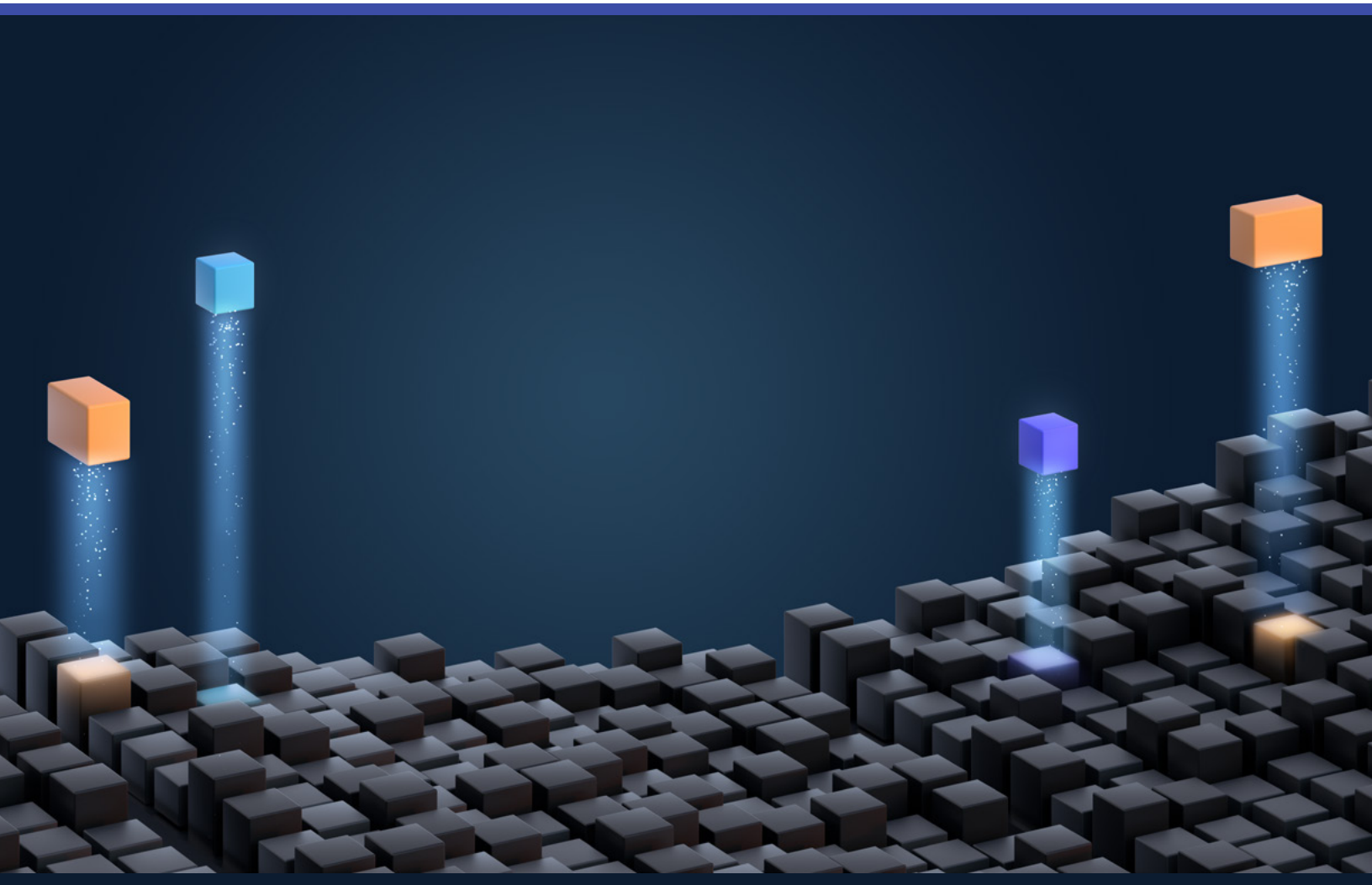
OpenLegacy
Digital to the Core

White Paper

How to evaluate current approaches to legacy integration

Integration and innovation at the core of modern digital platforms

Zeev Avidan
Chief Product Officer, OpenLegacy



www.openlegacy.com

Introduction

Integration is the glue that makes computer applications work together to provide meaningful outcomes. While many of the concepts and best-practices of integration have been around for decades, new ways of creating applications require re-evaluating these integration patterns. For example, integrating several monolithic applications is a challenge, but the rise of microservices and other distributed deployment patterns makes this challenge exponentially greater.

01

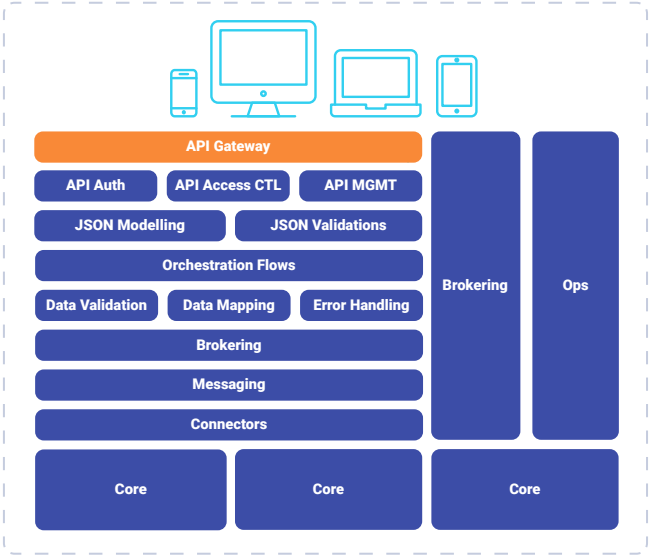
Technological landscape

When dealing with traditional 'pure' integration, specifically concerning integrating monolithic legacy applications, two approaches immediately surface as most common: the real-time connector and the asynchronous message-queue.

While both of these two approaches are still viable concepts in modern architectures, they do require careful considerations when implementing them in the real world. These considerations include topics such as:

- Resource and skill training
- Infrastructure complexity
- Time to market
- Development timelines
- Overall license and run-time costs

While many of the concepts and best-practices of integration have been around for decades, new ways of creating applications dictate a new look at these integration patterns. Integration is at the core of modern digital platforms and should keep in lockstep with the innovation they bring.



Homegrown or Middleware Solution

A recent Gartner prediction estimates that "Through 2020 integration will consume 60% of the time and cost of building a digital platform." This reflects the fact that integration is at the core of modern digital platforms and should keep in lockstep with the innovation they bring.

Source: Gartner, Inc.

02

Real-Time connectors

These types of connectors have been around since the early days of enterprise integration. They began as point-to-point connection protocols but later evolved into the hub-and-spoke model of EAI and the enterprise bus concept of ESBs. In these cases they

provide the last mile of connectivity to an application and are specifically designed for a certain protocol or language.

Due to their specificity, it became common for vendors such as iWay Software to provide suites of connectors which connect siloed applications into the middleware. Each connector would require its own setup and skills. This approach is still very common today with products such as IBM's z/OS connect which exposes a mainframe COBOL-based application in a JSON-REST protocol, to be consumed by API middleware.

1. Complex set-up and maintenance

These connectors usually rely on the legacy system to run and they require multiple other products and current OS versions to be installed.

