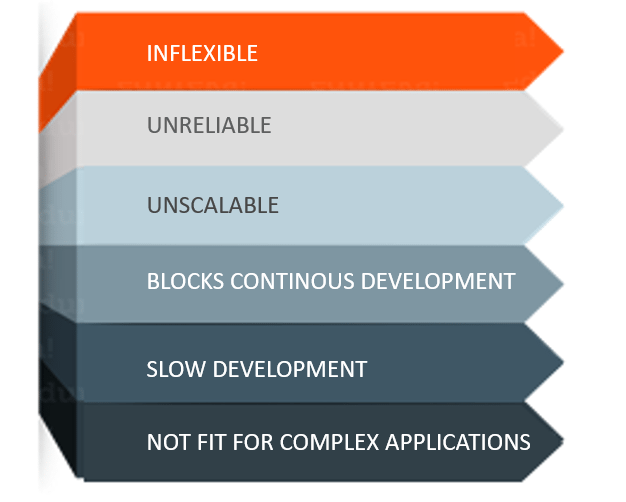
***What is Microservices?***

* To get an idea of **what is Microservices**, you have to understand how a monolithic application is decomposed into small tiny micro applications which are packaged and deployed independently.

**In this blog, you will learn about the following:**

* Why Microservices?
* What Is Microservices?
* Features Of Microservices Architecture
* Advantages Of Microservice Architecture
* Best Practices To Design Microservices
* Companies Using Microservices
* **Why Microservices?**
* Now, before I tell you about Microservices, let’s see the architecture **Monolithic Architecture.**
* In layman terms, you can say that it’s similar to a big container wherein all the software components of an application are assembled together and tightly packaged.

**Listed down are the challenges of Monolithic Architecture:**

* **Inflexible –** Monolithic applications cannot be built using different technologies
* **Unreliable –** Even if one feature of the system does not work, then the entire system does not work
* **Unscalable –** Applications cannot be scaled easily since each time the application needs to be updated, the complete system has to be rebuilt
* **Blocks Continuous Development –** Many features of the applications cannot be built and deployed at the same time
* **Slow Development –** Development in monolithic applications take lot of time to be built since each and every feature has to be built one after the other
* **Not Fit For Complex Applications –**Features of complex applications have tightly coupled dependencies

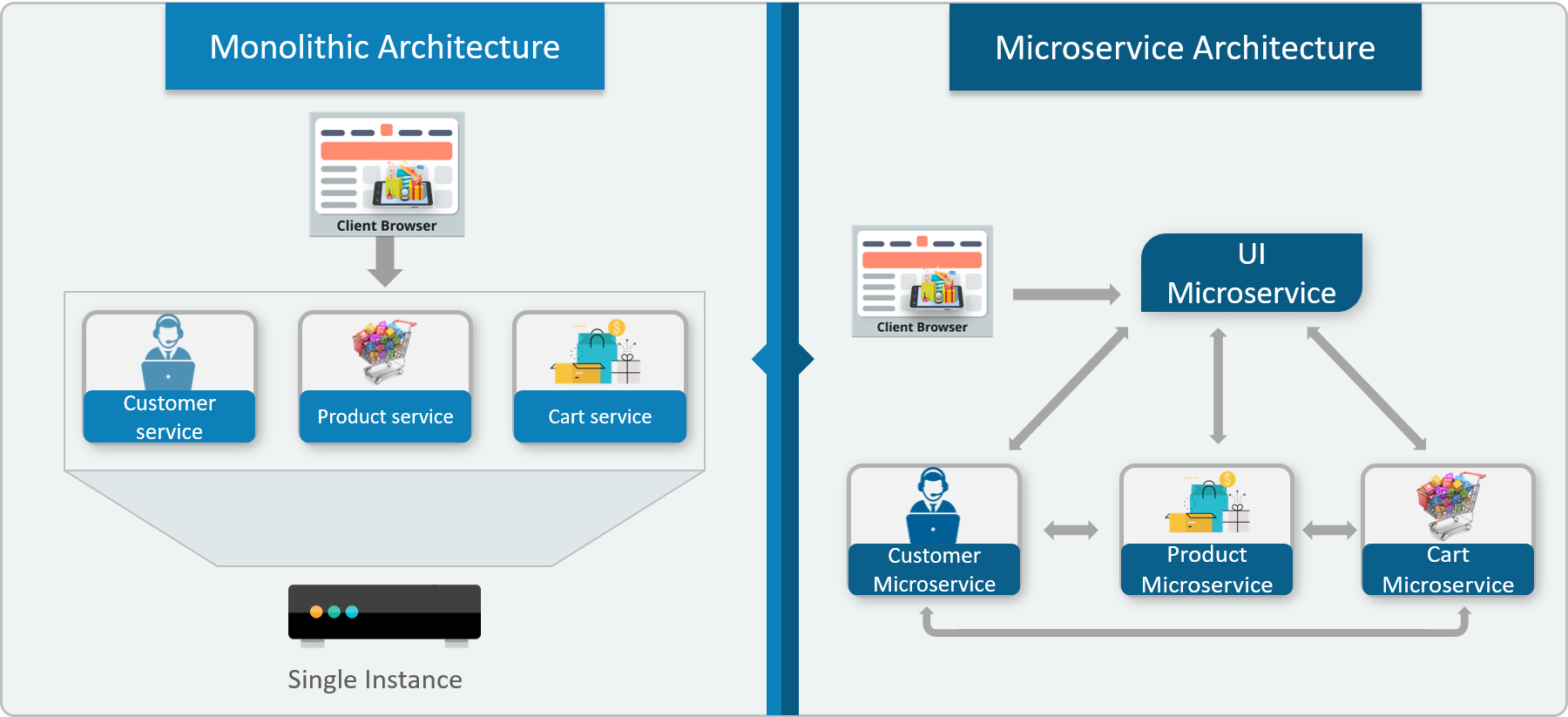
## ****What Is Microservices?****

**Microservices** is nothing but an architectural style that structures an application as a collection of small autonomous services, configured around a **business domain.**

****

In Microservice Architecture, each service is **self-contained** and implements a **single business functionality or capability.**

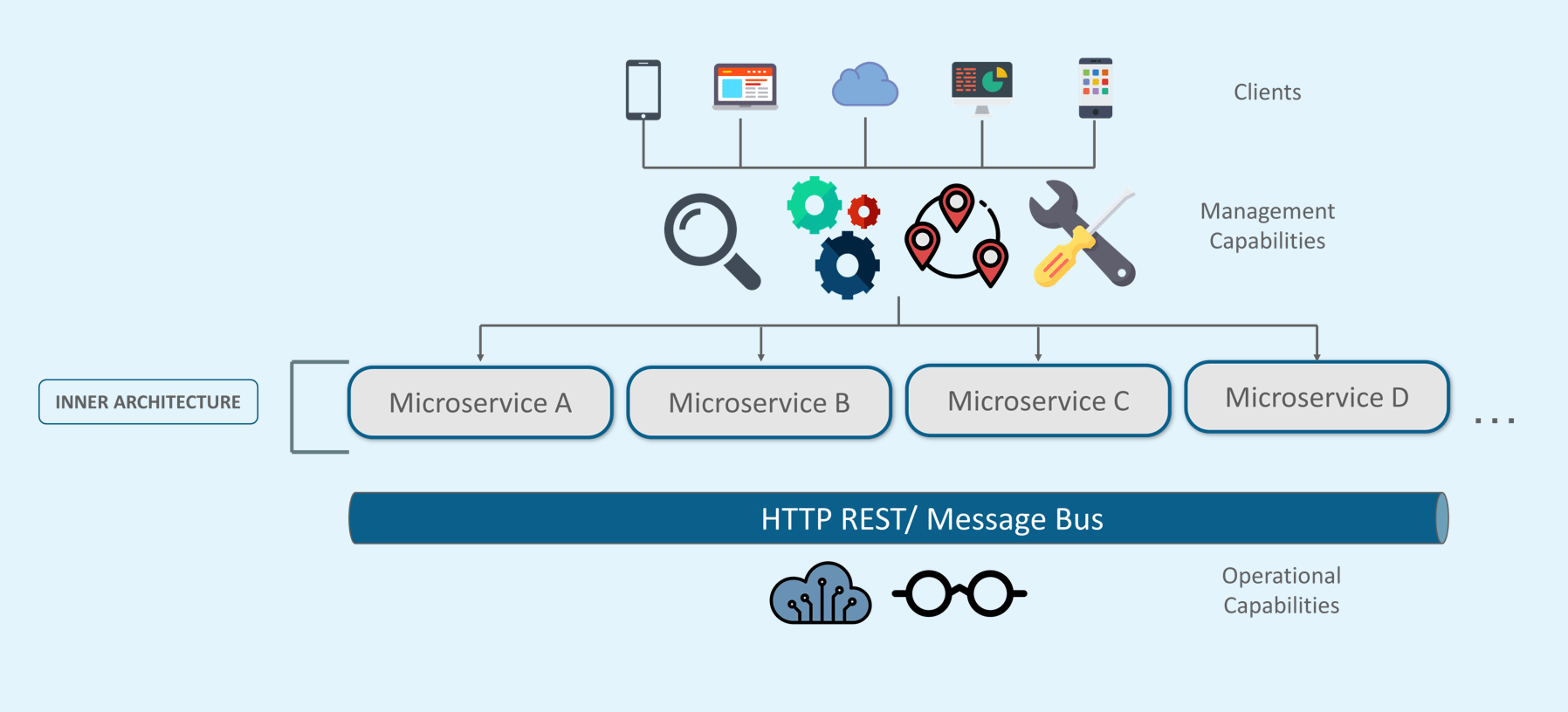
## ****Differences between Traditional Architecture and Microservices****

Consider an E-commerce application as a use-case to understand the difference between both of them.

The main difference we observe in the above diagram is that all the features initially were under a single instance sharing a single database. But then, with Microservices, each feature was allotted a different Microservice, handling their own data, and performing different functionalities.

Now, let us understand more about Microservices by looking at its architecture. Refer the diagram below:

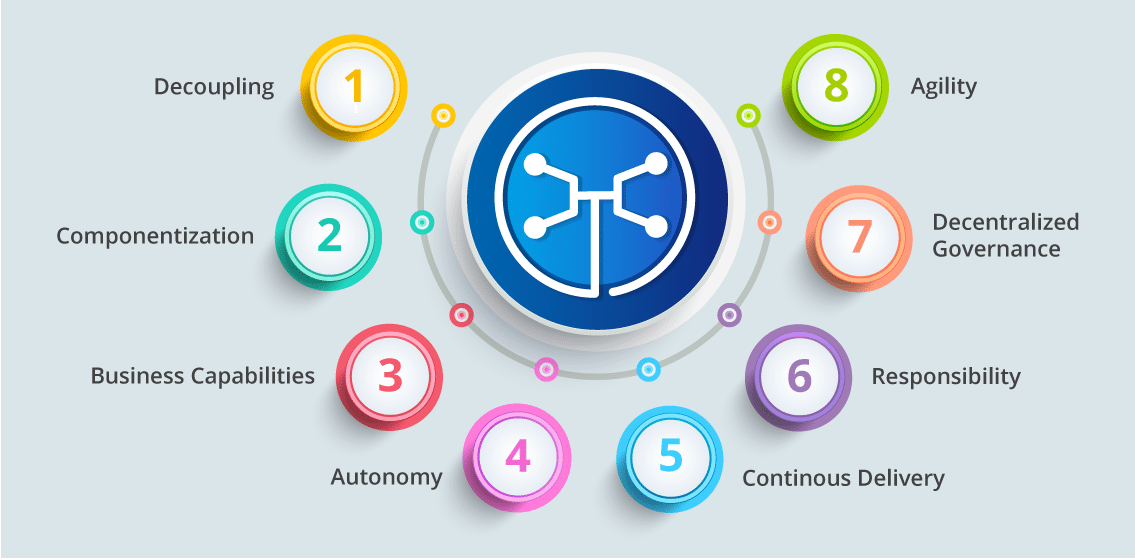
## ****Microservice Architecture****



Here in the above diagram we have fallowing.

