Redbooks

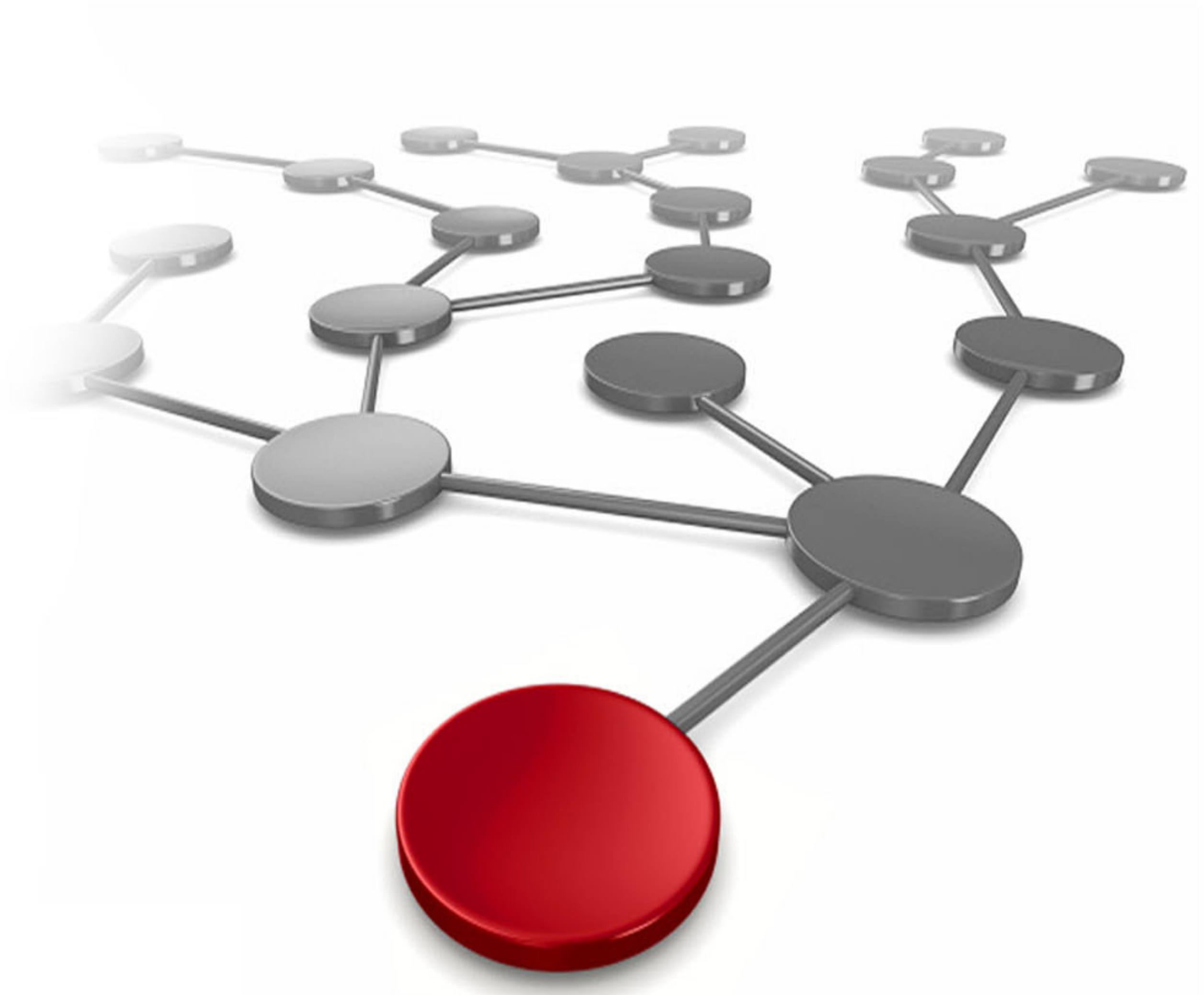
ibm.com/redbooks

IBM Z Integration Guide for Hybrid Cloud

Nigel Williams

Richard Gamblin

Rob Jones



IBM

Redpaper

IBM Redbooks

IBM Z Integration Guide for Hybrid Cloud

April 2020

REDP-5319-03

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

Fourth Edition (April 2020)

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

This document was created or updated on May 5, 2020.

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

CICS®
CICS Explorer®
CICSPlex®
DataPower®
DB2®
Db2®
IBM®
IBM API Connect®
IBM Cloud™
IBM Cloud Pak™
IBM Z®
MVS™
OMEGAMON®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
WebSphere®
z/OS®
z/VM®
z/VSE®

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Zowe, are trademarks of the Linux Foundation.

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

OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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-native applications, hosted in both public and private cloud infrastructures.

Leading organizations are now exploiting the best of both worlds by combining their traditional enterprise IT with cloud. This hybrid cloud approach places new requirements on the integration architectures needed to bring these two worlds together.

One of the largest providers of application logic and data services in enterprises today is IBM® Z, making it a critical service provider in a hybrid cloud architecture. The primary goal of this IBM Redpaper publication is to help IT architects choose between the different application integration architectures that can be used for hybrid integration with IBM Z®, including REST APIs, messaging, and event streams.

Authors

This paper was produced by a team of specialists from around the world working at the IBM Redbooks, Poughkeepsie Center.

Nigel Williams is an IT Specialist working in System Lab Services at the IBM Systems 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.

Richard Gamblin is the CTO for Digital Transformation for IBM Z and Member of the IBM Academy of Technology. Working with European clients, Richard aligns organization's Cloud, integration and application strategy and architecture to best exploit IBM Z technologies. He has worked in several technical roles in IBM, including as an Integration and Connectivity Specialist and an IBM WebSphere® Architect. Prior to joining IBM, Richard was a researcher at the University of Leeds, from where be obtained a doctorate in the field of Bioinformatics.

Rob Jones is Senior Technical Staff Member for APIs on IBM Z, working at the IBM Hursley Software Laboratory in the UK as the Chief Architect for the IBM z/OS® Connect Enterprise Edition product. Rob leads a world-wide team of software engineers and engages extensively in design thinking collaborations together with many IBM Z clients, but is also a part-time inventor with a growing patent portfolio, and author of several IBM Redbooks® publications. He previously led development of IBM CICS® Transaction Gateway products, and spent his early career in the communications maintenance team for CICS Transaction Server products.

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

Kim Clark
Mark Cocker
Dave Dalton
Andy Garratt
Andy Lyell
Matthew Leming
Anthony Papgeorgiou

Wayne Swales
Phil Wakelin
IBM UK

Aymeric Affouard
Arnauld Desprets
Yann Kindelberger
Vincent Lattuca
Eric Phan
IBM France

Bruce Armstrong
Leigh Compton
Mitch Johnson
IBM US

Special thanks to Andy Lyell for the graphics included in this 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 IBM Redbooks 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, IBM Redbooks
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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

Summary of changes

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

Summary of Changes for *IBM Z Integration Guide for Hybrid Cloud* as created or updated on May 5, 2020.

April 2020, Fourth Edition

This revision includes the following new and changed information.

New information

- Agile integration architecture
- Messaging and event streams
- Integration patterns with IBM Z
- IBM Cloud™ Pak for Integration

Changed information

- z/OS Connect EE Version 3 enhancements
- Real-world scenario updates

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

1

Introduction

Digital engagement has become one of the primary vehicles to create innovative new services for both regional and global consumers. Cloud platforms are a key enabler for these burgeoning digital channels, incorporating public, hybrid, and private cloud platforms. The digital applications and microservices that provide these new capabilities are primarily designed to exploit the agility of cloud platforms. But must also harness the business logic and data services that are held within enterprise systems, such as IBM Z.

Access to business logic and data that is hosted on IBM Z has long been available through multiple paths, with more options being introduced to specifically ease the integration for cloud native applications. Today, integration patterns such as using REST APIs, messaging, and event streams are key to unlocking the investment in core z/OS-based systems.

In this introduction we focus on the integration challenges brought about by the growth of hybrid cloud architectures.

This chapter includes the following topics:

1.1 Digital transformation and reinvention

Organizations are undergoing significant change as a result of new market forces. Some of these changes are in response to regulatory requirements, while others relate to the emerging requirements of consumers of their products or services. Over the last 5 to 10 years, a significant shift occurred in the marketplace regarding how consumers and employees expect to interact with businesses and government institutions. There has been a 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 an unprecedented speed of change. Consider that traditional 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 — measured in decades and years — with the mobile and digital age, in which consumer expectation has rocketed in a mere 5 years, and industry disruption can happen in a matter of months.

For a time, new market entrants like FinTechs were seen to be key disruptors to established business sectors like the financial sector. They harnessed the power of cloud-native infrastructures and exploited commoditized compute power to quickly create new kinds of service. They were quickly able to harness new technologies such as artificial intelligence (AI) and blockchain digital ledger technologies. However, established market leaders are now proving to be the greatest catalysts for change. These so-called *incumbent disruptors* now achieve success by combining their traditional core business logic and data assets with innovative new digital services in the cloud. They exploit the best of both worlds with a hybrid cloud architecture.

1.2 Transforming IT with hybrid multicloud

According to a recent IBM study[1], 94 percent of enterprises today have a mix of cloud models: public, dedicated, private, and hybrid. Sixty-seven percent of enterprises are using multiple public clouds, mixing software, infrastructure, and platform services from different providers. While only 20 percent of core business applications and workloads have moved to public cloud, with 80 percent still residing on premises in data centers.

This new hybrid and multicloud reality brings with it new considerations for IT leaders who must continue to operate business as usual, while embracing greater complexity. In the IBM study, IT leaders expressed concerns about how they'll connect traditional IT with these clouds to address their specific use cases. Seventy-three percent needed better ways to move apps, workloads, and data between clouds so they can adapt to change, optimize costs, and minimize lock-in risk. Sixty-seven percent worried about how they'll manage this new mix of cloud environments in a consistent way, across vendors, without impacting service quality, security and compliance.

Hybrid cloud allows for a mix of environments - public, dedicated, and private cloud as well as traditional enterprise IT — all working together on and off premises. Multicloud enables choice of cloud-based technologies — artificial intelligence (AI), blockchain, IoT, analytics, infrastructure as a service (IaaS) and platform as a service (PaaS) from more than one cloud provider.

[1] IBM Cloud White paper: Enterprise digital transformation with cloud

Fundamentally, this shift from traditional enterprise IT, to hybrid, multicloud environments brings with it very real architectural considerations, particularly the need to improve the integration of services in this mix. These new demands dictate the need for a more agile integration approach.

1.3 Agile integration

The pace of innovation in IT has changed dramatically. Iterations on requirements occur in near real-time, prototypes are prepared in weeks or even days, and new mobile apps are made available in months. Application development techniques needed to keep pace by introducing new approaches such as microservices, exploiting cloud infrastructures, that enable teams to work more independently.

Integration solutions have traditionally been used to simplify how one enterprise application communicates with another. These solutions manage protocol switching and data conversion, and support both synchronous and asynchronous connectivity mechanisms. This is well suited for tightly governed and controlled applications within an organization. However, the rise of cloud-hosted applications requires a more self-service approach where application interfaces are easier to use, enabling application teams to more rapidly share data and functions.

There is much more subtlety to this change than is apparent. It is necessary to think differently about how to align the people and skillsets that relate to integration. It is necessary to consider how to ensure that integration components embrace new architectural tenets such as microservices and that they capitalize on the benefits of new infrastructure platforms such as containers.

As shown in Figure 1-1 *agile integration* addresses these issues by looking at how to modernize an integration landscape based on people, architecture, and infrastructure.

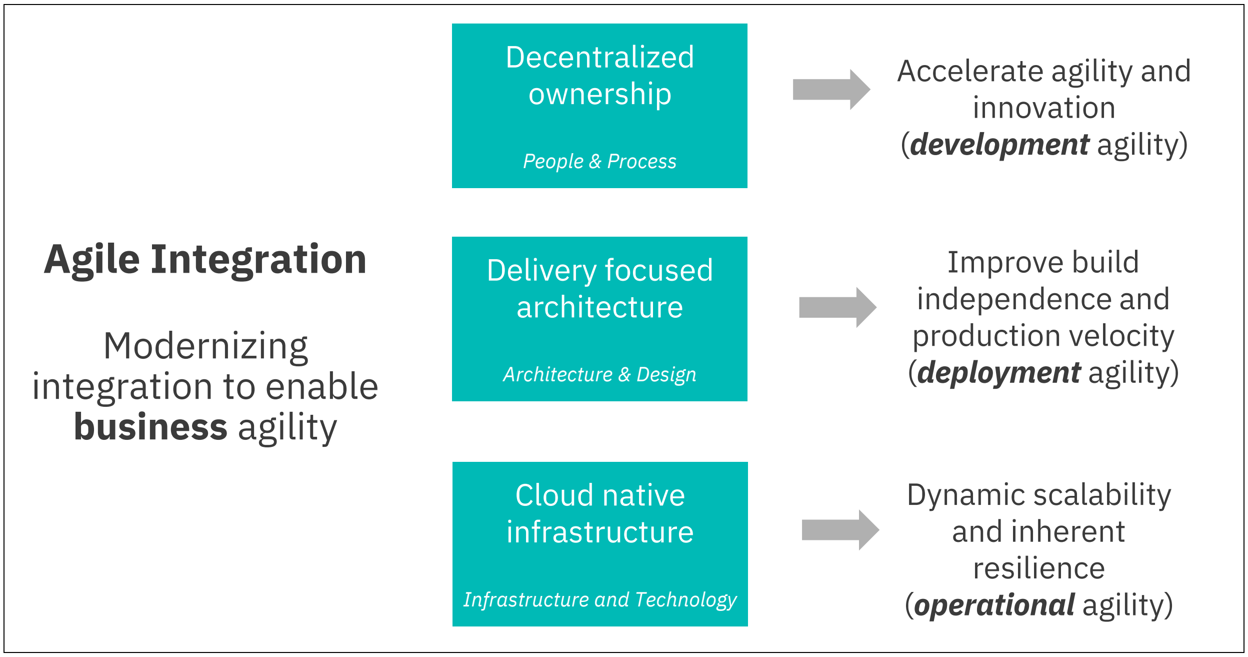

Agile Integration
Modernizing integration to enable **business** agility
Decentralized ownership
People & Process
Accelerate agility and innovation (***development*** agility)
Delivery focused architecture
Architecture & Design
Improve build independence and production velocity (***deployment*** agility)
Cloud native infrastructure
Infrastructure and Technology
Dynamic scalability and inherent resilience (***operational*** agility)

Figure 1-1 Three aspects of agile integration

Note: We define *agile integration* as a decentralized, microservices-aligned, portable approach to integration that addresses people, architecture and technology.

Integration is the secret weapon behind the great innovations of our time. Few, if any, new ideas are stand-alone applications. They always require data and functionality from other applications within the enterprise, and often from other enterprises.

However, today's integration challenge is less about low-level connectivity as most of these have been solved, or at least simplified. The challenge now is about velocity of change. How quickly can ideas be transformed into production, or at least into prototypes, so that new niches can be exploited?

This requires highly empowered and autonomous teams, that can self-provision the integration capabilities they need wherever they are, and yet still interact efficiently with other teams' capabilities.

Integration is front and center in enabling the business agility required to innovate faster than the competition. Agile integration addresses this need by rethinking the approach to integration based on people, architecture, and technology. It results in a decentralized, microservices-aligned, portable approach to integration:

- **People - Decentralized integration ownership:** Improve *development agility* by empowering teams with integration capabilities such that they can innovate in real time.
- **Architecture - Delivery focused integration architecture:** Improve *deployment agility*, using modern architectural and design practices such as API-led or event-driven integration with a focus on microservices applications.
- **Technology - Cloud portable integration infrastructure:** Improve *operational agility* by a platform agnostic, cloud-native approach to integration infrastructure.

For a detailed exploration of the principals of agile integration, see the IBM Redbooks publication *Accelerating Modernization with Agile Integration,* SG24-8452 (http://www.redbooks.ibm.com/abstracts/sg248452.html?Open).

1.4 Integration with IBM Z

The IBM Z mainframe has been widely adopted for delivering mission critical business applications and data services because of its industry-leading data privacy, security, and resiliency. IBM Z mainframes run operating systems including z/OS, IBM z/VM®, IBM z/VSE®, Linux on IBM Z, and z/TPF. It is common for multiple operating systems to run on a single mainframe.

Over the last 10 years, the adoption of Linux on Z has grown significantly bringing modern, cloud native applications to the platform. And more recent enhancements such as support for Red Hat OpenShift means that you can develop and deploy cloud native applications faster and more effectively on IBM Z.

Considering its continued strategic role in enterprise IT, IBM Z is an essential ingredient of a hybrid multicloud architecture. In this IBM Redpaper we focus on the hybrid integration of traditional mainframe z/OS applications.

Traditionally, integration with z/OS subsystems, such as IBM CICS, IMS, Db2® for z/OS and batch applications, has been based on file-based connectivity, point-to-point messaging, custom connectivity options, and more recently, web services. While these connectivity options are used widely, and have an important role in hybrid integration, they are being complemented by more contemporary integration architectures such as REST APIs and event streams.

- In Chapter 2, we review the main architectures that are used today for application integration.
- In Chapter 3, we look at the components of a hybrid integration architecture and explore the common integration patterns used with IBM Z.
- In Chapter 4, we describe the main IBM solutions and products that can be used to integrate with IBM Z applications in a hybrid cloud infrastructure.
- In Chapter 5, we review some real-world IBM Z integration scenarios.
- In Chapter 6, we conclude with a summary of the different integration architectures and solutions.

2

Architecture options for integration

This chapter describes the main architectures that are used today for application integration.

This chapter includes the following topics:

- “Integration options” on page 8
- “API enablement” on page 14
- “Messaging and event streams” on page 19

2.1 Integration options

In this section, we introduce the technologies that play a vital role in any enterprise that is looking to modernize their approach to integration. We also look at the architectural patterns that can help to enable a more agile integration architecture (see "Agile integration" on page 3).

Application integration, in one form or another, has always been a necessary part of all but the simplest of enterprise software solutions. Today, REST APIs, messaging, and event streams represent the integration technologies of choice across different industry sectors. This is partly due to the popularity of the microservices architecture (see "Microservices, APIs, and microservices applications" on page 9). Figure 2-6 shows the different ways of communicating between microservices within an application.

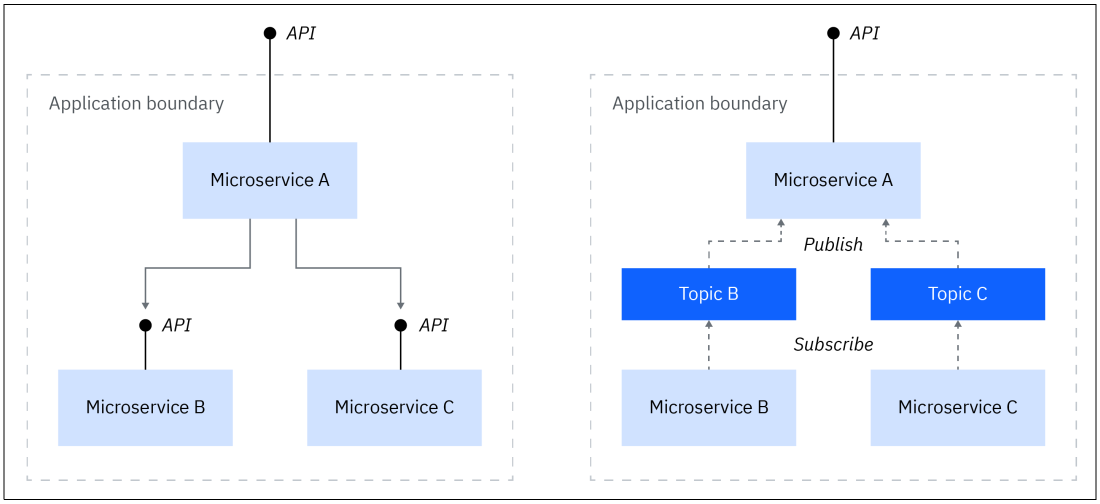


Figure 2-1 Communication between microservices within an application

The use of REST APIs offers a simple synchronous communication, but relies on both microservices being available to communicate at the same time. Within a microservices application, interaction patterns based on asynchronous communication might be preferred. For example, you might prefer event sourcing where a publish/subscribe model is used to enable a microservices component to remain up to date on changes that are happening to the data in another component.

