EMERGENCE OF MICROSERVICE ARCHITECTURE

brillio

Let's start something

www.brillio.com

TABLE OF CONTENTS

INTRODUCTION

The trends for an enterprise and/or consumer to be relevant digitally and the adoption of cloud-first & mobile-first approach has pushed product vendors and solution frameworks, to evolve from a closed and vendor only implementation business to a more vibrant and extensible framework. This enables the consumers to leverage functionality as-is and additionally extend on the core functions to address current or evolving business needs.

The rapid pace at which the technology is increasing has raised the level of customer expectations as well as the challenges faced by the business. This has forced the businesses to provide services on which the customers can build on. Let's look on some significant challenges that are faced by the organizations who are trying to stay ahead in the competition.

First of all, the technology advancement has enabled this world with lot of devices such as – smarter phones, smart watches, beacons and the list is expected to increase sooner. Everyone including the business partners, consumers and employees anticipate round the clock access to the business information, processes and notifications. To increase the customer experience and engagement, the organizations should keep the customers connected through the all the latest devices available in the market.

Secondly, to stay in the competition, the organizations have to innovate continuously and adapt to the newer trends. They need to be as flexible as possible to quicken their development process which will reduce their time-to-market significantly. With the liberal access to advanced technologies, it will not take longer for a competitor to launch similar innovation sooner in the market and gain the first-mover advantage.

The organizations have recognized that importance of enterprise architecture and its role in achieving the digital strategy. Companies which adopted Monolithic or Tightly-coupled SOA architecture are experiencing difficulties in going digital as it requires a more open architecture which can help them scale, develop and deploy swiftly.

EVOLUTION OF APPLICATION ARCHITECTURE

MONOLITHIC ARCHITECTURE

In Monolithic architecture, the applications are developed and deployed as a single entity. It is the simplest form with respect to the architectures as it does not involve many actors. The application would mostly have single database to which it communicates for storage and retrieval of the data. Even today, most small or mid-sized architectures run using this approach as the complexity of this architecture is low.

But when organizations try to add any feature as a part of innovation or try to integrate with external devices or third party application, they encountered serious problems. Since, the modules are dependent on each other, refactoring the code becomes a painful process. It involves going through or changing the whole of monolith and the change would impact everyfunction of the architecture.

ADVANTAGES	DISADVANTAGES
Simplest way of the architectures	Hard to Manage, test, debug
Suits projects with small and easy scope	Doesn't suit applications that evolve
Easy to develop and deploy	Hard to learn the application
Easy to scale size-wise by generating multiple instances of application	Scales only in one dimension - cannot scale feature-wise

MULTI-TIER ARCHITECTURE

The growing size of monolithic architecture demanded breaking of applications into logical tiers. By doing this the developers have the choice of working on the specific layer instead of the whole application, where the modification is required. The 3-tier architecture with layers – presentation layer, business logic layer and Data storage layer was most the prevalent. All three layers were maintained as independent module on isolated platforms.
Again in an evolving application, the business layer was becoming a monolith itself, which posed the same problems as monolith.

ADVANTAGES	DISADVANTAGES
Making changes on one layer without affecting the other layers.	Hard to Manage, test, debug
Secure business logic as there is no direct access to DB	Doesn't suit applications that evolve
Migration to a new presentation layer is easier	Hard to learn the application
Each layer is independent and accessible to independent set of developers	Scales only in one dimension - cannot scale feature-wise

SERVICE-ORIENTED ARCHITECTURE (SOA)

To overcome the complexity challenges monoliths, the developers were forced to think of a way to break the application under some logic. So they started visualizing the application as collection of business capabilities which formed as a basis for Service-Oriented Architecture.

In SOAs, the components were divided based on the functions and the services were created to support their functionalities. These services communicate via a common interface Enterprise Service Bus. Few organizations implemented this rightly and thus SOA helped those organizations overcome the challenges posed by monolithic architecture. Additionally, SOAs improved the reusability of code

SOA eventually became a buzz word and few misperceived implementation of SOA will suit any development project.This is overkill and actually increased the complexity. Some developers forcibly separated the functionalities where monolithic architecture could have done the job easily. In cases, where the scope is limited for SOAs, there is no reason for introducing the complexity at architectural level.

