**MicroServices:**

# PART I: What Is Microservices – Introduction To Microservice Architecture?

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. This blog will clear your understanding of how developers use microservices to scale their applications according to their need.

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

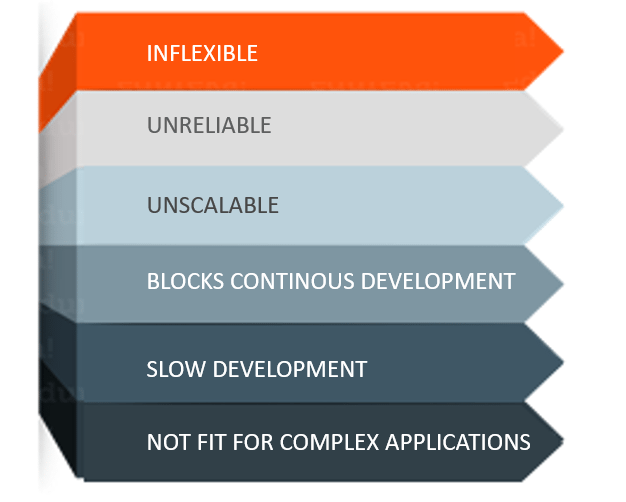
* Why Microservices?
* What Is Microservices?
* Features Of Microservice Architecture
* Advantages Of Microservice Architecture
* Best Practices To Design Microservices
* Companies Using Microservices

**Why Microservices?**

Now, before I tell you about microservices, let’s see the architecture that prevailed before microservices i.e. the **Monolithic Architecture.**

In layman terms, you can say that its 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 Continous 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?****

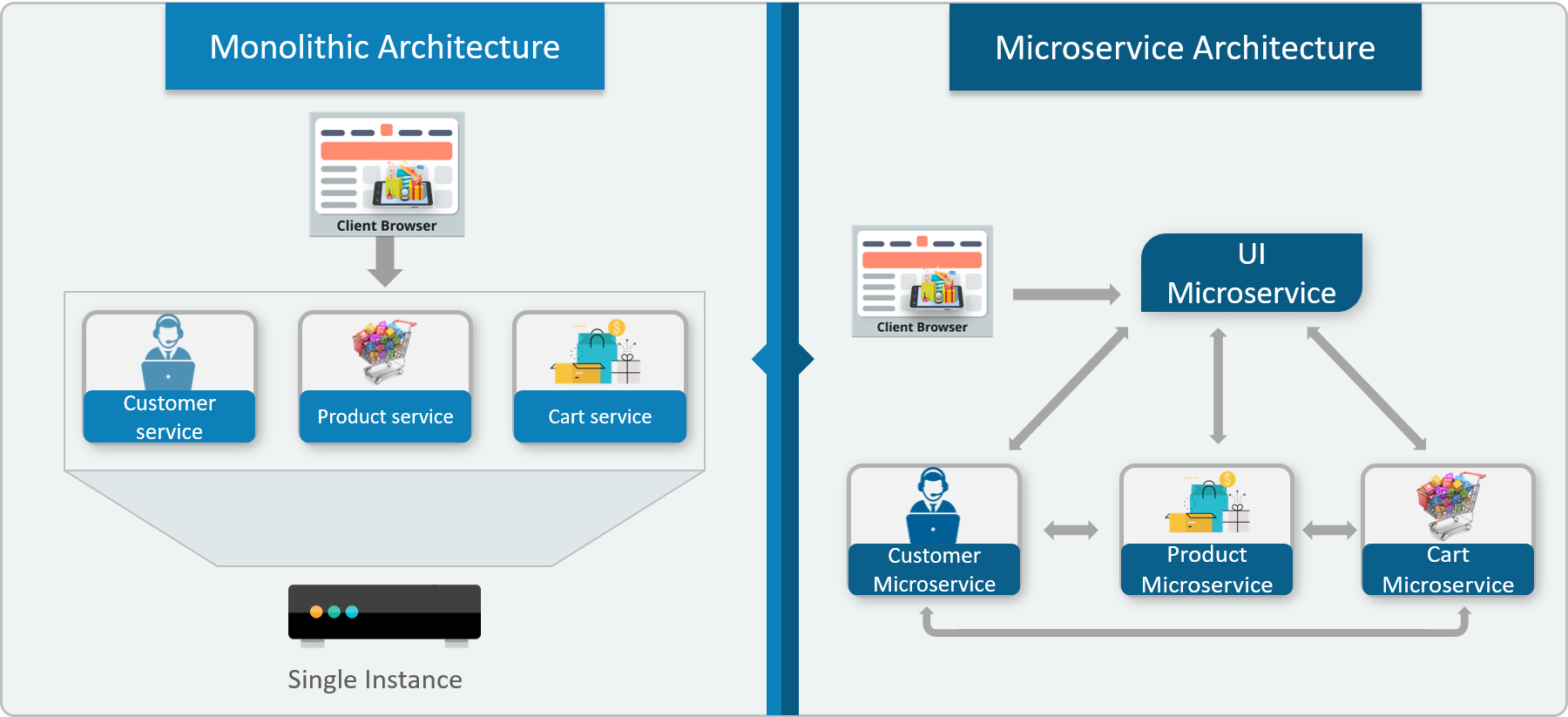
**Microservices**, aka [***Microservice Architecture***](https://www.edureka.co/blog/microservice-architecture/), is an architectural style that structures an application as a collection of small autonomous services, modeled around a **business domain.**



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

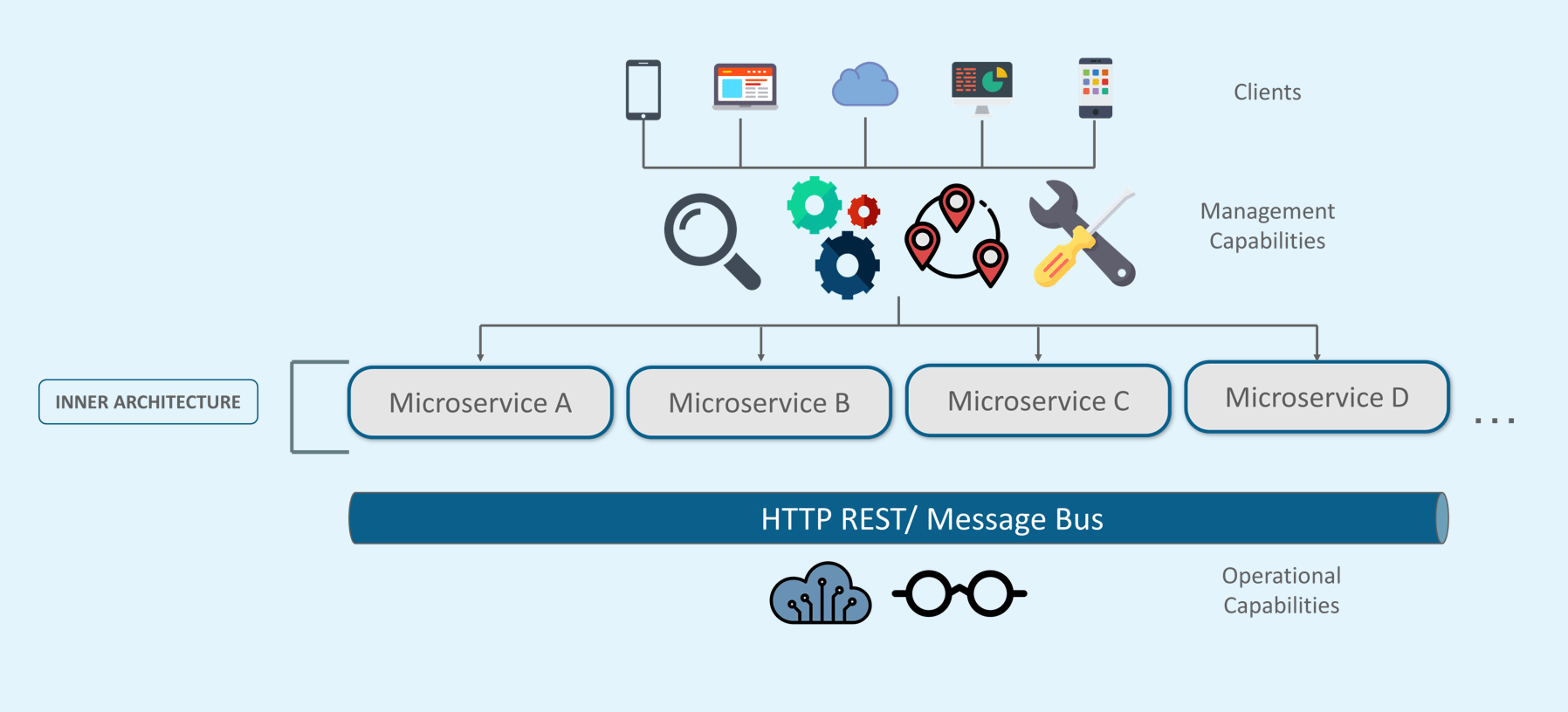
## ****Differences Between Traditional Architecture and Microservices****

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.

