WHITE PAPER

MICROSERVICES: THE GOOD, THE BAD, *AND THE BETTER*

USING MICROSERVICES AS A WRAPPER FOR A FASTER, LESS RISKY APPROACH TO DELIVERING NEW BUSINESS VALUE

PREPARED BY

JBS
Custom Software Solutions

TABLE OF CONTENTS

JBS Custom Software Solutions

Like more and more developer organizations these days, JBS Custom Software Solutions is a big believer in using microservices to speed the efficiency of software development, delivery and maintenance. This is especially true where lots of custom, legacy, or third-party applications are involved—the reality in which most larger enterprises find themselves. Over the years, companies have accumulated a lot of systems-related IT baggage. When a major initiative needs to leverage the services and data all those systems provide, they must deal with that baggage.

Despite the title, we don't think there's anything "bad" about microservices, but there are a few potential pitfalls for the careless or inexperienced. Things had gotten progressively more difficult for enterprise developers as they struggled to customize, integrate, and maintain new features in their monolithic systems. It is imperative that developers and their organizations digitally transform themselves to keep up with the times, rising customer expectations, and their competition.

That's where adopting a microservices approach comes in handy. We know that different projects call for a different approaches, some for which microservices might simply be overkill. We also know microservices aren't perfect, especially when improperly designed and implemented. As a developer-centric organization, we are sold on microservices and the benefits they promise — not to mention innovative ways of using them to deliver tangible business value faster.

This White Paper examines:

- A brief history of microservices
- How using microservices benefits both developers and their customers over continuing to customize and extend existing monolith applications
- A few of the pitfalls for the inexperienced
- A hybrid microservices-monolith approach that enables developers to integrate and extend the functionality of existing systems more rapidly and flexibly, with fewer resources.

> Since its introduction in 2014, over 60% of companies have adopted or plan to adopt a microservices architecture to reduce their time to market. 54% leverage it in their digital transformation efforts.
> — *Source Dzone.*

A (very) brief history of microservices

Just what is a microservice, really? A basic definition is in order.

Although arguably a variant of service-oriented architecture, microservices are a fairly new construct. The term "microservice" first appeared at an event for software architects in 2011, but it didn't gain wide exposure until 2014 when author and developer Martin Fowler started using it in his written works. Microservice architecture is a software architectural style where applications are orchestrated using a collection of loosely coupled, often fine-grained, interconnected services, rather than large functional "chunks." These microservices are usually deployed on containers or on a serverless architecture and communicate via APIs or streaming protocols.

These services are most often organized around specific business capabilities, instead of merely exposing application-centric functions provided by underlying back-end systems or datastores. Clients access these business capabilities via a central API gateway service that routes the requests and responses. Microservices allow developers to implement business logic and processes from the "outside," rather than having to endlessly add custom code into the

In one nutshell...

What are microservices?

Microservice architecture is a software architectural style where applications are orchestrated using a collection of loosely coupled, often fine-grained, interconnected services, rather than large functional "chunks."

This approach enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to more easily evolve its technology stack.

At the other end of the spectrum are monolithic applications, which perform a wide variety of functions (modules) and have a single code base.

> Microservice decomposition isn't just changing your code, it's changing your organization to match.
>
> *— Joseph Rose, JBS Custom Software Solutions*

enterprise's monolithic, legacy applications. And because their code bases are independent, IT DevOps teams can modify, test, and deploy them as needed without requiring massive end-to-end regression testing.

As noted, microservices have only been around for a handful of years. Early microservice pioneers—circa 2015—were primarily high-growth startups developing "net new" software applications and products from scratch. Netflix, the entertainment streaming giant, is a great example. Free from the ball-and-chain of existing legacy infrastructure, applications, people, or processes, the engineers could experiment with a whole new way of orchestrating enterprise services.

However, not every company needs (or has resources) to build an all-new class of enterprise app. (After all, how many companies have thousands of developers at their disposal like Amazon and Netflix?). The technique of breaking down and exposing finely grained business services has caught on with organizations across all industries. Today, over 60% of companies has adopted or plans to adopt a microservices architecture to reduce their time to market, and 54% leverage it in their digital transformation efforts. That means we've gotten a lot better at applying what and how those pioneers did, even if our development teams and budgets are a tiny fraction of their size.

In theory and in practice, microservices means taking something big and cutting it into smaller, more manageable pieces. It's the same approach we apply with modern agile project management principles, with optimizing manufacturing pipelines, with making CPU's faster. Smaller pieces of code—microservices—makes building, maintaining and deploying even complex applications more efficient and flexible.

Why microservices over a monolithic approach to digital transformation?

A monolithic approach presents the following challenges to any digital transformation effort:

- Difficult to turn business objectives into actionable items for multiple teams / competing priorities within different departments
- Deployments can be painful and IT systems can be heavily coupled
- Easy to introduce regressions into the codebase; afraid to touch anything
- Can be tough to scale up or down and to introduce new feature sets
- Painful to change frameworks / update to the latest version or introduce a new technology / large upfront design required

- Fragmented architecture that shifts / fractures over time

In summary: Creating or updating a monolithic application is simply not effective for digital transformation efforts.

