**1) What is microservices?**

Microservices architecture is an approach in which a single application is composed of many loosely coupled and independently deployable smaller services.

Microservices (or microservices architecture) are a [cloud native](https://www.ibm.com/cloud/learn/cloud-native) architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components, or services. These services typically

* have their own stack, inclusive of the database and data model;
* communicate with one another over a combination of [REST APIs](https://www.ibm.com/cloud/learn/rest-apis), event streaming, and [message brokers](https://www.ibm.com/cloud/learn/message-brokers); and
* are organized by business capability, with the line separating services often referred to as a bounded context.

While much of the discussion about microservices has revolved around architectural definitions and characteristics, their value can be more commonly understood through fairly simple business and organizational benefits:

* Code can be updated more easily.
* Teams can use different stacks for different components.
* Components can be scaled independently of one another, reducing the waste and cost associated with having to scale entire applications because a single feature might be facing too much load.

**2) Challenges with monolithic oriented architecture**

Enterprise Applications are often built in three main parts: a client-side user interface (rich desktop or web based), a server-side application, and a database (consisting of many tables inserted into a common, and usually relational, database management system). The server-side application receives requests, executes domain logic, retrieves and/or updates data from the database, and responds back to the client.

This server-side application is a monolith — a single logical executable. Any changes to the system involve building and deploying a new version of the server-side application. Some of the challenges with monolithic oriented architecture are as follows

* This simple approach has a limitation in size and complexity.
* Application is too large and complex to fully understand and made changes fast and correctly.
* The size of the application can slow down the start-up time.
* You must redeploy the entire application on each update.
* Impact of a change is usually not very well understood which leads to do extensive manual testing.
* Continuous deployment is difficult.
* Monolithic applications can also be difficult to scale when different modules have conflicting resource requirements.
* Another problem with monolithic applications is reliability. Bug in any module (e.g. memory leak) can potentially bring down the entire process. Moreover, since all instances of the application are identical, that bug will impact the availability of the entire application.
* Monolithic applications have a barrier to adopting new technologies. Since changes in frameworks or languages will affect an entire application it is extremely expensive in both time and cost.

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

Benefits of Microservices Architecture

* It tackles the problem of complexity by decomposing application into a set of manageable services which are much faster to develop, and much easier to understand and maintain.
* It reduces barrier of adopting new technologies since the developers are free to choose whatever technologies make sense for their service and not bounded to the choices made at the start of the project.
* Is the same service creating bottlenecks when demand spikes in the evenings? Scale out by deploying a few more instances of just that microservice. This is very different from monolithic applications, which may have very diverse requirements but must still be deployed together as a single unit.
* Microservices are just as much about team process and organization as technology. From Conway’s Law, microservices are changing how teams are structured, allowing organizations to create teams centered on specific services and giving them autonomy and responsibility in a constrained area. This approach helps the company rapidly adjust in response to fluctuating business demand, without interrupting core activities. It also makes it easier to onboard new staff quickly.
* Microservices are simpler, developers get more productive and systems can be scaled quickly and precisely, rather than in large monolithic blobs; an architecture that embraces DevOps practices.

Drawbacks of Microservices Architecture

* Microservices architecture adding a complexity to the project just by the fact that a microservices application is a distributed system. You need to choose and implement an inter-process communication mechanism based on either messaging or RPC and write code to handle partial failure and take into account other fallacies of distributed computing.
* Microservices has the partitioned database architecture. Business transactions that update multiple business entities in a microservices-based application need to update multiple databases owned by different services. Using distributed transactions is usually not an option and you end up having to use an eventual consistency-based approach, which is more challenging for developers.
* Testing a microservices application is also much more complex than in case of monolithic web application. For a similar test for a service, you would need to launch that service and any services that it depends upon (or at least configure stubs for those services)