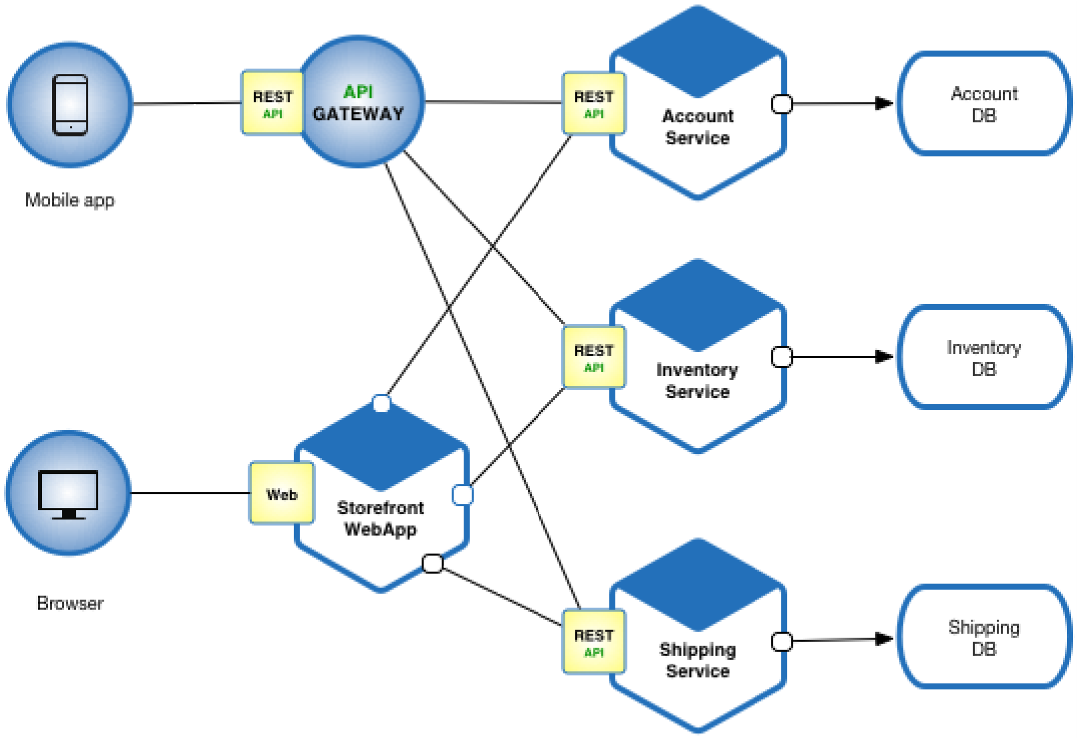
Microservice Architecture (Examples and Diagram)



