> **Need Of Microservice**

**Monolithic**: It's like a big container wherein all the software components of an application are assembled together and tightly packaged.

[Customer Service

Client Browser ---> [Product Sevice

[Car Service

\* All these services are in a single module and deployed as single instance i.e called as monolithic application.

**Problems:**

1. Large & Complex Application: As the application scales, it becomes complex.
2. Slow Development: As multiple teams working on each services it becomes difficult to understand and modify.
3. Blocks Continous Development: In order to update one component, you have to redeploy the entire application, which stops the background tasks.
4. Unscalable: We cannot scale each service individually, if customer service has to be scalled the other service has to be scalled too.
5. Unreliable: If one of the service goes down, the entire application fails.

> **Microservice**

It's an architechture style that structures an application as a collection of small autonomous services, modelled around a Business Domain

Ex: We have multiple services but all of these services, they don't share the data structure but they'll be communicating through API's.

**Advantage**: So each microservice can focus on one single business capability which leads to better quality and flow.

---> [Customer MicroService]

Client Browser ---> [Product MicroService]

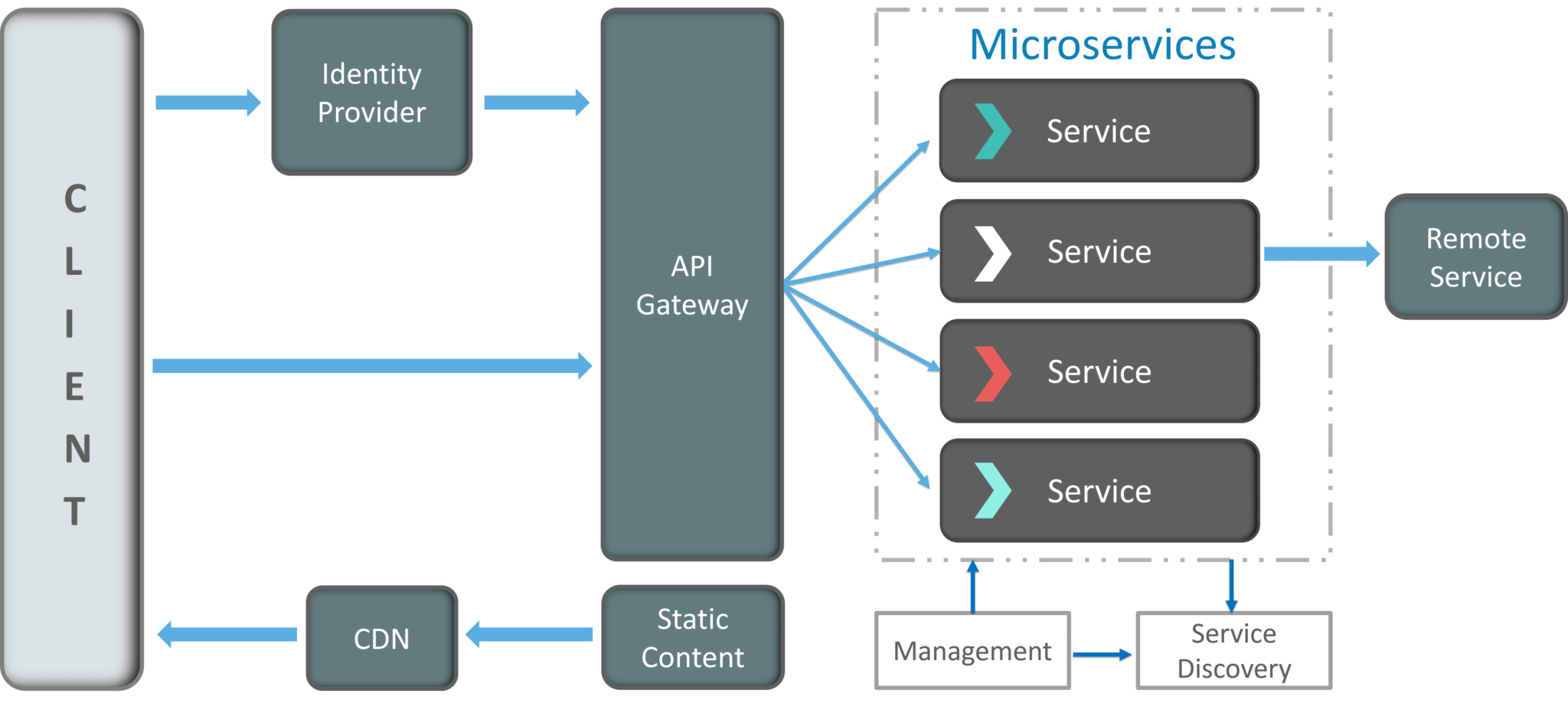
---> [Car MicroService]

\* All these services are seperate modules and are deployed at their individual instances.

