IBM Z Integration Guide for Hybrid Cloud and the API Economy

Richard Gamblin
Rob Jones
Nigel Williams
**Note:** Before using this information and the product it supports, read the information in “Notices” on page ix.

---

**Third Edition (July 2018)**

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

<table>
<thead>
<tr>
<th>Notices</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trademarks</td>
<td>x</td>
</tr>
<tr>
<td>Preface</td>
<td>xi</td>
</tr>
<tr>
<td>Authors</td>
<td>xi</td>
</tr>
<tr>
<td>Now you can become a published author, too!</td>
<td>xii</td>
</tr>
<tr>
<td>Comments welcome</td>
<td>xii</td>
</tr>
<tr>
<td>Stay connected to IBM Redbooks</td>
<td>xiii</td>
</tr>
<tr>
<td><strong>Summary of changes</strong></td>
<td>xv</td>
</tr>
<tr>
<td>July 2018, Third Edition</td>
<td>xv</td>
</tr>
<tr>
<td>February 2017, Second Edition</td>
<td>xv</td>
</tr>
<tr>
<td><strong>Chapter 1. Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Evolution of integration with core systems</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Business imperatives</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Multi-speed IT</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Mainframe connectivity landscape</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Hybrid cloud connectivity challenges</td>
<td>5</td>
</tr>
<tr>
<td>1.6 The API economy</td>
<td>5</td>
</tr>
<tr>
<td><strong>Chapter 2. Architecture options for service and API enablement</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 APIs</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1 Services and APIs</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2 Microservices and microservices applications</td>
<td>10</td>
</tr>
<tr>
<td>2.1.3 API management and the OpenAPI Initiative</td>
<td>11</td>
</tr>
<tr>
<td>2.1.4 Security standards for APIs</td>
<td>11</td>
</tr>
<tr>
<td>2.1.5 Advantages of APIs</td>
<td>12</td>
</tr>
<tr>
<td>2.2 REST and JSON</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 REST</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 JSON</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 Advantages of REST and JSON.</td>
<td>13</td>
</tr>
<tr>
<td>2.3 SOAP web services</td>
<td>13</td>
</tr>
<tr>
<td>2.3.1 Advantages of SOAP web services</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Messaging and events</td>
<td>15</td>
</tr>
<tr>
<td>2.4.1 Advantages of messaging</td>
<td>16</td>
</tr>
<tr>
<td><strong>Chapter 3. Hybrid integration architecture considerations</strong></td>
<td>17</td>
</tr>
<tr>
<td>3.1 Hybrid integration reference architecture with IBM Z</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Systems of Record</td>
<td>19</td>
</tr>
<tr>
<td>3.3 Systems of Engagement</td>
<td>19</td>
</tr>
<tr>
<td>3.4 Access layer</td>
<td>20</td>
</tr>
<tr>
<td>3.5 Integration layer</td>
<td>21</td>
</tr>
<tr>
<td>3.6 API gateway</td>
<td>22</td>
</tr>
<tr>
<td>3.7 Exposure gateway</td>
<td>23</td>
</tr>
<tr>
<td>3.8 Cloud affinity</td>
<td>23</td>
</tr>
<tr>
<td>3.9 Consumers of APIs</td>
<td>24</td>
</tr>
<tr>
<td><strong>Chapter 4. IBM integration solutions</strong></td>
<td>25</td>
</tr>
</tbody>
</table>
Chapter 6.  Summary .......................... 69
6.1  Integration architectures .......................... 70
6.2  Integration solutions .......................... 71

Related publications .......................... 73
IBM Redbooks .......................... 73
Online resources .......................... 73
Help from IBM .......................... 74
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](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.

<table>
<thead>
<tr>
<th>CICS®</th>
<th>IBM API Connect®</th>
<th>Rational®</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS Explorer®</td>
<td>IBM Cloud™</td>
<td>Redbooks®</td>
</tr>
<tr>
<td>CICSPlex®</td>
<td>IBM Watson®</td>
<td>Redpaper™</td>
</tr>
<tr>
<td>DataPower®</td>
<td>IBM Z®</td>
<td>Redbooks (logo)®</td>
</tr>
<tr>
<td>Db2®</td>
<td>IMS™</td>
<td>Watson™</td>
</tr>
<tr>
<td>DB2®</td>
<td>MQSeries®</td>
<td>WebSphere®</td>
</tr>
<tr>
<td>IBM®</td>
<td>RACF®</td>
<td>z/OS®</td>
</tr>
</tbody>
</table>

The following terms are trademarks of other companies:

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

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

LoopBack, and StrongOps are trademarks or registered 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 has paved the way for a huge growth in cloud and microservices applications, such as those hosted in IBM® Cloud. One of the most widely adopted 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 in on-premises services whilst taking advantage of the benefits of public cloud services. Over 90% of new client-facing apps rely on application logic and data services from mainframes, meaning IBM Z® is a critical service provider for cloud applications; for example, through REST APIs. 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 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 application integration architectures and solutions and make the best choice based on the specific requirements of the project.

This paper 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 the real world.

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 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.

Rob Jones is a Senior Technical Staff Member, working at the IBM Hursley Software Laboratory, in the UK, as the Chief Architect and Strategist for IBM z/OS® Connect Enterprise Edition. He has worked on the development of IBM CICS® Transaction Gateway products since 2002, specializing in the IBM z/OS platform. He also produced the first CICS TG plug-in for IBM CICS Explorer® tool.

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

Mark Cocker
Paul Cooper
Matthew Leming
Andy Lyell
Anthony Papgeorgiou
Ben Thompson
Phil Wakelin
IBM UK

Kyle Charlet
Haley Fung
IBM US

Aymeric Affouard
Eric Phan
IBM France

Frank van der Wal
IBM Netherlands

Special thanks to Andy Lyell for the graphics included in this paper, and to Matthew Lesher for the editing of the paper.

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

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
> Send your comments in an email to:
  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:
- 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 Integration Guide for Hybrid Cloud and the API Economy
as created or updated on July 27, 2018.

July 2018, Third Edition

This revision includes the following new and changed information.

New information
► IBM App Connect Enterprise
► New real-world scenarios for mainframe API enablement

Changed information
► z/OS Connect EE Version 3
► IBM MQ messaging REST API
► Support for creating APIs with z/OS Connect EE and the MQ service provider

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
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 business 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 are driving agility, enabling businesses to realize 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. APIs are being widely adopted, owing to the ease with which developers can discover and intuitively consume them.

Cloud platforms are a key enabler for these emerging digital services, incorporating public, hybrid, and private cloud platforms. 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 Cloud™. 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. The ability to use Z 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 business logic 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 applications and data.
This chapter includes the following topics:

- 1.1, “Evolution of integration with core systems” on page 2
- 1.2, “Business imperatives” on page 3
- 1.3, “Multi-speed IT” on page 4
- 1.4, “Mainframe connectivity landscape” on page 4
- 1.5, “Hybrid cloud connectivity challenges” on page 5
- 1.6, “The API economy” 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.

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

<table>
<thead>
<tr>
<th>Stage</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green screen</td>
<td>A restricted and controlled user community with a limited and controlled UI.</td>
</tr>
<tr>
<td>Client/Server</td>
<td>The user community remains limited; however, the UI is increasingly richer and developed independently.</td>
</tr>
<tr>
<td>World Wide Web</td>
<td>A global user community (anyone with a browser), and the UI is determined by a web developer and is dependent on browser capabilities.</td>
</tr>
</tbody>
</table>
As different kinds of users access these systems, the following terms emerged to distinguish between the domains and roles of different environments:

- **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. Traditionally, SoRs are hosted on-premises and are typified by IBM Z mainframes; however, as cloud applications mature, they themselves will become an integral part of the SoR. SoRs are often accessed through a System of Engagement.