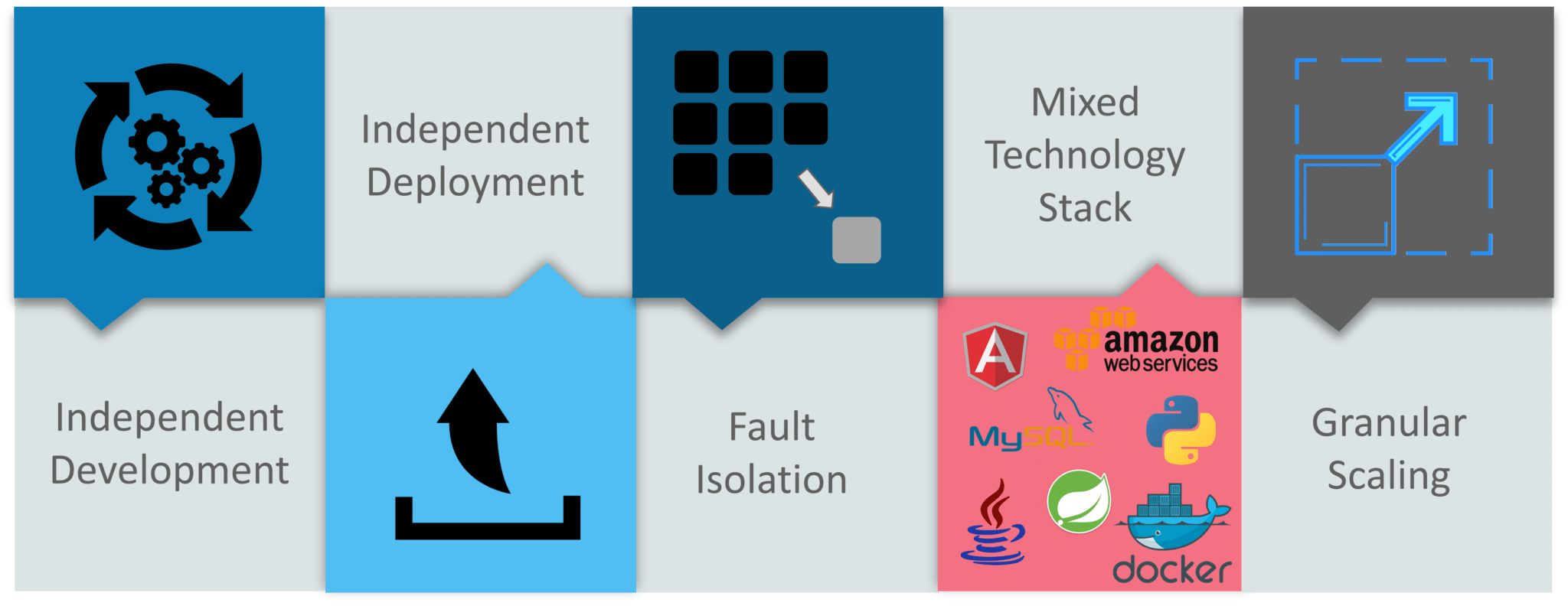
* Different clients from different devices try to use different services like search, build, configure and other management capabilities
* All the services are separated based on their domains and functionalities.
* These Microservices have their own **load balancer** and **execution environment** to execute their functionalities & at the same each Microservices have their own databases.
* All the Microservices communicate with each other through a stateless server which is either **REST** or **Message Bus**
* Microservices identifies their path of communication with the help of **Service Discovery**and perform their operation.
* Then all the functionalities performed by Microservices are communicated to clients via **API Gateway**
* All the internal points are connected from the API Gateway. So, anybody who connects to the API Gateway automatically gets connected to the complete system

**Microservices Features**



* **Decoupling** – Services within a system are largely decoupled (independent). So the application can be easily built, altered, and scaled
* **Componentization** – Microservices are treated as independent components so they can be easily replaced and upgraded
* **Business Capabilities** – Microservices are very simple and focus on a single functionality or capability.
* **Performance** – Developers and teams can work independently of each other, thus increasing speed.
* **Continuous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
* **Responsibility** – Microservices do not focus on whole applications as projects. Instead, they takes responsibility on individual functionality or capability of the application.
* **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
* **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

**Advantages of Microservices**



**Independent Development** – All Microservices can be easily developed based on their individual functionality

**Independent Deployment** – Based on their services, they can be individually deployed in any application

**Fault Isolation** – Even if one service of the application does not work, the system still continues to function

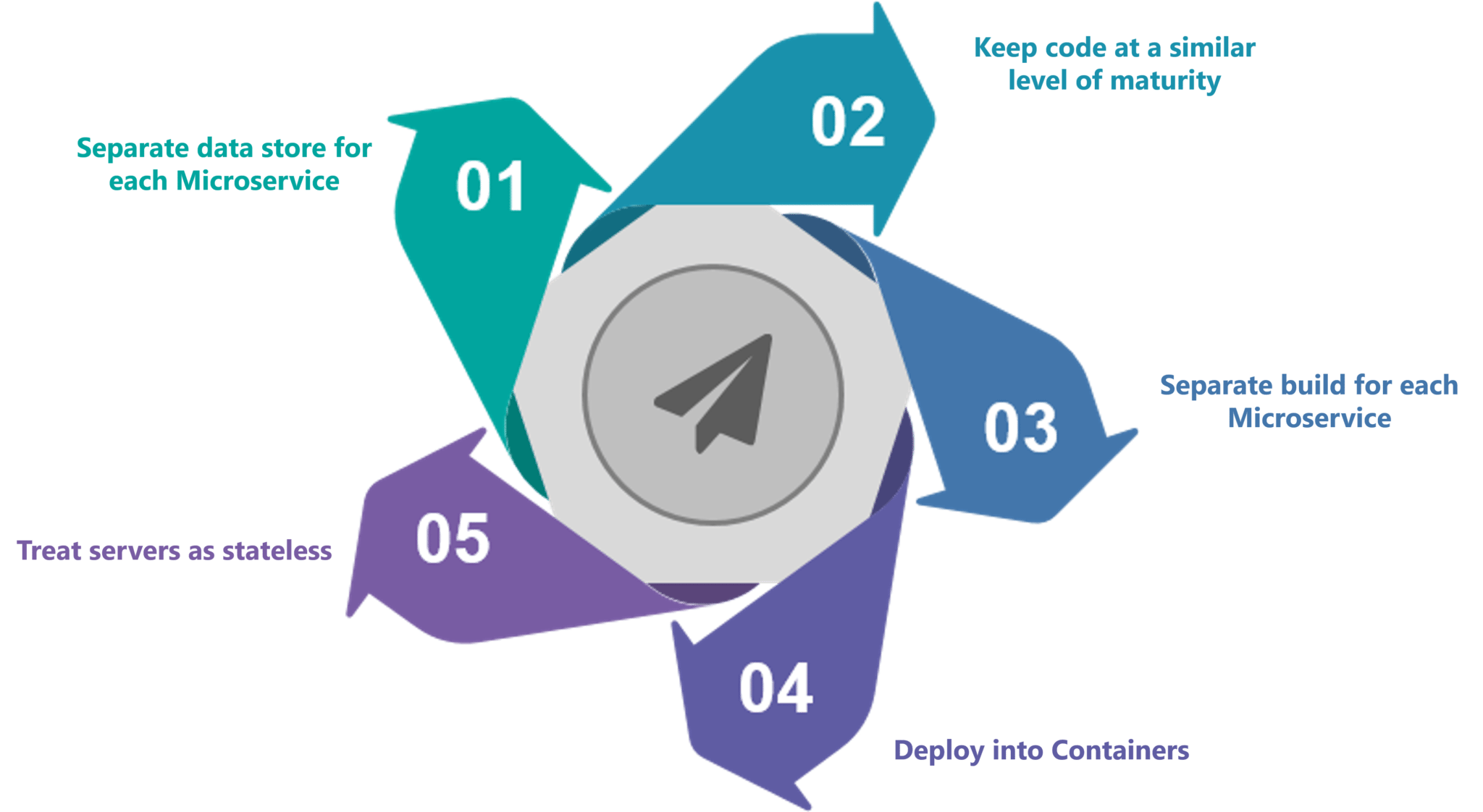
**Mixed Technology Stack** – Different languages and technologies can be used to build different services of the same application

**Granular Scaling** – Individual components can scale as per need, there is no need to scale all components together

## ****Best Practices to Design Microservices****

In today’s world, complexity has managed to creep into products. Microservice architecture promises to keep teams scaling and function better.

The following are the best practices to design Microservices:

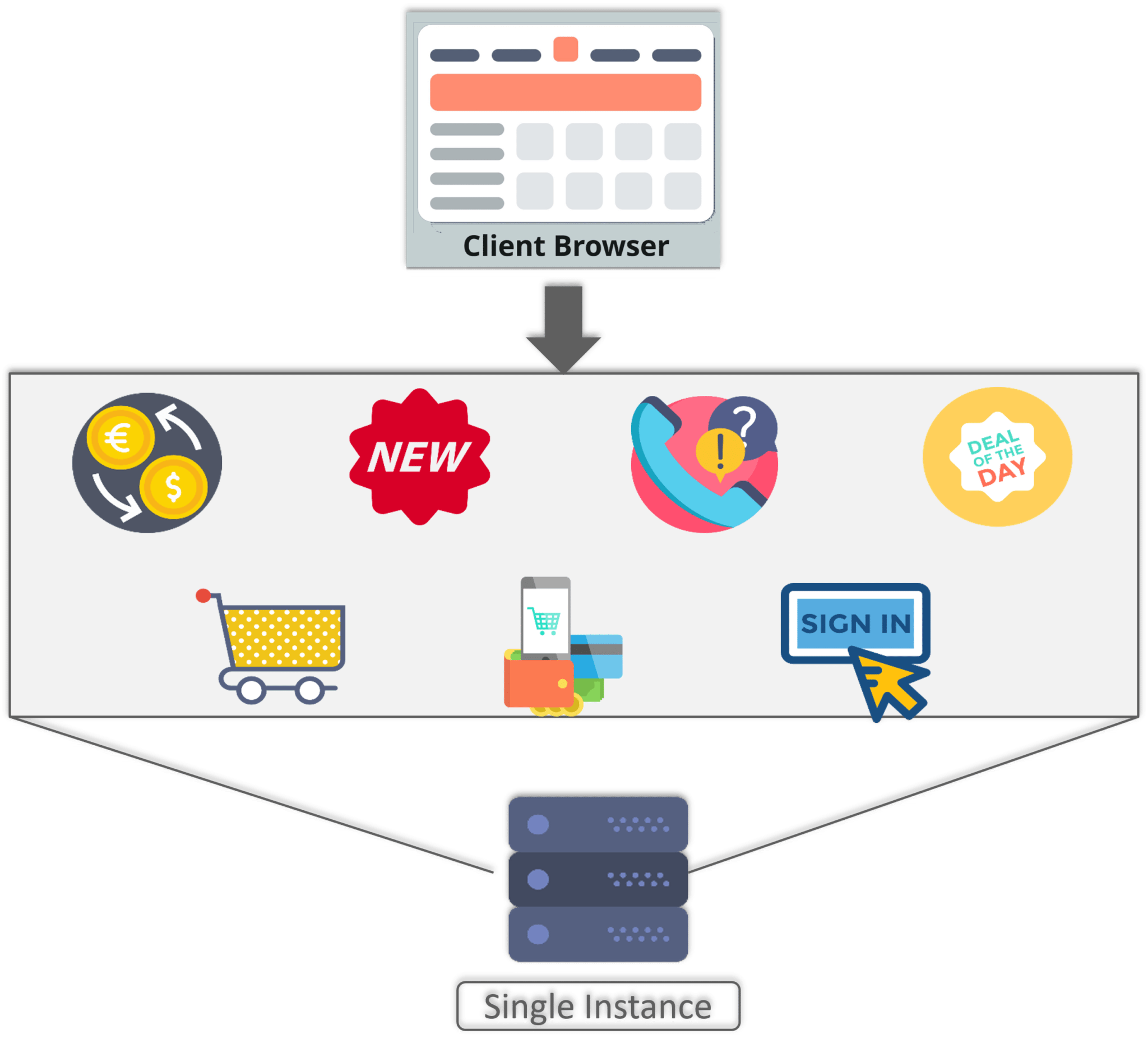


**Figure:** What Is Microservices – Best Practices to Design Microservices?

## ****Use-Case: Shopping Cart Application****

Let’s take a classic use case of a shopping cart application.

