Going Serverless: The Checklist for Enterprises with Legacy Technology

Introduction

The bank's chief architect was getting nervous.

It was 2018, nearly a year since the bank, one of the largest in North America, announced a sweeping digital transformation initiative to improve customer experience and enhance risk management.

The Board of Directors announced the initiative with great fanfare and high expectations. But in the year since launching, the bank only managed to deploy two digital services.

The project was plagued with challenges since the outset. The bank's initial approach used a heavyweight middleware/ESB to access functions residing in the mainframe environment and expose them in a cloud environment through APIs. But manual and error-prone coding, including custom Java development, meant long release cycles of 6-8 weeks for each API – too long to even think of using DevOps.

And the bank faced a challenge familiar to many large organizations: a shortage of specialized engineering talent, specifically the high-availability and scalability expertise to manage cloud deployment.

The bank's chief architect knew the success of the project was now at stake. Looking at the possibility of failure, he made a high-stakes decision: to embrace a serverless architecture to launch the new digital services.

The decision was far from obvious. The bank had never previously leveraged serverless functions, and in fact lagged its peers in the adoption of a microservices architecture. The technical leadership was skeptical that serverless was the right direction for an institution with heavy legacy infrastructure.

But the adoption of serverless technology transformed the bank. Within just a few months, the bank rapidly increased its development and deployment velocity, moving from months to mere days to develop and deploy functions.

This meant alignment with DevOps/NoOps for velocity and scale, including specialized tests and quality control indicators. The bank redeployed software engineering teams from server operations to core business challenges, ultimately saving millions of dollars.

Many enterprises with legacy infrastructure can benefit from making serverless architecture a core part of digital transformation initiatives. While there are plenty of resources out there on serverless, these organizations often lack guidance around how to bring their legacy architecture into a serverless strategy.

Along with highlighting the unique challenges and opportunities of serverless adoptions for enterprises with legacy technology, this ebook offers a serverless "launch plan": a checklist for organizations to ensure they're prepared to take off with serverless.

Let's get started.

Serverless: Why Now?

First a bit of terminology. The term “serverless” is a misnomer; there’s always a server involved in processing requests. Serverless refers to an architecture in which the server is provisioned and managed by a cloud provider.

This isn’t unique to serverless. What is unique is the degree to which the cloud provider removes the burden of managing the run-time environment. Microservices containers require a server to be provisioned around the clock even if there is no load. In contrast, a serverless architecture refers to one in which procedures (“functions”) that perform a single service are handed over to a function cloud to run in real time. They execute only when needed, after which time the computing instance shuts down.

As a result of this distinction, serverless architectures can be both significantly more cost-effective (since they operate on a “pay-as-you-go” model) and scalable (since server load is dynamically managed) than existing architectures. At the expense of added complexity, they unlock the potential to innovate in fundamentally new and faster ways.

In this context, serverless is situated within the industry’s long arc towards optimization for greater flexibility, speed of innovation, and ease of new technology adoption. This shift began several years ago with the shift away from monolithic architecture towards microservices – as a way of harnessing the power of APIs to link information systems and deliver more nimbly and scalably on customer needs. (See "The Great Revolution.")

The Great Revolution*

Over the past decade, three factors contributed to a massive leap forward in companies’ ability to harness IT to serve their customers.

- **The what:** application programming interfaces (APIs), guidelines that govern how systems interact with each other. The explosion in popularity and adoption of the web API over the past decade made it possible for business to share information like never before.
- **The how:** microservices – small, self-contained pieces of code that execute a business function. Microservices offered an alternative to monolithic architecture, a single, unified, indivisible application. Microservices allow teams to develop and iterate on new features without rewriting large portions of the codebase.
- **The where:** the cloud as an on-demand resource for data storage and computing. Cloud computing has offered the scalability and cost-effectiveness required to power new applications, even in traditional industries like financial services.

The on-demand connectivity of APIs, the agility unlocked by microservices, and the scalability of the cloud have led to huge advances in the ability to meet customer needs. Serverless, which even further decouples digital services from infrastructure management, is the next frontier in this revolution.

*Adapted from Romi Stein, “Why Now is the Time to Go Serverless”

Serverless is the natural extension of this progression, offering even greater granularity and ease of development and deployment. Additionally, serverless as an architecture is particularly well-suited to times of rapid change and disruption.

- **Cost savings:** Serverless is a natural choice for organizations in budget-conscious mode for two reasons. First, serverless only charges for the period during which the server executes the function. Second, the cloud provider takes on the burden of managing and optimizing server capacity, freeing up human capital to be redeployed to critical projects.
- **Adaptability to changing customer behavior:** it is difficult for businesses to make accurate long-term infrastructure investments when customer behavior, regulations, and the public policy environment shift regularly. Serverless automatically and dynamically scales up and down server capacity, providing needed flexibility as conditions change.

