Assignment 1

**1) What is microservices?**

Microservices are a way of breaking large software projects into loosely coupled modules, which communicate with each other through simple Application Programming Interfaces (APIs). Microservices have become increasingly popular over the past few years. They are an example of the modular architectural style, based on the philosophy of breaking large software projects into smaller, independent, and loosely coupled parts, which has gained prominence among developers for its dynamic and agile qualities in API management and execution of highly defined and discrete tasks.

**2) Challenges with monolithic oriented architecture**

In a monolithic application, the core problem is this: scaling monolithic is difficult. The resultant application ends up having a very large code base and poses challenges in regard to maintainability, deployment, and modifications.

#### **Fault Tolerence**

Monolithic applications have high module interdependency as they are tightly coupled. The different modules utilize functionality in such an intramodule manner that even a single module failure brings the system down due to the cascading effect, which is very similar to dominoes falling.

### **Scaling**

Factors such as availability of different means of communication, easy access to information, and open world markets are resulting in businesses growing rapidly and diversifying at the same time. With this rapid growth of business, there is an ever-increasing need to accommodate an increasing client base. Scaling is one of the biggest challenges that any business faces while trying to cater for an increased user base. Scalability is nothing but the capability of a system/program to handle the growth of work better. In other words, scalability is the ability of a system/program to scale.

#### **Deployment challenges**

The current application also has deployment challenges. It is designed as a monolithic application, and any change in the order module would require the entire application to be deployed again. This is time-consuming and the whole cycle will have to be repeated with every change. This means this could be a frequent cycle. Scaling could only be a distant dream in such a scenario

#### **Organizational alignment**

In a monolithic application, having a large code base is not the only challenge that you'll face. Having a large team to handle such a code base is one more problem that will affect the growth of the business and application.

#### **Modularity**

In respect to our monolithic application, where we may have an Order module, a change in the module Orders affects the module Stock and so on. It is the absence of modularity that has resulted in such a condition.

This also means that we can't reuse the functionality of a module within another module. The code is not decomposed into structured pieces, which could be reused to save time and effort. There is no segregation within the code modules, and hence, no common code is available.

#### **Big database**

Our current application has a mammoth database containing a single schema with plenty of indexes. This structure poses a challenging job when it comes down to fine-tuning the performance

**3) Any three advantage and disadvantage of microservices**

### Advantages of microservices

The advantages of microservices seem strong enough to have convinced some big enterprise players such as Amazon, Netflix, and eBay to adopt the methodology. Compared to more monolithic design structures, microservices offer:

* **Improved fault isolation**: Larger applications can remain mostly unaffected by the failure of a single module.
* **Eliminate vendor or technology lock-in**: Microservices provide the flexibility to try out a new technology stack on an individual service as needed. There won’t be as many dependency concerns and rolling back changes becomes much easier. With less code in play, there is more flexibility.
* **Ease of understanding:** With added simplicity, developers can better understand the functionality of a service.
* **Smaller and faster deployments**: Smaller codebases and scope = quicker deployments, which also allow you to start to explore the benefits of Continuous Deployment.
* **Scalability**: Since your services are separate, you can more easily scale the most needed ones at the appropriate times, as opposed to the whole application. When done correctly, this can impact cost savings.

### Disadvantages of microservices

Microservices may be a hot trend, but the architecture does have drawbacks. In general, the main negative of microservices is the complexity that any distributed system has.

Here’s a list of some potential pain areas and other cons associated with microservices designs:

* **Communication between services is complex**: Since everything is now an independent service, you have to carefully handle requests travelling between your modules. In one such scenario, developers may be forced to write extra code to avoid disruption. Over time, complications will arise when remote calls experience latency.
* **More services equals more resources**: Multiple databases and transaction management can be painful.
* **Global testing is difficult**: Testing a microservices-based application can be cumbersome. In a monolithic approach, we would just need to launch our WAR on an application server and ensure its connectivity with the underlying database. With microservices, each dependent service needs to be confirmed before testing can occur.
* **Debugging problems can be harder**: Each service has its own set of logs to go through. Log, logs, and more logs.
* **Deployment challengers**: The product may need coordination among multiple services, which may not be as straightforward as deploying a WAR in a container.
* **Large vs small product companies**: Microservices are great for large companies, but can be slower to implement and too complicated for small companies who need to create and iterate quickly, and don’t want to get bogged down in complex orchestration.