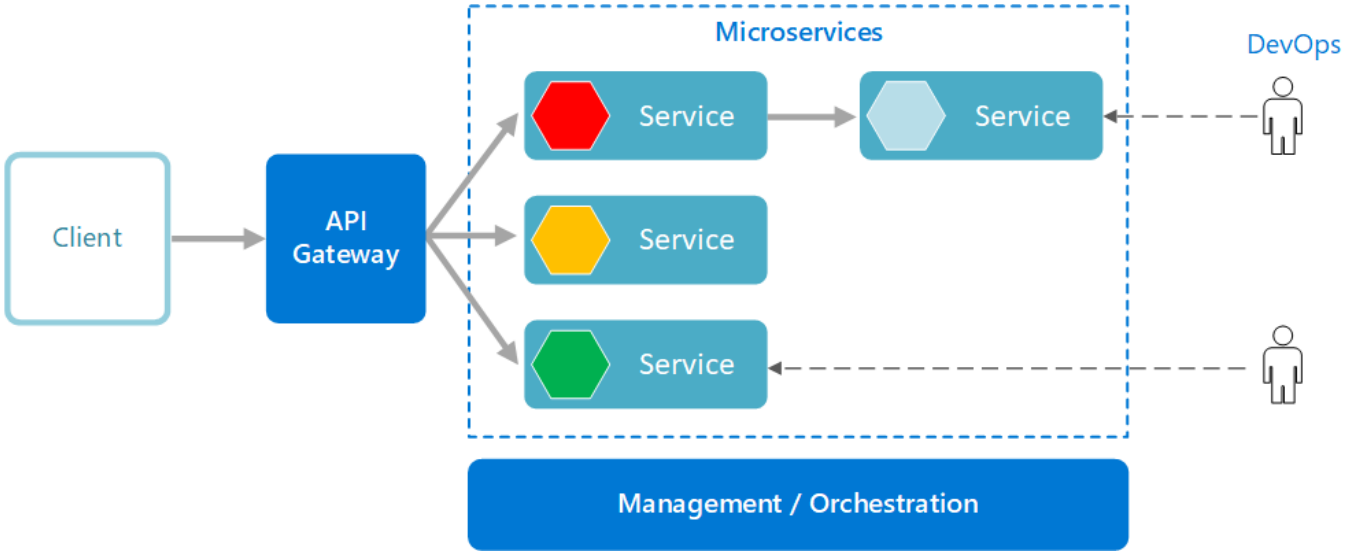
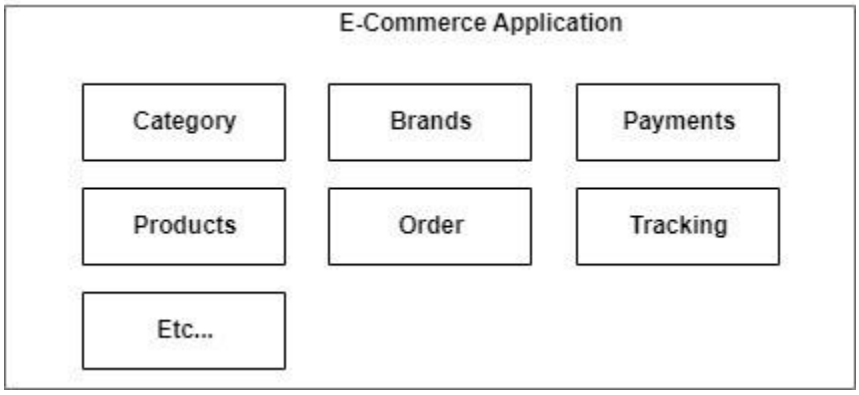
Introduction to Microservices

Before going to Microservices Architecture first, we will see the Limitation of Monolithic Limitations.



Monolithic:

developing a project with combining all modules in a single unit is known as a Monolithic application. eg: -



Here in the given image, you can clearly observe that there are different-different modules are there like Category, Brands, Payments, Products, etc. that see. All the modules are interacting with each other as per requirement.

Basically Mono (means one) and lithic application. all modules together and developed as a single deployable component is called a Monolithic application.

Advantages: 1) Monolithic applications are Easy to implement and support. 2) Design and coding can be done using a single framework (one framework). 3) Test, Debug, and maintenance are easy.

Limitations:

1) If we do some modification in one module then also, we need to full Operation to deploy the application like Re-Compile the app.

2) **Cascading Failure:** If one module is not working (Throwing exception/any other issue) entire application may be stopped/effected by the process.

[Deployment issues come].

3) Enhancements sometimes lead to design issues, as it is working on a single/one Framework.

4) Go on adding new modules, Project size (war size) increases and build, and deployment time even increased.

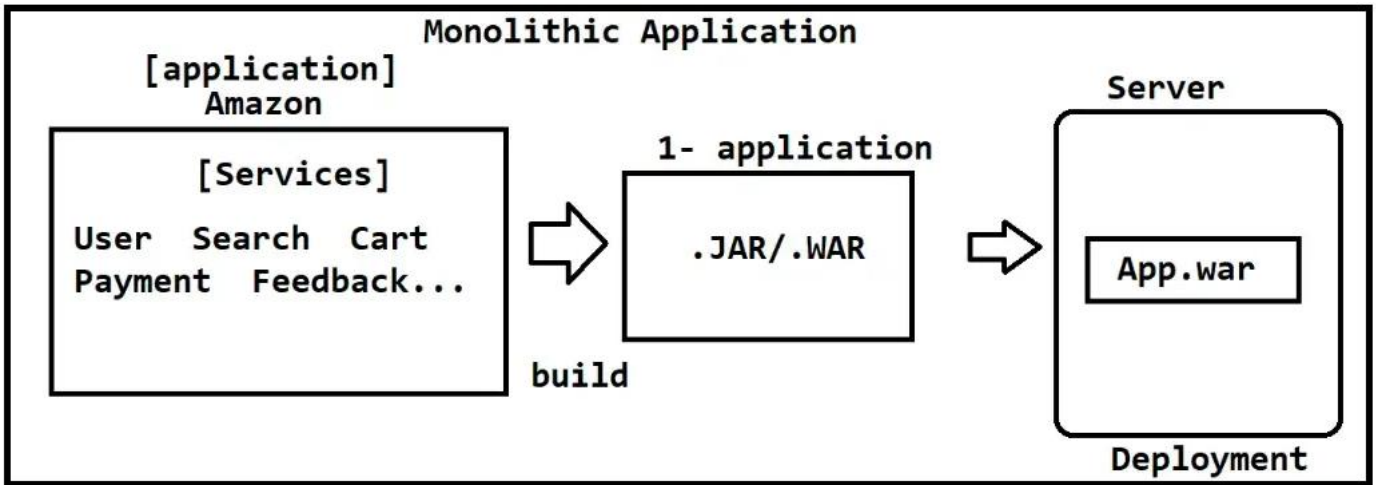
5) Creating multiple instances for load balancing leads to improper resource utilization.

6) Database Isolation properties: If the db table is in the locked state then other db operation on the same table so it leads to performance issues.

compile: .java --> .class
build : .class --> .jar/.war
deployment: place the build file in the server, starting the server.
instance: the successful deployment creates an instance.
[Application running under server]

Let's learn some application development terminologies:

Load: No. of coming requests to the server is known as a load.
Max-Load: No maximum request that a server can handle is known as Max-Load.
Load Factor: the Load factor is the current no of requests over the Max request.
eg: current load/Max-load
Note: The majority Load Factor lies between 0-1.
The default no of requests to the server is 200 but we can modify it also based on the configuration of the server.
Instance-id: A unique number that is given to an Instance is known as an Instance Id.



Scaling concept of the application:

Scaling: Scaling is the concept of enhancing the configuration of the application.
scaling is of two types:

1) Vertical Scaling
2) Horizontal Scaling

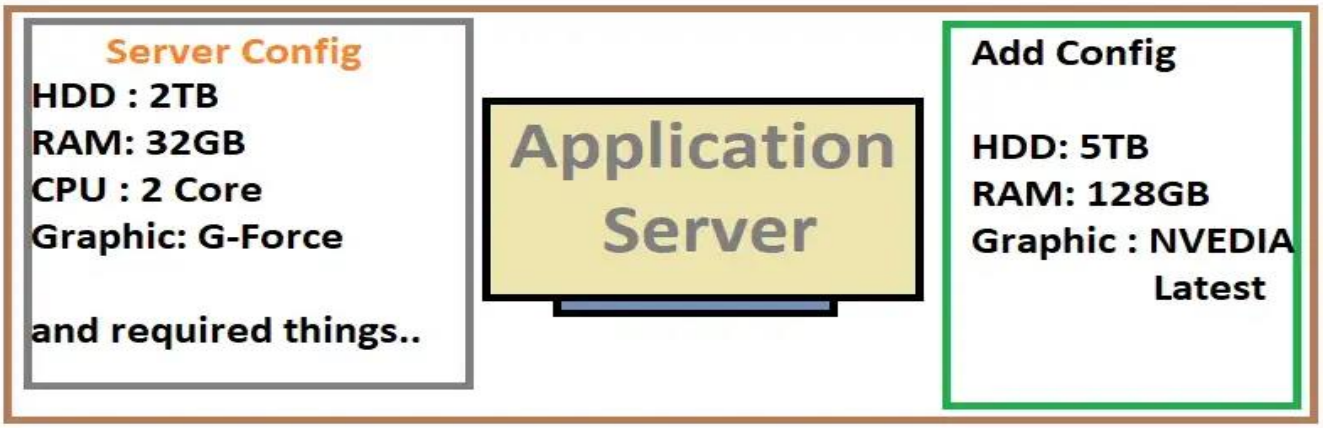
1) Vertical Scaling: Vertical scaling is nothing but enhancing the Machine configuration (Server config). So, the server-level configuration is known as vertical scaling.

eg: increasing CPU, Storage, memory, etc.