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



* 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 and  are further allotted to individual microservices
* These microservices have their own **load balancer** and **execution environment** to execute their functionalities & at the same time captures data in their own databases
* All the microservices communicate with each other through a stateless server which is either **REST** or **Message Bus**
* Microservices know their path of communication with the help of **Service Discovery**and perform operational capabilities such as automation, monitoring
* 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****

## Microservices Features - What Is Microservices - Edureka

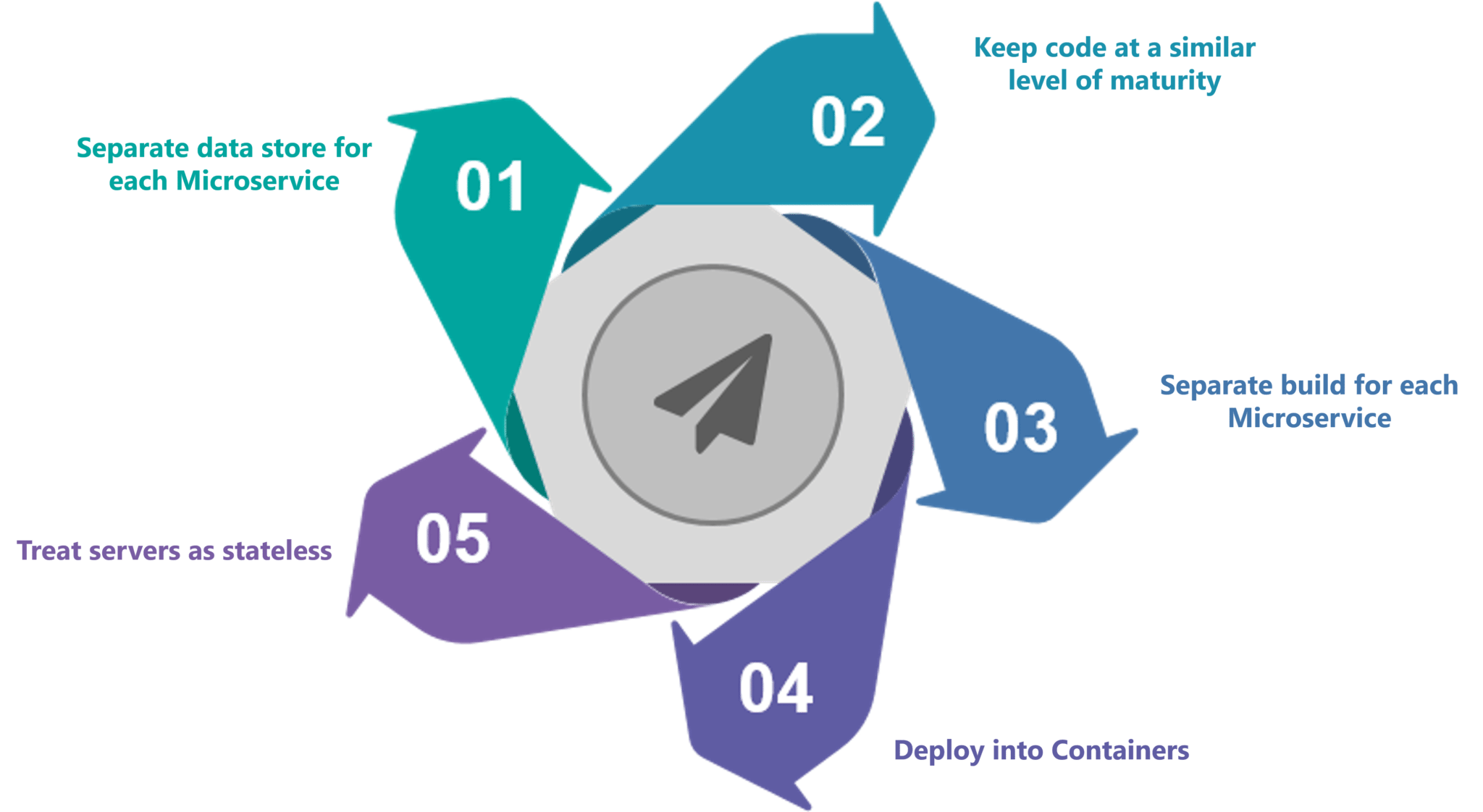
* **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
* **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
* **Business Capabilities** – Microservices are very simple and focus on a single capability
* **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
* **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
* **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
* **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:****

## Advantages Of Microservices - What Is Microservices - Edureka

* **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

**The following are the best practices to design microservices:**



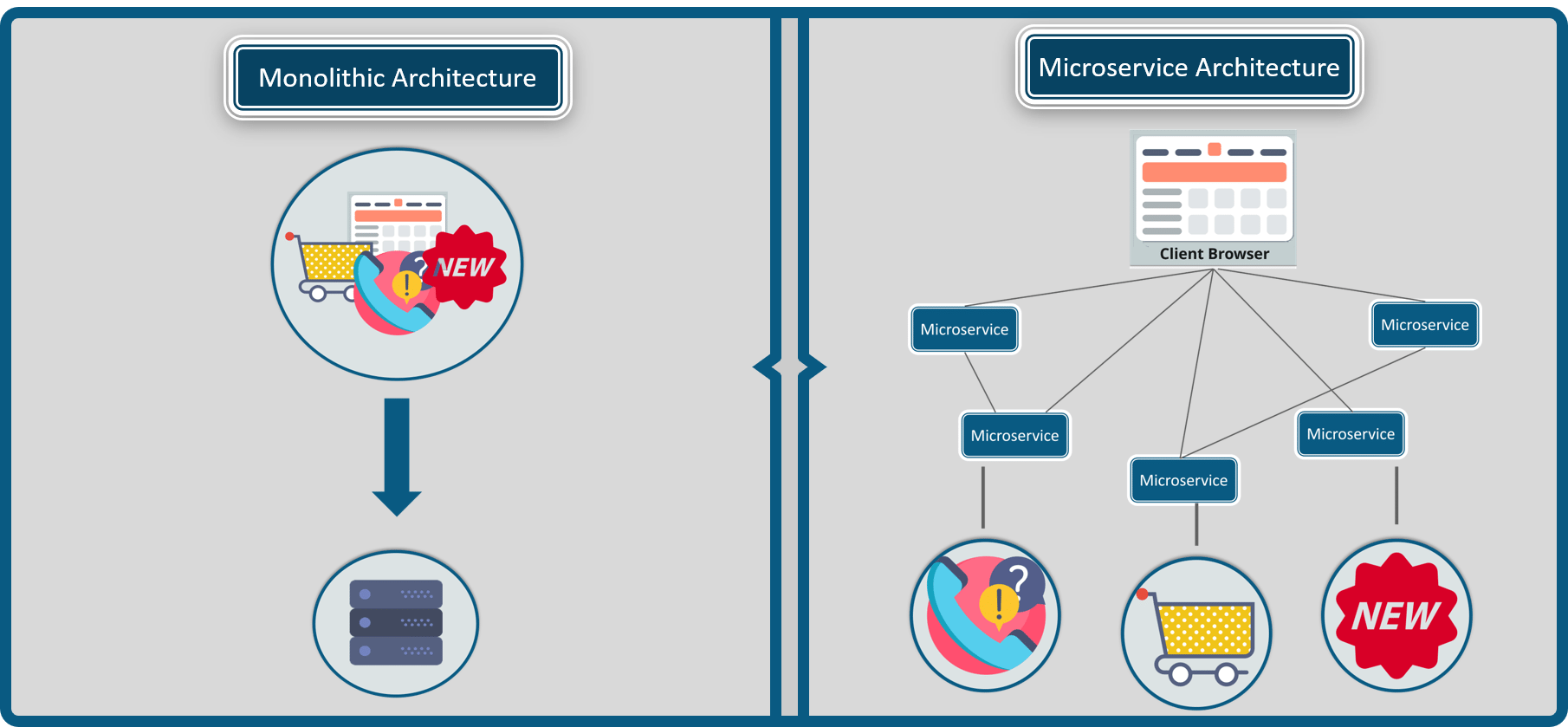
# PART II:

# Microservice Architecture – Learn, Build and Deploy Microservices

## Definition of Microservices:

*As such, there is no proper definition of Microservices aka Microservice Architecture, but you can say that it is a framework which consists of small, individually deployable services performing different operations.*

Microservices focus on a single business domain 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

* As a developer, when you decide to build an application separate the domains and be clear with the functionalities.
* Each microservice you design shall concentrate only on one service of the application.
* Ensure that you have designed the application in such a way that each service is individually deployable.
* Make sure that the communication between microservices is done via a stateless server.
* Each service can be furthered refactored into smaller services, having their own microservices.

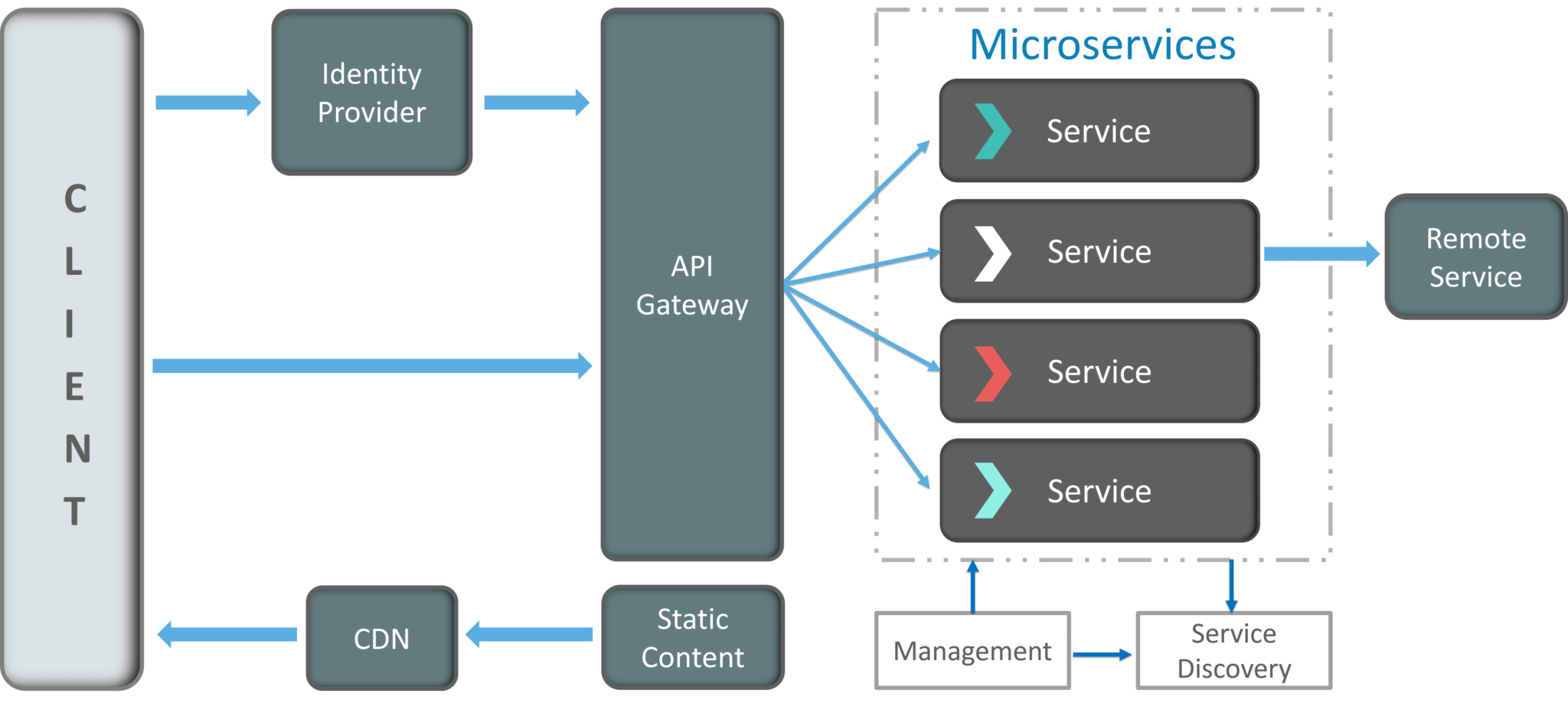
Now, that you have read through the basic guidelines while designing microservices, let’s understand the architecture of microservices.