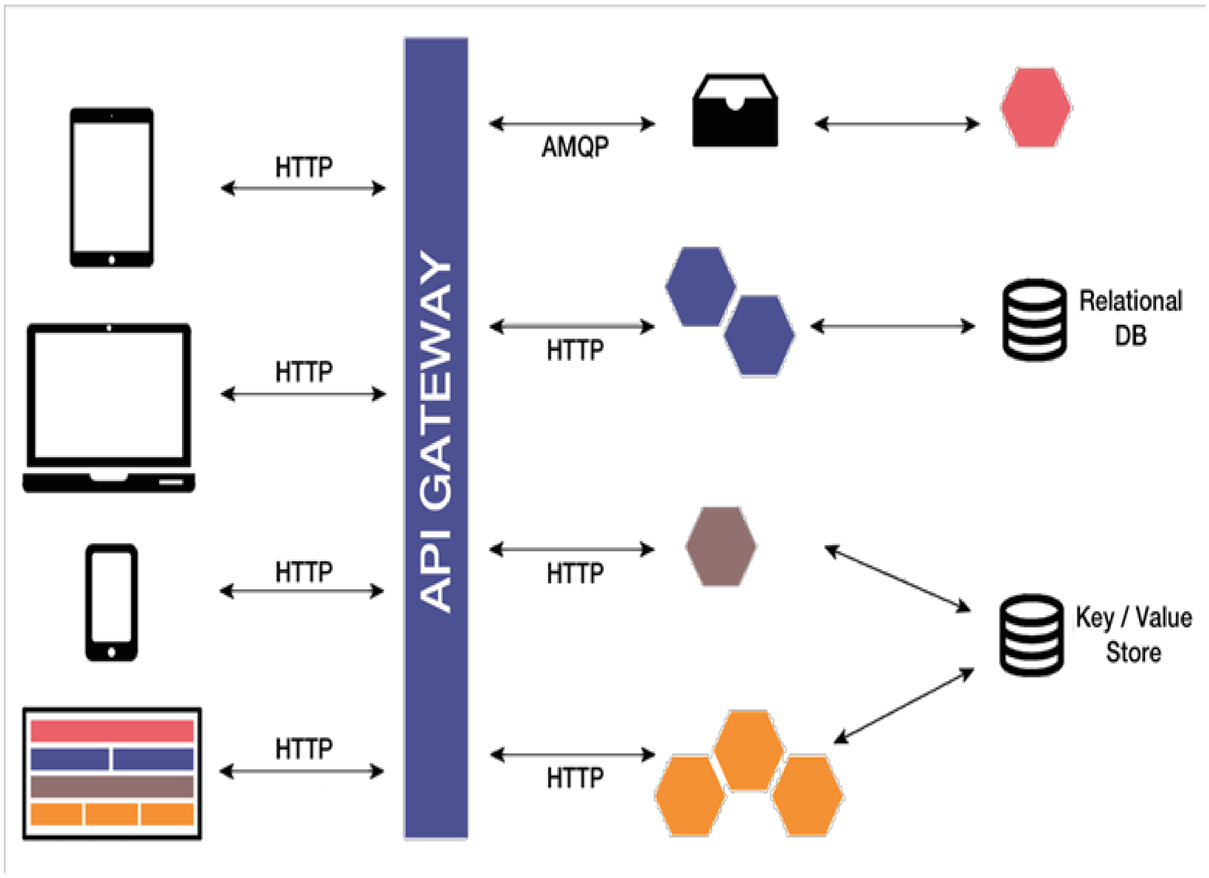
Introduction
Application architecture is an important point to consider when designing your application, over years, many different software development paradigms have come and gone, a constant theme, however, has been the need to componentize software systems in a modular way. Modularising software is beneficial for many reasons. It makes the source code easier to understand and follow, developers can visualize the codebase and know roughly where to look for defects – there by making maintenance more efficient.

Well-architected software systems are also easier to scale should the number of users using your application increase, not to mention, if new developers join your team, a well-architected system can make the onboarding process quicker and reduce developer rationing time as everyone is coding to a specific set of standards.

By the time you've read this blog post you'll have a solid understanding of what microservice architecture is, its benefits, if you should consider it and how to go about structuring an application to adopt this architectural paradigm, so let's get started!

What is Microservice Architecture?
In the past decades, applications have been developed in a monolith fashion, coded from top to bottom as one single unit, sometimes with no real structure or thought for future maintenance which can cause a range of problems. Poorly architected software solutions can also be problematic to debug or extend with new features and in general, aren't very nice to work on.

Microservice architecture is a form of service-oriented architecture (SOA) whereby software applications are built as a collection of loosely coupled services, as opposed to one monolithic software application. Each microservice can be created independently from the other, or even in a completely different programming language and run on their own.

