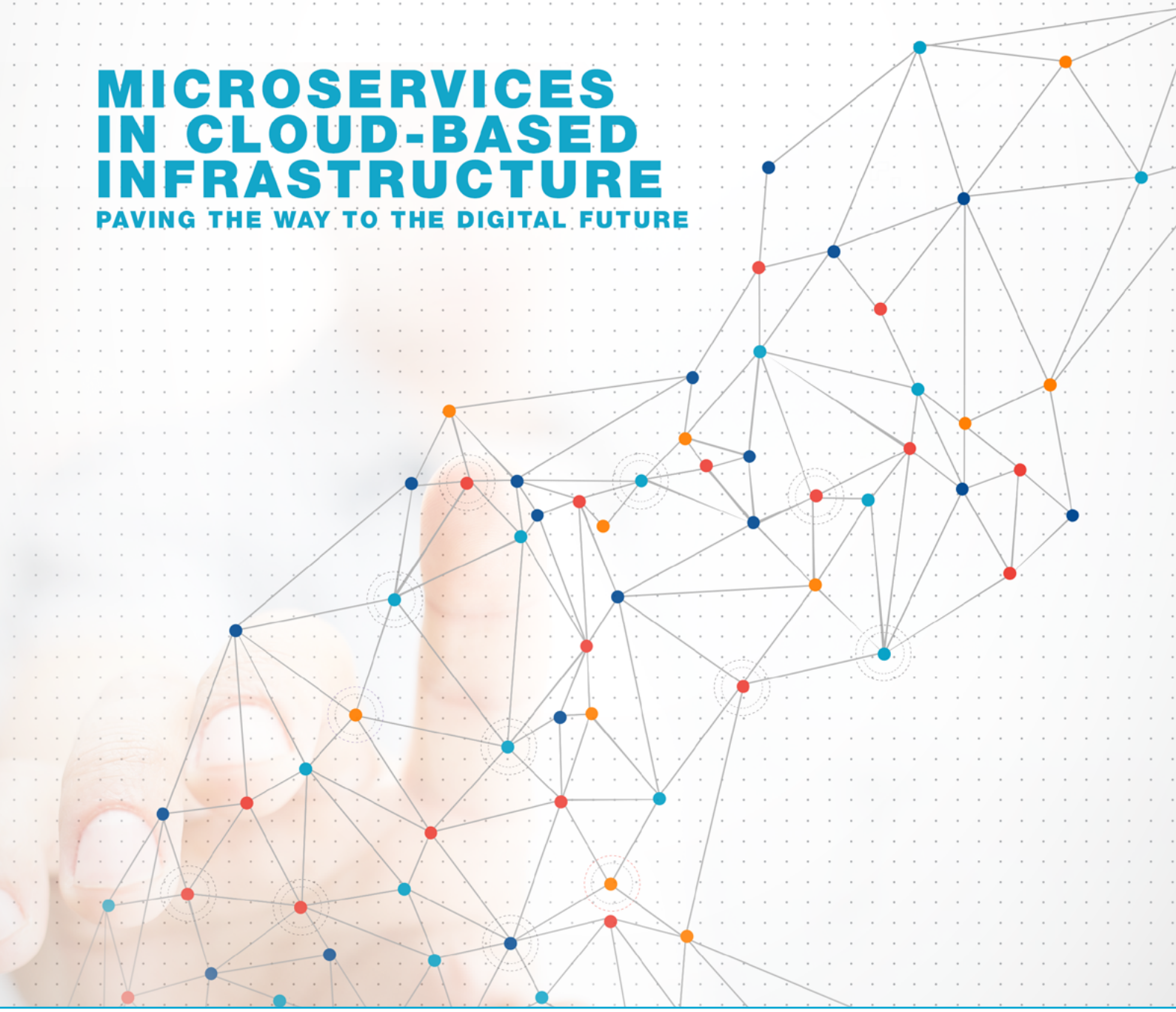
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Infrastructure Services | the way we see it

MICROSERVICES IN CLOUD-BASED INFRASTRUCTURE

PAVING THE WAY TO THE DIGITAL FUTURE



People matter, results count.

Introduction

Microservices demand a new way of constructing infrastructure capabilities – microservices-aware infrastructure has to follow a Lego®-Brick approach"

Everything comes in cycles. Back in 2005 we discussed, defined and developed service-orientated architecture (SOA) based solutions[1]. Today, when asked what the best-in-class software blueprint is, we tend to refer to microservices[2].

Behind both terms sits the same, simple concept – to develop and design independent, "utility-based" services that can drive and provide flexibility, security and agility[3].

We are moving from the 2nd to the 3rd platform (the 1st refers to the mainframe era (monolithic), the 2nd refers to client-server based computing and the 3rd is the web native landscape where users have access to apps and data from everywhere at any time with any device). More and more organisations are moving into a digital landscape where applications, data and automation are king.

Introduced 10 years ago, microservices are one way to design and build applications. Both a monolithic and/or SOA based approach is and will still be a valid option. However, the application of microservices based design is on the increase, in particular for organizations that make use of 3rd platform based capabilities. Microservices demand a new way of constructing Infrastructure capabilities – Microservices-aware Infrastructure has to follow a Lego®-Brick approach. The resulting infrastructure platform will have to be fully software based. This implies that control of the data center is fully automated by software so that hardware configuration, storage provisioning and network configuration are managed through software. This is in contrast to traditional data centers, where the infrastructure is typically defined by hardware and devices.