Traditional IT solutions might typically exploit these and other technologies across an enterprise, including for integration with IBM Z assets. However, fundamental shifts around how they are best employed make it difficult to gain the full spectrum of possible benefits, unless you take a more holistic approach to modernization.

Taking a big-bang approach to modernization is likely to be impractical, costly, and disruptive. Rather, an incremental and carefully curated approach to the adoption of new integration options is the recommended approach for enterprise clients wanting to embark on a digital transformation mission.

Selecting the right integration technology to meet the requirements of a given project remains a critical early decision. But planning for the adoption of agile integration techniques is now an equally vital consideration, because it can bring these benefits: agility in response to unforeseen future enhancements, flexible deployment options, scalability and extensibility around changing non-functional requirements.

2.1.1 Microservices, APIs, 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 runtime platform, access method, security model, or programming language involved is not necessarily prescribed, as illustrated in Figure 2-2.

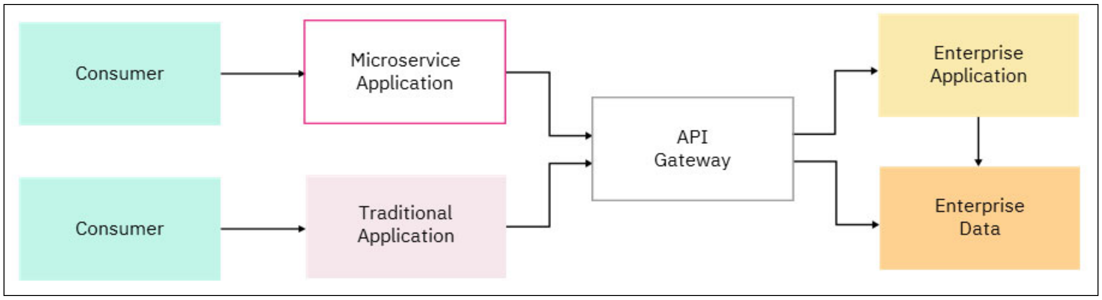


Figure 2-2 Combining a microservices architecture with APIs

A microservice application can be designed, developed, discovered, and externalized as 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 that is based on a microservices architecture. Where APIs and services are used to directly encapsulate SoR (System of Record) assets, for example IBM Z assets, their individual capabilities might be considered too fine-grained (or too “chatty”) for an engagement layer. In such cases, it makes sense to recast common sequences of API calls under a microservice application, externalized as a new API.

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 Clark, which is available online at:

http://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

The adoption of microservices encourages a fine-grained deployment of business function, and when applied with a considered approach, can deliver multiple benefits, including:

- Greater agility: They are small enough to be understood in isolation and changed independently.
- Elastic and independent scalability: Their resource usage can be tied to the business model.
- Discrete resilience: With suitable decoupling, changes to one microservice do not affect others at run time.

2.1.2 Containers and container orchestration

The era of the virtual machine delivered better value from hardware investments by ensuring that relatively light individual workloads could share the same underlying computing resources, thereby delivering a more efficient use of hardware capacity overall. In order to provide isolation, each workload could be built into a dedicated virtual machine, which included the full stack of application runtime and operating system. Software maintenance could also be performed more centrally, as virtual machine images could be rebuilt, versioned, and rolled-by a central team.

However, application consumption and reuse patterns demanded finer grain access and lifecycle of components through services. So, the virtual machine model came to feel somewhat heavy handed, and rebuilding entire machine images typically held back the agility.

Containerization refers to the refactoring, packaging, and running of discrete software components on a container technology runtime such as Docker. Most probably the containers are administered with a container orchestration technology such as Kubernetes. The Open Container Initiative[1] (OCI), a project of The Linux Foundation, defines open specifications for both container images ("image-spec") and container run times ("runtime-spec").

Container images are much more lightweight in comparison to a VM image. They take up less space on disk and memory, and can be started and stopped much more rapidly, because another entire operating system is not instantiated and started. Containers enable a more fundamental abstraction from the underlying infrastructure, providing a much more lightweight and portable virtualization model, as shown in Figure 2-3.

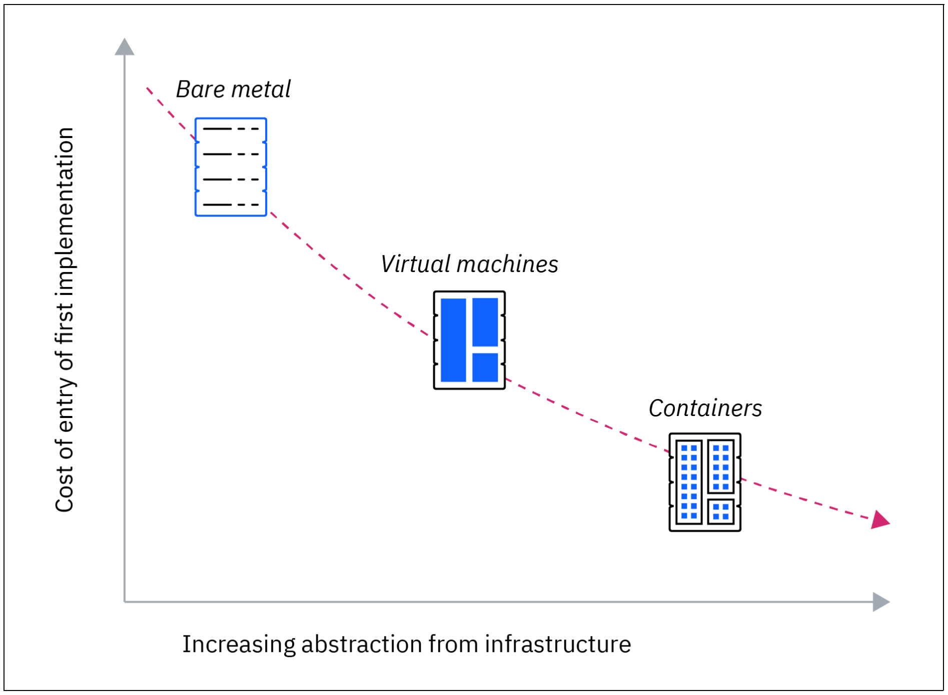

Bare metal
Virtual machines
Containers
Cost of entry of first implementation
Increasing abstraction from infrastructure

Figure 2-3 Abstraction from infrastructure

The adoption of containers offers an opportunity to modernize key business functions, and, with the right approach, can deliver a broad set of benefits including:

- **Agility and productivity**: accelerated development, improved consistency across environments, empowered autonomous teams that improve productivity and quality.

[1] https://www.opencontainers.org/

- **Fine-grained resilience**: independent deployment of highly available components to remove single points of failure.
- **Scalability and infrastructure optimization**: fine-grained dynamic scaling and maximized component/resource density to make best use of infrastructure resources.
- **Operational consistency**: homogeneous administration of heterogeneous components, reducing the range of skillsets required to operate the environments.
- **Component portability:** portability across nodes, environments, and clouds, ensuring choice when selecting platforms.

The drive toward a granular set of components — such as a decomposition of a monolithic application into microservices, combined with containerization — means that before long, there are very many more containers in play than virtual machines. Their propensity for simple provisioning and scaling soon further increases their proliferation within an enterprise organization, and so the question of control and management becomes critical to successful adoption.

A container orchestration platform provides a minimum framework for efficiently aligning resources with the containers, mechanisms to manage container lifecycle and scaling, load balancing across containers, routing between them, and control of how they are exposed beyond the container platform.

Figure 2-4 illustrates the fundamental differences between virtual machines and containers.

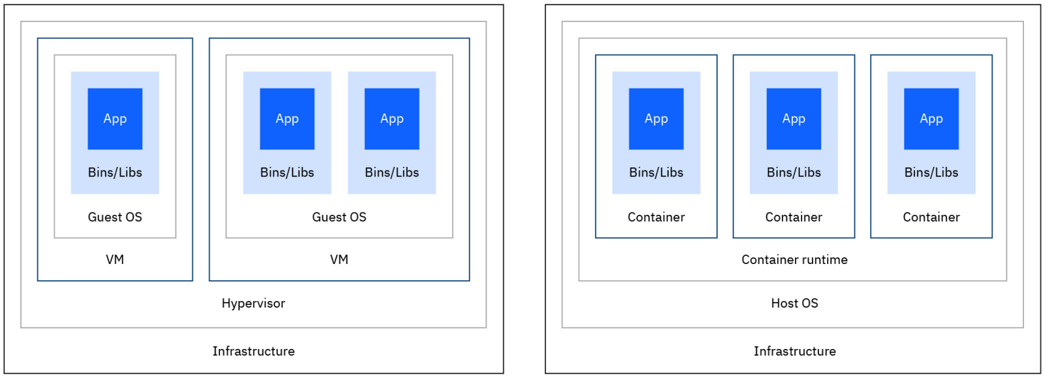


Figure 2-4 Difference between virtual machines and containers[2]

A key characteristic of a container is that it is small and fast because it uses some of the underlying host operating system's resources to run, rather than containing a whole and dedicated operating for each application, or group of applications. As such, many more containers can be placed on hardware than virtual machines, enabling a more efficient use of overall computing resource. Furthermore, their lightweight nature enables a radically different approach to how they are managed and scaled, which align perfectly with the needs of cloud-native applications.

The container platform standardizes almost all of the capabilities required of the enterprise system, rather than leaving those to bespoke, proprietary, or customized solution that might be highly aligned to a specific line of business, project, or team skill-base. This means more time can be spent on delivering functional value, less on non-transferable customizations of a given platform.

[2] Derived from https://www.docker.com/what-container

Note: For an in-depth analysis on the benefits of containerization, refer to the series of articles, *The true benefits of moving to containers* by Kim Clark and Callum Jackson available here:

https://developer.ibm.com/series/benefits-of-containers/

2.1.3 Agile integration architecture

In agile integration, the architectural patterns are tuned towards enabling the business to deliver changes to production robustly, and yet at high velocity, and of course with optimal use of resources. This plays out differently across the different integration capabilities of API management, application integration, and events and messaging. Each has changed in their own way to enable decentralized ownership, and a cloud-native infrastructure.

Delivery-focused integration architecture aims to improve deployment agility, using modern architectural and design practices such as API-led or event-driven integration with a focus on microservices applications. In the architectural space, each integration capability contributes differently to improving integration agility.

Integration is front and center in enabling the business agility required to innovate faster than the competition. An agile integration architecture, as shown in Figure 2-5, addresses this need by rethinking the traditional centralized approach, to integration based on people, architecture, and technology. It results in a decentralized, microservices-aligned, portable approach to integration.

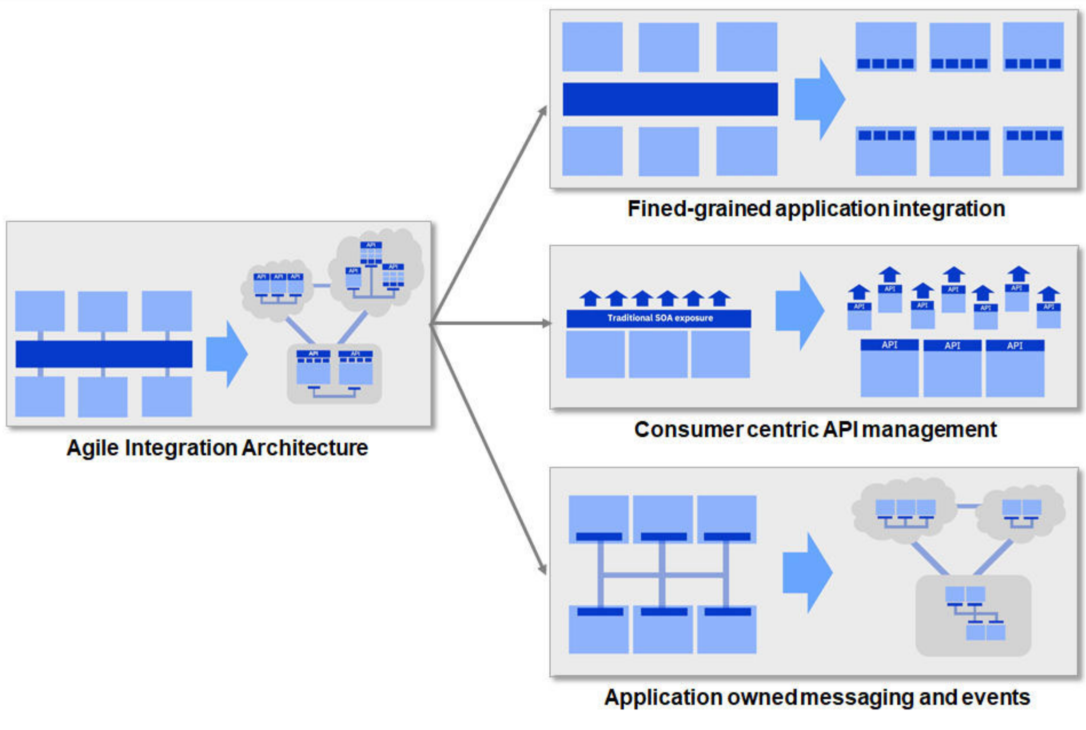

Fined-grained application integration
Traditional SOA exposure
API
Consumer centric API management
Agile Integration Architecture
Application owned messaging and events

Figure 2-5 Modern architectural and design patterns for agile integration

Each integration capability offers its own set of characteristics that are summarized in the following sections.

Fine-grained application integration

In “Microservices, APIs, and microservices applications” on page 9 we explored why microservices concepts have become popular in the application space. Therefore, we can quickly see how these principles can also be applied to the modernization of the integration architecture itself.

The centralized deployment of an *integration hub* or Enterprise Services Bus (ESB) pattern — where all integration components are deployed to a centralized ESB — has some benefits in terms of consistency and apparent simplicity and efficiency. However, it has been shown to introduce a bottleneck for projects. Furthermore, any deployment to the ESB runs a risk of destabilizing existing critical interfaces. Also, because the ESB is deployed on a single software instance, no individual project can choose to upgrade the version of the integration middleware to gain access to new features without impacting others.

An alternative approach is to break up the enterprise-wide ESB into smaller more manageable and dedicated pieces. Perhaps in some cases we can even get down to one runtime for each interface we expose. These fine-grained integration deployment patterns provide specialized, right-sized containers, allowing improved agility, scalability and resilience, and look very different from the centralized ESB pattern of the past.

For a detailed exploration of the decentralized ownership of integration components and the adoption of best practices, see the IBM Redbooks publication *Accelerating Modernization with Agile Integration,* SG24-8452 (`http://www.redbooks.ibm.com/abstracts/sg248452.html?Open`).

Consumer centric API management

An API-led integration strategy for connectivity between applications is now more or less taken as a given. API management allowed the ideas that were originally envisaged in SOA to mature. As a result, the standards around how interfaces are shared between applications, and ultimately between enterprises in an API economy have been refined and improved. A key lesson from this was the importance of creating and exposing APIs based on the needs of the API consumer.

APIs are now treated more like products than technical interfaces, and as such they need to be able to be marketed and potentially monetized. Socialization of APIs through slick developer portals that enable self-subscription and convenient ways to learn and test the interface are critical to the success of any API.

In today's multicloud world, it is also critical to be able to administer, catalog, and secure APIs from a single place even if they are exposed on many different cloud endpoints, further simplifying the consumer's experience.

The use of APIs for application integration is considered further in “API enablement” on page 14.

Application owned messaging and events

Asynchronous communication is perhaps more relevant than ever in today's multicloud world:

- Messaging, originally introduced to enable decoupled communication across disparate platforms, continues in that mission critical purpose, but now also takes up the same role in reliable communication across cloud boundaries.
- Events provide mechanisms to store an event history. This history provides an alternative source of information on data updates, enabling applications to selectively listen for notifications and build local data stores more suited to their needs.

However, it is no longer acceptable to have to wait on a highly specialized team to provision new asynchronous communication infrastructure. Teams need to be able to self-provision and configure queues and topics for immediate use as part of their event-driven integration projects.

Templatized and patternized mechanisms must be introduced to simplify provisioning tasks, and where possible provide them in as *in-situ*, multi-tenant managed services. The queue and topic configurations need to be application-owned so that developers can prototype and iterate on solutions rapidly.

Messaging and event streams for application integration is considered further in “Messaging and event streams” on page 19.

2.2 API enablement

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 have 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.

2.2.1 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. 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 reused 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 “REST and JSON” on page 16).
- A SOAP API is a web service that is made available as an API.

Figure 2-6 shows the co-existence of one solution that is based on 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 a System of Record (SoR).

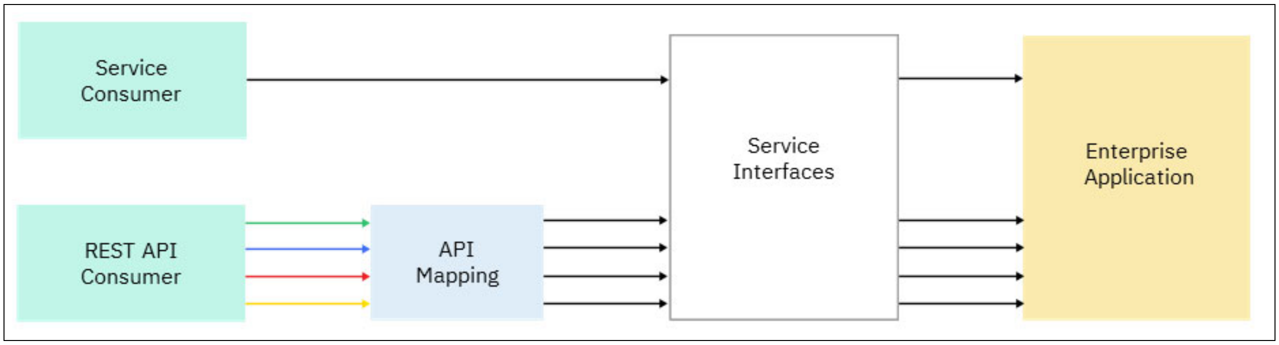


Figure 2-6 Concurrent consumption models for SoR assets as Services and APIs

A collection of independent services can be brought together under the auspices of an “API” facade. However, 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.

2.2.2 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.

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.

The HTTP protocol provides verbs such as GET, POST, PUT, and DELETE that are typically overlooked in an SOA-based solution. 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.

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.

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.