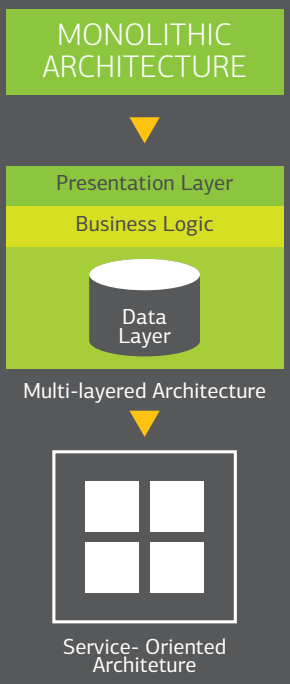
Besides, few vendors used this opportunity to sell the middleware platform, which added up to the already existing complexities and resulted in SOAP and ESBs. Though SOA was good architecture and solved major problems, there were no bounded concepts on how to use it which itself was the drawback of this architecture.

ADVANTAGES

- Loosely coupled than monoliths
- Easy to manage, test & debug
- Reusabilityof services
- Flexible and Easy to scale

DISADVANTAGES

- Migrating to SOA when there is no need to poses problems
- Change to SOA is big process and can't be undone
- High configuration cost as the application evolved
- SOA not implemented properly will be semi -monoliths



MICROSERVICES ARCHITECTURE (MSA)

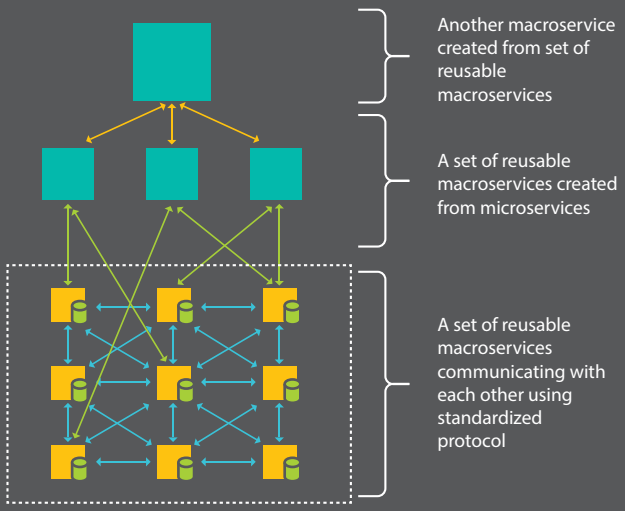
Microservice architecture is the latest method to built application by breaking the application into compostable services. The services created in this architecture are kept “smaller” i.e. one service focusses on one function only. Each service is managed by one small team and that team takes the complete ownership of the particular service. Similarly, there are lots of small services and thus, the development, testing and deployment of all these services can be carried out independently.

The evolution of microservices is considered as a reaction to the “vendor-driven” SOA.It is based on SOA with the necessary modifications required to make it effective. Adrian Cockcroft, who was responsible for MSA implementation in Netflix, defines it as *“loosely couple SOAs with bounded contexts”.*

MACROSERVICES

The combination of one or more microservices components to form a logical feature is called as macroservices. These macroservices can be exposed as APIs for integrating with internal or external applications. They serve as a feature which can be reused in multiple applications.

The following image shows the microservices architecture in action:



NEED OF MICROSERVICE ARCHITECTURE

The changing business needs has to be dynamically addressed by the developers. A more loosely-coupled architecture approach would help them to build and deploy applications with celerity.

As Gartner stated in Top 10 Strategic Technology Trends for 2016,

"Monolithic, linear application designs (e.g., the three-tier architecture) are giving way to a more loosely coupled integrative approach: the apps and services architecture. Enabled by software-defined application services, this new approach enables Web-scale performance, flexibility and agility."

