1. **What is microservices?**

Microservices is an architectural style that structures an application as a collection of services that are.

* small and Independent
* Highly maintainable and testable
* Loosely coupled
* Independently deployable
* Organized around business capabilities
* Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

1. **Challenges with monolithic oriented architecture.**

* **Management**

It becomes too large in size with time and hence, difficult to manage.

* **Understanding**

When a monolithic application scales up, it becomes too complicated to understand. Also, a complex system of code within one application is hard to manage.

* **Making changes**

It is harder to implement changes in such a large and complex application with highly tight coupling. Any code change affects the whole system, so it has to be thoroughly coordinated. This makes the overall development process much longer. We need to redeploy the whole application even for a small change.

* **Development**

Even if a single part of the application is facing a large load/traffic, we need to deploy the instances of the whole application in multiple servers. It is very inefficient and takes up more resources unnecessarily. Hence, horizontal scaling is not feasible in monolithic applications.

* **Reliability**

It is not very reliable as a single bug in any module can bring down the whole monolithic application.

* **New technology barriers**

It is extremely problematic to apply a new technology in a monolithic application because then the entire application has to be rewritten. It is very difficult to adopt any new technology which is well suited for a particular functionality as it affects the whole application, both in terms of time and cost.

1. **Any three advantage and disadvantage of microservices.**

* **Advantages**
* **Technological Flexibility**

While monolithic architecture always left the developers looking for the "right tool for the job," a microservice architecture offers coexistence of multiple technologies under one cover. Different decoupled services can be written in multiple programming languages. Not only does this enable developers to experiment but also scale their product by adding additional features and functionalities.

* **Agility**

Because microservices are deployed independently, it's easier to manage bug fixes and feature releases. You can update a service without redeploying the entire application and roll back an update if something goes wrong. In many traditional applications, if a bug is found in one part of the application, it can block the entire release process. New features may be held up waiting for a bug fix to be integrated, tested, and published.

* **Fault isolation**

If an individual microservice becomes unavailable, it won't disrupt the entire application, as long as any upstream microservices are designed to handle faults correctly (for example, by implementing circuit breaking).

* **Data isolation**

Microservice architecture gives developers the freedom to independently develop services. It is much easier to perform schema updates, because only a single microservice is affected.

* **Disadvantages**
* **Complexity**

A microservices application has more moving parts than the equivalent monolithic application. Each service is simpler, but the entire system as a whole is more complex.

* **Communication between services is complex**

Since everything is now an independent service, you have to carefully handle requests traveling 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.

* **Development and testing**

Writing a small service that relies on other dependent services requires a different approach than a writing a traditional monolithic or layered application. Existing tools are not always designed to work with service dependencies. Refactoring across service boundaries can be difficult. It is also challenging to test service dependencies, especially when the application is evolving quickly.

**References:**

* <https://docs.microsoft.com/en-us/azure/architecture/microservices/>
* <https://microservices.io/>
* <https://dzone.com/articles/9-fundamentals-to-a-successful-microservice-design>
* <https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/>
* <https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/>
* <https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/>