- **System of Engagement**

  Systems of Engagement (SoE) are those systems through which people or devices 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, typically surfaced through digital channels, are increasingly accessed by mobile apps 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. The emergence of Continuous Integration/Continuous Delivery—or DevOps—practices has enabled these seamless hand-offs between systems. The focus of this paper is on the connectivity and integration aspects of the delivery mechanism.

1.4 Mainframe connectivity landscape

There are multiple options 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:

- Message-based connectivity, such as IBM MQ
- File-based connectivity, such as IBM Connect:Direct®
- Standards-based connectivity, such as 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:

- Middleware-based technologies, such as IBM App Connect Enterprise (formerly 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 had 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 public 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 to an organization, and they require ways to independently discover and use enterprise services.
- **Security:** For 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 most common integration mechanism today for cloud applications to communicate with enterprise services is the use of APIs, giving rise to a new business model that is built on an API economy.

1.6 The API economy

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. Your API 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 businesses 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 that 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 with products, such as IBM CICS, IBM Information Management System (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 if integration options are aligned with other API providers. Similarly, mainframe applications can be enriched with calls to external APIs in order to take advantage of services such as public databases, credit checks, and risk scores.

The remainder of this paper examines the architectural approaches and technologies that are available that address these hybrid cloud integration challenges.
Chapter 2. Architecture options for service and API enablement

This chapter describes the main integration architectures that can be used for service and API enablement of IBM Z applications.

This chapter includes the following topics:

- 2.1, “APIs” on page 8
- 2.2, “REST and JSON” on page 12
- 2.3, “SOAP web services” on page 13
- 2.4, “Messaging and events” on page 15
2.1 APIs

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 paper 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 and are language agnostic, 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 IT practices matured, these functions have evolved to become discreet, consumable entities that are capable of delivering a valuable service in their own right.

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.

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).

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 Services and APIs

A modern enterprise is likely to have a rich catalog of services, developed under SOA initiatives and used by existing applications, within and outside of an organization. New projects choosing to adopt APIs might use existing services where appropriate or create new services as part of the overall API enablement. Services that are used by a new API project might also be used independently, shared by other API enablement projects, or remain entirely private to one API.

APIs can be used to provide an 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 13). 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.

Today, the ubiquity of HTTP has made it the de-facto communications protocol of choice for devices, ranging from industrial sensors to private secure links between servers within an enterprise. Support for connectivity through HTTP is simply assumed by application developers today. When proven within an organization or technical community, “best-of-breed” APIs rapidly become elemental components that are re-used time and time again by application developers, throughout suites of application suites. This in itself can lead to real value, simply in terms of a consistent user experience for the application user or adoption of standard practices.

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 12).
- A SOAP API is a web service that is made available as an API.
- The HTTP protocol provides verbs such as GET, POST, PUT, and DELETE that are typically overlooked in an SOA-based solution. A naturally-RESTful API uses the HTTP verbs to imply semantic meaning, leading to a separation between a given resource (for example, a single record) and the logical operation (for example, update) upon that resource instance. This separation between operation and resource encourages a desirable degree of freedom between the two in the resulting interface and typically produces an API that is intuitive for consumers.
Figure 2-1 shows the co-existence of one solution that is based upon an API architecture alongside an SOA-based solution, where a mixture of new or existing services to the enterprise application might be used to access the System of Record (SoR).

![Figure 2-1 Concurrent consumption models for SoR assets as Services and APIs](image)

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 Microservices and microservices applications

A **microservice** is a bounded entity with a well-defined interface that provides a functional capability for application developers (but not an externalized service in its own right). A microservice might optionally depend on other microservices or directly accessible resources.

A microservice application is a discrete entity with a well-defined interface that encapsulates a specific set of externalized business functions. A microservice application is composed from one or more underlying microservices, where the application run time platform, access method, security model, or programming language involved is not necessarily prescribed, as illustrated in Figure 2-2.

![Figure 2-2 Combining a microservices architecture with APIs](image)

A microservice application can be designed, developed, discovered and externalized as a an API. Similarly, (internal) microservices might be designed, developed, discovered and used as an API. While this might be an aesthetically pleasing employment of architectural symmetry, point-to-point messaging is the more prevalent choice within a microservices application.

Themed collections of APIs and services focused around a given business function, or resource type, might form key components of an overall solution based upon a microservices architecture. Where APIs and services are used to directly encapsulate SoR assets, their individual capabilities might be considered too fine-grained (or too “chatty”) for an engagement layer. In such cases, it makes sense to re-cast common sequences of API calls under a microservice application, externalized as a new API.
2.1.3 API management and the OpenAPI Initiative

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 IBM Z business assets more easily usable for the rapidly expanding mobile and cloud-based application development community.

A key success metric for API-enablement of IBM Z assets is discovery and ease of consumption. An API enablement technology must make IBM Z 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) 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.

Discovery of a self-describing API through a marketplace with the social capabilities, such as number of users, ratings, and lifecycle updates allows 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.

2.1.4 Security standards for APIs

The digital economy encourages organizations to liberate what might historically have been some of their most prized, and guarded, services and data. The objective is to create new sources of revenue, influence, market-share, or perhaps simply good-will. Attributes of trust and security must be implicit around the provision and use of such services, in order to have any chance for widespread adoption. Security by obscurity does not make the grade in this age, but common models to address some of the most difficult challenges around security are gaining acceptance in the marketplace.

The OpenAPI Initiative specification defines a selected range of standard security schemes, any of which might be used for a given API, but which are typically adopted across an enterprise API catalog. These schemes include basic authentication, an API key that can be specified as a header or as a query parameter, OAuth 2.0 common flows (as defined in RFC6749) and OpenID Connect.

Given the open-ended nature of API consumption, whether that be restricted to the internal scope of an enterprise or across enterprise boundaries, the adoption of open standards for security maximizes the chances for adoption, interoperability, and compliance. Security schemes such as OpenID Connect have evolved to a level of maturity and acceptance today that they are leading the way regarding how a myriad of previously unlinked systems can be combined under a common approach to security.

Note: For a further insight into the evolution from services, APIs, and microservices in an enterprise environment, see the article Microservices, SOA, and APIs: Friends or enemies?, by Kim J. Clarke (IBM), which is available online at: https://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html
However, not all stakeholders looking at a digital transformation project today will necessarily have such an end-to-end security scheme available but might have adopted certain aspects of these technologies. Authorization tokens, such as JSON Web Token (JWT) and Security Assertion Markup Language (SAML), can also be employed for APIs and services today (using HTTP headers) without full adoption of an overarching scheme, such as OAuth 2.0 or OpenID Connect. Although already providing functionality within a defined application or organizational scope, such solutions are often based upon home-grown conventions for the lifecycle of these authorization tokens.

The partial adoption of such schemes, or elements of them, has led to a divergence of local conventions around the creation, expiration, and distribution of authorization tokens, making integration for middleware solutions possible but often inconsistent in style. Therefore, early adoption of the open standard-based security schemes is desirable to maximize flexibility in the future as digital transformation projects widen in scope and ambition. After an enterprise looks to augment its own capabilities by employing APIs from providers that exist beyond its direct influence, open standard-based security schemes are not only desirable, but likely to be mandatory.

### 2.1.5 Advantages of APIs

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 parsers available for many programming languages. A JSON data structure is shown in Example 2-1.

