##### ****Microservices using ASP.NET Core****

In this growing fast-paced world, the amount of data and internet usage are proportionally increasing, and so more reliable and fast responding software systems are required, Unlike the older way of application development in Monolithic architecture which causes high maintenance cost, more downtime during upgrades made to existing monolithic architected software is not reliable. So, the Microservices Architecture of developing applications came into the picture.

Earlier software architecture build contains all business functionalities, Database calls, and UI designed in a single bundle. Like Asp.Net Webforms, MVC as a collection of single projects.

It has its disadvantages, the larger the application grows, the harder it is to quickly resolve the technical bugs/problems and to update the app with the new features. The Microservice architecture-based approach for building applications helps solve these real-time issues and provides more space for agile development methods and faster response from applications.

##### ****What are Microservices?****

Microservices are the architectural approach to build applications from small to large scale applications. With this architectural approach, an application is broken down into the smallest components, independent of each other. Unlike Monolithic architecture, where all the functionalities are targeted to build into a single project/application, Microservices helps to separate functionalities to develop in a more modular way and all modules work together to accomplish the specific targeted tasks.

##### ****Advantages of Microservices****

Microservices give development teams and testers a faster approach through distributed development. Provides the ability to develop multiple Microservices simultaneously. This means more developers working on the same app but different functional modules, at the same time, this results in a less deliverable time of application to the client. Below are some critical points you can observe positively in Microservices.

###### **Faster Development**

Since the application is designed into smaller modules development time is shortened, a microservices architecture supports a more agile development process. Multiple teams can work on each module/feature to develop and deliver separately.

###### **High scalability**

As user demand for some services in application grows, you can deploy those services across multiple servers, and infrastructures, to meet your needs. As much as the usage of services increases by end-users, easily the deployment and extending of Microservice can be handled by load balancing.

###### **Resilient & Independency**

The independent services, after properly developed and orchestrated to not impact one another. This means that one service failed to work the whole app won’t go down, unlike the monolithic application model.

###### **Easy of Deployment**

Because the microservice-based applications are more modular than traditional monolithic applications, the problems that came with application deployments and application downtimes are reduced.

###### **Accessibility for Development**

Since the larger app is broken down into smaller modules, developers can easily understand, update, and enhance the new changes, which results in fast development cycles, if planned with agile development methodologies.

###### **More open**

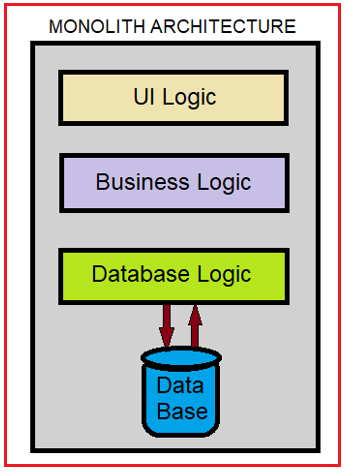
Microservices are not a particular language/technology-dependent, developers are free to choose any best language and technology. In our current article, we will discuss more the .Net technology stack.

##### ****Monolith Architecture****

First, we will understand the traditional way of building applications called Monolith Architecture, which is the widely used architectural pattern while developing applications. Still, we can see these architecture applications exist in the real world, it is designed in a way that the entire application built is ultimately a single piece, no matter how much you try to de-couple them by using Patterns and tiers – 1/2/3.

All the services or business functionalities would be tightly coupled within the single Solution. The major thing to note is that while publishing the application, you would have to deploy them to a single server only.

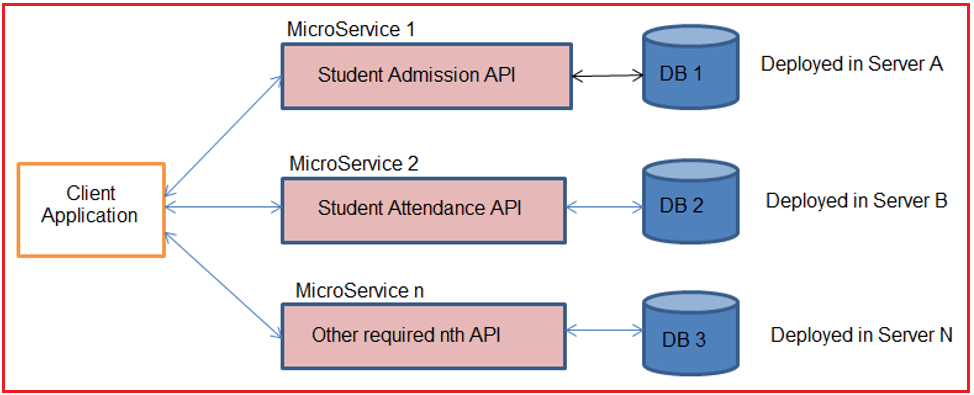
While it is still an effective way to build applications, there are some drawbacks associated with Monolith Architecture. Any small to midscale Applications would do just fine with this Architecture, but when you scale up further to extend the capabilities of existing software, you would have to make a few compromises as well as face whole application downtimes while deploying new versions/bug fixes.



From the above diagram, you can observe different logics or layers you want to integrate with your application are consolidated into a part of the single application itself. Thus, it is a big problem to maintain and add a new feature and deployment of the application in this architecture.

##### ****Microservice Architecture****

As we mentioned using Microservice architecture the application is divided into various components or modules, with each module serving a particular purpose. And these components are called Microservices all to gather. These components are not dependent on the application itself. Each of these components is truly independent in all technical manners. Because of this robust separation, you can have separately dedicated Databases for each component i.e., Microservice as well as can deploy them to separate Hosts & Servers.



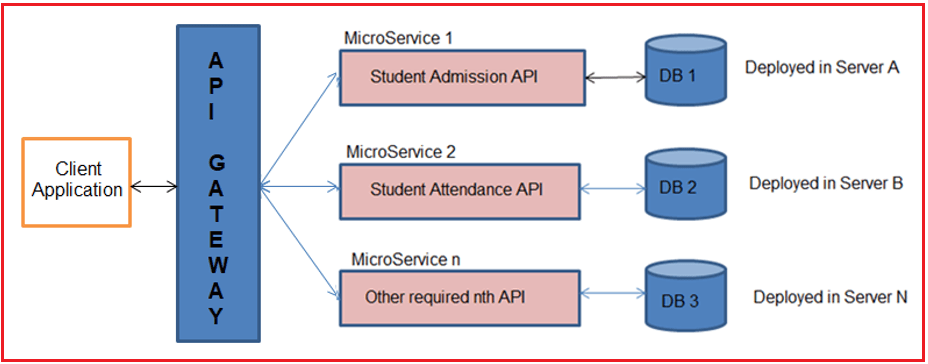
In the above picture, we can see the Client UI application either developed in .Net technology or Android or IOS or Angular, etc., will interact with our decoupled APIs called Microservices developed in .NET CORE technology, and they are deployed on multiple servers.

##### ****Microservice Vs Monolith Architecture****

|  |  |
| --- | --- |
| **Monolithic** | **Microservices** |
| Dependent & compounded project development of User Interface, Business functional Components, and Database Logics. | Multiple Services are developed with a single purpose each. |
| Single Database due to complex coupled development | Dedicated Database for each microservice due to modularity. |
| Single programming Language dependency for the Backend (Business logic) | Ability to choose multiple technologies/languages for each microservice. |
| The solution has to be deployed into a single Server. Physical Separation of deployment and maintenance is complex. | Each of the Microservice can be deployed anywhere on the web with ease. |
| Different business functionalities technically called modules or components will be tightly coupled to the application itself. | Loosely Coupled Architecture because of the modular approach. |

##### ****Understanding API Gateway (Ocelot Gateway with example)****

API Gateway is nothing but a middleware layer of directing incoming HTTP request calls from Client applications to specific Microservice without directly exposing the Microservice details to the Client and returning the responses generated from the respective Microservice.



Ocelot is an Open-Source API Gateway for the .NET/Core Platform which is officially supported by Microsoft. What it does is simple. It mimics masking multiple microservices existing behind that the client does not have to worry about the location of each and every Microservice.

Ocelot is widely used by Microsoft and other tech giants as well for Microservice Management. The latest version of ocelot supports only .NET Core applications build on the 3.1 version and above.

##### ****Features of API Gateway****

**API virtualization:**API Gateways acts as a single point entry for all the microservices configured, and avoids direct availability of microservices to clients, and hides versioning details of microservices.

**Serves as an additional layer of security microservices:**API gateways prevent malicious attacks by providing an additional layer of protection from attack vectors and hackers like SQL Injection, XML Parser exploits, and denial-of-service (DoS) attacks, and forged form data submissions.

**Decreased microservice complexity**: Authorization techniques such as JWT and other concerns of development can constitute more time for the development of each microservices. An API gateway can handle these concerns on its own and removes the development burden from your API code.