1.What are microservices?

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

• Highly maintainable and testable

• Loosely coupled

• Independently deployable

• Organized around business capabilities

• Owned by a small team

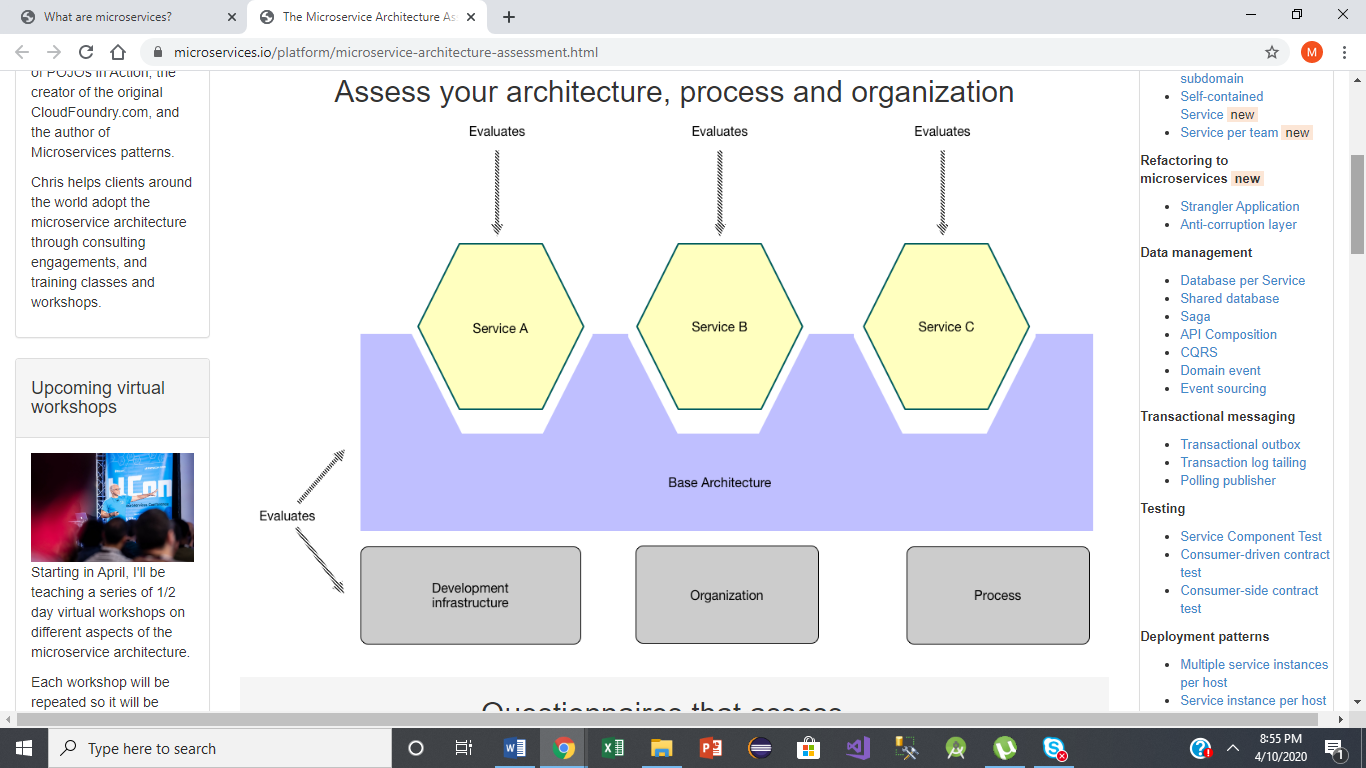
microservice apps consist of multiple independent components that are glued together with APIs.