So, Shop.Customer.in / Shop.Product.in / Shop.Car.in

Different instances but can communicate by shared API's.

**Architecture:**



Services are small, independent and loosely coupled.

Each of these services have separate code base which can be managed by small development team and deployed independently.

Management: Responsible for placing services on nodes, identifying failures, rebalancing services across nodes.

Service Dicovery: The task is to basically maintain a list of services and at which nodes they’re located on, It enables service look up to find the endpoint for a service

API Gateway: Entry point of client, so this client won’t call all the services directly, it’ll first go to this API gateway which will forward the cal lto the appropriate services on the backend. So It might aggregate the responses from several services and return the aggregated response.

**Features:**

* **Small Focused:** Simplicity and can be re-written and maintained without any extra effort.
* **Loosely Coupled**: Each Services are independent of each other.
* **Language Neutral:** Any services can be written in any language
* **Bounded Context**: Each Microservice doesn’t need to understand the implementation of other microservice

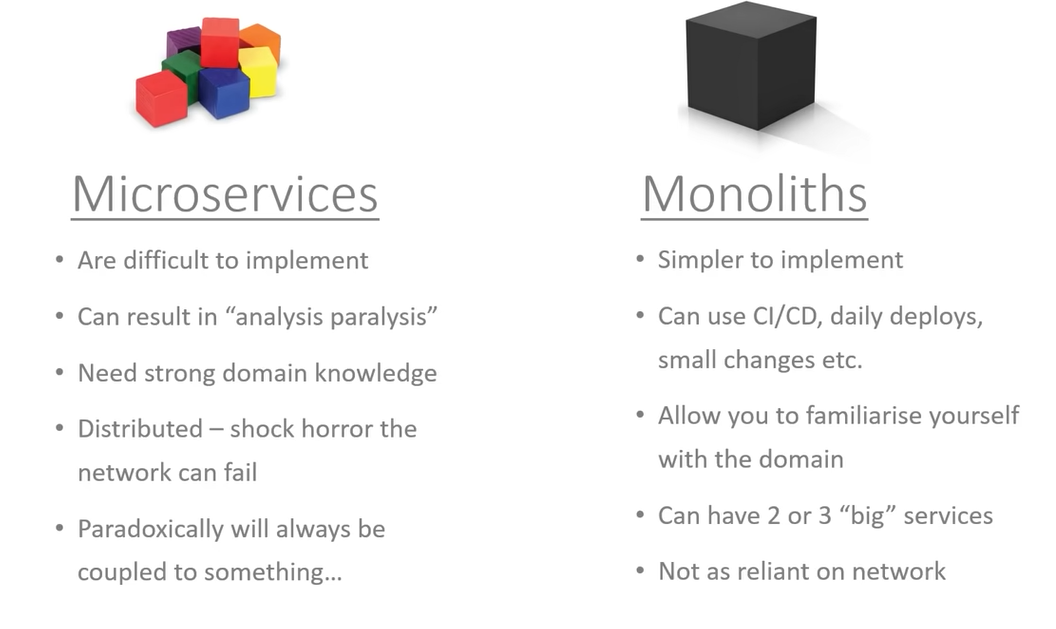
**Advantages:**

* Independent Development: Each team can focus on specific microservices.
* Independent Deployment: Any team can update/deploy the microservices at any point of time.
* Fault Isolation: If any one service goes down the other won’t be affected.
* Mixed Technology Stack: Any technology can be used for any microservice.
* Granular Scaling: Any microservices can be scalled independently.

