PERSPECTIVE

Unraveling Microservices: Higher Agility or Hype?

Abstract

We live in an ever-changing world with E-commerce starting to look up, and every business starting to embrace digital in their game plan. Businesses, in their need for creating a digital drapery have triggered the need to break down business functions to even smaller, manageable micro functions. Architects, from time to time, do look at innovative ways to simplify the architecture; but at times, they knowingly or unknowingly end up complicating it, trying to cater to different requirements. A long-running project will have its own set of production rollouts from time to time and hence, the architecture needs to remain receptive and nimble-footed for abrupt changes.

Even though the idea is not entirely new, and might seem like an offshoot of distributed architecture, the microservices architecture is still worthy of deliberation as it talks about simplifying the way we look at enterprise architecture. In this paper, we look at why microservices architecture might be a better fit in a complex, enterprise architecture scenario that needs more customer orientation. We will also delve into the impact of microservices on service-oriented architecture (SOA), and subsequently the use of enterprise service bus (ESB). We will also look at a few examples of microservices adoption, and how they are being implemented.

By the end of it, you will be able to decide on whether the microservice architecture style suits all projects, or if it should be restricted to a certain genre of projects while resorting to SOA for the rest.

Infosys
be more

Vendors will not miss the opportunity to 'microservices-wash' their tools and platforms, to get your attention. Some will be better suited to microservices than others. While not a panacea, I can see the potential for microservices to change the way we build, maintain, and operate applications. When delivered with discipline they help applications become more evolvable, more portable, and more adaptive, particularly as organizations look to migrate application workloads to private or public cloud platforms

- Gartner Research

Economic potential in 2015

2–3 billion more people will have access to the Internet

$5–7 trillion potential economic impact of automation of knowledge work

Microservices Architecture: What drives it?

According to the National Retail Federation (NRF), there were 133.7 million unique holiday shoppers in 2014. Total shopping – including multiple trips by the same shopper – was down during the weekends (233.3 million in 2014, down from 248.6 million in 2013). The average person spend was $380.95, down from $407.02 and comScore reported that E-commerce spending increased by 32% for Thanksgiving Day, and 26% for Black Friday. Both days surpassed the billion-dollar mark for the first time in history. Cyber Monday broke $2 billion in desktop sales. IBM reported that the number of consumers using their mobile devices increased by two-thirds from the previous year. In 2013 itself, Black Friday overtook Cyber Monday in the growth of online E-commerce transactions (74%, compared to 44%). The distinction is expected to cease in the coming years. Overall, a survey claimed that 42% of holiday shoppers' budget was spent online. Specifically, 18–32 year olds were driving the shopping traffic.

Further, the 2014 Black Friday shoppers crashed the websites of many leading retailers (like BestBuy and HP) within the first few hours of operations.

To prepare for 2015, all retailers need to answer certain questions to be competitive and make the best of the biggest shopping event of the year –

- Can their site handle the biggest shopping day of the year?
- Can they attract more shoppers to the doorsteps of their store?
- Can they shorten the time required for customers to enter, fill their cart, and complete their purchase?
- How can they roll out deals over the hours, throughout the weekend, and make the inventory available in-store, for faster delivery?

Microservices architecture tries to address this issue, prepares clients for massive increases in customer traffic, and helps intelligently evolve the commerce catalog. It also seeks to reduce the points of failure and efficiently utilizes digital marketing and mobile apps for big box retailers, as well as for leading brands to get the best out of the shopping season.

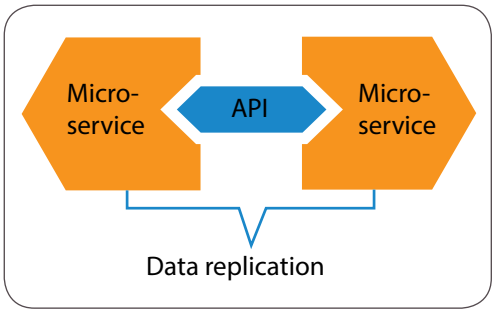
"A microservice is a loosely coupled service-oriented architecture with bounded contexts"[3]

– Adrian Cockcroft

Director of Web Engineering, and Cloud Architect, pioneer in engineering microservices.

Microservices can be considered as an architectural style where services are built, keeping smaller functions in mind. Each microservice is a miniature model of the whole architecture itself. These services can then be logically grouped, depending on business functions, and then deployed into the containers. There are multiple thoughts on how a microservice should be structured. Ideally, microservices architecture should aid multiple streams of development with different languages and platforms.

Even though the architecture definition and structure is not clearly defined, in principle, microservices architecture (MSA) points to breaking down monolithic or composite architecture into smaller manageable pieces of code, addressing business functions, and easily deployable in containers.



