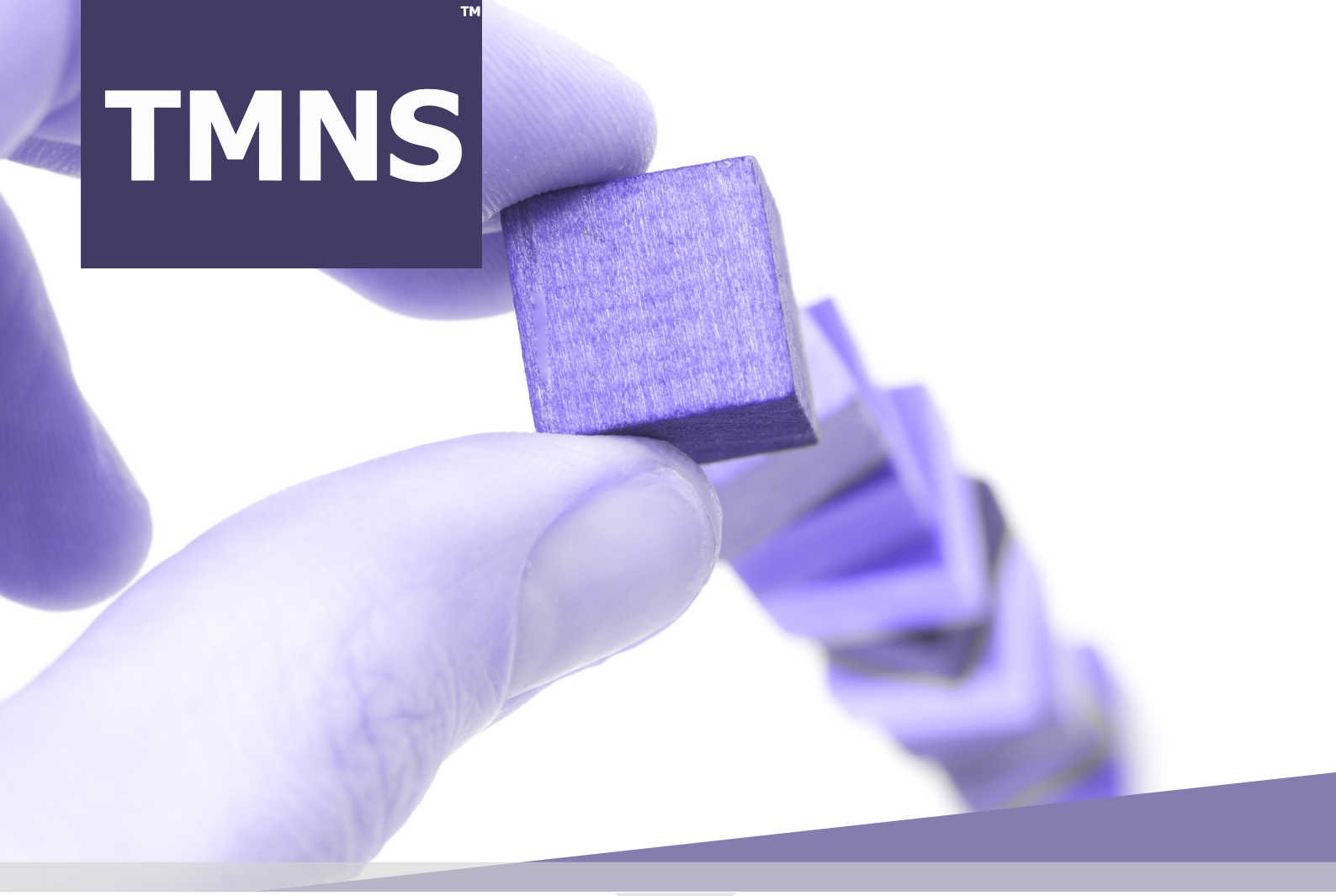
TMNS™



Microservices

This white paper

Introduction to Microservices

'Microservices' is the new buzzword in IT. Though the concept is not new, it can be an enabler for improving your IT delivery process. Microservices architecture is a software architecture style in which a single application is divided in small services, each running on it's own, but communicating with each other using lightweight mechanisms like HTTP resource API.

The good-old approach to software

The most natural way to create an application is to build it as one single unit of code: a so-called Monolithic Architecture. They will have a (often browser-based) user interface and data is stored in a relational database, but the server-side application is still a monolith, one piece of code.

" ...That means regression test and (re)deploy the whole application, with downtime as a result. Besides that, scaling might not be possible, as the capability was not built-in from the start."

And that's where Microservices come in

As this frustrated people, the Microservice Architecture evolved. Applications are built as a collection (or suite) of services. Services are developed independently and focus on doing a small task, allowing them to be deployed independently. These services need a bare minimum of centralized management and it is possible to combine different programming languages and use different data storage technologies.