When you open a shopping cart application, all you see is just a website. But, behind the scenes, the shopping cart application has a service for accepting payments, a service for customer services and so on.

Assume that developers of this application have created it in a monolithic framework. Refer to the diagram below:

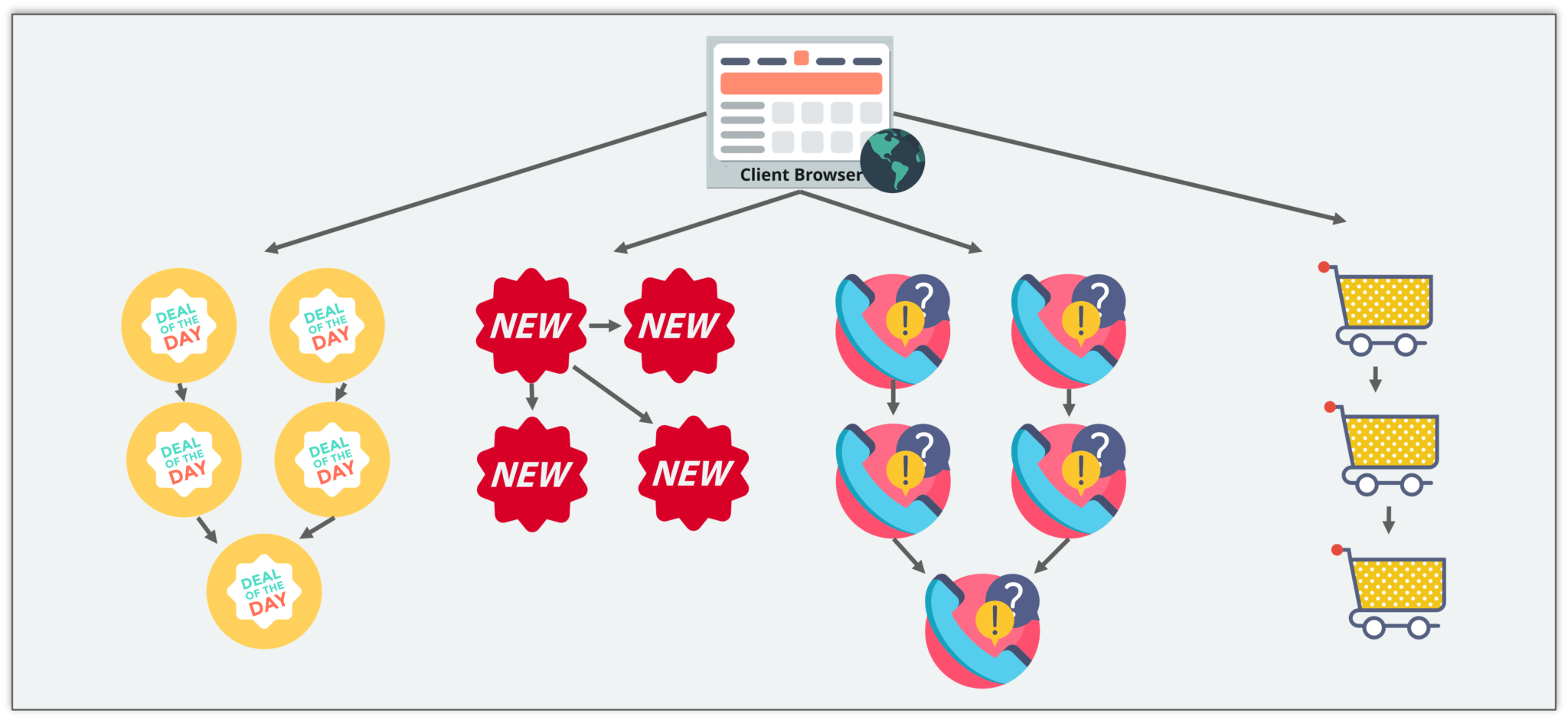
**Figure 8:** What Is Microservices – Monolithic Framework of Shopping Cart Application?

So, all the features are put together in a single code base and are under a single underlying database.

Now, let’s suppose that there is a new brand coming up in the market and developers want to put all the details of the upcoming brand in this application.

Then, they not only have to rework on the service for new labels, but they also have to reframe the complete system and deploy it accordingly.

To avoid such challenges developers of this application decided to shift their application from a monolithic architecture to Microservices. Refer to the diagram below to understand the Microservices architecture of shopping cart application



**Figure 9:** What Is Microservices – Microservice Architecture of Shopping Cart Application?

This means that developers don’t create a web Microservice, a logic Microservice, or a database Microservice. Instead, they create separate Microservices for search, recommendations, and customer services and so on.

This type of architecture for the application not only helps the developers to overcome all the challenges faced with the previous architecture but also helps the shopping cart application to be built, deployed, and scale up easily.

## ****Companies using Microservices****

There is a long list of companies using Microservices to build applications, these are just to name a few:



**Figure 10:** What Is Microservices – Companies Using Microservices

# **Microservice Architecture – Learn, Build and Deploy Microservices**

**In this we will learn about the following:**

* Definition Of Microservice Architecture
* Key Concepts Of Microservice Architecture
* Pros And Cons Of Microservice Architecture
* UBER – Case Study

## ****Definition of Microservices****

As such, there is no proper definition of Microservices but we can say that it is a framework which consists of small, independent deployable services performing different operations.

Microservices focus on a single business functionality that can be implemented as fully independent deployable services and implement them on different technology stacks.

**Key Concepts of Microservice Architecture**

Before you start building your own applications using Microservices you need to be clear about the scope, and functionalities of your application.

Following are some guidelines to be followed while discussing Microservices.

**Guidelines While Designing Microservices**

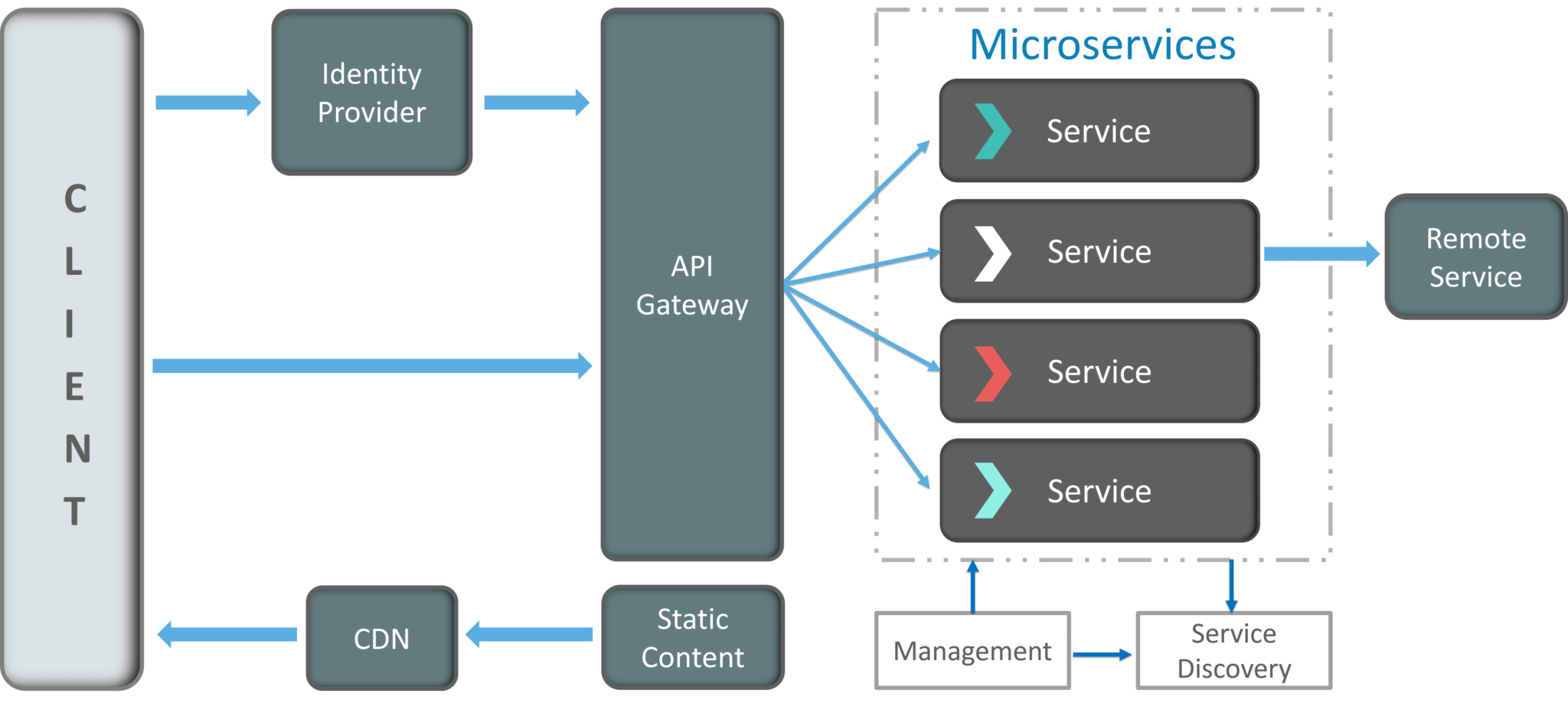
* Separate the domains and be clear with the functionalities.
* Each Microservice shall concentrate only on one service of the application.
* Each service is individually deployable.
* The communication between Microservices is done via a stateless server.
* Each service can be furthered refactored into smaller services, having their own Microservices.

**How Does Microservice Architecture Work?**

A typical Microservice Architecture (MSA) should consist of the following components:

1. Clients
2. Identity Providers
3. API Gateway
4. Messaging Formats
5. Databases
6. Static Content
7. Management
8. Service Discovery

Refer to the diagram below.



I know the architecture looks a bit complex, but let me simplify it for you.

**1. Clients**

The architecture starts with different types of clients, from different devices trying to perform various management capabilities such as search, build, configure etc.

**2. Identity Providers**

These requests from the clients are then passed on the identity providers who authenticate the requests of clients and communicate the requests to API Gateway. The requests are then communicated to the internal services via well-defined API Gateway.

**3. API Gateway**

Since clients don’t call the services directly, API Gateway acts as an entry point for the clients to forward requests to appropriate Microservices. Here API Gateway works as a front controller who identify the request send the request to particular Microservice.

**The advantages of using an API gateway include:**

* All the services can be updated without the clients knowing.
* Services can also use messaging protocols that are not web-friendly.
* The API Gateway can perform cross-cutting functions such as providing security, load balancing etc.

After receiving the requests of clients, the internal architecture consists of Microservices which communicate with each other through messages to handle client requests.

**4. Messaging Formats**

There are two types of messages through which they communicate:

* **Synchronous Messages:**

In Synchronous messaging client waits for the server to respond to a message. Messages are able to flow in both directions, to and from. Essentially it means that synchronous messaging is a two way communication. I.e. Sender sends a message to receiver and receiver receives this message and gives reply to the sender. Sender will not send another message until it gets a reply from receiver.

In the situation where clients wait for the responses from a service, Microservices usually use **REST (Representational State Transfer)** as it depends on a stateless, client-server, and the **HTTP protocol**.

* **Asynchronous Messages:**

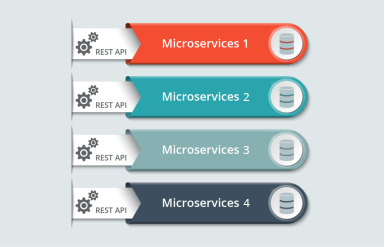
In Asynchronous messaging client does not wait for a message from the server. An event is used to trigger a message from a server. So even if the client is down, the messaging will complete successfully. Asynchronous Messaging means that, it is a one way communication and the flow of communication is one way only.

In the situation where clients do not wait for the responses from a service, for this type of messaging Microservices usually use protocols such as **AMQP, STOMP, and MQTT**. These protocols are used in this type of communication since the nature of messages is defined and these messages have to be interoperable between implementations.