Example 2-1  JavaScript that uses JSON-encoded array to represent structured names of employees

```javascript
var employees = [
    {"name":"Rob","surname":"Williams"},
    {"name":"Nigel","surname":"Gamblin"},
    {"name":"Richard","surname":"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 data structures to describe most resources. JSON is considered to be a simple representation of data for humans (or at least programmers) 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 IBM Z 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 intranet.
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 2-3.

![Figure 2-3 SOAP message](image)

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 Example 2-2).

**Example 2-2 Soap body**

```xml
<employees>
  <employee>
    <name>Rob</name>
    <surname>Williams</surname>
  </employee>
  <employee>
    <name>Nigel</name>
    <surname>Gamblin</surname>
  </employee>
  <employee>
    <name>Richard</name>
    <surname>Jones</surname>
  </employee>
</employees>
```
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 and events

The connectivity mechanisms described previously are predicated on direct and synchronous connectivity between two applications. Rather than exchanging information directly, messaging architectures place messages 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. This queue manager can also deliver messages across a network to other queue managers. The full benefits of a messaging architecture are realized when you integrate 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 assured 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 new digital channels and microservices applications.

The advent of event-driven architectures, where multiple cascading actions need to be taken upon one or more real-world events, are heavily reliant on messaging architectures. The increasing complexities of hybrid applications increasingly rely on this de-coupled, event-driven architecture where, for example, thousands of Internet of Things (IoT) devices might flow packets of information through an event bus, upon which applications can retrieve the relevant data.

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 presentation, application, data, 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. The focus of this paper is on the secure integration of applications.

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 reference architecture with IBM Z” on page 18
- 3.2, “Systems of Record” on page 19
- 3.3, “Systems of Engagement” on page 19
- 3.4, “Access layer” on page 20
- 3.5, “Integration layer” on page 21
- 3.6, “API gateway” on page 22
- 3.7, “Exposure gateway” on page 23
- 3.8, “Cloud affinity” on page 23
- 3.9, “Consumers of APIs” on page 24
3.1 Hybrid integration reference architecture with IBM Z

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 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.

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 Systems of Record

When contemporary digital applications present information 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 mainframe.

Access to SoRs depends on a number of factors, including the type of SoR that is used (for example, CICS, IMS, or Db2) and the type of interfaces made available (which can be API-, SOAP-, or message-based).

An SoR may be implemented as one of the following types of application:

- Traditional enterprise application
  
  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 support 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 application
  
  By contrast, these applications are a collective of multiple, loosely coupled components (as shown in yellow and blue boxes in Figure 3-1 on page 18). 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 one another. The communications models for these applications are almost always based on a combination of messaging and REST APIs. Owing the agile nature of microservices applications, they were originally regarded as a System or Engagement (SoE) pattern. However, more recently these microservices applications are maturing to provide SoR capability.

3.3 Systems 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. An SoE application may be implemented as a traditional application, however, new applications are likely to use the microservices model.
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 SoR application. To facilitate fast and agile development of SoE applications, developers must be able to quickly find services that are available. Ideally, a developer should be able to search a catalog of services and retrieve the associated definition of that service; in the case of an API, developers must be able to locate the 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 layer 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 requesters and service providers are needed in the enterprise?
  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 and to bill either 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.

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 whether the service is available internally within an organization or beyond.

API gateways can be implemented to address different requirements. Figure 3-1 on page 18 shows two logical Gateways: the Exposure Gateway to support requests from external sources and the API Gateway to support internal requests. The ability to deploy multiple 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 intuitive to use and simple to look up from a searchable catalog.

- Do you want to reach new markets, customers, and partners?
  By making core business functions available as APIs to external consumers, 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 Exposure gateway

The most effective mechanism to ensure that proper access management policies are enforced is by using a centralized exposure or 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 exposure 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, an exposure gateway can be deployed as a physical appliance with tamper-proof protection.

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

► Is an exposure 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 and exposure 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

An exposure 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 of security 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.

3.8 Cloud affinity

There are multiple deployment options for each of the architectural components discussed previously, which highlights the truly hybrid cloud nature of enterprise IT today. Figure 3-1 on page 18 shows the following distinct deployment domains:

► Enterprise represents the traditional server environments, where single or related workloads reside on dedicated servers that are owned and managed by that organization.
• *Private Cloud* represents where multiple application and data components are deployed, removed, and re-deployed on demand across a multi-server landscape, all of which resides within that organization’s data centers.

• *Public Cloud* represents where multiple application and data components are deployed, removed, and re-deployed on demand and are hosted in environments that are hosted externally to the organization.

The cloud affinity shown in Figure 3-1 on page 18 illustrates how architectural components shown on right side of the figure, such as the SoR, are more commonly hosted in an enterprise domain, whereas components shown on the left side of the figure are more commonly hosted in a cloud domain. These are not mutually exclusive and the cloud philosophy of “choice with consistency” means that organizations are increasingly free to deploy applications and data services to the domain that is most appropriate to the qualities of service that they must deliver.

### 3.9 Consumers of APIs

There are multiple types of consumers of APIs that are surfaced by the exposure gateway, including mobile, web, IoT, partners, and public cloud applications. In an API economy, these consumers typically adopt a pay-per-use model and are concerned only with the reliability and responsiveness of the APIs for which they are paying. As such, it is critical to consider the path from the API consumer through to the SoR. In some cases, it is necessary to pass through multiple architectural components, as shown in Figure 3-1 on page 18 however, greater efficiency and responsiveness can be achieved by consuming the API at source in the SoR.

Consider the following scenarios:

• *Get new quotation*, which involves combining business logic from multiple applications, rules engines and historical data sources

• *Get account details*, which involves retrieving a set of data that is entirely encapsulated within a single SoR

In the first scenario, it is necessary to use multiple tiers of the hybrid architecture in order to combine and aggregate services. In the second scenario, the API managed in the exposure gateway can directly call the API in the appropriate SoR, which is more efficient than flowing through multiple layers of the architecture. In this case, efficiency can be measured in terms of API response time and development and operational effort. As such, wherever possible, it is essential to minimize the path length and consume APIs as close as possible to the provider.
IBM integration solutions

This chapter describes the main solutions and products that can be used for REST API enablement of IBM Z 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 26
- 4.2, “CICS” on page 30
- 4.3, “IMS” on page 36
- 4.4, “Db2 for z/OS” on page 38
- 4.5, “IBM MQ” on page 40
- 4.6, “IBM App Connect Enterprise” on page 43
- 4.7, “IBM API Connect” on page 47
- 4.8, “IBM DataPower Gateway” on page 50
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. This capability is referred to as the API provider support.

z/OS Connect EE also provides the capability that allows z/OS-based programs to access any RESTful endpoint, inside or outside the enterprise, for example a cloud-based microservice. This framework enables applications running in CICS or IMS and batch programs to call RESTful APIs through z/OS Connect EE. This capability is referred to as the API requester support.

4.1.1 API provider

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

![Figure 4-1 z/OS Connect EE API provider](image)

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 many types of z/OS asset, including CICS and IMS applications, Db2 data, and MQ queues and topics.

2. **Service providers**
   
A z/OS Connect EE service provider forwards requests to a System of Record (SoR). The following service providers are included with z/OS Connect EE:

