**WHAT IS MICROSERVICES AND WHY IT IS IMPORTANT IN TODAY’S IT WORLD?**

In today’s technological world, IT has become a crucial part for any business to rapidly grow and flourish. The more you are connected via networks, the more is the tendency of a business to grow. This is because for any business, if it has the larger audience to attract and most importantly the target audience, it could reach to the list of the top notch companies.

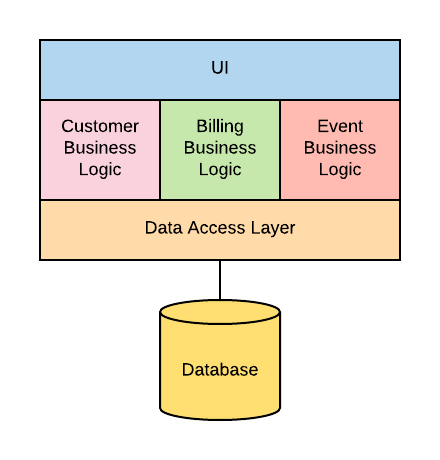
Companies that run businesses either via selling products or providing services or the ones that render services all they need is to develop their own computerized application systems where they can set up their databases, communicate with their departments via networks and provide platforms for the customers to do businesses in order to generate revenues.

**Here the importance of Microservices arises!** For any computerized application or software system, it is necessary to understand what kind of architecture it must encompass in order to fulfill its requirements. Here the architecture refers to the fundamental structure or block of a software system, how it manages its sub-systems? And how its sub-systems relate each other. In a nutshell, architecture acts a blueprint for the whole software system just like in the case of construction of a building where you first need the designs for laying out the necessary tasks.

But before going into the details of Microservices, we must first go through monolithic architectures which were the first developed in the early software development.

**Monolithic Architecture- A Predecessor to Microservices**

A monolithic architecture is a single application where all the functions are put together in a single unified package (tightly coupled) which serves as a one place to store everything. Let’s have a look at the following business model to understand this architecture:



As shown in the above image, all the components of the system resides in one single place including the User Interface layer, the business logic layer and the data access layer. Building applications in a monolith is an easy and natural process, and most projects start this way. But adding functionality to the codebase causes an increase in both the size and complexity of the monolith, and allowing a monolith to grow large comes with disadvantages over time.

**Disadvantages of Monolithic Architecture**

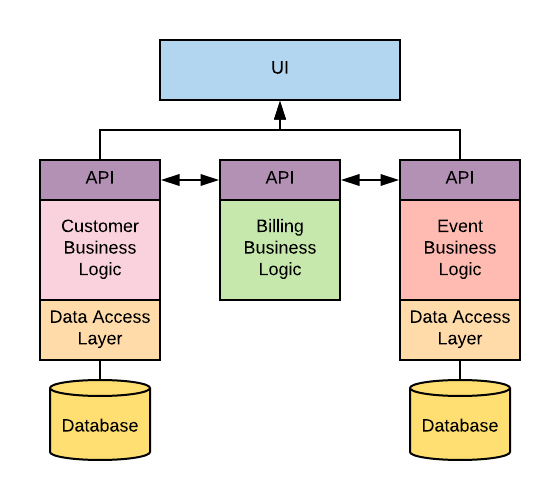
* Not suitable for big and complex applications
* Restriction of the technology stack inside the monolith. Especially as the application grows, the ability to move to a different technology stack becomes more and more difficult, even when the technology proves to no longer be the best choice
* Hard to understand, maintain and get up to speed
* Monolithic applications are difficult to scale up once they get larger. Also, different application components have different resource requirements- one might be CPU intensive while another might memory intensive. With monolithic architecture, we cannot scale each component independently.
* Restriction of bringing technology inside the monolith i.e. extremely hard to try/adopt new technologies/ architectures. Since changes in languages or frameworks affect an entire application, it requires efforts to thoroughly work with the app details, hence it is costly considering both time and efforts.

So what’s the alternative to building a monolith? Taking the monolith and breaking it up into microservices.

**Microservices**

**Microservice** is a software development technique that structures an application as a collection of loosely coupled services, where services are fine grained and protocols are lightweight.

These services interact with each other via APIs. An API is usually a portion of microservice, allowing for interaction with the microservice itself. The application architecture of Microservice would look like this:



As you can see from the above image, the business logic has been divided into separate sections each encompassing its own service called “Microservice” unlike Monolithic architecture which acts as a whole single unit. With this advent, the complexity of the application started to reduce as the different services have well defined interactions with each other. Moreover, the UI layer from before only needs to interface with the customer and event microservices, removing a dependency for the billing microservice on the UI. On the other end, the billing microservice does not need to store data, so it doesn’t have a data access layer or a database. Instead, it interacts and processes data directly from both the customer and event microservices. With these characteristics, this architecture has a definite edge over Monolithic architecture which leads to several advantages over monoliths.

**Pros of Microservice Architecture**

* Microservice architecture gives developers the freedom to independently develop and deploy services
* A microservice can be developed by a fairly small team
* Code for different services can be written in different languages (though many practitioners discourage it)
* Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)
* Easy to understand and modify for developers, thus can help a new team member become productive quickly
* The developers can make use of the latest technologies
* The code is organized around business capabilities
* Starts the web container more quickly, so the deployment is also faster
* When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
* Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system)
* Easy to scale and integrate with third-party services
* No long-term commitment to technology stack

**Cons of Microservice Architecture**

* Due to distributed deployment, testing can become complicated and tedious
* Increasing number of services can result in information barriers
* The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
* Being a distributed system, it can result in duplication of effort
* When number of services increases, integration and managing whole products can become complicated
* In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
* Developers have to put additional effort into implementing the mechanism of communication between the services
* Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams