For many years in the past, we have been building systems and applications with functionally distinguishable aspects are all interwoven. However with the rapid development speed of science and technology, this trend changes to microservices which containing architecturally separate components. Let's take deeply a look to microservice:

Follow Martin Fowler - software developer and author: "...the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."

Microservices are:
Highly maintainable and testable
Loosely coupled
Independently deployable
Organized around business capabilities
Owned by a small team

Each microservice is a small application that has its own hexagonal architecture consisting of business logic along with various adapters. Some microservices would expose a REST, RPC or message-based API and most services consume APIs provided by other services. Other microservices might implement a web UI.

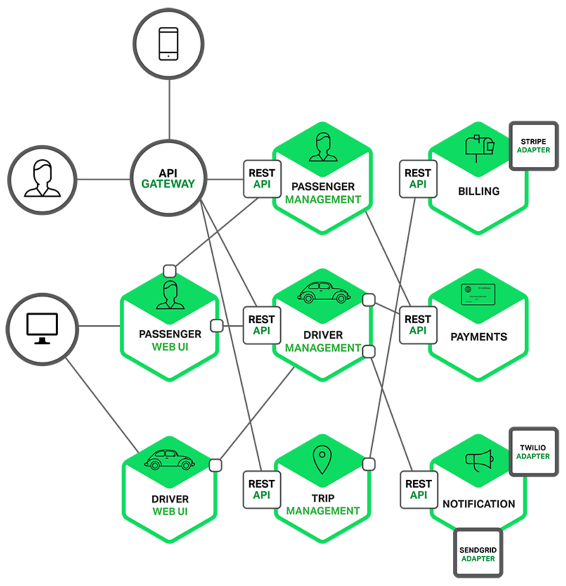
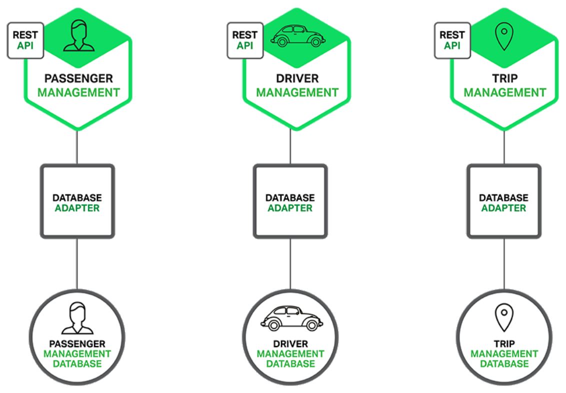
And what are microservice's benefits and disadvantages over a monolithic architecture

Benefits	Disadvantages
Independent components: - All the services can be deployed and updated independently, which gives more flexibility. - A bug in one microservice has an impact only on a particular service and does not influence the entire application. - Also, it is much easier to add new features to a microservice application than a monolithic one. Easier understanding: Split up into smaller and simpler components, a microservice application is easier to understand and manage. Flexibility in choosing the technology: It reduces barrier of adopting new technologies since the developers are free to choose whatever technologies make	Extra complexity: Since a microservices architecture is a distributed system, you have to choose and set up the connections between all the modules and databases. Also, as long as such an application includes independent services, all of them have to be deployed independently. System distribution: A microservices architecture is a complex system of multiple modules and databases so all the connections have to be handled carefully. Cross-cutting concerns: When creating a microservices application, you will have to deal with a number of cross-cutting concerns. They include externalized configuration, logging, metrics, health checks, and others.

sense for their service and not bounded to the choices made at the start of the project. Better scalability: Microservice architecture enables each service to be scaled independently.	Testing: A multitude of independently deployable components makes testing a microservices-based solution much harder.

The Microservices architecture pattern is the better choice for complex, evolving applications. Actually, the microservices approach is all about handling a complex system, but in order to do so the approach introduces its own set of complexities and implementation challenges.

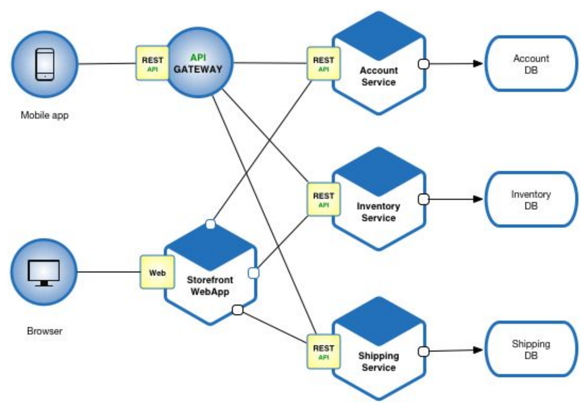
Two architectural patterns explained:

Below is an app - taxi booking service which is built based on microservice pattern.
Users and drivers access to their own web application or mobile app in handling service.
User can book, manage their trip, payments and get notification from server.

Each functional area of the application is now implemented by its own microservice. Each backend service exposes a REST API and most services consume APIs provided by other services. For example, Driver Management uses the Notification server to tell an available driver about a potential trip. The UI services invoke the other services in order to render web pages. Services might also use asynchronous, message‑based communication.
The API Gateway is responsible for tasks such as load balancing, caching, access control, API metering, and monitoring.

Furthermore, Rather than sharing a single database schema with other services, each service has its own database schema.

 Mobile app REST API API GATEWAY Browser Web Storefront WebApp REST API Account Service Account DB REST API Inventory Service Inventory DB REST API Shipping Service Shipping DB	You are building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them. The application consists of several components including the StoreFrontUI, which implements the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders. The application consists of a set of services. You apply microservice to build this app.