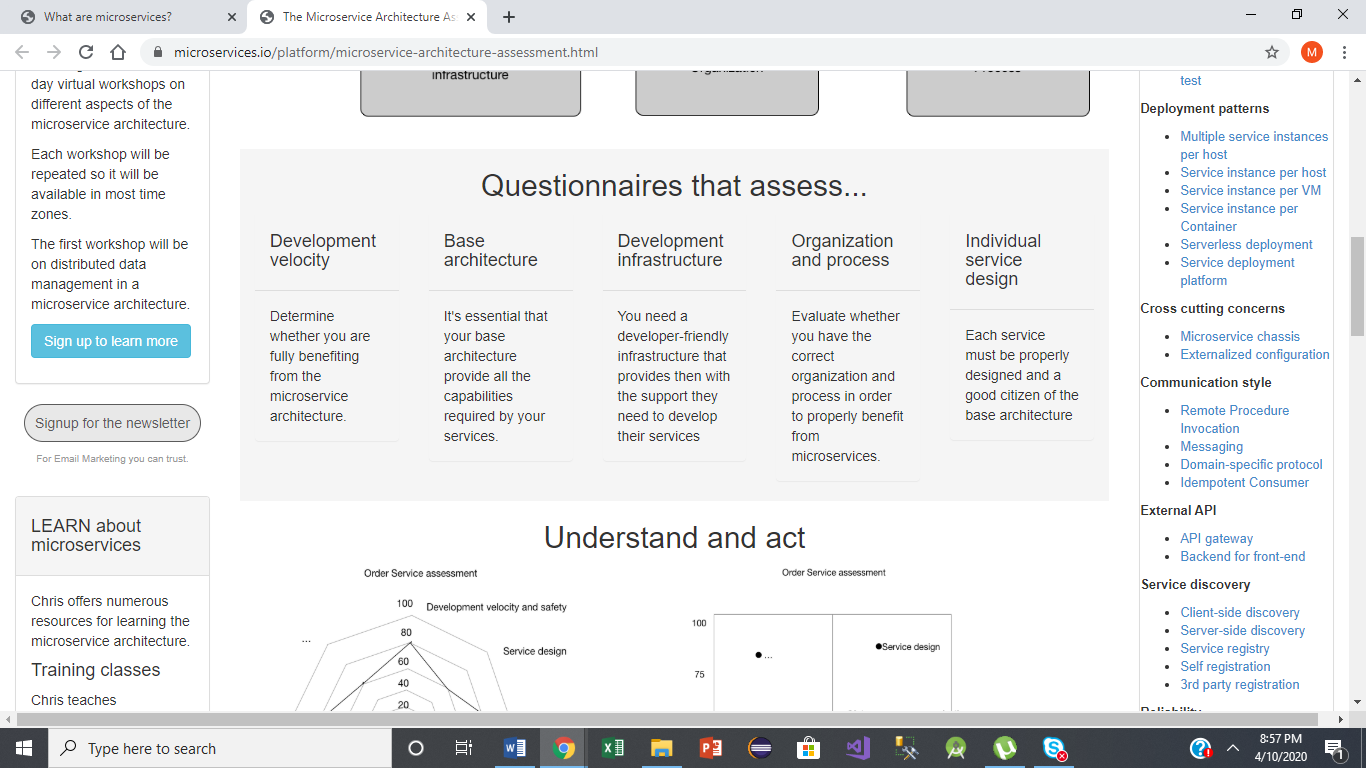
2.The pattern language is your guide

The microservice architecture pattern language is a collection of patterns for applying the microservice architecture. It has two goals:

3.Assess your architecture

If you have built an application with the microservice architecture then you must asses it in a proper way. The platform assesses what you have built and identifies what needs to be improved. It reduce architectural and organizational risk and maximizes the benefits of the microservice architecture





4.Applying the microservice architecture pattern language

The microservice architecture pattern language consists of numerous groups of patterns. The value of a pattern language exceeds the sum of it’s individual patterns because it defines these relationships between the patterns:

• Predecessor – a predecessor pattern is a pattern that motivates the need for this pattern. For example, the Microservice Architecture pattern is the predecessor to the rest of the patterns in the pattern language except the monolithic architecture pattern.

• Successor – a pattern that solves an issue that is introduced by this pattern. For example, if you apply the Microservice Architecture pattern you must then apply numerous successor patterns including service discovery patterns and the Circuit Breaker pattern.

• Alternative – a pattern that provides an alternative solution to this pattern. For example, the Monolithic Architecture pattern and the Microservice Architecture pattern are alternative ways of architecting an application. You pick one or the other. These relationships provide valuable guidance when using a pattern language. Applying a pattern creates issues that you must then address by applying successor patterns. The selection of patterns continuously recursively until you reach patterns with no successor. If two or more patterns are alternatives then you must typically pick just one. In many ways, this is similar to traversing a graph.

5.DECISIONS:

Decisions to apply architectural pattern involves 3 crirtical decions.

Decision #1: Monolithic architecture or microservice architecture?

The first decision you must make is whether to use a Monolithic architecture pattern or the Microservice architecture pattern. If you pick the Microservice architecture pattern you must choose numerous other patterns to deal with the consequences of your decision.

Decision #2: How to decompose an application into services?

If you have decided to use the microservice architecture you must define your services. There are two main options,

• Decompose by business capability – define services corresponding to business capabilities

• Decompose by subdomain – define services corresponding to DDD subdomains

This patterns yield equivalent results: a set of services organized around business concepts rather than technical concepts.

Decision #3: how to maintain data consistency and perform queries?

A key feature of the microservice is the Database per Service pattern. It’s alternative, the Shared Database pattern is essentially an anti-pattern and best avoided. The Database per service pattern dramatically changes how you maintain data consistency and perform queries. You will need to use the Saga pattern. You will often need to implement queries using the Command Query Responsibility Segregation (CQRS) pattern.

6.Creating a new product is all about risk. And choosing the right architecture

is an essential step toward success. If you’re considering between a monolithic, service-oriented, microservice, and serverless architecture, this blog post will help you make the right choice.

Monolithic architecture

Monolith is an ancient word referring to a huge single block of stone. Though this term is used broadly today, the image remains the same across fields. In

software engineering, a monolithic pattern refers to a single indivisible unit. The concept of monolithic software lies in different components of an application being combined into a single program on a single platform. Usually, a monolithic app consists of a database, client-side user interface, and server-side application. All the software’s parts are unified and all its functions are managed in one place. Let’s look at the structure of the monolithic software in detail.