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

```
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 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 makes 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, 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 provides a standard way for defining an API.

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.

2.2.4 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.

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 on 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.

2.2.5 Advantages of REST 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.

The use of REST APIs offers the following additional 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.

2.3 Messaging and event streams

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.

In messaging, 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.

In *event streaming*, an event streaming platform provides facilities to publish and subscribe to streams of records, to store streams of records with fault toleration, and process streams of records in real-time. The two classes of application that typically exploit an event streaming platform are real-time streaming for data pipelines between applications, and real-time streaming applications that transform or react to streams of data.

At a high level, messaging and event streaming technology might appear to have overlapping capabilities. This is often due to the fact that they both can be used for the same core asynchronous interaction patterns. However, on a deeper review of the capabilities of each technology, it will become clear that they achieve these patterns in very different ways to serve different purposes. It is critical to select the right technology for the job.

Figure 2-7 illustrates the primary architectural patterns used in asynchronous integration solutions.

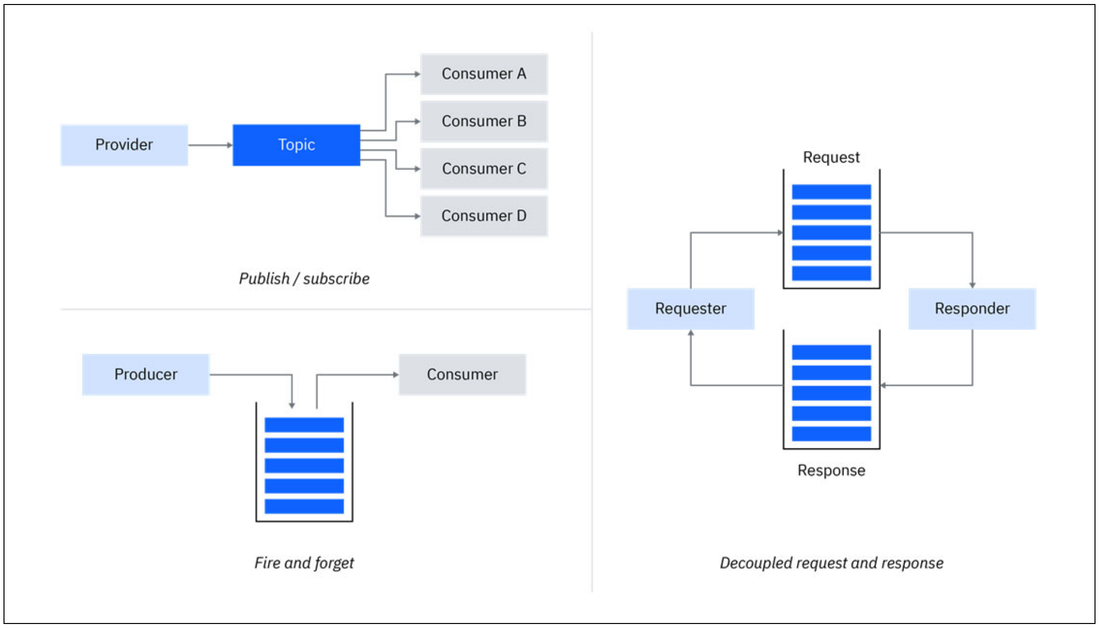


Figure 2-7 Asynchronous integration patterns

The different patterns are described here:

Fire and forget: A requesting application sends a message or event for processing to another application. The requesting application wants to assure the message or event has been sent, but in this scenario, it does not expect a response. The requesting application might in the future submit another request to determine the status of the original request. However this is a separate interaction, and indeed might not even be done via messaging or events.

Decoupled request/reply: In common with a synchronous call over a protocol such as HTTP, a requesting application sends request message/event to a target application and requires a response. However, in contrast to synchronous HTTP calls, the requesting application can choose to continue processing and be called back with a response when this becomes available. Messaging facilitates this, as the location and availability of the target application can be decoupled from the requesting application.

Publish/subscribe: The previous two interaction styles have a one-to-one relationship between the requesting and target applications. In other scenarios it is often desirable to send messages/events to multiple target applications. For instance, a message/event that is published with an airline's flight change might be of interest to the passenger mobile application, to the itinerary application, and many others also.

An event streaming technology could be used to implement a messaging solution, or a messaging provider used to implement an event solution. However, each case would be an anti-pattern. Each model facilitates the communication of data between systems, but the underlying capabilities and usage of the technology is different.

The increasing complexities of hybrid applications increasingly rely on a de-coupled, *event-driven architecture* where multiple cascading actions need to be taken upon one or more real-world events.

2.3.1 Messaging

Messaging enables a decoupled architecture where an intermediary is placed between two applications, systems, or services for communication.

This component is often referred to as a messaging provider. The messaging provider achieves decoupling by providing queues onto which applications can place messages for later retrieval by target applications. This asynchronous communication between the systems means that the applications no longer depend on each other's availability.

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.

Figure 2-8 shows a basic messaging interaction.

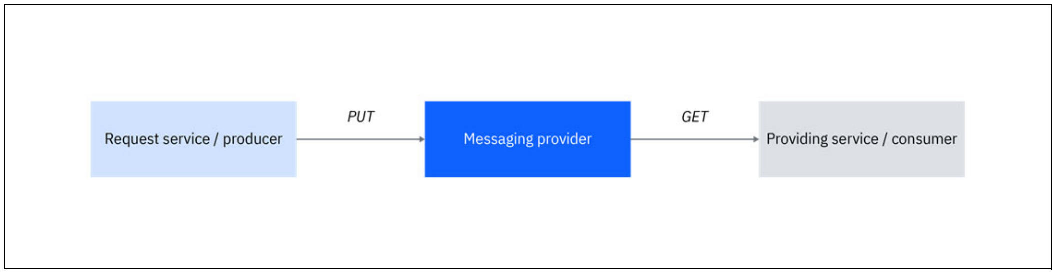


Figure 2-8 Basic messaging interaction

Messaging enables qualities of service like reliable delivery, enhanced security, and workload distribution. It is most commonly used to enable the following use cases:

- Transient data

  Data is stored until a consumer has processed the message, or it expires. The data does not need to be persisted longer than required, and it is actually beneficial from a system resource point of view that it does not.

- Request/reply

  While some messaging operations follow the fire-and-forget pattern, the most common use case is *request/reply*.

- Reliable delivery

  Messages can be delivered using a suitable level of reliability, according to specific delivery requirements. Examples include once-and-once-only delivery, assured delivery, and transactional PUT or GET behavior.

Some examples of industry-leading messaging providers include IBM MQ, ActiveMQ, and RabbitMQ. IBM MQ is used by a large number of mainframe clients to implement the preceding use cases with IBM Z applications.

Security standards for messaging

TLS provides security for messages while they are in transmit between two queue managers, or between a client application and a queue manager. However, if TLS alone is used, message data remains unencrypted when it resides on message queues.

Some messaging providers, for example IBM MQ, extend TLS support by also providing end-to-end data protection of messages, by enabling signing and encryption at the message level. This guarantees that message data has not been modified between when it is originally placed on a queue and when it is retrieved, and also ensures that the message data is private.

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 highly scalable solutions.
- Enables buffering of workloads between applications, such that one application does not overload another with requests.
- Supports one-to-one and one-to-many publishing of messages.
- Ensures that business-critical information is not lost, and is delivered to the recipient once and once only.

2.3.2 Event streams

Event streams are based on the publish/subscribe integration pattern, but with the capability to significantly scale both the number of events occurring and subscribers listening for those events. Event streaming technology allows an enterprise to collect, store, and distribute events across all their applications at massive scale.

The stateless nature of many consumers of events introduces the need to retain an event history. An event stream typically represents a collection of events that are generated by one or more systems, or applications, and need to be processed by one or more other applications or systems.

Figure 2-9 shows an event stream with multiple consumers.

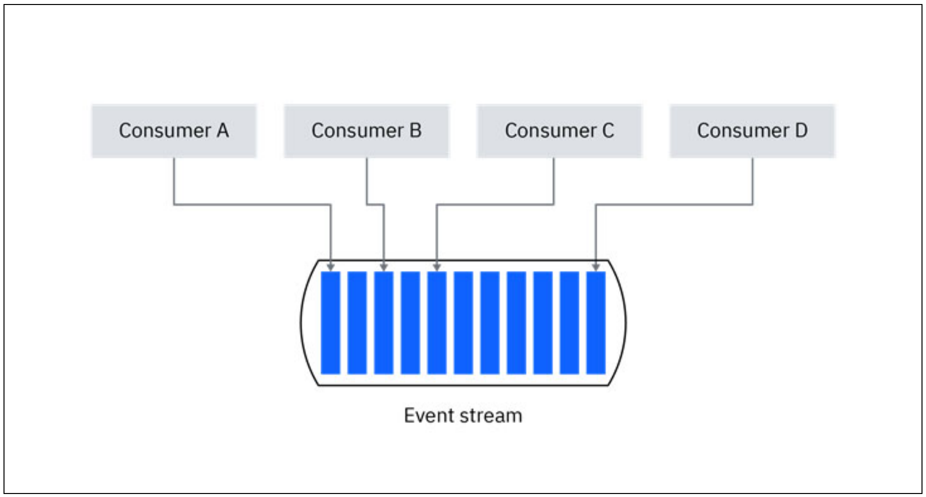


Figure 2-9 Event stream consumers

Event streams lend themselves to an alternative set of asynchronous integration use cases compared with messaging, albeit with some overlap around publish/subscribe use cases:

- Stream history

  When retrieving events, often consumers are interested in the historical events, not only the most recent. There are many instances when this is valuable, such as retrieving the historical trends of a system's availability. Therefore, every event needs to be appended, and made available to consumers, and depending on the configuration, a certain number or volume of events will be stored prior to removal.

- Scalable subscription

  An event stream can be consumed by large numbers of subscribers, with limited performance impact as the number of subscriptions is increased, because an event stream maintains only one copy of each event.

- Immutable data

  When an event is placed in the stream history, it cannot be changed or removed. It can be considered *immutable* data. This enables consumers to rely on the assumption of consistent replay, and reduces the complexity of replicating the data from a consistency point of view.

Consumers of events don't necessarily want to read all events that pass through the event stream. So publishers of events specify a *topic* and subscribers listen only to the topic(s) they are interested in.

There are two key components to any event streaming technology: a server that stores events and manages the topics, and a client that allows applications to interact as a provider or consumer of events.

Several technologies provide event streaming capabilities. The market is leaning toward Apache Kafka as the de facto standard. Apache Kafka is an open source project that was originally created by LinkedIn and donated to the community in 2011. Some vendors provide commercially supported versions of Kafka and the IBM offering is Event Streams.

Note: IBM is also a key contributor to the Apache Kafka open source project as committers.

Using event streaming with event-driven applications introduces new integration patterns that gather events from multiple sources. The events might need to be extracted from mainframe databases and applications, analytics systems, SaaS systems, or other cloud-based applications.

Security standards for event streams

In much the same way as messaging, event stream communications secure data both on the wire and at rest. Over the wire, protocols such as TLS can be configured and, at rest, disk encryption secures data on the event streams servers.

In addition, authorization to topics is enforced upon the clients, both providers and consumers, by Access Control Lists, with specific implementation dependent on the product employed.

Advantages of event streams

The use of event streams offers the following advantages:

- Event streaming technology is designed to handle millions of events a second.
- Increases in the number of subscribers to a topic has a minimal impact on performance.
- An event history is maintained which means that consumers of events do not need to maintain event state.

3

Hybrid integration architecture considerations

Integration patterns between cloud-hosted services and mainframe applications can be considered from different perspectives, including presentation, application, data, and security integration. The goal of these patterns is to enable any application to consume Systems of Record applications or data, quickly, simply and securely.

In this chapter, we review the key considerations and architectural components needed for any cloud-native application to reuse mainframe applications and data.

This chapter includes the following topics:

3.1 Hybrid integration architecture

An effective integration architecture must be sufficiently flexible to encompass applications and services that span public and private clouds, as well as traditional enterprise systems. Historically, there was a clear boundary of users, applications and hosting environments that were either within, or beyond the enterprise. Over time, more complex requirements were placed on the integration components, to support newer models for self-service, monetized billing and event streaming.

We are now in a world where third partner applications sit beyond traditional enterprise boundaries; increasingly, an organization's own applications reside in public cloud environments and, indeed, may move over the lifetime of the application. This places new requirements on integration, in which services can be discovered and consumed using REST APIs, messaging and event-driven architectures.

In this section, we look at the major components of a hybrid integration architecture (see Figure 3-1).

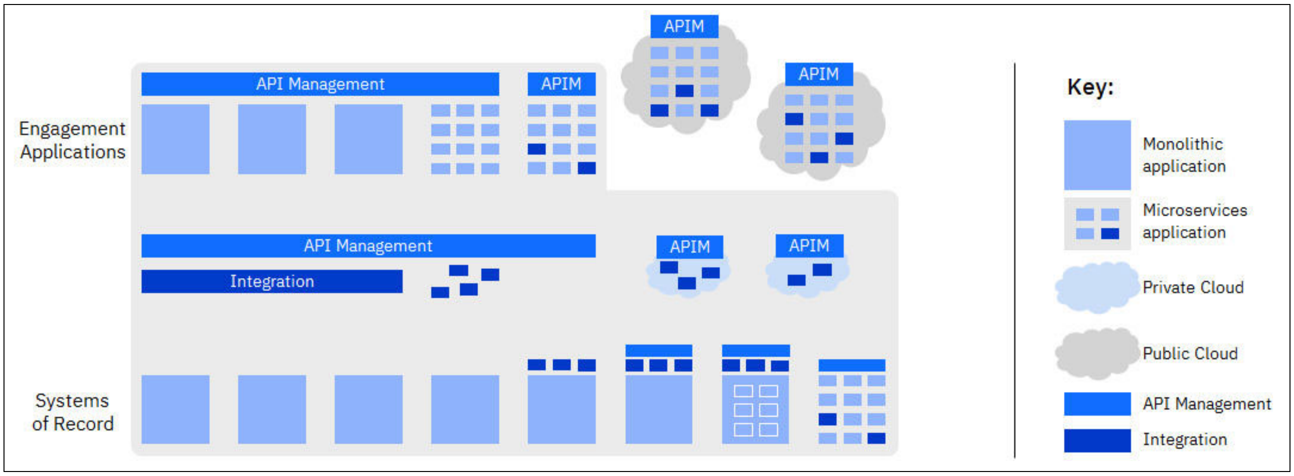


Figure 3-1 Hybrid integration architecture

Figure 3-1 shows a simplified representation of the different types of application, integration layers and infrastructure (on-premises and cloud) used by enterprises today.

One of the pervasive industry trends is the adoption of a cloud native approach, where applications and data services are designed for container platforms and hosted in cloud infrastructures. Figure 3-1 shows how cloud native infrastructures can sit alongside traditional infrastructure. It is important to note that organizations will almost certainly have a hybrid infrastructure and so the integration architecture needs to work for both environments.

The major components of a 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 the IBM Redbooks publication *Accelerating Modernization with Agile Integration,* SG24-8452:
http://www.redbooks.ibm.com/abstracts/sg248452.html?Open

3.1.1 Engagement applications

Engagement 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.

Figure 3-1 on page 26 illustrates this evolution of the architecture and deployment infrastructure of engagement applications. Historically, engagement applications have been monolithic in nature, with each monolith supporting a different channel. However, these applications are increasingly being refactored to exploit microservices architecture patterns. Alongside this change in application architecture, is a move to deploy these microservices applications in a cloud infrastructure.

3.1.2 Systems of Record

When engagement applications provide information to consumers, it is vital that they can access current and accurate data, irrespective of the engagement channel. For example, a cloud-hosted, third-party application, must provide the same result as queries that are made using traditional web or call center 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 the 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, such as API-, SOAP-, or message-based.

The SoR applications are most commonly implemented as monolithic applications, meaning that they are developed to a set of standards, often in a single language such as COBOL or Java, and 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 include messaging, SOAP web services, and increasingly REST APIs. These monolithic SoR applications and data platforms are shown on the lower left side of Figure 3-1 on page 26.

