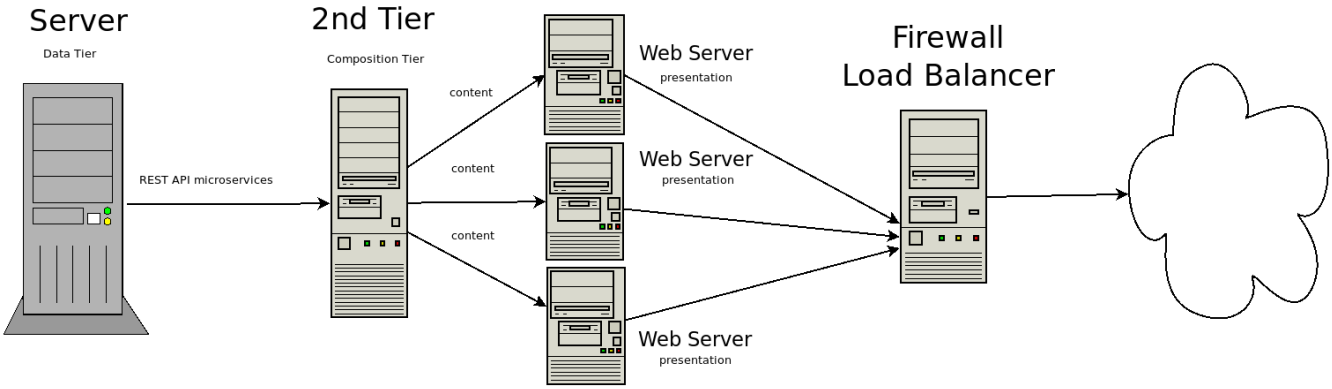
Web-based legacy modernization for IBM™ i using Open Source solutions

"Let Linux be Linux, and let i be i!"



Jack J. Woehr

jwoehr@softwoehr.com

SoftWoehr LLC

PO Box 51
Golden CO 80402-0051

Abstract

Webified IBM i service delivery over private intranet and/or public Internet is the goal. The challenges experienced by legacy organizations are examined. Prevalent practices and tool chains are summarized. Development pathways striving for the virtue of simplicity and employing open source exclusively or nearly exclusively are evaluated.

Table of Contents

Challenges

Expectations

After an era of exodus, businesses remaining dependent on installations of the IBM i are increasingly challenged to deliver solutions in the realm of modern intra/Internet applications.[1]

Increasingly, client platforms are mobile devices, so solutions from previous decades such as 5250 emulation in the browser and predominately static web pages are no longer competitive. Webification of 5250 applications has achieved some presence, but the architecture of such applications does not

1The same challenges apply to other the legacy communities such as z/OS and z/VSE. Experience with those platforms colors our view of the problem under discussion.

always meet the expectations of the modern user.

The problem goes beyond presentation. There is the need for a new era of application development to further empower the end user, deliver more sophisticated analytics, while maintaining system security.

Development

The criteria and techniques of designing and deploying web and mobile applications are generally understood but incompletely assimilated within the legacy community.

Many factors contribute to the difficulty most organizations encounter in the course of legacy modernization. Among those factors are:

- The small number of programmers in the legacy community overall compared to Windows and *nix communities.
- The small number of legacy programmers having in-depth knowledge of web architecture and methods.
- In an industry which is accustomed to having few available choices but having those choices stamped with industry-wide acceptance and approval, there is a lack of clarity as to what truly are the mainstream choices for legacy modernization.

The appeal of multi-tier

In addition to the basic development challenges of legacy modernization, organizations which achieve some level of webification native on IBM i often discover the need to migrate towards a multi-tier architecture where the server is the data source and/or content source (e.g., providing data in JSON or XML) and another platform, such as Linux, is the web server and presentation layer.

Our cover illustration depicts schematically a three-tier solution.

- The Server (IBM i) is providing data and program semantics via microservices.
- The 2nd tier is validating and authorizing requests, relaying them to the server, and composing the returned data into content.
- The 3rd tier web servers are building the visual presentation and serving the end user.

Among the attractions of multi-tier are:

- The IBM i is a less-than-ideal outward-facing web server for a variety of convenience, performance, and economic reasons.
- Security (no direct end-client route to the server)
- Division of effort by platform and the ability to replace the presentation layer at any time without rewriting server-side code.
- Ease of presentation tier development on the defacto platform of reference (Linux) for the Web

These last two points have been summarized in the slogan, "Let Linux be Linux, and let i be i!"