The next question that may come to your mind is how do the applications using Microservices handle their data?

**5. Data Handling**

Each Microservice can have a private database to capture their data and implement the respective business functionality.

****

We can say that Microservices supports inter-process communication for different technology and database.

**6. Static Content**

After the Microservices communicate within themselves, they deploy the static content to a cloud-based storage service that can deliver them directly to the clients via **Content Delivery Networks (CDNs)**.

Apart from the above components, there are some other components appear in a typical Microservices Architecture:

**7. Management(Load Balancing)**

This component is responsible for balancing the services on nodes and identifying failures.

**8. Service Discovery**

Acts as a guide to Microservices to find the route of communication between them as it maintains a list of services on which nodes are located.

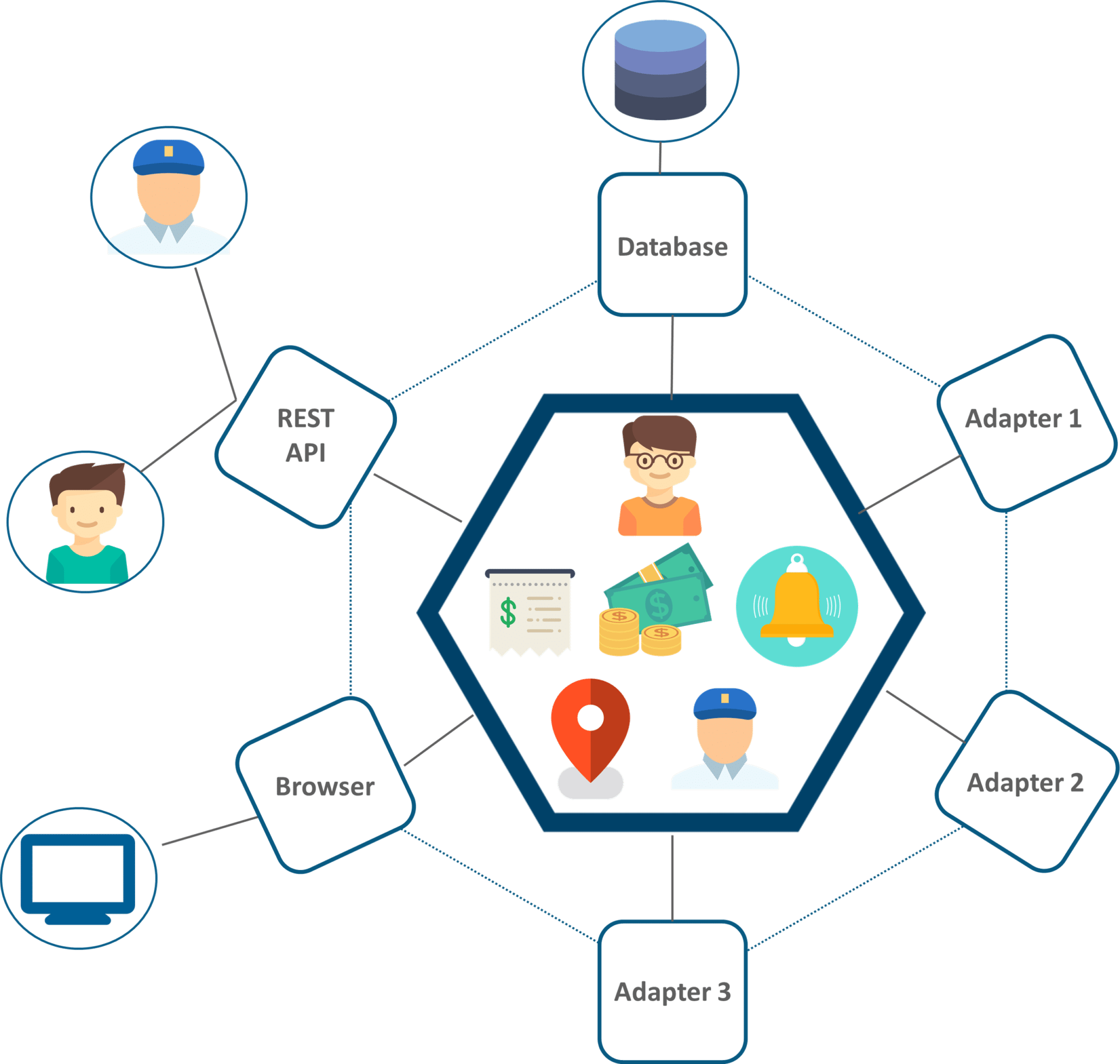
## ****Pros and Cons of Microservice Architecture****

|  |  |
| --- | --- |
| **Pros Of Microservice Architecture** | **Cons Of Microservice Architecture** |
| Freedom to use different technologies | Increases troubleshooting challenges |
| Each Microservice focuses on single business capability | Increases delay due to remote calls |
| Supports individual deployable units | Increased efforts for configuration and other operations |
| Allows frequent software releases | Difficult to maintain transaction safety |
| Ensures security of each service | Tough to track data across various service boundaries |
| Multiple services are parallels developed and deployed | Difficult to move code between services |

**UBER CASE STUDY**

**UBER’s Previous Architecture**

Like many startups, UBER began its journey with a monolithic architecture built for a single offering in a single city. Having one codebase seemed cleaned at that time, and solved UBER’s core business problems. However, as UBER started expanding worldwide they rigorously faced various problems with respect to scalability and continuous integration.



**Figure 4:** Monolithic Architecture Of UBER – Microservice Architecture

The above diagram depicts UBER’s previous architecture.

* A REST API is present with which the passenger and driver connect.
* Three different adapters are used with API within them, to perform actions such as billing, payments, and sending emails/messages that we see when we book a cab.
* A MySQL database to store all their data.

So, if you notice here all the features such as **passenger management, billing, notification features, payments, trip management and driver management were composed within a single framework**.

**Problem Statement**

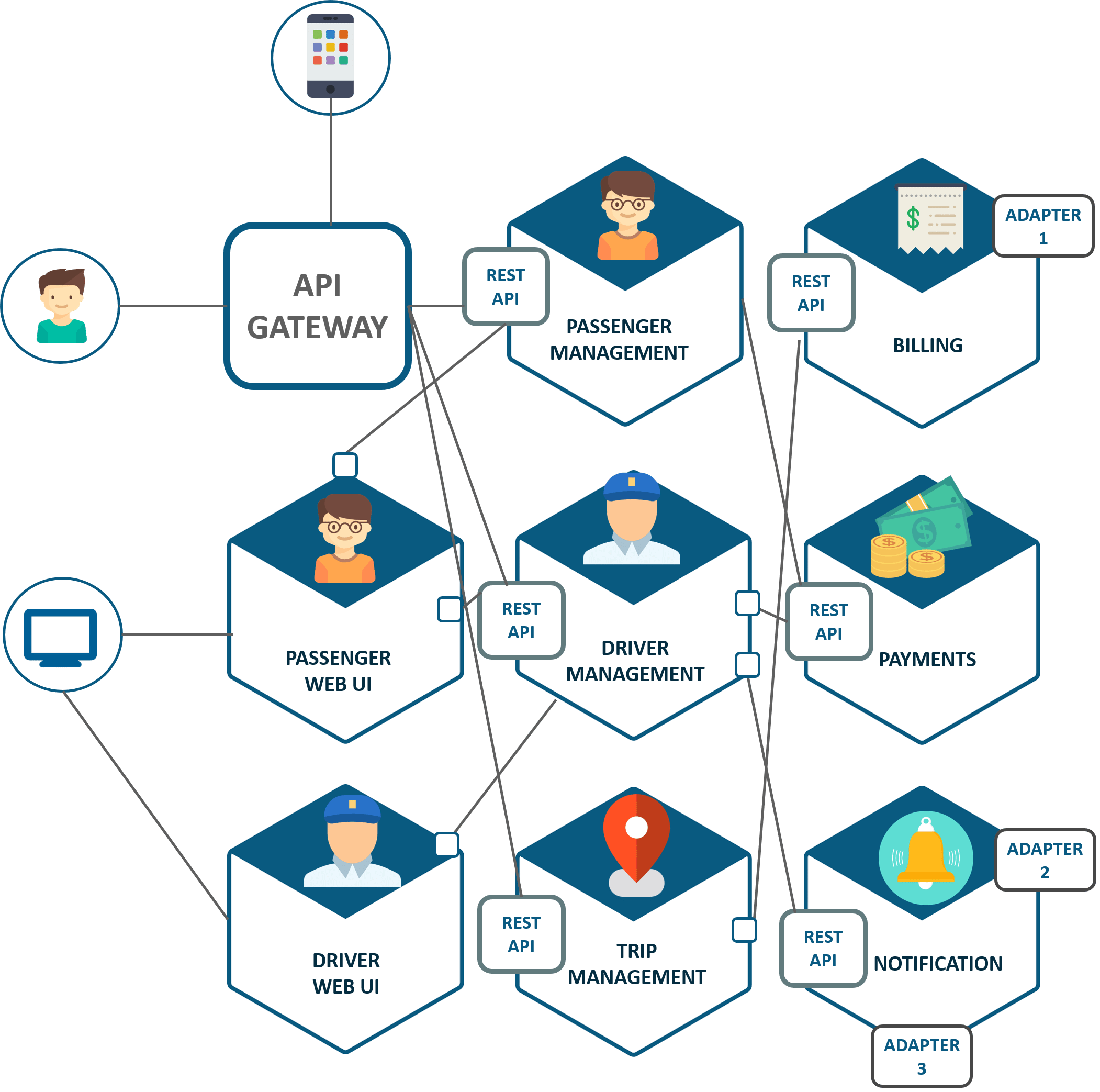
While UBER started expanding worldwide this kind of framework introduced various challenges. The following are some of the prominent challenges

* All the features had to be re-built, deployed and tested again and again to update a single feature.
* Fixing bugs became extremely difficult in a single repository as developers had to change the code again and again.
* Scaling the features simultaneously with the introduction of new features worldwide was quite tough to be handled together.

**Solution**

To avoid such problems UBER decided to change its architecture and follow the other hyper-growth companies like Amazon, Netflix, Twitter and many others. Thus, UBER decided to break its monolithic architecture into multiple codebases to form a Microservice architecture.

Refer to the diagram below to look at UBER’s Microservice architecture.



* The major change that we observe here is the introduction of API Gateway through which all the drivers and passengers are connected. From the API Gateway, all the internal points are connected such as passenger management, driver management, trip management and others.
* The units are individual separate deployable units performing separate functionalities.
* For Example: If you want to change anything in the billing Microservices, then you just have to deploy only billing Microservices and don’t have to deploy the others.
* All the features were now scaled individually i.e. the interdependency between each and every features was removed.
* For Example, we all know that the number of people searching for cabs is more comparatively more than the people actually booking a cab and making payments. This gets us an inference that the number of processes working on the passenger management Microservice is more than the number of processes working on payments.

In this way, UBER benefited by shifting its architecture from monolithic to Microservices.

**Differences between the Microservices vs SOA**

Before going difference between Microservices and SOA, first of let’s talk about Monolithic architecture.