Increasingly, SoR applications are evolving to become more componentized and to adopt loosely coupled architectures. This approach paves the way for them to become full microservices applications. In that case, each component performs a single function, can be developed in a language that is best suited to the function, and can be scaled independently from the others. The communications models for these applications are almost always based on a combination of messaging and REST APIs. This evolution of the SoRs is shown on the lower right side of Figure 3-1 on page 26.

3.1.3 API management

As organizations productize their business functions through REST APIs, it is essential for these APIs to be managed so that only approved individuals or applications are able to discover and consume the APIs. API management simplifies 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.

API management enables access to APIs and essential services, such as security, governance, monitoring, and analytics. For example, metering API invocations and enablement of rate limiting and charging.

API management can also provide the underlying technology to support simple message-format translation and version and change management. It can be deployed in the same physical or virtual server as a security gateway, depending on whether the service is available internally within an organization or beyond.

API management can be implemented to address different requirements. Figure 3-1 on page 26 shows multiple logical instances of API management, which is associated with engagement applications, integration components, and SoRs.

The ability to deploy multiple API management instances has advantages that are related to the different non-functional requirements that are associated to the underlying component. For instance, managed APIs that expose functions from the engagement applications, must provide additional levels of protection associated with queries from outside the enterprise. This is in contrast to API management deployments alongside a SoR, that must provide enhanced API discovery and simplified access to core application and data assets.

However, it is equally easy to see how this might quickly get out of hand. Does each gateway need its own API manager and developer portal? It would be painful if each team had to manage their own API management infrastructure alongside their implementation.

In the case of API management, the important thing to decentralize is the ownership over the ability to administer APIs (the provider perspective) and the ability to discover, subscribe, and use them (the consumer's perspective).

A good API management solution should provide strong multi-tenant capabilities so each API is defined, managed, and administered only by the team that created it. An API gateway should be able to expose APIs from multiple separate implementations and provide good isolation. For example, managing heavy traffic through one API (and perhaps limiting it in relation to the policy that is defined for its consumers) should have no effect on the performance characteristics of any other APIs also passing through the gateway concurrently.

Consider the following questions when you decide what role API management 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.

- Do you need to de-centralize ownership of APIs?

  The chosen API management solution should offer strong support for multi-tenancy, and distributed API ownership and deployment.

3.1.4 Integration

An integration component provides a means to connect a service requester with a service provider. Each invocation from a service requester (for example, an engagement application) results in one or many invocations to a service provider (for example, a mainframe SoR). Integration components often handle request composition, event handling, data synchronization, and adapter-based technology integration.

An integration component commonly supports the request protocols and data formats that are associated with RESTful APIs and cloud services and augments the capabilities of the API management components.

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

Traditionally, integration was seen to be a single logical component, or layer, with the responsibility for enterprise integration falling to a single team or domain within an enterprise. More recently, integration has proven most effective when it is embedded as discrete components alongside the applications that require it. This is illustrated in Figure 3-1 on page 26. The left side shows a single, logical integration layer and on the right side, there are multiple integration components, which are deployed where they are needed.

As an example, some integration components can be deployed as part of a microservices-based engagement application hosted in a public cloud; while other integration components can be created and deployed alongside the SoRs, perhaps to provide a façade for a bespoke application.

Consider the following questions when you decide what role integration components should play in a hybrid integration architecture:

- How many different types of service requesters and service providers are needed in the enterprise?

  The value of an integration component 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 component plays a crucial role in facilitating application integration. An integration component also makes it easier to introduce systems into the application integration architecture if business mergers and acquisitions occur.

- Does the engagement 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 component can provide correlation for asynchronous invocations by using a messaging engine.

- Is a centralized integration layer used in the application integration architecture today?

Probably the most compelling reason for the use of a centralized 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 engagement application requests are subject to the same governance policies.

If there is a centralized integration layer today and there is nothing major on the roadmap for the applications that use the integration layer, it does not make sense to decompose the integration layer into multiple decentralized components.

- Do you need to de-centralize ownership of integration components?

  Consider the decentralization of integration components where the autonomy it brings is required by the organization. The chosen solution should offer strong support for multi-tenancy, and distributed integration component ownership and deployment.

3.2 Integration patterns with IBM Z

The components of the integration architecture do not sit in isolation but are connected by the flow of requests between engagement applications and SoRs. In this section we look at the common integration patterns used with IBM Z:

- Aggregation
- Direct API call
- Call-out
- Event stream

3.2.1 Aggregation pattern

One of the most common integration patterns used with IBM Z is when an application issues a request that requires coordinated access to multiple SoRs. This is the aggregation pattern shown in Figure 3-2.

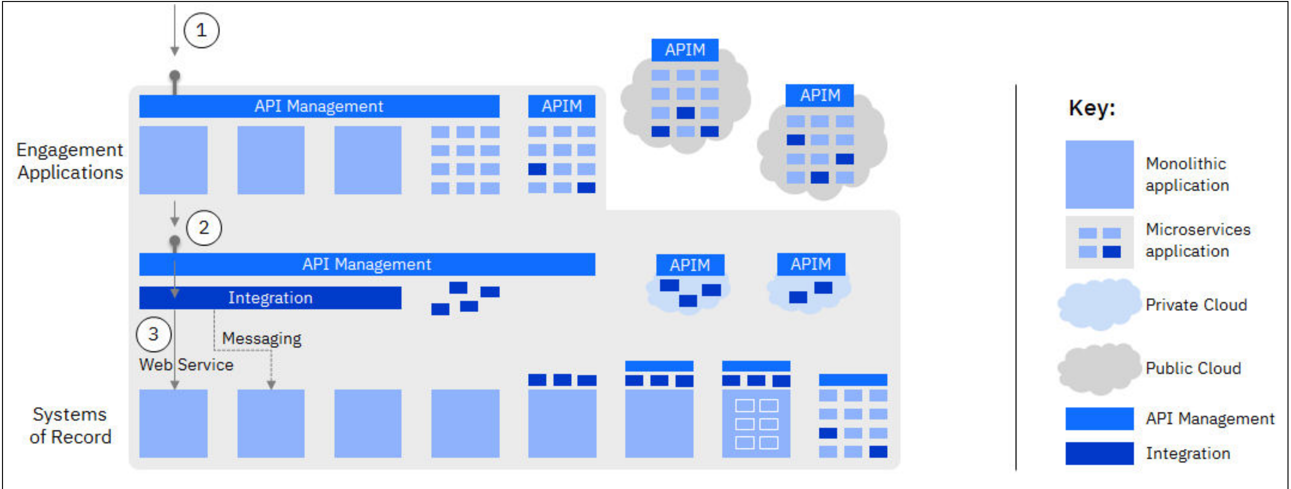


Figure 3-2 Aggregation pattern

Figure 3-2 shows the following sequence of steps:

1. A request is made to an externally facing API, that is managed and secured with API management. The API management layer directs the request to the relevant engagement application.
2. To satisfy the request, the engagement application needs access to additional services, which are provided via a single internally managed API. This API is available to internal applications only.
3. The services required by the engagement application reside on two SoRs, one accessible by a web service and the other accessible by messaging. The managed internal API calls a service in the integration layer that handles the data and protocol transformation, and issues the requests to the SoRs.

Not shown in Figure 3-2 on page 30 is the response leg, where the integration layer correlates the requests to the SoRs, aggregates the SoR responses and provides a single response to the engagement application via the managed API.

3.2.2 Direct API pattern

As integration components become embedded as part of an application domain, it opens the possibility for more efficient, direct connectivity to services that naturally fulfil a request, requiring no enrichment or modification. This is the direct API pattern shown in Figure 3-3.

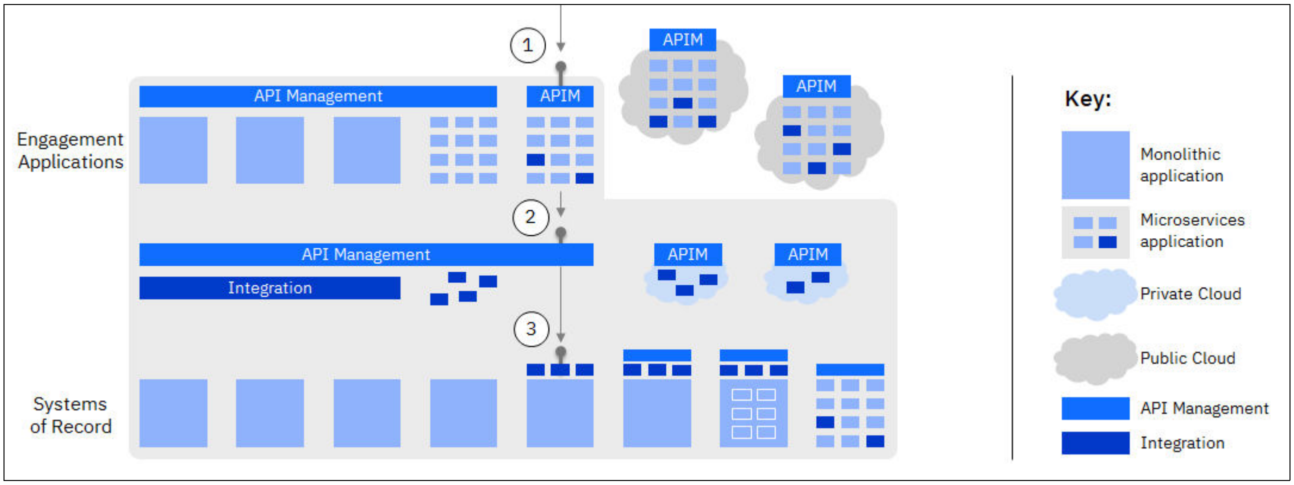


Figure 3-3 Direct API pattern

Figure 3-3 shows the following sequence of steps:

1. A request is made to an externally facing API, that is managed and secured with API management. The API management layer directs the request to the relevant engagement application.
2. The engagement application, comprised of a set of microservices, requires access to a service hosted in the SoR. This is accessible via a managed API, accessible to internal applications only.
3. In this pattern, an integration component has been co-located with the SoR to provide a REST API that can be consumed directly by the API management layer. This speeds up the delivery of new services because no prototol switching is necessary. And the SoR

team is responsible for the creation and deployment of the REST API (they do not need to rely on a centralized integration team).

Not shown in Figure 3-3 on page 31 is the response leg, where the integration component provides a response to the engagement application via the managed API.

3.2.3 Call-out pattern

Embedding integration components alongside the SoR has opened the possibility of calling APIs from these applications and augmenting them with services that may reside in other SoRs or cloud applications. This is the call-out pattern shown in Figure 3-4.

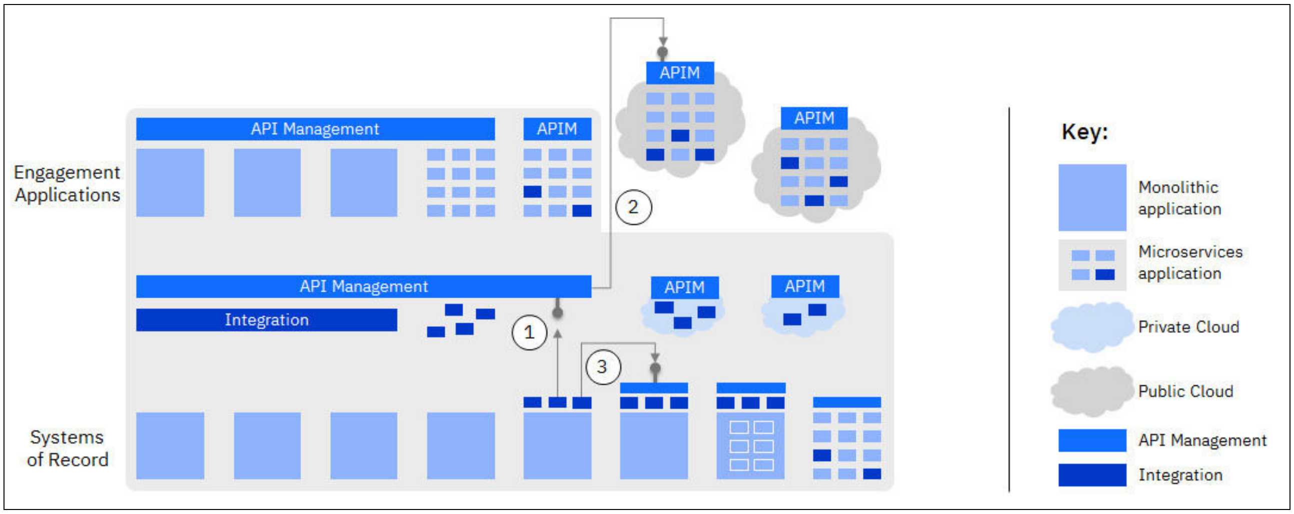


Figure 3-4 Call-out pattern

In this pattern, integration components are co-located with the SoR to provide a REST API call-out capability. In this example, the SoR application requires access to two services; a public cloud service, and an on-premise service provided by another SoR.

Figure 3-4 shows the following sequence of steps:

1. An initial request is made to the API management layer which manages calls to external services.
2. The API management layer forwards the request to the public cloud service.
3. The SoR application next calls a managed API that is co-located with another API-enabled SoR application. In spite of both SoR applications being implemented differently and independently, they are able to communicate directly via standard REST APIs.

Not shown in Figure 3-4 are the response legs, where the SoR application receives responses from the called APIs.

3.2.4 Event stream pattern

Sometimes it is necessary to disseminate information to multiple systems simultaneously. This can be enabled using an event-driven architecture in which event streams are used to publish events which are then consumed by subscribed consumers. This is the event stream pattern shown in Figure 3-5.

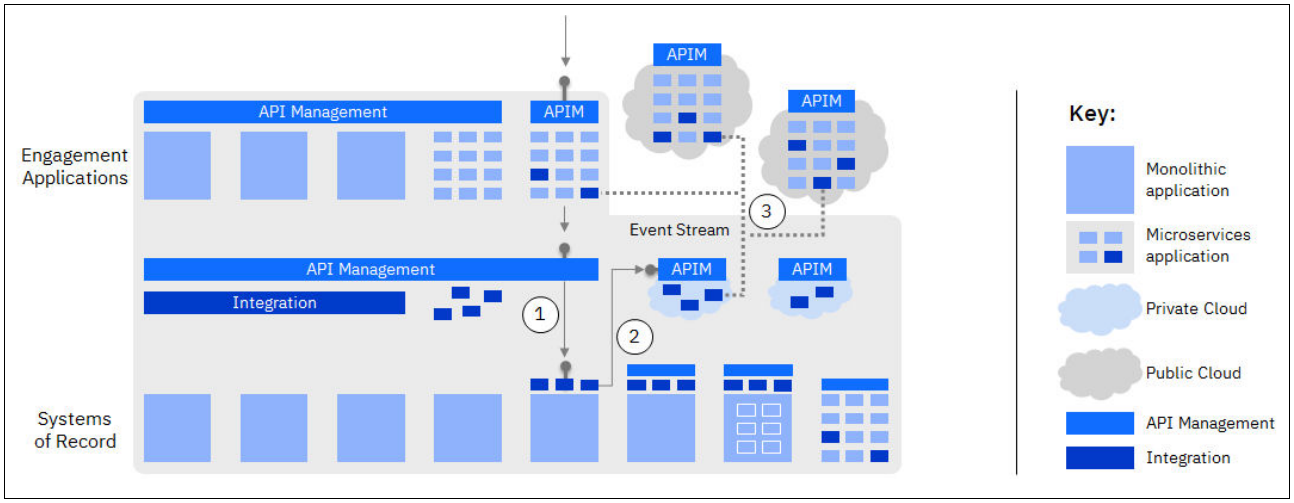


Figure 3-5 Event stream pattern

In this pattern, an event stream representing changes to customer records is generated by a mainframe SoR application and consumed by a number of cloud and on-premise applications.

Figure 3-5 shows the following sequence of steps:

1. An initial request is made to the API management layer which then calls the SoR REST API (as shown in the “Direct API pattern” on page 31).
2. The SoR application calls a private cloud hosted integration service via a managed API.
3. The integration service publishes the updates from the SoR application, for example an address change, to an event stream. The event stream is then consumed by a number of internal and external consumers.

4

Hybrid integration solutions for IBM Z

This chapter describes the main IBM solutions and products that can be used to integrate with IBM Z applications in a hybrid cloud infrastructure. We then look at specific considerations for the main z/OS subsystems. And we also introduce the open source project Zowe which offers a set of system APIs and a command-line interface that allows developers and system administrators to interact with IBM Z in a cloud-native way.

This chapter includes the following topics:

4.1 IBM integration solutions

This section describes the main solutions and products that can be used to integrate with IBM Z applications in a hybrid cloud infrastructure:

- 4.1.1, “IBM z/OS Connect Enterprise Edition” on page 36
- 4.1.2, “IBM API Connect” on page 41
- 4.1.3, “IBM DataPower Gateway” on page 44
- 4.1.4, “IBM App Connect” on page 45
- 4.1.5, “IBM MQ” on page 47
- 4.1.6, “IBM Event Streams” on page 49
- 4.1.7, “IBM Cloud Pak for Integration” on page 49

We provide an overview of each solution and guidance on when to use each one.

4.1.1 IBM z/OS Connect Enterprise Edition

IBM z/OS Connect Enterprise Edition (z/OS Connect EE) provides a common entry point 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. APIs 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 CICS, IMS, and other z/OS applications to call RESTful APIs through z/OS Connect EE. This capability is referred to as the *API requester support*.

API provider

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

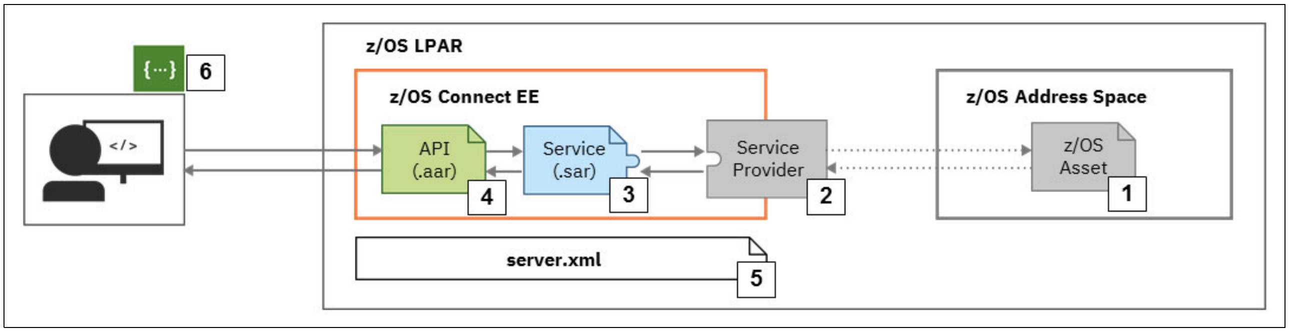


Figure 4-1 z/OS Connect EE API provider

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 , “z/OS Connect EE with CICS” on page 51)

   – An IMS service provider for connecting to IMS (see 4.2.2, “IMS” on page 53)

   – A REST Client service provider for connecting to a REST service (HTTP/JSON endpoint), for example, a Db2 REST service (see , “z/OS Connect EE with Db2 REST services” on page 56)

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

   – An IBM MQ service provider for putting or getting messages from an IBM MQ queue (see , “z/OS Connect EE with IBM MQ” on page 58).

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

   – IBM 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 (DVM) 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.

