CONVERGENCE OF MICRO SERVICES AND API

brillio

Let's start
something

www.brillio.com

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.

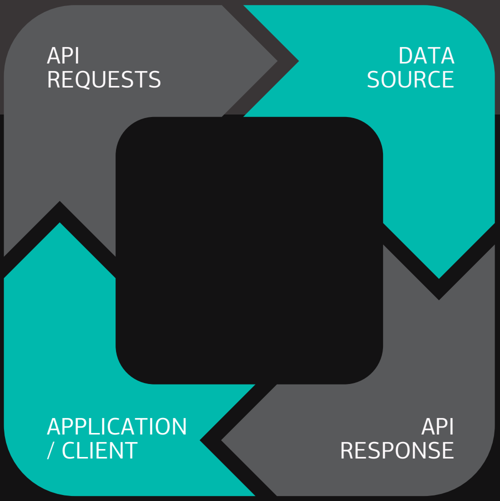
Secondly, to stay in the competition, the organizations have to innovate constantly and adapt to the newer trends. They need to be as flexible as possible to quicken their development process which will reduce their time-to-market significantly. With the liberal access to advanced technologies, it will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

Finally, staying on legacy ways may not be of much help for the organizations to scale up in terms of both technology and size. This means that the organizations have to look for newer ways of revenue generation. In such competitive environment it is harder, though not impossible, for most of the organizations to grow in silos. This mandates the need for collaborating with other organizations or third parties to “co-create” their futures.

APPLICATION PROGRAMMING INTERFACE

The APIs are interface that provides developers the access to an organization's proprietary software. It's a software intermediary that makes it possible for the application programs to interact with each other and share data. API development and deployment requires extensive domain knowledge in operating an enterprise-class infrastructure. APIs can be treated as the core product for digital business. They define the business capabilities, data and processes of an organization. APIs expose an organization's valuable services to the customers and business partners.

Figure 1 Working Of Api



GROWTH OF APIS

APIs have been in use for years as a basic tool for effective development. They have aided in increasing the reusability of features for the developers and in effective customization for the organizations. It was in early 2000's when APIs moved to cloud which allowed software to interact with any service and platform through network. APIs are growing exponentially per year.

The following graph shows the growth of public registered APIs every year:

PUBLIC APLSREGISTERD

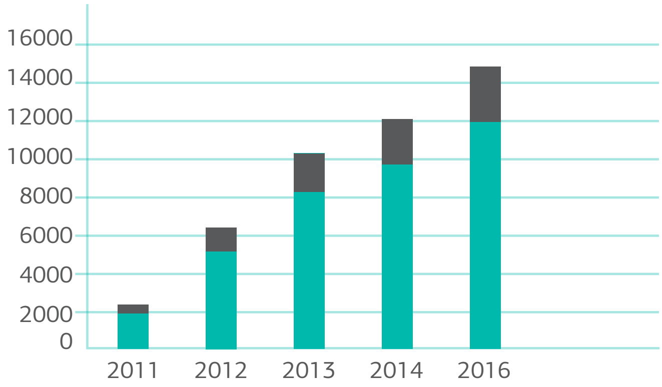


Figure 2 Source: Smartbear.com [Infographics]

TYPES OF API

The two major types of APIs that are in market are: REST and SOAP

- Representational State Transfer (REST) APIs involve simple way of sending and receiving data between the client and server. It doesn't have many standards defined. It is a light-weight API and the data can be sent through it in JSON or XML or even plain text. This is the most used API type.
- Simple Object Access Protocol (SOAP) is a method of transferring messages or small information over the internet. SOAP messages are formatted in XML and sent through HTTP protocol.

The other types of APIs are JavaScript and XML-RPC (remote Procedure Call)

ARCHITECTURES OF API PLATFORM

API PLATFORM USING SOA ARCHITECTURE

The SOA-API convergence was once regarded as the best practice approach for the solution and platform development. The proponents of SOA pitched about its loft benefits both in business and technical perspectives. Though the benefits are real, getting those benefits is the tougher task.