A customer-focused industry

The architecture world is ever-changing with new styles, and is trying to innovate at each step. The need for faster, secure data has led architects to innovate and be more forward-thinking in their approach.

With time, the industry has become more customer-focused. Service-oriented architecture has matured as an architecture style and has grown to web-oriented architecture, and subsequently customer-oriented architecture. The retail key phrase of `customer is king' can be now related to architecture because the face of every underlying business process is customer experience. A customer or a prospect can drive the architecture in today's enterprise.

Enterprises now resort to gamification to plot the customer journey or scenarios and map the experience. This can then be molded into an architecture, exposing specific functions needed as services, primarily in the REST protocol. Netflix is the best example of how the customer journey and business has spearheaded the modeling of the underlying architecture. Adrian Cockcroft was the mastermind in the company's evolution from a traditional development model, which used to produce a monolithic DVD-rental application, to a microservices architecture. Many smaller groups are now responsible for the end-to-end development of hundreds of microservices that work together to stream digital entertainment to millions of Netflix customers daily.

World is moving to digital

The new digital world looks at `IT' to be responsive to changes. As we have tried to explain in the above sections, the need of the hour is to change according to what the customer asks for. This, aided with mobiles, tablets, and wearables, brings the customer or prospect very near to the provider. A customer expecting extended hours of business and the provider ensuring that it is available via mobile / web applications insists on 24X7 availability of the underlying function, and thus becomes key to service delivery.

Quoting Forbes: "One of the timeless principles is 'know your customers,' meaning you'd best stay on your toes or you'll be watching the ever-changing whims of your customers leave you behind". So true, is it not? Look at ourselves as customers and see how we change our stance with respect to response, speed, and timeline. This is when the need of a flexible architecture, which provides accurate data kicks in. End story of all the development methods stress the need for a faster time-to-market catering to a changing customer journey.

What traditional SOA offers vs. MSA

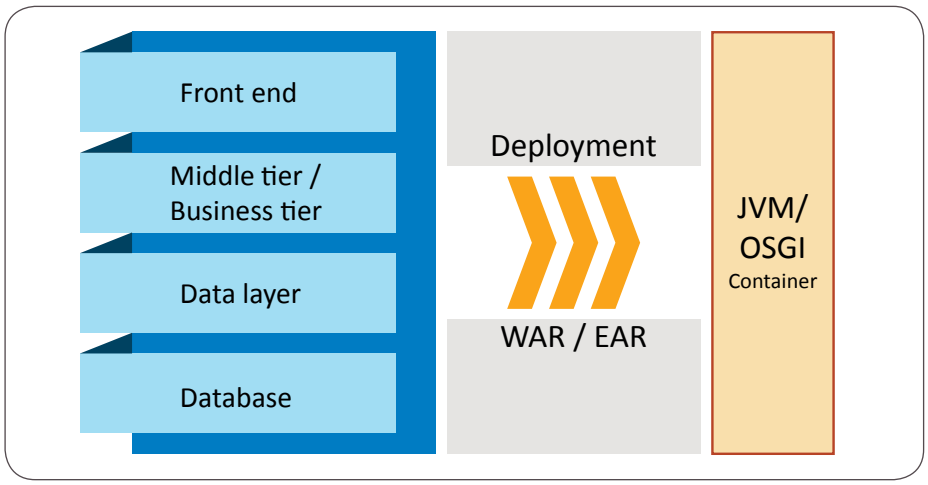
Let us now see what a typical service-oriented architecture offers. It essentially exposes the business functions in an organization via logical grouping of services. Martin Fowler sees microservices architecture as being a subset of SOA and calls it "SOA done right."[1] SOA focuses more on reuse of services. It is less flexible to change, as the emphasis is more on the service rather than the functionality itself. A single service providing common data might look to be the ideal solution in the SOA world. A multitude of services might listen to this conduit, thus aiding the reusability of the service. A change needed in the schema however, would mean bringing down all the services listening to this conduit.

Microservices, on the other hand, would group the business capability and hence the single data service will be split into multiple services, each catering to a specific need. This would mean that the services talk to each other via APIs and a change required in any one of these services would not affect other services.

Microservices Architecture: Why is it gaining prominence?

Conway's Law: "Organizations that design systems are constrained to produce designs that are copies of the communication structures of these organizations. "

One of the basic reasons behind an enterprise testing the microservices waters might be the law stated above. Organizations now tend to move away from creating designs that are tightly coupled functionally, to those that are loosely coupled in nature, providing autonomy to independent teams who manage them.