A picture containing text

Description automatically generated

**Disadvantage:**



**Communication Between Microservices**

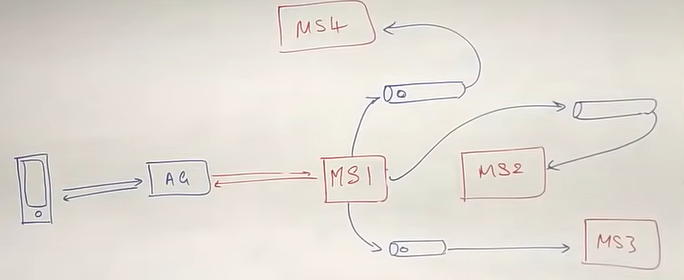
**Monolithic Communication:**

If we talk about monolithic architecture the communication between the services was happening synchronously and one service may depend on other services.

Client ------> Function -------> Microservice-1 ------> Microservice-2

Ex: If client requests for some function to be performed and it calls a microservice 1 to get the required operation but the microservice 1 is dependent on microservice 2 to complete it’s operation thus the disadvantage is that it need service availability every time the call is happening and the response time is delayed.

**Microservice Communication:**



Now since the services are independent of each other the data required by one service can be fetched from the queue/data pipelines.

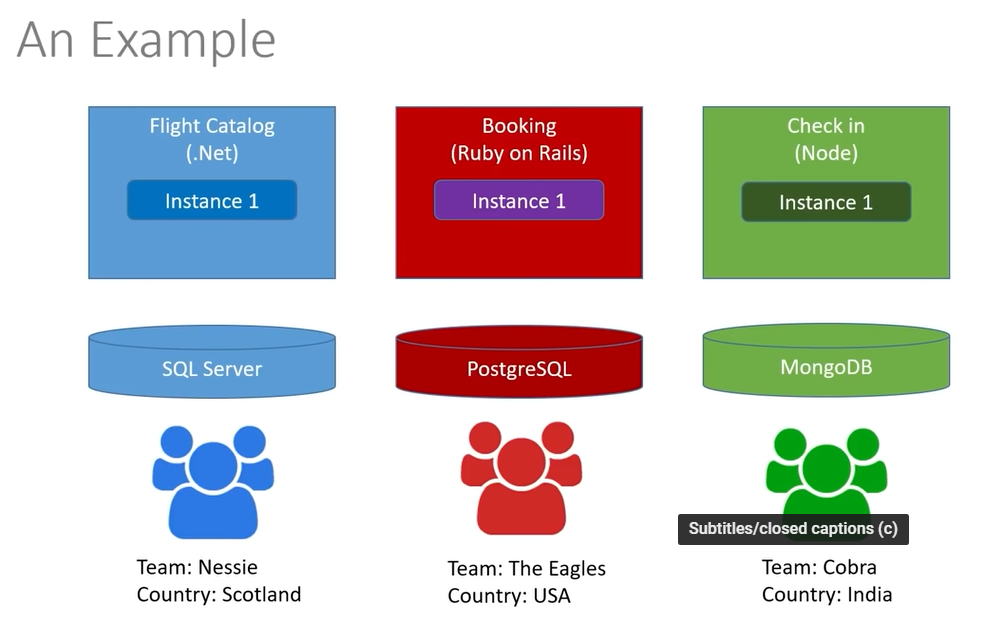
Ex: if MS1 wants to give data to all other Microservices it can give it to the queue and the other services can fetch from the queue.

Advantages:

Faster API’s: Since the call is asynchronous.

Decoupled Services: Independent services

**Microservice Example:**



Multiple Services having their own instances with different database and technology.