     For more information on DVM see the IBM Redbooks publication *Accelerating Digital Transformation on Z Using Data Virtualization*, REDP-5523 (`http://www.redbooks.ibm.com/abstracts/redp5523.html?Open`).

   – 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. You can also define multiple HTTP status codes for an API operation. Rules can be defined that determine the status code and unique response mapping, that are returned in the HTTP response.

   Behind each API operation, 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).

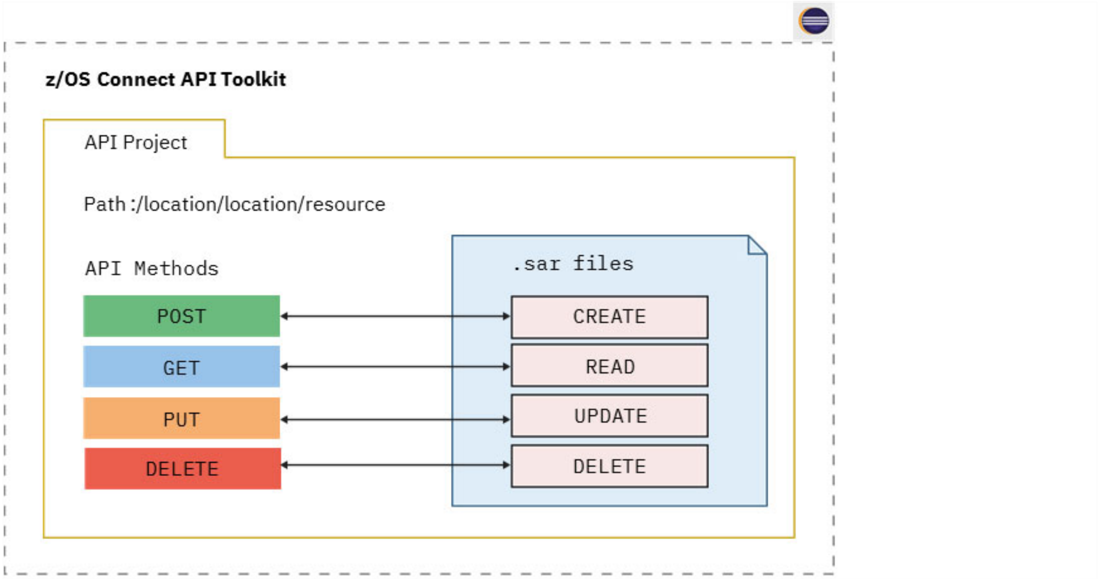


Figure 4-2 z/OS Connect EE API mapping

   Figure 4-2 shows the relationship between API operations 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. The `server.xml` configuration file defines the z/OS Connect EE feature, locations for API and service archive files, security configuration, policy rules, and other configuration elements.

   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.

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 4.1.2, “IBM API Connect” on page 41).

See 5.1, “Implement Open Banking APIs with z/OS Connect EE” on page 64 for an example scenario that features the use of the API provider support of z/OS Connect EE.

API requester

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

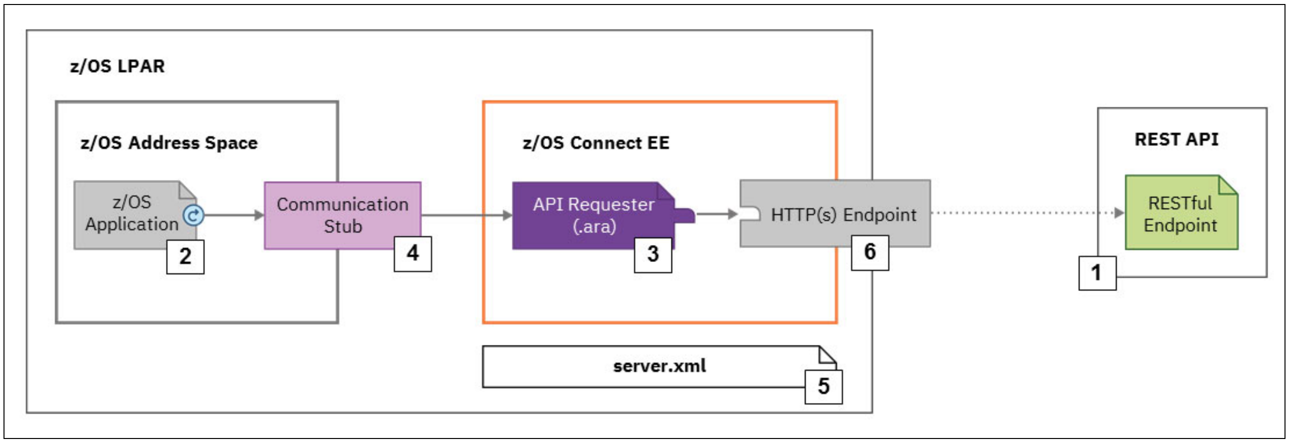


Figure 4-3 z/OS Connect EE API requester

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 z/OS 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 with the Liberty `server.xml` file. The server.xml configuration file defines the z/OS Connect EE feature, location for API requester archive files, security configuration, and other configuration elements

6. HTTP(S) endpoint

   The HTTP(S) endpoint of the REST API, which is configured in the `server.xml` file. The connection to the external API provider can be secured with TLS and z/OS Connect EE can be configured to send a security token such as a JWT to the RESTful API.

See 5.2, “Call out to external services using z/OS Connect EE” on page 67 for an example scenario that features the use of the API requester support of z/OS Connect EE.

z/OS Connect EE runtime

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 System Authorization Facility (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 propagate the distributed user ID or mapped RACF user ID onto the SoR (for example, CICS, IMS or Db2). 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 SAF authorization, SMF (System Management Facility) activity recording, and logging JSON payloads.

The interceptor framework is used by other vendors to integrate their solutions with z/OS Connect EE, for example:

- IBM OMEGAMON® for JVM on z/OS can be used to monitor the status of a z/OS Connect EE API workload.
- IBM Z Common Data Provider can be used to stream z/OS Connect EE audit data to an analytics platform such as Elasticsearch, Apache Hadoop or Splunk.
- AppDynamics can be used to track API requests processed by z/OS Connect EE.

Note: For more information on monitoring a z/OS Connect EE API workload, see `https://developer.ibm.com/mainframe/docs/managing-api-workloads/`

You can also write your own interceptors that implement the z/OS Connect EE **`com.ibm.zosconnect.spi.Interceptor`** SPI.

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 (including CICS and IMS applications) to call external REST APIs
- Enable interoperability between z/OS Connect EE and API solutions for management, monitoring, transaction tracking, and analytics.
- 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 Developer:

https://ibm.biz/zosconnectdc

4.1.2 IBM API Connect

IBM API Connect is a comprehensive platform from IBM for managing the API lifecycle which has two main focuses: the first is providing best in class API management tooling, and the second is having a cloud native solution. This allows users to create, manage, and secure applications that are deployed across a variety of on-premises (including IBM Z) and cloud environments.

The key phases in the API lifecycle are shown in Figure 4-4.

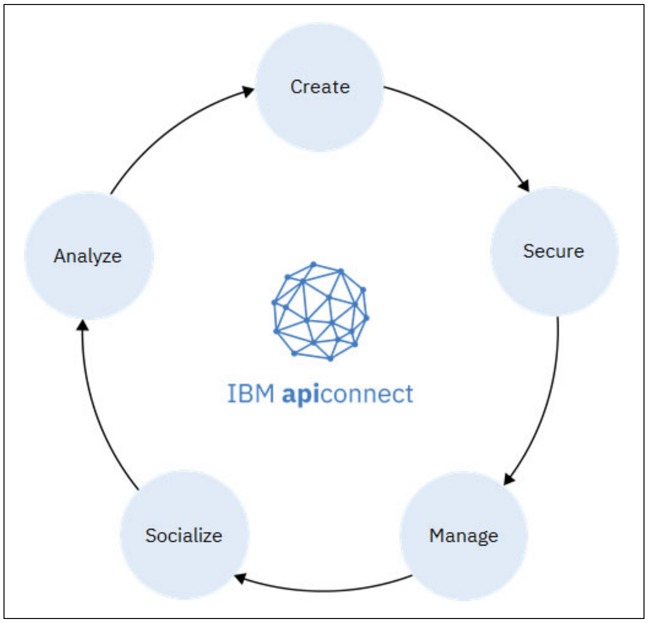


Figure 4-4 API lifecycle steps

The phases of the API lifecycle are as follows:

- **Create**: Develop and write API definitions from an API development environment, eventually bundling these APIs into consumable products, and deploying them to production environments.
- **Secure**: Leverage the best-in-class API gateway, gateway policies, and more, to manage access to APIs and back-end systems, including IBM Z subsystems. The API gateway is

based on the IBM DataPower® Gateway (see 4.1.3, “IBM DataPower Gateway” on page 44).

- **Manage**: Governance structures are built into the entire API lifecycle, from managing the view/edit permissions of APIs and Products being deployed, to managing what application developers can view and subscribe to when APIs are deployed.
- **Socialize**: Leverage an advanced Developer Portal that streamlines the onboarding process of application developers, and can be completely customized to an organization's marketing standards.
- **Analyze**: Developers and Product Managers are given real-time analytics on API traffic patterns, latency, consumption, and more, so that they can make data driven insights into their API initiatives.

API Connect deployment options

IBM API Connect includes four major components: the API Manager, API Analytics, Developer Portal, and the API gateway. These four components can be deployed in a variety of hybrid and multicloud topologies. The infrastructure can either be deployed and managed by an IBM team in an IBM Cloud environment, or it can be deployed and managed by customers in their own dedicated environment or third-party cloud.

In addition to the above components there is a Cloud Manager to manage the API Connect topology and a developer toolkit for offline API definition.

IBM API Connect is available as a separate product or as part of the IBM Cloud Pak™ for Integration (see 4.1.7, “IBM Cloud Pak for Integration” on page 49).

How IBM API Connect works with z/OS Connect EE

API Connect can be used to secure and manage 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 4-5.

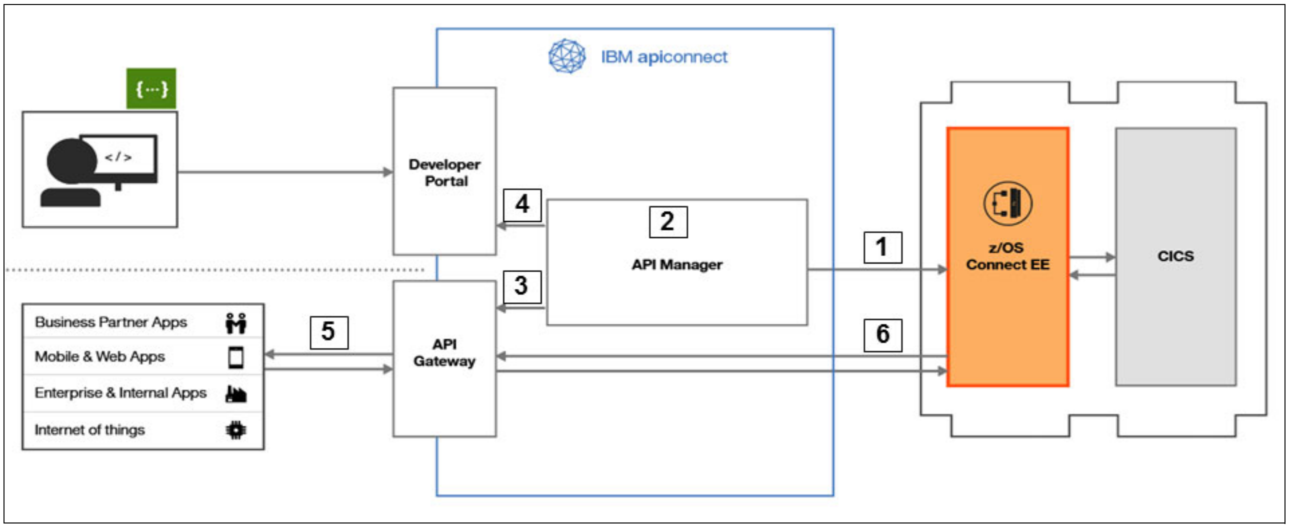


Figure 4-5 Securing and managing APIs with API Connect

The following steps are shown in Figure 4-5:

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 subscribe to 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 REST API that is hosted by z/OS Connect EE.

See 5.3, “Build a managed API framework using API Connect” on page 69 for an example scenario that features the use of API Connect with z/OS Connect EE.

When to use IBM API Connect

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

- Extend the value of your mainframe assets by socializing 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.
- Adapt the interface of a mainframe service by performing simple or technical transformation, for example, converting REST/JSON to SOAP/XML.
- 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 Developer:

https://developer.ibm.com/apiconnect/new/

4.1.3 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 4-6.

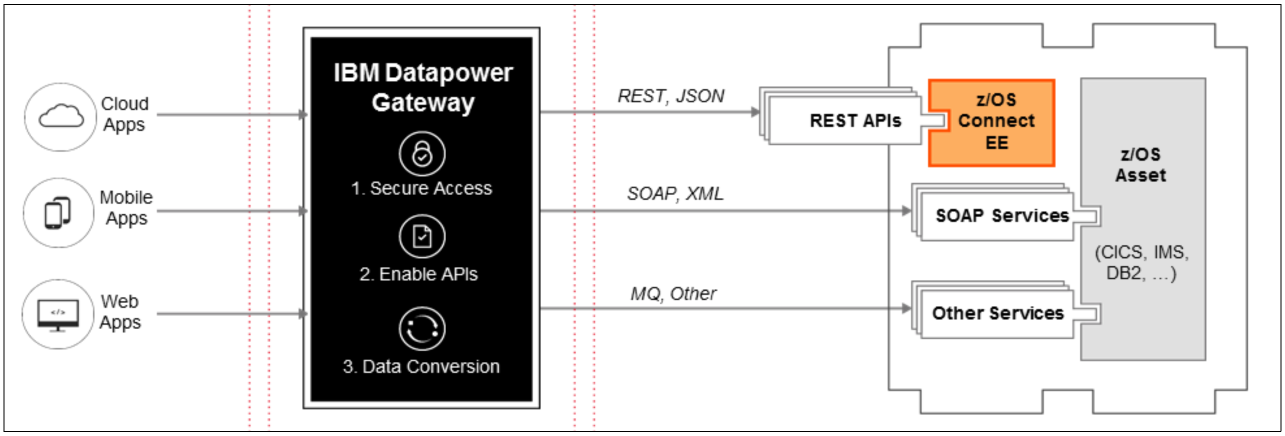


Figure 4-6 IBM DataPower Gateway

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

- 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 and authorization security standards, including SAML, OAuth 2.0, OpenID Connect and JWT
  - Auditing
  - Threat protection for XML and JSON
  - Message validation and filtering
  - Centralized management and monitoring point
  - Traffic control and rate limiting
- IBM DataPower is used as the runtime for the API gateway component of IBM API Connect. It enables secure API access and traffic management.
- IBM DataPower supports a wide range of protocols and data transformation capabilities, for example, from XML to byte array structures.

DataPower deployment options

The DataPower Gateway is available in physical, virtual, cloud, Linux, and container form factors.

> **Note:** 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.

IBM DataPower is available as a separate product or as part of the IBM Cloud Pak for Integration (see 4.1.7, “IBM Cloud Pak for Integration” on page 49).

When to use an IBM DataPower Gateway

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

- Protect mainframe 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 Developer:

https://developer.ibm.com/datapower/

4.1.4 IBM App Connect

IBM App Connect provides market-leading application integration, enabling synchronous and event-driven integrations that provide extensive adaptation to on-premises and cloud-based applications.

Figure 4-7 shows the main components of IBM App Connect and how they interact.

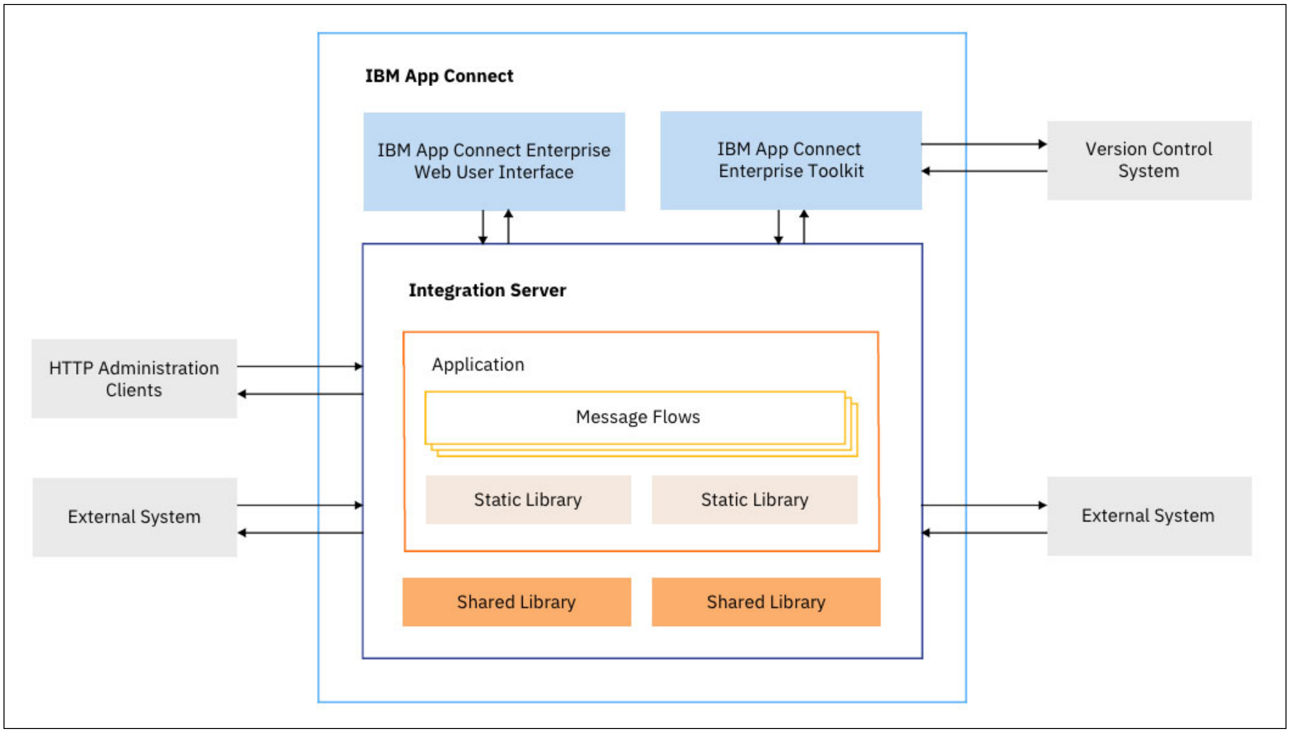


Figure 4-7 IBM App Connect

App Connect supports a non-coding approach to integration, building message flows, and exposing them as RESTful APIs without having to be an API development expert. App Connect allows you to orchestrate calls across multiple applications, including mainframe applications (see , “How IBM App Connect works with z/OS Connect EE” on page 46).

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.

For integration design, App Connect features include a desktop user interface coupled with browser-based tooling, which brings together the teams that own and manage the data with teams that have the context to apply it. Digital businesses rely on data that is delivered in the right context, to the right client touch point, at the required time.

The tooling provides new integrated tooling experiences for a spectrum of users across the digital enterprise:

- The core IT teams that manage the key systems and packaged applications
- Knowledge workers and line-of-business integrators
- Integration specialists that tackle more detailed and challenging requirements

App Connect provides connectivity options across cloud service applications, cloud platforms, and existing on-premises applications. It provides an extensive set of connectors for packaged applications and other assets that include:

- Customer relationship management (CRM) systems
- Enterprise resource planning (ERP) systems
- Files
- Databases
- Messaging systems
- Mainframe applications

App Connect is a proven solution for handling the transformation of thousands of messages per second. Message parsing in App Connect is based on a highly optimized engine that minimizes memory usage and optimizes CPU performance.

How IBM App Connect works with z/OS Connect EE

Sometimes a business function requires multiple calls to different applications. Figure 4-8 shows how App Connect can be used to aggregate and orchestrate calls to different z/OS Connect EE APIs.

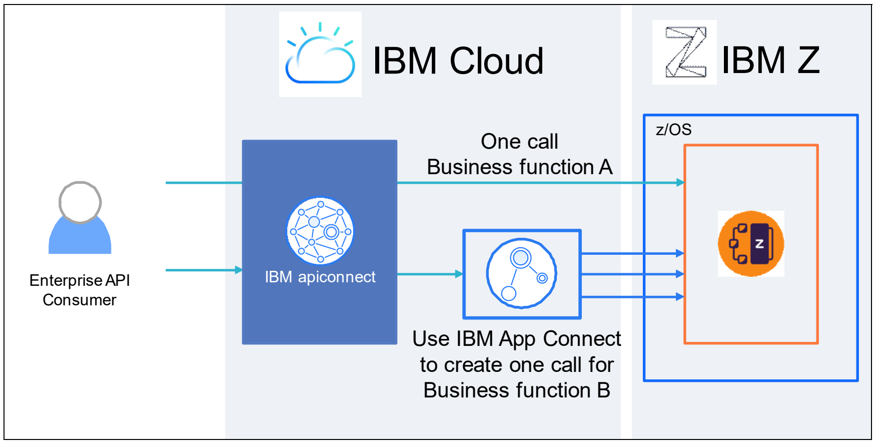


Figure 4-8 API orchestration with IBM App Connect

In Figure 4-8 business function A requires a single call to a z/OS Connect EE API, whereas business function B requires multiple calls. A message flow in App Connect uses conditional logic to perform dynamic API calls based on the responses that are returned from the previous calls.

The API for business function B created using App Connect is then brought into the API Manager component of IBM API Connect to handle the management, securing, and socializing to development teams.

App Connect deployment options

IBM App Connect (formerly known as Integration Bus) is available as a separate product and can be deployed on-premises, in a container, to IBM Cloud Private, or on a public cloud. IBM App Connect certifies container technologies such as Kubernetes and OpenShift to allow managed production-grade deployments.

IBM App Connect is also part of the IBM Cloud Pak for Integration (see 4.1.7, "IBM Cloud Pak for Integration" on page 49).

The increasingly lightweight runtime of IBM App Connect makes it a natural fit for cloud-native usage, where containers are started and stopped in seconds. Furthermore, the runtime has been radically enhanced to enable container image-based deployment, so that you can package an integration along with its configuration in a container image. This enables fine-grained deployment of individual integrations on their own runtime, with independent scaling, and availability. This is a fundamental aspect of the move to agile integration.

See 5.5, "Integrate with App Connect" on page 76 for an example scenario that features the use of App Connect.

When to use IBM App Connect

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

- 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.
- Enable event-driven integration between mainframe applications and range of cloud and on-premises apps.
- Use the App Connect built-in connectors for integrating with packaged applications, including Salesforce, JDEdwards, SAP and Siebel.

For more information about IBM App Connect Enterprise, see the following website:

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

4.1.5 IBM MQ

IBM MQ is the gold standard for enterprise messaging. It makes life easier for developers by supporting multiple operating systems, hybrid cloud environments, agile development processes, and microservices architectures with an all-in-one messaging backbone.

It enables applications and services to communicate reliably without calling each other directly, introduces process independence into the application architecture, improves fault tolerance and reliability throughout the system, and offers once and once only message delivery.

IBM has provided enterprise messaging in containers since 2015 and certifies container technologies such as Kubernetes and OpenShift to allow managed production grade deployments. Its lightweight nature makes it a natural fit for cloud native environments, allowing runtimes to start up in seconds. Figure 4-9 shows how traditional on-premises MQ environments (for example MQ for z/OS) can be expanded to a hybrid multicloud deployment.

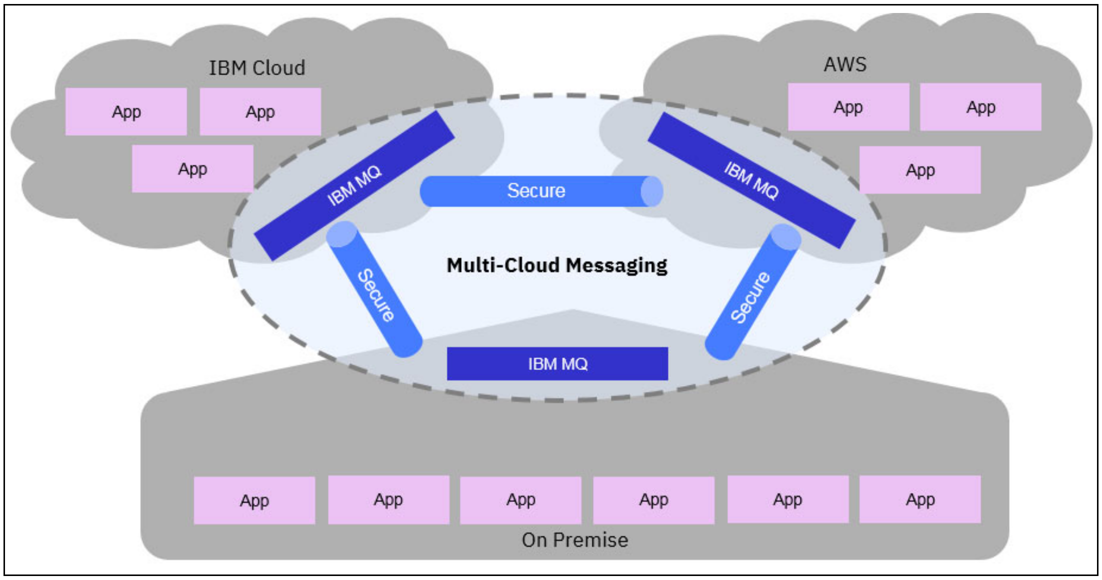


Figure 4-9 IBM MQ