SOA is based on the concept of services. Many major organizations moved from monolithic architecture to SOA for reducing the complexity in applications development and scaling. In SOA, each service may perform one or more activities or may be a combination of one or more service operations depending upon the implementation. This essentially means that, each service is built as a separate piece of code which makes it easier for the developers to reuse the code.

LIMITATIONS OF SOA

Some organizations which migrated to SOA from monolithic architecture started to realise that they were in a mess once again. This is evident from the real life examples of the companies like GILT.com, PayPal.

GILT.com migrated from monoliths to a more loosely coupled architecture that was similar to SOA. They created services that are aligned to its business lines. But again they realised that each service created was itself is equal to monoliths. Any new features/services they added went into the existing services created. Adding new services was also a challenge.

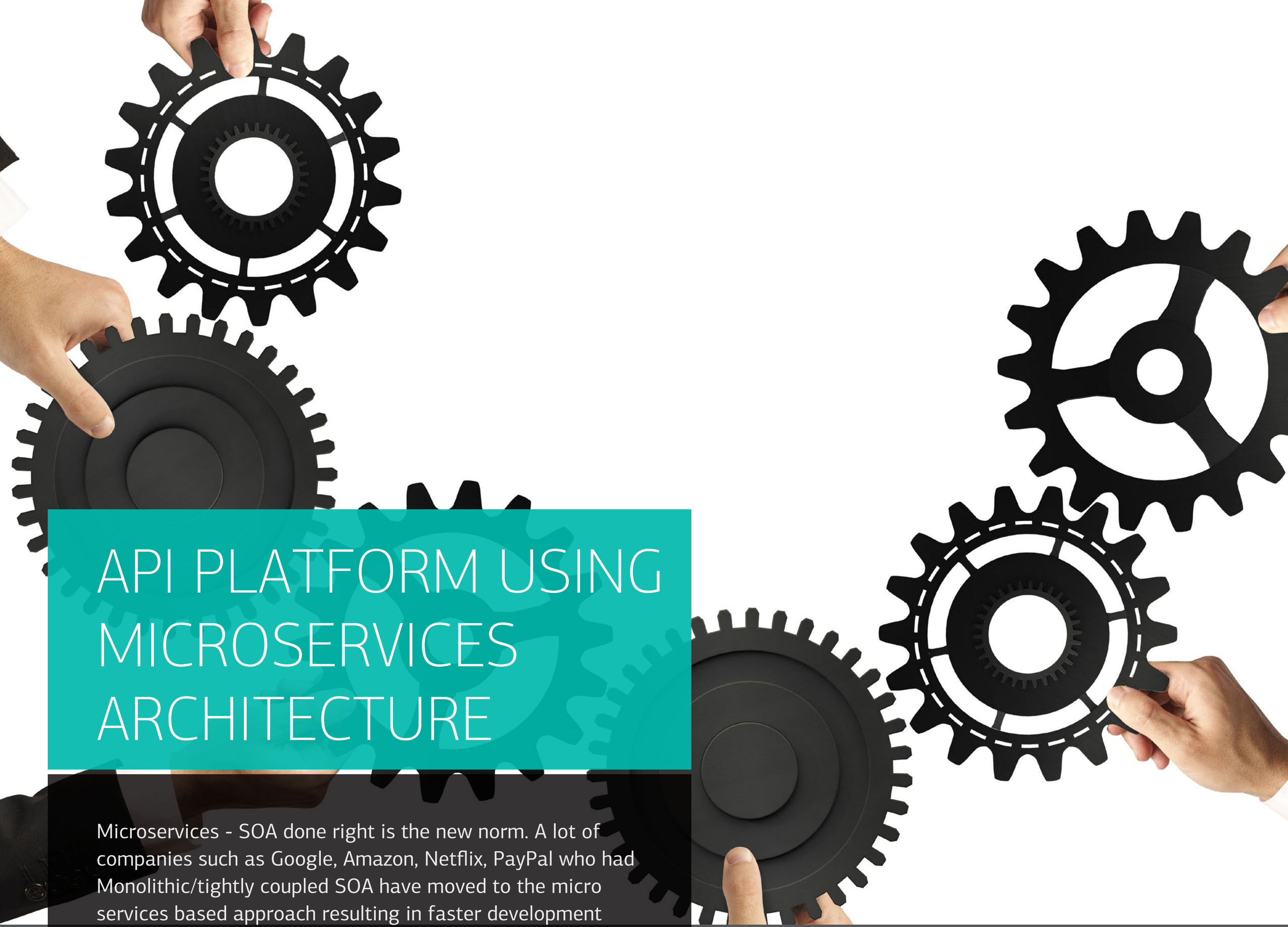
PayPal too, encountered a similar problem. They had a tightly coupled SOA architecture with project specific APIs which made their integration process painful and expensive. Additionally, the increasing number of APIs has made the API discovery process painful.