## How Does Microservice Architecture Work?

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

* **Clients**
* **Identity Providers**
* **API Gateway**
* **Messaging Formats**
* **Databases**
* **Static Content**
* **Management**
* **Service Discovery**

Refer to the diagram below:



Though 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.

**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.

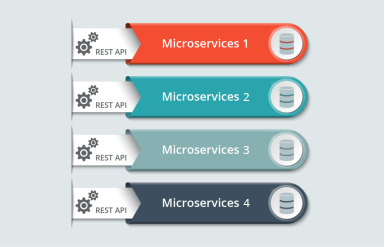
**4. Messaging Formats**

There are two types of messages through which they communicate:

* **Synchronous Messages:** In the situation where clients wait for the responses from a service, Microservices usually tend to use **REST (Representational State Transfer)** as it relies on a stateless, client-server, and the **HTTP protocol**. This protocol is used as it is a distributed environment each and every functionality is represented with a resource to carry out operations
* **Asynchronous Messages:** In the situation where clients do not wait for the responses from a service, Microservices usually tend to use protocols such as **AMQP, STOMP, 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.

**5. Data Handling**

* Well, each Microservice owns a private database to capture their data and implement the respective business functionality. Also, the databases of Microservices are updated through their service API only. Refer to the diagram below:



The services provided by Microservices are carried forward to any remote service which supports inter-process communication for different technology stacks.

**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**

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**

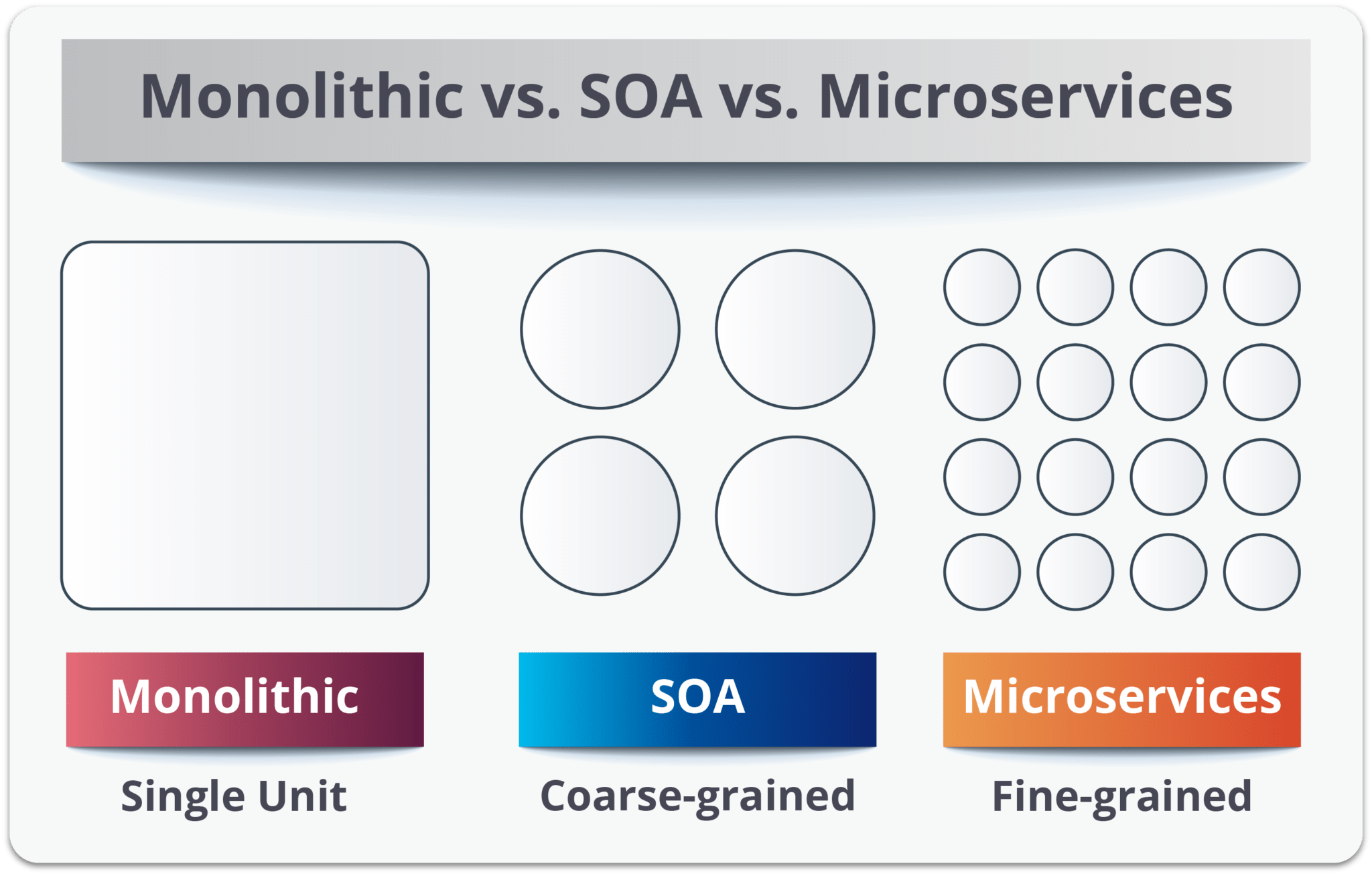
Refer to the table below.

|  |  |
| --- | --- |
| **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 parallelly developed and deployed | Difficult to move code between services |

**PART III:**

# Microservices vs SOA: What’s the Difference

So, before we deep dive into the differences between the Microservices vs SOA, let me just tell you the basic differences between the Monolithic architecture, SOA, and Microservices.



In layman terms, **Monolithic** is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.

A **Service-Oriented Architecture** is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

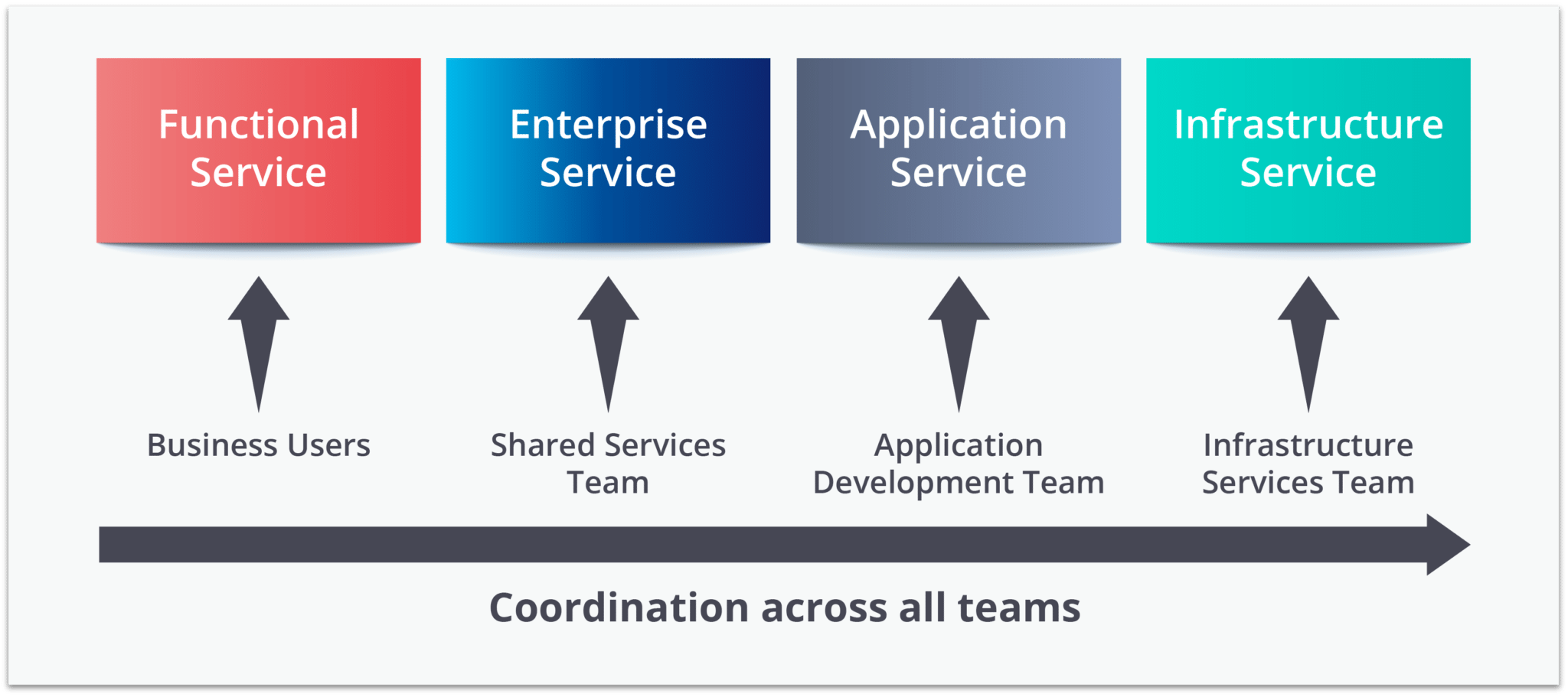
**Microservices**, aka [Microservice Architecture](https://www.edureka.co/blog/microservice-architecture/), is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.

## ****Microservices vs SOA:****

When comparing Microservices vs SOA, they both rely on services as the main component but they vary greatly in terms of service characteristics.

## ****Service Oriented Architecture****

It defines 4 basic service types as depicted below:



**Business Services:**

* They are coarse-grained services that define core business operations.
* Represented through XML, Business Process Execution Language (BPEL) and others.

**Enterprise Services:**

* Implements the functionality defined by business services.
* They mainly rely on application services and infrastructure services to fulfill business requests.

