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

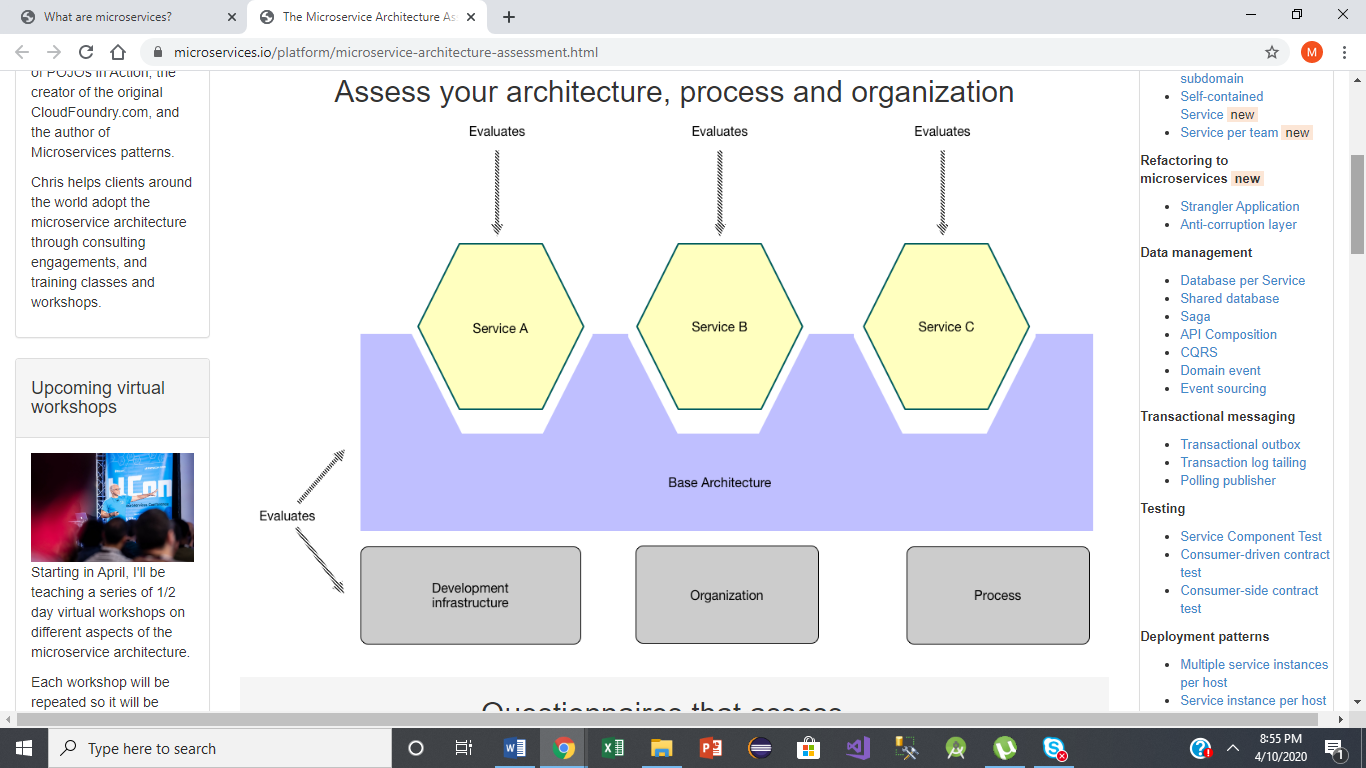
microservice apps consist of multiple independent components that are glued together with APIs.