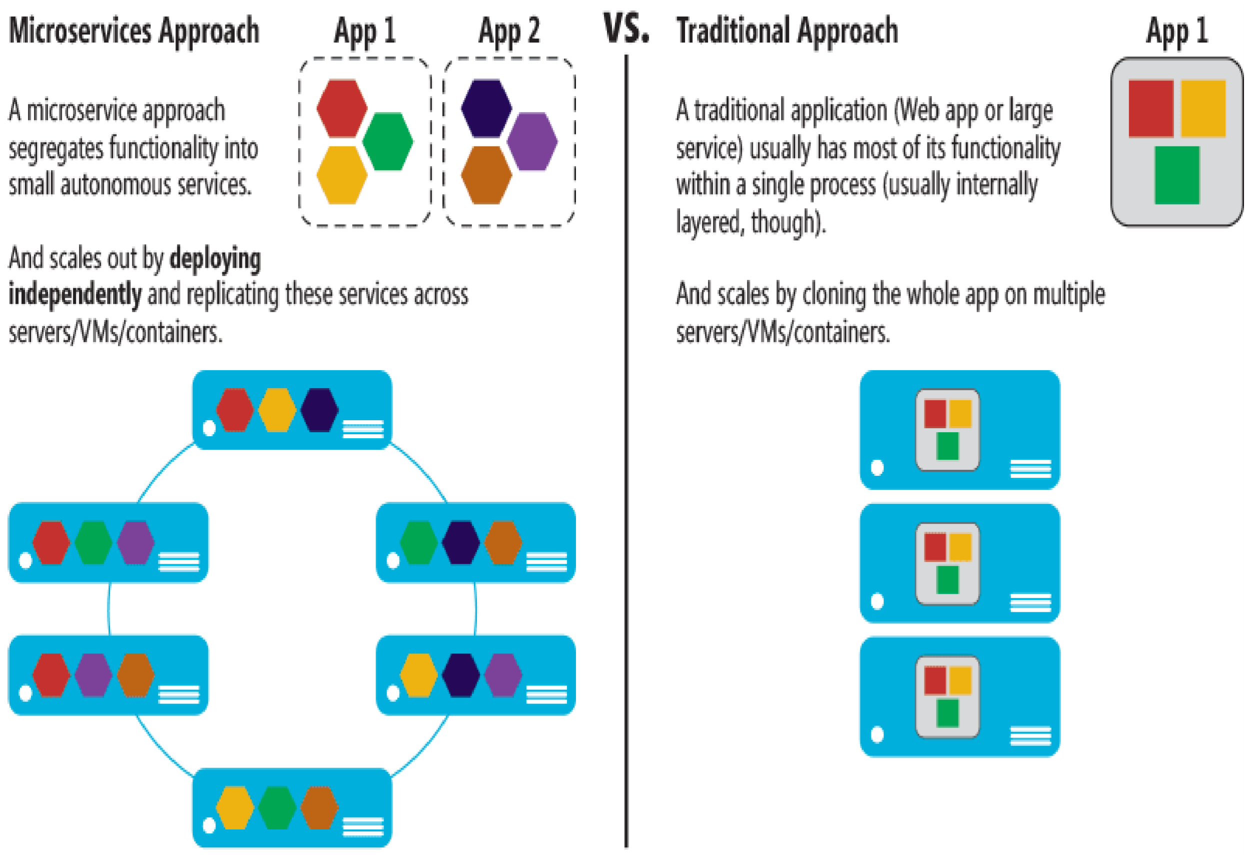


At the core of it all, each microservice tries to satisfy the specific user story or business requirement that you're working on. It is an ideal architectural design pattern in today's ever increasing interconnected world and helps support multiple platforms and devices that can span the cloud, mobile, Internet of Things (IoT) and even wearable devices.

A popular way to implement microservices is to use protocols such as HTTP/REST alongside JSON, as an architectural design pattern, we're seeing a lot of the main SaaS providers adopt microservices into their solutions and services.

From Microsoft to IBM and much more, software vendors and service providers are implementing microservice architecture. Major players like Twitter, Netflix, eBay, and Amazon have also favored microservice architecture as it helps with the development, scalability and continuous delivery of their services. This brings us to some of the benefits of microservices architecture.

Benefits of Microservice Architecture

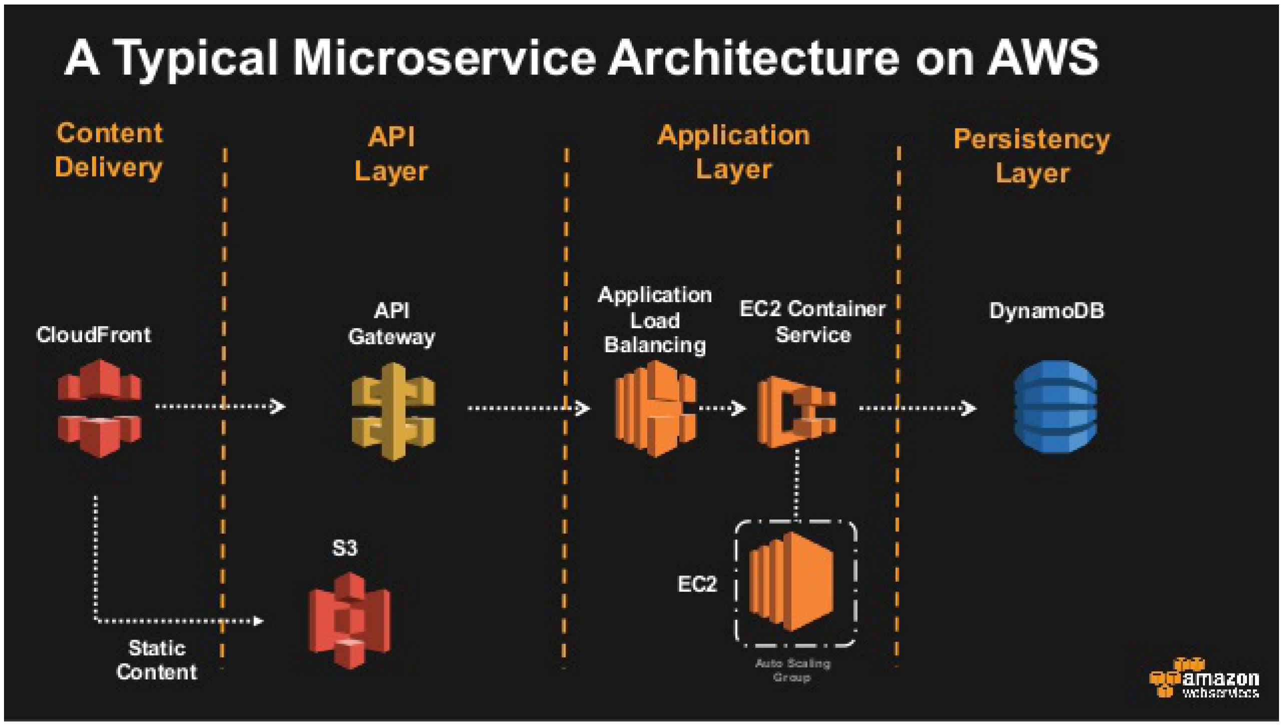

Microservices Approach
App 1
App 2
VS.
Traditional Approach
App 1
A microservice approach segregates functionality into small autonomous services.
And scales out by **deploying independently** and replicating these services across servers/VMs/containers.
A traditional application (Web app or large service) usually has most of its functionality within a single process (usually internally layered, though).
And scales by cloning the whole app on multiple servers/VMs/containers.

As an architectural practice, microservice architecture has been growing in popularity in recent years, this is down the benefits that it can bring software development teams and the enterprise. As software increased in complexity, being able to componentize functional areas in the application into sets of independent (micro) services can yield many benefits, which include, but are not limited to the following:

1. **More efficient debugging** – no more jumping through multiple layers of an application, in essence, better fault isolation
2. **Accelerated software delivery** – multiple programming languages can be used thereby giving you access to a wider developer talent pool
3. **Easier to understand the codebase** – increased productivity as each service represents a single functional area or business use case
4. **Scalability** – componentized microservices naturally lend themselves to being integrated with other applications or services via industry standard interfaces such as REST.
5. **Fault tolerance** – reduced downtime due to more resilient services
6. **Reusability** – as microservice are organized around business cases and not a particular project, due to their implementation, they can be reused and easily slotted into other projects or services, thereby reducing costs.
7. **Deployment** – as everything is encapsulated into separate microservices, you only need to deploy the services that you've changed and not the entire application. A key tenet of microservice development is ensuring that each service is loosely coupled with existing services.

But don't just take our word for it, here are some real-world examples of microservices in action how this architectural design pattern as benefited Walmart and Amazon.

Microservice Architecture in Action in the Real Word



Here we explore how adopting microservice as a software architecture could add real business value and bring a whole range of benefits for Amazon and Walmart.

Microservices rescue Walmart's aging software architecture
The Canadian arm of retail giant Walmart has serious issues with their existing software architecture, especially around Black Friday – for two consecutive years. At its peak, the Walmart site couldn't handle 6 million page views per minute and ultimately made it practically impossible for visitors to have any positive sort of user experience.

Part of the problem was that Walmart's software architecture was design for the internet of 2005 which was centered around desktops and laptops. The use of mobile, smart and IoT devices hadn't fully peaked back then. The firm decided to re-platform its legacy system in 2012 using microservices and have set a target that by 2020 to be able to service 4 billion connections!

By migrating to a microservices architecture, Walmart identified that:

Mobile orders increased by 98% –
Zero downtime on Black Friday and Boxing Day (Canadas Black Friday)
Conversions increased by 20%

Amazon
Amazon the retail and now logistics giant is no stranger to delivering software at scale. Rob Birgham, senior AWS Product Manager shared a story of how microservice architecture was used in conjunction with DevOps.

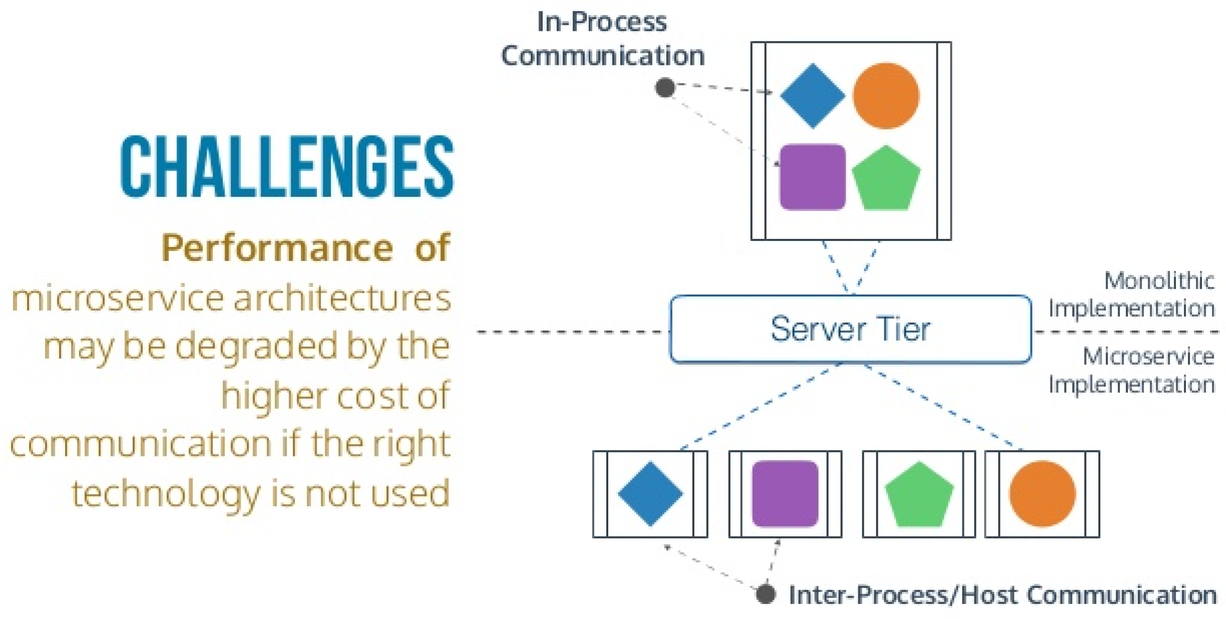
Back in the early 2000's Amazons site was developed as one, big, monoliths solution and to be fair, a lot of business started out that way. Over time, however, as functional areas were built, from multiple teams of developers or bugs were fixed, not to mention, as the platform grew, the job of integrating all this work started to impact on overall productivity.