The common problem observed from the similar cases are listed out below

There is no clear definition of how small or big the service should be. This resulted in design flaws and each service created was itself equal to monoliths. This defied the whole purpose of adapting SOA over monolithic architecture.

While scaling up or making changes, the change in one service affected other dependant services and again all the teams have to sit together to solve the classic “integration” problem.

Besides all these, the flexibility quotient was missing. The dependency on technology was a bottleneck for all these organizations. For example in case of GILT, they used ruby on rails in monoliths, JAVA in the SOAs and had problems in scaling up. They were stuck to a technology stack and could not leverage the advantages of other available technologies.



API PLATFORM USING MICROSERVICES ARCHITECTURE

Microservices - SOA done right is the new norm. A lot of companies such as Google, Amazon, Netflix, PayPal who had Monolithic/tightly coupled SOA have moved to the micro services based approach resulting in faster development time, faster iteration and less downtime - this has added up to the revenue of these companies.

Key attributes any modern API platform should address is depicted in the diagram below,

PLATFORM QUALITIES	CURRENT SCENARIO	TARGET SCENARIO
API Access Scope	Internal / External	Universal
API Design	Project Specific	Product
Architecture	Monolithic / tightly coupled SOA	Microservices
Technology	Single Technology Stack	Flexible & Standardized
API Intergration	Expensive & Time Consuming	Takes Less than X minutes only

Figure 3: API Platform Qualities

WHY MICROSERVICES?

With the changing need of business demands and the change in the way developers and architects build the application, a more loosely coupled architecture approach would help them to quickly build and deploy applications. In the Top 10 Strategic Technology Trends for 2016, Gartner states

"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."

This means that changing business needs have to be dynamically addressed by the developers.

Gartner also acknowledges the rapid growth of Microservices architecture in the digital world by stating,

“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 prediction can be validated using real life examples of PayPal, Amazon, Netflix and GILT.com having already shifted from monolithic/SOA to microservice architecture.Microservices architecturehelped these organizations make “focused” service components which were then combinedto form public APIS that reflectedthese organization’s actual business capabilities.

In the case of PayPal, the shift to microservices did miracles for their business. It resulted in a start of another S-curve for them. Before adapting microservices architecture, they made project specific APIs using tightly coupled architecture. They faced lot of complaints regarding their APIs and SDKs. Now, after shifting to microservices the APIs are more aligned to their capabilities. They were able to bring more number of customers on-board after this shift

Besides these advantages, when PayPal wanted to enter into new technology such as payment through beacons, they did that with ease. They used the appropriate technology stack for separate services and had separate teams working on it. This did not disturb the setup of other services and other teams.