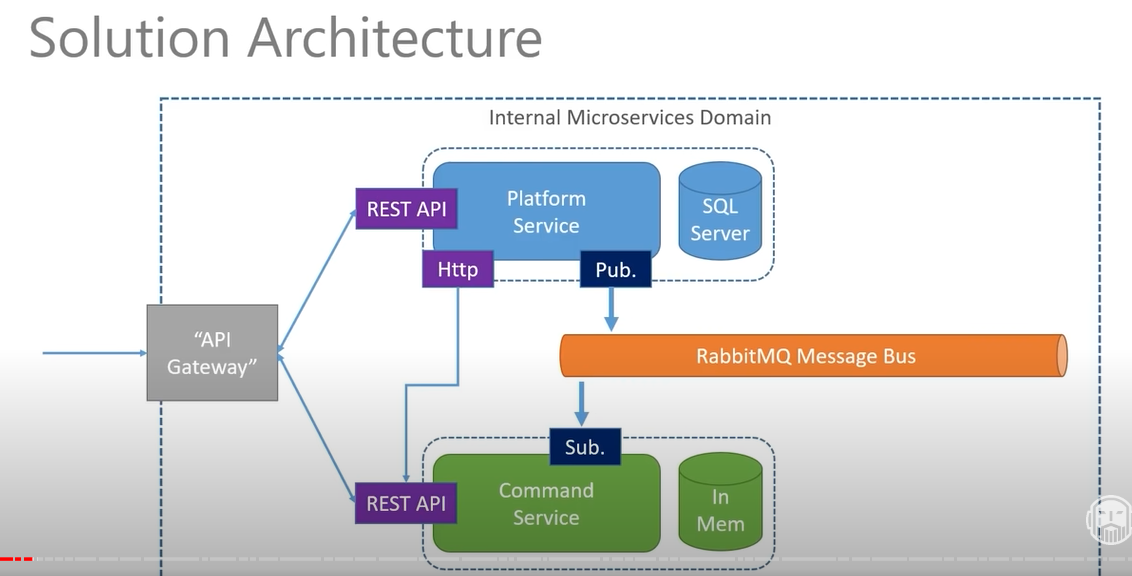
**Project Application**

First Service:

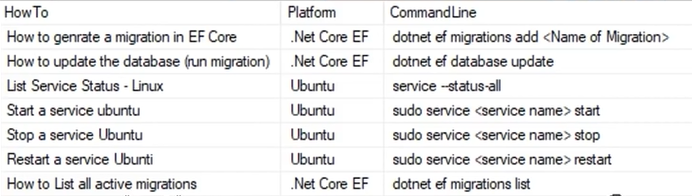


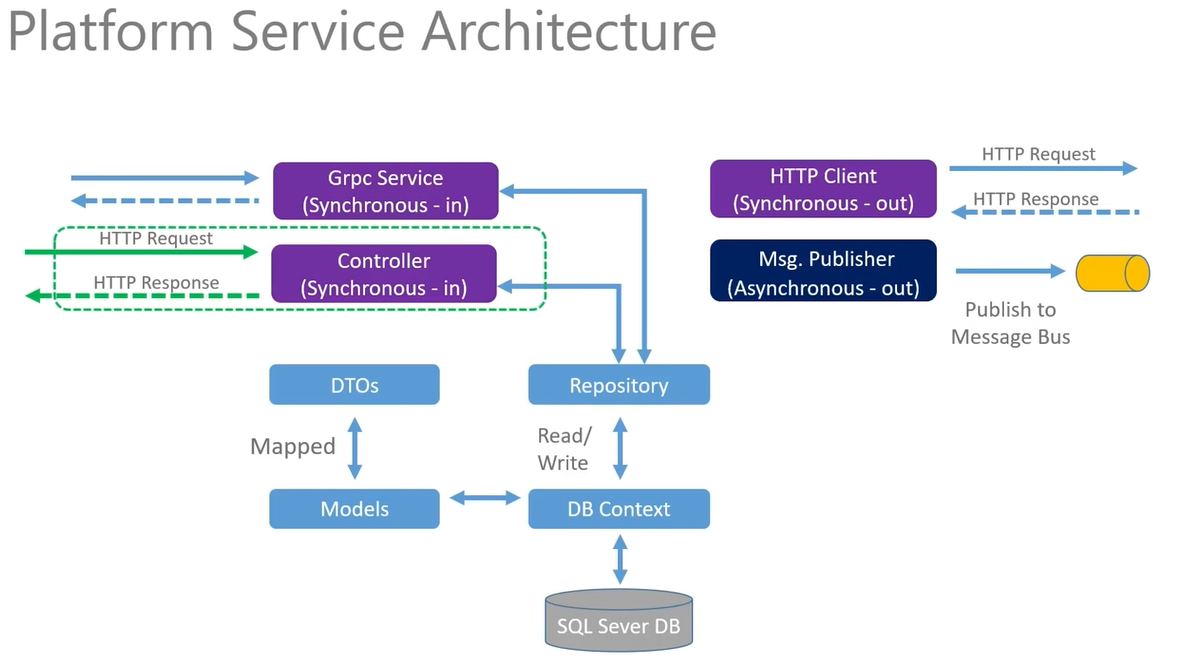
Second Service:





Command Service Database Table Sample:





Diagram

Description automatically generated