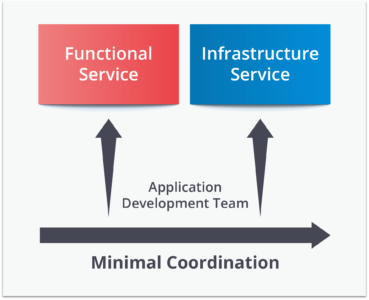
**Application Services**

* Fine-grained services that are confined to a specific application context.
* A dedicated user interface can directly invoke the services.

**Infrastructure Services**

* Implements non-functional tasks such as authentication, auditing, security, and logging.
* They can be invoked from either application services or enterprise services.

**Microservices** have limited service taxonomy. They consist of 2 service types as depicted below.

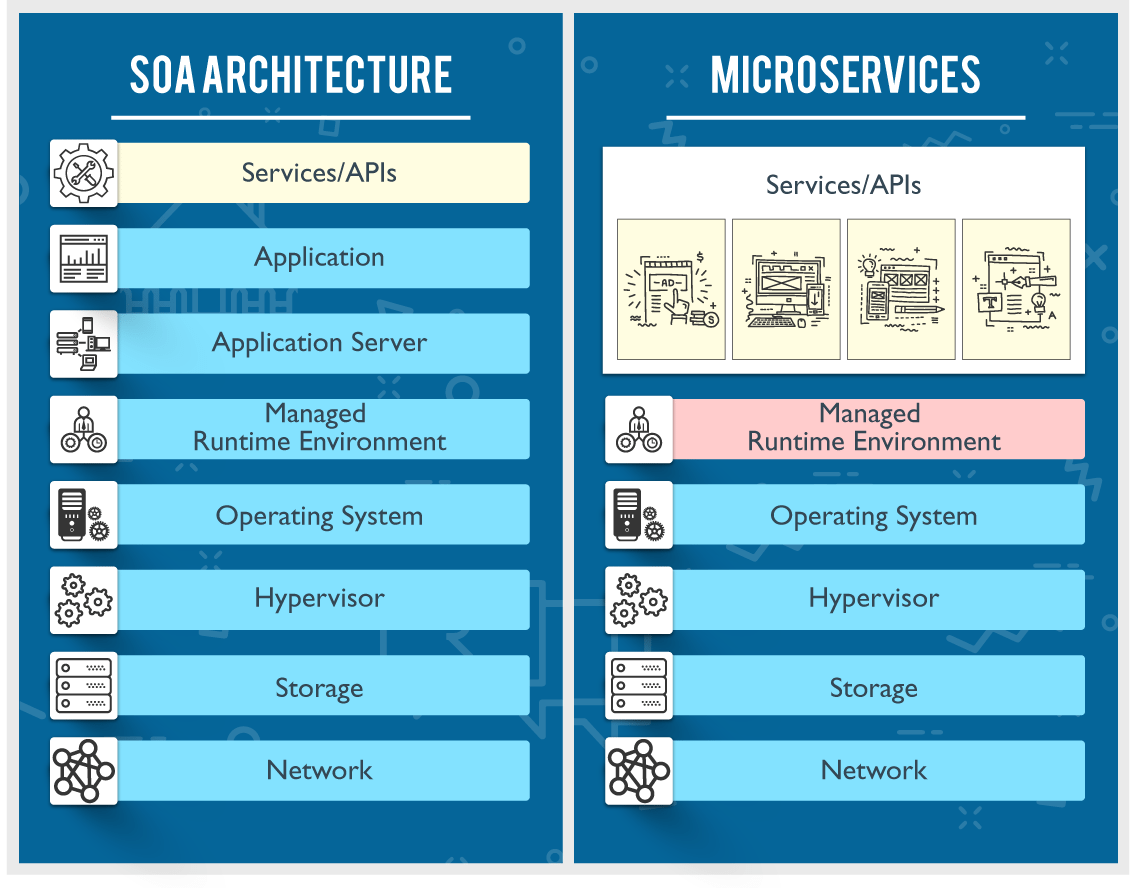


**Functional Services:**

* They Support specific business operations.
* Accessing of services is done externally and these services are not shared with other services.
* As in SOA, infrastructure services implement tasks such as auditing, security, and logging.
* In this, the services are not unveiled to the outside world.

**Major differences between SOA and MSA:**

* **Service Granularity**: Service components within microservices architecture are generally single-purpose services that do one thing really, really well. With SOA, service components can range in size anywhere from small application services to very large enterprise services. In fact, it is common to have a service component within SOA represented by a large product or even a subsystem.
* **Component Sharing**: Component sharing is one of the core tenets of SOA. As a matter of fact, component sharing is what enterprise services are all about. SOA enhances component sharing, whereas MSA tries to minimize on sharing through “bounded context.” A bounded context refers to the coupling of a component and its data as a single unit with minimal dependencies. As SOA relies on multiple services to fulfill a business request, systems built on SOA are likely to be slower than MSA.
* **Middleware vs API layer**: The microservices architecture pattern typically has what is known as an API layer, whereas SOA has a messaging middleware component. The messaging middleware in SOA offers a host of additional capabilities not found in MSA, including mediation and routing, message enhancement, message, and protocol transformation. MSA has an API layer between services and service consumers.
* **Remote services**: SOA architectures rely on messaging (AMQP, MSMQ) and SOAP as primary remote access protocols. Most MSAs rely on two protocols – REST and simple messaging (JMS, MSMQ), and the protocol found in MSA is usually homogeneous.
* **Heterogeneous interoperability**: SOA promotes the propagation of multiple heterogeneous protocols through its messaging middleware component. MSA attempts to simplify the architecture pattern by reducing the number of choices for integration. If you would like to integrate several systems using different protocols in a heterogeneous environment, you need to consider SOA. If all your services could be exposed and accessed through the same remote access protocol, then MSA is a better option.



| **SOA** | **MSA** |
| --- | --- |
| Follows “**share-as-much-as-possible**” architecture approach | Follows “**share-as-little-as-possible**” architecture approach |
| Importance is on **business functionality** reuse | Importance is on the concept of “**bounded context**” |
| They have **common** **governance** and **standards** | They focus on **people** **collaboration** and freedom of other options |
| Uses **Enterprise Service bus (ESB)** for communication | Simple messaging system |
| They support **multiple message protocols** | They use **lightweight protocols** such as **HTTP/REST** etc. |
| **Multi-threaded** with more overheads to handle I/O | **Single-threaded** usually with the use of Event Loop features for non-locking I/O handling |
| Maximizes application service reusability | Focuses on **decoupling** |
| **Traditional Relational Databases** are more often used | **Modern Relational Databases**are more often used |
| A systematic change requires modifying the monolith | A systematic change is to create a new service |
| DevOps / Continuous Delivery is becoming popular, but not yet mainstream | Strong focus on DevOps / Continuous Delivery |

**Note:** *In the end, I will say it is not that simple to tell which architecture is better than other. It mainly depends on the purpose of the application you are building.****SOA****is better suited for large and complex business application environments that require integration with many heterogeneous applications i.e. smaller applications are not a good fit for SOA as they don’t need a messaging middleware component.****Microservices****, on the other hand, are better suited for smaller and well-partitioned, web-based systems in which Microservices give you much greater control as a developer. The conclusion is that since they both have different architecture characteristics but it mainly depends on the purpose of the application you are building.*

## ****PART IV: Top 10 Reasons To Learn Microservices:****

Here are my top 10 reasons to learn microservices:

* [High Paying Jobs](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#High%20Paying%20Jobs)
* [Use minimal resources with reduced the cost of ownership](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Use%20minimal%20resources%20with%20reduced%20the%20cost%20of%20ownership)
* [Promotes the best big data practices](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Promotes%20the%20best%20big%20data%20practices)
* [Reduces risk](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Reduces%20risk)
* [Provides granular scaling](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Provides%20granular%20scaling)
* [Provides high-quality code](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Provides%20high-quality%20code)
* [Offers cross-team coordination](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Offers%C2%A0cross-team%20coordination)
* [Flexibility to use various tools for the required task](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Flexibility%20to%20use%20various%20tools%20for%20the%20required%20task)
* [Provide continuous delivery](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Provide%20continuous%20delivery)
* [Easy to build and maintain applications](https://www.edureka.co/blog/top-10-reasons-to-learn-microservices/#Easy%20to%20build%20and%20maintain%20applications)

**Easy To Build & Maintain Applications:**

As and when the products built by developers become stable and are out on the market to be used by the customers, the team of developer’s splits into mainly the following activities.

* Implementation of new features
* Fixing Bugs
* Changing existing features

In such situations, if the products are based on a monolithic framework, then each and every change to the code base has to be passed through all the stages of build, maintain & deploy.

So in such situations, a microservice comes as a savior!!

Microservices resolves organizational based issues, making it easy to debug and test applications. With the help of this framework, continuous delivery, testing process and ability to deliver error-free applications improve drastically.