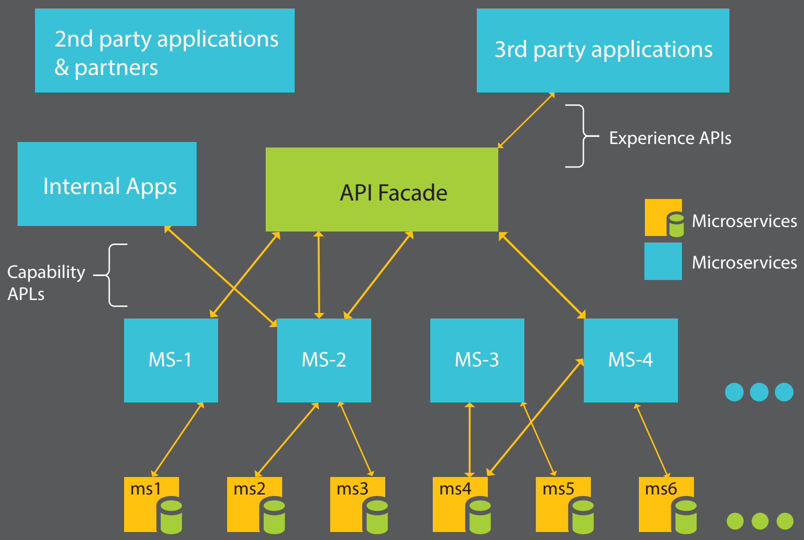
As a result of all these changes, their integration process has become easier and costs less. They also made the API discovery process easier through their developer platform.

ARCHITECTURAL CHANGE

Classic SOA was focused around creating organization wide architecture to make sure that the people in organization could reuse the resources. Microservices, though seen as another form of SOA, induces agility into the process.

Microservices architecture enables the architects and developers to break the application logically into small services that are narrowly focused. The services communicate with each other through networks using inter-process communication mechanism. Now the application can be updated or scaled up service-wise and this significantly minimizes the development and deployment time for the developers

The following figure shows the skeleton of microservice architecture:



2nd party applications & partners

3rd party applications

Experience APIs

Internal Apps

API Facade

🟨 Microservices

🟦 Microservices

Capability APLs

MS-1

MS-2

MS-3

MS-4

ms1

ms2

ms3

ms4

ms5

ms6

Figure 3 Microservices and API Setup

- Each microservice component has its own database and a set of teams that owns the component
- The microservice component contains few lines of code with the focus on only one service. This removes the ambiguity regarding the service size that was faced with SOA
- A set of microservice components forms Macroservices. The Macroservices forms the capability APIs which can be used by the internal application.
- The Capability APIs pass through API façade, the goal of which is to articulate internal systems and make them useful for the developers. Façade provides a virtual layer between the interface on the top and implementation on the bottom.
- The exposed APIs can be used by the external developers for integrating with their application with the organization capabilities

CULTURAL CHANGE

Conway's law states that the software interfaces structure of a system will reflect the social boundaries of that organization. This basically translates into the context that the organization that wants to make a shift to microservice architecture should organize the staffs in a DevOps setup rather than managing in silos.

The Microservices architecture emphasizes on one team owning one service. Each microservice will be treated as a product where and hence will have its own manager, developer, tester, DB specialist and other required resources. These resources are responsible for the focused service provided by the microservice component resulting in seamless process of development, testing, debugging and integration

SDK DEVELOPMENT

Apparently, organizations are shifting to mobile-first strategy to improve customer experience and increase customer engagements with their brand. This can be supported with a survey result from Nielsen which shows that 89% of consumers prefer mobile applications over mobile web. Having a well-built SDK is crucial if an organization's strategy includes making its capability available across the platforms and devices.

A good SDK improves developer experience using which developers can build great mobile applications. The SDKs can be created to support various platforms and programming languages.

Besides, providing SDK has wide range of functions such as:

- API management
- Security Management
- Starter sample code
- Performance monitoring
- User Data collection for Analytics

The following picture represents the ownership of teams in microservices architecture