Gartner also acknowledges the rapid growth of Microservice architecture by stating that,

"Microservice architecture is an emerging pattern for building distributed applications that support agile delivery and scalable deployment, both on-premises and in the cloud. Containers are emerging as a critical technology for enabling agile development and microservice architectures. Bringing mobile and IoT elements into the app and service architecture creates a comprehensive model to address back-end cloud scalability and front-end device mesh experiences. Application teams must create new modern architectures to deliver agile, flexible and dynamic cloud-based applications with agile, flexible and dynamic user experiences that span the digital mesh."

This can be validated with the real life examples of PayPal, Amazon, Netflix and GILT.com having already shifted from monolithic to microservice architecture. This shift helped these organizations make "focused" service components which are small, independent and can be managed by a small two-pizza team, as Amazon calls it.

BEST PRACTICES FOR DESIGNING MICROSERVICES

Architecting, deploying and maintaining microservices, if not managed correctly, can result in over-consumption of human as well as infrastructure resources. The following best practices would keep the systems efficient and scalable while the microservices based solutions are implemented.

- Cache highly-used microservices to avoid the network traffic congestion
- Adhere to DevOps cultural practices. Development resources need to own the changes all the way to production and support them
- Pass the data between microservices through http headers to minimize the traffic congestion
- Implement message queueing services wherever possible
- A-synchronize everything possible to improve the performance
- Implement access restriction protocols to reduce the availability of microservices to avoid unauthorized access
- Keep separate data store for each microservices
- Do separate build for each microservices
- Replace silos with microservice teams

ADVANTAGES OF MICROSERVICES

The benefits of adopting microservices

Fault Isolation: The larger applications can remain mostly unaffected by the failure of one or two services

Technology Independent: Removes the constraint on long-term commitment to one technology stack. If you want to try out a new technology stack on any individual service, you can go ahead. This essentially means that the dependency concerns are lesser than they are in monolithic services and rolling back any blunder done is also easier

Easy to Learn: Makes it easier for a new developer or an architect to understand the functionality of the application

Easy to Manage: A small team takes the complete ownership for the microservice component. A smaller project is apparently easier to manage than big projects.

DISADVANTAGES OF MICROSERVICES

Having explained and said so much about microservice doesn't mean that we have found a perfect architecture. It may be a buzz word in industry but no technology is a silver bullet. It is good to know the drawbacks of a technology before adopting it

Latency Problems: Developing microservices means development of distributed systems. There may be multiple requests travelling between the modules which possibly might induce latency

Developer Mind-Set: Developers tend to concentrate on decomposing the services and write fine grained services. The goal is to decompose the application into compostable microservices

Inter-Process Communication: Managing the complexity of Inter Process Communication mechanism could be challenging. Implementing retry logic, addressing latency issues and partial failures due to the unavailability of services would be critical.

Shared DB Aarchitecture: Segregation of data by consumer and securing the services is another critical need

Service Rollout: Ensuring consistency of upgrades, rollout of service updates has to be orchestrated through a well-defined automated process.

CONCLUSION

Building a complex API platform is inherently difficult. A monolithic or a tightly coupled SOA architecture makes sense only in the case of simple, lightweight platform.

The implementation of microservices based API platform seems to solve the problems that arises out of monolithic or tightly coupled SOA

- With no technology dependency using microservices, newer services can be created in lesser time to support any latest devices launched in the market
- There are no more hurdles for innovations. The time-taken to develop and deploy a minor feature is less than a day
- Integration is not painful anymore. The organization can now collaborate with customers more easily to make the organization's capabilities reach bigger set of audience

Microservices based SDK can be a strategy for the organizations to align IT assets with business capabilities, business resources and business processes. Of course as Fred Brooks said 30 years ago, there are no silver bullets. But if you are looking for a distributed system to simplify your API platform and develop SDK that can help developers creating applications and integrating them quickly, then microservices based SDK should fit you.