2) Horizontal Scaling: Horizontal scaling is nothing but creating multiple instances of the application based on the requirement.
(Re-Deploying the same application and Un-Deploying the application).

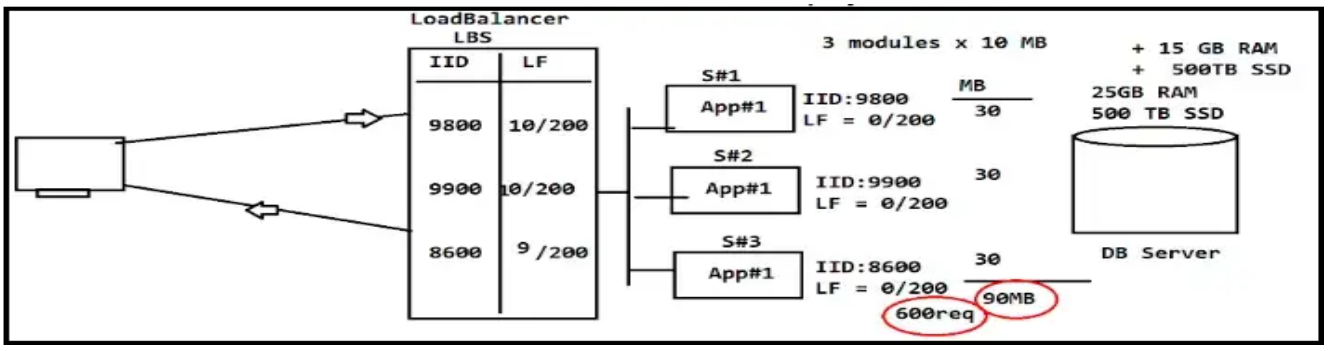
Load Balancer: Load Balancer is the component that is used to transfer the request on the same application instances means if the request came to the payment app, but the app is completely busy then it will transfer the request to the same application different instance based on Load Factor.

it means that it transfers the request to the less load factor-based instances.



Vertical Scaling

In the given picture you can clearly see that there is only a hardware level increment, so this is Vertical Scaling.



Horizontal scaling in Monolithic application

Here we can clearly see that we are using the Load Balancer to transfer the upcoming request to the different-different instances based on the Load Factor. In the Image, LF means Load Factor.

But there is a problem in the Monolithic application some of them are given above and some related to Horizontal Scaling I'm answering here-

The problem is that you can think that we can create multiple instances for handling large no. of requests. but if we do so then it will lead to addressing a problem. The problem is that it will use resources irritatedly. so, we don't create multiple instances of the Monolithic application because it is not the best practice to create multiple instances of the Monolithic application.

So now we will go on the path of the Microservices:

Microservices:

Microservices is an Architecture where each service or functionality, module is separate applications. it means that each service like Search and Category and Product is a separate application. and interacting with each other using a concept called Intercommunication.

Microservice is decoupled architecture that is used to develop large and complex, RAD (Rapid application development) applications.

Other Definitions:

1) An Independently deployable component created using Rest web services.
   It is even called De-coupled Architecture.
2) Take 1 Module/Service as one Project/Application
   Develop independently and you can have multiple instances.
3) Even supports linking with other modules/apps.

Now here we can create multiple instances of application because now we have separate applications for each service. so, if the service is receiving a huge no. of requests, then we can create multiple instances and other services don't need to create multiple instances so now we can save resource utilization also.