THE GOOD: The Core Advantages of a Microservices Approach

What makes microservice architecture so much better than older monolithic application architecture? It's like comparing traditional waterfall project methodology to Agile. (In fact, agility is one of the primary benefits of microservices.)

Agility.

Microservices make your organization more agile.

Conway's Law says that how your software components interact is a reflection of how your organization communicates and interacts. Say you have a team of 40 people that maintain a monolithic e-commerce application with a single code base. Do you know how difficult it is to schedule (much less run) a meeting with that many people? But, if you ever want to modify that monolith, you almost have to involve all those people in some capacity, because such changes impact everyone.

With microservices, the services are independent, therefore the people are more independent. Instead of one monolith and a team of 40, you might have five smaller e-commerce services each with a team of eight. Or even smaller teams, since there is not as much busy-work to figure and handle all the code dependencies inherent in monoliths. Smaller teams are simply easier to manage, and they can get more done in less time.

Performance benefits.

Microservices can automatically scale up or down.

When you need to ramp up the performance of a monolith, you can apply the usual techniques: adding more RAM, faster processors, bigger servers and so forth. But these techniques will only get you so far. When you reach a physical or financial limit, you might then use a technique called sharding, a technique used to decompose a massive database into smaller databases and distribute traffic evenly to improve performance. Sharding is an effective technique that can help even a monolith perform better, although you have to modify the monolith to understand the implementation. Unlike monoliths, microservices can be deployed on independent hardware, so you get scaling and performance benefits of "sharding" out of the box. Unlike database sharding, microservices each have

their own, independent database that holds only the information that service needs. This means that each microservice can independently and comprehensively scale up or down, saving costs by scaling down low traffic services in the cloud. With auto-scaling, your microservice architecture can practically tune itself.

Ability to migrate architecture.

Microservices enable and simplify migrating architectures.

An often-understated benefit of microservices is that it allows you to more easily migrate architectures. Refactoring a monolith means ripping the whole thing apart, rewriting the code, putting it all back together, performing massive regression testing, and then deploying it. And you have to repeat the process every time you want to change it. It's like moving a huge boulder (or even the whole mountain) every time.

With microservices, transitioning architectures is like moving individual stones instead of the whole mountain. Say your current architecture has five services, and your target architecture has seven, and four of the five existing microservices will carry over to your new architecture unchanged. All that's required is to retire one service and build three new ones. Nothing to tear apart and put back together again. Be mindful to make changes like this backwards compatible or implement a transition and deprecation plan.

And there's a host of other benefits...

Besides avoiding the headaches of monolithic architecture, microservices simply make the development cycle faster, more agile, and more productive. It is easier to...

- **Onboard people** because there's less code to look at.
- **Code and maintain** because there are fewer dependencies and spaghetti code.
- **Document the system**—for the same reasons!
- **Set up a deployment pipeline**—you only have to deploy what changed.
- **Test** because you only have to test use cases that involve the components that changed (not a full end-to-end regression test suite).
- **Update frameworks**—and this is a big one. Say you want to migrate from .NET 3.5 to .NET 4.0 or any other framework update for that matter. With a significantly smaller code base as compared to a monolith, microservices make this easier and the potential "fallout" of upgrading those different components is reduced.

Microservices Address the Challenges of a Monolithic Approach to Digital Transformation

- **Removes dependencies** and thus allows for faster development
- Makes deployments **easier and automated**
- Reduces coupling, blocking between teams, and is e**asier to maintain**
- More cohesive architecture and **better scalability**
- Language-agnostic and allows using appropriate tools **for each service independently**
- Easier and **faster introduction of new features** with less overall impact

The Road from Waterfall Monolith to Agile Microservices

Back in the 1970s, when you wanted a software application built, you just told developers to “go do it” and hope they came up with something that worked. You usually got a bunch of spaghetti code and a giant mess. After all, software development was a fairly new thing with few rules. Eventually we formalized a traditional software development lifecycle (SDLC) which included requirements, analysis, design, implementation, and testing phases. After months, (maybe years?) of planning, the developers would at last start coding—and you hoped the resulting system would work. Of course, by then the business’ needs had totally changed—and you were stuck deploying a monolith the business no longer wanted but couldn’t function without.

That resulting monolith is like a giant tapestry of interwoven threads where it’s impossible to tug on one thread and not have a million others come with it.

Starting around 2000, Agile methodology became (and has remained) popular. With this method, you would perform mini sprints of requirements, analysis, design, and implementation. While an improvement over traditional SDLC, it had a lot of the same challenges. Each mini sprint still required the input and agreement of the broader team along the way.

Whether traditional or agile, we still end up with a release train, code freeze, end-to-end testing, regression testing, user acceptance testing and so on. If everything looks good, we set “the big night,” schedule

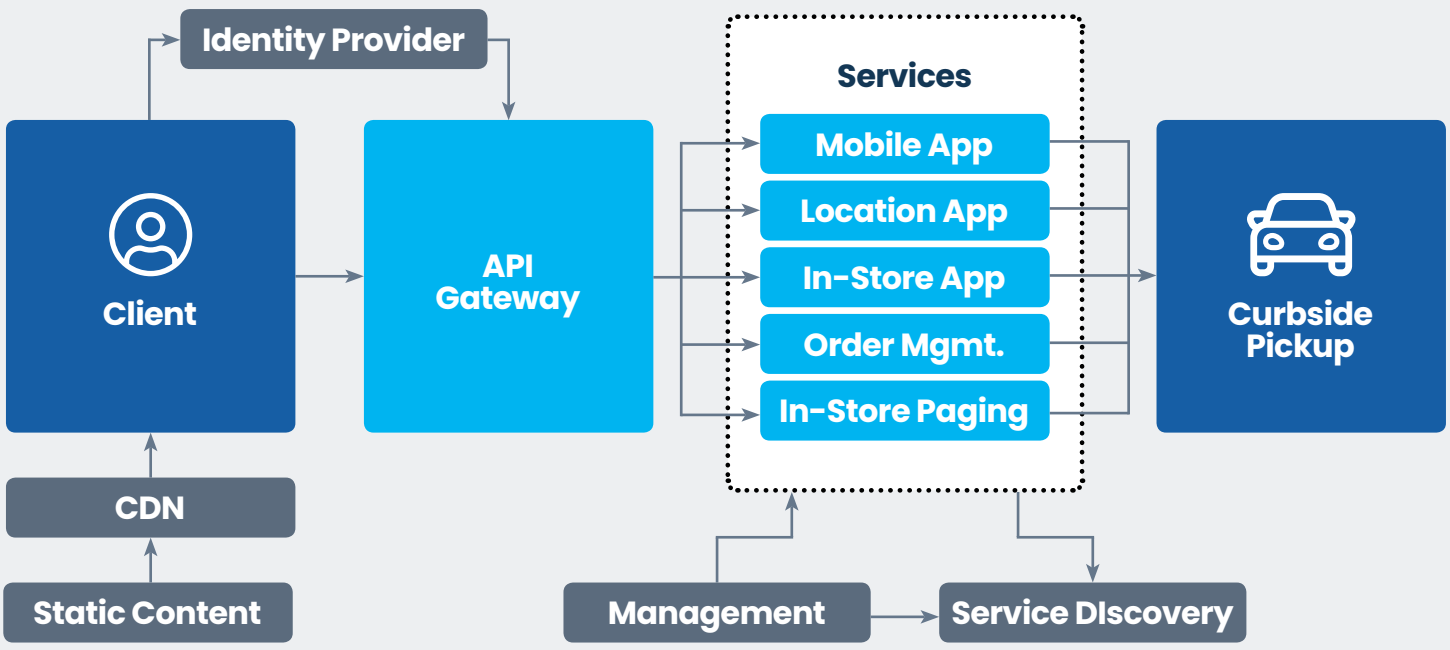
downtime for system maintenance, get a hundred people on a conference bridge—and pray.

With the microservices approach, we decompose the requirements into the services we've identified as necessary and define which services need to communicate with which other services.

Only persons associated with those services need to agree on—for example—what accounting needs from order management or what e-commerce should send the order system. Because these services all work independently, you can start building each one as soon as you agree on the interface requirements. And each can be tested and deployed independently—so there's no need for a "big night" where no one gets any sleep.

When the last microservice is completed and deployed, you simply flip on the switch and you're done.

A microservice blueprint for a retail curbside pickup implementation



THE BAD: Microservices Don't Cure Every Software Development Ill

Microservices solve a myriad of problems for developers, allowing them to deliver software solutions that solve complex problem for their organizations or clients. But as we're fond of saying, "There's no such thing as a free lunch." Microservices don't solve everything, and they come with their own set of issues developers have to work around or solve. Perhaps they're not "bad," but they are things to keep in mind if you want your microservice architecture to be a success.

01. Don’t do microservices just for the sake of doing microservices

First, remember that not every monolith is so large and complicated that it needs to be decomposed. Some are cohesive and well architected, making them easy to code against. As the saying goes, if it’s not broken, don’t try to fix it.

Further, remember that Netflix, Amazon, Google and other “giants” took the lead in defining the early best practices for microservices. However, a lot of these practices only make sense if you’re doing billions of transactions and have an army of thousands of developers at your disposal. That’s why it’s important to frame and scale what you do with microservices to the needs and resources of your own company and the applications you are trying to build and maintain.

02. Coding may be simpler—but beware of multiple services interacting

When you introduce multiple services, you should take into account potential performance considerations of their interactions.

Let’s use an Order service as an example. Say you want to look for patterns in orders for male shoppers between 30 and 35. You might consider first going to the Customer service to get the list of all shoppers that match this criteria, then send those to the Order service to get all their orders. But if there are a million customers in this age range and only 500 that have placed an order, that approach can be horribly inefficient and slow.

03. Be aware of de-normalized data and asynchronous updates.

One way to address the above issue is to de-normalize the customer data into the order data. However, what happens when a key piece of customer data changes, such as someone getting married and changing their name? If your Order history service relies on de-normalized customer data, it will continue to show the old last name, even if the Customer name has been updated. At some point, the Customer service will replicate and update the de-normalized data used by the Order service, but that may take time. While that may be fine for reporting, it isn’t fine for real-time interactions with a customer.