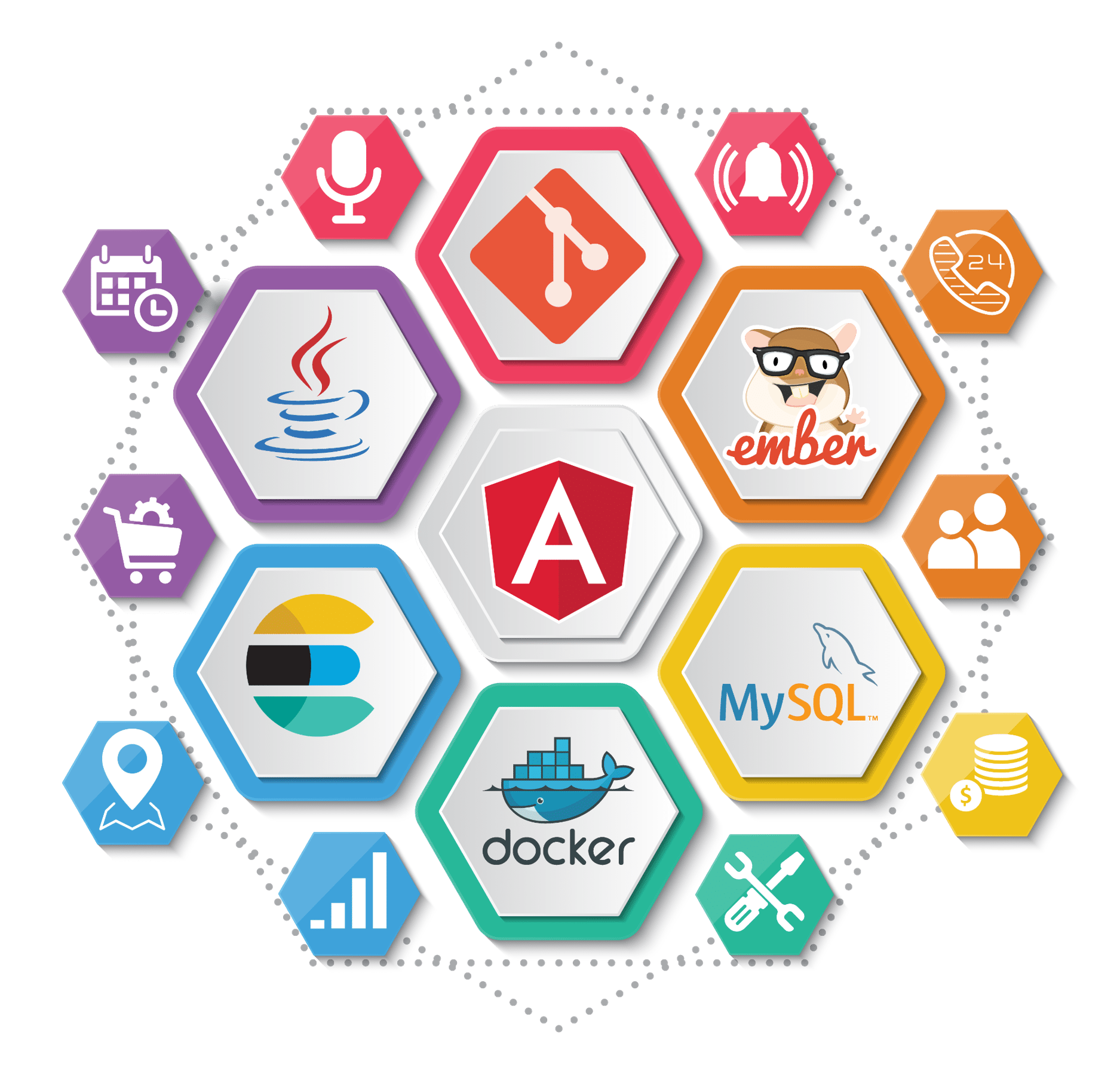
## ****Provides Continous Delivery****

Unlike monolithic applications where dedicated teams work for each discrete functions like handling database, maintaining server-side logic, microservices uses the continuous delivery model to handle the complete lifecycle of an application.

Developers, operations, testing teams work simultaneously on a single service performing activities such as building, testing and debugging.

## ****Flexibility to use various tools for the required task****

Microservices architecture encourages to use the most appropriate technology for the specific needs of the service. Each service has the freedom to use its own language, framework, or ancillary services.  Even with the use of such diverse frameworks the services still communicate easily with the other services in the application.



## ****Offers cross-team coordination****

The traditional service-oriented architectures (SOA) involve heavyweight inter-process communications protocols.

But, microservices follow the concept of decentralization and decoupling the services so that they act as separate entities. So, in Microservices Architecture each team handles various entities and then communicates with each other to handle different functionalities.

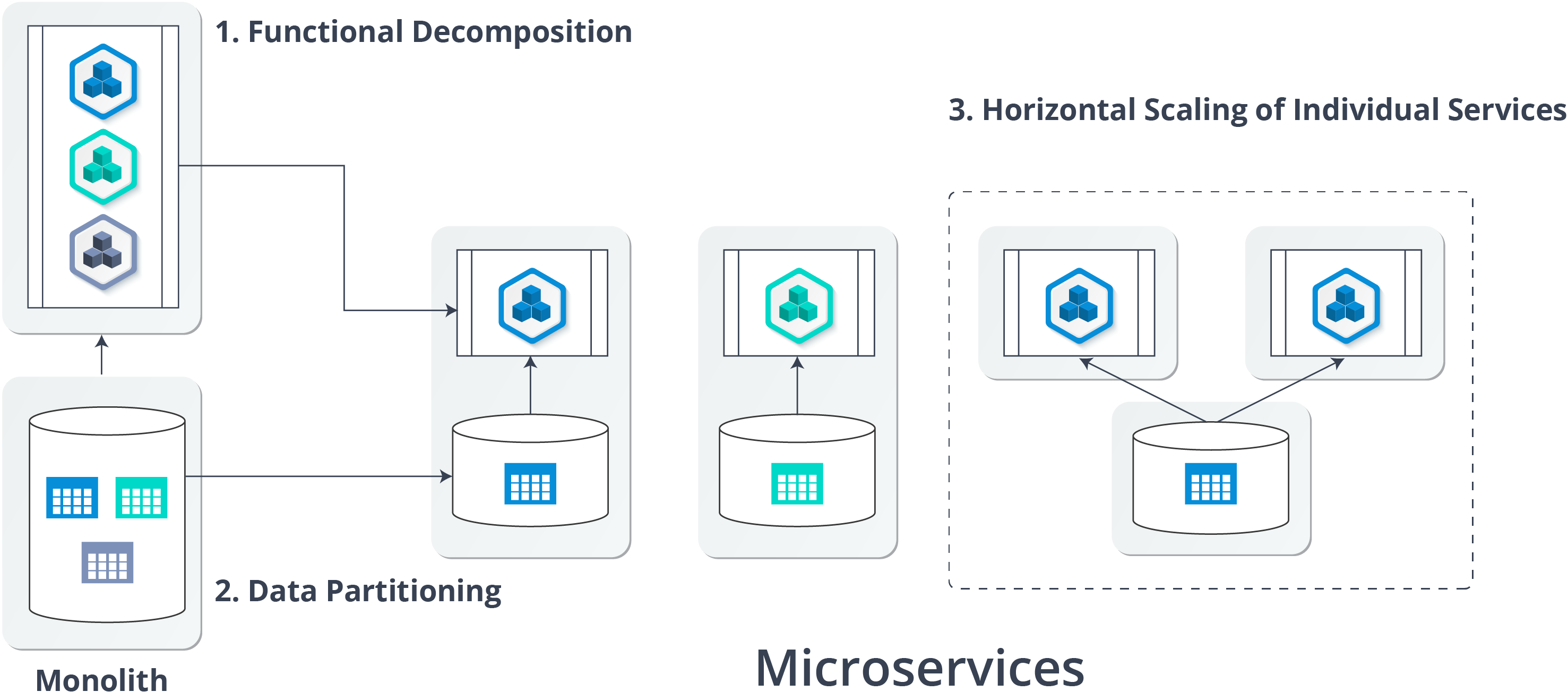
## ****Provides high-quality code****

Following the architecture of microservices, the complete framework is modularized into discrete components. This helps the application development team focus on one particular job at a time. So, this, in turn, simplifies the overall coding and testing process.

## ****Provides granular scaling****

If you talk about scalability, then microservices outperforms many other architectural choices out there.

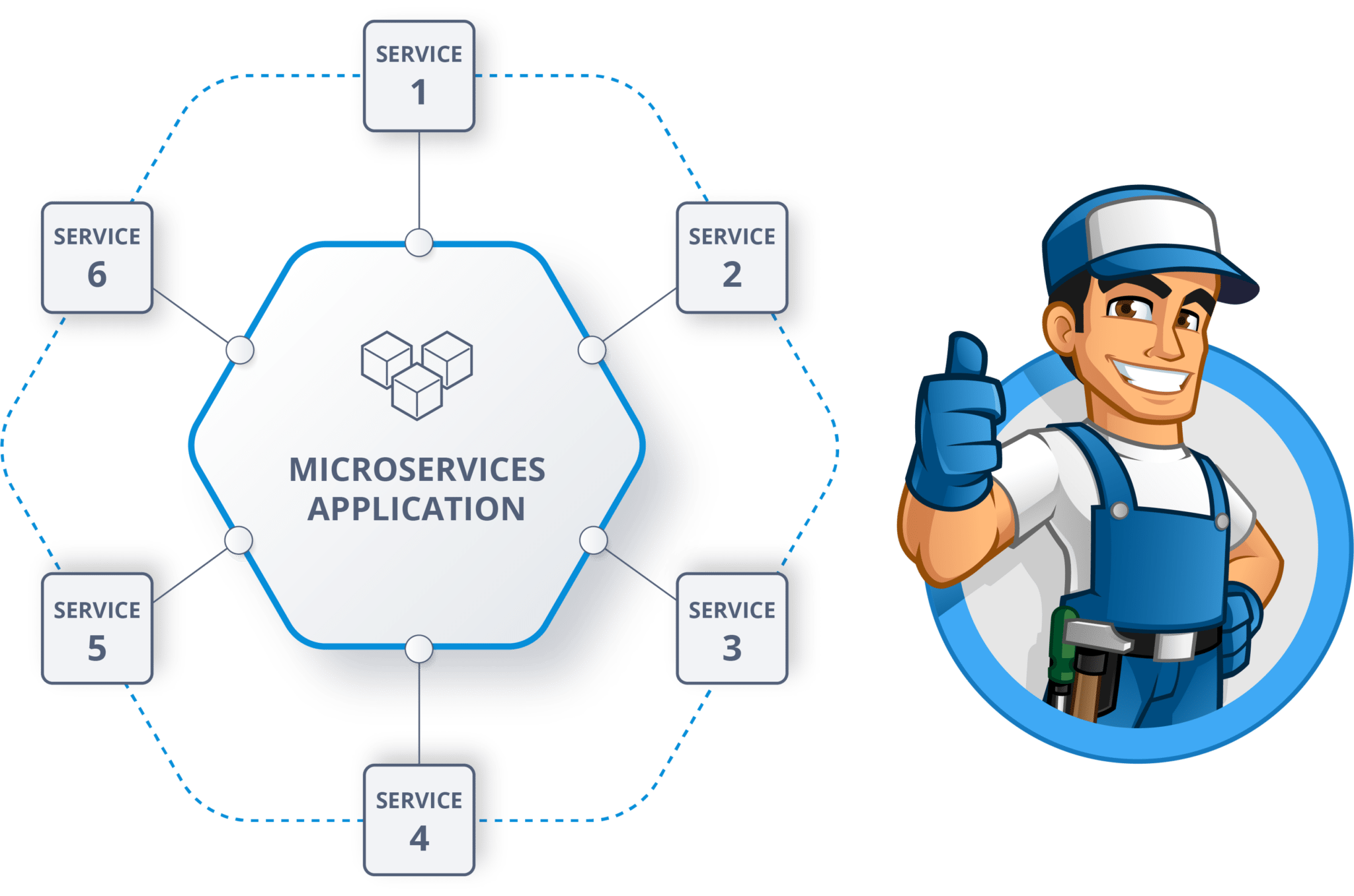
As each service is a separate component in the framework, you can scale up a single function or service without having to scale the entire application. Business-critical services can be deployed on multiple servers for increased availability and performance without impacting the performance of other services.