Tiers and fears in Windows™ shops

Some businesses interested in open source modernization of their presentation tier lack Linux knowledge or already are committed to Windows, C#/Visual Basic, and .NET. In this case, you might find extending your Windows Server infrastructure as one tier of a multi-tier solution to be a good fit. C# is a robust language, and Microsoft's recent efforts to become a good open source citizen means that the ecosystem is growing. If the server-side service strategy is sound, the presentation tier can reside on any reasonable platform.

Prevalent practice

The IBM i since AS/400 days has hosted web services. NET.DATA was the AS/400's first foray into higher level web application design methods, and its metatag paradigm lives on in modern open source approaches such as Embedded JavaScript templates.

Java soon dominated the approved solution matrix as IBM brought WebSphere to all its platforms. WebSphere is problematic as an IBM i legacy modernization approach in the present era for a number of reasons.

- WebSphere, due to its resource demands, was and is perhaps less well-suited to IBM i than to z/OS. There has been a steady stream of off-migration of WebSphere applications to AIX and Wintel platforms where they tend to become legacy maintenance problems in their own right.
- J2EE itself is becoming a legacy approach. The number of those advocating J2EE's complex architecture in comparison with more loose-jointed modern frameworks is in decline, as are the number of convincing arguments as to why J2EE is necessary. It will endure for decades due to large transaction systems, but J2EE is clearly in its long afternoon.
- In the current market, the ability of legacy shops to attract programmers who remain working in J2EE is weak.

Several vendors offer proprietary paradigms and toolchains, often based in part on open source. These offerings exhibit a wide range of tradeoffs in price, design characteristics, ease of development, effectiveness, support, and comfort level.

Once aboard, it's hard to leap off the train. In choosing one of these supported offerings, the obvious heuristic is to pick the one most resembles what your team already knows.

Others are pursuing simpler modernization strategies and turning to open source to the greatest extent possibly. IBM is certainly encouraging this trend with its heroic open source efforts on the PASE side of IBM i, efforts which are transforming IBM itself and the way it delivers components of IBM i.

Open source on legacy

IBM, which has been deeply involved in open source from early Apache web server days, has year by year increasingly encouraged and empowered the adoption of open source solutions. Nowadays, IBM i has gone all-out in supporting open source, delivering and tuning the PASE environment and populating PASE's application space with RPM archives of major tools and languages ported for that

purpose by IBM employees and delivered via the `yum` installation manager.

"Mostly-open-source"

In the course of this evolution, PHP at first dominated the "mostly open source" approach to legacy modernization on IBM i, having arrived first and been blessed by IBM's partnership with Zend.

PHP by its nature requires a runtime engine. The Zend organization emerged early in the PHP history with a layered-license engine that far surpassed other alternatives. Zend licenses an enhanced runtime and toolkit to the enterprise while their core engine continues to dominate open source usage.

Zend's toolkit achieves IBM i database connectivity via IBM connectors and execution of native programs and service programs via open source XMLSERVICE which the toolkit wrappers in extension modules.

Multi-tier PHP

Many thousands of lines of very effective PHP code have been written native on IBM i.

Multi-tier PHP presents the issue of the database and command execution connector, which is native on the IBM i. For remote connectivity to XMLSERVICE one needs a separate and pricey IBM licensed program, the Db2 CLI Connector. There is movement on this issue in IBM and in the marketplace, as well as one or two emerging open source solutions.

In the meantime, multi-tiering prior native PHP applications is an effective and oft-adopted strategy.

Other open source approaches

Perl

We should briefly mention Perl, which was the first interactive web application language widely deployed on the World Wide Web. Perl is present on PASE, but has little or no traction as an IBM i legacy webifier, as the era of Perl CGI is long past.

Python

Python makes building bidirectional Python-to-Python bridges between ILE and remote systems very easy.

REST[2] [3]

The compelling modern method of delivering web services is to offer REST APIs. There are variants and followers-on to the REST paradigm, but since the mid-2000's "REST-like" has been dominant.

---

2 Architectural Styles and the Design of Network-based Software Architectures, Roy Thomas Fielding, Doctor of Philosophy in Information and Computer Science, University of California, Irvine, 2000

3 REST API Tutorial

Ideally, REST partitions web applications into exchanges between the client and the server where each client request encapsulates all the state the server needs to fulfill the request.

In turn, the server responses offer resource discovery, guiding the client to the resources it will need to further its request.

REST is

- flexible
- language-agnostic
- well-characterized
- mature
- comprehended by the entire web development community

A full discussion of REST is beyond scope here, but an excellent jump into the pool at the deep end is the paper RESTful Service Best Practices by Todd Fredrich.

