Monolithic Architecture

Monolithic software is designed to be self-contained, wherein the program's components or functions are tightly coupled rather than loosely coupled. In a monolithic architecture, each component and its associated components must all be present for code to be executed or compiled and for the software to run. In monolithic architecture-based applications, multiple components are combined into one large application. Consequently, they tend to have large codebases, which can be difficult to manage over time.

Furthermore, if one program component must be updated, other elements may also require rewriting, and the whole application must be recompiled and tested. The process can be time-consuming and may limit the agility and speed of software development teams. Despite these issues, the approach is still in use because it does offer some advantages. Also, many early applications were developed as monolithic software, so the approach cannot be completely disregarded when those applications are still in use and require updates.

Microservices Architecture

Microservices allow a large application to be separated into smaller independent parts, with each part having its own responsibility. To serve a single user request, a microservices-based application can call on many internal microservices to dial its response. Containers are a well-suited microservices architecture example, since they let you focus on developing the services without worrying about the dependencies. Modern cloud-native applications are usually built as microservices using containers.

A microservices architecture is a type of application architecture where the application is developed as a collection of services. It provides the framework to develop, deploy, and maintain microservices architecture diagrams and services independently. Within a microservices architecture, each microservice is a single service built to accommodate an application feature and handle discrete tasks. Each microservice communicates with other services through simple interfaces for the end user.

Differences between SOAP and REST

SOAP	REST
1. Representational state transfer (REST) is a set of architectural principles used for communication between applications at API level.	1. Simple object access protocol (SOAP) is an official protocol used for communication between applications at API level.
2. It is a protocol with many built-in rules.	2. It is not a protocol but rather a set of architectural principles.
3. It is used to build wed services that use different languages.	3. It is also used to build web services that use different languages.
4. It always returns XML files	4. It returns messages in a variety of formats like HTML, XML, plain text, JSON files.
5. It is heavy in nature making the applications consume more CPU, RAM.	5. It is lightweight in nature consumes less amount of resources.
6. It has in built security.	6. It doesn’t have in built security.