   - A CICS service provider for connecting to CICS (see 4.2.1, “z/OS Connect EE with CICS” on page 31)
   - An IMS service provider for connecting to IMS (see 4.3.1, “z/OS Connect EE with IMS” on page 36)
– A REST Client service provider for connecting to a REST service (HTTP/JSON endpoint), for example, a Db2 REST service (see 4.4.2, “z/OS Connect EE with Db2 REST services” on page 39)

– A WebSphere Optimized Local Adapter (WOLA) service provider for connecting to WOLA-enabled applications, for example, a custom long-running task

Other IBM products also provide integration with z/OS Connect EE, for example:

– IBM MQ includes a service provider for putting or getting messages from an IBM MQ queue (see 4.5.2, “z/OS Connect EE with IBM MQ” on page 41).

– IBM Rational® Host Access Transformation Services (HATS) provides a unified way for REST APIs to access 3270-based applications. The APIs from HATS can be combined with z/OS Connect EE to enable more meaningful API names and flexible API parameters.

– IBM Data Virtualization Manager for z/OS includes a service provider that enables direct access to data, for example, VSAM and sequential file data, with Select, Insert, Update, and Delete functions using a RESTful interface.

– IBM File Manager for z/OS includes a service provider that enables access to data sources through File Manager for z/OS.

You can also 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. For example, to create a service from an existing CICS COBOL application, you import the copybook that defines the program interface into the z/OS Connect EE API Toolkit. You then use the API toolkit to define the service interface, including:

– To assign fields to constant values
– To rename fields to make them more intuitive
– To selectively omit fields from the interface

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 message. 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 different API paths on which the specific services are implemented.

The API mapping model provides 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 (as shown in Figure 4-2).

Figure 4-2 shows the relationship between APIs and the service archive (.sar) files, which contain the information that is needed by the z/OS Connect EE service provider to install and provide the service and to enable the service as a JSON asset.

5. server.xml

z/OS Connect EE is based on Liberty server technology; therefore, the z/OS Connect EE server is 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 or to import into an API management system, such as IBM API Connect® (see 4.7, “IBM API Connect” on page 47).

**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.
4.1.2 API requester

An overview of the API requester support in z/OS Connect EE is shown in Figure 4-3.

![Figure 4-3 z/OS Connect EE API requester](image)

The following components are shown in Figure 4-3:

1. REST API
   
   This component is the RESTful endpoint that is described in a Swagger document.

2. z/OS application
   
   This component is the CICS, IMS, or batch application that needs to call the REST API.

3. API requester archive
   
   Based on the Swagger document of the REST API, you use the z/OS Connect EE Build Toolkit to generate the artifacts for the API requester. The artifacts include the API requester archive (.ara) file to be deployed to the z/OS Connect EE server, and API information file and data structures that are used by the z/OS application program for calling the REST API.

4. Communications stub
   
   The z/OS Connect EE communication stub is the module that establishes an HTTP connection from the z/OS application to the z/OS Connect EE server.

5. server.xml
   
   The z/OS Connect EE server is configured in the Liberty server.xml file.

6. HTTP(S) endpoint
   
   The HTTP(S) endpoint of the REST API which is configured in the server.xml file.

See 5.5, “Call out to external services using z/OS Connect EE” on page 65 for an example of using the API requester support of z/OS Connect EE.

4.1.3 z/OS Connect EE run time

z/OS Connect EE is based on Liberty server technology and is lightweight and easily configurable. It benefits from the security foundation of Liberty, for example, for authentication using a variety of mechanisms including open standards like JSON Web Tokens (JWTs) and encryption based on the Java Secure Sockets Extension (JSSE).

z/OS Connect EE also 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 URIs can be classified
and measured so that you have accurate data about how many times an API is called and the
performance characteristics of the API.

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.

You can use z/OS Connect EE policies to adjust how an API request is processed, based on
the HTTP header values sent in by the client. You create rule sets to define the condition and
actions, then enable z/OS Connect EE policies to apply those actions to API requests. For
example, a policy that routes requests to a specific SoR based on the value of an http header.

4.1.4 When to use z/OS Connect EE

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

► Create APIs from existing z/OS assets using a tool-based approach that requires no
programming.
► Simplify the REST API development process by making the mainframe application owner
responsible for creating APIs from z/OS assets, and calling external APIs from z/OS
applications.
► Support the discovery of defined APIs by using the OpenAPI standard to share API
definitions as Swagger documents.
► Allow z/OS applications (CICS, IMS and batch) to call external REST APIs
► 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 specialty engines.

For more information about z/OS Connect EE, see IBM developerWorks:
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
► Web applications in CICS Liberty
► CICS 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 (see 4.1.1, “API provider” on page 26). Using z/OS Connect EE, a CICS program can also call any RESTful endpoint (see 4.1.2, “API requester” on page 29).

In API provider mode, z/OS Connect EE can be configured to connect to CICS using the CICS service provider that uses an IPIC connection. Consider the use of z/OS Connect EE with the CICS service provider when you want to provide intuitive, workstation-based tooling that enables a developer to create REST APIs from CICS applications.

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

![Figure 4-4 z/OS Connect CICS service provider](image)

As shown in Figure 4-4, 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 and a JSON schema representation of the service interface.

The CICS service provider supports COMMAREA or channel interface programs. The request message is converted from JSON to a byte array and the CICS program is invoked using the CICS service provider across an IPIC connection. The same z/OS Connect EE instance can be used for API enablement of different SoRs (CICS, IMS, Db2, and so on). Consider the use of z/OS Connect EE with the CICS service provider when you want to provide intuitive, workstation-based tooling that enables a developer to create REST APIs from CICS applications. For more information about other advantages of the use of z/OS Connect EE for the creation and deployment of APIs, see 4.1.4, “When to use z/OS Connect EE” on page 30.
### 4.2.2 Web applications in CICS Liberty

To provide REST or SOAP interfaces to CICS applications, you can develop a presentation layer that uses web application technology. Eclipse-based web development tools provide the development platform to create these applications, and the CICS Explorer SDK provides the support to build, package, and deploy them to run in CICS Liberty.

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 4-5.

![Figure 4-5 Web application in CICS Liberty](image)

When developing a Java program to link to a CICS COBOL program, you typically need to map the data fields from a record structure to specific Java data types. To do this, you can generate Java helper classes from a COBOL copybook using the IBM Record Generator for Java V3.0. Alternatively, the IBM Rational J2C tooling that is supplied with IBM Developer for IBM Z Enterprise Edition can be used for the same purpose.

The Java program can also 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 Link to Liberty capability 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 Web applications in CICS Liberty

Consider the use of custom Web applications in CICS Liberty 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 specialty 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 4-6.

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 has been 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**

JSON web services is a CICS technology for enabling programs as JSON services. These can be RESTful services or request-response remote-procedure-call style services. The technology is like that used for SOAP web services and uses the web services assistant to create binding files that are used for JSON message transformation. These binding files are deployed into CICS as WEBSERVICE resources.

You can configure either a Java-based or non-Java service provider pipeline. 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.

**Note:** IBM's strategic solution for creating JSON-based CICS services is z/OS Connect EE, which offers many additional integration options and capabilities.

A non-Java service provider pipeline is appropriate when a non-Java solution is required, for example, on machines in which no zIIP specialty engines are configured. Some workloads might also realize performance and throughput benefits; however, none of the processing is eligible for offloading to zIIP specialty engines.

An overview of the non-Java service provider pipeline is shown in Figure 4-7.

![Figure 4-7 CICS JSON web services (non-Java service provider)](image)
As shown in Figure 4-7, 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 basic JSON web service that matches an existing CICS program interface.
- Implement a solution that does not require a CICS JVM.
- Develop a CICS service requester application that invokes an external REST API.

### 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 4-8.

**Figure 4-8  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 or with CICS VSE.
- Reuse a CICS TG infrastructure; for example, a set of cloned CICS TG daemons that enable high availability.
Use Java-based message transformation that can be offloaded to zIIP specialty engines (when CICS TG daemon runs on z/OS).

Implement message transformation on a distributed platform.