To achieve full digital maturity, and to accelerate the digital journey, an organisation's infrastructure landscape has to conform to a clear set of design patterns – for example, ease of use, self-service, agility and flexibility.

However, it can be tricky to navigate around all the terms involved, so to provide some guidance on microservices, this document was created with two main intentions: To provide a detailed view of the impact microservices is having on infrastructure; and to examine how today's infrastructure should be designed to support microservices.

Gunnar Menzel

VP, Chief Architect, Infrastructure Services

1 See Capgemini's SOA (Service Orientated Architecture) point of view papers as well as whitepapers [1,2] were we outlined the notion of "everything as a service" in 2005.

2 The term "microservice" was discussed at a workshop of software architects near Venice in May, 2011

3 https://en.wikipedia.org/wiki/Microservices

Service Terminology Unravelled

There are many architectural buzzwords, such as service-orientated architecture (SOA), event-driven architecture, microservices and software defined data centre (SDDC). In the absence of standard definitions, some common views are that services are business or technological capabilities, or physical products or an architecture approach. In fact, a service has many abstraction levels as the term “service” can be used in different contextual settings:

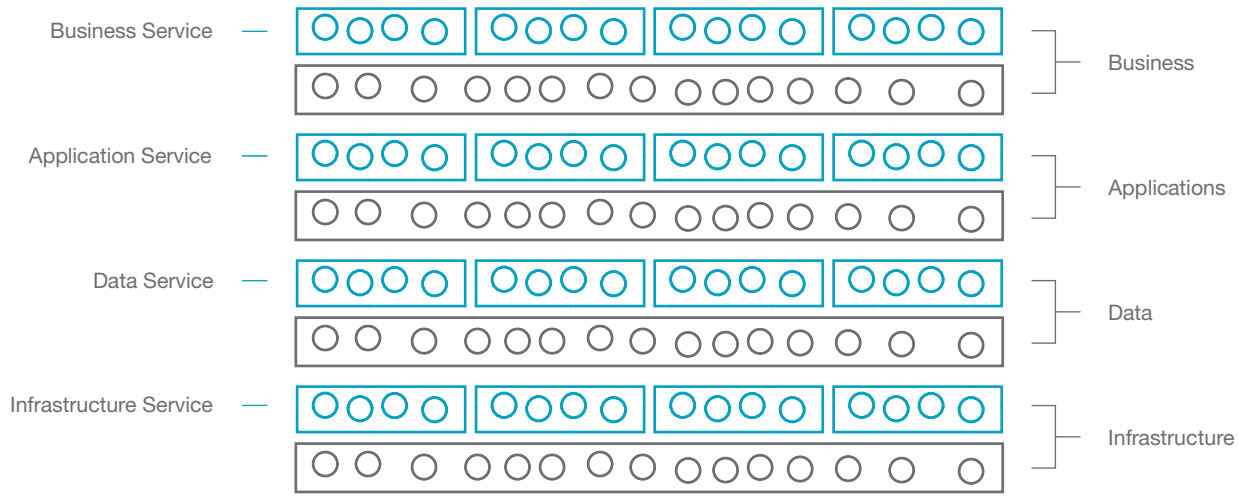


Figure 1 | The different types of services

Service (Oriented) Architecture

A design pattern mostly related to an enterprise architecture related approach, which aligns business, information, information systems and technical infrastructure

Business Services

A design approach for business operating models, which combines business processes and business events, and which has defined value contracts

Application Services

A design approach to deliver application functions supporting Business Services, implemented through a variety of technology solutions and standards (this is the area microservices is mostly related to)

Web Services

A special application service implemented using Web services standards for mass access, specifically to receive and return XML documents, for example within a defined contract; pervasive standards make the difference

Information Services

A design approach to deliver information to application services, implemented through a variety of technology solutions and standards

Infrastructure Services

Specialized or shared infrastructure services, which support application, Web and information services

The main intention of a service based approach is to develop or create fine grained capabilities that can easily be used in different context settings.

Microservices

Microservices relates to a design pattern that refers to a number of independent application services delivering one single business capability in an independent, loosely connected and self-contained fashion.

Following the SOA style (see below), microservices are more autonomous, highly independent and much smaller or finer grained than their SOA based counterparts.

Microservices architecture can be viewed as a marriage between component-oriented architecture and service-oriented architecture. Software as a suite is composed of many small business components with very specific business domain responsibility. Their interface to the outside world is through an application program interface (API) of clearly defined services.

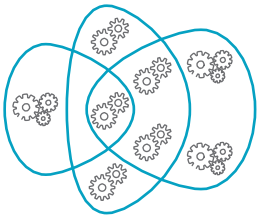
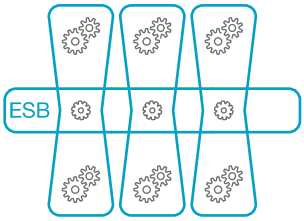
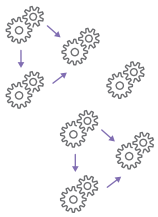
ONE SIZE FITS ALL	FINE-GRAINED COMPONENTS	SUPERFINE
<2000 Monolithic Approach / 1st Platform	**2000–2010** Service-oriented / 2nd Platform	**>2010** Microservices / 3rd Ptatform
		
A software design pattern that includes all functional and non-functional features into one “box.”	SOA is mainly about “exposing” discrete components of an application as web services.	Independent application services delivering one single business capability in an independent, loosely connected and self-contained fashion.

Figure 2 | The three main software patterns

One area where microservices departs radically from SOA is in the ownership model for the services: with microservices, only a single team or person will develop and change the code for a given service – Netflix is one organisation that is using this approach. The reason microservices has taken off is wholly down to it being a realisation of many aspects of Eric Evans’ Domain Driven Design [1] – in that they can be created and maintained by small, independent teams.

Loosely coupled services can be updated independently of each other; a group of small services that has to be updated together are not microservices because they are not loosely coupled. This also applies to sharing a database across services, where updating a particular service means changing the entire schema (or where a change to a shared schema results in a change to >1 microservice – an “antipattern”).

Microservices Best Practice

As outlined before, microservices and SOA have a lot in common. A service oriented approach is mainly about "exposing" discrete components of an application, as web services and SOA based applications are comprised of more loosely coupled components that can be used in different contexts. SOA was a key pattern to move from the 1st to the 2nd platform: Microservices are key to moving organizations onto the 3rd platform.

Developing and providing a detailed description of a way to define microservices-based applications would go far beyond the scope of this document. However, to understand what characteristics an infrastructure landscape has to comply with in order to support microservices-based applications, it is important to define microservices-based design principles. The key design criteria for microservices are related to the service descriptions.

"Good" microservices:

1. Have fine-grained capabilities that are stateless[4] and are "stupid".
2. Have well-defined interfaces, which hide how the service is executed (i.e. I care about the interface, not how the service is executed "under the hood").
3. Use a "very loosely-coupled" approach (one service can be changed without impacting another).
4. Are "disposable", and have "no ESB[5] – "dumb pipes and smart services".
5. Are autonomous, explicitly versioned and highly independent.
6. Are stupid – i.e. they "do one thing, and do it well".
7. Are fully cost and value defined.

Following these design principles leads to a microservices-based architectural style; to develop and implement microservices-based applications, other aspects must be covered. What is important from an infrastructure perspective is that microservices based applications demand a "utility-based" approach where infrastructure is "invisible" to the requesting microservice. Utility-based infrastructure service delivery refers to the packaging of infrastructure resources, such as computation, storage and data transport, similarly to metered services like traditional public utilities (such as electricity, water, natural gas and the telephone network). In other words: a utility infrastructure service is a metered service supplied through shared infrastructure, in which services are separated, supporting the above defined design principles.

> *"A utility infrastructure service is a metered service supplied through shared infrastructure, in which services are separated, supporting the design principles as defined previously."*

4 This is becoming less the case in certain architectures. There was a talk at Strange Loop conference in 2015 around stateful microservices. Statelessness does therefore not determine whether something is a "microservice". It is, however, something to be avoided unless absolutely necessary.

5 ESB = Enterprise Service Bus

Microservices and Infrastructure

For microservices to work, the underlying infrastructure capability has to support the microservices design pattern and must focus on all relevant non-functional characteristics (availability, stability, reliability, performance and security). Microservices will require "infrastructure" out of the box, which can be constructed in a "Lego®-based" approach [2].

Infrastructure near capabilities like compute, storage, network, as well as infrastructure related capabilities like load balancing, replication, etc., must have a "plug and play" approach and the implementation should be invisible to the consuming microservices.

One key enabler of invisibility is clearly cloud; using a cloud approach to infrastructure will accelerate the adoption of microservices based applications.

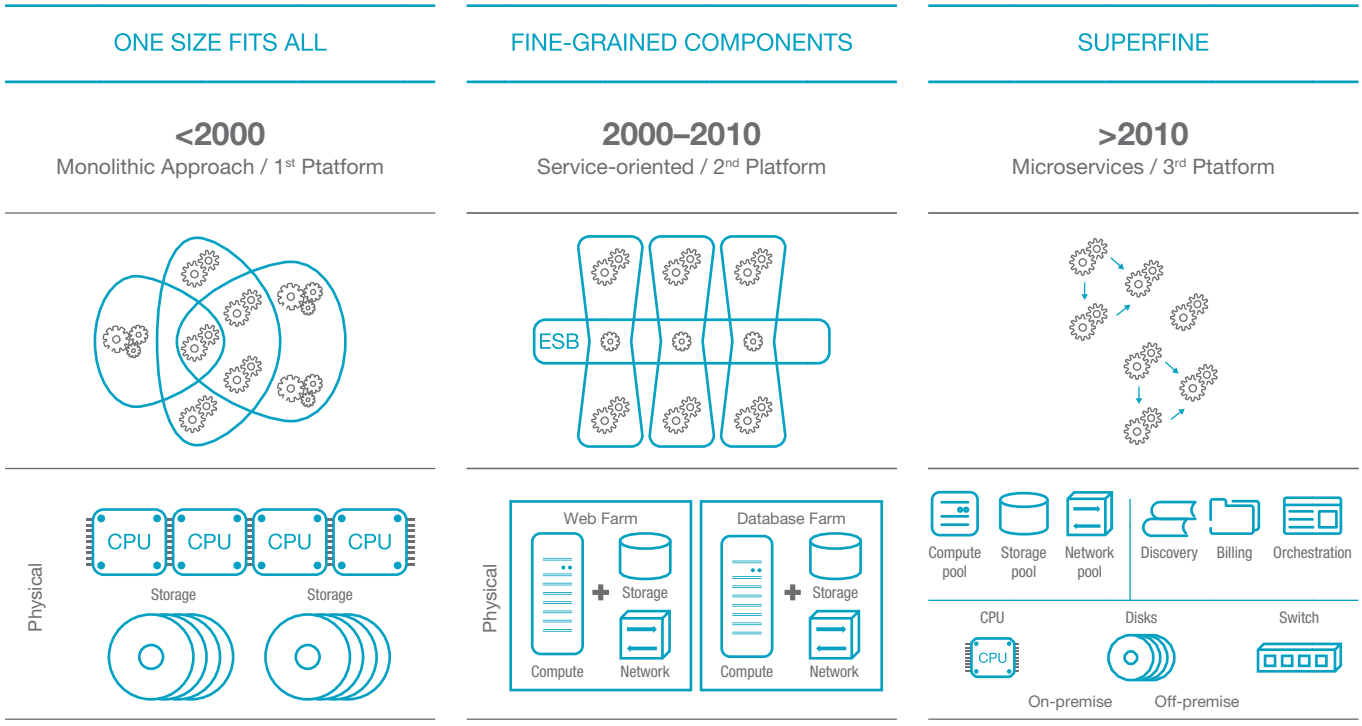


Figure 3 | Software pattern and the related infrastructure blueprint

Infrastructure, or more precisely, technical infrastructure, is the combination of all technical components needed to store, manipulate and transmit data used or consumed by information systems (applications) in a particular context; with certain performance characteristics and set against a set of so-called non-functional requirements.

Microservices and Infrastructure Design Principles

To create “invisibility” for microservices, infrastructure services have to comply with particular characteristics.

Looking closer, it means that a microservices developer must have not only the ability to create, use, move, expand, contract, delete and compute, but also have network and storage capability. This includes all customer facing infrastructure services.

This is where concepts like convergence, hyper convergence and fully convergent infrastructure (or better software bases) deployed on or off-premise using private, public or hybrid cloud will be the template for many organizations to make use of microservices.

A microservices based infrastructure platform should be fully software based.

This means that control of the data center is fully automated by software, so that hardware configuration, storage provisioning and network configuration are managed through software. This is in contrast to traditional data centers where the infrastructure is typically defined by hardware and devices where changes might reqiure modification executed by manual (hands-on) activities.

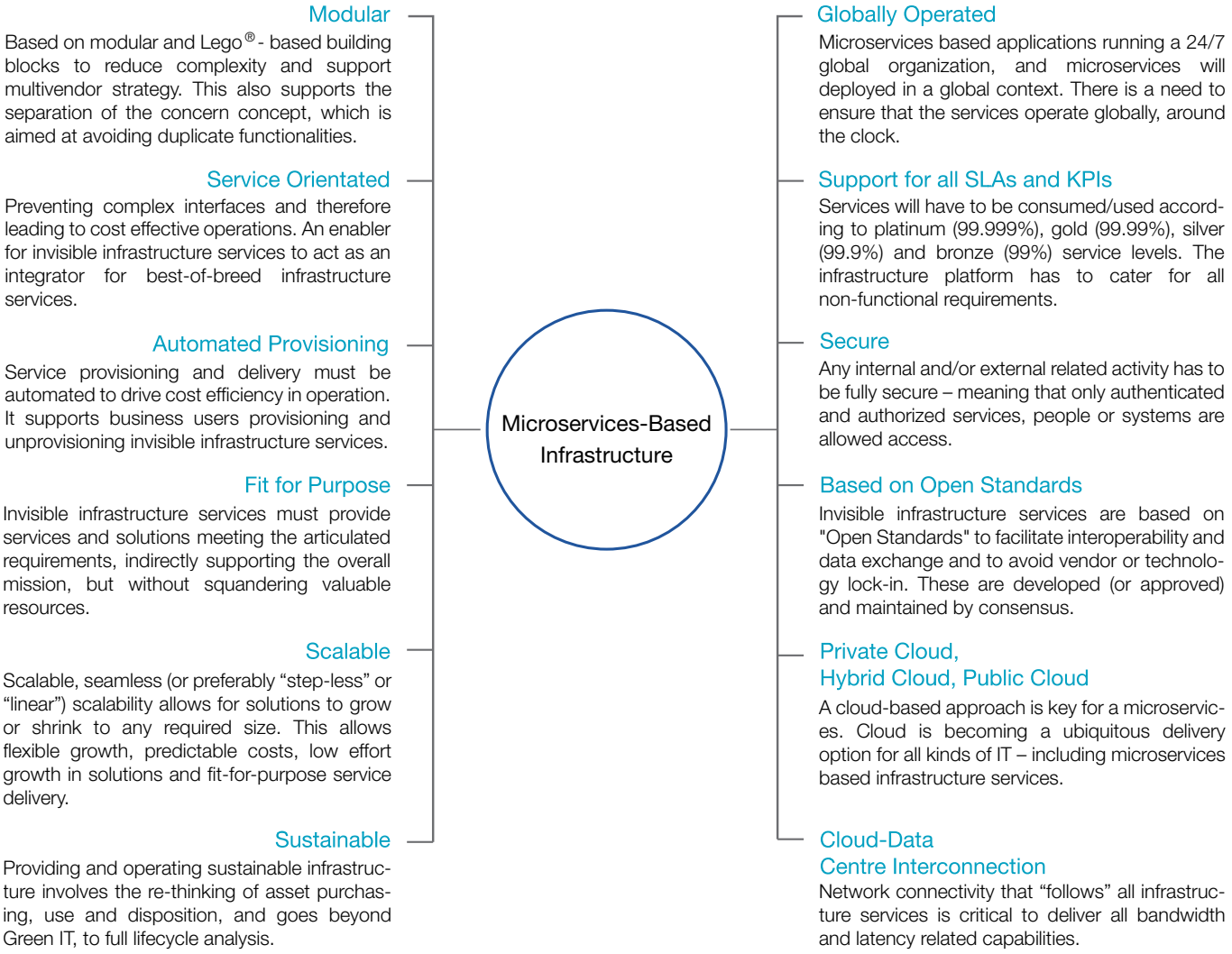


Figure 4 | Infrastructure characteristics to support microservices

The Lego®-Brick Approach

Microservices requires infrastructure services that are able to comply with the key service related requirements as outlined before. This means that a person (or a microservice) who requires infrastructure services, should be able to construct and consume infrastructure capabilities following the Lego®-brick principle by using a “simple” shopping list and delivering utility-type services.

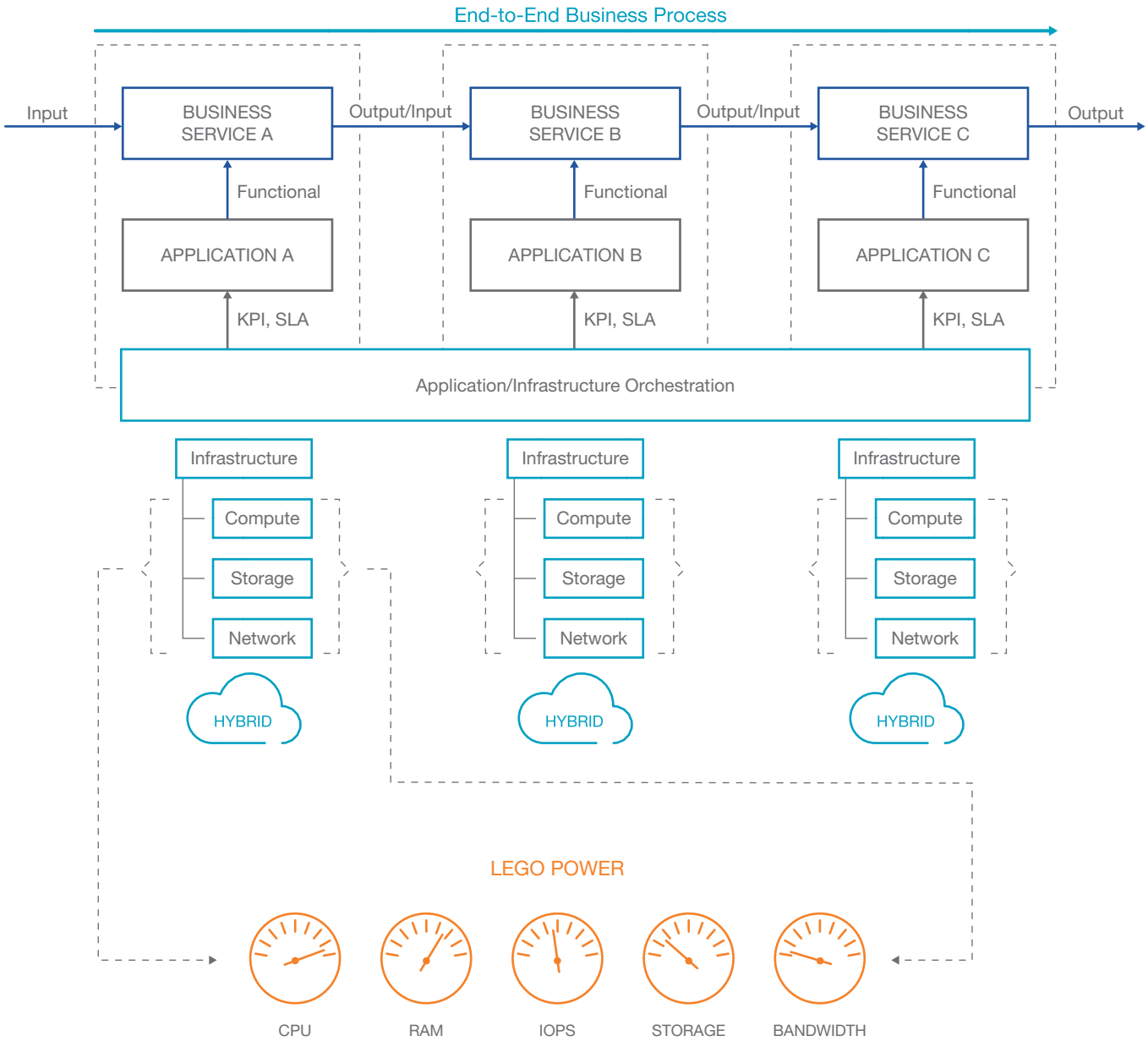


Figure 5 | Utility based Lego® power

Infrastructure services should have standard building blocks (such as server, storage and network), as well as additional components that are constructed from these building blocks (such as web server, pre-production environment, etc.). Each standard building block and the additional components should be able to cater for different SLAs and KPIs without modification.

For instance:

- For a Web server:
  - A simple web server compliant with Bronze SLA or
  - A web server compliant with Platinum SLA

- A data storage solution that:
  - Is optimized for standard database storage or
  - Can provide maximum capacity or
  - Is highly available and highly performing

- A full application environment to:
  - Develop the application further or
  - Perform full-load and performance testing or
  - Run-critical training for end users

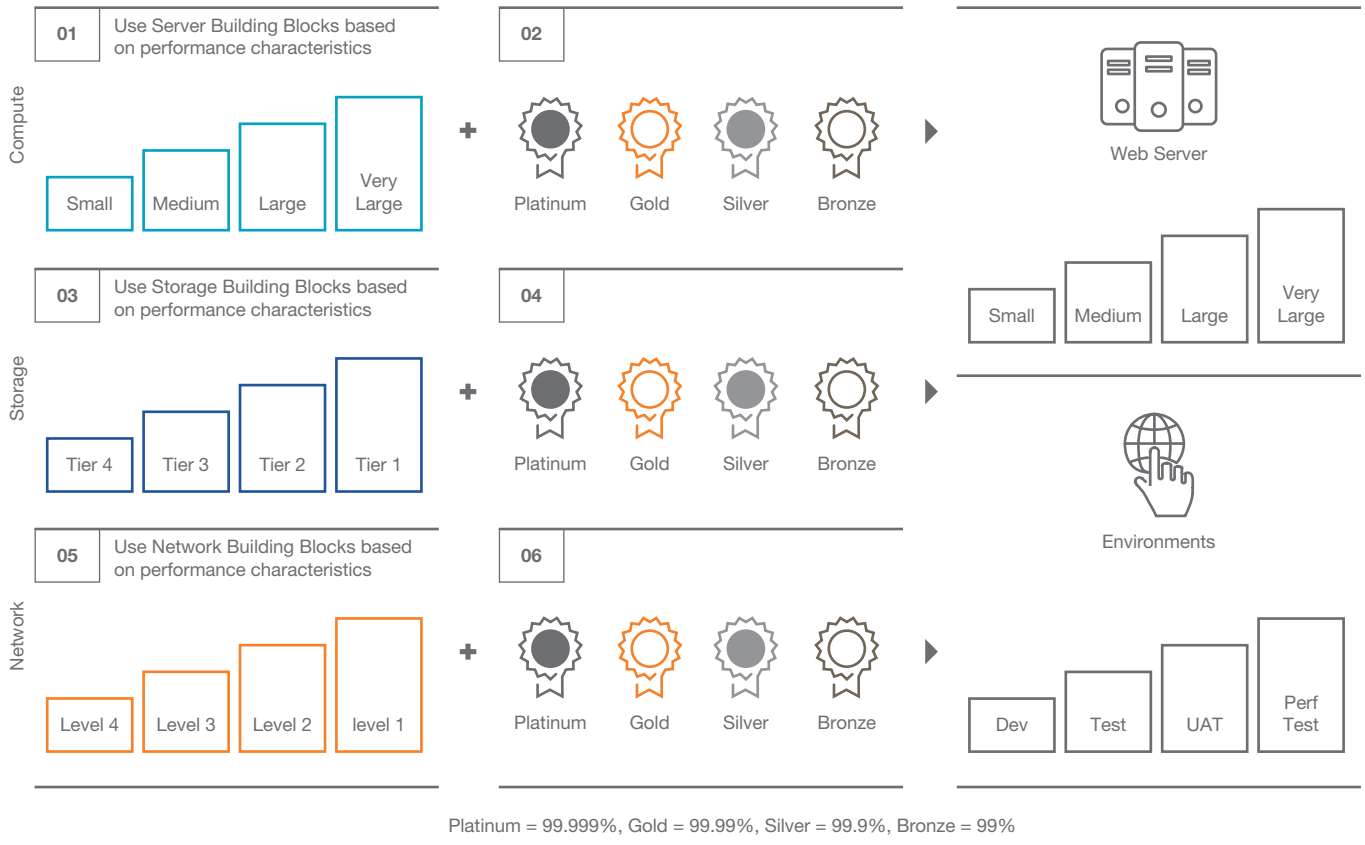


Figure 6 | Infrastructure Lego® based approach

Key components of this “shopping list Lego®-based approach” are the service level characteristics. Each client environment is different. However, each client will have a certain set of service levels – from Platinum (99.999%) to Bronze (99%), and of course a whole set of different non-functional requirements that will shape the underlying infrastructure. Examples are:

Scalability:	Ensure the solution supports current and projected business volumes
Reliability centering:	Ensure the solution provides an appropriate level of robustness in support of business processes
Manageability:	Ensure the solution can be managed and maintained efficiently and effectively
Availability:	Ensure the solution provides the required levels of service

Both the service level, from Platinum to Bronze, and the non-functional requirements can be translated into very specific infrastructure related requirements, from which standard Lego®-based infrastructure components can be defined.

Each pre-assembled infra Lego® block will be supplied with standard or pre-defined “behaviors” and each infra Lego® block will be able to cater for certain availability, stability, performance and security related requirements – so called service characteristics. As mentioned above, there are four main categories: platinum, gold, silver and bronze, which are defined as illustrated below.

PLATINUM	GOLD	SILVER	BRONZE
Mission Critical “5 nines”	Highly Critical “4 nines”	Critical “3 nines”	Non-Critical “2 nines”
• 99.999% availability • 2h RTO, 5min RPO • 7 days 24 hours • Two DC implementation • Hot live standby • Data restore <=2h • 1h response for P1 calls • Highly secure	• 99.99% availability • 4h RTO, 30min RPO • 7 days 24 hours • One data center • Cold standby in second data center • Data restore <=4h • 2h response for P1 calls • Highly secure	• 99.9% availability • 12h RTO, 4h RPO • std backup • Extended weekend office hours • Data restore <=8h • 4h response for P1 calls • DR – Tape • Medium secure	• 99% availability • 72h RTO, no RPO • std backup • Extended office hours • 4h response for P1 calls • Low security
Downtime pa 5.26 min	Downtime pa 52.6 min	Downtime pa 8.76 h	Downtime pa 3.65 days
Cost + + + + +	Cost + + + +	Cost + + +	Cost + +

Figure 7 | Possible[6] service level characteristics

Note that this grouping and the detailed business service characteristics are only examples. Each client environment is different and for some a gold business service (or IT Application) might have a 6-hour recovery time objective and a 45-minute recovery point objective.

In the microservices world, infrastructure services have to be consumed in a repeatable, automatic and simple fashion. Non-functional requirements are a key consideration when constructing infrastructure services as these will act as functional infrastructure requirements to be created, used, moved, expanded, contracted and deleted as needed.

6 Each client environment is different and therefore the detail provided here are only examples

Microservices Infrastructure – The Lego® Blueprints

As per the Lego® principle, each construction using infrastructure Lego® bricks (ILB) will follow a different blueprint to cater for the different set of non-functional requirements and behaviors needed. As with real Lego® bricks, there are endless options to build a plane, a house or castle – whatever is needed. The only restrictions are the physical dimensions and characteristics.

> *Using the same sized bricks, you can build any model, on any scale"*

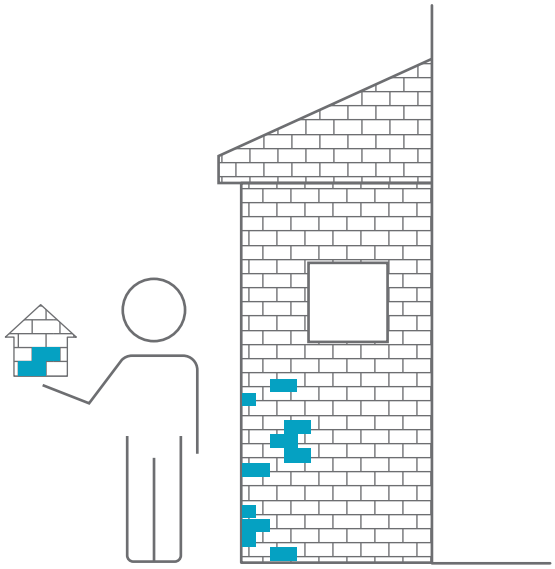


Figure 8 | Lego®'s scalabilty, one size fits all

Using the Lego® principle in an infrastructure context, the same principles apply – one set of ILBs can be used to create one pre-production environment and the same can be used to create a highly available production environment. The key is to use standard ILBs and construct the target based on a clear blueprint.

The Resulting Conceptual Model

A microservices based infrastructure has to fit into an enterprise-wide architecture model[7] that covers all related infrastructure, as outlined in this document.

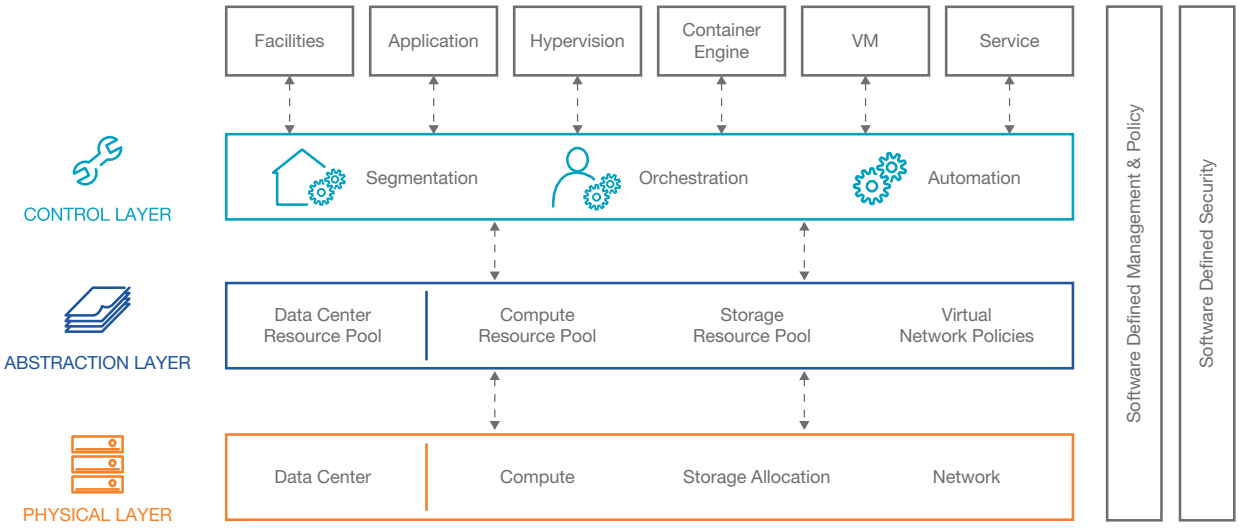


Figure 9 | Infrastructure Lego® based approach

In particular, aspects, such as segmentation, orchestration, automation, pooling and security in a cloud context are key conceptual capabilities an organisation should consider when embarking on using microservices. Such a conceptual model can help with performing precise product selection as part of transforming from a SOA to a microservices based approach.