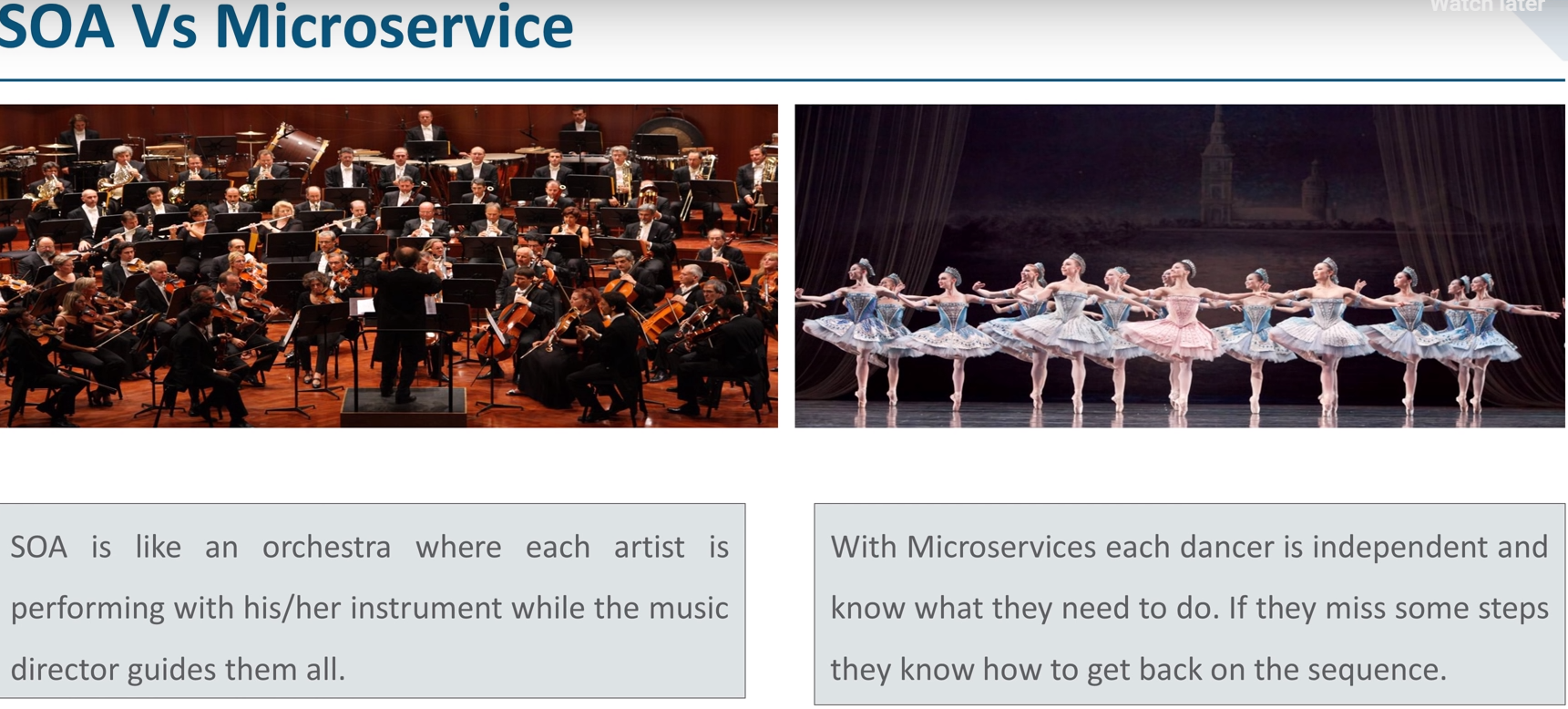
**Monolithic application(MA)** can be taken as a container which is hosting a number of components, these component could be a part of software application. All the components are hosted together and delivered together that is why it is called single unit architecture.We get lot of challenges with Monolithic application.

1. They are not flexible
2. MA cannot be built using different technologies.
3. They are un-reliable i.e. if one features of system does not work then whole system does not work.
4. MA are not scalable. For a small also a complete system has to re-build.
5. MA development is very slow as after completing one feature we can build another feature.
6. MA does not fit for complex architecture as we cannot use different technologies.

Now let’s talk about Service oriented Architecture SOA:

In SAO services are broken down i.e. now in SAO a big application are broken down into smaller components that is why it called coarse grained architecture. Here in SOA let’s say if we have one software application inside which it provide 4 or 5 features then all the 4 or 5 features are delivered by 4 or 5 services and each of these services would have multiple tasks inside them and these smaller tasks together deliver as one particular features and whole software application consists of these features

But in case of Microservices it is little bit more different here in Microservices features or services are further broken down into multiple task level services and that is why it called fine grained architecture.



As per above example we can say that in SOA each of them work together but they are controlled by one directory hence we can say SOA is centrally governed architecture. While in case of Microservices we don’t have any centrally governed architecture that is why Microservices is called de-centralized architecture.

**Now let’s talk about difference in coordination of SAO and MS.**

**Incomplete ……Need to complete from videos.**

# **Microservices Tutorial – Learn all about Microservices with Example**

In this Microservices tutorial, the following topics will be covered:

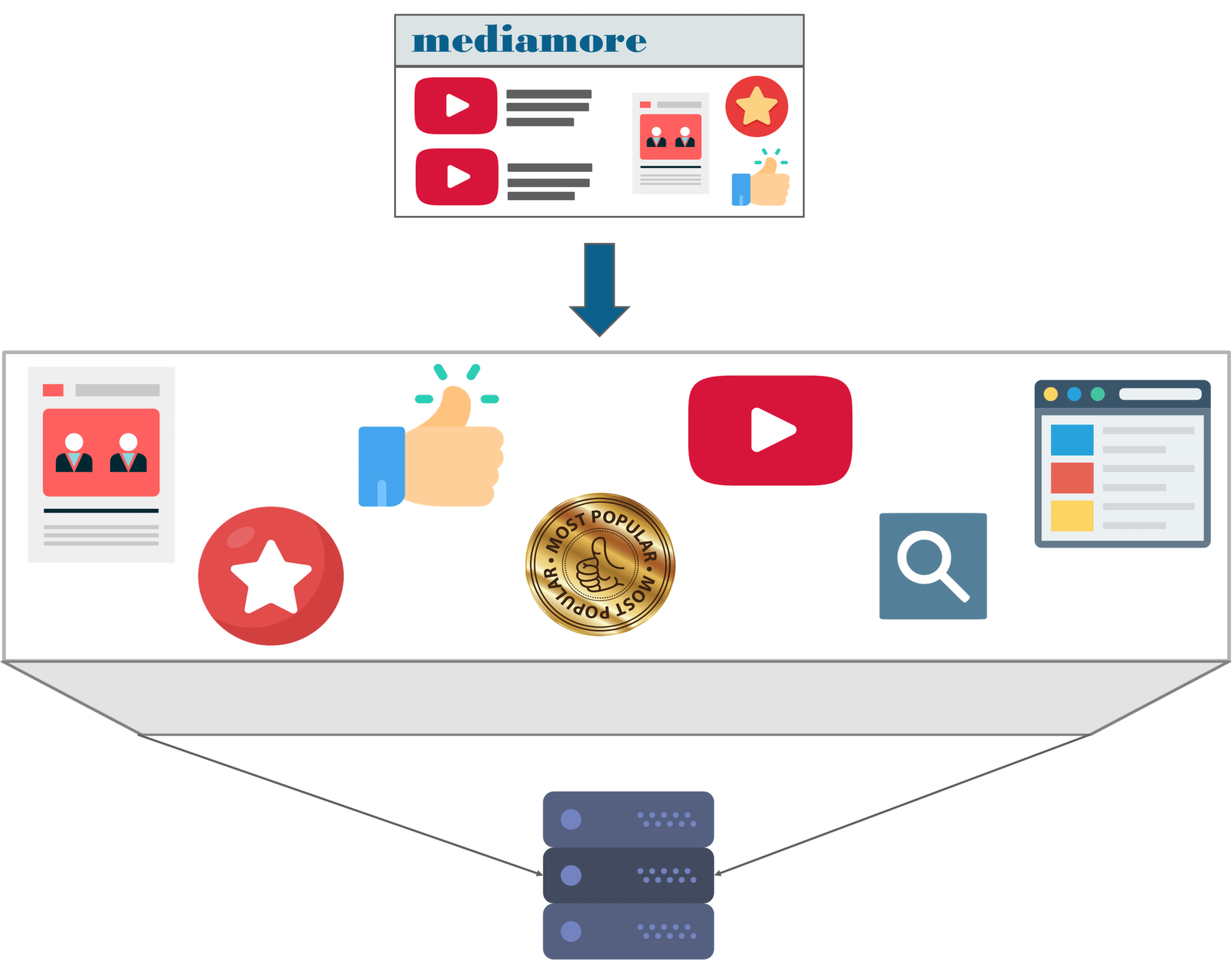
* [Monolithic Architecture](https://www.edureka.co/blog/microservices-tutorial-with-example#MonolithicArchitecture)
* [Challenges of Monolithic Architecture](https://www.edureka.co/blog/microservices-tutorial-with-example#ChallengesofMonolithicArchitecture)
* [What is Microservices](https://www.edureka.co/blog/microservices-tutorial-with-example#WhatisMicroservices)
* [Microservice Architecture](https://www.edureka.co/blog/microservices-tutorial-with-example#MicroserviceArchitecture)
* [Microservices Example – Demo](https://www.edureka.co/blog/microservices-tutorial-with-example#Hands-On)

**Let us explore the concepts of Microservices through a use-case on Mediamore.com.**

Mediamore is an entertainment company which provides streaming media and videos online. It consists of various TV Shows in different languages. Like many other companies Mediamore started its journey with a monolithic architecture.

Let us now explore the monolithic framework of Mediamore.

## ****Monolithic Architecture****



Refer to the above diagram. We can infer (conclude) that all the features such as the search, user-info, recommendations, video playlist and others are put on a single database using single code.

Now, I can tell you the challenges faced by the developers while using a monolithic framework by using some scenarios.

**Scenario 1: Scalability:** Let’s assume that the developers want to update the playlist according to most popular tv shows and also simultaneously want to update all videos to HD quality.

The developers cannot scale the application simultaneously. New instances of the same application have to be created every time a new feature has to be developed or deployed.

**Scenario 2: Agility:** Assume that developers want to make immediate changes in the application.

The monolithic application can definitely accommodate these changes. But, the problem here is that the developers have to rebuild the code for every small change.

**Scenario 3: Hybrid Technologies:** Suppose developers of this application are comfortable with various technologies like JAVA, C++,.NET, C#.

Even though they are comfortable with various technologies, they still have to build large and complex applications on a single technology.

**Scenario 4: Fault Tolerance:** Let’s suppose that a specific feature is not working in the application.

The complete system goes down because of this problem. In order to tackle this problem, the application has to be re-built, re-tested and also re-deployed.

So, how did the developers of Mediamore overcome these complexities?

Developers thus decided to re-architect their monolithic application into multiple individual deployable components, called as Microservices.

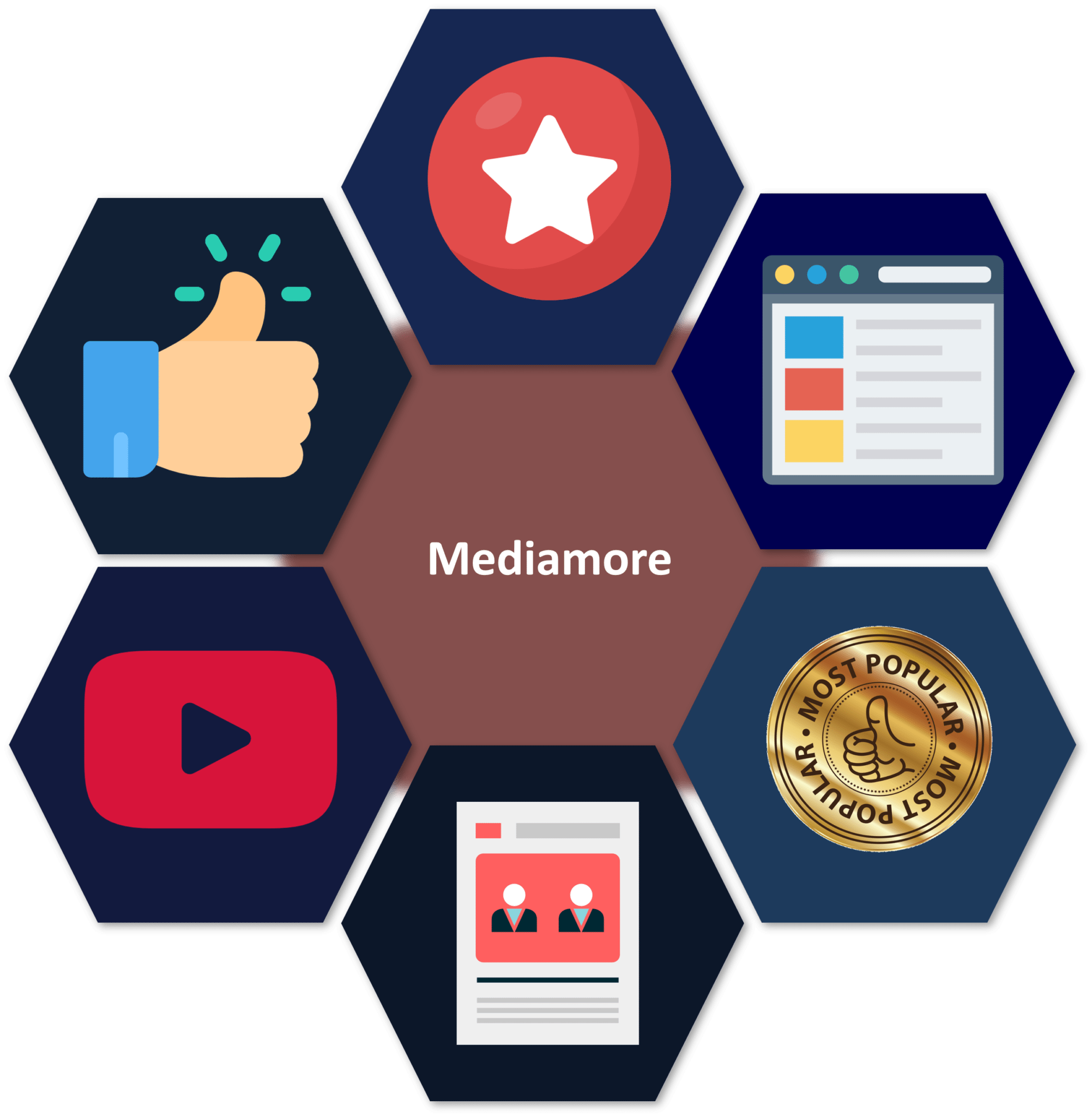
**Here lies the million dollar question!**

## ****What is Microservices?****

Microservices is an architecture wherein all the components of the system are put into individual components, which can be built, deployed, and scaled individually.