### 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 (see 4.1.1, “API provider” on page 26). Using z/OS Connect EE, an IMS program can also call any RESTful endpoint (see 4.1.2, “API requester” on page 29).

In API provider mode, z/OS Connect EE connects to IMS using the IMS service provider.

An overview of the use of z/OS Connect EE with the IMS service provider is shown in Figure 4-9.

![Figure 4-9  z/OS Connect EE IMS service provider](image)

The REST client invokes an API using the interface that is shared in a Swagger document, as shown in Figure 4-9. 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 and a JSON schema representation of the service 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 can then access IMS DB, Db2, or other subsystems.

Consider the use of z/OS Connect EE 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 4.1.4, “When to use z/OS Connect EE” on page 30.
For more information about the use of the IMS service provider, see IBM developerWorks:
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 4-10.

Figure 4-10   IMS SOAP Gateway

The SOAP Gateway runs on z/OS and acts as a 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.

Note: The IMS SOAP Gateway is functionally stabilized, although IBM continues to provide support for existing functionality. See IBM Software Announcement 216-048 for more information.

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.
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. 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.

Db2 native REST services use 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. You can also issue the BIND SERVICE subcommand to create a new REST service. An overview of how a Db2 REST service is created and then called from a REST client is shown in Figure 4-11.

**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.

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 4-12.

The REST client invokes an API using the interface that is shared in a Swagger document, as shown in Figure 4-12. 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, for example, using open standards like JSON Web Tokens.
- Manage API access control and audit by using the z/OS Connect interceptor framework.

4.5 IBM MQ

IBM MQ is 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.

This section describes the following main integration solutions that can be used for MQ API enablement:

- MQ messaging REST API
- z/OS Connect EE

Note: HTTP is not a transactional protocol; therefore, no transactional coordination of messaging operations performed by REST clients is possible.

4.5.1 MQ messaging REST API

The messaging REST API comes as standard with IBM MQ from IBM MQ Version 9.0.4 and is enabled by default. You can use the messaging REST API to send and receive IBM MQ messages in plain text format.

- Applications can issue an HTTP POST to send a message to IBM MQ or an HTTP DELETE to destructively get a message from IBM MQ. The messaging REST API is integrated with IBM MQ security.
- The messaging REST API is enabled using the mqweb server, which is a Liberty server integrated with IBM MQ (Figure 4-13).
When a client application uses the REST API to perform a messaging action on an IBM MQ queue object, the application programmer needs to construct a URL to represent that object. The URL describes which host name and port to send the request to and describes a particular object. The HTTP method determines the messaging action that is to be performed on the resource, and additional information is sent in path parameters and query parameters. The messaging REST API is integrated with IBM MQ security. The caller must be authorized to access the specified queue.

Note: The MQ messaging REST API supports only UTF-8 text, so the z/OS application must perform data conversion itself.

When to use the IBM MQ messaging REST API
Consider the use of the IBM MQ messaging REST API when you want to:

- Create an MQ-aware REST client application to send and receive messages to and from queues.
- Allow users to put and get messages to any queue that they are authorized to access without needing any special configuration to expose those queues as REST APIs.

4.5.2 z/OS Connect EE with IBM MQ

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 or topics. You can achieve this configuration without being concerned with the coding that is required to use asynchronous messaging. The 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 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 MQ service provider enables the following distinct types of service:

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

The JSON schemas, service archive (SAR file), and binding files that are used for mapping JSON messages to MQ messages are created using utilities that are provided by z/OS Connect EE. The service can be called directly, or the API toolkit can be used to create a REST API that is then associated with the service. In combination with values from the HTTP headers and URI path and query parameters, JSON fields can be mapped to the JSON request and response schema that represent the MQ message.

Two-way requests
A two-way request allows a REST client to perform request-reply messaging against a pair of queues, as shown in Figure 4-14.

![Figure 4-14 z/OS Connect EE with MQ service provider and two-way service](image)

The client issues an HTTP POST request that specifies a JSON payload, as shown in Figure 4-14. The API defines the REST interface that you want to enable for the MQ queue, including the format of the URI and the specific two-way service on which the API is implemented. The API mapping allows inline manipulation of requests, such as mapping HTTP headers, pass-through, redaction, or defaulting of JSON fields.

Configuration definitions in the `server.xml` file provide information about the JMS destination, and the MQ service provider puts the message on the Request queue. An IBM MQ-based application, such as CICS or IMS, gets 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, converts it to JSON, and returns it as the response body of the HTTP POST to the REST client.

z/OS Connect EE can 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 requests
In the one-way request, 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.

The API defines the REST interface that you want to enable for the MQ queue, including what HTTP verbs are used and the one-way service(s) on which the API is implemented. For example, when a REST client issues an HTTP POST with a JSON payload, the API will map the request to a one-way service, and the MQ service provider will put the message on the target queue or topic (see Figure 4-15). When the REST client receives an HTTP 200 response, this response is confirmation that the message was successfully placed on the queue.

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 EE can be configured to convert it to JSON.

When to use z/OS Connect EE with the MQ service provider
Consider the use z/OS Connect EE with the MQ service provider when you want to perform the following tasks:

- Create a REST interface to an MQ-based mainframe application that can be used by client application programmers without knowledge of MQ.
- Use the security capabilities of z/OS Connect EE (for example, OpenID Connect or OAuth 2.0) to secure REST APIs for MQ-based applications.
- 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.

4.6 IBM App Connect Enterprise

When the product we now know as IBM App Connect Enterprise (formerly WebSphere Message Broker, and then IBM Integration Bus) was first released, it was positioned to become the all-encompassing integration tool for a then entirely on-premises corporate IT landscape. It was eventually able to connect anything you can think of to anything else you can think of. Today, this capability includes anything in the cloud and in the Internet of Things (IoT).
However, in the beginning, this technology was limited to IBM MQSeries® (now IBM MQ) as the supported transport technology. This transport technology made it easy and quick to build integration logic between applications that used IBM MQ, but adjustments were needed to message formats or routing. Typical usage patterns would intercept request and response messages, perform transformations and possible enrichment with database contents, and pass the resulting messages on to their destinations.

Messages can also be routed to different destinations based on their contents or even distributed to multiple destinations simultaneously, even with a different format for each. Messages could also be saved in database tables, which allowed the creation of “utility” type services, such as database inquiries, updates, inserts, and deletions. By using IBM MQ’s support for global transactions, complex transactions of this nature could be protected by the extended architecture (XA) protocol for two-phase commit transactions.

In general, IBM App Connect Enterprise inherited all the strong features that IBM MQ was renowned for, such as its speed, ease of use, and widely-acknowledged high reliability.

Integration developers use the Eclipse-based toolkit to create integration logic in the form of graphically built message flows. Depending on their specific function, these flows might need extra constructs to run, for example:

- Programming code in Java, Extended Structured Query Language (ESQL), PHP Hypertext Preprocessor (PHP), or any .NET language
- XML Schema Definitions (XSDs) and Web Services Description Language (WSDL) files to describe message formats or services

Message flows are built by wiring together functional modules that are graphically represented by processing nodes through connection terminals with specific uses. Most nodes have an input terminal where they receive the message under processing and a number of output terminals that pass the message on to the next step in the flow. Terminals are also provided to connect to catch logic for exceptions or failure handling.

It is typical for IBM App Connect Enterprise that for the key functions, such as transformation and routing, it provides a number of alternative ways to do them. Transformations, for instance, can be accomplished by using graphical mapping, stylesheets (XSLT), or a number of programming languages, such as Java, ESQL, PHP, or any of the .NET languages (under Windows.)

The functions of IBM App Connect Enterprise can be extended by adding custom-made, possibly user-developed, parsers or processing nodes. For example, WebSphere Transformation Extender is available to run as a plug-in node and a parser for IBM App Connect Enterprise.