Monolithic

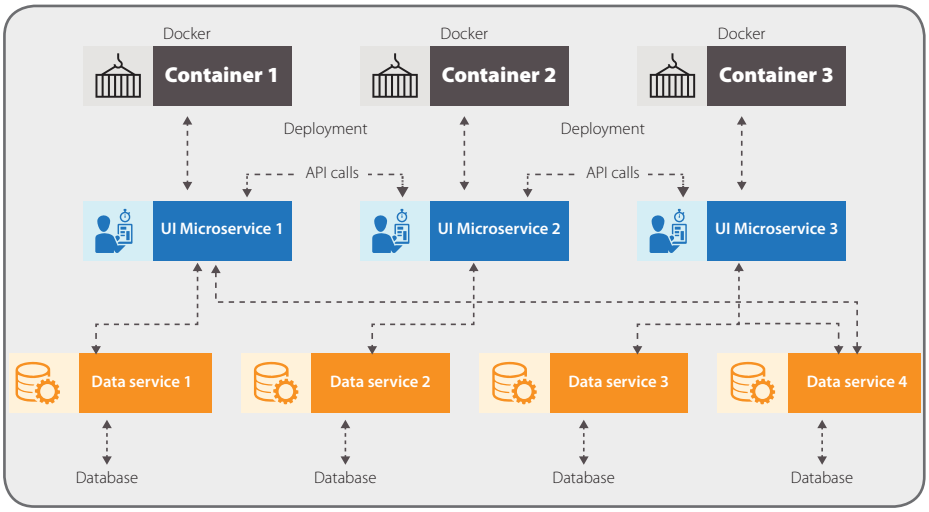
Need of manageable domain-driven services in an enterprise

The business need of embracing the digital world, led to re-thinking of the meaning of `loose coupling'. Any service that cannot be updated independently and without affecting other services is not loosely coupled. We saw in the architecture section above that, microservices emphasis is domain-driven (refer *Domain Driven Design* by Eric Evans; *In search of certainty* by Mark Burgess). If writing a service requires too much knowledge of other functions / surroundings or services then it also means that the service is not exactly domain-driven.

Need of service scalability – web scale

The square and rectangular boxes that we normally draw while designing applications generally follow the same concept of having a front end or a presentation tier, a middle tier of services that follow a logical grouping of the end functions, and at last the entities that throw data, wrapped in a different set of services. Even though this does follow a decent modular design, the deployment deposits a singular WAR or an EAR file into a single container (either JVM or OSGI). We can call this a monolithic application. A change in the flow requires the engine to be brought down and the services to be redeployed which definitely means you have a service-down window.

While these types of monoliths definitely have benefits, both in development and deployment, they work best for simple to medium architectures. Now consider a larger scope and complex scenario where you deal with a multitude of platforms, varying versions of software, and different languages. Here, the single-pipe structure that we build will be a big obstacle, and support and development teams would struggle to maintain it. It hinders services scalability if it needs to cater to varying nonfunctional requirements including huge TPS (transactions per second). An infrastructure upgrade can be cumbersome and might result in the rebuilding of the entire application.



Microservice architecture

Consider the same case if we start decomposing the architecture into multiple microservices, which can then interconnect to achieve the desired functionality. Refer to the diagram above. We will have multiple front-end services talking to multiple micro middle services and back-end services. What we achieve is an independently deployable stand-alone service. Furthermore, each service can be scaled and designated to use a level of hardware sizing needed only for that service.

Need of continuous deployment

Speeding up the deployment is as important as speeding up the development. The world has moved from a continuous virtual cloud deployment, to multiple releases and multiple deployments in containers. MSA suits this, whereas in an SOA environment, we might find difficulties in implementing hot deployment of services. A build pipeline with a complex infrastructure, a huge schema, along with third party integration will be very difficult to deploy, ending up in a multitude of other issues. On the contrary, brief services wrapping a manageable set of business features can be deployed more easily into a container, in continuous delivery. A big tick mark against this for MSA.