What Microservices can teach us

Microservices Architecture is a good place to start your Integration Layer design principles, giving you tremendous opportunities to continuously deliver robust software with great business value.

Microservice Architecture vs. Service Oriented Architecture

"The Microservice Architecture is the anarchists' answer to the monopolization of integration."

The key ingredient of Microservices Architecture (MsA) is not just the smaller-than-SOA-services, but the decentralization and simplification of the development of these services.

In an MsA, different teams are encouraged to develop simpler services, to release them quickly and to release them often. This is hard to accomplish using a Service Oriented Architecture (SOA), where services are rarely completely loosely coupled and often part of a Mikado-style web of services.

It is true that Microservices are smaller services than services and are typically found in a Service Oriented Architecture, but that is about as far as this comparison goes. The Microservices Architecture includes certain additional concepts surrounding these 'smaller services' and this is where the differences with SOA begin to show.

Pro's of Microservices

+ Individual components
Just like in machines, it makes sense to build software in components. Microservices Architecture sees services as individual components that can be individually developed and replaced, just like in your car. No need to replace the whole thing when you can replace the component.

+ Organized around business capabilities
Microservices are organized around business capabilities, not around technical capabilities. So we do not have separate user interface-, database- and middleware-teams, but cross-functional teams, e.g. a Telecoms Fulfillment team.

+ Re-use of services
Microservices are project-agnostic, i.e. they provide business capabilities (e.g. 'lookUpCustomer') instead of providing project-specific functionality. This enables re-use of services and thus reduces costs.

+ Logic communication between components
Just like in machines, it makes sense to build software in components. Microservices Architecture sees services as individual components that can be individually developed and replaced, just like in your car. No need to replace the whole thing when you can replace the component.

+ Only use what you need
Enterprise Architectures tend to be prescriptive about technology platforms and standards to use. But like in life, you only want to use the tools you need. An electrician doesn't need an axe. In Microservices Architecture, the approach is to use tools as they're needed and let it evolve into a set of tools and standards.

+ Decentralized Data Management
Enterprise Applications tend to be three-tier, with the data in a central database. Applications have their own view on the data. In Microservices Architecture, each component manages its own data. Though companies need measures to prevent data inconsistency, this approach simplifies data management and by that: costs.

+ Design for failure
When you build Microservices, your applications need to be able to tolerate failure. After all, a service can be unavailable. As paradoxical as it sounds, this in fact makes your whole architecture more robust. Murphy can't hit you when you anticipate he's around there somewhere.

+ Evolutionary Design
When breaking down an originally monolithic application into components (Microservices), start with the parts of your application used or changed most often. As this is an iterative process, the design should change or 'evolve' as well.