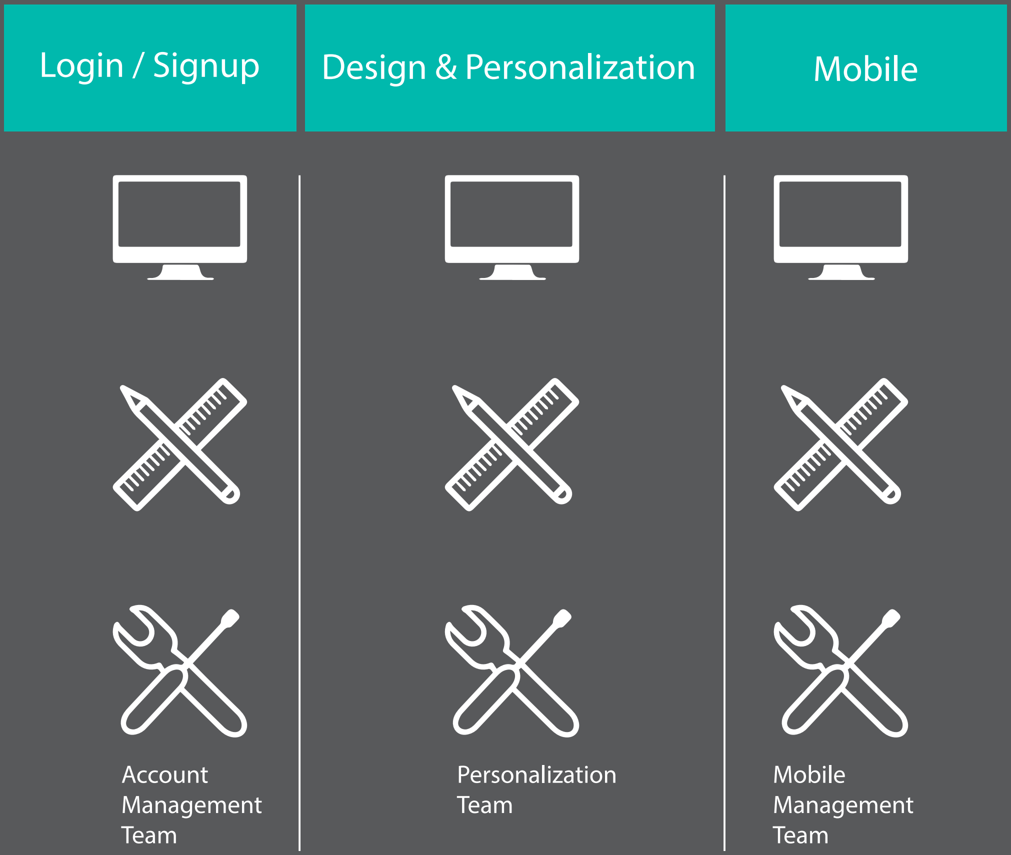


Figure 4: Microservies Ownership

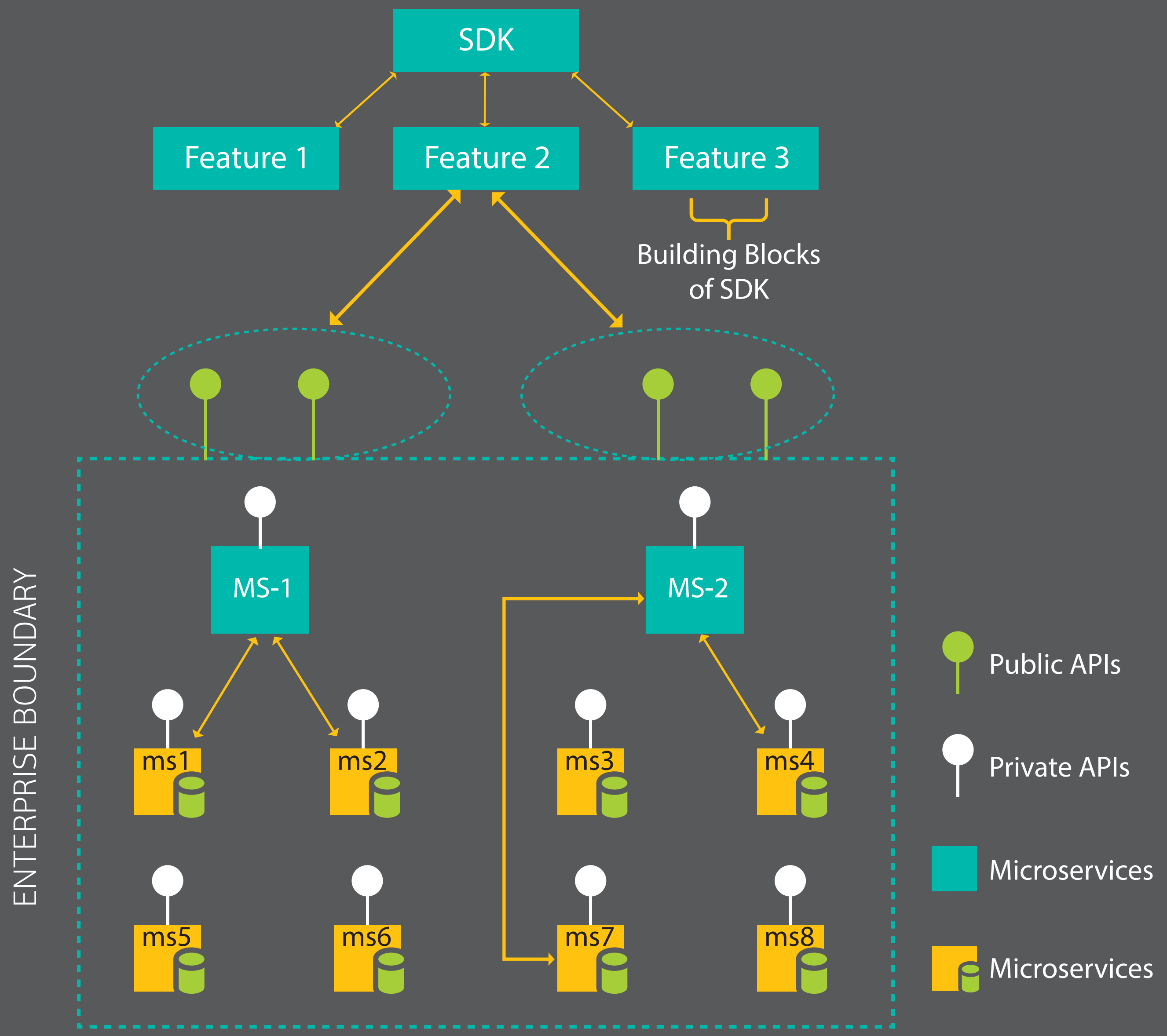


Figure 5: SDK Development Setup

By creating SDKs the brand can have control over which SDKs can be accessed by whom.
In the above picture,

- Each microservice components expose it's capability through an API.
- The microservices components can combine to form a Macroservice. This macroservice component can also expose its capabilities through APIs.
- The private APIs will be used by the internal developers for development of internal applications or web services
- The public APIs will be exposed to the external developers. The Public APIs are formed by various combinations of private APIs, thus masking the private APIs from the external visibility
- The public APIs shall be combined to form different features of a platform. They serve as the building block of SDKs.
- Now, these features are readily available for the developers, which saves significant amount of development time in recreating them.

ADVANTAGES OF SDKS

The following are the advantages of building an SDK:

Simplify Integration:

Time is a crucial factor for the integration process. It is important to do the integration process quickly and correctly to take the application quicker to the market. Also the integration is carried out by resources with high skill sets. So it makes sense to take as less time as possible for this important step. Having an SDK simplifies the process integration of APIs that requires complex use cases

Abstraction and Security:

SDKs can be used to encrypt the underlying UI or APIs to secure the quality of data

Metrics:

SDKs can be helpful in collecting the API usage pattern of the user and do the required modification or change the stage of the API lifecycle as required. For example, if no users are using a particular API, it is better to retire the API safely rather than retaining it.

Ensuring Best Practices:

Providing an SDK helps the developer by reducing the bad code, thereby, avoiding delays in the entire business. Inefficient code can corrupt the services of the whole organization. SDK helps avoiding such bad practices. Also SDK helps ensuring that the right rules and practices are followed throughout the publisher network.

Thus, providing SDK makes a developer's life simpler and ensures that right practices are followed, keeping critical systems secure.

CHALLENGES IN ADOPTING MICRO SERVICES BASED SDK

Developer mind-set

Developers tend to concentre 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 architecture

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.

Programming Languages

Ensuring support for a large developer ecosystem through a variety of programming languages could be another challenge.

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 arise 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 a 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.