Let me explain you with a simple analogy.

You must have seen how bees build their honeycomb by aligning hexagonal wax cells. They initially start with a small section using various materials and continue to build a large beehive out of it. These cells form a pattern resulting in a strong structure which holds together a particular section of the beehive. Here, each cell is independent of the other but it is also correlated with the other cells. This means that damage to one cell does not damage the other cells, so, bees can reconstruct these cells without impacting the complete beehive.

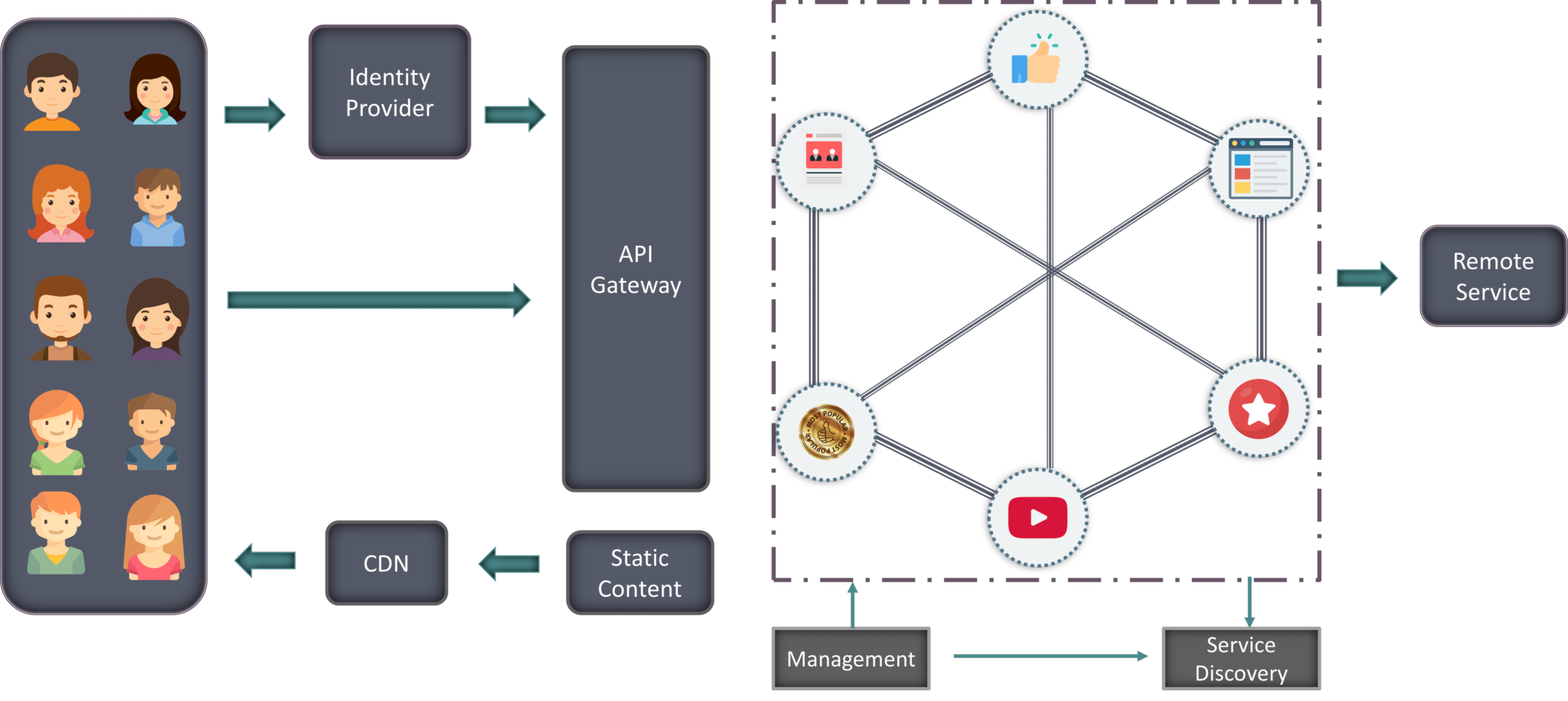


Refer to the above diagram. Here, each hexagonal shape represents an individual service component. Similar to the working of bees, each agile team builds an individual service component with the available frameworks and the chosen technology stack. Just as in a beehive, each service component forms a strong Microservice architecture to provide better scalability. Also, issues with each service component can be handled individually by the agile team with no or minimal impact on the entire application.

The next question that may come to your mind is how the different components of Microservice architecture work together.

But, before that, let me list down the components of the Microservice architecture.

Refer to the below diagram.



* **Clients** – Different users from various devices send requests.
* **Identity Providers** – Authenticates user or clients identities and issues security tokens.
* **API Gateway** – Handles client requests.
* **Static Content** – Houses all the content of the system.
* **Management** – Balances services on nodes and identifies failures.
* **Service Discovery** – A guide to find the route of communication between Microservices.
* **Content Delivery Networks** – Distributed network of proxy servers and their data centers.
* **Remote Service** – Enables the remote access information that resides on a network of IT devices.

Let me now brief you on how these components work together on Mediamore by considering a scenario.

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

### **Scenario:**

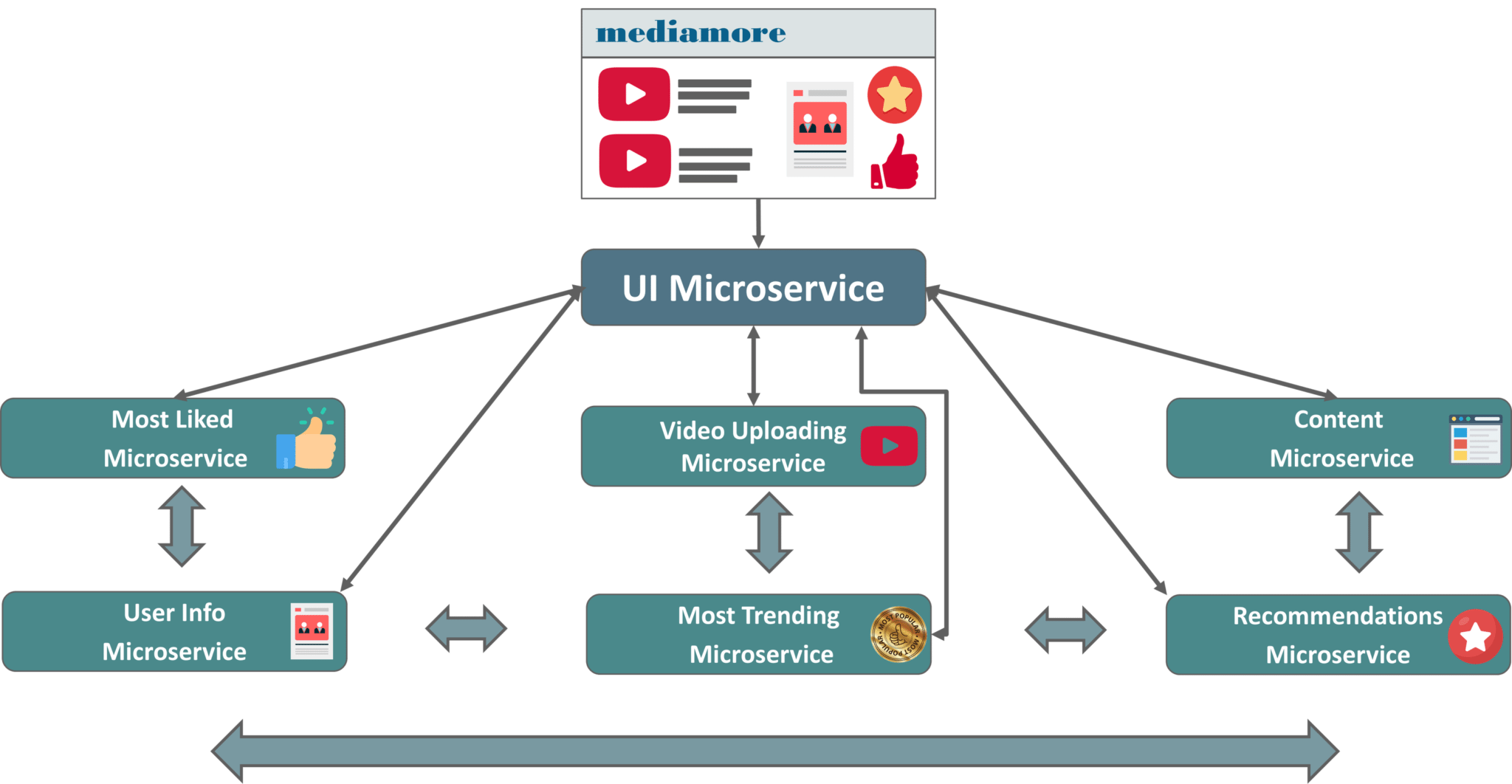
Alice is an avid user of Mediamore. She uses Mediamore regularly to watch her favorite series online. She recently missed watching an episode of her favorite TV show.

When Alice logs in to the application, she sees the most recommended content on her home page. After some searching, she finally finds her TV Show.