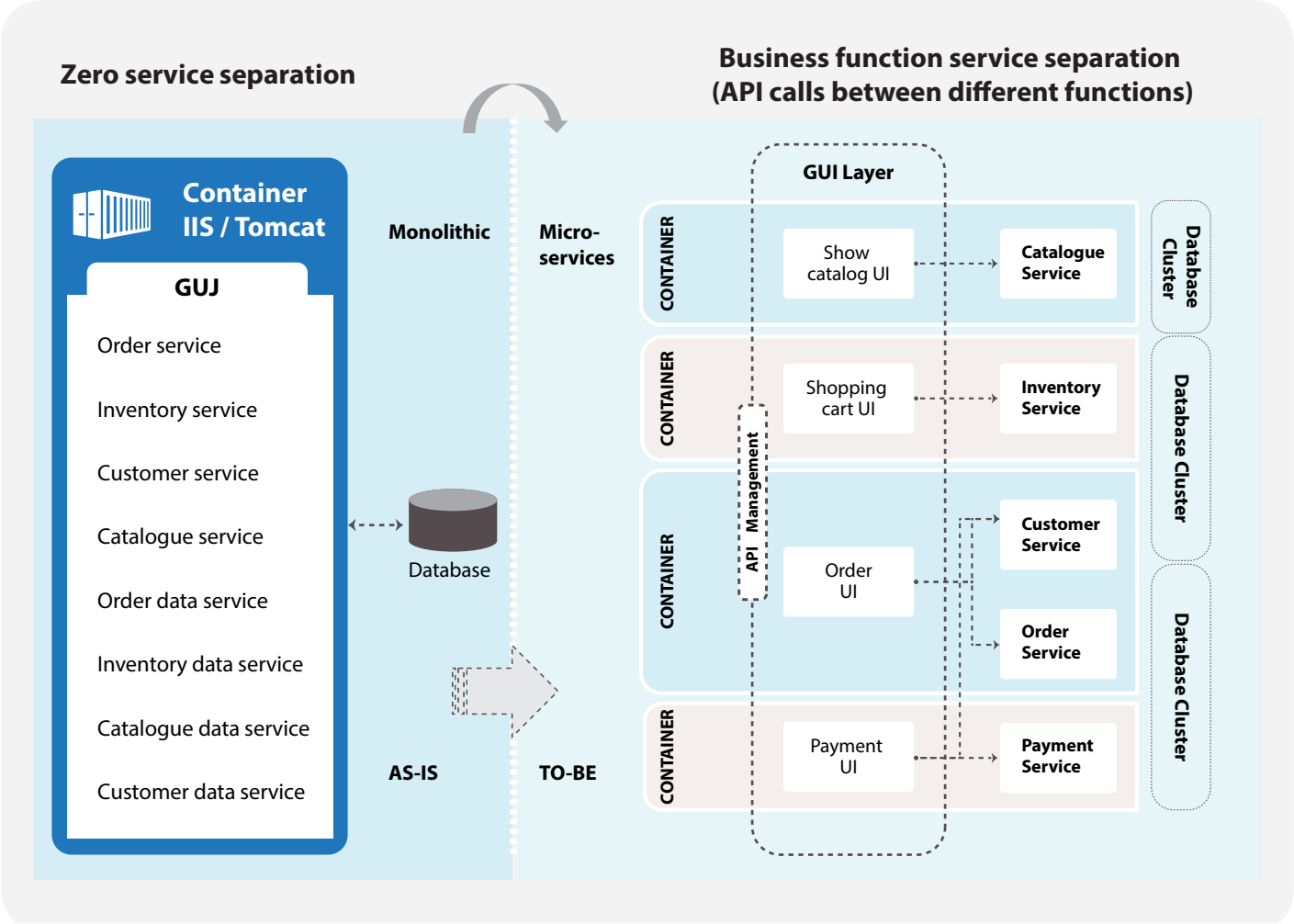
Need for high performance

We envision a resilient and high-performance architecture for enterprises. MSA with multiple smaller building units easily soaks this in. A production failure of one of the microservices will affect only that service, while the rest of the services will keep running. This means that even though deployment failures are feature-influential, they are limited only to a small area.

Microservice: As Is – To Be

The diagram below shows the eventual shift from a monolithic- to a microservices-based model. Here, a simple E-commerce web portal with multiple service calls is depicted. You can see that the erstwhile architecture had a monolithic model with services packed together in a single EAR / WAR file and placed in a container. Even though the services were defined and deployed, they were not scalable and had a severe impact on production scenarios, when one service needed to undergo changes.

Now, the same architecture is structured as individual services catering to multiple business functions and deployed separately in the transitioned diagram. A change in inventory service affects only the 'shopping cart' section of the UI, and not the other experiences like the catalog and orders section. Similarly, a user can continue seamless shopping even if the payment service is down for some time. This clearly underlines the advantages of service-split architecture based on microservices.