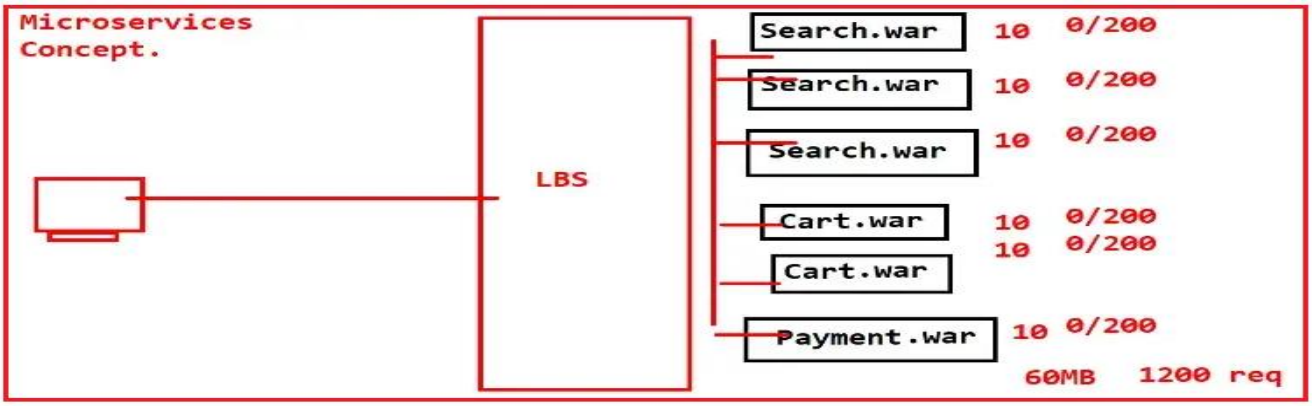
Here one application means one Microservices.

Advances:

1. If one Microservices is stopped or enhanced, then it will not affect other microservices. (so, no need to stop other applications)
2. If one microservices is not working, then it will not affect other microservices.
3. As they are independent, we can do parallel coding build and deployment.
4. Creating Multiple instances may not increase memory for all services. It is even an effective process.
5. Cascading Failure is solved.
6. Performance issue is solved in the microservices architecture.

Limitations:

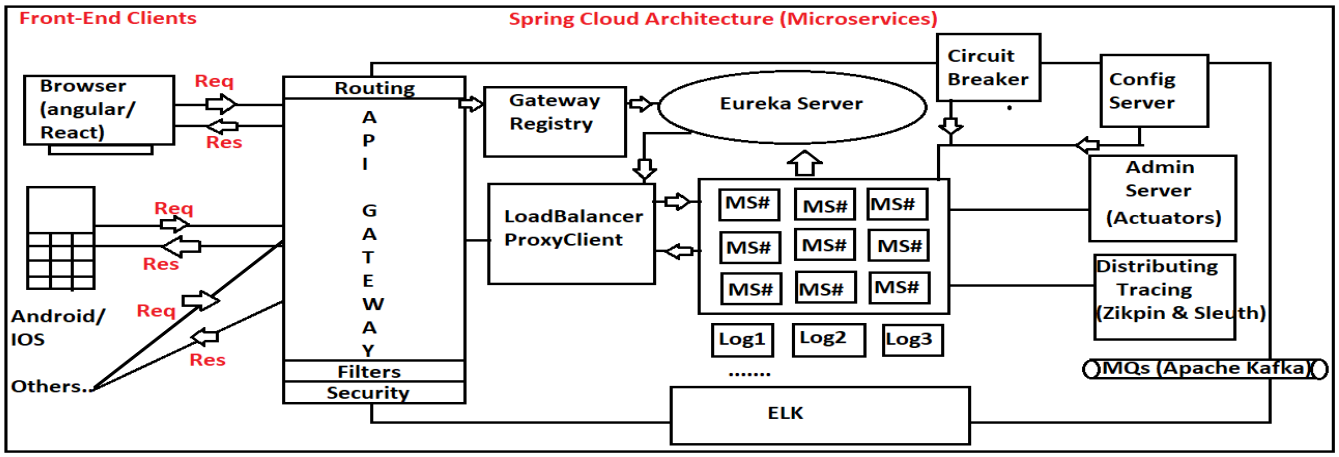
1. It will take more time to develop, and it is complex to manage them.
2. Testing and the debugging process is very complex and lengthy.
3. It is recommended to use Rest web services in each application.
4. Sometimes duplicate code may occur (Entities, Utils and etc…).



Horizontal Scaling in Microservices

Microservices is a theory every programming language has its own implementation. I'm familiar with Java, Spring boot, and Spring cloud so I'll go with these Frameworks over the Java programming language.

Microservices architecture in simple view:



Microservices Architecture

In the given above Image we can see multiple Concepts that are used in microservices application development.

1. Eureka Server: Service Register and discovery (used to search the MS applications).
2. Config Server: Config server is used to store similar common Key-Val pair properties to a central location like git.
3. Admin Server: Admin Server is used to visualize the applications actuator.
4. Distributing Tracing: ZIpkin and Slueth will be used to know the flow or trace the application flow.
5. MQs: MQs are used to achieve Asynchronous communication between the applications.
6. Circuit Breaker: A circuit Breaker is used to prevent cascading failure.
7. Load Balancer: Load Balancer is used to transfer the request to the instance of the same application.
8. API-Gateway: API-Gateway is used to Route the request and filter the request. and use SSO to achieve security.
9. Security (SSO): To Achieve the one-time login for multiple applications.
10. ELK: ELK is used to achieve Log Aggregation.