The Dividends of Microservices

The main aim of this article is to explain the major ideas and principles of microservices. This paper clearly indicates that the microservices architectural style is an important idea and a one worth consideration for enterprise applications. Some of the big giants of the industry who are in some way pioneering this architectural style include Amazon, the UK Government Digital Service, Netflix, Oracle, Uber, Red Hat Openshift, The Guardian, SoundCloud and Bluemix. In addition there are plenty of other organizations that have long been doing what can be stated as microservices. The positive results from these organizations reflect the useful and beneficial prospects of microservice architecture.

Microservices Defined

Microservice applications are composed of small, independently versioned, and scalable customer-focused services that communicate with each other over standard protocols with well-defined interfaces. It is an approach to application development in which a large application is built as a suite of modular services. Each module supports a specific business goal and uses a simple, well-defined interface to communicate with other modules.

Properties of Microservices are:

- Services are organized around capabilities, e.g., user interface front-end, recommendation, logistics, billing, etc.
- Microservices are developed by a small engineering team and can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best.
- They consist of code and state, both of which are independently versioned, deployed, and scaled.
- Interaction with other microservices is done over well-defined interfaces and protocols.
- Services remain consistent and available in the presence of failures.

On similar basis a microservices-based architecture can be well defined as:

- Symmetrical rather than hierarchical.
- It lends itself to a continuous delivery software development process. A change to a small part of the application only requires one or a small number of services to be rebuilt and redeployed.
- It is distinct from a service-oriented architecture (SOA) in a way that SOA aims at integrating various (business) applications whereas several microservices belong to one application only.

The philosophy of the microservices architecture can be summoned as the Unix philosophy of "Do one thing and do it well".

- The services are small - fine-grained to perform a single function.
- The organization culture embraces automation of deployment and testing. This eases the burden on management and operations.
- The culture and design principles embraces failure and faults, similar to anti-fragile systems.
- Each service is elastic, resilient, composable, minimal, and complete.

Dividends of Microservices

In order to explain the dividends of microservice styled architecture it's useful to compare it to the monolithic style. A monolithic application is built as a single unit and Enterprise Applications are often built in three main parts: a client-side user interface, a database and a server-side application. During the client-server era, the focus was intended on building tiered applications by using specific technologies in each tier. The interfaces tended to be between the tiers, and a more tightly coupled design used between components within each tier.

There are benefits to such a monolithic design approach. It's often simpler to design and has faster calls between components, since these calls are often over IPC. Also, everyone tests a single product, which tends to be more people-resource efficient. However, the downside is that a tight coupling between tiered layers results that it cannot be scaled at individual components. If any fixes or upgrades are to be performed, a developer has to wait for others to finish their testing. Thus, it is more difficult to be agile.

Monolithic applications can be successful, but increasingly there are feelings of downsides associated with them. Change cycles are tied together, thus a change made to even a small part of the application, requires the entire monolith to be rebuilt and deployed. Over time it's often hard to keep a good modular structure, making it harder to keep changes that will affect only one module within that module. Scaling of a module requires scaling of the entire application.

The solution to these downsides is use of microservice architectural style: building applications as suites of services. Microservices address these downsides and more closely align with the preceding business requirements. The benefits of microservices are that each one typically encapsulates simpler business functionality, which you scale up or down, test, deploy, and manage independently. One important benefit of a microservice approach is that teams are driven more by business scenarios than by technology, which the tiered approach encourages.

In practice, smaller teams develop a microservice based on a customer scenario, using any technologies they choose. In other words, the organization doesn’t need to standardize tech to

maintain monolithic applications. Individual teams that own services can do what makes sense for them based on team expertise or what's most appropriate for the problem to be solved.

Microservices come to help in solving the downsides developers have with large applications that require change cycles to be tied together. In a monolithic service-oriented architecture (SOA) deployment, each small change mean that the entire monolith needs to be rebuilt and this, in turn, means that re-builds don't happen as rapidly as they should. In a microservices oriented architecture, each microservice runs a unique process and usually manages its own database. This not only provides development teams with a more decentralized approach to building software but also allows each service to be deployed, re-built, re-deployed and managed independently.

At its simplest, the microservices design approach is about a decoupled federation of services, with independent changes to each, and agreed-upon standards for communication.

In a nutshell the monolithic application and microservices architecture application can be compared as follows:

- A monolithic app contains domain-specific functionality and is normally divided by functional layers, such as web, business, and data.
  However, a microservice application separates functionality into separate smaller services.
- A monolithic app can be scaled by cloning it on multiple servers/VMs/containers.
  On the other hand, a microservices architecture approach scales out by deploying each service independently, creating instances of these services across servers/VMs/containers.

There are certain worth noting profits associated with microservices as stated earlier which help us to choose it over other architecture approach:

Microservices can be written in any programming language and can use any framework

The developers of the services are free to choose whatever language or framework they want, depending on their skills or the needs of the service. In some services, the performance benefits of C++ might be valued above all else while in other services, the ease of managed development in C# or Java might be most important. In some cases, developer may need to use a specific third-party library, data storage technology, or means of exposing the service to clients.

Microservices allows code and state to be independently versioned, deployed, and scaled

However individuals choose to write their own microservices, the code and optionally the state can be independently deployed, upgraded and scaled. For scaling, understanding how to partition both the code and state is challenging. When the code and state use separate technologies, the deployment scripts for our microservice need to be able to cope with scaling

them both. This is also about agility and flexibility, so developer can upgrade some of the microservices without having to upgrade all of them at once.

Interacts with other microservices over well-defined interfaces and protocols

Generally, service communication uses a REST approach with HTTP and TCP protocols and XML or JSON as the serialization format. From an interface perspective, it is about embracing the web design approach. But microservices can be used with binary protocols or developer's own data formats.

Microservices Remains consistent and available in the presence of failures

Today, dealing with unexpected failures is one of the hardest problems to solve in a distributed system. Much of the code that we write as developers is handling exceptions, and this is also where the most time is spent in testing. But the problem is more involved than writing code to handle failures. The issue is what happens when the machine where the microservice is running fails? Not only this microservice failure is to be detected, but also something is needed to restart the microservice. However, a microservice can be easily resilient to failures if it is given restart often on another machine for availability reasons. This involves the original state to be saved on behalf of the microservice, to recover this state somewhere, and to be able to restart successfully. In other words, there needs to be resilience in the computer as well as resilience in the state or data for no data loss and the data remains consistent.

Conclusion

To summarize, the microservice approach is to compose your application of many smaller services. Smaller teams develop a service that focuses on a scenario and independently test, version, deploy, and scale each service so that the application as a whole can evolve. Designing with a microservice approach is not a panacea for all projects, but it does align more closely with the business objectives. Starting with a monolithic approach may be acceptable if we know that later we will not have the opportunity to rework the code into a microservice design if necessary. More commonly, we begin with a monolithic app and slowly break it up in stages, starting with the functional areas that need to be more scalable or agile.

In today's world the changing business needs are:

- To build and operate a service at scale in order to reach customers in new geographical regions.
- Faster delivery of features and capabilities to be able to respond to customer demands in an agile way.
- Improved resource utilization to reduce costs.

These business needs are affecting how we build applications. The organizational experiences have reflected that as more and more teams began to build applications for the cloud, many of them realized the benefits of taking a microservice-like approach. The objective of this architecture is to reduce the complexities of building applications, so that organizations do not have to go through as many costly redesigns. Start small, scale when needed, deprecate services, add new ones, and evolve with customer usage, that's the microservice approach. We also know that in reality there are many other problems yet to be solved to make microservices more approachable for the majority of developers. Containers and the actor programming model are examples of small steps in that direction, and we are sure that more innovations will emerge to make this easier.

References

'Microservices' by James Lewis - Principal Consultant at ThoughtWorks and member of the Technology Advisory Board. (http://martinfowler.com/articles/microservices.html)

Microservices – Wikipedia (https://en.wikipedia.org/wiki/Microservices)

Microservies blog on opensoure.com - https://opensource.com/resources/what-are-microservices

Why a microservices approach to building applications? – an arcticle by Mark Fussell (https://azure.microsoft.com/en-us/documentation/articles/service-fabric-overview-microservices/)