So, Microservices make it easy to identify scaling bottlenecks and then resolve those bottlenecks at a per-microservice level.

## ****Reduces risk****

Each service is a separate entity in the microservices framework, and this allows localized changes, higher confidence in the quality and end-to-end regression scenarios.



So, even if one service or component of the application is down, then, the complete application doesn’t go down. Instead, only that service or component particular needs to be rebuilt by the developers.

Therefore, this reduces the risk of complete fall of your business application!!

## ****Promotes Big Data Practices****

Microservices own their private databases to collect, ingest, process and deliver the data to implement their respective business functionality.

## ****Use minimal resources with reduced the cost of ownership****

Multiple teams **work** on independent services so that they can be deployed easily. This increased efficiency of microservices reduces infrastructure costs, minimizes downtime, optimizes the resources and makes the code reusable. So, with the help of these services, you don’t have to operate on extensive machines, but basic machines will do for you.

# Spring Boot Microservices: Building Microservices Application Using Spring Boot

## ****Why there is a need for Spring Boot?****

Spring Boot enables building production-ready applications quickly and provides non-functional features:

* Embedded servers which are easy to deploy with the containers
* It helps in monitoring the multiples components
* It helps in configuring the components externally

So, let us see what the challenges with Microservices Architecture are

## ****Challenges with Microservice Architectures****

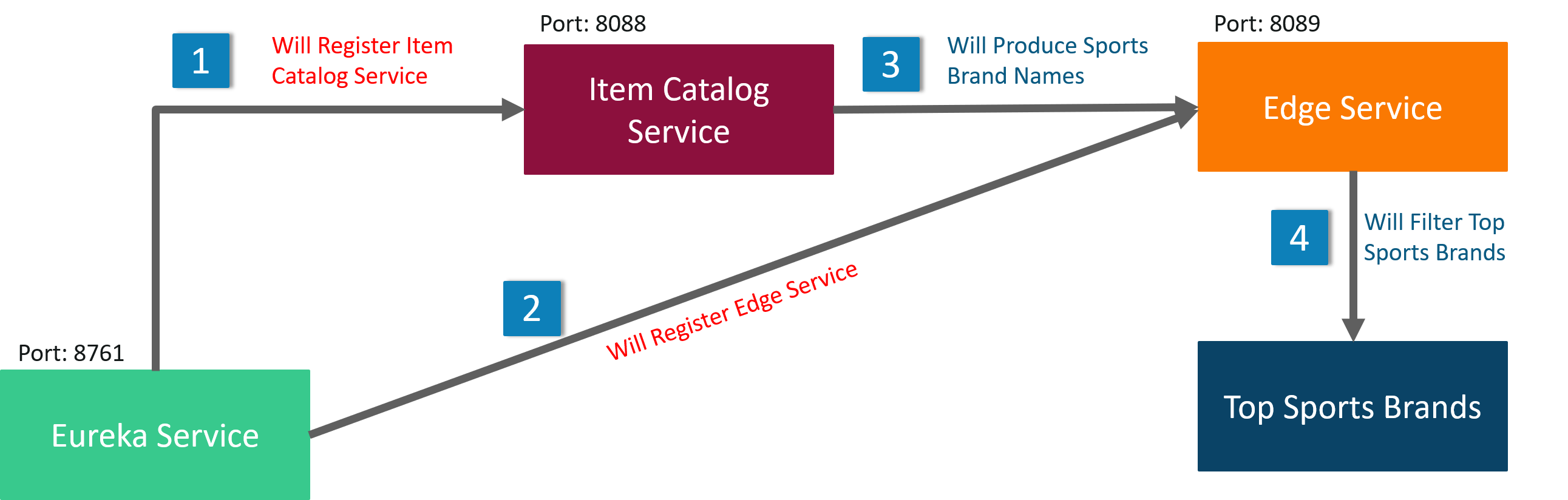
While developing a number of smaller microservices might look easy, but there are number of inherent complexities that are associated with microservices architectures. Let’s look at some of the challenges:

* **Automating the Components:** It becomes difficult to automate everything because there are a number of smaller components instead of a monolith, i.e. Builds, Deployment, Monitoring, etc.
* **Perceptibility:** There are number of small components to deploy and maintain which sometimes becomes difficult to monitor and identify problems. It requires great perceptibility around all the components.
* **Configuration Management:** There is a great need to maintain the configurations for the components across the various environments.
* **Debugging: It** becomes difficult to probe each and every service for an error. Centralized Logging and Dashboards are essential to make it easy to debug problems.
* **Consistency:** You cannot have a wide range of tools solving the same problem. While it is important to foster innovation, it is also important to have some decentralized governance around the languages, platforms, technology and tools used for implementing/deploying/monitoring microservices.

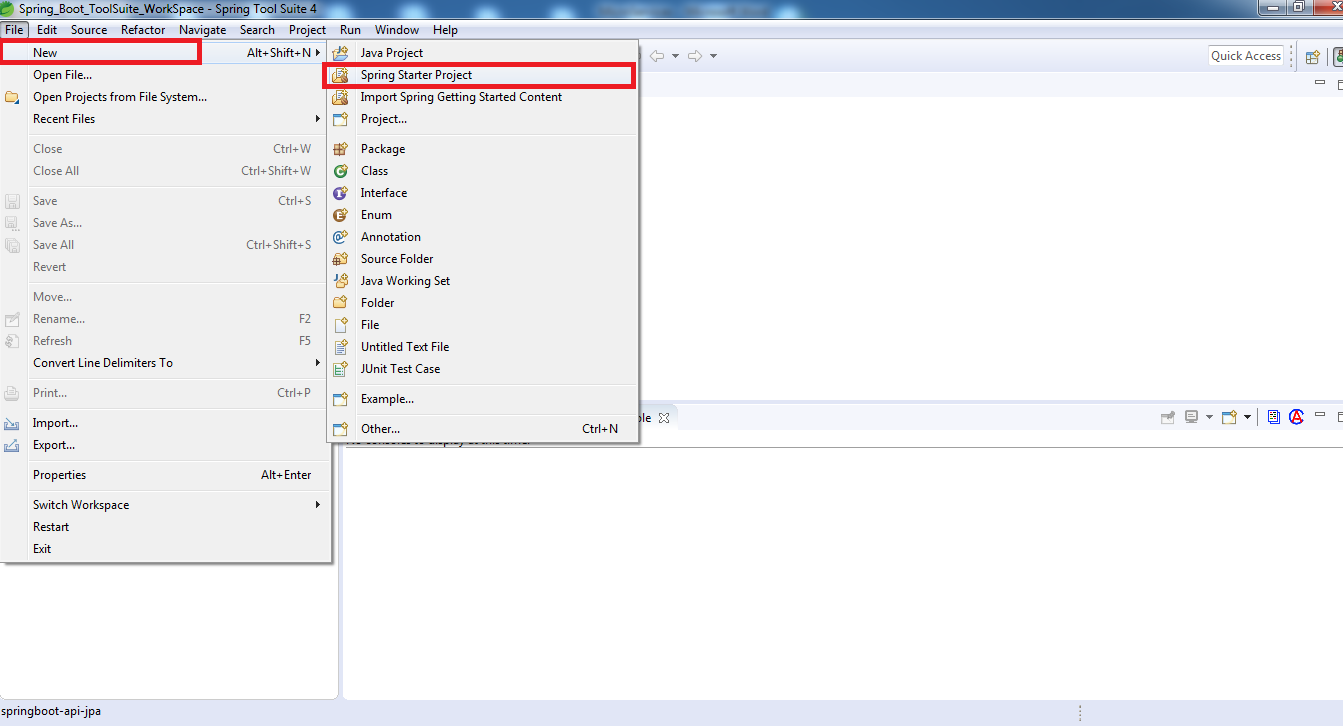
## ****Spring Boot Microservices: Building a Top Sports Brands Architecture with Spring Boot:****

In this spring boot microservices example, we will be creating Top Sports Brands application which will be having 3 services:-

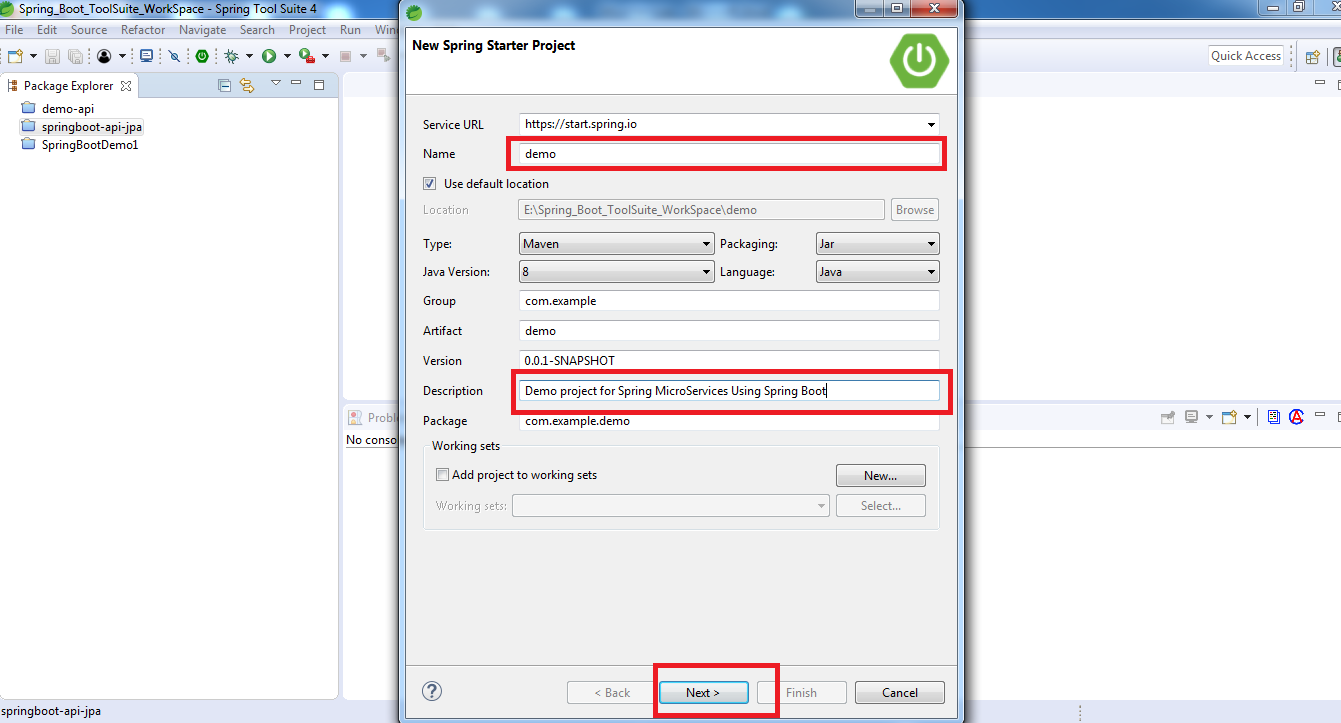
* **Eureka Service**– This Service will register every microservice and then the client microservice will look up the Eureka server to get a dependent microservice to get the job done. This Eureka Server is owned by Netflix and in this, Spring Cloud offers a declarative way to register and invoke services by Java annotation.
* **Item Catalog Service –**This service will generate the list of Sports brands which are popular in the market.
* **Edge Service** – It is similar to the standalone Item service created in Bootiful Development with Spring Boot and Angular. However, it will have fallback capabilities which prevent the client from receiving an HTTP error when the service is not available