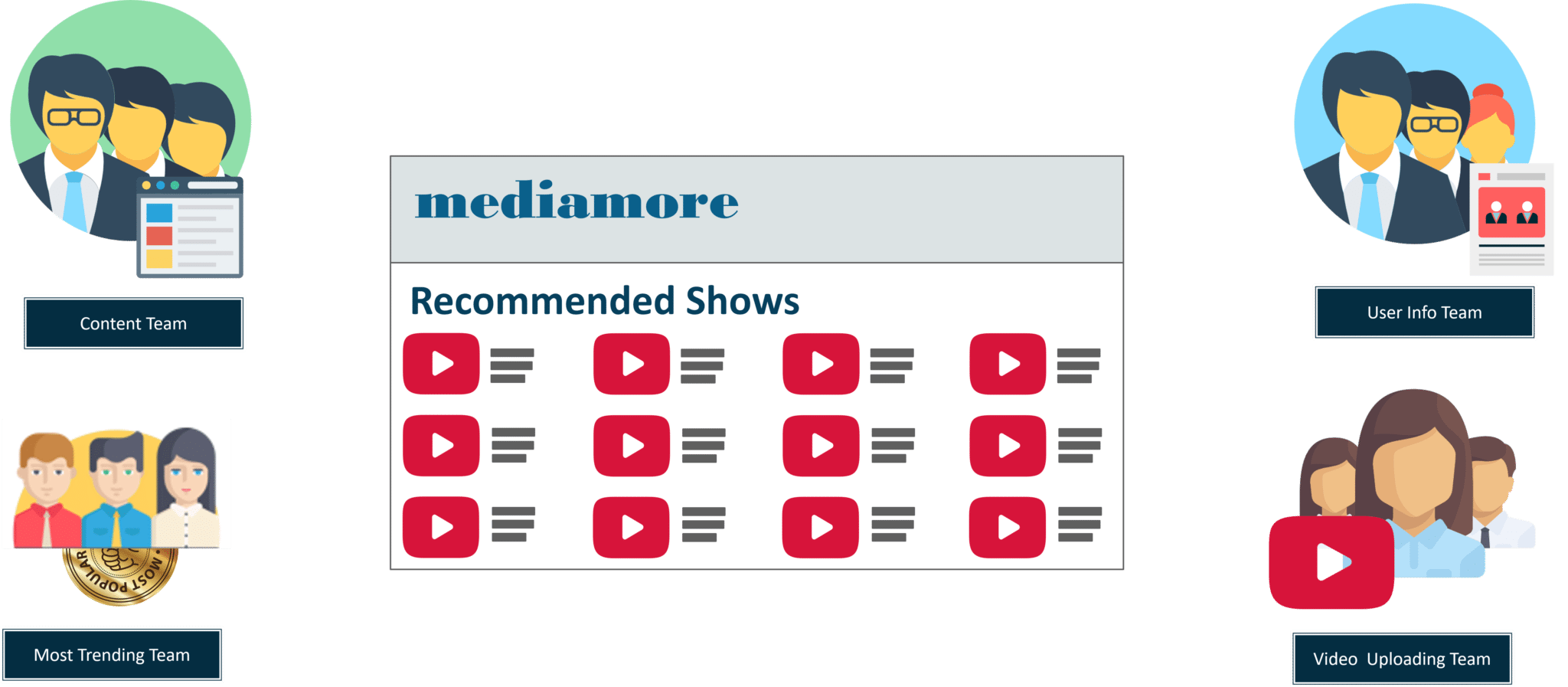
But, what if Alice wants to get her TV Show with a single click.

**How will the developers work together to fulfill Alice’s request?**

1. Alice’s request is passed on to the **Identity Provider**. Identity provider thus authenticates Alice’s request by identifying her as a regular user on Mediamore.
2. These requests are passed to the **API Gateway** which acts as an entry point for Alice to forward her requests to the appropriate Microservices.
3. Each feature has its own working Microservice, handling their own data. These Microservices also have their own**load balancers** and **execution environments**to function properly.



**Refer to the diagram below. Each Microservice is handled by a small agile team such as content team, video uploading team, most trending team, search team etc.**



**Figure:** Division of Teams of Mediamore – Microservices Tutorial

* The content team consists of millions of TV Shows that the application provides.
* The video uploading team have the responsibility to upload all the content into the application
* The most trending team houses the most trending shows according to the geographical location of users and so on.

These small teams of developers relate each and every piece of content with the metadata that describes the searched content. Then, metadata is fed into another Microservice i.e. the search function which ensures Alice’s search results are captures into the content catalog.

Then, the third Microservice i.e. most trending Microservice captures the trending content among all the Mediamore users according to their geographical locations.

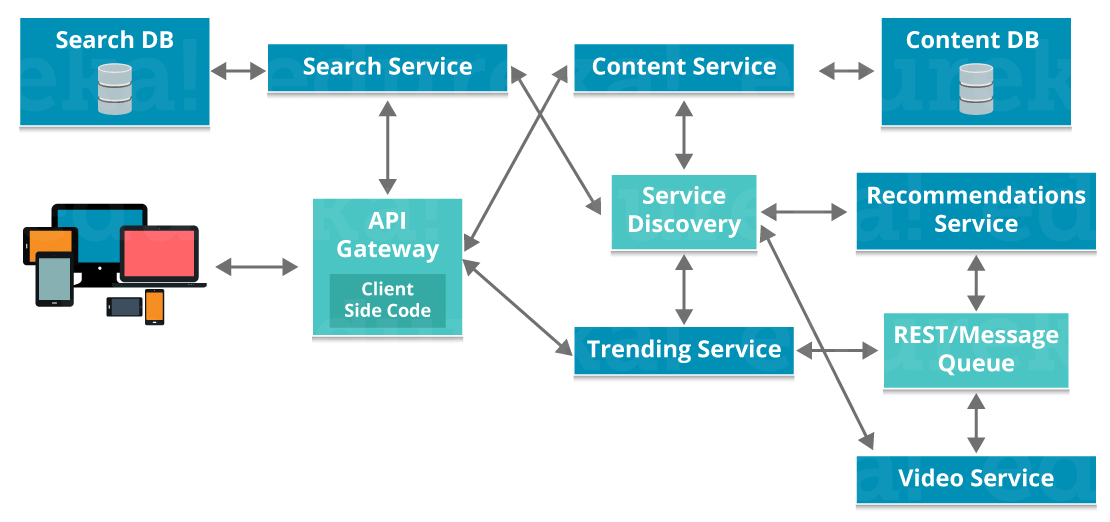
The content from this Microservice is what Alice sees when she first logs into Mediamore.

These individually deployable Microservices are put in specific containers to join the application. Containers are used to deliver the code to the sector where deployment is required.

But before they join the application to work together, they have to find each other to fulfil Alice’s request.

**How do these Microservices find each other?**

Microservices use service discovery which acts as a guide to find the route of communication between each of them. Microservices then communicate with each other via a stateless server i.e. either by HTTP Request/Message Bus.



**Figure 6: Communication between Microservices – Microservices Tutorial**

These Microservices communicate with each other using an **API Gateway**.

After theMicroservices communicate with each other, they deploy the **static content** to a cloud-based storage service that can deliver them directly to the clients via **Content Delivery Networks (CDNs)**.

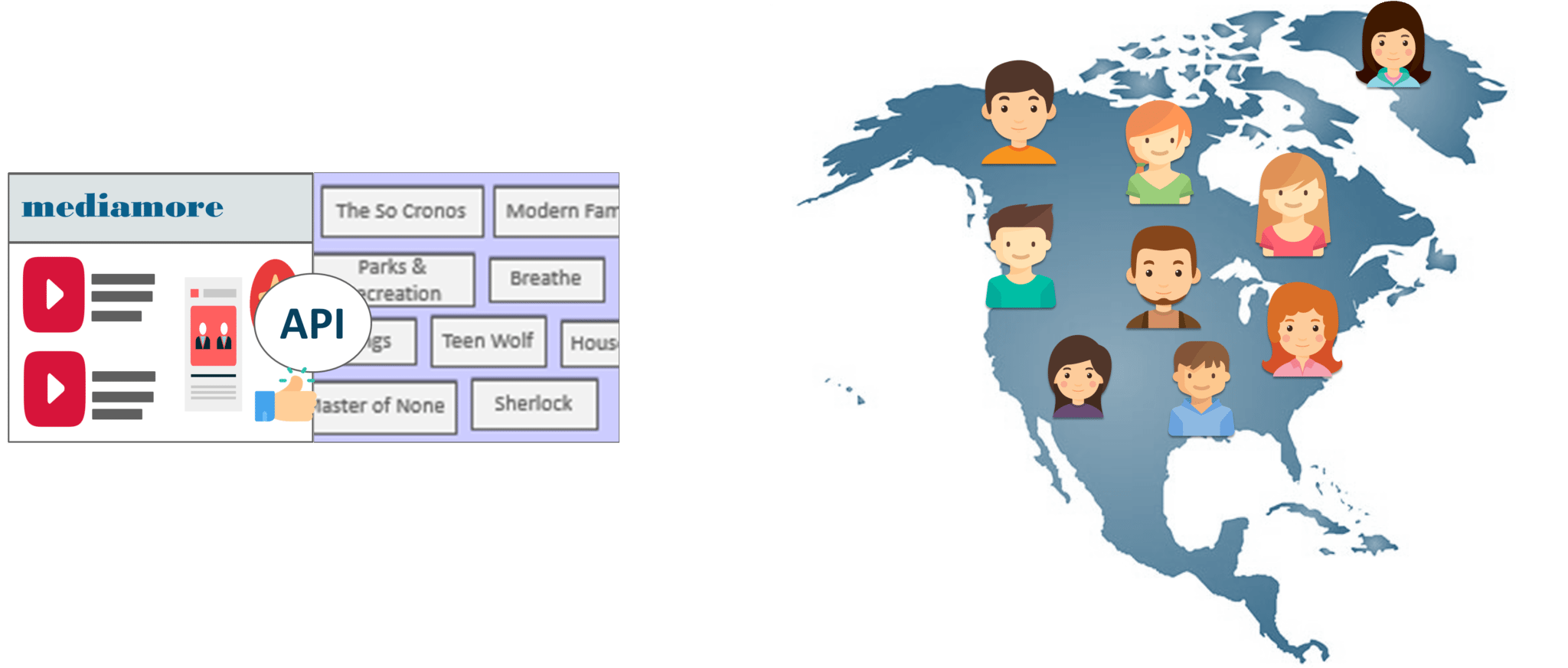
So, when Alice searches for her TV Show, the search Microservice communicates with the content catalog service in API about what is Alice searching for and then these Microservices compare the typed words with the metadata they already have.

Once the teams of developers capture the most typed words by Alice, the analytics team update the code in recommendations Microservice and compare Alice’s most viewed content and preferences to popular content among other users in the same geographical region.

This means that the next time Alice logs on to the application, she not only sees the most popular content but also finds a personalized playlist which contains the shows she has previously viewed.

In this way, Alice’s request is fulfilled by the development team in a quick manner as they did not have to build the complete application again and just had to update the code to deploy this new functionality.

So this way Microservices invoke parallel environments to satisfy millions of customers with varying interests.



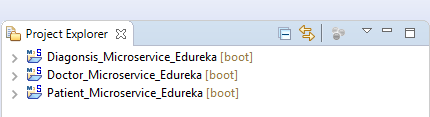
**Figure 7:** **Representation of how to search operation is performed with the help of API – Microservices Tutorial**

Now in this Microservices Tutorial, my next section will focus on Hands-On.

## ****Microservices Example – Demo****

To demonstrate the concepts of Microservices, I have created 3 Maven Projects called as Doctor\_Microservice\_Edureka, Diagnosis\_Microservice\_Edureka, and Patient\_Microservice\_Edureka using Spring Boot.

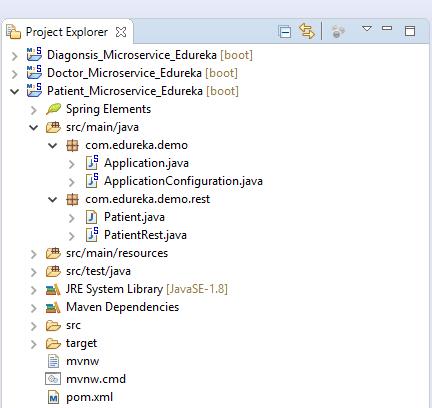
Refer to the snapshot below.



Before you understand how these 3 projects interact with each other. Let me brief you on the files of these projects.

To explain this I will consider the project  **Patient\_Microservice\_Edureka** and list down its basic files.

Refer to the snapshot below.



