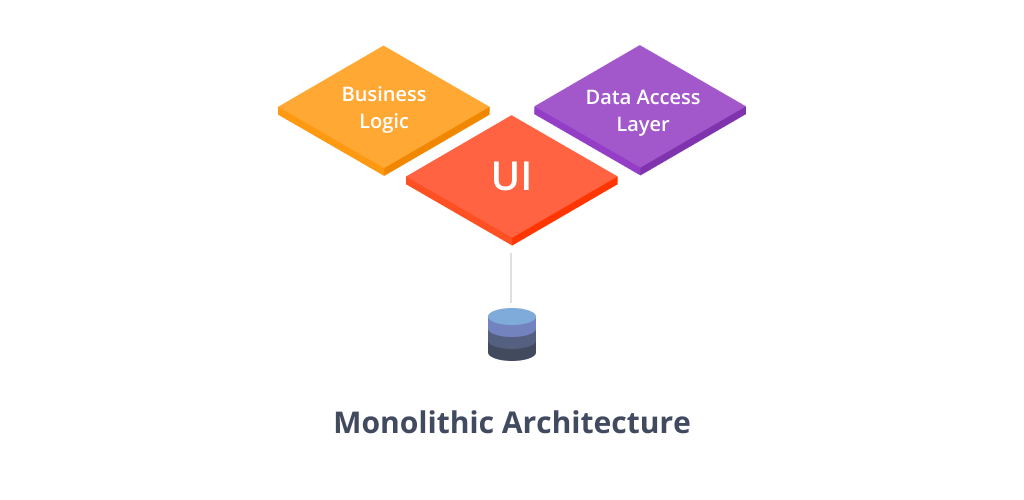
|  |
| --- |
| PIAIC IOT |
| MICROSERVICES |
| Assignment # 01 |

|  |
| --- |
| Maryam Qadeer Ahmed  4-11-2020 |

Background

There are two major types of software architecture: monolithic and microservices. The latter has become extremely popular in recent years. At the same time monoliths are still there and programmers work with them as well. It means that for many the question “monolithic vs. microservices architecture” is still topical when it comes to app development.



A monolithic architecture is a model of software structure that is created as one piece. If changes are required within one feature, it will influence the work of the entire process and other features because they are parts of one process.

The monolithic architecture has one major advantage - simplicity. Due to its simple structure there is no need to perform many complicated operations and extra activities that are required for complex systems.

Simplicity is good but there are cases when it is not enough to be completely happy with the process of app development and its results.

Let’s consider an example. As we know, our brain consists of two hemispheres. The left one is responsible for rational and logical things while the right hemisphere performs functions that are connected with creativity. They don’t delegate their tasks to each other but they have some communication channels. Moreover they work concurrently and that’s exactly what we want.

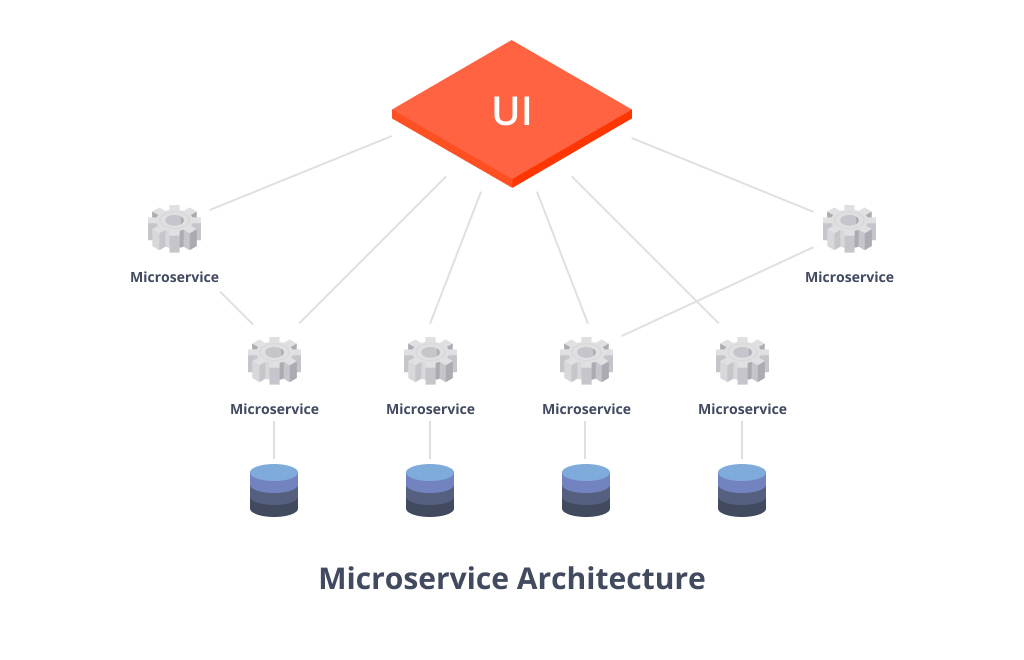
The term "micro web services" was first used by Dr. Peter Rogers during a conference on cloud computing in 2005. "Microservices" themselves premiered at an event for software architects in 2011, where the term was used to describe a style of architecture that many attendees were experimenting with at the time.

Introduction

Robert C. Martin coined the term single responsibility principle which states “gather together those things that change for the same reason, and separate those things that change for different reasons.”

A microservices architecture takes this same approach and extends it to the loosely coupled services which can be developed, deployed, and maintained independently. Each of these services is responsible for discrete task and can communicate with other services through simple APIs to solve a larger complex business problem.

Microservices appeared as an alternative to monoliths in order to solve all issues and bottlenecks caused by the limitations of the monolithic architecture.



So, what are microservices in general? These are standalone processes which serve certain purposes but they have far less responsibilities comparing to common Monoliths apps. Ideally, one microservice have to serve only one purpose.

Advantages

The advantages of microservices are as follows:

We do not just cover use cases when we apply a microservice approach. By decoupling the microservice from the main app and delegating a task to a separate service we save our application from getting potentially bigger and heavier. As a bonus, instead of one heavy process we will have several processes with a correct server load distribution.

If a part of your microservices ecosystem becomes a bottleneck, you are free to scale it horizontally, independently of other app components. All you should do is organize correct communication between these services. If you don’t need the microservice anymore, you can unplug it easily without the risk of breaking something in the main app (depends on your communication type).

All microservices require working by the black box principle. Nevertheless, you are always free to change something if needed to make sure there is no overhead.

Microservices architectural style promotes reusability. If we take care of microservices independence from the project we will receive autonomous microservices that can be used in other projects for similar tasks.

All microservices can be fully autonomous. Thus, any collapse in one of them won’t affect the work of the main app. Of course, this collapse will destabilize microservice it is responsible for, but all the rest will remain stable. Testing becomes easier when microservices are separated. You don’t have to boot the whole system to run unit tests on the microservice.

Despite so many positive sides, microservices have certain drawbacks. It's time to review them in order to understand whether they are more significant than their benefits.

Disadvantages

The disadvantages of microservices are as follows:

Harder to develop. You will have to build a pipeline by yourself: loading of libraries, connections initialization, classes ecosystem, and loading of the microservice. You will have to take care of monitoring, logging, and tracing the microservice and the whole system in general because it is unclear how it will work in the production. Moreover, you will need to provide a system that will ensure its fault-tolerance.

Take more time for development. The above mentioned disadvantage naturally results in much bigger deadlines than those that are required for customizing the tools out of the box.