Microservices architecture: paradigm with multiple patterns, not one-way

Here, we discuss a few adoption patterns for MSA

- Greenfield adoption
- ESB aggregation where an ESB aggregates the business functions
- Shared data design where data is shared across multiple microservices
- Legacy modernization programs that are aimed at replacing large and complex monolithic applications running on legacy platforms, with agile and scalable solutions
- Omni-channel initiatives that are aimed at providing a seamless experience to the customer across channels, by aggregating data and services from across the enterprise

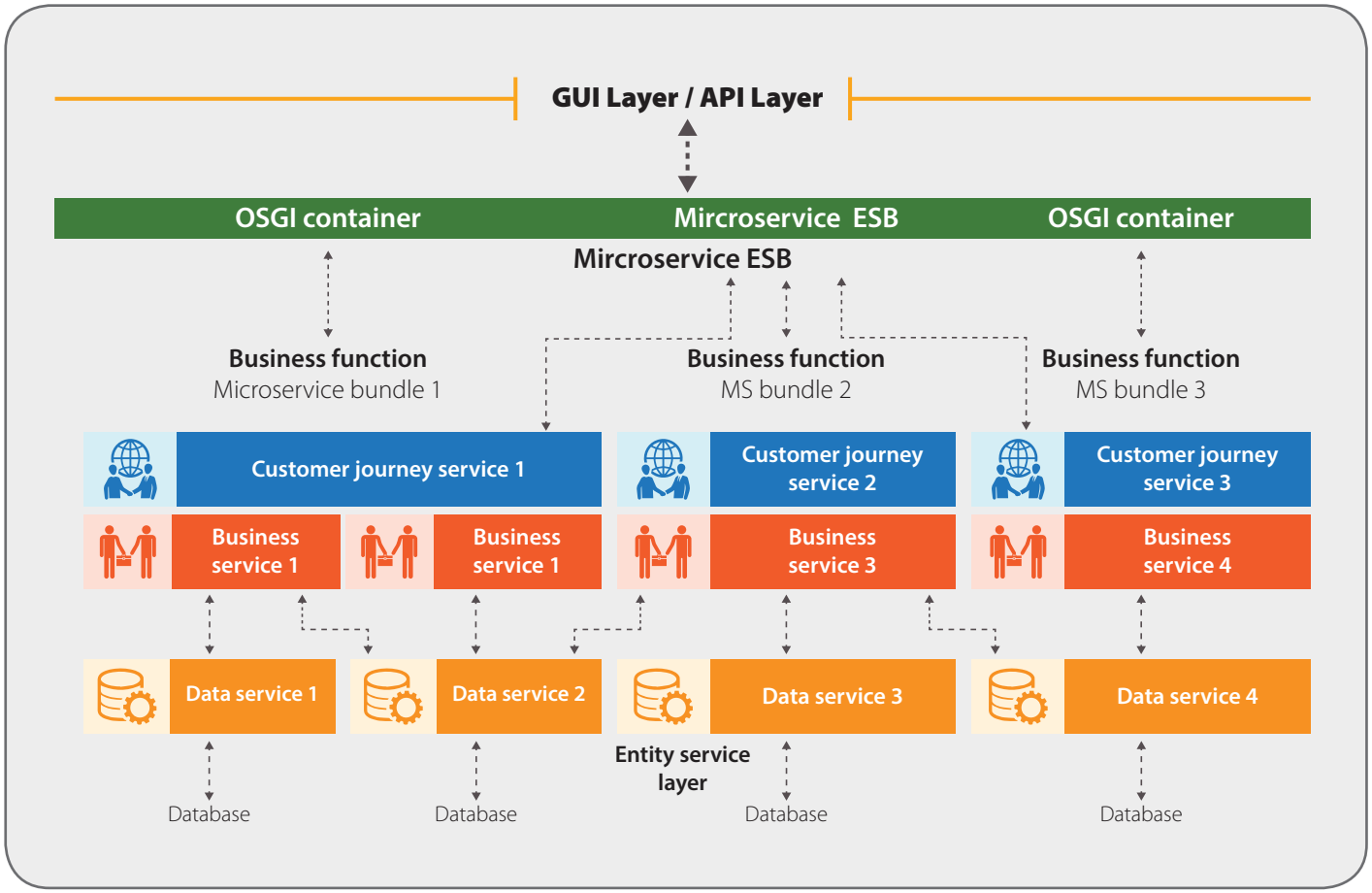
Greenfield project in MSA

This is probably the easiest of adoptions. There is no baggage of the old architecture and one can focus on the task ahead. In any case, initial assessment and design remain the same. You need to underline the boundaries and the functions that need to expose APIs. While defining microservices, it is not necessary that the service itself be very small. Some can be larger, to address a particular functionality. The diagram above for microservices holds good here.

ESB aggregation adoption pattern

Another interesting adoption pattern is to use an ESB as an aggregator. The underlying microservices are tagged together in a bus and then exposed to the outer world via REST services. This layer can be extended to include validation and enrichment based on functional needs.

