***Principles of microservices***

* Each microservice should only be responsible for a single function or process
* Microservices should not share code or data
* Independence and autonomy are more important than code re-usability
* Microservices are not allowed to communicate directly with each other. They should make use of an event/message bus to communicate with one another

***Microservices benefits***

* Improves modularity by making it easier to understand, develop and test the system
* Reduces complexity by having a smaller code base per microservice
* Allows you to update functionality no or minimal affect on the rest of the system
* Greatly reduces the chance of breaking something in an unrelated part of the system
* Allows for a more controlled collaboration in a team of developers that are working on the same system at the same time
* Enables continuous delivery and development of large, complex applications by applying the principle of “divide and conquer”
* Services can be deployed independently without having to wait for the entire system to be published
* It creates an architecture that is highly scalable
* Allows for deployments to multiple cloud and on-permise infra-structure environments
* Take advantage of emerging technologies (frameworks, programming languages, etc.) while evolving an existing system
* Allow new team members to become productive quicker since they can start developing new functionality without having to learn the entire system

***Anti-patterns***

* Everything should be micro except the database
* Microservices will magically solve poor development practices
* There is no need for coordination between development teams
* Making the technologies behind the microservices your key focus

***Building blocks***

* Deploymenet-viewpoint of a typical microservices architecture
* Microservices as RESTful APIs
* Client to Microservice communication through an API gatewat
* Event driven communication between microservices using an event bus
* Securing microservices

***Microservices architecture***

1. Client application the presentation layer of the architecture
2. Each microservice would typically contain the domain or business logic and database access logic required
3. API gateway allows client to microservice communication over http

***Microservices as RESTful APIs***

A RESTful API is a web API or service that is based on the architectural style known as representational state transfer.

REST defines how client application can communicate with a RESTful API over HTTP

Client request generally consists of a URI, an HTTP verb, a request header, and an optional request body

*HTTP verbs*

Post

Get

Put

Patch

Delete

***Client to Microservice communication through an API gateway***

…

***Event-Driven communication using an event bus***

What is an event bus?

Allows microservices to communicate with each other without having to know about each other.

This type of communication is based on the publish ¿/subscribe pattern, which is similar to the observer pattern.

The event bus takes up the role of the middleman and sits between the publisher and subscriber

***Securing microservices***

* *Using external authentication providers such as OAuth 2.0*
* *Adding an authentication layer in the API gateway*
* *Creating your own authentication microservice*