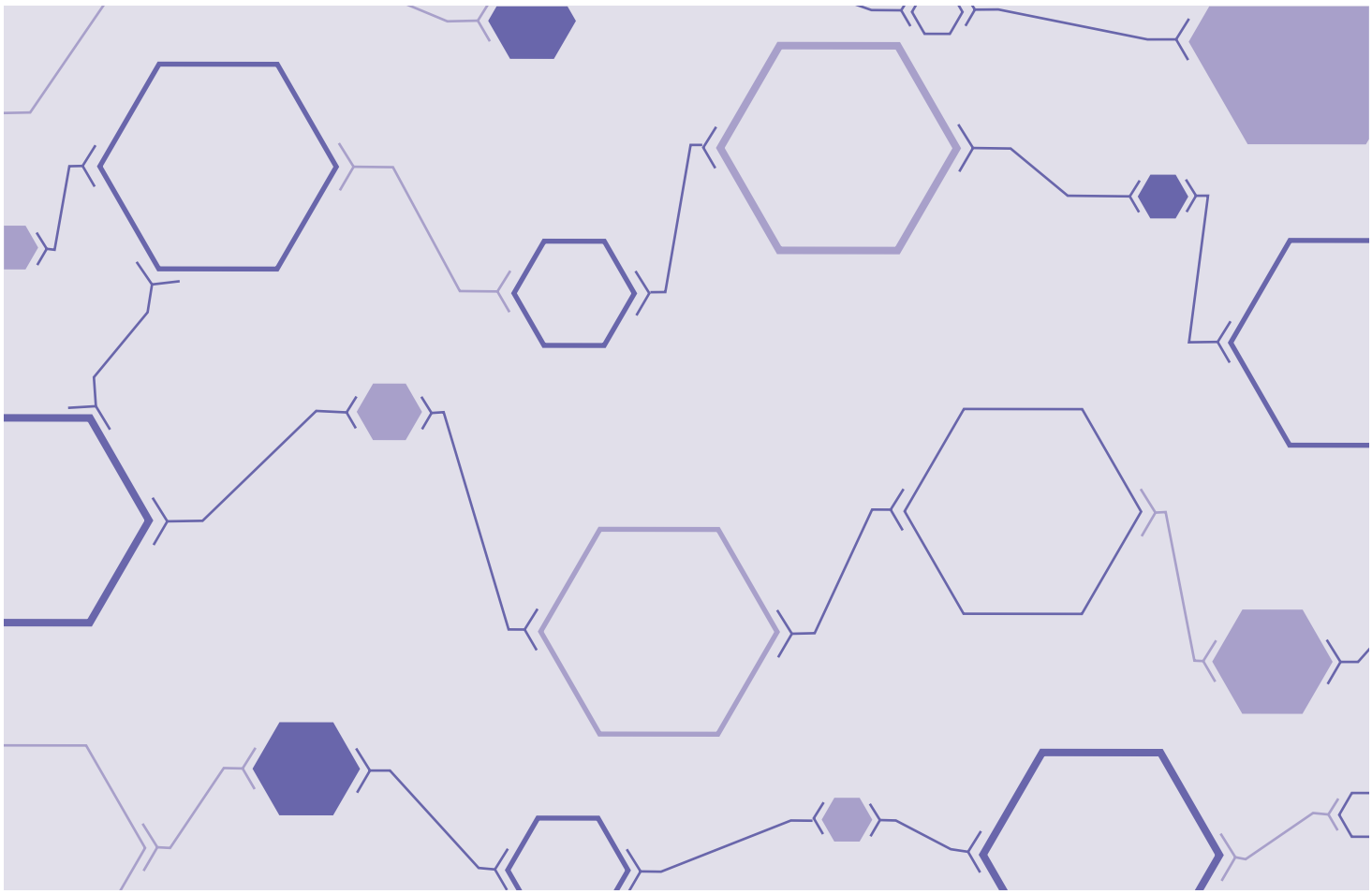
Advantages and disadvantages of SOA

SOA and ESB

An SOA is most often implemented on an Enterprise Service Bus (ESB). Such an ESB is almost always implemented centrally by one department in the organization. This department is made responsible for managing the scale and complexities that come with a centralized ESB.

To handle these complexities all kinds of processes are formed around the ESB, for example service lifecycle management, governance processes, etc. These processes are not necessarily a bad idea and to a certain degree they are all needed to make sure that the benefits of SOA can be harvested.

However, the downside is that all too often this leads to institutionalization of the ESB in the form of a Center of Excellence (CoE) which develops its specific service-frameworks and monopolizes all the integration effort of the organization. The common effect of this is that when the organization starts scaling up the use of the ESB, they experience a drop in effectiveness and velocity of the development team.



Microservice Architecture
vs. Service Oriented Architecture

As mentioned earlier, the focus of these Microservices is on business capabilities rather than technical capabilities. This means that the entire stack of components underlying these Microservices is managed within one team of experts. This gives more autonomy to the teams that develop these services. Decentral governance of service development sometimes proves difficult to adopt in organizations as it could clash with existing culture. However, these differences are never insurmountable. The autonomy of the team reduces the need to have agreements and formal specifications between the database, middleware, business logic and UI. This is an 'internal' advantage of MsA over traditional SOA. Which may have its effects on 'the outside' as well.

"A proper Microservice is 'designed for failure'"

A proper Microservice is 'designed for failure' and offers a service independent of other services. This may seem impossible as services inevitably dependent on other services. However, the idea behind this concept is that services do not require other services to be available all the time. They are designed to be resilient to failures and will simply create a backlog of work for those services that were temporarily unavailable. This way services can be designed to be independent of each other operationally.

Technically a service interface still needs to be defined and this will result in dependencies as well. Other services from other teams of experts will rely on these Microservice interfaces. Although there is no final answer to this dependency problem, MsA offers some interesting solutions which it borrows from RESTful Architecture and HTTP such as HATEOAS and ETags.

Microservice Architecture
vs. Service Oriented Architecture

Caching options
Due to the small size of Microservices, the information is very suitable for caching. Combine this with -again- entity-tags, a concept from the HTTP protocol, and efficient client-side caching is possible. This is a tremendous simplification of infrastructure compared to centralized caching solutions.

How come we didn't think of this before?
With all the hype surrounding the Microservices Architecture it may seem that there are no advantages left to SOA at all.

Why did we even do SOA and ESB in the first place?
There are still some significant advantages that SOA holds over MsA and which are harder to address in a MsA. A typical SOA on an ESB provides much greater control over what is disclosed and how it is disclosed by enforcing service policies on the boundaries. Regularly an organization deploys adapters to integrate existing systems with the Service Bus. These adapters sport a framework that is supportive of the policy enforcement. Given the decentral nature of MsA this would be harder to implement and enforce from within the organization.