The diagram shows the typical use of a bus in MSA. A customer journey, a business service or a set of them, and a data service would comprise a microservice. This is then hosted on an ESB. These services can be called by the GUI or the API management layer

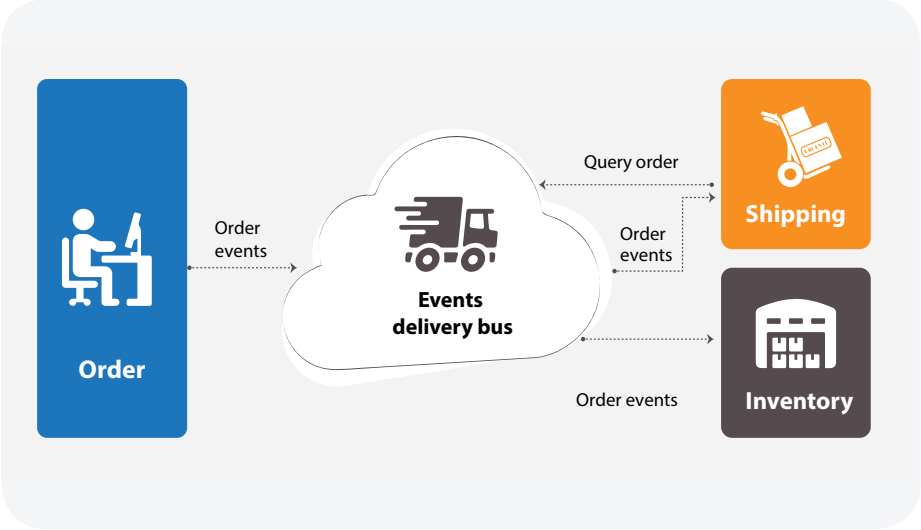


Shared data adoption pattern

If you look at the ESB diagram above, you can see that the data services are shared between multiple microservices. This can be one of the commonly used patterns across the industry. Here, there exists a dependency between the service and the underlying data services. The ultimate goal of achieving the customer functionality is carted by multiple data service calls.

Legacy modernization: ESB as an event bus

In legacy modernization programs that follow an MSA, the next generation application is going to be built as a set of collaborating services that are self-contained, in terms of the modular functionality they are expected to expose. Nevertheless, large complex monoliths have evolved because of the complex nature of the businesses they help run. This will necessitate the orchestration of microservices based on each business scenario. This brings about the need for an ESB in the role of a microservices orchestrator.



The above diagram explains how an order service generates order events, which are aggregated in an event bus and are routed to a shipping service or inventory, based on the need.

The guiding principle of microservices architecture, as discussed above, is that it should be independently scalable; and this is possible only when the service has its share of persistence. This brings about the need for data level integration between collaborating microservices. The recommended approach is to have application level event publish-subscribe mechanism between the microservices. This brings about the need for an ESB as the event bus.

Omni-channel initiatives

Enterprises are adopting the MSA for omni-channel initiatives by building the enterprise-level microservices over their existing technology solutions across channels. In an SOA-based approach, this would mean the creation of service composites that could impede scalability. In a microservices approach, the data from across channel-specific applications is fed into an enterprise-level data cache that belongs to each microservice.

For example, a retailer could build an enterprise-level product inventory cache to provide inventory visibility across the enterprise, with the inventory-check service deployed as a microservice. Here ESB is required as the message bus for fetching / routing the inventory information from the inventory silos across the enterprise, like the store inventory system and the warehouse management systems.

All of the above instances show that microservices are not tied to a unique technology. Hence, we can leverage ESB platforms to build microservices, leveraging its core capabilities of transport protocol conversion, data transformation, and database integration. A bus / gateway helps you to correlate between microservices, in-memory tracking of events, and real-time visualization. Thus, ESB platforms have evolved to support microservice development and deployment.

"Why not microservices?" "What will you miss?"

A lot of people talk about MSA and we begin to wonder whether we need to change our approach towards enterprise architecture. Not really. MSA definitely has its advantages, but it is not necessary that all projects can easily adopt it. If the adopting organization's services are not well defined and lack a proper service contract, then it is more pain than gain, going the MSA way.

The adoption patterns we discussed above can be extended to several other ones, like a proxy pattern, and a simple mediation pattern, depending on the need of the hour.

Greenfield implementations are where an MSA shines most, not forgetting that the container in which it runs, should be open to hot deployment. For example – an OSGI container like Karaf, Equinox, etc.

A heavily GUI-centric project, like an E-commerce portal, should move towards MSA keeping the customer / business-driven dynamic changes in mind.

Organizations that need flexible runtime deployments, automated monitoring, a central control center, etc., will also be an ideal candidate for MSA.