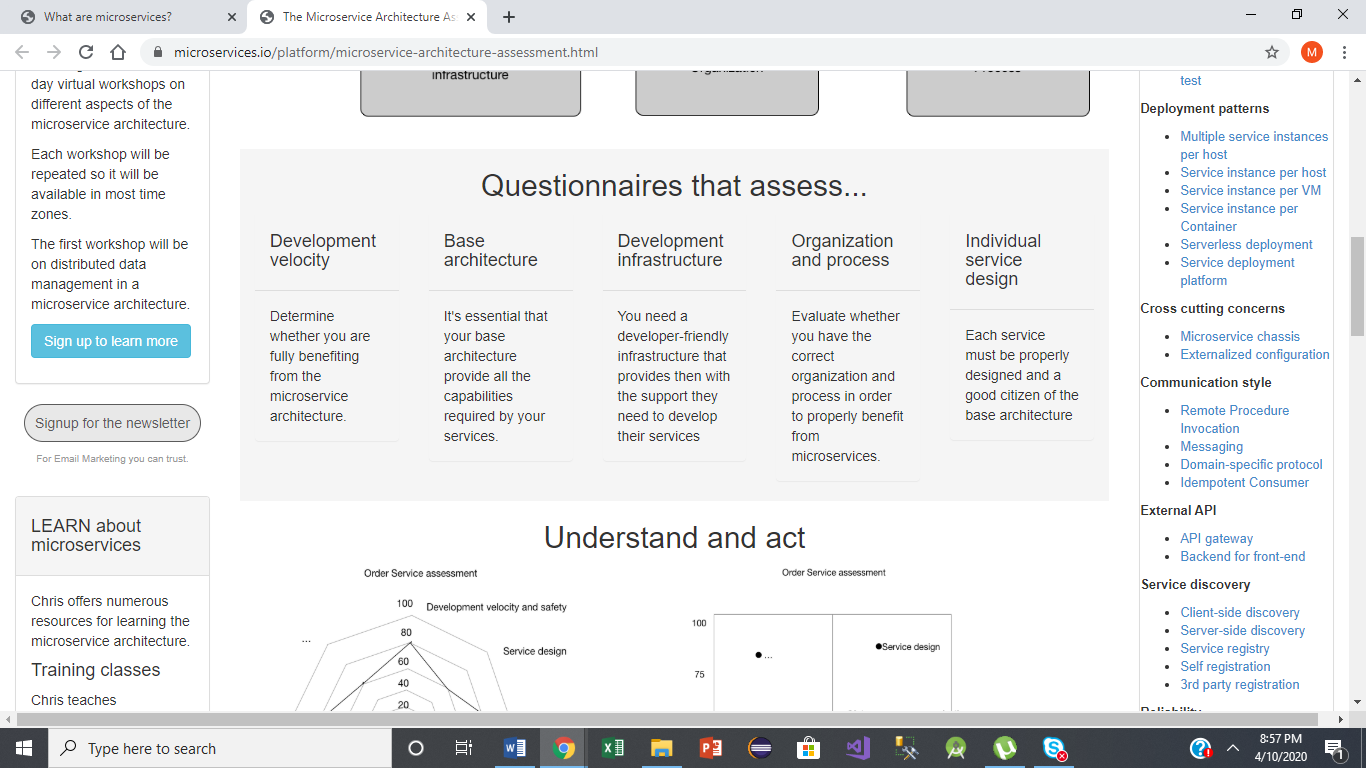
2.The pattern language is your guide

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.

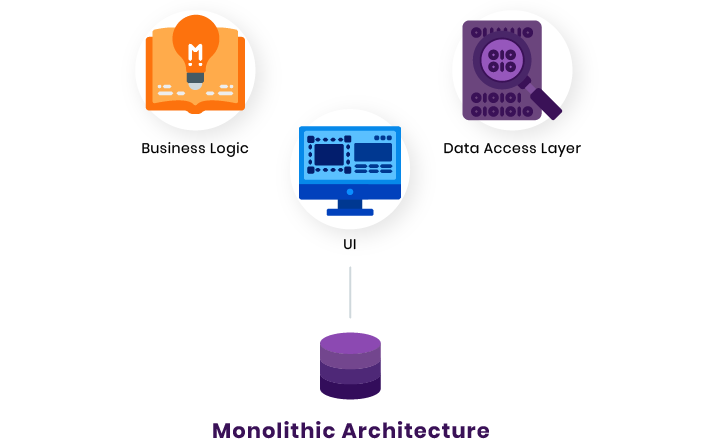
6.Creating a new product is all about risk. And choosing the right architecture

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.