The graphical development environment leads to an easy and fast development cycle. The graphically represented message flows are to a large extent self-documenting, which leads to better team development dynamics and easier support. Comprehensive built-in testing facilities, such as tracing, step-by-step execution, and message recording, further improve developer productivity.

An overview of each of the components of App Connect Enterprise is shown in Figure 4-16 on page 45. The following productivity tools are available:

- An ever-growing library of executable samples
- Numerous wizards that fast track the start of a project
- A pattern capability that allows developers to generate message flows from IBM-provided or customer-authored patterns that represent common flow designs or architectures
> A comprehensive monitoring feature that can collect and emit information at any point in a message flow
> Message flow statistics, which are optionally extremely fine-grained
> Message record and replay

![IBM App Connect Enterprise](image)

**Figure 4-16 IBM App Connect Enterprise**

By the time service-oriented architectures (SOAs) became popular and the need for a mediation tool called ESB was postulated (around 2005), it was discovered that earlier incarnations of IBM App Connect Enterprise had been just that, and more, all along.

IBM App Connect Enterprise supports the following transport protocols:

- IBM MQ
- JMS 1.1 and 2.0
- HTTP and HTTPS
- Web services (SOAP and REST)
- File
- Enterprise Information Systems (including SAP and Siebel)
- TCP/IP
- Supported message formats include binary formats (C and COBOL), XML, and industry standards (including SWIFT, EDI, and HIPAA). Custom formats can also be defined.

IBM App Connect Enterprise’s event handling features include the techniques that are used by the various input nodes to detect changes in the environment. From messaging protocols, which by their nature automatically alerts the broker of arriving messages to the polling of FTP servers or file directories or database triggers employed by the Database Input node, the most appropriate method is implemented for each transport medium. Another key feature is the ability to publish information in a number of publish/subscribe environments (such as IBM MQ and MQTT). Complex events can be configured by using the Collector node, where a set of rules can describe sets of incoming events that will then be combined to create one outgoing “complex” event.
High availability configurations of IBM App Connect Enterprise can be created in a number of ways, most easily by using the high availability capability of an underlying multi-instance IBM MQ queue manager, or Shared Queues on z/OS.

Over the years, IBM App Connect Enterprise has been enhanced by many added capabilities and usability improvements, while also becoming much more compact, agile, and performant.

Because it is the central integration hub in many large enterprise systems, IBM App Connect Enterprise is typically found in on-premises deployments, where such enterprise systems are located. But in a hybrid integration context, IBM App Connect Enterprise is also often found in private cloud implementations, such as IBM Cloud Private.

IBM also offers App Connect Enterprise on Cloud in a public cloud format, running inside Docker containers and positioned for easy access to and from enterprise environments. This implementation is still evolving, and at this time does not fully support all of the capabilities of IBM App Connect Enterprise on-premises. The main reason for this is that many of those capabilities are directed at accessing resources of the on-premises systems on which IBM App Connect Enterprise is usually found.

Instead, IBM App Connect Enterprise has been (and will continue to be) enhanced by a number of new capabilities that facilitate linking between IBM App Connect Enterprise and various other systems that are typically found in the cloud, including many popular cloud native applications and, in particular REST APIs.

The following REST API integration capabilities are provided by IBM App Connect Enterprise:

- 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 App Connect Enterprise Toolkit, which can be used to import a Swagger document.

### 4.6.1 When to use IBM App Connect Enterprise

Consider the use of IBM App Connect Enterprise for hybrid cloud and API integration with IBM Z when you want to perform the following tasks:

- Reuse an application integration service that is based on IBM App Connect Enterprise 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 App Connect Enterprise, see the following website:

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, IBM Business Partners, and other stakeholders to work together to develop and manage the entire API lifecycle from one foundational platform.

The steps of the API lifecycle include creating, running, managing, and securing APIs, as shown in Figure 4-17.

Figure 4-17  API lifecycle steps

The steps of the API lifecycle are as follows:

- **Create**
  Develop and write the API definition and implementation, and test the API.

- **Run**
  Package and deploy the API. Ensure that the API is hosted securely on a stable platform.

- **Manage**
  Create and manage self-service portals that expose the API to API consumers. Monitor the set of rules and conditions that govern the API to ensure it is fulfilling its intended purpose, and make adjustments if necessary. Retire and archive the API when appropriate.

- **Secure**
  Incorporate access control, monitoring, and logging to properly secure the API.

**Note:** You can deploy API Connect in the enterprise or as a cloud offering as part of IBM Cloud.
4.7.1 How IBM API Connect works with z/OS Connect EE

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 4-18.

The following steps are shown in Figure 4-18:
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 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 4-19.

![Diagram](image)

**Figure 4-19  Creating and running APIs with 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 4-19:

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 IBM developerWorks:

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 4-20.

![Figure 4-20 IBM DataPower Gateway](image)

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

- As a security gateway, IBM DataPower secures access to corporate data and services, while optimizing delivery of the workload. It provides 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 Cloud using the IBM Cloud 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.

**Note:** IBM DataPower Gateway is available in physical, virtual, cloud, Linux, and Docker form factors.

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 IBM developerWorks:

https://developer.ibm.com/datapower/
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 provides an outline of the following real-world scenarios:

- A CICS web services enablement project
  (See section 5.1, “Enable SOAP web services” on page 54.)
- An implementation of Java-based REST APIs using CICS Liberty
  (See section 5.2, “Develop Java-based REST APIs” on page 56.)
- An implementation of Open Banking APIs using z/OS Connect EE
  (See section 5.3, “Implement Open Banking APIs with z/OS Connect EE” on page 59.)
- Creation of a managed API framework using API Connect
  (See section 5.4, “Build a managed API framework using API Connect” on page 62.)
- Calling out to external services from a CICS application using z/OS Connect EE
  (See section 5.5, “Call out to external services using z/OS Connect EE” on page 65.)
5.1 Enable SOAP web services

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 nondisruptive 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.
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 services 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.

- **API Gateway**
  
  IBM API Connect is used as an API gateway that converts REST and JSON requests from mobile and web clients to XML. These requests are then sent to mainframe applications using IBM MQ or CICS web services. The API gateway is deployed to a physical DataPower appliance for optimum security and performance.

### 5.1.4 Next steps

Bank A has rolled out the first wave of CICS web services which are accessed as REST APIs via the API Gateway. Additional capabilities of API Connect will next be used to improve service management and governance.

### 5.2 Develop Java-based REST APIs

Java is an increasingly popular programming language for new z/OS applications. The z/OS Java products provide the same, full function Java APIs as on all other IBM platforms. This scenario focuses on the creation of custom Java applications that enable REST APIs based on a CICS PL/I core banking applications.

#### 5.2.1 Introduction

Bank B is a long-established financial services provider that wants to capitalize on their mainframe investment by offering new services through a set of REST APIs. The key project goals are as follows:

- Deliver modern intuitive standard-based APIs that enhance the developer experience.
- Enable delivery at speed.
- Reuse existing Java development and JVM management skilled resources.
- Enable Java client application developers to build server-based Java APIs.
- Create a cost-effective solution.
- Modernize the application delivery pipeline.

The question to be addressed by this project is whether to use an off-the-shelf product for REST API enablement or to develop a custom solution.
5.2.2 Key decision factors

Bank B considered several factors when deciding how to API enable the CICS PL/I core banking applications. The bank sees the capabilities delivered by these APIs as being key to their digital strategy and a potential significant differentiator with competitors. As such, retaining full control over the experience is an important requirement. The key decision factors are reviewed next.

**Mainframe application interface**

Today, proprietary XML messages are sent into CICS using a custom CICS TCP/IP sockets-based solution. The existing COMMAREA-based PL/I programs are limited to 32 KB message lengths so returning large amounts of data requires multiple requests to be made.

The new solution should support a more loosely REST/JSON interface that gives the developer complete control over the interface, for example, the REST API should control client interactions by including conditional links (URIs) in the JSON response messages. The solution should also support long messages without the need for multiple round trips.

**Application integration infrastructure**

The core banking applications are based on CICS TS and Db2 z/OS, and IBM CICSPlex® SM is used to manage a CICSplex of regions that support the applications.

A security gateway provides authentication and identity propagation services.

**Hybrid integration requirements**

Bank B provides financial services to over one million clients that use modern web and mobile applications to access their accounts. These SoE applications are deployed on-premises today but cloud implementations are being considered for the future.

The enablement of REST APIs is seen as a significant stepping stone to the creation of a seamless omni-channel customer experience, whether the customer is using a desktop or mobile device for online banking.

**Non-functional requirements**

The following main non-functional requirements have been identified:

- **Skills**
  
  Maximize use of exiting Java skills and federate these skills across SoE and SoR development projects in order to develop a friction-less API deployment process.

- **Performance and cost**
  
  The solution must perform as well as or better than the existing solution and enable cost advantages.

- **Security and identity management**
  
  The solution must be consistent with the existing security model.

5.2.3 Solution architecture

Bank B chose to develop a Java solution based on the CICS Liberty support. New Java integration logic is developed using the OSGi framework. The Java integration layer uses hypermedia links to control the flow of client interactions. The solution is shown in Figure 5-2.
The solution features the following components:

- **Liberty JVM server**
  
  CICS Liberty was chosen as the runtime server because of its simple XML-based configuration, colocation with the core banking applications and tight integration with RACF. Java applications running in CICS benefit from being eligible for zIIP offload which can significantly reduce the application cost of ownership.

  The Java applications run in different CICS regions than the PL/I core banking programs. This allows the CICS Java regions to be configured specifically for running Java and to be upgraded independently of the CICS application owning regions (AORs).

- **Custom Java components**
  
  A CICS Liberty server hosts the following custom components:
  - A Web application that uses JAX-RS to create the RESTful user interface for an API
  - OSGi services that support dynamic deployment and are shared across multiple APIs
  - A Trust Association Interceptor (TAI) that is used to switch the CICS user context to the user ID sent by the Security Gateway

  Wherever possible, Java components are tool-generated and share the same delivery pipeline as the PL/I components.

- **PL/I core banking applications**
  
  The PL/I core banking applications are called from the Web application using the JCICS API. Program link requests are dynamically routed to CICS AORs using CICSPLex SM. CICS support for channels and containers is used to enable the transfer of long messages and to structure the message into logical blocks.

- **Security gateway**
  
  The security gateway provides authentication and identity propagation services. End users are authenticated against an enterprise LDAP directory and the user’s identity is flowed to the mainframe for access control.
5.2.4 Next steps

Bank B has now deployed many APIs using their custom Java solution. Work is ongoing to migrate older services to the Java-based solution, and to consider how request results can be cached. Bank B is also considering if z/OS Connect EE can be used for REST API enablement of certain applications.

5.3 Implement Open Banking APIs with z/OS Connect EE

This scenario focuses on the API enablement of an IMS application and data services to support an Open Banking initiative.

5.3.1 Introduction

One leading industry that is fast exploiting the API economy is the financial services sector. Under the umbrella term of Open Banking, financial institutions are providing consumers choice over the way in which banking services and data can be accessed and used. The idea is that the customers of a given Bank will be able to grant permission to third parties who will be able to retrieve that customer's banking data and issue payments on their behalf. From the customer's perspective, this enables them to go to a single provider and access all accounts across different Banks. From a Bank's perspective, they must provide a new kind of business channel that provides an easy-to-use and yet secure and robust access mechanism between the third party, acting on behalf of the customer, and the customer's accounts. The most widely adopted standard for this new Open Banking channel is REST APIs.

Although Open Banking initiatives are emerging throughout the world as an innovative new offering, in certain markets, such as Europe and Japan, Open Banking initiatives are being driven by regulatory mandate. As a result, in addition to the ease-of-use and security challenges presented by Open Banking, these regulatory-driven initiatives must also offer Open Banking services at speed.

Bank C is a major European bank that must adhere to the regulatory mandate for Open Banking. Today, they offer a mix of traditional and digital multi-channel access methods, including branch, ATM, call center, web and mobile. They have recently implemented two API gateways, one that surfaces APIs to external parties, and another that surfaces APIs to internal applications and service consumers.

The next step for Bank C is to streamline and accelerate delivery of Open Banking services through their API gateways, from their IMS core banking system. The following key questions are to be addressed in this project:

- Is the existing integration mechanism with the IMS core banking system fit for purpose for Open Banking?
- What alternative approaches could simplify and speed up integration between the API gateway and core banking systems?

5.3.2 Key decision factors

Bank C explored multiple factors when deciding on the best approach for surfacing core banking services to the API gateway. The primary factors were speed to market, ease of implementation and security. The key decision factors are reviewed next.
Mainframe application interface
The core banking functions in IMS are accessed in the following ways:

- An in-house developed C application that resides on z/OS provides synchronous connectivity services.
- An MQ solution for pseudo-synchronous and batch connectivity services.

In both of these cases, application developers creating new applications must have an awareness of the COBOL copybook structures required to call the IMS programs. These copybook structures commonly contain in excess of 50 fields. In many cases, the external API only exposes 2-5 fields. However, the application developers must still populate all of the fields of the copybook in order to correctly run the IMS program.

The new solution should provide synchronous connectivity that simplifies the mainframe application interface.

Application integration infrastructure
The core banking application runs on IMS TM and IMS DB. The application integration architecture includes two API gateways, one surfacing APIs externally to the organization, and a second surfacing APIs to internal consumers. Consumption of the IMS application is via a home-grown application or MQ.

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

Hybrid integration requirements
The advent of Open Banking regulations mandate that Bank C must offer API services. The Bank have chosen two API gateway technologies that offer native support for REST APIs, which has been adopted as the standard of choice across all platforms and applications. In order to adhere to this standard and become simpler to consume, the core banking team are seeking to surface REST APIs directly from IMS.

Non-functional requirements
The following key non-functional requirements have been identified:

- Simplification
  The solution must simplify the developer experience of discovering and calling core banking services.

- Optimize performance and scalability
  The solution must meet the performance and scalability expectations of Bank C, particularly as Open Banking initiatives are forecasting a double-digit growth in the number of requests over the next 3-5 years.

- Security
  The solution must meet industry standard API security models, specifically OAuth 2.0 and JWT.

5.3.3 Solution architecture
Bank C chose to implement a solution based on z/OS Connect EE (see Figure 5-3) because the adoption of REST APIs throughout the enterprise gives them the speed, simplification, and security model they need for enterprise-wide Open Banking services.
The solution features the following main components:

- **z/OS Connect EE**
  
z/OS Connect EE provides a REST API interface for the IMS core banking application. It receives requests containing only the essential pieces of information needed to run the request and augments the request with the numerous flags and fields needed to run the IMS program. z/OS Connect EE also provides a security framework that enables APIs to be secured using JSON Web Tokens (JWTs).

- **Internal API gateway**
  
The internal API gateway is used to surface APIs accessible to all applications and services within the boundary of Bank C. This includes other enterprise applications, as well as the external API gateway.

- **External API gateway**
  
The external API gateway is used to surface APIs to consumers outside the boundary of Bank C. For example, APIs are accessed through the web and mobile channels, and also more recently via the Open Banking channel.

- **Open Banking channel**
  
The Open Banking channel is offered by Bank C and is open to third parties who have permission to access and act on behalf of a customer’s behalf, under the terms and conditions of Open Banking.

### 5.3.4 Next steps

Bank C successfully completed a Proof of Concept with z/OS Connect EE, concluding it gives them significant agility benefits, reducing time to service creation from 2 months to less than 1 day. This is partly as a result of the tooling available in z/OS Connect EE, and partly because of the separation of expertise in the process. Their finding was that the speed at which the z/OS skilled specialist could create an API, that shares a common lexicon with the API consumer, significantly accelerated the end-to-end process.

At present, Bank C is bringing z/OS Connect EE to production and exploring the next applications to API-enable. These include card processing and mortgages systems that are based on CICS DB2 applications. It is expected that these new APIs will be deployed quickly as much of the security and management framework will have already been put in place for the IMS-based APIs.
5.4 Build a managed API framework using API Connect

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 though 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  Build a managed API framework using API Connect](image)