As pointed in the sections above, ESBs are getting leaner and fitter. One such case is the movement of TIBCO from Business Works 5.x to Business Works 6.x. BW 6.x is the fitter, healthier OSGI-based service bus, which can host microservices with ease. Its Equinox-based OSGI container aids in hot deployment and is developer-friendly, with a more open eclipse-based IDE. JBOSS Fuse from Red hat which uses Karaf is another example.

What Amazon and Netflix found with MSA was the ease of change of functions without disturbing the experience of the user. If you are into the E-commerce space, and need the same flexibility, then yes, it is time to change. A banker who needs a channel revamp, or a manufacturer who is at the tipping point of performance optimization can also start scratching the microservices space.

Microservice architecture: Recommendations to transform or re-engineer

Let us take a step back and see what enterprises look like now

- Historically, applications built in stand-alone
- Multiplication of these applications over time
- Evolution and need for synthesis systems
- Front-end systems implement a limited multichannel approach, resulting in a complex landscape (multiple interfaces, data integration)

Well, these are quite the same across most of the enterprises. Every organization that is trying to elevate itself to a global omni-channel, service-based, and API-driven digital experience will have to go through the pain of simplifying the current architecture landscape.

Joining the microservice bandwagon

One can pursue the microservices journey progressively through small projects or through disruptive enterprise architecture initiatives. It would seem wiser to move the smaller projects out of the way as this gives more space to study the impact and make funding available. Leveraging a structured approach to use SMAC (Social Mobile Analytics Cloud) in enterprise architecture can be another way of embracing microservices. An SOA will still be the basic need to access any touchpoint, but it will have to be more granular with the concept of microservices pitching in.

The steps of breaking down the enterprise architecture should be:

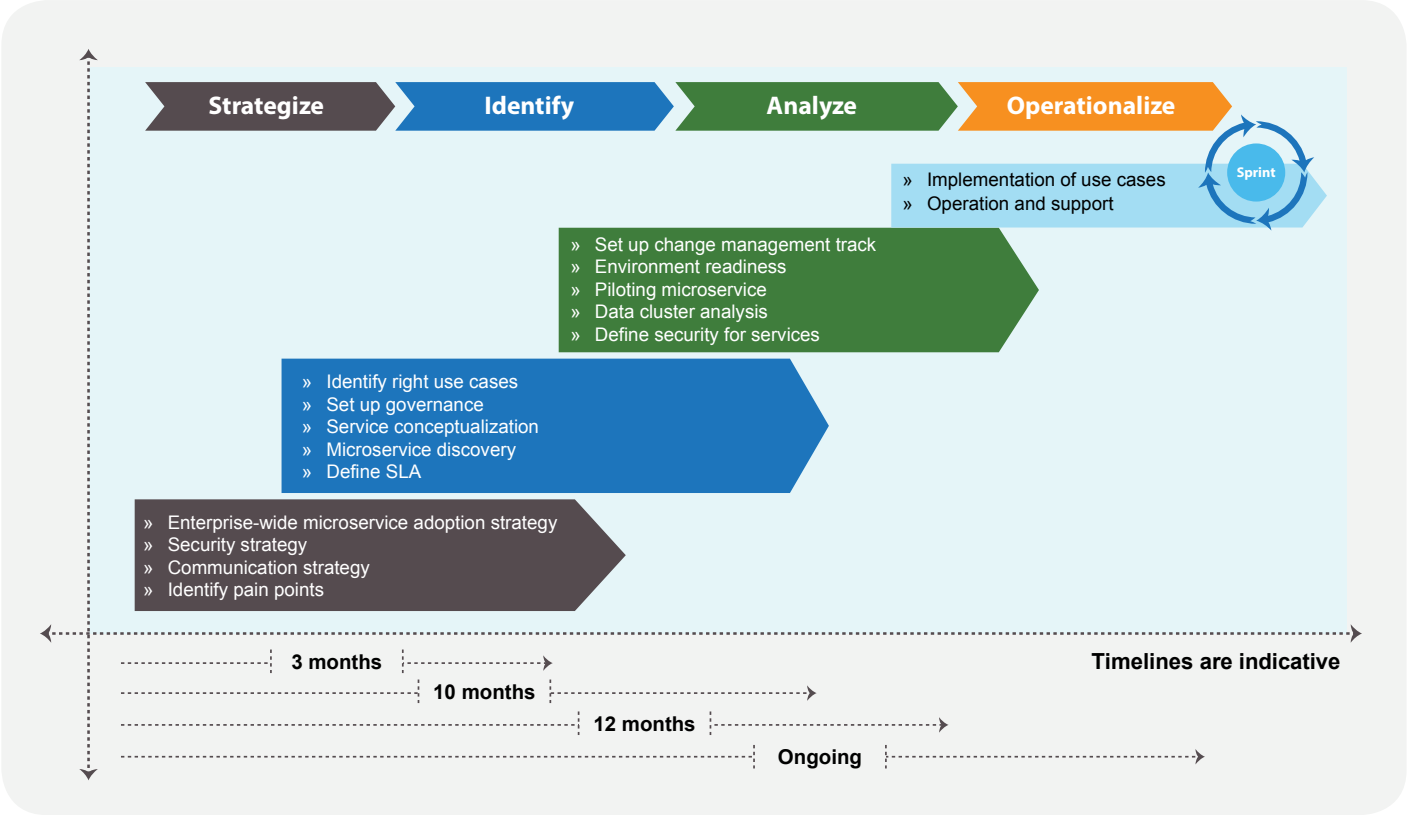
- Identify the right use case – Not every use case might fit into MSA and it is important to identify the services that define the end-users' journey.
- How micro is micro – As defined in above sections, one needs to be sure of the size of the service. It is more important to address the functionality as a whole, rather than sizing the microservice. Chopping down complex functions into multiple smaller chunks will be a good approach.
- Define the communication patterns – This is a vital step in defining MSA because it is important to interact effectively with each microservice to derive maximum benefit. There will be multiple scenarios where numerous microservices will cater to a single functionality.
- Define the service boundary – Since service forms a point of flexibility, it is important to define the boundaries of action for the services.
- Identify the tools – Applying microservices need not rely on a specific tool. Organizations have a lot to choose from when it comes to developer tools. A POJO would be as efficient as any other product in the market that claims to simplify microservices development.
- Security for MSA – A layered approach to separate critical data from non-critical data. Zonal security can be implemented with specific services being a part of the secure zone.