**Pom.xml** – Dependencies are added for the creation of REST services.

**Application.java** – Identical class in all three projects. Acts as an initiator to Spring Boot.

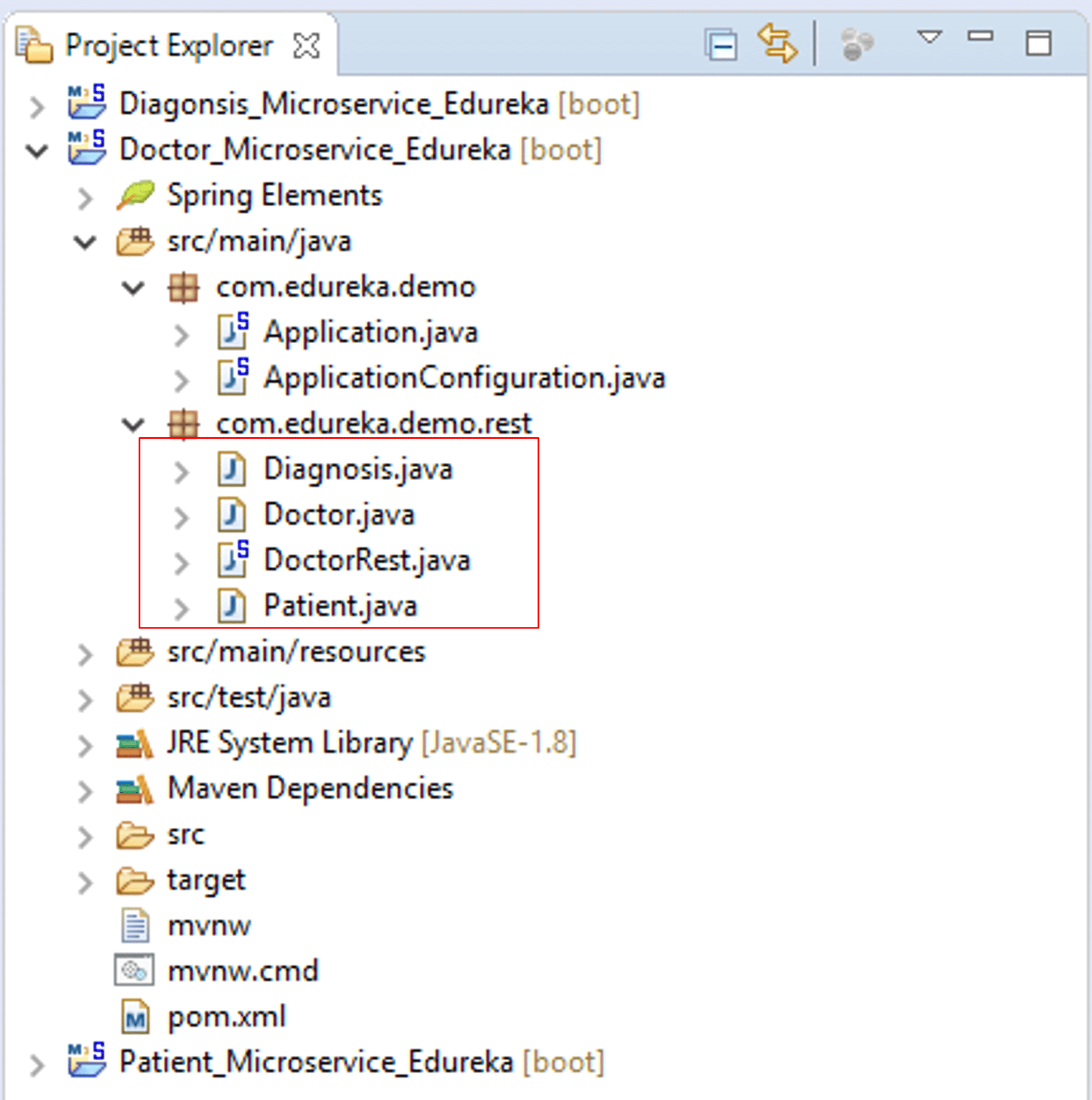
**ApplicationConfiguration.java** – Research configuration class responsible for exposing REST services for application users.

**Patient.java** – A simple class consisting of input such as the patient’s name, id, email.

**PatientRest.java** – Starts the implementation of the REST services in the project.

In this way, similar files are created for the other 2 projects with some additional files in Doctor\_Microservice.

REST services are thus created to search patients and the diagnosis. Keys of patient and diagnosis are passed as a parameter to a method (PatientDetails) in Doctor\_Microservice.This method gets the data of the patients and diseases.

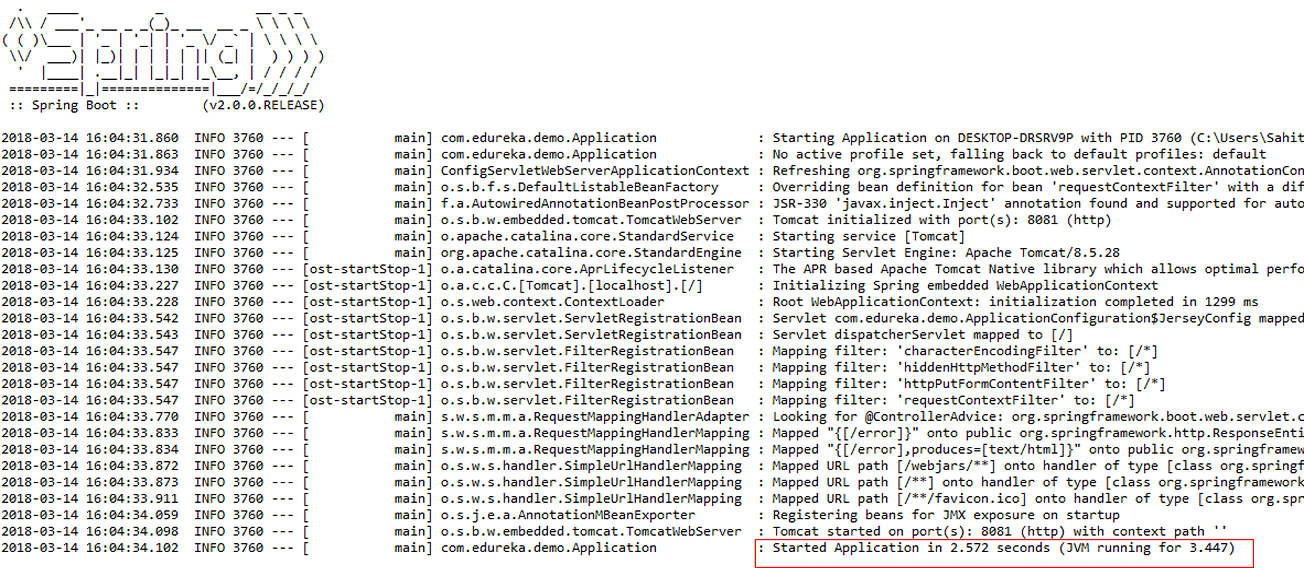


Refer to the snapshot above. Here, we observe that Patient and Diagnosis classes are included in Doctor\_Microservice\_Edureka. These classes are cloned from their original projects.

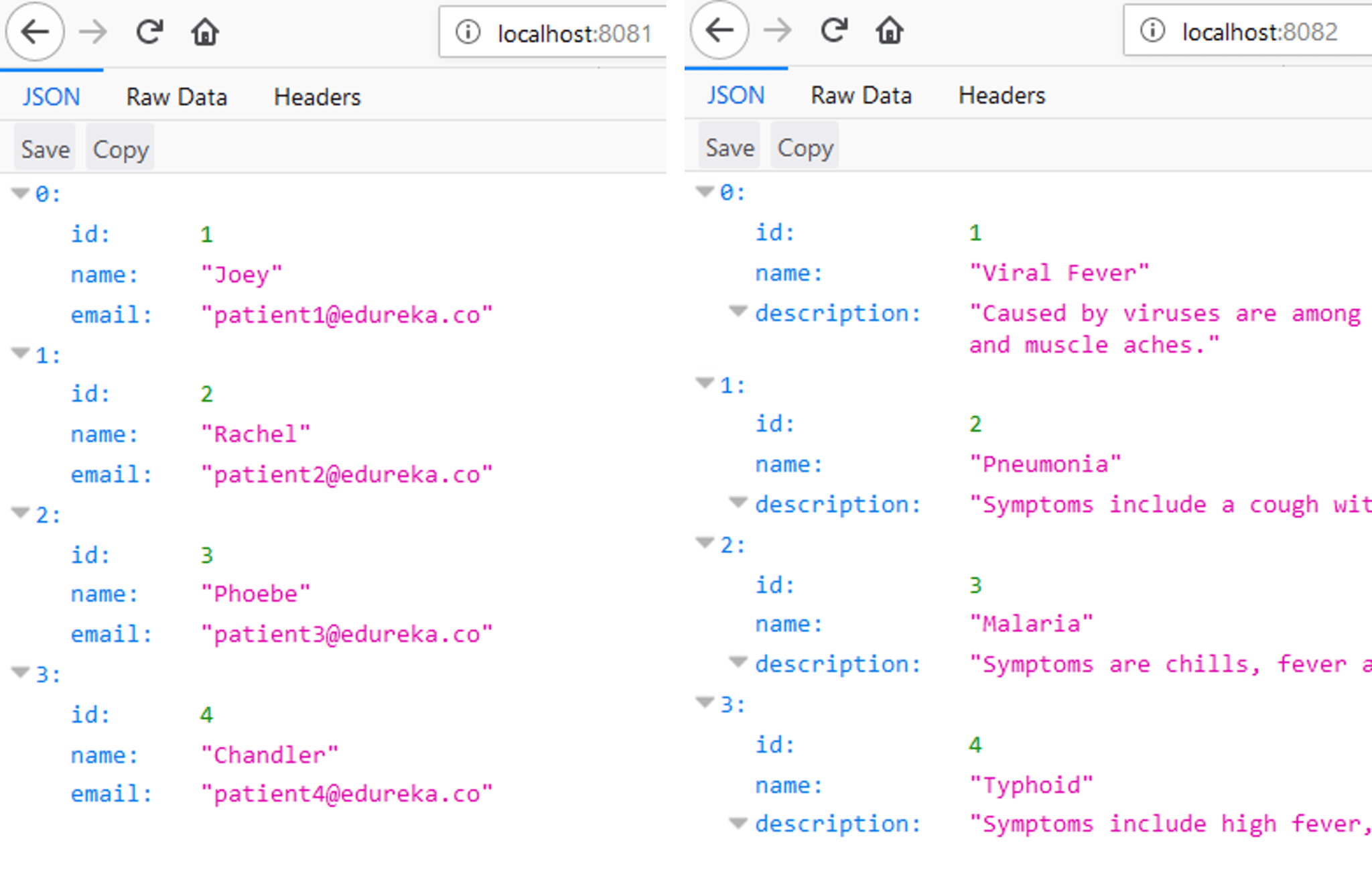
Then to start the REST services, I have initialized **Patient\_Microservice\_Edureka**on port **8081**, **Diagnosis\_Microservice\_Edureka** on **8082** and the **Doctor\_Microservice\_Edureka** on port **8083**.

With the above configurations, when I run each service simultaneously, 3 different console windows run in the Eclipse console window.

As you can see below that after we run the projects, Spring Boot generates a boot log. This log consists information on initializing tomcat and its associated resources. The last line of the log indicates us whether our application has started or not.



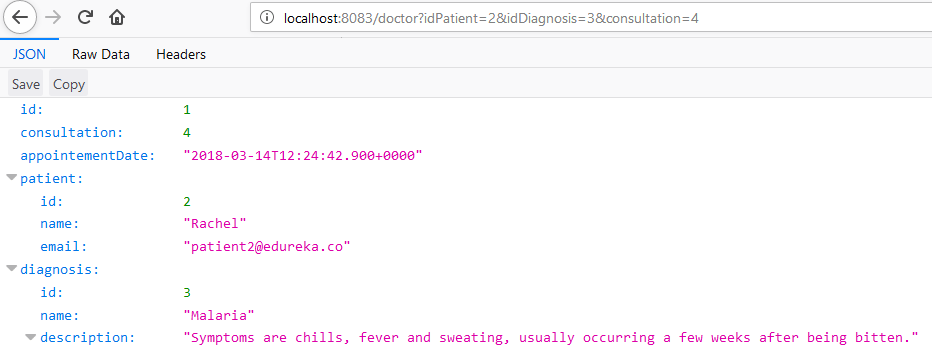
To test if the patient and diagnosis services are functioning properly, one a browser (Mozilla Firefox) and go to **http://localhost:8081/** and**http://localhost:8082/** URLs. You should get the outputs as shown below:



Finally, to test the functionality of the **Doctor\_Microservice\_Edureka**, I simulated the information of the patient with id 2, suffering from a disease of id 3  and consulting a doctor with consultation 4.

The URL used is as follows: **http://localhost:8083/doctor?idPatient=2&idDiagnosis=3&consultation=4**

Refer to the snapshot below for the output.



In this way, 3 Microservices interact with each other to produce the desired results.