7 The same applies to SOA; with some key differences

Microservices Infrastructure – Possible Implementation Steps

No client environment will move to use or provide microservices in one step[8] – and to ensure that existing infrastructure services are able to support and provide microservices based capabilities, the following four steps should be considered:

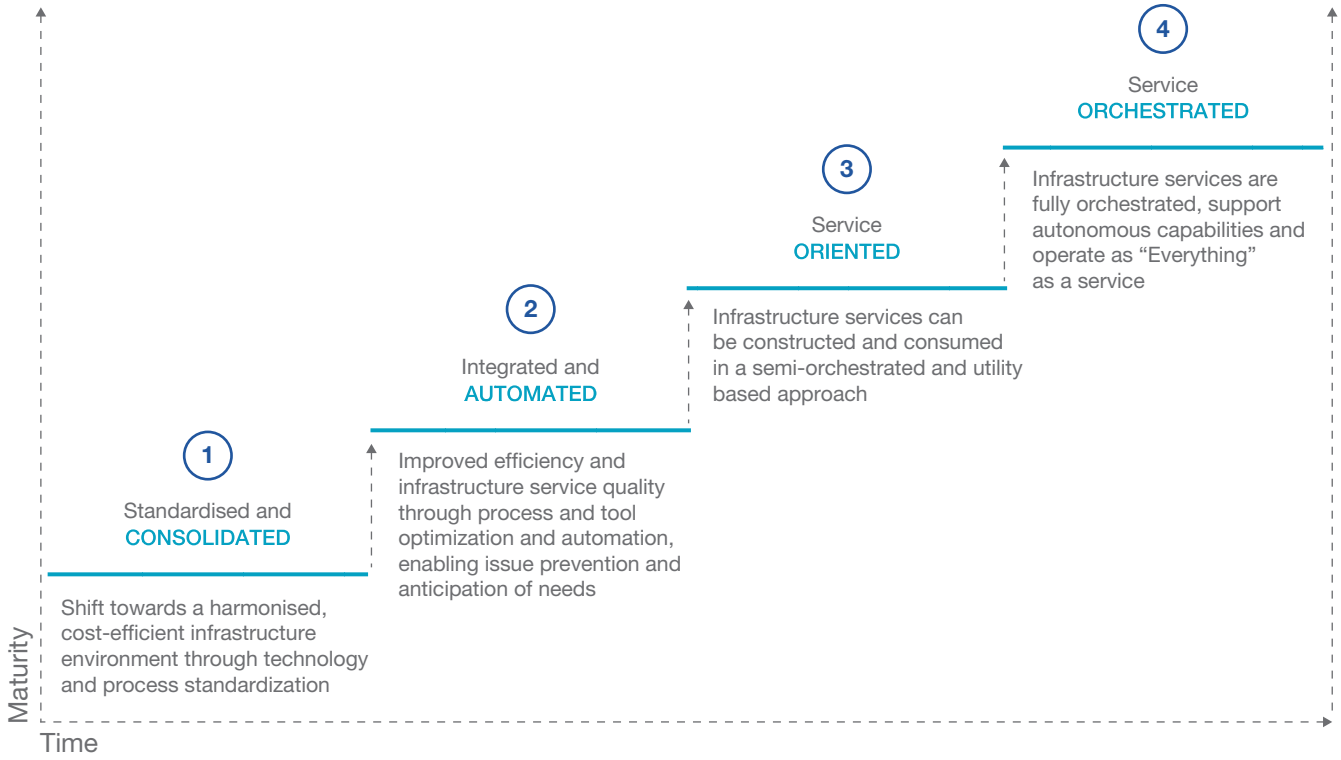


Figure 10 | Microservices implementation steps

8 Unless the implementation is "green field" of course

Summary

Microservices are not always the best option: Monolithic and SOA based solutions still have (and will continue to have) a place in IT going forward. However, microservices is the state-of-the-art software pattern, and it is impacting the way we design, build and operate infrastructure services.

The key to success from an infrastructure perspective is applying a Lego®-based principle to create infrastructure services that follow a "plug, play and orchestrate" approach and where the implementation is invisible to the consuming microservices.

However, the path to a microservices-based infrastructure landscape is not straight – service complexity, implementation cost, the existing landscape as well as the ever-changing new environment must be overcome. Successful organizations that are making use of microservices-based infrastructure follow the four-step implementation plan: consolidation, automation, orientation and orchestration.

Appendix: References

[1] Eric Evans, 2003, Domain-driven Design: Tackling Complexity in the Heart of Software http://www.amazon.co.uk/Domain-driven-Design-Tackling-Complexity-Software/dp/0321125215Service-Oriented Enterprise: How To Make Your Business Fast, Flexible and Responsive, Capgemini 2005, http://www.cio.co.uk/cmsdata/whitepapers/3651/soe_flexibleresponsive.pdf

[2] Virtual Lego, Technovision 2016, https://www.capgemini.com/blog/cto-blog/2015/11/technovision-2016-virtual-lego

Further Reading

From Big to Small, Capgemini 2005, http://www.nlondon.bcs.org/pres/am2jan05.pdf

Service-Oriented Enterprise: How To Make Your Business Fast, Flexible and Responsive, Capgemini 2005, http://www.cio.co.uk/cmsdata/whitepapers/3651/soe_flexibleresponsive.pdf

The Second Machine age – How DevOps radically shifts organisations to accelerate to survive: https://www.linkedin.com/pulse/second-machine-age-how-devops-radically-shift-survive-gunnar-menzel?trk=mp-reader-card

Martin Fowler, http://martinfowler.com/articles/microservices.html

Tom Huston https://smartbear.com/learn/api-design/what-are-microservices/

Andrew Harmel Law http://capgemini.github.io/architecture/microservices-reality-check/

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

For more details contact:

Gunnar Menzel
Infrastructure Services
Vice President, Chief Architect Officer
40 Holborn Viaduct, London EC1N 2PB
Gunnar.Menzel@capgemini.com

About Capgemini

With more than 180,000 people in over 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2015 global revenues of EUR 11.9 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Learn more about us at
www.capgemini.com

The information contained in this document is proprietary. ©2016 Capgemini. All rights reserved.
Rightshore® is a trademark belonging to Capgemini.

EDGE 1602.2016.04

People matter, results count.