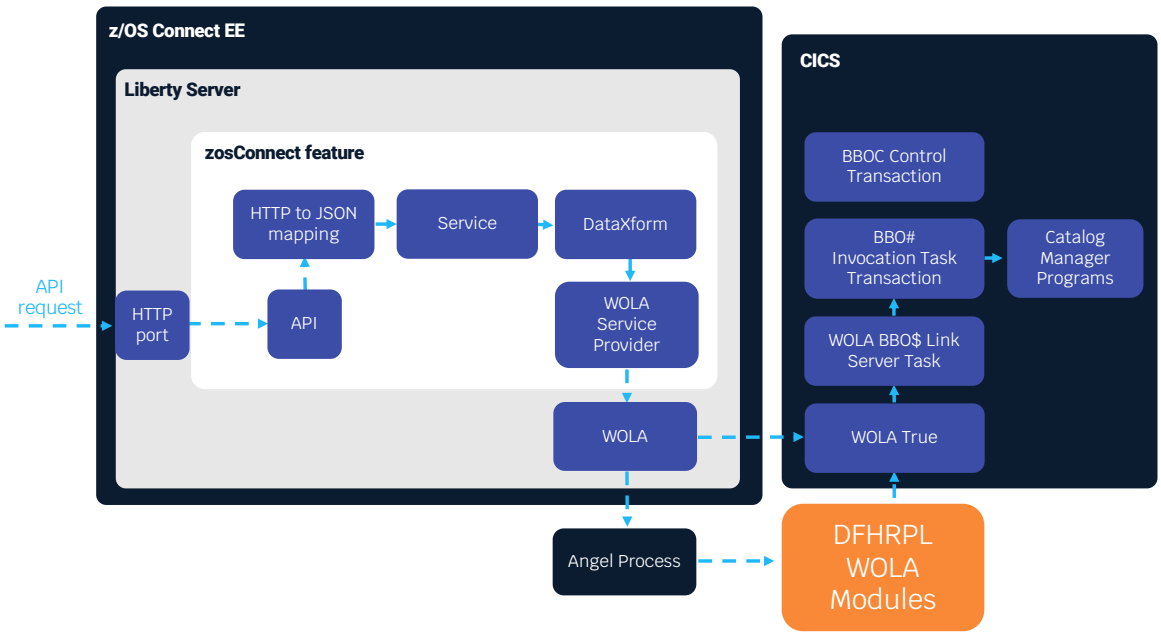
For example, IBM z/OS Connect requires not only a very recent release of z/OS and CTS, but in some cases also the installation of: IBM z/OS explorer, WebSphere server, WOLA (WebSphere z/OS Optimized Local Adapters), IBM's API connect and Zos Connect EE server.

2. Black-boxes

Since last-mile connectors are tightly coupled with the legacy applications, they tend to be closed and un-configurable. Changes and customizations must be made in higher-level layers even at the expense of performance or complexity. Dealing with dynamic message formats, request context and stateful invocations becomes a labor-intensive and time-consuming effort.

3. Lifecycle automation

Another problem arising from the closed nature of these connectors is their inability to support DevOps-type automation. Testing, versioning and deployment of the APIs produced by these connectors are all proprietary and are hard-to-integrate with the normal development lifecycle. This leads to prolonged development times and longer release cycles which impact velocity and agility.



4. Heavy infrastructure needed

When using connectors which are just the last mile of the integration and require many other layers to produce a solution, the architecture topology and infrastructure needed will be heavy, multi-layered, slow and complex.

These architectures stand in contrast with modern, light and flat microservices architectures which require almost no middleware and treat integration as a feature rather than a hindrance. It is all too common for organizations to deploy a modern state-of-the-art microservices architecture on top of an old-style ESB integration stack, preventing them from enjoying the benefits of their effort.

03

Asynchronous message queues

Again, asynchronicity is not a new integration concept. During the 1990s and 2000s it became a standard for Service Oriented Architectures (SOA) and was ubiquitous with products like IBM's MQ.

While there might be many reasons to use the asynchronous messaging model, there were two factors that contributed to its past success:

- Asynchronous messaging correlated well with SOA's orchestration concepts such as pub-sub.
- Asynchronous messaging provided fault-tolerance using guaranteed-delivery capabilities which were required in regulated industries and played an especially important role in a world where hardware availability and scalability was limited compared to today.