04. Deployments can be tricky, if you’re not disciplined.

If your team is used to deploying a single service (think monolith),

you might now have 15 microservices to deploy. What if you deploy 14 services and the 15th one fails? That's why backwards compatibility is really important. One of the key tenets of microservices patterns and continuous deployments is backwards compatibility. If you refactor a monolith, you don't have to worry about backwards compatibility: you can just deploy the whole thing. When you refactor a microservices, you can't do it without maintaining backwards compatibility, or the other services that rely on it can (and will) break.

05. It can be hard to know how big (or small) your microservices should be.

Knowing the size of how big and how small your microservices should be is really hard. There's a lot of factors you have to take into account, but there's no one deterministic method you can use to decompose and set up your microservices architecture. If you do it right, it's a great pattern. If you do it poorly, it's worse than the monolith.

Think of it this way. If a company the size of Netflix has 1,000 or even 2,000 software engineers, and the maintain say 1,000 services, they've got two people for every service. They're in great shape. On the other hand, if you have five engineers, you can't expect to decompose your business so finely that you have 300 services. Five people maintaining 300 services is impossible.

> When you are new to microservices, knowing how big or how small your microservices should be is really, really hard. If you do it right, it's a great pattern. If you do it poorly, it's worse than a monolith.
>
> *— Joseph Rose, JBS Custom Software Solutions*

06. Service discovery is problematic.

If you have a monolith such as an e-commerce application, you communicate with it at my-e-commerce.com or some such domain name. Easy. If you have a microservice that talks to seven other services, how do you know where they are? Are you going to use a domain name?

What if you need to migrate it elsewhere? What if they added more capacity to the cluster? How are you going to load balance it? What if one service is down? And the questions go on. The

synchronization and orchestration of how you discover routes between services, where services are, and whether they're up or down is significantly more complicated with microservices.

07. Reporting and analytics can be problematic, too.

Most analytics and business intelligence solutions use a central repository for all the data, such as an operation data store, a data warehouse or data lake. What often drives complexity in BI is how many data sources it has to pull from. If you take three monoliths and decompose them into 70 services, that's a lot more data your BI has to pull in for a wider variety of things.

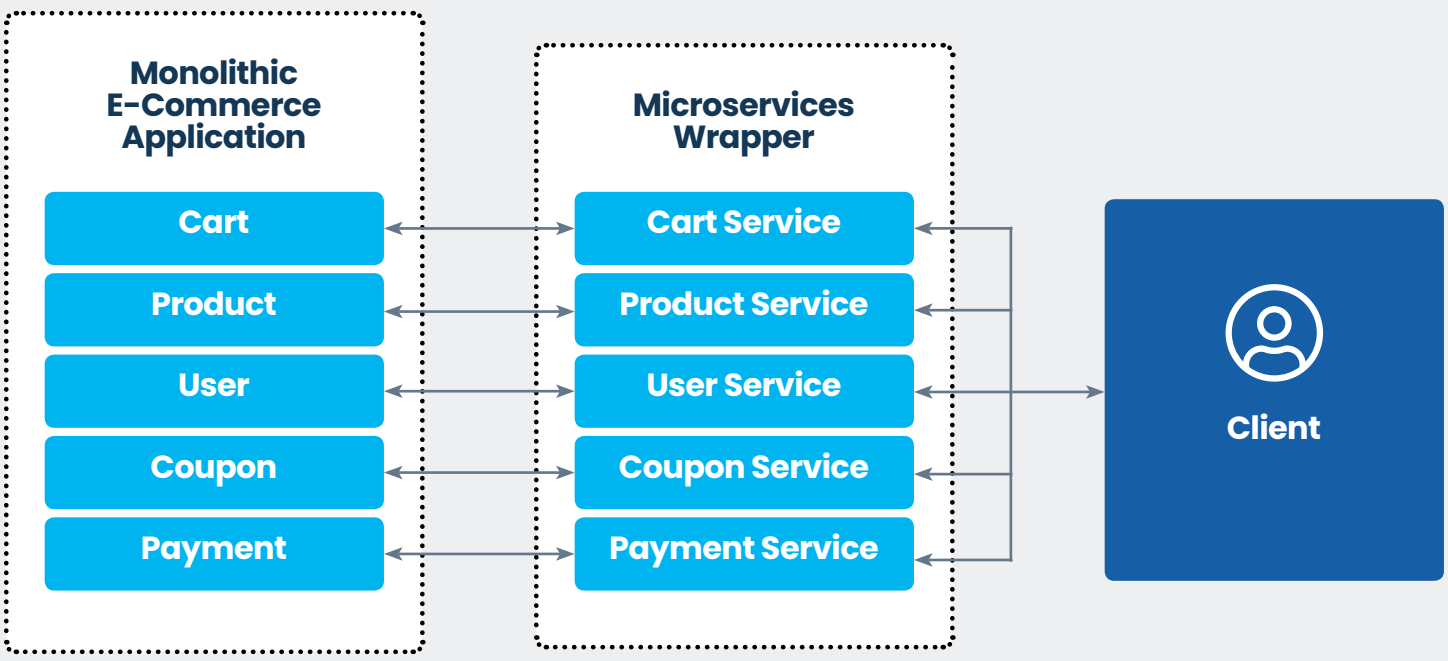
For analytics, you should avoid using a de-normalized copy of the data, even though there are situations where your microservices may need it for efficiency. If you de-normalize your User or Customer data into your Order service, you should never have that service send such data to your analytics or BI component: it's not the system of record. When you decompose your app into microservices, you have to be aware of which data is from the system of record as opposed to a copy being used for a particular service or use case.