Messaging is traditionally administered by dedicated teams within IT. “As-a-service” delivery lets enterprises create self-service portals that enable line of business (LoB) or individual users to request changes to the messaging infrastructure independently, such as creating or deleting queues or provisioning new resources for applications. Messaging middleware naturally works well within serverless or microservices architectures common in cloud-native development.

IBM MQ is available as a separate product and is also part of the IBM Cloud Pak for Integration (see 4.1.7, “IBM Cloud Pak for Integration” on page 49).

Note: MQ for z/OS is not part of the IBM Cloud Pak for Integration.

See 5.5, “Integrate with App Connect” on page 76 for an example scenario that features the use of IBM MQ.

When to use IBM MQ

Consider the use of IBM MQ for hybrid cloud integration with IBM Z when you want to perform the following tasks:

- Decouple the cloud-based application from the mainframe application, such that integration is possible even if one or other of these applications is temporarily unavailable.
- Enable very high messaging rates between mainframe and cloud-based applications.

- Enable the sending and receiving of batches of messages between two applications.
- Enable transactional and assured delivery of messages.

For more information about IBM MQ, see the following website:

https://www.ibm.com/cloud/mq

4.1.6 IBM Event Streams

Apache Kafka scales to handle millions of messages a second, suitable for any organization's needs. However deploying a production environment can be daunting, with the configuration of Kafka brokers, zookeepers, administration agents and so on. IBM Event Streams builds on the open source Apache Kafka event streaming technology by enhancing ease of use and enterprise deployment.

IBM Event Streams is provided on public and private cloud environments to meet the needs of clients. It can also run on-premises in a fully containerized offering, allowing the deployment to benefit from cloud native best practices such as easy installation, management and scaling of the solution. Running IBM Event Streams on Linux on IBM Z allows you to extend the event backbone closer to mainframe applications.

Many organizations use both IBM MQ and Apache Kafka for their messaging needs. Although they're generally used to solve different kinds of messaging problems, users often want to connect them together for various reasons. You can set up connections between IBM MQ and Apache Kafka or IBM Event Streams systems. For example, IBM MQ can be integrated with mainframe applications while Apache Kafka is commonly used for streaming events from web applications. The ability to connect the two systems together enables scenarios in which these two environments intersect.

IBM Event Streams is available as a separate product and is also part of the IBM Cloud Pak for Integration (see 4.1.7, "IBM Cloud Pak for Integration" on page 49).

When to use IBM Event Streams

Consider the use of IBM Event Streams for hybrid cloud integration with IBM Z when you want to perform the following tasks:

- Enable a mainframe application as a consumer of events for an event stream.
- Enable a mainframe application as a producer of events for an event stream.
- Enable very high event rates between mainframe and cloud-based applications.

For more information about IBM Event Streams, see the following website:

https://www.ibm.com/cloud/event-streams

4.1.7 IBM Cloud Pak for Integration

IBM Cloud Paks are enterprise-ready, containerized software solutions that give clients an open, faster and more secure way to move core business applications to any cloud. Each IBM Cloud Pak includes containerized IBM middleware and common software services for development and management. IBM Cloud Paks run wherever Red Hat OpenShift runs and are optimized for productivity and performance on Red Hat OpenShift on IBM Cloud.

The Cloud Pak for Integration contains the following integration capabilities:

- **API Lifecycle**: Create, secure, manage, share, and monetize APIs across clouds while you maintain continuous availability.
- **Application and Data Integration**: Integrate all of your business data and applications more quickly and easily across any cloud — from the simplest SaaS application to the most complex systems — without worrying about mismatched sources, formats, or standards.
- **Enterprise Messaging**: Simplify, accelerate, and facilitate the reliable exchange of data with a flexible and security-rich messaging solution that's trusted by many of the world's most successful enterprises. Ensure you receive the information you need, when you need it, and receive it only once.
- **Event Streaming**: Use Apache Kafka to deliver messages more easily and reliably and to react to events in real time. Provide more-personalized customer experiences by responding to events before the moment passes.
- **High-Speed Data Transfer**: Send, share, stream and sync large files and data sets virtually anywhere, reliably and at maximum speed. Accelerate collaboration and meet the demands of complex global teams, without compromising performance or security.
- **Secure Gateway**: Create persistent, security-rich connections between your on-premises and cloud environments. Quickly set up and manage gateways, control access on a per resource basis, configure TLS encryption and mutual authentication, and monitor all of your traffic.

Whereas these same integration capabilities are available by installing separate integration products, the Cloud Pak for Integration provides additional benefits, including:

- A platform navigator enables a single entry-point for users. It allows users fast access to all of the integration capabilities. Through this unified experience, it is simple to navigate across any of the product capabilities.
- An asset repository allows users to share integration assets across various integration capabilities in the platform. For example, an OpenAPI specification stored in the repository can be directly imported within the API management user interface.
- The underlying container platform takes care of collecting the logs from all components in use. The benefit is that all logs are in one place and the monitoring and dashboards are built upon this information. The Cloud Pak for Integration also includes visibility to how messages and data are being processed through embedded end-to-end tracing capabilities.

For more information about IBM Cloud Paks, see the following website:

https://www.ibm.com/cloud/paks

> **Note:** IBM intends to deliver the Red Hat OpenShift Container Platform and IBM Cloud Paks on Linux on IBM Z and IBM LinuxONE platforms. See https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=AN&subtype=CA&htmlfid=872/ENUSAP19-0517&appname=STG_XS_IDEN_ANNO

4.2 z/OS subsystem considerations

In this section we look at hybrid cloud integration considerations for the most common z/OS subsystems:

- 4.2.1, “CICS Transaction Server for z/OS” on page 51
- 4.2.2, “IMS” on page 53
- 4.2.3, “Db2 for z/OS” on page 54
- 4.2.4, “MQ for z/OS” on page 57

We focus on REST API enablement of these z/OS subsystems.

4.2.1 CICS Transaction Server for z/OS

CICS is used extensively for high-volume transaction processing. Consider the following integration solutions that can be used for accessing CICS applications as REST APIs:

- z/OS Connect EE
- Web applications in CICS

z/OS Connect EE with CICS

z/OS Connect EE enables the creation and deployment of APIs that reuse CICS applications, and a CICS program can also call any RESTful endpoint (see 4.1.1, “IBM z/OS Connect Enterprise Edition” on page 36).

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. An overview of the use of the CICS service provider is shown in Figure 4-10.

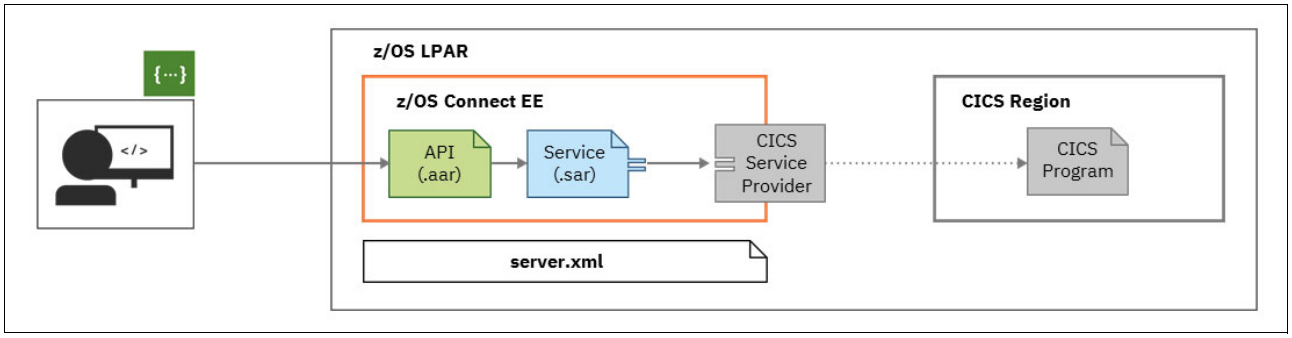


Figure 4-10 z/OS Connect CICS service provider

As shown in Figure 4-10, 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).

See 5.1, “Implement Open Banking APIs with z/OS Connect EE” on page 64 for an example scenario that features the use of z/OS Connect EE with CICS.

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, “When to use z/OS Connect EE” on page 40.

Web applications in CICS

To provide a REST interface to CICS applications, you can develop a Web presentation layer that runs in CICS and connects to CICS business logic programs. The most common way of doing this is to write Java programs that use CICS Liberty support but you can also develop Node.js applications using JavaScript.

You can use the IBM CICS SDK for Java in CICS Explorer® or the CICS-provided artifacts on Maven Central to create, package, and build Java applications for CICS. To develop a RESTful service, your program can use the Java API for RESTful Web Services (JAX-RS). 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 4-11.

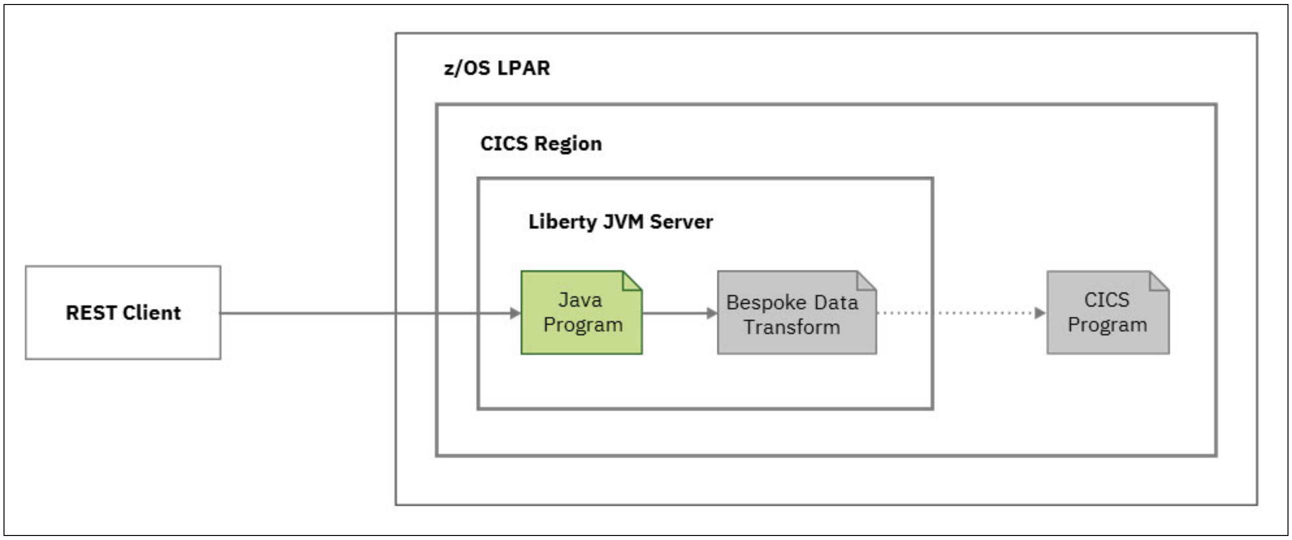


Figure 4-11 Web application in CICS Liberty

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 runtimes, such as IBM MQ. It can use a wide range of Liberty features and CICS features to integrate with CICS qualities of service.

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.

See 5.4, “Develop Java-based REST APIs” on page 73 for an example scenario that features the use of Java-based REST APIs in CICS.

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.
- 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/

4.2.2 IMS

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

z/OS Connect EE enables the creation and deployment of APIs that reuse IMS applications, and an IMS program can also call any RESTful endpoint (see 4.1.1, “IBM z/OS Connect Enterprise Edition” on page 36).

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 4-12.

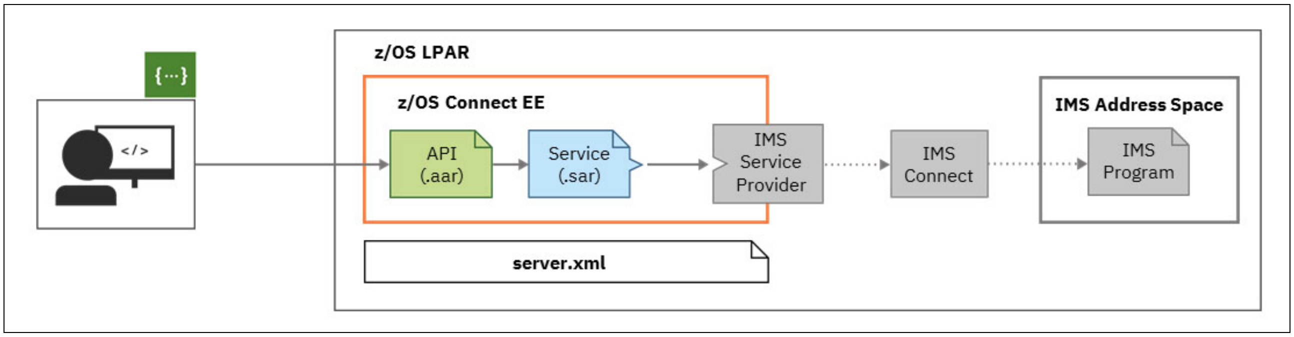


Figure 4-12 z/OS Connect EE IMS service provider

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 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.

See 5.1, “Implement Open Banking APIs with z/OS Connect EE” on page 64 for an example scenario that features the use of z/OS Connect EE with IMS.

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 , “When to use z/OS Connect EE” on page 40.

For more information about the use of the IMS service provider, see IBM Developer:

https://ibm.biz/zosconnectdc

4.2.3 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 integration solutions that can be used for accessing Db2 data as REST APIs:

- Db2 REST services
- z/OS Connect EE with Db2 REST services

Db2 REST services

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.

You can define 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 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 4-13.

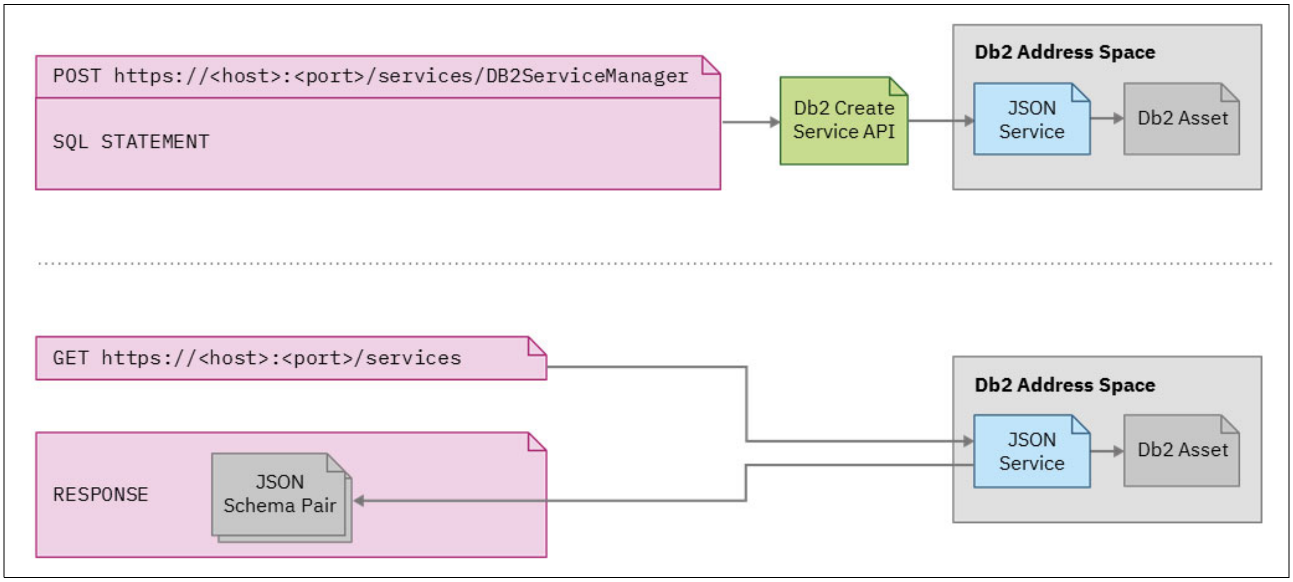


Figure 4-13 Db2 REST services

> **Note:** A Db2 stored procedure call can also be specified as input to the API used for creating a Db2 REST service.

An authorized user can discover and invoke the service through a REST HTTP client. Db2 accepts the HTTP POST request, processes the JSON request body, runs the bound SQL statement, and returns any output in JSON.

Db2 REST services do not support different HTTP verbs, API mapping, or discovery using a Swagger document. However, these capabilities can be achieved by creating an API layer in front of the Db2 REST service using z/OS Connect EE. For more information, see , “z/OS Connect EE with Db2 REST services” on page 56.

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.
- As a step towards creating a REST API with z/OS Connect EE.

z/OS Connect EE with Db2 REST services

z/OS Connect EE enables the creation and deployment of APIs that reuse Db2 REST services. z/OS Connect EE can be configured to connect to Db2 using the REST service provider.

An overview of using z/OS Connect EE with Db2 REST services is shown in Figure 4-14.

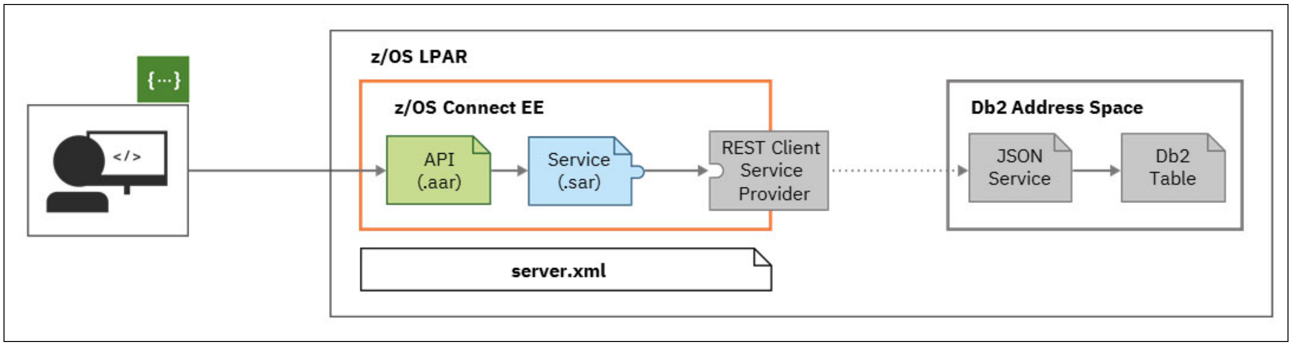


Figure 4-14 z/OS Connect EE with Db2 REST services

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

The service definition provides information about the location of the Db2 REST service. The JSON request message is forwarded to Db2 using the REST Client service provider.

The z/OS Connect EE API Toolkit is used to create a Db2 service project that is based on the Db2 REST service. The resulting service archive is then used to create an API.

Note: The support of Db2 REST services with z/OS Connect EE supersedes the Db2 Adapter for z/OS Connect.

See 5.1, “Implement Open Banking APIs with z/OS Connect EE” on page 64 for an example scenario that features the use of z/OS Connect EE with Db2.

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:

- Support the discovery of defined APIs by using the OpenAPI standard to share API definitions as Swagger documents.
- Provide intuitive, workstation-based tooling that enables a developer to create REST APIs from Db2 data.
- 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.

4.2.4 MQ for z/OS

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.

MQ for z/OS often plays a key part within the enterprise messaging backbone (see 4.1.5, “IBM MQ” on page 47). It is used widely for asynchronous connectivity with mainframe systems.

This section describes the following integration solutions that can be used for accessing MQ-based z/OS applications as REST APIs:

- 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.

MQ messaging REST API

The messaging REST API comes as standard with IBM MQ 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, an HTTP GET to browse messages, or an HTTP DELETE to destructively get a message.
- The messaging REST API is enabled using the `mqweb` server, which is a Liberty server integrated with IBM MQ (Figure 4-15).

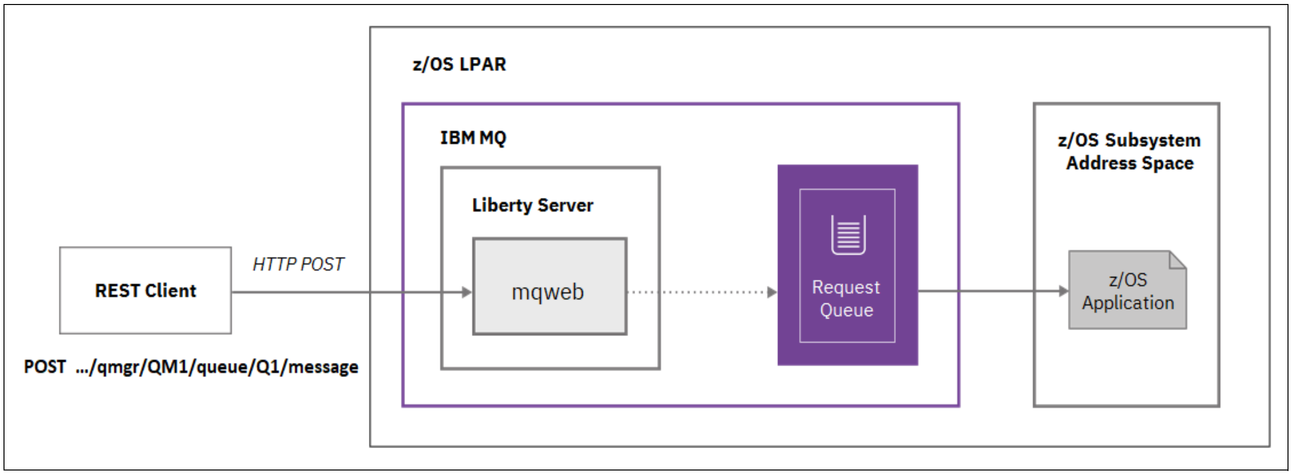


Figure 4-15 MQ messaging REST API

When a client application uses the REST API to perform a messaging action on an IBM MQ queue or topic 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 or topic.

Note: The MQ messaging REST API supports only UTF-8 text based payloads. While MQ for z/OS will automatically convert between different code pages, it is the application's responsibility to perform any other necessary data conversion.

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 messages to queues or topics, and receive messages from queues.
- Allow users to put and get messages to any queue or topic that they are authorized to access without needing any special configuration to expose those queues or topics as REST APIs.