The price to pay for these asynchronous solutions was mainly in complexity and performance (not considering the actual price tag on middleware (e.g. MQ) products which might be substantial). This price seemed relatively small in a world driven by limited, internal data-consumers, already complex architectures and limited velocity needs.

Today's requirements pose a difficult challenge to these assumptions: fault-tolerance can be achieved using new approaches, consumers of data are much greater in numbers and variety, architectures are reduced in complexity and the need for velocity in deploying changes is greater than ever.

This, of course, does not mean that asynchronous messaging is not relevant or should not be used in modern architectures, but it does change the calculations on where and when it should be deployed. Simply put, message-queuing should not be the default anymore, but instead a thoughtful decision.

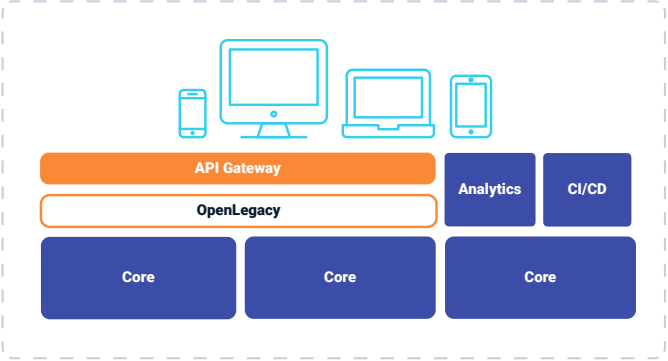
The price to pay for these asynchronous solutions was mainly in complexity and performance (not considering the actual price tag on middleware (e.g. MQ) products which might be substantial).

04

OpenLegacy's solution

OpenLegacy provides a solution to many of these issues and represents a new, modern and unique approach to integration. By leveraging concepts like code-gens, microservices and standard open-source frameworks, OpenLegacy provides the next generation of integration platforms.

Instead of black-box connectors hidden behind layer after layer of integration, OpenLegacy automatically generates a single deployable unit which is a microservice ready-to-run. This microservice consists of an API interface, a business logic payload and a Java SDK which wraps a legacy functionality.



OpenLegacy automatically generates a single deployable unit which is a microservice ready-to-run.

The entirety of this artifact is visible and changeable with readable standard Java code using annotations.

With this approach, no set-up or additional infrastructure is needed. The microservice is deployable anywhere and communicates natively with the legacy system. No third party products, additional installations or legacy skills are needed.

Since the entire code-base is available and visible, any change in business logic, integration logic, channel logic or connector logic can be made either directly to a specific microservice or, using a template, to all generated microservices. This also allows for a seamless integration with DevOps and automation processes using standard tools such as Git, Jenkins etc.

The nature of the solution allows for a flat integration architecture by matching the application architecture and essentially enabling integration without middleware. Fault-tolerance is managed using circuit-breakers and horizontal scalability, as well as log-analysis for recoverability.

OpenLegacy provides a solution to many of these issues and represent a new, modern and unique approach to integration.

Summary

The increasing importance of integration and changing needs requires new and modern approaches. Traditional approaches such as real-time connectors fall short because of their closed, proprietary and limited nature. They require a large complex middleware infrastructure to support them. While still viable for certain cases, message-queuing approaches should not be the default way of integrating. OpenLegacy provides a solution which solves the challenges of traditional integration while providing a way to rapidly generate microservices delivering core functionality.

About OpenLegacy

OpenLegacy's Digital-Driven Integration enables organizations with legacy systems to release new digital services faster and more efficiently than ever before. It connects directly to even the most complex legacy systems, bypassing the need for extra layers of technology. It then automatically generates APIs in minutes, rapidly integrating those assets into exciting new innovations. Finally, it deploys them as standard microservices or serverless functions, giving organizations speed and flexibility while drastically cutting costs and resources. With OpenLegacy, industry-leading companies release new apps, features, and updates in days instead of months, enabling them to truly become digital to the core.