Benefits of Serverless

Serverless offers the promise of unlocking additional agility in development and deployment, cost savings, and overall acceleration of innovation. But what does it actually look like in practice when organizations with legacy technology adopt a serverless architecture?

Based on the results from the bank described in the introduction, the outcome from large-scale serverless adoption can be game-changing: millions of dollars of cost savings a year, and an enhanced competitive stance around driving digital transformation initiatives through greater agility and innovation, cost savings, and focus on core business challenges.

We recommend that technology leaders creating a business case for serverless adoption break out anticipated benefits as the bank did below in order to create a vision of what success would look like for their organization.

Agility

Before serverless adoption

- 6-8 weeks to develop and deploy a single service
- Inability to adopt DevOps due to long cycle times

After serverless adoption

- 15 minutes development and 1-2 day deployment per function
- Alignment with DevOps/NoOps

Cost

Before serverless adoption

Millions of dollars a year on governance and provisioning costs

After serverless adoption

Significant reduction in millions of instructions per second (MIPS), with cost savings of roughly $1 million/year

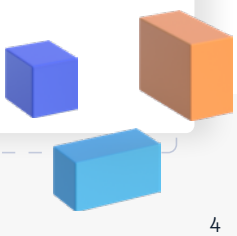
Focus

Before serverless adoption

Internal software teams spent managing server operations

After serverless adoption

Redeployment of teams to more business-critical areas – accelerating innovation around customer experience



Challenges for organizations with legacy technology

Organizations with legacy technology face a variety of challenges on the path to adoption of any kind of digital architecture.
Legacy systems are often:

- Difficult and idiosyncratic to parse (e.g., legacy systems like CICS, IMS, Tandem, Unisys, SAP, Oracle, and VSAM all come with complex application software architecture that requires difference metadata parsers); and
- Not built for easy interoperability with other software applications, typically requiring highly specific connectors.

A common approach for enterprises with legacy technology is to use middleware, often an enterprise service bus (ESBs), in order to integrate legacy infrastructure with other parts of the technology stack.

But leveraging middleware/ESBs to enable digital services on top of legacy systems spawns additional complexity, maintenance requirements, and cost .

One architect referred to this as a "lasagna architecture" that impedes the goals of agile development and deployment.

These challenges are not unique to serverless architecture. They exist for organizations looking to build a microservices architecture on top of their legacy assets. But they create particular stumbling blocks for organizations deploying serverless functions.

Serverless functions execute only when needed, after which the computing instance shuts down. This places particular pressure on the function's ability to initialize quickly to avoid unacceptable lag in serving requests.

Middleware/ESB integrations rely heavily on Java implementations, resulting in large function size and a run-time library that introduces latency (see the "Serverless Adoption Readiness Checklist") – making them a major obstacle in serverless adoption.

Any viable serverless solution for an enterprise with legacy technology must connect directly to the underlying legacy systems, without changes to the backend.

The Serverless Adoption Readiness Checklist

As we've seen, enterprises with legacy infrastructure stand to gain tremendously from incorporating a serverless strategy into digital transformation efforts. But the legacy-specific challenges of integration and performance should give some pause.

For these organizations, "going serverless" shouldn't be a spontaneous exploration. Rather, it's helpful to think of it as a highly-coordinated launch: one that requires a checklist to guide critical preparations and decision-making before, during, and after.

Here's the checklist that enterprises with legacy technology use to ensure that serverless adoption is a success.

Step 1
Flight Plan

Key Objective:

Determine scope. Which business problems make sense to solve with serverless (versus other architectures – e.g., microservices)?

Best Practice:

Microservices and serverless complement each other, offering different strengths. Microservices work best for long-running or computation-intensive jobs, while serverless is a good fit for event-driven jobs that benefit from automated scaling.

Flight Plan In Action:

A leading Latin American bank was stifled by post-merger legacy systems spread across multiple countries. Mainframe programmers used COBOL in Colombia, Java programming was done in Panama, and a third-party global systems integrator maintained the infrastructure. As part of its digital modernization initiative, the bank evaluated a number of improvements to customer experience. They ultimately decided on AWS Lambda serverless, for their innovative payment processing portal – a perfect fit for the type of short-running, event-driven requests.

Step 2
Systems Check

Key Objective:

Define an integration strategy. How do you plan to connect a serverless architecture to your legacy systems?

Best Practice:

Improve operational efficiency by bypassing layers of complex ESB/SOA-based middleware. Seek a vendor providing a direct, digital integration with your legacy systems (mainframes, midrange systems, and other databases or applications) – without any changes to the backend.

Systems Check In Action:

A leading European financial institution had a vision of being the "bank of the future" which meant a best-in-class digital experience for customers and innovative new banking partnerships. But all of its core data resided in an IBM z/OS IMS DC mainframe environment. It struggled with managing multiple outsourced IT vendors, who were bogged down by multiple layers of middleware. Ultimately, the bank determined that in order to accelerate towards its vision, it needed a digitally-native IMS DC integration offering direct mainframe connectivity – without any middleware.

