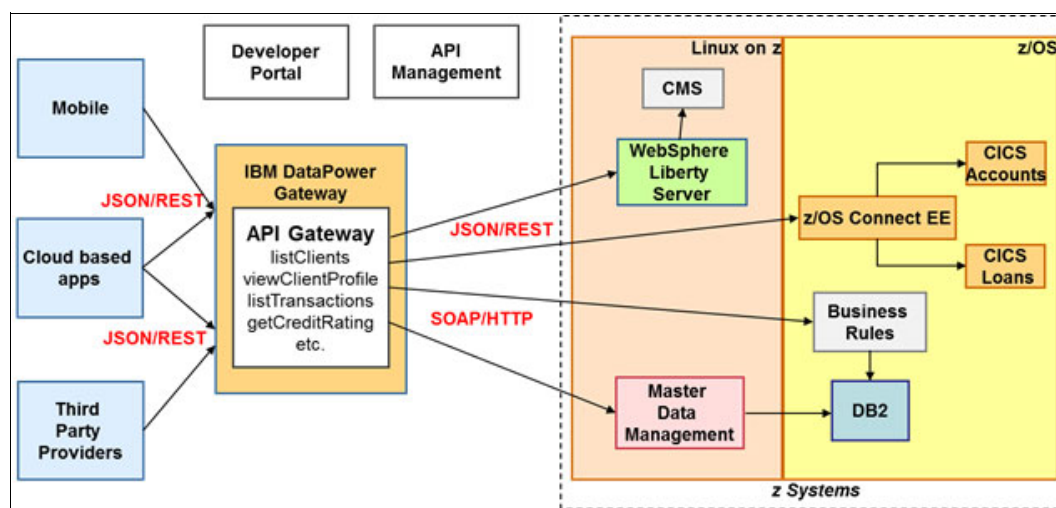


Figure 5-4 API enablement of core systems

The solution features the following major components:

- ▶ **z/OS Connect EE**  
z/OS Connect EE provides a REST interface for the CICS Accounts and Loans applications (used by the viewClientProfile and listTransactions APIs).
- ▶ **Business Rules**  
IBM ODM provides a SOAP interface that is used by the getCreditRating API to calculate a risk score for a client.
- ▶ **Master Data Management**  
IBM MDM contains customer profile information that is used by the listClients, viewClientProfile, and listTransactions APIs.
- ▶ **WebSphere Liberty**  
A WebSphere Liberty server provides a REST interface to a Cards Management System (CMS) that is used by the viewClientProfile API.
- ▶ **IBM DataPower Gateway**  
IBM DataPower Gateway is used as a security gateway and provides a secure and highly available run time for the deployed APIs.
- ▶ **IBM API Connect**  
IBM API Connect is used to manage the API lifecycle to create API versions and to revert to previous versions when required. A developer portal is used to communicate information about the APIs, such as the API's interface, API documentation, and code samples. This information allows the developers to test and try the APIs. The API Manager is used for API deployment and management, and the IBM DataPower is used as an API Gateway.

#### 5.4.4 Next step

Bank D is establishing an API framework that will be called from mobile apps, including the bank advisor app.

There is recognition in the Bank that the API framework can be easily extended to support up-coming regulations that require the Bank to provide more open access to certain services; for example, to allow third parties to initiate payments.





## Summary

Today, progressive companies are exposing their mainframe applications as REST APIs. But what is the best way to do REST with the mainframe? The options to help you to make a more informed choice were compared and contrasted in this Redpaper publication.

In this chapter, we summarize the different integration architectures and solutions, and provide high-level recommendations for when to use each one.

## 6.1 Integration architectures

The mainframe supports several integration architectures that can be used in hybrid integration projects. These integration architectures are compared in Table 6-1.

Table 6-1 Common integration architectures

Integration architecture	Description	Recommendation
APIs and API management	Architecture for creating, assembling, managing, securing, and socializing web application programming interfaces (APIs).	Use when: <ul style="list-style-type: none"><li>▶ Business functions must be discoverable</li><li>▶ Enterprise applications must be extended to a system of developers and new markets</li><li>▶ Need high degree of operational governance</li></ul>
Web services	Service-oriented architecture that is based on standards, such as SOAP, XML, and WS specifications.	Use when: <ul style="list-style-type: none"><li>▶ XML is the predominant data payload</li><li>▶ WS standards are required</li><li>▶ Reusing an existing SOA infrastructure</li></ul>
REST	Resource-oriented architecture that is based on HTTP URL and verbs and JSON. De-facto standard for SoE applications, such as mobile and cloud applications.	Use when: <ul style="list-style-type: none"><li>▶ JSON is the primary data payload</li><li>▶ Intuitive and simple interface for developers is required</li></ul>
Messaging	Asynchronous transport mechanism.	Use when: <ul style="list-style-type: none"><li>▶ Assured delivery is required</li><li>▶ Enabling publish/subscribe applications</li><li>▶ Reusing a messaging infrastructure</li></ul>



## 6.2 Integration solutions

Different IBM integration solutions can be used in hybrid integration projects with the mainframe. The main solutions are compared in Table 6-2.

Table 6-2 IBM integration solutions

Integration solution	Recommendation	Description
z/OS Connect Enterprise Edition	Provides a REST API Gateway on z/OS to invoke CICS, IMS, DB2, IBM MQ, and other applications natively. Includes tooling for creating and deploying APIs.	Use when: <ul style="list-style-type: none"><li>▶ Require workstation-based tooling for creating RESTful APIs that are based on z/OS assets</li><li>▶ Enabling discovery of defined APIs based on OpenAPI (Swagger 2.0) standard</li><li>▶ Common REST JSON enablement solution is required for different z/OS based applications and subsystems</li></ul>
IBM API Connect	API management solution for creating, running, securing, and managing APIs.	Use when: <ul style="list-style-type: none"><li>▶ Extending the value of mainframe assets by socializing APIs to various developers and partners</li><li>▶ Full API lifecycle support is required</li><li>▶ Controlled access and strong governance is required</li></ul>
IBM DataPower Gateway	SOA and API security gateway.	Use when: <ul style="list-style-type: none"><li>▶ Securing hybrid integration access to the main-frame</li><li>▶ Deploying APIs to a secure and efficient API Gateway</li></ul>
IBM Integration Bus	Comprehensive integration services with support for any-to-any transformation.	Use when: <ul style="list-style-type: none"><li>▶ Have complex integration requirements; for example, service orchestration</li><li>▶ Diverse data and protocol formats are used</li><li>▶ Industry standard message formats must be supported</li></ul>
IBM MQ	Asynchronous message transport for reliable delivery of messages.	Use when: <ul style="list-style-type: none"><li>▶ Enabling bidirectional messaging connectivity</li><li>▶ Publish/subscribe pattern is required</li></ul>



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *CICS and SOA: Architecture and Integration Choices*, SG24-5466
- ▶ *IBM CICS and the JVM server: Developing and Deploying Java Applications*, SG24-8038
- ▶ *Implementing IBM CICS JSON Web Services for Mobile Applications*, SG24-8161
- ▶ *IMS Integration and Connectivity Across the Enterprise*, SG24-8174
- ▶ *The Power of the API Economy: Stimulate Innovation, Increase Productivity, Develop New Channels, and Reach New Markets*, REDP-5096
- ▶ *An Architectural and Practical Guide to IBM Hybrid Integration Platform*, SG24-8351

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Online resources

These websites are also relevant as further information sources:

- ▶ Digital reinvention: Preparing for a very different tomorrow. 2013:  
<http://www.ibm.com/services/us/gbs/thoughtleadership/digitalreinvention/>
- ▶ Swagger:  
<http://swagger.io/specification>
- ▶ z/OS Connect EE:  
<https://ibm.biz/zosconnectdc>
- ▶ CICS and Java:  
<https://developer.ibm.com/cics/>
- ▶ DB2 REST Services:  
<https://ibm.biz/zos-connect-db2-rest-services>
- ▶ IBM MQ:  
<https://ibm.biz/BdsGHu>
- ▶ IBM Integration Bus:  
<http://www.ibm.com/software/products/en/ibm-integration-bus>

- ▶ IBM API Connect:  
<http://developer.ibm.com/apiconnect/>
- ▶ IBM DataPower Gateway:  
<http://www.ibm.com/support/knowledgecenter/SS9H2Y>

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)





REDP-5319-01

ISBN 0738455903

Printed in U.S.A.

Get connected