THE BETTER: "Decomposing" a Monolith using Microservices as a Wrapper

As you can see, there are a host of benefits that come with adopting a microservice architecture over a monolithic approach—and there are a few drawbacks, too. How, then, can you gain the benefits of microservices and overcome the majority of the challenges they present? One of the methods with which JBS Custom Software Solutions has had the most success is by taking a hybrid approach. Instead of continued monolith customization or a completely purist decomposition into many microservices, this hybrid approach wraps high-level business domains in the monolith within microservices. This "microservices as a wrapper" approach uses either the **adapter pattern** or strangler design pattern.

An **adapter** exposes domain-specific capabilities of the underlying monolith or its components as though they were themselves microservices. This is ideal for a monolith that has fully realized APIs for its various domains—including equivalent APIs for all user operations without requiring any interactions via a user interface (UI). If that pre-condition is met, you can develop services that wrap domains—such as order management, identity services, payments, and so on—or specific components of that monolith and present those capabilities as microservices.

A hybrid microservice-monolith approach to retail curbside pickup implementation



The **strangler pattern** is an extension or evolution of the adapter pattern. While the adapter pattern assumes the monolith will live on forever, the strangler pattern assumes the ultimate goal is to get rid of the monolith altogether. For example, in the diagram above, the Product service might start as an adapter that calls the monoliths APIs to accomplish its purpose. Over time, we can build more functionality into the adapter until eventually it no longer needs the monolith and becomes its own standalone service.

In our example, the monolithic e-commerce application has hundreds of API endpoints covering a wide variety of domains—Product APIs, Cart APIs, User APIs, Order APIs, and so on. You can build “wrappers” (adapter microservices) that call these APIs and orchestrate the capabilities for each domain, making it simple for external client applications to access these business functions. In doing so, you eliminate the need to customize (and maintain) the monolith to accomplish this orchestration, instead putting the business logic in the wrappers or in the client application itself. This provides API-based clients of your microservices feature-rich and domain-specific data, seamlessly decomposing your monolith.

This approach provides many of the same benefits as decomposing the entire monolith application. You can:

- Break the team into smaller groups.
- Negotiate the contracts between a single adapter and an external team (not “the whole company”).
- Deploy each adapter independently.
- Avoid further time- and resource-intensive customizations to the

monolith.
- Migrate more and more of custom business logic from the monolith over time (strangler pattern) if you wish.

Perhaps the biggest payoff of using microservices is having a wrapper that allows your team to build and deploy new functionality much faster—and without disturbing the underlying monolithic engine. You can then cease further customization of your customized monolith and mandate that any new changes will be made in the adapter. And if the monolith is already extensively customized—as is the case for most enterprises—you can start to migrate the custom business logic out of the e-commerce platform and into the adapter.

Getting started: Enabling smooth, transparent migration away from a monolith

In addition to rapidly enabling new business functionality, microservice adapters allow you to start migrating the rest of the business away from the monolith.

The first step is to build the adapter microservices, of course. You then inform the owners of all existing clients of the monolith's services that the adapters are available. Over time, you can build out the adapters into full-fledged services which other client applications can use. Since the interface contracts are defined up front, you can inform the stakeholders of what's coming, should they wish to start their migrations in parallel. In reality, though, it can take significant time before the rest of the enterprise has migrated their apps from calling the monolith to calling the microservice wrappers.

But that's okay: There's no hurry. The monolith continues to function, there's no disruption to client applications that call it directly. The key thing for these stakeholders to understand and accept is that customizations to the monolith will cease. When new business logic is needed for their applications, it should be developed in the new microservice architecture. This allows the enterprise sufficient time to adapt over time—to migrate from monolith to microservices while the engine is still running, so to speak.

A Real-World Example: Using Microservices as a Wrapper for Retail Curbside Pickup

As an example of how the hybrid approach can work in the real world, JBS Custom Solutions was able to deliver a full implementation of retail curbside pickup in only six weeks, start to finish. Here are the primary use cases involved in that effort.

It is interesting to note a side-benefit of using this approach, namely that wrappers present a consistent interface regardless of what services or components are behind them. In this example, the microservice adapter had three distinct types of “back-end” services that needed to be wrapped.