Step 3
Countdown

Key Objective:

Plan a development approach. How will your organization accelerate the development and deployment of serverless functions?

Best Practice:

Development and deployment can be slow, manual, and error-prone if developers must "reinvent the wheel" with each function. Use a low-code environment with automated code generation capabilities and templates to significantly speed the process end-to-end.

Countdown In Action:

A bank trying to implement serverless functions discovered that developers – who were working in a fully manual coding environment – were spending unnecessary time creating functions, and that the resulting functions were error-prone and difficult to maintain. The bank moved its serverless development to a platform offering auto-code generation capabilities and standard templates. As a result, the bank saw a significant reduction in the time required to develop new functions. The standardization among its serverless code base meant less time required to onboard new digital developers and less time spent on function maintenance over time.

Step 4
Blastoff

Key Objective:

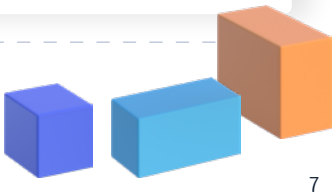
Solve for performance challenges. How can your organization overcome the "cold start" challenge that pervades serverless deployment, particularly with legacy systems?

Best Practice:

Java libraries (used extensively by ESB and middleware providers) lead to significant latency in initializing serverless functions. This latency is problematic for two reasons. First, it creates an unacceptable user experience – for example, a response time of several seconds to a critical customer request. Second, it ends up directly increasing cost, since functions charge by execution time. To avoid this issue, bypass the middleware or ESB and directly access a digital representation of the legacy asset – and leverage a lighter-weight Node.js implementation for significantly faster initialization.

Blastoff In Action:

Despite having a variety of serverless functions ready for deployment, a North American bank with a mainframe architecture struggled with performance. Their functions were built on their ESB and invoked as a Java library to access mainframe data. The functions took 2-4 seconds to initialize: an eternity for customer-facing processes like account login. To solve the problem, the bank migrated to a direct digital integration with their mainframe – and a Node.js-based environment purpose-built for serverless runtime. As a result, they saw a 20x performance improvement, reducing "cold start" latency to just 100-150 milliseconds.



Step 5
Orbit

Key Objective:

Create a strategy for monitoring and maintenance. How can your organization track functions running in production and continuously reuse mainframe data for new service development?

Best Practice:

Invest in a platform to help organize and monitor your serverless functions built on legacy technology (particularly function performance and usage). Additionally, ensure that mainframe data used in functions is continuously available to the digital development team – both to simplify maintenance, and to ensure reuse in subsequent digital development efforts.

Orbit In Action:

After implementing a serverless architecture to support customer-facing applications, a financial institution successfully launched dozens of functions. But without mainframe application data easily accessible to the team, it was time-consuming for digital developers to perform maintenance on existing functions and to launch new functions. The team identified a solution to enable them to reuse and integrate mainframe application data with digital services over a unified platform, significantly reducing maintenance time and accelerating the development of new services.

What's Next?

Serverless technology represents an extraordinary opportunity for organizations with legacy infrastructure. It allows them to reallocate time and resources from infrastructure maintenance towards core business problems, accelerating the process of launching digital services. The results speak for themselves: agility and innovation, cost savings, and greater focus on core strategy and customer experience.

Nevertheless, enterprises with legacy technologies face unique challenges: integration with legacy systems and performance obstacles like the "cold start" problem. These organizations need a pragmatic, strategic approach to overcoming these obstacles.

It's our hope that the Serverless Adoption Readiness Checklist helps equip you with the tools to do so.

Wherever you are on your serverless journey, it's our hope that the Serverless Adoption Readiness Checklist equips you with the tools to accelerate FaaS adoption in your organization. The checklist represents best practices distilled from enterprises with legacy infrastructure that are using serverless technology to innovate and serve their customers more effectively.

We wish you the best of luck on your own serverless launch!

About OpenLegacy Serverless

OpenLegacy Serverless is the first natively-serverless hybrid integration platform built to enable organizations with legacy backends to rapidly build, deploy, and run serverless APIs. The technology extends OpenLegacy's patented engine for core legacy application integration with a new serverless offering that delivers out-of-the-box scale and high availability, eliminates cold start latency, and accelerates development with a low-code platform.

Learn more about OpenLegacy Serverless and try it free at serverless.openlegacy.com.

Rapidly Build, Deploy, and Run Serverless APIs

Out-of-the-box scale and availability, stunningly low latency, and a low-code platform to accelerate the development and deployment of serverless functions.

Seamlessly Integrate Legacy Systems

Patented technology for automated integration with mainframes, midrange systems, and other legacy systems – without any changes to the backend.

Maximize the Potential of Existing Cloud Investments

Built to run on AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions.