Microservices When to Use:

1. Scalability
2. When the business boundaries are clear.
3. High speed delivery

Principles:

1. High Cohesion Low Coupling  
   high cohesion means keeping parts of a code base that are related to each other in a single place. Low coupling, at the same time, is about separating unrelated parts of the code base as much as possible.
2. Single Responsibility Principle and Business Boundary to be defined properly
3. Failure tolerance, One microservice failure should not bring down the other applications.
4. CI/CD automation
5. Decentrailsed data as each service has its own data to have autonomous property.

https://www.techtarget.com/searchapparchitecture/tip/Follow-these-10-fundamental-microservices-design-principles

Difference between virtualisation and containerisation is that, Virtualisation creates a guest OS on top of the existing OS and then spins up a VM as a total machine. Containerisation spins up containers on the existing OS, and the containers use the existing OS resources.

IDEALS:

https://www.infoq.com/articles/microservices-design-ideals/

UML terminology

grpc

top down == function driven

how to identify microservices:

event drivem

strict bounded context

each service owns its own data

SOA - Does not consider opex, Operations, Not for cloud

microservices came into picture for cloud

SOA - coarse grained services which communicate via Enterprise Service Bus.

Microservices communicate via language agnostic protocols via network.

###############################################################################

Inversion of Control

Separate what-to-do part from when-to-do part.

Ensure that when part knows as little as possible about what part; and vice versa.

There are several techniques possible for each of these steps based on the technology/language you are using for your implementation.

The inversion part of the Inversion of Control (IoC) is the confusing thing; because inversion is the relative term. The best way to understand IoC is to forget about that word!

--

Examples

Event Handling. Event Handlers (what-to-do part) -- Raising Events (when-to-do part)

Dependency Injection. Code that constructs a dependency (what-to-do part) -- instantiating and injecting that dependency for the clients when needed, which is usually taken care of by the DI tools such as Dagger (when-to-do-part).

Interfaces. Component client (when-to-do part) -- Component Interface implementation (what-to-do part)

xUnit fixture. Setup and TearDown (what-to-do part) -- xUnit frameworks calls to Setup at the beginning and TearDown at the end (when-to-do part)

###############################################################################

Points to be followed:

1. Every fuction should have single responsibility

SOA vs Microservice

Granularity: Course grained services Fine-grained services

Ease of Deployment: Requires recreating and redeploying entire application Each service can be built and deployed independently

Remote Call Overhead: Low communication overhead High communication overhead due to an increase in remote calls

Speed of Deployment: Slow deployment speeds Rapid, continuous, and automated deployment

Persistence: All services in SOA share data storage Each service is free to choose its own data storage

Ease of On-Boarding: Semi-difficult to onboard new developers as the scope of the entire application may need to be understood Easy to onboard new developers, as there is no need to understand the scope of the entire application

Communication Method: Communicates through an enterprise service bus Communicates via API layer with lightweight protocols like REST

Scalability: Can be challenging to scale Extremely scalable through the use of containers