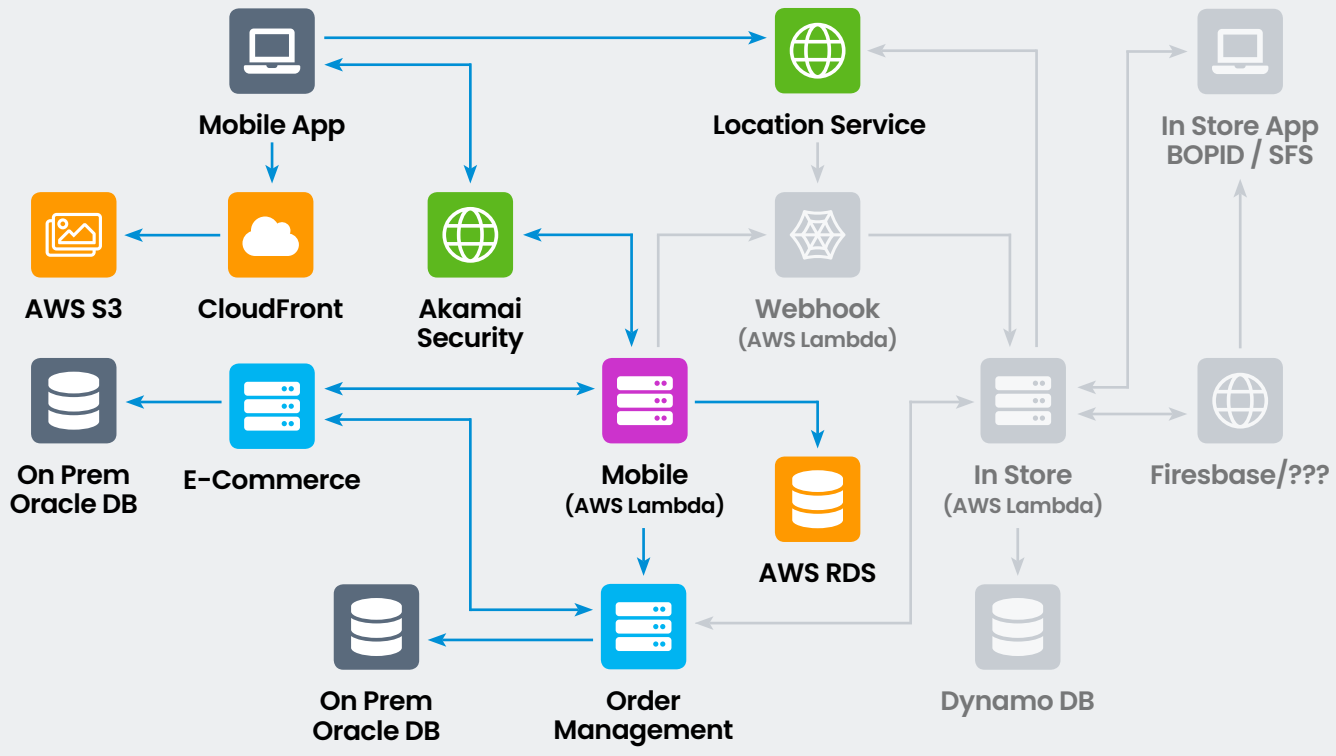
- Monolith with no existing microservice architecture (order management system)
- Monolith with existing microservices that had to be modified while maintaining backward compatibility (e-commerce platform)
- All-new functionality (store location/notification system application)

In all three cases, the microservice wrappers developed presented a consistent interface with which the retail curbside solution could interact to deliver the new business functions. Further, all those wrappers became available for other client applications to call.

