When Should You Pick A Microservices Architecture?

In this lesson, we will learn about the pros and cons of the Microservice Architecture & when should we pick it for our project.

We'll cover the following

Pros of Microservice Architecture

No Single Points Of Failure

Since microservices is a loosely coupled architecture, there is no single point of failure. Even if a few of the services go down, the application as a whole is still up.

Leverage the Heterogeneous Technologies

Every component interacts with each other via a *REST API Gateway interface*. The components can leverage the polyglot persistence architecture & other heterogeneous technologies together like *Java*, *Python*, *Ruby*, *NodeJS* etc.

Polyglot persistence is using multiple databases types like *SQL*, *NoSQL* together in an architecture. I'll discuss it in detail in the database lesson.

Independent & Continuous Deployments

The deployments can be independent and continuous. We can have dedicated teams for every microservice, it can be scaled independently without impacting other services.

Cons Of Microservices Architecture

Complexities In Management

Microservices is a distributed environment, where there are so many nodes running together. Managing & monitoring them gets complex.

We need to setup additional components to manage microservices such as a node manager like *Apache Zookeeper*, a *distributed tracing* service for monitoring the nodes etc.

We need more skilled resources, maybe a dedicated team to manage these services.

No Strong Consistency

Strong consistency is hard to guarantee in a distributed environment. Things are *Eventually consistent* across the nodes. And this limitation is due to the distributed design.

I'll discuss both Strong and eventual consistency in the database chapter.

When Should You Pick A Microservices Architecture?

The microservice architecture fits best for complex use cases and for apps which expect traffic to increase exponentially in future like a fancy social network application.

A typical social networking application has various components such as messaging, real-time chat, LIVE video streaming, image uploads, Like, Share feature etc.

In this scenario, I would suggest developing each component separately keeping the *Single Responsibility* and the *Separation of Concerns* principle in mind.

Writing every feature in a single codebase would take no time in becoming a mess.

So, by now, in the context of monolithic and microservices, we have gone through three approaches:

1. Picking a monolithic architecture
2. Picking a microservice architecture
3. Starting with a monolithic architecture and then later scale out into a microservice architecture.

Picking a monolithic or a microservice architecture largely depends on our use case.

I'll suggest, keep things simple, have a thorough understanding of the requirements. Get the lay of the land, build something only when you need it & keep evolving the code iteratively. This is the right way to go.