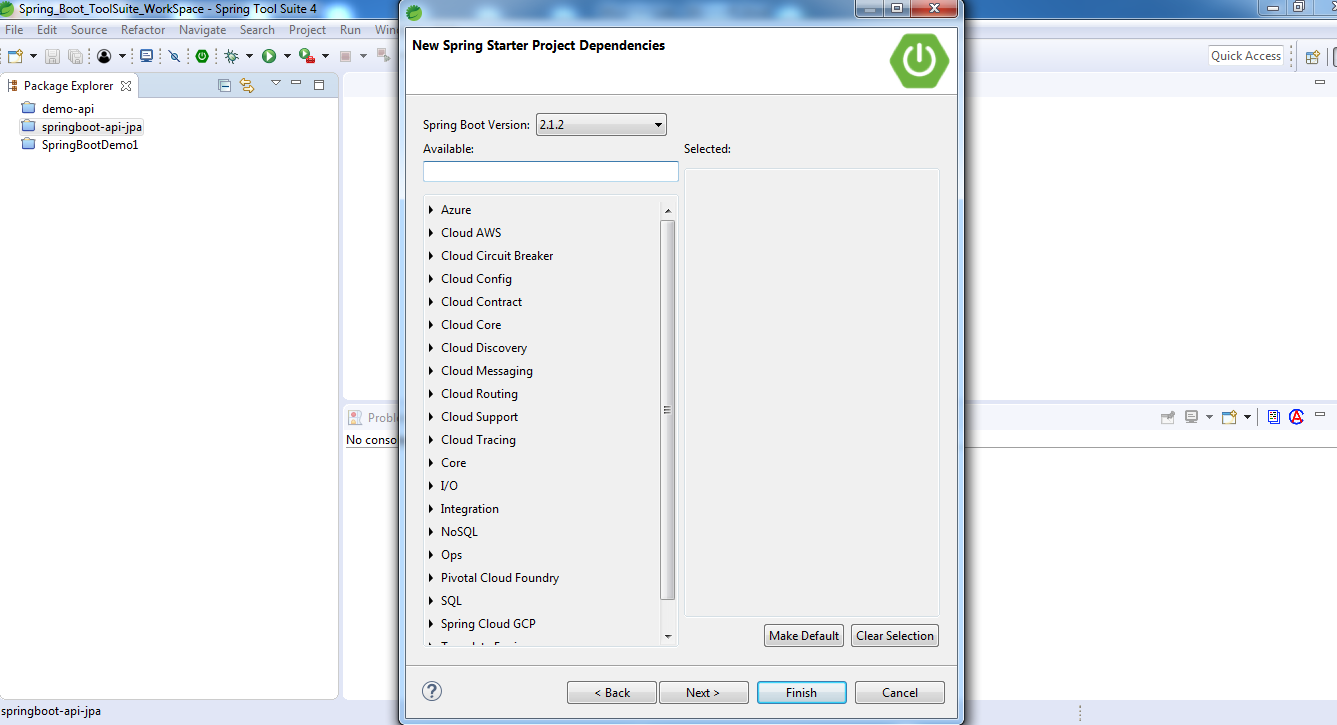
Step1:



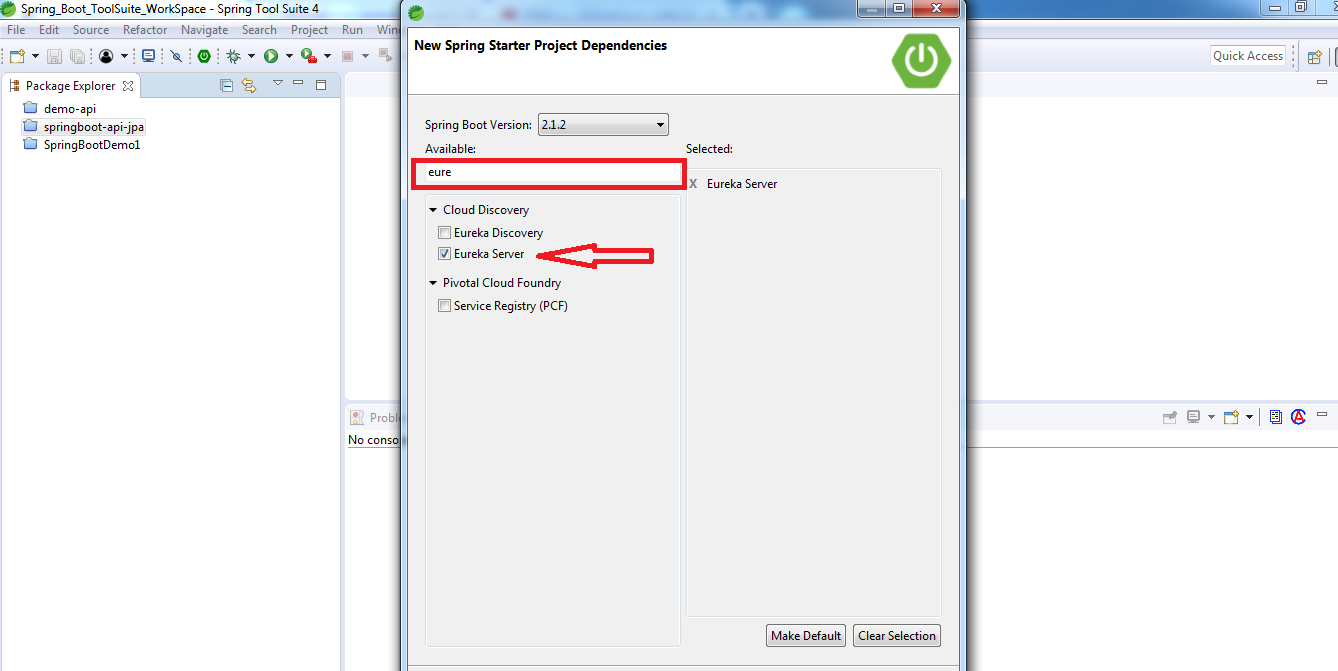
Step2:



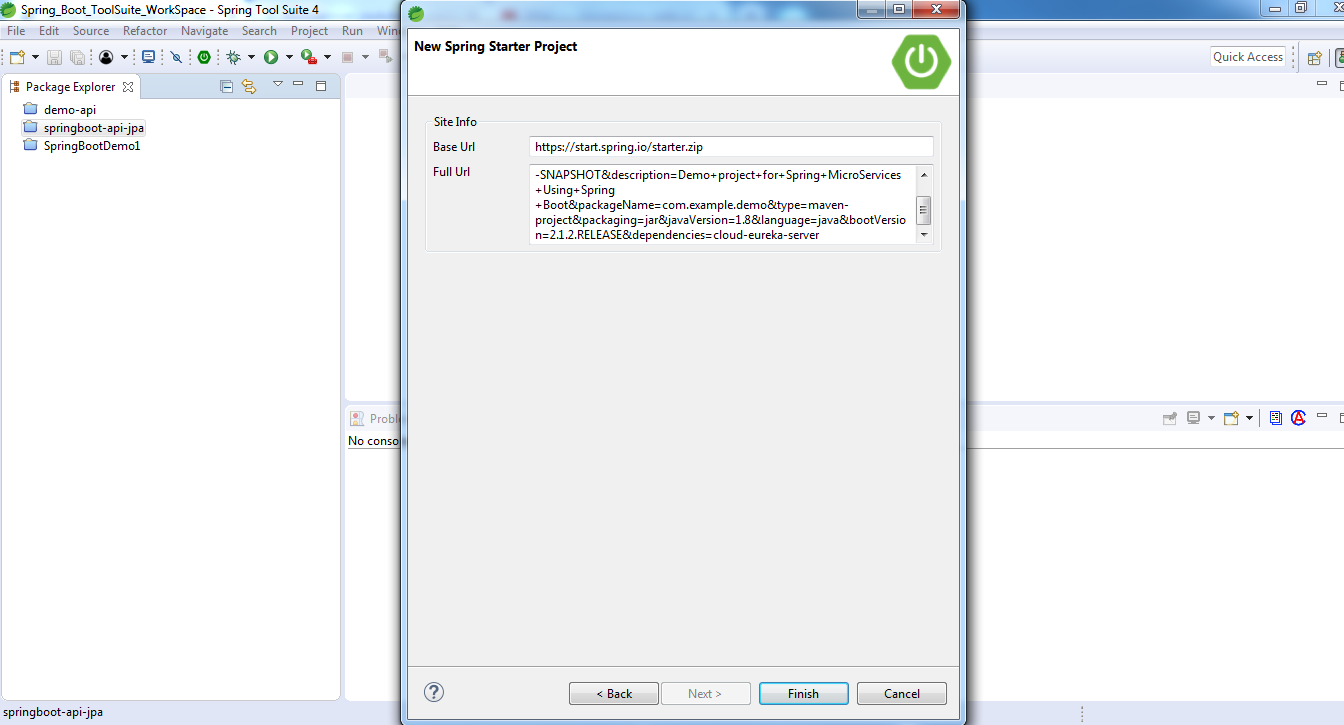
Step3:



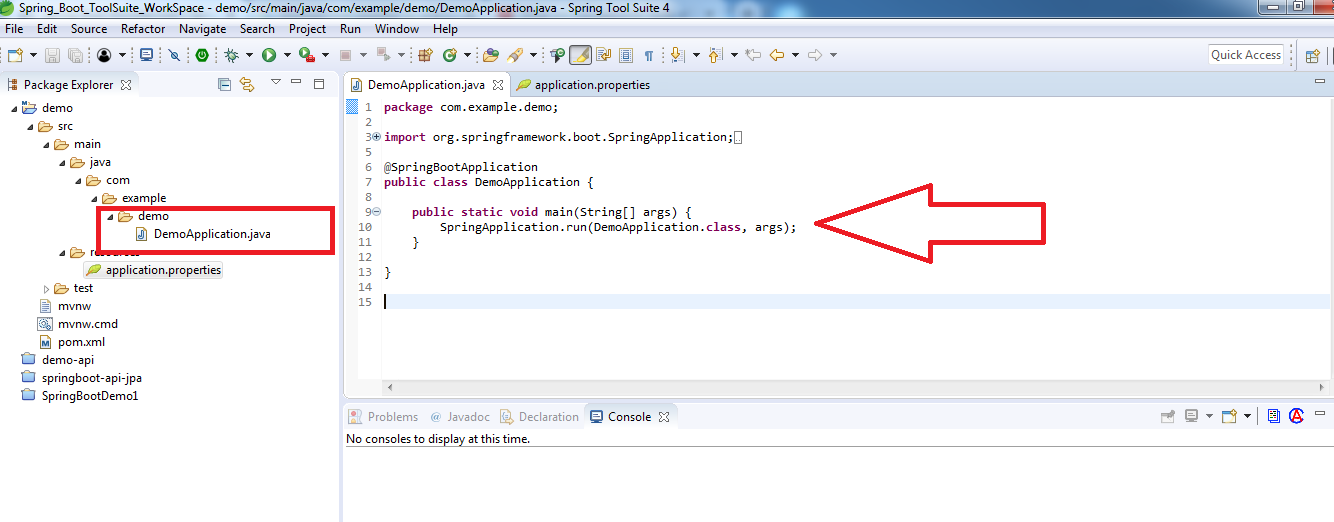
Step4:



Step5:



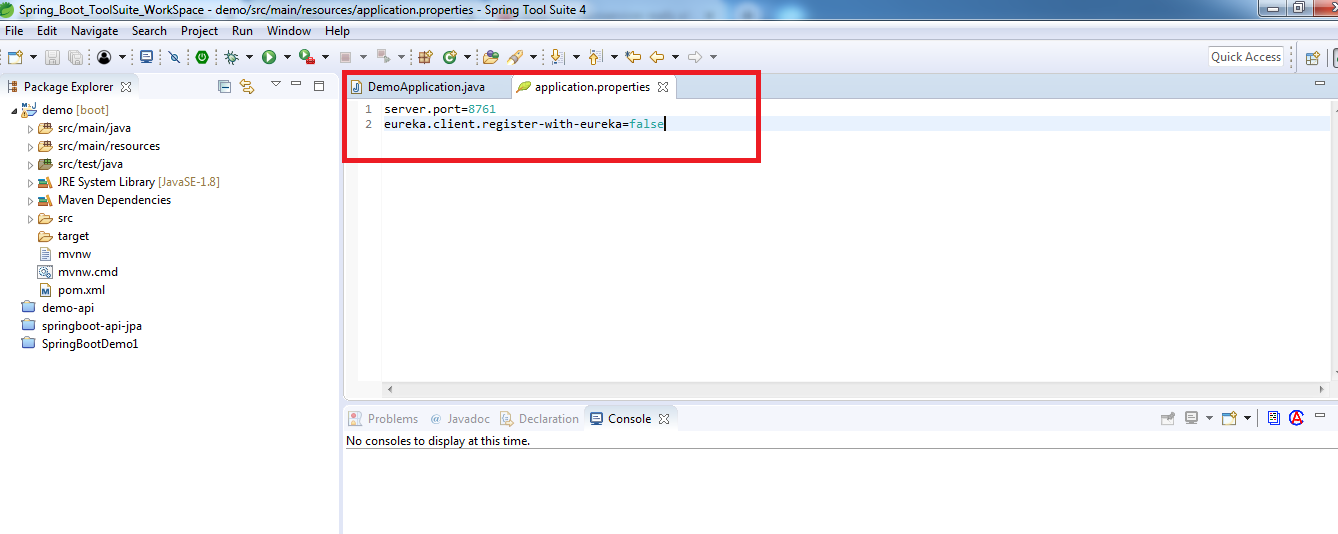
Step6:



Step6:

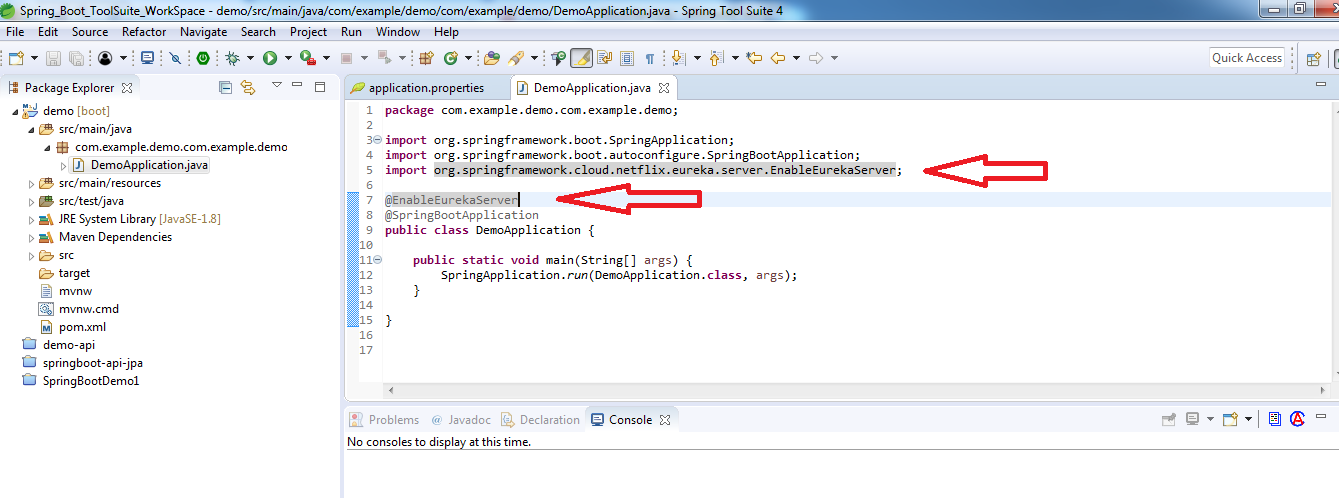
Now, modify demo/src/main/resources/application.properties file to add a port number and disable registration.

server.port=8761  
eureka.client.register-with-eureka=false

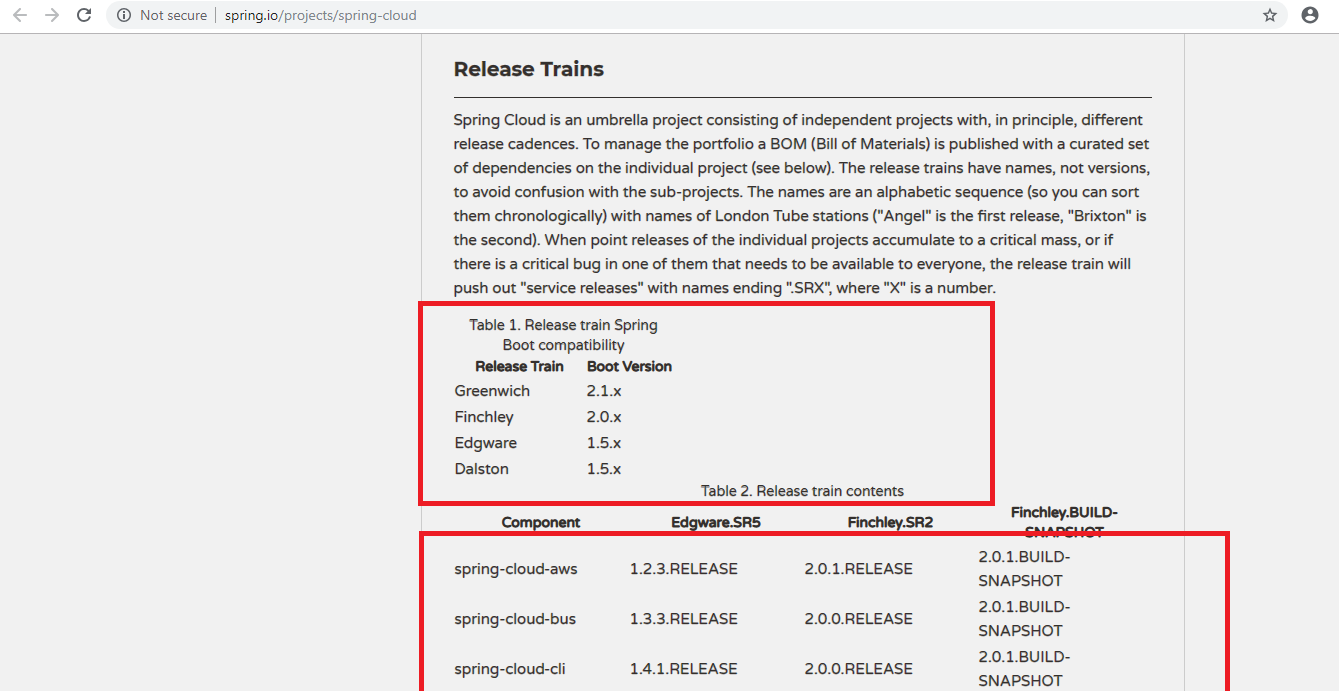


Step7:

Open demo/src/main/java/com/example/EurekaServiceApplication.java and add @EnableEurekaServer above @SpringBootApplication.



To start the Application: Right Click on the **Project** –> **Run As** –> Click on “**Spring Boot App**“



|  |
| --- |
|  |