A transformed architecture would provide

- Data in clusters, center for all processes and systems
- Integration of different data clusters from other IT organizations (B2B)
- Simplifies and realizes multichannel approach
- New functionalities tap into data clusters
- Usage of contemporary tools and design paradigms like API and MSA

Microservices adoption roadmap

A successful microservices roadmap addresses the various tenets of MSA, like adoption pain points, security concerns, service life cycle management, and versioning. Shown below is how an adoption roadmap would be for an enterprise moving into the micro-world.



Microservice architecture: not without challenges…

We have been talking about only the happy scenarios that drive MSA. However, it is also important to look at the challenges we will face while getting into MSA. Microservices is not a small piece of code but a multitude of them. This also means that all of them have to be carefully grouped and placed in multiple servers. Each of them might be on an entirely different technology stack, thus, it is important to be aware of the following –

- Log and monitoring: We are not talking about a single software piece, handling one set of deployments, where it is easy to log and monitor. Imagine many deployments, talking to many services, and all of them need to be wired together and monitored. We need to identify software that can visualize effectively a 1000+ services and innumerable flows. This, if not done carefully, can be a nightmare in itself.
- Versioning: Service versioning and managing of hundreds of services can lead to mismatches
- Scaling: While MSA does boast of easy scaling up, maintaining these servers across multiple data centers, which may be spread across continents, will be another challenge.
- Identification of MSA failure patterns: Adrian says that this area needs to mature. It would take time to identify and communicate common microservice failure patterns[4].

Microservices: efficient and agile

Amazon is one of the best examples where one can see microservices implemented successfully. Amazon set up a highly scalable, distributed SOA after a lot of trial and error. Here, instead of each page calling a single service, services were split into many smaller functionality-based services. This enabled Amazon to rework / repair or scale any of the services without the end user being affected. Choosing the simplest tools to develop and deliver micro-level functions was a success that Amazon had. With this, they have been able to provide customers what they want, at a faster pace than any competitor does. This architecture combined with cloud computing makes the architecture economical in the long run.

References:

[1]Martin Fowler's article: http://martinfowler.com/articles/microservices.html

[2]Sam Newman. Building Microservices. ISBN 978-1-4919-5035-7

[3]Adrian Cockroft. Slideshare Microservices and Cloud: http://www.slideshare.net/adriancockcroft/qcon-new-york-speed-and-scale

[4]Adrian Cockcroft. Microservices: the good, the bad, and the ugly. Red hat webinars

About the Author

Manoj Oommen

Enterprise Digital Integration Services, Infosys.

Manoj is a technology evangelist and leads the Digital Integration - SOA Center of Excellence at Infosys. He has over 15 years of experience in architecting and Implementing projects involving SOA, API, EAI, BPM, and B2B environments, with a diverse product background. He is an avid open source learner, practitioner, and contributor, and has successfully guided technology initiatives, and driven business-critical projects across multiple verticals.

For more information, contact askus@infosys.com

Infosys be more