z/OS Connect EE with IBM MQ

The IBM MQ service provider is supplied with z/OS Connect. The provider 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 following types of service are supported:

- A two-way service allows a RESTful client to perform request-reply messaging on a pair of queues.
- A one-way service can be used to provide a RESTful API on top of a single IBM MQ queue or topic.

The z/OS Connect EE API Toolkit is used to create an MQ service project based on the request and response copybooks of the z/OS application[1]. The resulting service archive is then used to create an API. 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.

The MQ servicer provider uses the MQ classes for JMS (Java Message Service) to connect to a queue manager. These classes provide two different types of connection:

- In bindings mode, connections are made directly to the queue manager by using the Java Native Interface (JNI). Bindings mode connections tend to provide better performance than client mode but require the z/OS Connect EE server and IBM MQ queue manager to be running in the same LPAR.
- In client mode, connections to the queue manager are made over TCP/IP. Client mode connections allow greater flexibility than bindings mode as the IBM MQ queue manager can be running on a different z/OS LPAR to the z/OS Connect EE server. However they require more configuration, which might include setting up TLS, and typically do not perform as well as bindings mode connections.

[1] The API toolkit has been enhanced to support the creation of IBM MQ services in the z/OS Connect EE open beta.

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 4-16.

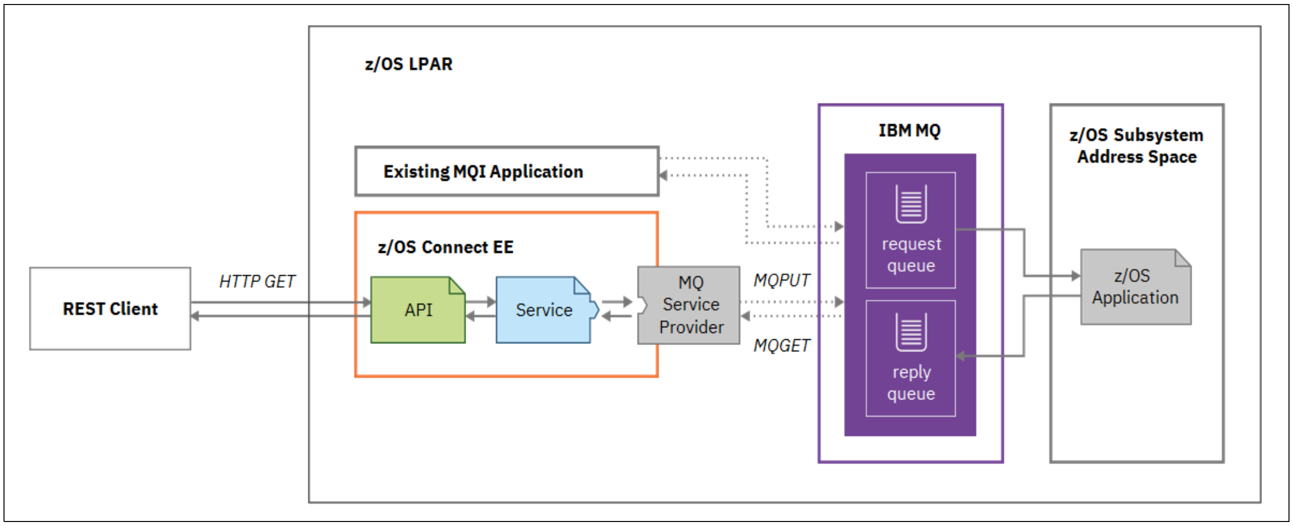


Figure 4-16   z/OS Connect EE with MQ service provider and two-way service

The client issues an HTTP GET request that specifies a JSON payload, as shown in Figure 4-16. 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.

Note: Any HTTP verb can be used to invoke a two-way service.

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 identifies this message by using a correlation identifier, takes its payload, converts it to JSON, and returns it as the response body of the HTTP GET 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 and process it as they do with any other message. They are not aware of the fact that a REST client sent it.

See 5.1, “Implement Open Banking APIs with z/OS Connect EE” on page 64 for an example scenario that features the use of z/OS Connect EE with IBM MQ.

One-way requests

In a one-way request, REST clients can send a message to a queue or topic, or receive a message from a queue.

The API defines the REST interface that you want to enable for the MQ queue or topic, 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 could map the request to a one-way service configured for sending messages. The MQ service provider will then put the message on the target queue (see Figure 4-17). When the REST client receives an HTTP 200 response, this response is confirmation that the message was successfully placed on the queue.

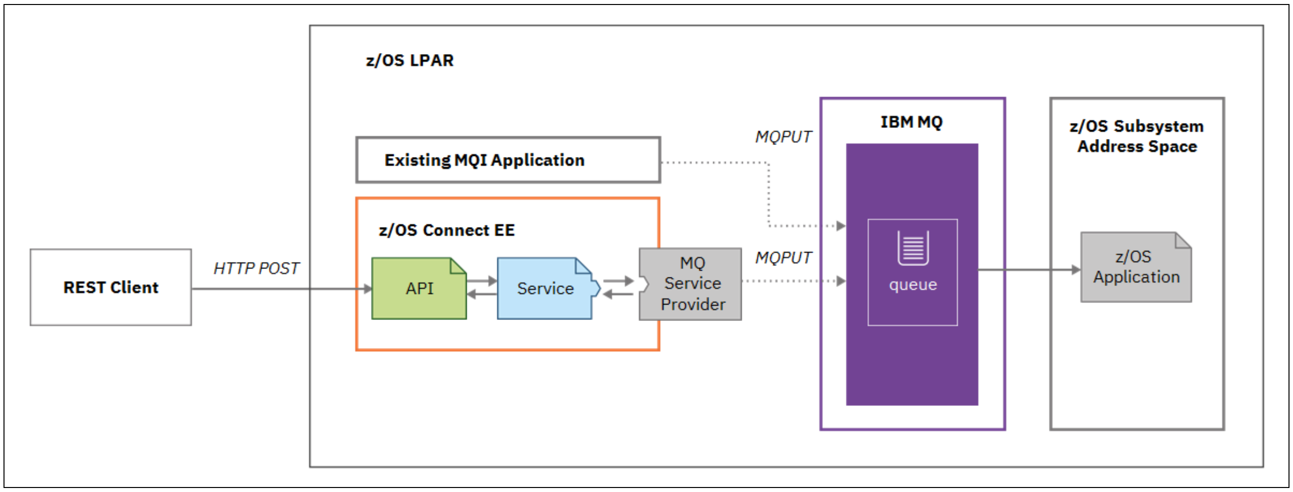


Figure 4-17 z/OS Connect EE with MQ service provider and one-way service

Any HTTP verb can be used to invoke a one-way service. The actual operation that is performed on the underlying IBM MQ queue or topic is defined by using the **`messagingAction`** property.

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:

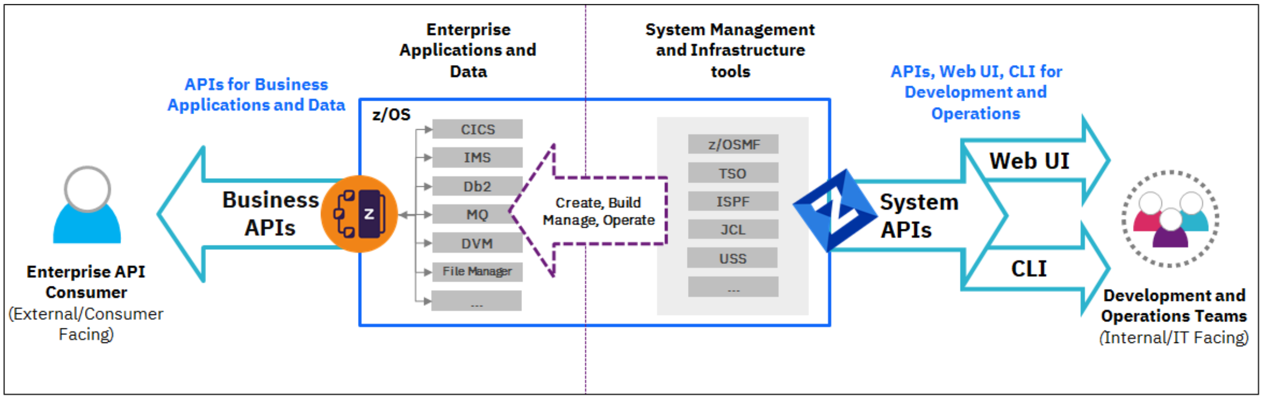
- Create a REST interface to an MQ-based mainframe application that can be used by client application programmers without knowledge of MQ.
- Use the interceptor framework of z/OS Connect EE, for authorizing, auditing, and monitoring REST API requests to MQ-based applications.

4.3 Zowe

Zowe is an open source project under the Linux Foundation created to provide technologies for the benefit the IBM z/OS platform and is available for anyone, including Integrated Software Vendors (ISVs), System Integrators, and z/OS customers.

Zowe offers modern interfaces to interact with z/OS and allows you to work with z/OS in a way that is similar to what you experience on cloud platforms today. You can use these interfaces as delivered or through plug-ins and extensions that are created by clients or third-party vendors. Zowe is especially appealing to the next generation of IT staff using z/OS.

Figure 4-18 shows the system APIs provided by Zowe that are used for development, system programing and operations. The figure also shows business APIs that are deployed to z/OS Connect EE and used for accessing mainframe applications and data.



Enterprise Applications and Data
System Management and Infrastructure tools
APIs for Business Applications and Data
APIs, Web UI, CLI for Development and Operations
z/OS
CICS
IMS
Db2
MQ
DVM
File Manager
...
Create, Build Manage, Operate
z/OSMF
TSO
ISPF
JCL
USS
...
Business APIs
System APIs
Web UI
CLI
Enterprise API Consumer
(External/Consumer Facing)
Development and Operations Teams
(Internal/IT Facing)

Figure 4-18 Zowe System APIs

Zowe consists of the following components:

- **Zowe Application Framework**: A web user interface (UI) that provides a virtual desktop that contains apps that allow access to z/OS function. Base Zowe includes apps for traditional access such as a 3270 terminal and a VT Terminal, as well as an editor and explorers for working with JES, IBM MVS™ Data Sets, and UNIX System Services.

  The Zowe Application Framework is intended to break down product silos that have grown up over the years by providing a task-oriented view of z/OS services.

- **z/OS Services**: Provides a range of APIs for the management of z/OS JES jobs and MVS data set services.

- **Zowe CLI**: A command-line interface that lets application developers interact with the mainframe in cloud-native way. The Zowe CLI lets application developers use common tools such as integrated development environments (IDEs), shell commands, bash scripts, and build tools for mainframe development.

  It provides a set of utilities and services for application developers that want to become efficient in supporting, building, and deploying z/OS applications quickly. The Zowe CLI can be used, for example, to deploy z/OS Connect EE APIs and to configure the connected z/OS subsystems.

- **API Mediation Layer**: Provides a gateway that acts as a reverse proxy for z/OS services for working with MVS Data Sets, JES, as well as working with z/OSMF REST APIs.

  The API Mediation Layer provides security, load balancing and high availability options for accessing mainframe system APIs.

Note: IBM provides a distribution of Zowe for use with IBM products. For additional information see:

https://developer.ibm.com/mainframe/products/ibm-z-distribution-for-zowe/

For more information about Zowe, see the following website:

`https://docs.zowe.org/`

5

Real-world scenarios

In this chapter, we summarize several integration scenarios that integrate cloud-based, 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:

- An implementation of Open Banking APIs using z/OS Connect EE
  (See section 5.1, "Implement Open Banking APIs with z/OS Connect EE" on page 64.)
- Calling out to external services from a CICS application using z/OS Connect EE
  (See section 5.2, "Call out to external services using z/OS Connect EE" on page 67.)
- Creation of a managed API framework using API Connect
  (See section 5.3, "Build a managed API framework using API Connect" on page 69.)
- An implementation of Java-based REST APIs using CICS Liberty
  (See section 5.4, "Develop Java-based REST APIs" on page 73.)
- An integration between CICS and Salesforce using IBM App Connect
  (See section 5.5, "Integrate with App Connect" on page 76.)

5.1 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.

5.1.1 Introduction

One industry that has quickly embraced the API economy is the financial services sector. Under the umbrella term of Open Banking, financial institutions are providing consumers choice over how 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 A 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 A is to streamline and accelerate delivery of Open Banking services through their API gateways, from their IMS and CICS core banking systems. The following key questions are to be addressed in this project:

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

5.1.2 Key decision factors

Bank A considered multiple factors when it decided 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 aim is to hide the underlying mainframe implementation so that API developers can access the core banking applications as REST APIs in the same way that they access other core systems. The key decision factors are reviewed next.

Mainframe application interface

Today, the core banking functions are accessed in the following ways:

- An in-house developed C application — hosted in a stand-alone address space running on z/OS — provides synchronous connectivity services to IMS.
- An MQ-based application is used for pseudo-synchronous connectivity to CICS

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

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

Application integration infrastructure

The core banking applications that run in IMS and CICS are to be accessed from the internal API gateway. The target solution must be compatible with the existing application integration interfaces and not require changes to the core banking applications.

Hybrid integration requirements

The advent of Open Banking regulations mandate that Bank A must offer API services. The Bank has chosen a private cloud deployment for the internal API gateway to enable the greatest possible portability. This takes advantage of a cloud-native infrastructure, using the cluster management, scaling, and availability capabilities of the cloud platform in which the gateway runs.

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.
  - The solution must support the large number of test teams and environments.
- Optimize performance and scalability

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

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

5.1.3 Solution architecture

Bank A chose to implement a solution based on z/OS Connect EE (see Figure 5-1 on page 66) 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. z/OS Connect EE provides a common tool set and runtime for creating and deploying APIs based on different mainframe applications and data.

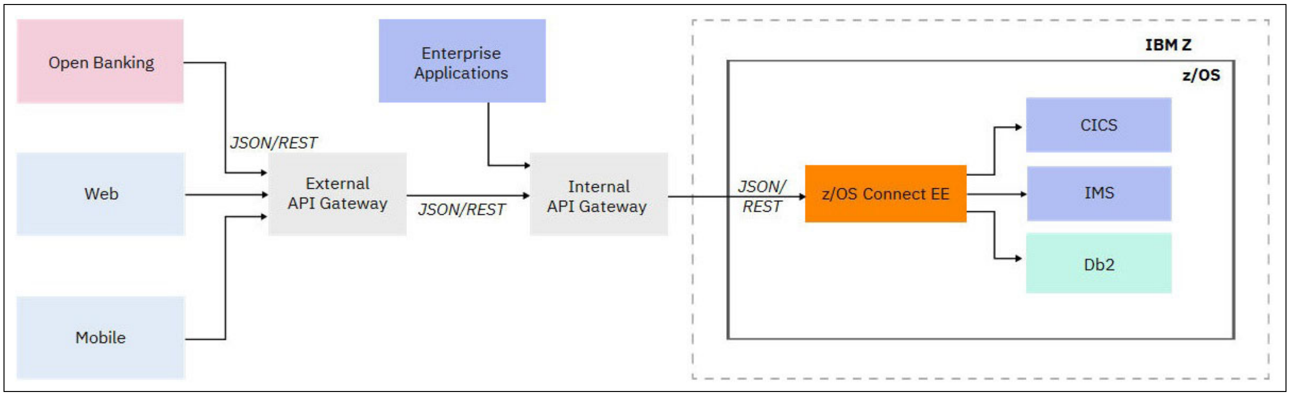


Figure 5-1 REST API enablement for Open Banking

The solution features the following main components:

- z/OS Connect EE

  z/OS Connect EE provides a REST API interface for the IMS and CICS core banking applications.

  The IMS service provider is used for connectivity to the IMS core banking applications. API request messages contain only the essential pieces of information needed to run the request. The API Toolkit is used to create services that augment the request messages by assigning values to the fields that are needed to run the IMS programs.

  The MQ service provider is used for connectivity with the MQ-based CICS core banking application. The API mappings allow inline manipulation of requests such as mapping HTTP headers and defaulting of JSON fields. The services mappings convert JSON data to MQ messages so that the CICS application continues to work in the same way as before.

  The REST Client service provider is used to send REST requests directly to Db2, for example, for read-only operations that use pre-defined SQL queries. The API mapping model of z/OS Connect EE interprets the request by inspecting the URI, HTTP headers, and JSON body, and then sends the request to a Db2 REST service.

  z/OS Connect EE policy support is used to route requests from z/OS Connect EE to different IMS test environments based on the value of a custom HTTP header. This reduces the number of z/OS Connect EE test instances that need to be configured.

  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 hosted on the Bank's Private Cloud Platform, Red Hat OpenShift, and is used to surface APIs accessible to all applications and services within the boundary of Bank A. This includes other enterprise applications, as well as requests from Third Party Providers (TPPs) that are processed first by the external API gateway.

- External API gateway

  The external API gateway is used to surface APIs to consumers outside the boundary of Bank A. For example, APIs that 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 A and is open to TPPs who have permission to access account and payment information on behalf of the bank’s customers.

5.1.4 Next steps

Bank A successfully completed a proof of concept with z/OS Connect EE, concluding that it gives them significant agility benefits, reducing service creation time from months to days. This is partly due to the efficiencies gained by using the z/OS Connect EE API Toolkit, and partly due to the process simplification in creating APIs deployed in the API gateway.

At present, Bank A is preparing z/OS Connect EE for production deployment and exploring the next set of z/OS applications to API-enable. Bank A is also reviewing what changes are required to the MQ-based CICS core banking application in order to use the CICS service provider rather than the MQ service provider.

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

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

5.2.1 Introduction

In the same way that engagement 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-premises, 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 website. 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.

5.2.2 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 that is 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 x 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.

5.2.3 Solution architecture

Insurance company A chose to implement a solution based on the API requester support in z/OS Connect EE (see Figure 5-2).

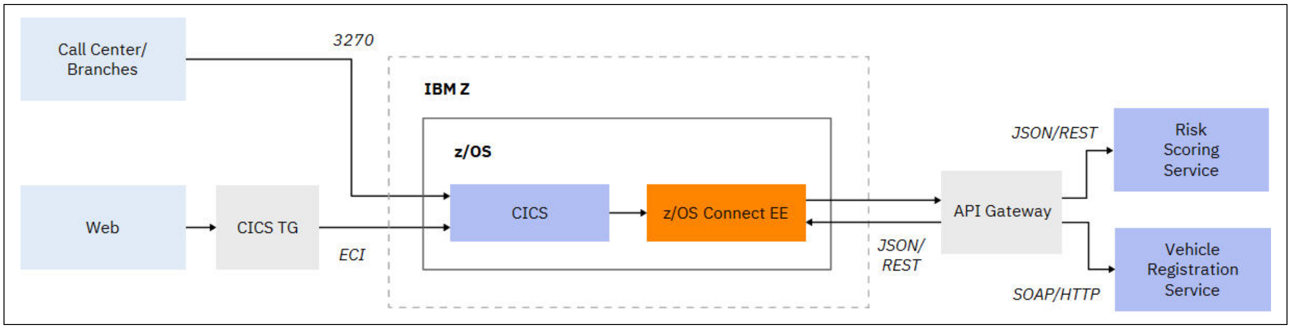


Figure 5-2 Call out to external services using z/OS Connect EE

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.