Days or sometimes whole weeks were dedicated to merging developer changes into a master version of the product, merge conflicts had to be resolved which again, all had an impact on productivity.

Realising quickly that operating like this wasn't sustainable, Amazon took the decision to decouple their monolith codebase by adopting a microservice architecture. Each service was responsible for a single purpose as was accessible through web service APIs.

Once this exercise was complete, this paved the way for Amazon to create a highly decouple architecture where each service could operate independently – providing each developer adhered to the respective web service interface definitions.

Challenges of Microservice Architecture


CHALLENGES
Performance of microservice architectures may be degraded by the higher cost of communication if the right technology is not used
In-Process Communication
Server Tier
Monolithic Implementation
Microservice Implementation
Inter-Process/Host Communication

As with every new software programming architecture, each has the list of pros and cons, it's not always peaches and cream and microservices aren't an exception to this rule and it's worth pointing some of these out.

- **Too many coding languages** – yes, we listed this as a benefit, but it can also be a double-edged sword. Too many alternative languages, in the end, could make your solution unwieldy and potentially difficult to maintain.
- **Integration** – you need to make a conscious effort to ensure your services as are loosely couple as they possibly can be, otherwise, if you don't, you'll make a change to one service which has a ripple effect with additional services thereby making service integration difficult and time-consuming.

- **Integration test** – testing one monolithic system can be simpler as everything is in "one solution", whereas a solution based on microservices architecture may have components that live on other systems and/or environments thereby making it harder to configure and "end to end" test environment.
- **Communication** – microservices naturally need to interact with other services, each service will depend on a specific set of inputs and return specific outputs, these communication channel's need to be defined into specific interfaces and shared with your team. Failures between microservices can occur when interface definitions haven't been adhered to which can result in lost time.

So now that we've talked about what microservice architecture is, what some of its benefits are and look at a few examples of microservice architecture in the real-world as well as the benefits of this paradigm, you might be wondering if microservices architecture is for you or your software project and it's a valid question to ask.