Other common SOA components that (almost always) come out of the box are monitoring, logging, error handling and integration. Again, MsA has a bigger challenge addressing these points. Another advantage, which is at the same time a disadvantage of SOA, is the ability to develop standardized messages which greatly enhances integration and orchestration efforts. This blessing in disguise comes with its dark side: rigidity. A typical SOA-service is notoriously hard to modify just because of this standardized message model and its integration dependencies.

Now what do we do?
The advantages of Microservice Architecture over Service Oriented Architecture are clear, however they may not always fit the goals of the organization. To decide whether a SOA or MsA is a better fit, one should look at the needs of the organization, the results the organization wishes to achieve, the cultural fit and the existing competencies that are available or need to be attracted. Of course TMNS is always willing to help bring your ambitions to life!

Because Microservices are small, separate components it's easier to automatically test, and less risky to deploy changes. This opens the door to Continuous Delivery, enabling a quicker time-to-market with less costs and risks.

With Continuous Delivery we strive to make deployments as boring as possible. To assure that we try to deploy often and find errors as soon as possible. A number of best practices in a Microservices Architecture help us to achieve that.

Smart endpoints and dumb pipes

By making sure your pipe – read your Enterprise Service Bus – does only what it is meant for (transportation and transformation), you create highly independent services. All the logic is in the endpoint and no logic is in the pipe. This makes it easier to test the part of the application that you are working on without the need to have an environment with the complete system on it.

Of course you need to do a complete end to end test (and automate that). However, you can deliver higher quality in one part of the system because you can be more confident than in a monolithic architecture that is it works on itself. It will work in the whole.

By taking the logic out of the pipe you make it easier to build quality into your applications. No big dependency management system which prevents you from detecting problem in an early state. The introduction of techniques like service virtualization is easier in this setup and helps to maintain a high quality.

Easy deployment pipeline

When implementing Continuous Delivery, a pipeline is created to move software from development through production. The purpose is to do this often and to get feedback to the developer quickly. With a monolithic application setting up the pipeline can be very complex. In order to prepare the environments needed a lot of dependencies have to be met.

The goal of Microservices is to make a part of a larger system that can be taken to production on its own. The only dependencies you have are therefore in your own part of the system. This generally makes it a lot less difficult to set up environments. And thus makes it easier to deploy fast and to automate the complete cycle.

You build it, you run it

Teams working with Microservices tend to like the "you build it, you run it" approach. The team is not only responsible for delivering good software, but also for the performance of the software in the production environment. This gives a big incentive for the team to deliver software with high quality. And the team will make sure that, by automating tests and testing as early as possible, good quality is delivered.

This again removes a dependency. Not a technical one but an organizational one. The team itself has to be able to run the software and maintain it. So there is no need to create cross organizational processes to maintain the software

Resilience

When applying Continuous Delivery we try to ship code to production as often as possible. That is sometimes seen as a risk. In many organizations releases are big events that are done only a few times per year. The idea is that releases are a big risk and should be avoided as much as possible. And because of dependencies it is not possible to do it more often.

With Microservices you create a system that is very resilient. When one Microservice fails, your system is still running. It might lose a part of the functionality but the rest is still working and serving your customers. That reduces the risk of deployment. Even if a deployment breaks, the impact is small. And it is easy to roll back.

ABOUT TMNS

TMNS strives to improve the Customer Experience and system agility of companies by offering overall integration technology solutions and services. With over 150 top-notch consultants, situated at offices in The Netherlands, Switzerland, Germany and Serbia, TMNS supports a diversity of clients in becoming more responsive in order to adapt to the continuously changing market dynamics. Customers industries are telecommunications, financial services, utilities, life sciences retail and manufacturing. TMNS has built strong expertise on Business Process Management, DevOps, Enterprise Application Integration, Business Intelligence and Cloud Based Solutions.

CONTACT

Prinses Catharina Amaliastraat 5
The Hague 2496 XD
The Netherlands

T: +31 70 3011 720
W: www.tmns.com
E: info@tmns.com

REFERENCES

Ansems, S. (2015, June 2).
Microservices: just another word for 'tiny-SOA'? Retrieved September 1, 2015.
Ansems, S. (2015, June 2).
Why Microservice Architecture? Retrieved September 1, 2015.
Fowler, M., & Lewis, J. (2014, March 24).
Microservices. Retrieved June 17, 2015.
Van Bart, A. (2015, June 17).
Microservices and Integration: Doing IT right? Retrieved September 1, 2015.
Van Halem, G.J. (2015, August 7).
Microservices and Continuous Delivery: Reducing Dependency and Risk. Retrieved September 1, 2015.