7.Pros of a monolithic architecture

Simpler development and deployment

There are lots of tools you can integrate to facilitate development. In addition, all actions are performed with one directory, which provides for easier deployment. With a monolithic core, developers don’t need to deploy changes or updates separately, as they can do it at once and save lots of time.

Fewer cross-cutting concerns

Most applications are reliant on a great deal of cross-cutting concerns, such as audit trails, logging, rate limiting, etc. Monolithic apps incorporate these concerns much easier due to their single code base. It’s easier to hook up components to these concerns when everything runs in the same app.

Better performance

If built properly, monolithic apps are usually more performant than microservice-based apps. An app with a microservices architecture might need to make 40 API calls to 40 different microservices to load each screen, for example, which obviously results in slower performance. Monolithic apps, in turn, allow faster communication between software components due to shared code and memory.

8.Cons of a monolithic architecture

Codebase gets cumbersome over time

In the course of time, most products develop and increase in scope, and their structure becomes blurred. The code base starts to look really massive and becomes difficult to understand and modify, especially for new developers. It also gets harder to find side effects and dependencies. With a growing code base quality declines and the integrated development environment (IDE) gets overloaded.

Difficult to adopt new technologies

If there’s a need to add some new technology to your app, developers may face barriers to adoption. Adding new technology means rewriting the whole application, which is costly and time-consuming.

Limited agility

In monolithic apps, every small update requires a full redeployment. Thus, all developers have to wait until it’s done. When several teams are working on the same project, agility can be reduced greatly.

9. MicroService over Monolith?

There are lots of examples of companies that have evolved from a monolithic approach to microservices. Among the most prominent are Netflix, Amazon, Twitter, eBay, and PayPal. In order to determine whether microservices are suitable for your project, let’s define the pros and cons of this approach.

10.

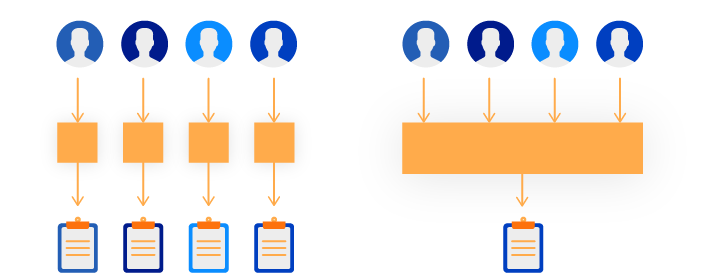
Pros of microservices

Easy to develop, test, and deploy

The biggest advantage of microservices over other architectures is that small single services can be built, tested, and deployed independently. Since a deployment unit is small, it facilitates and speeds up development and release. Besides, the release of one unit isn’t limited by the release of another unit that isn’t finished. And the last plus here is that the risks of deployment are reduced as developers deploy parts of the software, not the whole app.

Increased agility

With microservices, several teams can work on their services independently and quickly. Each individual part of an application can be built independently due to the decoupling of microservice components. For example, you may have a team of 100 people working on the whole app (like in the monolithic approach), or you can have 10 teams of 10 people developing different services for the app. Let’s imagine this visually.



Increased agility allows developers to update system components without bringing down the application. Moreover, agility provides a safer deployment process and improved uptime. New features can be added as needed without waiting for the entire app to launch.

Ability to scale horizontally

Vertical scaling (running the same software but on bigger machines) can be limited by the capacity of each service. But horizontal scaling (creating more services in the same pool) isn’t limited and can run dynamically with microservices. Furthermore, horizontal scaling can be completely automated.

Cons of microservices

Complexity

The biggest disadvantage of microservices lies in their complexity. Splitting an application into independent microservices entails more artifacts to manage. This type of architecture requires careful planning, enormous effort, team resources, and skills. The reasons for high complexity are the following:

• Increased demand for automation, as every service should be tested and monitored

• Available tools don’t work with service dependencies

• Data consistency and transaction management becomes harder as each service has a database

Security concerns

In a microservices application, each functionality that communicates externally via an API increases the chance of attacks. These attacks can happen only if proper security measurements aren’t implemented when building an app.

Different programming languages

The ability to choose different programming languages is two sides of the same coin. Using different languages make deployment more difficult. In addition, it’s harder to switch programmers between development phases when each service is written in a different language