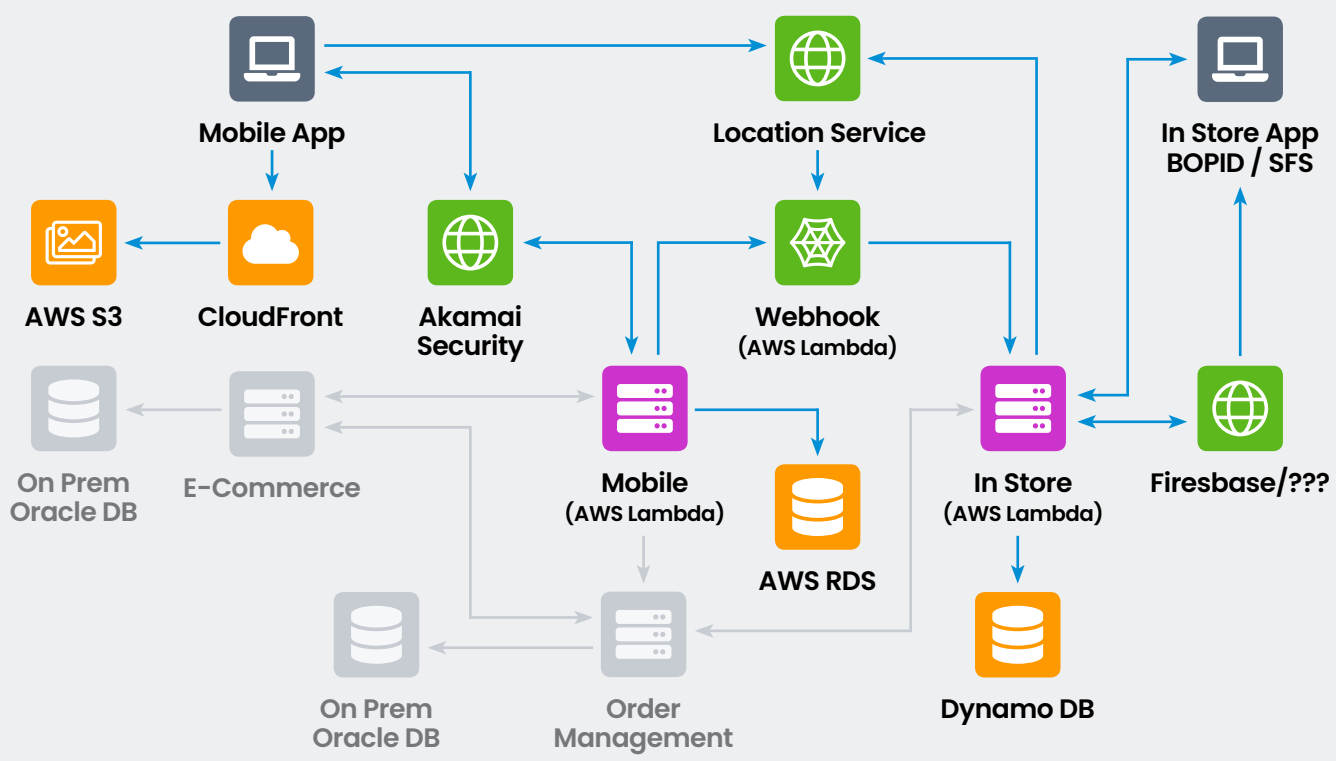
Curbside Architecture

Third Party · Microservices Wrapper · Microservice · AWS Services

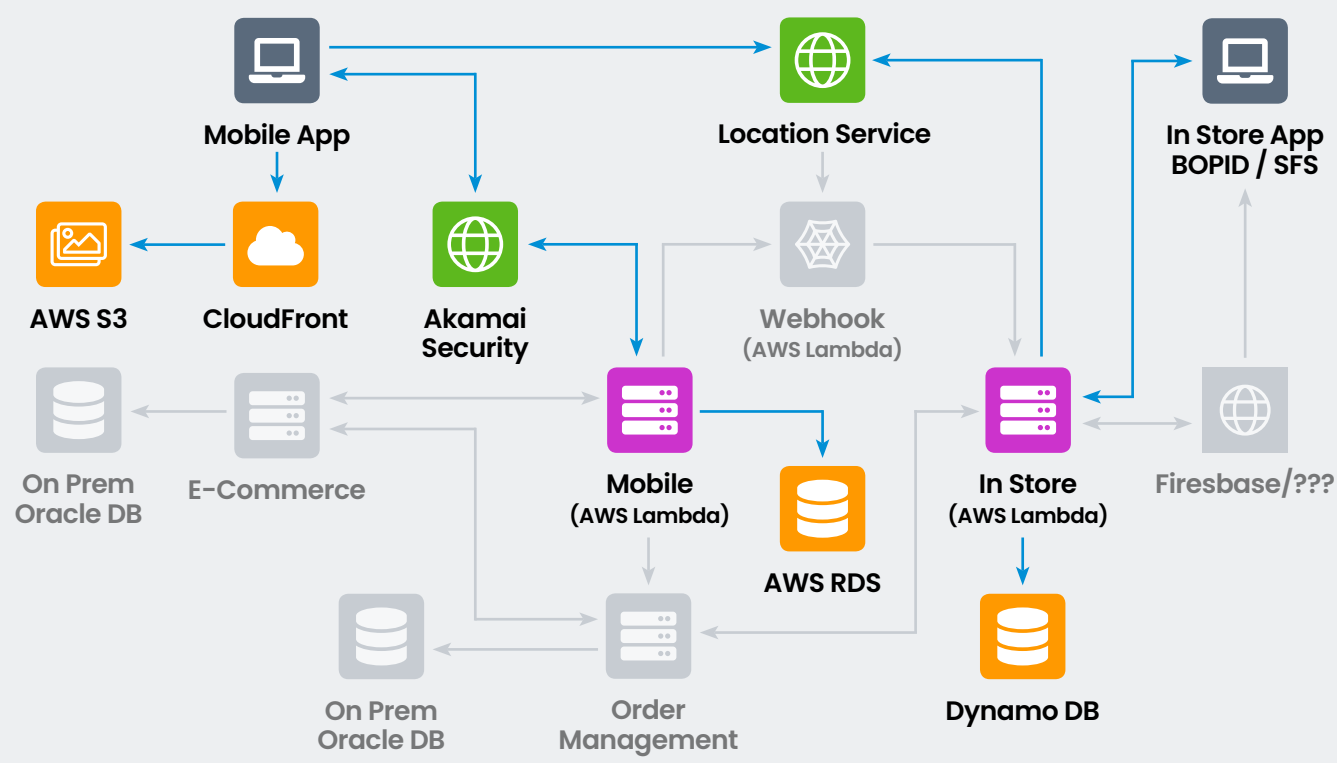
User Places Order



User Arrives



User Picks Up Order



Why AWS Lambda is Ideal for Microservice Adapters

You've decided the benefits of using microservices to "decompose" are significant enough to take the plunge. Now, what type of infrastructure are you going to have to manage and scale, on top of what you've already got in-house? If you use the cloud, the answer may be next to "none."

At JBS Solutions, we think AWS Lambda is ideal for deploying microservices, because...

It's serverless

There's no need to buy, provision or manage new servers for your microservices—just upload your code to Lambda, and it automatically runs on demand.