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 steps

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.

5.5 Call out to external services using z/OS Connect EE

This scenario focuses on calling external services from a CICS automobile insurance application.

5.5.1 Introduction

In the same way that SoE applications need to access SoR applications on the mainframe as APIs, mainframe applications often need to make calls to APIs available on external systems, either within the organization or outside the organization, on-premise or in the cloud.

Insurance company A is a large insurance company that provides life, automobile, health, and accident insurance. The automobile insurance application runs in CICS and is accessed either directly by call center sales personnel or via a company web site. Car insurance quotes are provided based on the vehicle type and driver's insurance history.

The insurance company wants to provide the most accurate real-time car insurance quotes by using the most up-to-date information about the vehicle and driver. To do this, the company wants to take advantage of two external services that are available as APIs:

- A public vehicle registration SOAP-based web service that returns vehicle details
- A real-time driver risk scoring cloud-based service that is available as a REST API

For simplicity, it is preferred to use a single protocol for calling all external services.

5.5.2 Key decision factors

Insurance company A considered different solutions for calling external services from the CICS insurance application, including z/OS Connect EE, CICS SOAP web services and CICS web support. A significant factor is the amount of development coding required. The key decision factors are reviewed next.

Mainframe application interface
CICS provides different ways to invoke external services. For example, using the INVOKE SERVICE command, a CICS application can call an XML-based service. However, the
Insurance company expects that most external services in the future will be available as REST APIs, so they prefer a REST-based solution.

The new solution should provide tooling for creating artifacts and code snippets that minimize the amount of required custom coding.

**Application integration infrastructure**

The automobile insurance application runs on CICS TS and Db2 and is currently accessed by emulated 3270 terminals used by call center staff, and by CICS Transaction Gateway used by web applications.

A parallel project is putting in place an API management solution based on IBM API Connect. All calls from enterprise applications to external APIs will pass through the API Gateway component of API Connect to enforce security and traffic management.

**Hybrid integration requirements**

The insurance company wants to take advantage of third-party APIs to use the data and services of other organizations, and to reduce development costs. Some of these APIs are available on public cloud environments.

**Non-functional requirements**

The following main non-functional requirements have been identified:

- **No impact on availability**
  
  The insurance application runs 24/7 to provide quotes to clients using the company’s web application. Calling out to the risk-scoring and vehicle registration APIs should not impact availability.

- **Performance**
  
  The solution must have minimal impact on service response times and General Processor (GP) CPU cost.

- **Monitoring**
  
  The number of calls to the external APIs must be monitored, for example, to track calls to fee-based APIs.

**5.5.3 Solution architecture**

Insurance company A chose to implement a solution based on the API requester support in z/OS Connect EE (see Figure 5-5).
The solution features the following main components:

- **z/OS Connect EE**

  z/OS Connect EE can be used by CICS, IMS or batch applications to call any RESTful endpoint, inside or outside the enterprise.

  z/OS Connect EE was chosen for the following reasons:
  - Based on the Swagger document of the external REST API you can use the z/OS Connect EE Build Toolkit to generate the artifacts used for the API request. These include the API requester archive (.ara) file that is deployed to the z/OS Connect EE server, and data structures that are used by the z/OS application program to call the REST API.
  - Most of the z/OS Connect EE processing can be offloaded to zIIP specialty engines, therefore minimizing the impact on GP CPU.
  - Timeouts can be set in z/OS Connect EE such that the CICS insurance application does not need to wait indefinitely for a response from the external APIs. The application provides provisionary quotes in the event that the external APIs are not available.

- **API Gateway**

  IBM API Connect is used as an API gateway that converts REST/JSON requests from z/OS Connect EE to the message format expected by the external APIs. The API gateway also monitors and logs all calls to external APIs, and applies the specific security policy that is compatible with the security requirements of the APIs.

### 5.5.4 Next steps

Insurance company A has successfully implemented this API requester scenario using z/OS Connect EE. The next step is to use z/OS Connect EE API provider support to API enable some of the z/OS insurance applications. This will simplify the integration of these mainframe hosted applications with the different channel applications.
Summary

The needs for faster speed to market, simpler interoperability within and outside the organization, and better customer experience, are driving the API economy. And as API adoption and the implementation of hybrid architectures, become universal, the requirement to interoperate with the mainframe is high on the priority list of many of the world's largest companies.

In this paper, we reviewed the different ways to integrate with the mainframe, presented a hybrid integration reference architecture and reviewed several real-world scenarios that highlight the decision factors for choosing one solution over another.

We conclude with a summary of 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>
<thead>
<tr>
<th>Integration architecture</th>
<th>Description</th>
<th>Recommendation</th>
</tr>
</thead>
</table>
| APIs and API management       | Architecture for creating, assembling, managing, securing, and socializing web application programming interfaces (APIs). | Use when:  
|                               |                                                                             | ▶ Business functions must be discoverable  
|                               |                                                                             | ▶ Enterprise applications must be extended to a system of developers and new markets  
|                               |                                                                             | ▶ Need high degree of operational governance |
| 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:  
|                               |                                                                             | ▶ JSON is the primary data payload  
|                               |                                                                             | ▶ Intuitive and simple interface for developers is required |
| Web services                  | Service-oriented architecture that is based on standards, such as SOAP, XML, and WS specifications. | Use when:  
|                               |                                                                             | ▶ XML is the predominant data payload  
|                               |                                                                             | ▶ WS standards are required  
|                               |                                                                             | ▶ Reusing an existing SOA infrastructure |
| Messaging                     | Asynchronous transport mechanism.                                           | Use when:  
|                               |                                                                             | ▶ Assured delivery is required  
|                               |                                                                             | ▶ Enabling publish/subscribe applications  
|                               |                                                                             | ▶ Reusing a messaging infrastructure |
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**

<table>
<thead>
<tr>
<th>Integration solution</th>
<th>Recommendation</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Connect Enterprise Edition | Provides a single common gateway for REST HTTP calls to assets and data on z/OS operating systems. Also provides the capability that allows z/OS-based programs to access any RESTful endpoint, inside or outside the enterprise. | Use when:  
- Want to use workstation-based tooling for creating RESTful APIs that are based on z/OS assets  
- Enabling discovery of defined APIs based on OpenAPI (Swagger 2.0) standard  
- Requiring z/OS applications (CICS, IMS and batch) to call external REST APIs  
- Want to deploy common REST API enablement solution for different z/OS based applications and subsystems |
| IBM API Connect Enterprise | API management solution for creating, running, securing, and managing APIs. | Use when:  
- Extending the value of mainframe assets by socializing APIs to various developers and partners  
- Controlled access and strong governance is required |
| IBM DataPower Gateway | SOA and API security gateway. | Use when:  
- Securing hybrid integration access to the main-frame  
- Deploying APIs to a secure and efficient API Gateway |
| IBM App Connect Enterprise | Comprehensive integration services with support for any-to-any transformation. | Use when:  
- Have complex integration requirements; for example, service orchestration  
- Diverse data and protocol formats are used  
- Industry standard message formats must be supported |
| IBM MQ | Asynchronous message transport for reliable delivery of messages. | Use when:  
- Enabling bidirectional messaging connectivity  
- Publish/subscribe pattern is required |
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, and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- OpenAPI and 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 App Connect Enterprise:

  https://www.ibm.com/cloud/app-connect-enterprise

- IBM API Connect:

  https://www.ibm.com/cloud/api-connect
IBM DataPower Gateway:
https://developer.ibm.com/datapower/

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services