**Why Microservices?**

This article presents the overview of the advantages and disadvantages of adopting a microservices architecture as comparing to the monolithic one.

Many companies have started to adopt the concept of Microservices such as Amazon, Netflix and others.

Microservices are one of the up-to-the-minute topics in the software industry, and many organizations want to adopt them. Especially helpful is the fact that DevOps can play very well with microservices.

But the question is what is Microservices? And why we need to adopt the concept of it?

To understand the Microservices we will first discuss about the monolithic software. In monolithic software we mostly used the three-tier architecture:

* Presentation layer
* Business layer
* Data access layer

Suppose a traditional web client post a request so the business layer is the one who manages database and stores the information of client’s request and the UI shows the data to the User.

Though there are numerous problems with this architecture. All code of **Presentation, Business and Data access layer** is maintained in a same code base. Although logically we divide the services like JMS Service and Data-Access Service, they are on the same code base and deployed as a single unit. Even though you created a multi-module project, there will be the high coupling means high dependency among the modules.  Although you use a distributed environment, it runs under single process context.

Say a JMS service want to use the data access layer. The JMS container needs the data access layer jars (libraries) and the jars upon which the data access layer is dependent (second level dependencies).

In this concept, there are lots of **pain points**, and the architecture is very inflexible in nature.

Here are some of the problems you face with a monolith.

**Problem 1:**

As there is one code base and modules are dependent on each other, minimal change in one module needs to generate all artefacts and needs to deploy in each server pool in a distributed environment.

Suppose in a multi-module project that the JMS module and business module are dependent on the data access module. A simple change in the data access module means we need to re-package the JMS module and business module and deploy them in their server pool.

**Problem 2:**

In monolithic software we have a team where different developers such as backend, frontend developers and database programmers etc. They all are expert in their specific context or domain, but they have a little knowledge about other layers so whenever there will be critical problem occurs the blame game will start, and they blame each other for the damage or loss.  Not only that, but it takes additional time to decide which layer's problem it is and who needs to solve the issue.