    The Swagger document of the risk scoring REST API can be used directly with the z/OS Connect EE Build Toolkit. In the case of the vehicle registration service, the API Gateway exposes a REST API and then converts REST/JSON messages to the SOAP/XML format expected by the vehicle registration service.

  – 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.

5.2.4 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.

5.3 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.

5.3.1 Introduction

Bank B 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 B places customer service as its top priority to encourage growth and has invested in new mobile 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.

Based on the needs to deliver new mobile services quickly and to comply with Open Banking regulations, Bank B 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:

- Automatic build, deployment, and testing of APIs
- Self-discovery of APIs
- Robust security and traffic shaping
- Reusing internal services
- Visibility and monitoring of APIs

5.3.2 Key decision factors

Bank B considered several factors when it decided 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 B 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 operations, 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 and associated APIs are 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 x 7.
- API lifecycle

  API creation, deployment, and testing must be automated, and support self-discovery and versioning.
- 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.

5.3.3 Solution architecture

Bank B chose to adopt a solution that is based on IBM API Connect and z/OS Connect EE. This solution supports a wide set of REST APIs that are used by the bank's internal and external clients. Figure 5-3 shows the mainframe systems and API operations used by the bank advisor mobile app.

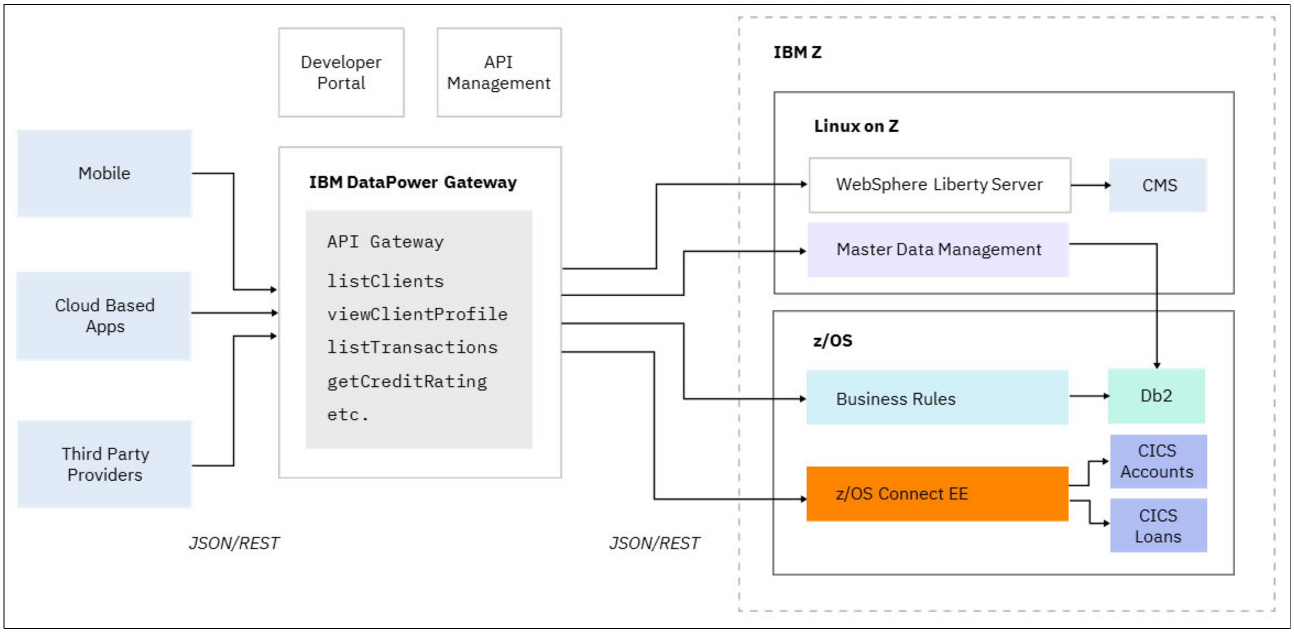


Figure 5-3 Build a managed API framework using API Connect

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 operations).

  Developers use the API Toolkit to develop and test APIs in provisioned development z/OS Connect EE instances.

  A Jenkins based CI/CD (continuous integration and continuous delivery) pipeline is then used to automatically build, deploy, and test APIs to the pre-production and production environments. The CI/CD pipeline uses the following components:

  - A Git repository for storing z/OS Connect EE projects
  - z/OS Connect EE Build Toolkit for building services and APIs
  - JFrog Artifactory for storing z/OS Connect EE service archive (.SAR) and API archive (.AAR) files
  - Zowe and the z/OS Connect EE RESTful administration interface for deploying services and APIs
  - Newman for automated testing

- Business Rules

  IBM ODM provides a SOAP interface that is used by the getCreditRating operation 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 operations.

- WebSphere Liberty

  A WebSphere Liberty server provides a REST interface to a Cards Management System (CMS) that is used by the viewClientProfile operation.

- 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 new API versions and to revert to previous versions when required. A developer portal is used to communicate information about the APIs, 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 IBM DataPower is used as an API Gateway.

5.3.4 Next steps

Bank B has established an API framework that is used extensively by mobile and cloud-based apps, including the new bank advisor app.

The CI/CD pipeline that is used for deploying APIs to z/OS Connect EE is being extended to automatically build and deploy new APIs when CICS application changes are installed.

5.4 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.

5.4.1 Introduction

Bank C 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.

5.4.2 Key decision factors

Bank C considered several factors when it decided 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 C provides financial services to over one million clients that use modern web and mobile applications to access their accounts. These engagement 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 engagement application 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.

5.4.3 Solution architecture

Bank C 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 5-4 on page 75.

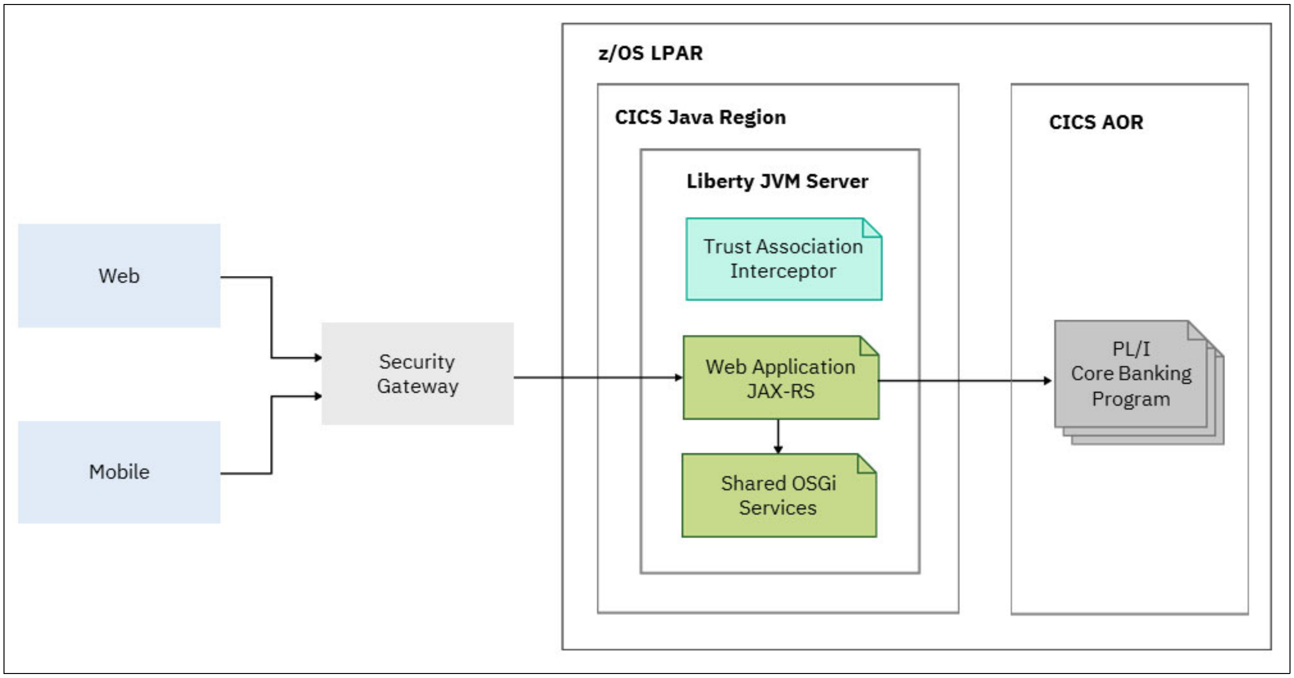


Figure 5-4 Develop Java-based REST APIs

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 message 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

5.4.4 Next steps

Bank C 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 C is also considering if z/OS Connect EE can be used for REST API enablement of certain applications.

5.5 Integrate with App Connect

This scenario focuses on the integration of a CICS application with Salesforce using IBM App Connect.

5.5.1 Introduction

As applications and data are increasingly hosted on cloud infrastructure, there is an increasing requirement to maintain consistency between cloud and on-premises data stores.

Bank D has recently implemented a Salesforce customer-relationship management (CRM) solution to collect, store, manage, and interpret data from many customer-related activities. The bank needs to maintain consistency between the Salesforce CRM system and the financial applications that run in CICS, for example, when new accounts are created, customer addresses are updated, and so on.

The initial requirement is to replicate data updates made to CICS VSAM data to Salesforce. A follow-on requirement is to replicate data updates made to the set of Salesforce customer objects to CICS.

The following key questions are to be addressed in this project:

- How to enable data replication between CICS and Salesforce?
- How to transform the data emitted by CICS to the format required by Salesforce?

5.5.2 Key decision factors

Bank D considered several factors when it decided how to enable the integration between CICS and Salesforce. The primary driver is to enable the integration quickly and to build a solution that can be reused for bidirectional data replication. The key decision factors are reviewed next.

Mainframe application interface

Today, the CICS packaged financial applications are accessed synchronously using the CICS web support, and asynchronously using messaging.

The new solution should not require any changes to the CICS applications. Data records from CICS formatted in byte array structure must be transformed to the JSON or CSV formats supported by the Salesforce customer objects. The data transformation solution must also take care of code page conversion from EBCDIC encoded data to ASCII encoding.

Application integration infrastructure

Bank D is an extensive user of IBM MQ and is already using IBM App Connect for a variety of integration requirements.

Hybrid integration requirements

To integrate between the on-premises CICS financial applications and a SaaS application like Salesforce running in a public cloud requires that the bank's security and privacy policies are met.

Non-functional requirements

The following main non-functional requirements have been identified:

- Scalability

  The solution needs to handle thousands of CICS data updates per second.

- Batch processing

  Updates to account and customer data in CICS must generate events in real time, however, the solution must also support batch processing.

- Security

  All network communications must be encrypted and the connection to Salesforce must be secured.

5.5.3 Solution architecture

Bank D chose to implement a solution based on IBM App Connect and CICS event processing. You can use App Connect with Salesforce by configuration and data mapping without a need for coding, and CICS events can be defined and controlled without the need to modify the CICS financial applications. The solution is shown in Figure 5-5.

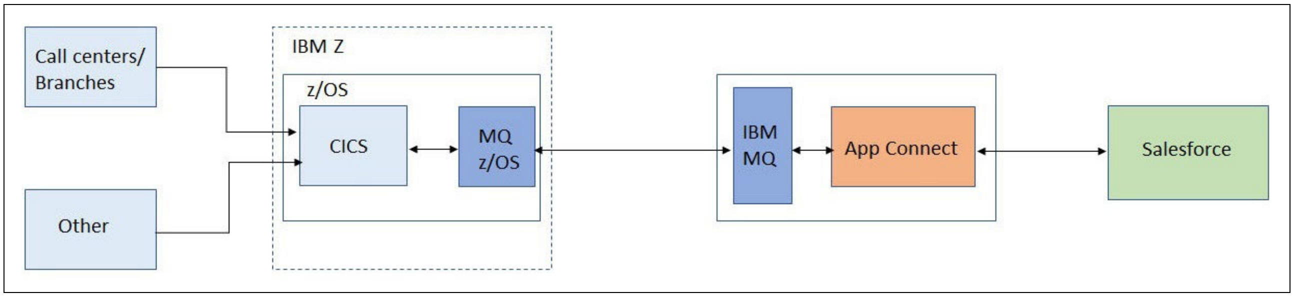


Figure 5-5 Integrate with IBM App Connect

The solution features the following components:

- App Connect

  App Connect provides connectivity options across cloud service applications, cloud platforms, and existing on-premises applications. It includes connectors for integrating with Salesforce and other non-IBM solutions.

App Connect enables Bank D to automatically update Salesforce in real time, adding the account and customer data that the CICS financial applications generate.

The App Connect message flow:

  - Reads the data from an MQ queue or a file (for batch processing).
  - Formats the source data (byte array) to the target format required by the Salesforce customer objects (JSON or CSV).
  - Encodes the data to ASCII from an EBCDIC encoding.
  - Uses a SalesforceRequest node to create or update records in the Salesforce.com system via a REST API. The connection to Salesforce is encrypted using HTTPS and secured using the OAuth 2.0 support that is provided with the Salesforce connector.

App Connect is designed to enable scaling to many thousands of messages per second and provides a way of updating Salesforce objects in a performant, secure, and reliable architecture.

- CICS event processing

  CICS event processing enhances business flexibility by providing a non-invasive way of enhancing applications to emit events to a variety of different destinations. You can specify, capture, and emit business events without changing the application. These events can be consumed by another CICS application or placed on an IBM MQ queue for consumption in a variety of ways, including by App Connect.

  The Event binding editor of CICS Explorer is used to define business events. You can specify that an event is emitted when the application issues any of the event enabled EXEC CICS API commands or when an application program is initiated. In the case of the CICS financial applications used by Bank D, events are captured when the accounts and customer VSAM data sets are updated.

  CICS event processing provides a high-performance, manageable and scalable environment, which has minimal impact on the performance of the CICS financial applications.

- IBM MQ

  IBM MQ is used to transfer messages between CICS and App Connect. The messages that are put by the CICS event processing adapter are sent from MQ z/OS across an encrypted MQ channel to the MQ queue manager used by App Connect.

5.5.4 Next steps

The use of App Connect for integrating the CICS financial applications with Salesforce achieves a return on investment in days rather than months. A similar solution can also be used for replicating data updates from Salesforce to CICS.

6

Summary

The need to deliver new functions at speed and the huge growth in cloud services means that the requirement to interoperate with the mainframe within a hybrid cloud environment is high on the priority list of many of the world’s largest companies.

In this paper, we reviewed the key considerations for planning hybrid cloud integration with the mainframe. We also 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 6-1 Common integration architectures

Integration architecture	Description	Recommendation
APIs and API management	Architecture for creating, assembling, managing, securing, and socializing web application programming interfaces (APIs).	Use when: ▸ 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, HTTP verbs and JSON. De-facto standard for engagement applications.	Use when: ▸ JSON is the primary data payload ▸ Intuitive and simple interface for developers is required
Messaging	Asynchronous transport mechanism.	Use when: ▸ Assured delivery is required ▸ Enabling publish/subscribe applications ▸ Reusing a messaging infrastructure
Event streams	Implementation of publish/subscribe pattern at large scale.	Use when: ▸ Need to support very large number of events ▸ Need to support many producers and consumers of events

6.2 Integration solutions

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

Table 6-2 IBM integration solutions

Integration solution	Recommendation	Description
z/OS Connect Enterprise Edition	Provides a common toolset and runtime for REST HTTP calls to applications and data assets that reside on z/OS. 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 tool-based approach for creating RESTful APIs that are based on z/OS assets. ▸ Enabling discovery of defined APIs based on OpenAPI (Swagger 2.0) standard. ▸ Simplify the REST API development process by making the mainframe application owner responsible for creating APIs from z/OS assets. ▸ Requiring z/OS applications (CICS, IMS, and batch) to call external REST APIs.
IBM API Connect	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 are required
IBM DataPower Gateway	SOA and API security gateway.	Use when: ▸ Securing hybrid integration access to the mainframe ▸ Deploying APIs to a secure and efficient API Gateway

Integration solution	Recommendation	Description
IBM App Connect	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
IBM Event Streams	IBM implementation of Apache Kafka.	Use when: ► Enabling mainframe application as producer or consumer of events for an event stream. ► For very high messaging rates.
IBM Cloud Pak for Integration	Containerized IBM integration middleware that runs wherever Red Hat OpenShift runs.	Use when: ► Deploying integration components using container-based, decentralized and microservice-aligned approach. ► Enabling unified development experience (platform navigator and asset repository) across IBM integration capabilities.

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.

- *Accelerating Modernization with Agile Integration,* SG24-8452
- *An Architectural and Practical Guide to IBM Hybrid Integration Platform*, SG24-8351
- *CICS and SOA: Architecture and Integration Choices*, SG24-5466
- *IMS Integration and Connectivity Across the Enterprise*, SG24-8174

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 API Connect:

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

  https://www.ibm.com/products/datapower-gateway
- IBM App Connect:

  https://www.ibm.com/cloud/app-connect
- IBM MQ:

  https://www.ibm.com/cloud/mq

- IBM Event Streams:

  https://www.ibm.com/cloud/event-streams

- IBM Cloud Paks:

  https://www.ibm.com/cloud/paks

- Zowe:

  https://docs.zowe.org/

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5319-03

ISBN 0738458627

Printed in U.S.A.