Table of Contents

[Microservices 3](#_Toc521072789)

[Services are independently deployable 3](#_Toc521072790)

[Services are often processes that communicate over a network 4](#_Toc521072791)

[Synchronous 4](#_Toc521072792)

[Asynchronous 4](#_Toc521072793)

[Standardized Communication Architecture 5](#_Toc521072794)

[HTTP 5](#_Toc521072795)

[JSON 5](#_Toc521072796)

[SOAP?? 5](#_Toc521072797)

[REST 5](#_Toc521072798)

[Services are easy to replace 5](#_Toc521072799)

[Services are organised around capabilities 6](#_Toc521072800)

[Services can be implemented using Polyglot Programming 6](#_Toc521072801)

[Services are fine grained 6](#_Toc521072802)

[Services are built around business capabilities 6](#_Toc521072803)

[Services are designed to fail 8](#_Toc521072804)

[Natural modular structure 8](#_Toc521072805)

[Encourages infrastructure automation 8](#_Toc521072806)

[Automated Deployment & Testing 9](#_Toc521072807)

[Virtualisation 9](#_Toc521072808)

[Traditional Virtualisation 10](#_Toc521072809)

[Containers 10](#_Toc521072810)

[Docker 10](#_Toc521072811)

[Linux Containers 10](#_Toc521072812)

[Monitoring 10](#_Toc521072813)

[Adheres to principles 10](#_Toc521072814)

[Provides scalability 10](#_Toc521072815)

[Introduces the concept of you build it you own it 10](#_Toc521072816)

[Service-Orientated Architecture 10](#_Toc521072817)

[References 12](#_Toc521072818)

[Figure 1 3](#_Toc521082142)

[Figure 2 7](#_Toc521082143)

[Figure 3 8](#_Toc521082144)

[Figure 4 9](#_Toc521082145)

[Figure 5 11](#_Toc521082146)

[Figure 6 12](#_Toc521082147)

# Microservices

(Cover concept describe characteristics, principles, how it can be implemented)

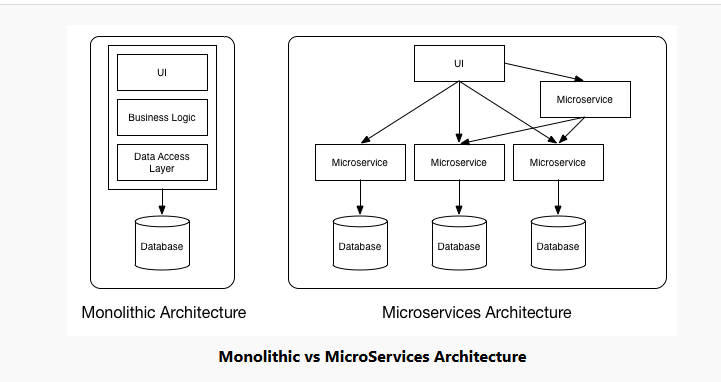
This chapter discusses the microservice architecture. It also discusses why this architecture is being preferred over the service-orientated architecture.

The term “Microservice” was first used at a software architects workshop held in Venice in May 2011 (Fowler, 2014). Participants began to realise they were using and exploring the same architectural style. It was a year later it was decided as the more appropriate name (Fowler, 2014).

As Microservices Architecture (MSA) is a relatively new architecture, there is no official industry consensus regarding the properties nor definition of MSA.

According to Sam Newman (Newman, 2015) Microservices are: “small, autonomous services that work together”. Martin Fowler (Fowler, 2014) continues this definition by saying “microservices are a way of designing software applications as suites of independently deployable services”.

Microservices help break the boundaries of large applications and create smaller systems (the services) that are used to build applications.



Figure

The implementation of MSA is open to interpretation. Though there are some defining characteristics that are commonly cited (Wikipedia, 2015) (Gupta, 2016) (Fowler, 2014):

## Services are independently deployable

Unlike traditional monolithic applications that contain several modules that may be dependent on several libraries, environment etc. Each service, within a microservice architecture, can be deployed independently. All dependencies: database, library dependencies and execution environments such as web servers or Virtual Machine (VM) are contained within each service. This ability is what enables each service to deploy independently and be essentially autonomous (Gupta, 2016). Therefore, each individual service will only contain the dependencies it needs. And can be deployed for use and perform its intended function. Even if no other service is available, the deployed services will perform their intended function, barring failure: hardware, software etc.

As services can be deployed on different machines, a distributed system of services, this can help with fault tolerance. If only a single service is on a machine that fails for whatever reason. Then it will not affect the rest of the services increasing the fault tolerance of the system or program.

Each of the independent services deployed combine to make the intended application(s). Such as an E-Commerce web application, news web application etc.

## Services are often processes that communicate over a network

(synchronous, asynchronous and any more applicable)

Each service can be deployed on different machines or they can all be deployed from a single machine. Depending on the quantity of microservices created and the hardware/software availability of businesses.

Even though no service should have a dependency on another service, it is accepted that, at times, services may need to communicate with each other to update information stored in the services’ database. This type of communication be done in numerous ways:

### Synchronous

This type of communication involves communicating with a remote server and transferring blocks of data in a continuous and consistent timed manner (Science, 1999). Primarily designed for transmission of large blocks of data, it is real-time, bi-directional communication between the client and server.

The main disadvantage of this form of communication is the remaining code to execute, from the client side, must wait for a response to this communication before the rest of the thread can execute (Microsoft, 2018). This could result in delays and high levels of Latency.

### Asynchronous

This form of communication is inherently event driven. The client does not request an initiation for things to be done. Instead the client states something has happened and then assumes the other parties involved (server side etc.) know what to do. (Newman, 2015). Asynchronous communications are inherently decoupled.

The client code sender does not wait for a response to continue with code execution. (Microsoft, 2018)

### Standardized Communication Architecture

Standardized Communication Architecture (SCA), in terms of Microservices, are used to transfer resources (object of models e.g. customer, order etc.) from client side to server side.

#### HTTP

(describe verbs?)

This type of communications is “Battle-tested” and broadly available transport protocols (Schroeder, n.d.). It is an application layer protocol for transmitting hypermedia documents like HTML (Mozilla, 2005). Designed for communication between web browsers and web servers, it follows the classic client-server model.

This architecture provides the following verbs(methods): GET, POST, PUT, DELETE. These allow requests from the client to the server involving resources.

As it is a stateless protocol, it does not keep any data(state) between requests. Each request/command is executed independently (Mozilla, 2005).

#### JSON

#### SOAP??

#### REST

Representational State Transfer (REST) is an architectural style for systems built on the web (Schroeder, n.d.). REST is used to build on protocols and standards like HTTP. It uses HTTP, and similar protocols, to do more than static web content.

Using the verbs available through HTTP, the REST architecture style prevents the need to create a multitude of different methods to do the same thing (Newman, 2015). For example, an object of a new customer created would only need to call the verb POST to request the server create a new resource and perform the request. GET would only need to be called to retrieve the representation of a resource.

REST can also make use of the large HTTP ecosystem: caching proxies, load balancers, monitoring tools, security protocols etc. These allow REST architectures to handle large volumes of HTTP traffic and route them in a transparent and fair way (Newman, 2015). And handle the security of communications from basic authentication to client certificates.

It uses the HTTP verbs to manipulates resources. This can be used to implement the CRUD (Create Read Update Delete) principle when using databases.

## Services are easy to replace

## Services are organised around capabilities

– front-end UI, logistics, billing etc.

## Services can be implemented using Polyglot Programming

Different programming languages, databases, hardware and software environments can be used, depending on what suits best

## Services are fine grained

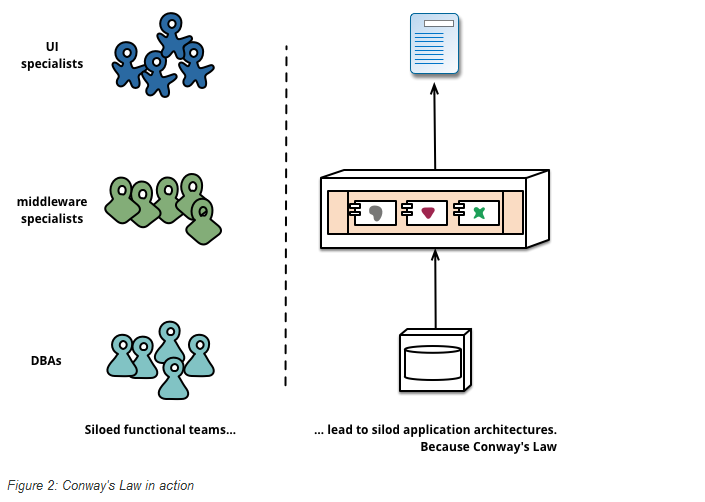
small, performing only one function –

Each service is designed to have only one responsibility, perform a single functionality of the business. This single responsibility principle also encompasses the database in use. Instead of having a single database used in a monolithic system, each microservice can have its own database to store the data. This is a change from the Monolithic style where a single database would be used to store the systems data. Although, several microservice could access the same database, again, depending on the context of the software system.

The services are described as “fine grained” referring to the granularity of the Services themselves. The actual size of the service varies depending on the context of the business. How small or fine grained a service must be is open for discussion. A general agreement is that the codebase of the service should be manageable by a small team of people. For example, Amazon has coined the phrase “the two-pizza teams” – each team numbers around 8-10 people. The number you can feed off two pizzas (Hoff, 2007). Meaning the code base for the microservice would only be “big enough” for this number of people to handle. Similarly, Jon Eaves, of RealEstate.com.au, characterizes a microservice as something that could be rewritten in two weeks (Newman, 2015).

## Services are built around business capabilities

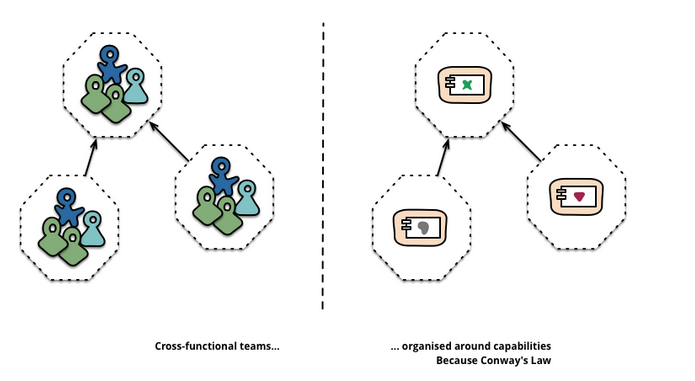
The more recognised model of focusing on the technology layer of applications results in creating different development teams for: UI, server-logic, database etc. Having such separated teams separated like this, simple changes to any aspect can result in delays of development due to cross-team communication, budgets etc. This results in an application created that follows Conway’s law: “*Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.*” (Conway, 1968)



Figure

(Fowler, 2014)

The Microservice approach changes the view of division with development teams. This approach involves organising teams around developing services based on the businesses capabilities – accounts service, basket service, payment service, catalogue service, car insurance service etc. These services implement the software needed for that business area: UI, storage, external collaborations etc. The teams become cross-functional consequently, including the full range of skills required for development: database, project management, UI etc.



Figure

(Fowler, 2014)

## Services are designed to fail

By creating services as components, applications that use these services need to be designed so they tolerate any failure of services (Fowler, 2014). This introduces another layer of complexity when designing microservices. Services can and will fail, but the effect on users must be limited.

It is important to detect failures as quickly as possible and restore them, automatically if possible. The MSA puts emphasis on real-time monitoring of the application; checking architectural (database requests per second) and business relevant (orders per minute received etc) metrics (Fowler, 2014). Monitoring can lead to early warning signs of a failure, that will allow development teams to proactively investigate.

This will likely lead to complex, and expensive, monitoring systems in place – The microservice teams need to know which services, running in different processes, have failed. This is generally achieved by having separate, sophisticated monitors and logging setups for each service available. Monitoring such things as: up/down status and operational/business relevant metrics (Fowler, 2014).

Famously, Netflix created a set of tools dubbed the “Simian army” that were developed solely to generate various kinds of failures and test the robustness of their system. (Yury Izrailevsky, 2011).

## Natural modular structure

As described previously, each service is a component of the system and, thus, enforces a modular structure where each service is seen as a module. This implementation of Modular Programming (techopedia, 2018) encompasses many of its benefits:

Each service can be used many times by many users. This benefit is adhered too as the code written, for each service, is not repeated. Only the service needed is called upon, through events etc., and executes the required task(s).

This modularity allows for each service to be used in conjunction with other applications. As service independence is a requirement of the MSA, the services can be used in the creation of multiple applications.

A disadvantage with the modular structure of the microservice architecture occurs in the debugging of the system.

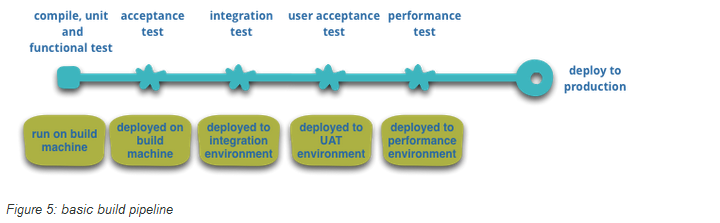
## Encourages infrastructure automation

As different businesses have different needs, the number of microservices developed can differ greatly. As the Microservice architecture adheres to the distributed systems model; microservices will be deployable on multiple servers/machines and it can become difficult, almost impossible, to develop, test, deploy, monitor each microservice manually. Regardless of the size & number of software development teams. The answer to this is to introduce automation.

### Automated Deployment & Testing

The development of the Cloud and Amazon Web Services (AWS) have reduced the operational complexity of building, deploying and operating microservices (Fowler, 2014). These infrastructure automation techniques have evolved extensively over the last few years.

Current systems built using microservices are development by teams with extensive Continuous delivery (discussed later) experience (Fowler, 2014). Software built using this principle make extensive use of the above infrastructure automation techniques. This helps to streamline and automate the testing of builds.



Figure

(Fowler, 2014)

At each stage of automated testing in the above diagram, that is successful, the build goes “up” the pipeline to the next testing environment and results in automated deployment.

Testing and deployment automation brings several advantages:

* It allows teams to choose when they want to deploy the microservice(s) they are in control of. Potentially deploying new builds multiple times in a short period of time.
* Newly developed/updated features are in the hands of customers faster
* Allows for frequent testing of new features
* Teams can experiment new processes, algorithms etc. Lokesh Gupta (Gupta, 2016) describes this as Enterprises being given the ability to experiment and fail fast.

But there are disadvantages.

A potential risk to continuous deployment/integration is “moving fast and breaking things” (Hunter, 2017). If backward compatibility is not diligently adhered too, for versioning deployments, a change can be released that results in the microservice breaking and affecting consumers.

### Virtualisation

The more distributed a system of microservices is, the greater the demand for managing hosts. Hosting can be expensive, if each host requires its own server, the costs involved can build rapidly. The ability to split up physical machines into smaller parts to accommodate many hosts has initiated an increased demand for virtualisation software. Over the last few years there has been advances in this area:

#### Traditional Virtualisation

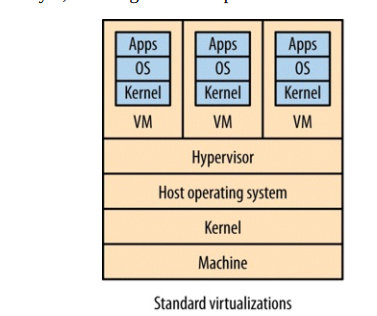
More traditional Virtualisation techniques such as VMWare or Amazon Web Services (AWS) has yielded huge benefits in reducing the overhead of host management (Newman, 2015) by using Virtual Machines (VMs). Gareth Roy describes VMs as dynamically provisioned, managed and run a variety of Operating systems, depending on user requirements (Roy, et al., 2015).

By splitting up a physical server into multiple separate hosts, each can run different things. However, a physical server can only be split up into smaller pieces, as it’s hardware allows. Increasing the number of VMs reduces the overall space available to each. A balance needs to be found for the number of VMs on each physical server, so as each host (VM) has the required hardware (CPU and memory) to perform its function correctly (Newman, 2015).

Standard Virtual machines operate on the physical structure of the server they are on. The physical infrastructure has a host operating system. Hypervisor is run on this – a type of software that allows the local host machine to operate multiple VMs as guests (Shaw, 2017). The hypervisor maps resources like CPU and memory from the virtual host to the physical host. Secondly, it acts as a control layer allowing manipulation of the VMs themselves (Newman, 2015).

The infrastructure of VMs allow them to run their own operating systems, with the Kernel. The Ms are kept isolated from the underlying physical host and the other virtual machines by the hypervisor

As seen here:



Figure

(Newman, 2015)

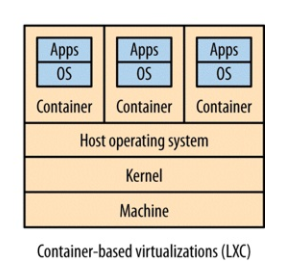
The main problem with this type of virtualisation is the resources needed by the hypervisor to perform its role. The hypervisor needs CPU, Memory, I/O etc, that could be used elsewhere. Therefore, the more hosts the hypervisor manages the more resources it requires. This can result in constraints on creating more hosts on the physical machine. This results in diminishing returns when “slicing up” a physical machine into smaller and smaller parts, as more resources goes into the overhead of the hypervisor (Newman, 2015). This is further emphasized by Stephen Soltesz et al (Soltesz, et al., 2017) by stating each VM running the same Kernel and OS, the degree of isolation offered comes at the cost of efficiency relative to all running applications on a single Kernel. Gareth Roy et al (Roy, et al., 2015) further reinforces this disadvantage by describing VM: “From a resource utilisation perspective however, VMs are considered a heavyweight solution”. Further saying: “upon instantiation a VM will make a complete copy of an operating system and all associated services leading to an increase in resource consumption when compared to “bare metal” deployment”.

#### Containers

Container based Virtualisation is described, by Gareth Roy et al, as Operating System (OS) – level virtualisation (Roy, et al., 2015). They continue this description by stating the OS creates a virtual environment (VE) consisting of a set of system resources allocated to processes, encapsulating and isolating the execution of the process(es) from the surrounding system.

Also describing the difference between containers and VMs: Containers do not need a copy of the OS to run, only the required libraries for the process to execute. This makes containers more lightweight, when deployed, compared to VMs (Roy, et al., 2015).

As seen here:



Figure

##### Docker

Docker, described by Sam Newman, is a platform built on top of light weight containers (Newman, 2015). Docker automates much of the work around handling containers. This is further emphasised by docker integration with development environments like Visual Studio using the ASP.NET core framework (Anderson, et al., 2018).

Newman further states that docker has the following benefits: manages container provisioning, handles some network problems and provides its own registry concept to store and version Docker applications (Docker applications being its version of a VM image for deployment).

There is a number of technologies currently being developed to take advantage of Docker

### Monitoring

RESEARCH!!!!

## Adheres to principles

* Domain Driven Design, fine-grained interfaces, DevOps, Polyglot programming/persistence, single responsibility

These principles will be discussed in further detail later.

## Provides scalability

## Introduces the concept of you build it you own it

Depending on the scale of the services provided by companies, there can be hundreds of microservices within a company’s software system structure. For example, Amazon, one of the first companies to embrace MSA, has over 150 services and can up to 100 of these services to build a single web page (Hoff, 2007). Services include: Search, register, account, catalogue, refine, recommendations etc.

# Service-Orientated Architecture

(Provide a basic overview – not too much detail?)

# References

Ciuffolettia, A., 2015. Automated deployment of a microservice-based monitoring infrastructure. *Procedia Computer Science,* Issue 68.

Conway, M., 1968. *How do Committees Invent?.* [Online]   
Available at: http://www.melconway.com/Home/Committees\_Paper.html

Fowler, M., 2014. *Microservices.* [Online]   
Available at: https://www.martinfowler.com/articles/microservices.html

GarethRoy-etal, 2015. Evaluation of COntainers as a virtualisation alternative. *J. Phys.: Conf. Ser.,* Issue 664 022034.

Gupta, L., 2016. *Microservices - Definition, Principles and Benefits.* [Online]   
Available at: https://howtodojava.com/microservices/microservices-definition-principles-benefits/#principles

Hoff, T., 2007. *Amazon Architecture.* [Online]   
Available at: highscalability.com/amazon-architecture

Hunter, T., 2017. *Advanced Microservices.* 1st ed. s.l.:apress.

Microsoft, 2018. *.NET-microservices for containerized NET applications.* 2nd ed. Washington: Microsoft Developer Division.

Mozilla, 2005. *MDN Web docs - HTTP.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/web/HTTP

Newman, S., 2015. *Building Microservices.* 1st ed. s.l.:O'Reilly Media.

Rouse, M., 2016. *Monolithic Architecture.* [Online]   
Available at: https://whatis.techtarget.com/definition/monolithic-architecture

Schroeder, D. A., n.d. *Microservice Architetures.* s.l., Codecentric.

Science, T. C., 1999. *synchronous and asynchronous data transmission.* [Online]   
Available at: https://teachcomputerscience.com/synchronous-and-asynchronous/

Shaw, K., 2017. *What is hypervisor.* [Online]   
Available at: https://www.networkworld.com/article/3243262/virtualization/what-is-a-hypervisor.html

techopedia, 2018. *Modular Programming.* [Online]   
Available at: https://www.techopedia.com/definition/25912/modular-programming

Wikipedia, 2015. *Microservices.* [Online]   
Available at: https://en.wikipedia.org/wiki/Microservices

Yury Izrailevsky, A. T., 2011. *The Netflix's Simian Army.* [Online]   
Available at: https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116