It scales automatically

Likewise, there's no need for extra servers hanging around when they aren't needed—AWS Lambda processes each trigger individually, running your code in parallel with as many (or as few) resources as your workload requires at any given moment.

It's cost-effective

That automatic scalability saves you money, too, because with AWS Lambda, you only pay for the compute time your microservices consume, when they consume them—and even that is metered and billed in 100ms units.

It performs consistently

And because, unlike physical infrastructure you maintain on your own, Lambda is always ready to respond within milliseconds to any request for your microservices and allows you to optimize code execution.

Plus, the flexibility, scalability, and reliability of AWS Lambda is ideal for microservices that can be orchestrated for almost any digital transformation initiative you can imagine, including...

- **Data processing-on-demand**
- **High-performance serverless backends**
- **Real-time file processing**
- **Powerful web applications**
- **Real-time video and audio stream processing**
- **Rich, personalized mobile app experiences**
- **Machine learning model data preprocessing and real-time prediction**
- **And many more**

Why wouldn't you deploy your microservices in the cloud?

A Hybrid Microservice-Monolith Approach Makes Sense for Today's Organizations

There is no doubt that the rapid adoption of microservice architecture in the last few years has caused a fundamental shift in how we build new enterprise applications. The fact is that most organizations can't start from scratch. They have dozens if not hundreds of existing applications built with monolithic architecture and limited developers to come in and simply replace those applications wholesale.

Does this mean those organizations are stuck with their monolith systems? Not at all. The bigger the monolith and the more custom business logic code you add to it, the more money, time and staff it costs to maintain it.

Do they have to fund years-long projects to convert all their monoliths to a microservice architecture, going head-on with some of the drawbacks that exist therein? No again.

The power of the hybrid approach is clear. Using microservices as a wrapper enables organizations to rapidly create and deploy new business functionality, gaining the benefits of microservices while minimizing the downside. It leaves your

Most organizations don't get to start from scratch. They have dozens if not hundreds of existing applications built with monolithic architecture. And they don't have an unlimited army of developers to come in and simply replace those applications wholesale.

Microservices as a wrapper enables software developers to build complex, engaging solutions for just about any industry: retail, financial, education, healthcare, high-growth startups—you name it.

monoliths relatively untouched while exposing their domain services as small, agile microservices that organizations can use to orchestrate business logic—rather than embedding that logic in custom code in the monoliths.

Delivering an entire curbside pickup solution in less than six weeks for a major retailer is just one example of the hybrid approach success. Our approach enables software developers to build complex, engaging solutions for just about any industry: retail, financial, education, healthcare, high-growth startups—you name it. That's because enterprises in every industry struggle with modernizing infrastructures that are based on monolithic systems.

So, why isn't everyone doing this? For one thing, it is a new pattern that not many development teams are familiar with. For another, when executed poorly, you can easily negate all the potential benefits. One of the main reasons is that many internal IT and development shops don't yet have experience with the latest technologies or the agile development needed to quickly and correctly deliver a microservice architecture. Even the most dedicated and enthusiastic team may not be quite ready and able to embrace the agility needed to deliver the innovation their organizations require.

Some of the tell-tale signs that your development process is too rigid, and you really should look at microservices are when an IT development team says it needs:

- To compromise the user experience for the convenience of existing systems' infrastructure and architecture
- Weeks of planning before starting to write any code
- Months of effort before there's anything to show for it
- Complex plans for release chains and synchronizing releases
- Piles of money for new hardware, dev, test, and production environments

Any organization can benefit from adopting a microservice architecture, not just giant companies, but it does take experience to get it right. It is hard to get the domain decomposition and size of the services correct without having some experience, so be careful jumping into microservices. What's important is that you use microservices pragmatically to fit your company, your own needs, and your own resources—or you risk losing most if not all of the benefits.

For more information on how JBS Custom Software Solutions' hybrid approach to microservices can address your modernization and digital transformation efforts, contact us today.

JBS Custom Software Solutions

EXPERIENCE MATTERS.

JBS Custom Software Solutions has been delivering leading-edge software and application-based solutions for our clients for over 20 years. It is important to us, as it is to you, to deliver a quality product on-time and on-budget. Most of the custom development that we've delivered over the years is to support proprietary business models and innovative technological solutions. This is the core of our business and our experience ranges from conception (design, R&D) to delivery (code complete, deployment). Our rich history is filled with successful proprietary applications and systems developed specifically for our partners, and range in size from high-growth startups to large, multi-billion dollar enterprises.

As no two projects are alike, we invest considerable time upfront on projects to deeply understand your business challenges and goals. In developing business software solutions for our partners, we focus on your culture, in-house team capabilities and strategic business goals, always dedicated to delivering appropriate, profit-focused solutions.

Contact us today to see how The JBS Approach—teams comprised entirely of senior level experts deployed alongside your existing resources—create a more efficient process to deliver solution that meet your unique business objectives with a higher success rates than the industry average—to produce a better, faster return on your investment.

LET’S BUILD SOMETHING TOGETHER.

Contact Us Today ›

888-421-1155

jbssolutions.com

610 Chadds Ford Drive.
Suite M23
Chadds Ford, PA 19317-7354

JBS
Custom Software Solutions