Reasonable development strategies do not adhere too closely to rigid ideology. "REST-like" flexible APIs are probably the best path in this regard as long as they are accessible intellectually to practitioners of interactive services design. JSON remains a very easy-to-use data interchange format.

Microservices

The most compelling development paradigm to emerge in the REST revolution is the microservice boom. The simplifying insight, "Do one thing well," has resulted in architecture where service delivery via REST APIs has become less monolithic and better factored.

Individual microservices are sometimes specified, architected, coded and spun up for testing the same day.

The microservice paradigm is language-agnostic. Dominant in microservice coding today are Node.js and Go language, but it doesn't really matter: the twin pillars of microservices are

1. Factoring
2. REST API

which can be implemented in any major IBM i language and a few minor ones.

Microservices on the web, coded in a variety of languages, have achieved wide adoption and have been successful. The *New York Times* has published its own internal microservice framework called Gizmo, which is coded in Go language.

A passable examination of microservices by vendor OpenLegacy targeting IBM i is Accelerating the Digital Journey from Legacy Systems to Modern Microservices. OpenLegacy's offers an IDE-hosted framework in Java for rapid development, deployment and management of microservices on a second tier communicating with hosts and servers primarily via JDBC.

Microservices in the context of IBM i

Without adhering too rigidly to one or another ideological definition of the paradigm, on IBM i, a microservice "does one thing and does it well", leveraging where possible

- extant programs and service programs
- extant queries and stored procedures
- new and emerging facilities of Db2

The last point can hardly be emphasized too much. Alongside the open source revolution on IBM i, the dynamism of Db2 at the present time is striking. Microservices can be service wrappers around clever Db2 snippets. All this is in line with, "Let i be i."

Pitfalls

Microservices are not a panacea and offer their own development pitfalls.

- It is possible to create unnecessary overhead spinning up threads and communicating between them.
- Maintenance can become burdensome when factoring monolithic codebases into several smaller ones.
- Administration can offer challenges, especially in tracking versions of **`require`** components installed by NPM.
- No paradigm will magically rescue a project whose design is flawed to begin with.

Node.js

Node.js is the first or second most popular microservice language. The Node.js world is factored into user-provided packages, thousands of them, managed by a delivery system NPM, the Node Package Manager, that embeds desired packages into your own microservice package, taking dependencies into account automatically as well as updating them on request, packaging your own package, and deploying and launching it.

Node.js is present on IBM i and maturing rapidly. The **`idb-connector`** provides Db2 connectivity and the **`itoolkit`** allows calls to native programs via XMLSERVICE.

Among the Node.js packages are several microservice frameworks that make delivering a well-factored collection of microservices easy, or at least easier.

As well as RDi, Javascript-aware editors such as Atom, jEdit, and VSCode can edit files in the IBM i Integrated File System (IFS) via SFTP. The developer logs into PASE via SSH with a project view open in a favorite workstation source editor.

Miscellaneous Open Source

There continue to emerge new frameworks and tools targeted at IBM i. There are many open source projects useful in stringing together multi-tier services. Examples include:

- db2sock provides a db2 driver native on PASE which can be called from an arbitrary scripting language.
- ILEastic is a C and RPGLE project which allows microservices to be coded in RPGLE . It embeds an HTTP server in your ILE RPG code.
- YAJL Yet Another Jason Library provided by longtime IBM i guru Scott Klement to support JSON in RPG.

Conclusion

There are many possible strategies for delivering legacy modernization and transformation via web applications. We have examined a few here, but not exhaustively.

Using open source predominately or exclusively is quite attractive as the tool and language implementations mature on IBM i and the available packages to install on command multiply.

Multi-tier architecture is particularly compelling, allowing the core services layer to mature and stabilize native on IBM i independently of more rapidly changing presentation requirements, as long as a contractual interface, e.g., REST or some variant thereof, is maintained.

With a view stretching from the present to the maintenance future, an obvious open source strategy is to code microservices on the server (likely in Node.js) to be leveraged by a composition tier which delivers content to a presentation tier, all behind a firewall and/or load balancer such as OpenBSD. The 2nd-tier and 3rd-tier can reside on one or more Linux VMs in the data center.

These Linux VMs could even be on the same Power™ server which is running the IBM i instance.

Acknowledgements

Useful criticism and help with content were provided by:

- Aaron Bartell
- Calvin Buckley
- Bill Gravelle

Jack J. Woehr
2018Q3
Fairmount, Colorado