Microservices are a hot topic in system design interviews. It is important to know why we use them instead of monolithic systems. The short answer is: Scalability. The detailed one would be:

Advantages:

1) The microservice architecture is easier to reason about/design for a complicated system.

2) They allow new members to train for shorter periods and have less context before touching a system.

3) Deployments are fluid and continuous for each service.

4) They allow decoupling service logic on the basis of business responsibility

5) They are more available as a single service having a bug does not bring down the entire system. This is called a single point of failure.

6) Individual services can be written in different languages.

7) The developer teams can talk to each other through API sheets instead of working on the same repository, which requires conflict resolution.

8) New services can be tested easily and individually. The testing structure is close to unit testing compared to a monolith.

Microservices are at a disadvantage to Monoliths in some cases. Monoliths are favorable when:

1) The technical/developer team is very small

2) The service is simple to think of as a whole.

3) The service requires very high efficiency, where network calls are avoided as much as possible.

4) All developers must have context of all services.

Technical Explanation

Monolith Architecture involves building a single unified application as a single unit. This architecture typically consists of a client-side user interface, a server-side application, and a database. It is a traditional model for designing software applications. In a monolithic architecture, all components of the application are tightly coupled and run as a single service. This means that if you need to update or change one part of the application, you have to redeploy the entire application.

Microservices Architecture, on the other hand, breaks down the application into a collection of smaller, interconnected services. Each service is self-contained and implements a specific business functionality. These services communicate with each other using well-defined APIs. Microservices are independently deployable, which allows for more flexibility in development, deployment, and scaling.

Technical Example:

Monolith: Imagine a web-based e-commerce application where the user interface, business logic (product catalog, user management, order processing), and database are all part of a single codebase and operate on a single platform. To update the product catalog feature, the entire application must be tested and redeployed, even if no other parts are changed.

Microservices: In contrast, consider the same e-commerce application designed as microservices. The product catalog, user management, and order processing functionalities are developed, deployed, and scaled independently. If the product catalog service needs an update or requires more resources to handle increased load, it can be done without affecting the other parts of the application.

Laymen's Explanation

Monolith is like a big factory where everything needed to create a product is under one roof. All the departments (such as design, manufacturing, shipping) work closely together but if one part of the factory needs to be updated or fixed, it might affect the whole operation, and sometimes you have to shut down everything to make changes.

Microservices are like a village of small, specialized shops. Each shop (service) does one thing really well, like one for woodworking, another for painting, and yet another for packaging. If the woodworking shop needs to upgrade its tools, it doesn't affect the painting or packaging shops. They all work independently but come together to create the final product. This way, if one part needs changes, it doesn't stop the whole village from working.

Real-World Laymen Example

Monolith: Imagine a restaurant where the same kitchen prepares all types of dishes: appetizers, main courses, desserts, etc. If the kitchen needs to be cleaned or a new dish is being introduced, it might slow down or temporarily stop the entire food preparation process.

Microservices: Now picture a food court with multiple small kitchens, each specializing in a different type of dish. If one kitchen closes for cleaning or to update its menu, the others keep serving their specialties. Customers can still enjoy meals from other kitchens without interruption, and each kitchen can update its menu or equipment without affecting the others.

Advantages and Disadvantages
Monolith:

Advantages:

Simplicity: A monolithic application is straightforward to develop, test, deploy, and scale as a single unit.
Development Speed: Early in the project lifecycle, development can be faster because everything is in one place.
Less Operational Overhead: There's no need to handle the complexities of distributed systems early on, making operations simpler.
Disadvantages:

Scalability Issues: Scaling specific functionalities of the application can be challenging since you have to scale the entire application, even if only one part requires more resources.
Flexibility: It's harder to adopt new technologies or frameworks because the entire application might need to be rewritten or significantly refactored.
Deployment Risk: Updating one part of the application requires redeploying the entire application, increasing the risk of downtime.
Microservices:

Advantages:

Scalability: Individual components can be scaled independently, allowing for more efficient use of resources.
Flexibility: Teams can choose the best technology stack for their service, enabling innovation and faster adoption of new technologies.
Resilience: Failure in one service doesn't necessarily bring down the whole system. Services can be designed to handle failures gracefully.
Faster Time to Market: Independent development and deployment of services can increase development speed for new features.
Disadvantages:

Complexity: Managing multiple services, inter-service communication, and data consistency can introduce complexity.
Operational Overhead: Requires robust infrastructure for deployment, monitoring, and logging across services.
Network Latency and Load: Inter-service communication over the network can introduce latency and load, impacting performance.

When to Use?
Monolith:

Early Stage Projects: When the project is small to medium in size, the simplicity of a monolith can speed up development and testing.
Simple Applications: For applications with a well-defined scope that are unlikely to require scaling of individual components.
Limited Resources: When the team is small or lacks experience with distributed systems, starting with a monolith can be more manageable.
Microservices:

Scalability Requirements: When different parts of the application are expected to scale at different rates.
Technological Flexibility: When you want to use different technologies or programming languages within the same application.
Large Teams: Large development teams can work more efficiently by focusing on specific services, reducing dependencies and conflicts.
Complex Applications: For applications that are expected to grow in complexity, starting with microservices can make it easier to manage and evolve the system.
Choosing between monolith and microservices should be based on the specific needs of the project, the team's expertise, and the expected scale and complexity of the application.