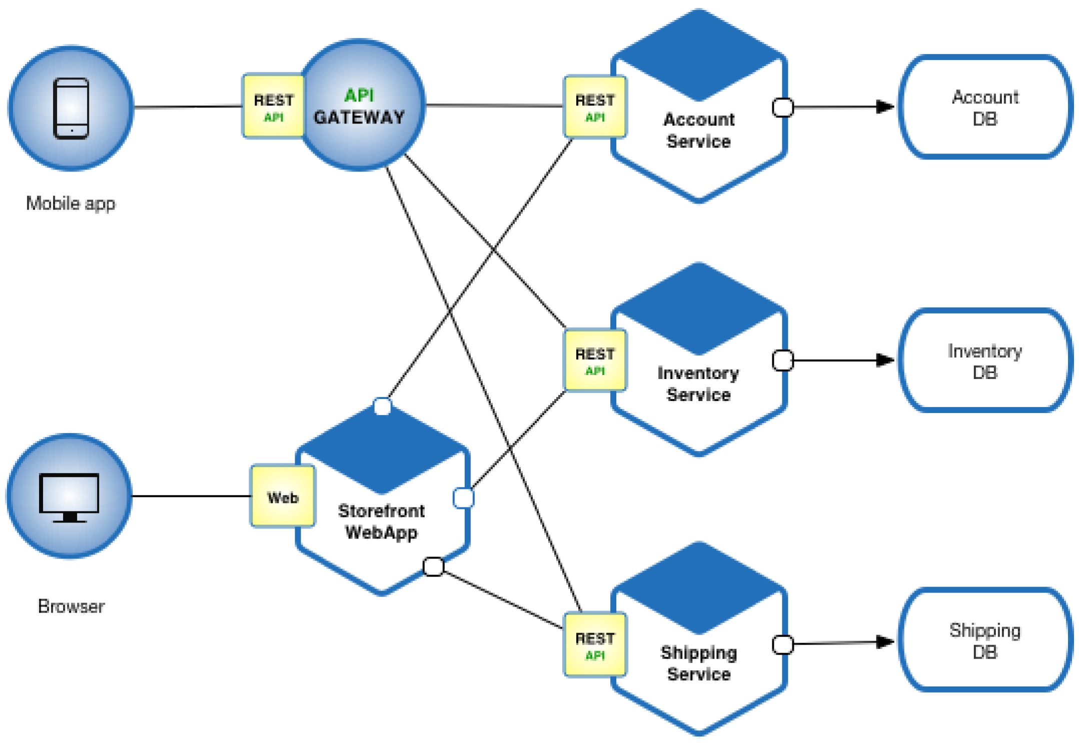
Should You Adopt Microservice Architecture?

You don't want to implement a microservice architecture in your project just for the sake of it, so we've put together some points to consider that will help you decide if that's architectural pattern is right for you or your project.

If you answer yes to any of these questions, then you might want to consider implementing a microservice bases architecture:
- Is your current application monolith and difficult to maintain?
- Do you anticipate that your application will have to deal with high volumes of traffic?

- Is modularisation and code reusability the main priority?
- Does your application need to be accessed on multiple device types such mobile, IoT and web?
- Do specific areas your application need to be able to scale -on demand?
- Are you looking to improve your software products build and release process?

If you answer yes to most of these, however, you can probably get away with a traditional monolithic application architecture:
- Do just need to ship and MVP, to test the market?
- Do already have a stable product/team that will continue to work on the product until you retire it?
- Is your product "in the wild", generating revenue and the user community is happy? If so, no sense in reinventing the wheel



Now that we've introduced microservice architecture, discussed some of the benefits, respective challenges and looked a few examples of how microservices have been deployed in the real world, it's time to look at a basic microservice in terms of its architecture and how it can be designed. In the image below, you can the following 3 microservices:
- Account Service
- Inventory Service
- Shipping Service

Each microservice is accessed in one of two ways in this fictitious application:
- From an API gateway (via a mobile app)
- From a Web application (via the user's web browser)

Note: There could be theoretically may be more as the architecture lends itself to that.

You can see that each microservice also exposes a dedicated REST API, this is the interface which defines the operations that can be executed against the respective microservices and will detail the data structures, data types, or POCOSs that can be passed into the microservice as well as its return types. For example, the Inventory service REST API definition may contain an endpoint called GetAllProducts which allows consumers of the microservice to fetch all products in the e-commerce store, this could return a list of Product objects is JSON, XML or even C# POCO.

As each microservice has its own set of responsibilities and can only ever interact with its respective database, a solution architected like this makes it easier for more than one developer to make changes to the system. By implementing a microservice like the one we've just detailed, you will gain some of the benefits that we outlined earlier. As you roll out an architecture using this approach, it's important to ensure that each microservice can operate completely independently, otherwise, it defeats the purpose of pursuing this architectural approach.

Summary
So, there you have it, an overview Microservices architecture. In this blog post, we've looked at microservices architecture, we covered the key concepts, the benefits this architectural design pattern can bring to the enterprise and how it can help software development professionals better structure their products.

We've also outlined the main components of an application that adopts a microservice architectural pattern and give you some ideas as to why you might want to introduce microservices to your project. Feel free to comment or share this article with your friends or colleagues, or if you have a comment then leave one below!