Micro Services

Table of Contents

[Introduction 2](#_Toc143728397)

[What is Monolithic Architecture? 2](#_Toc143728398)

[Advantages 2](#_Toc143728399)

[Disadvantages 2](#_Toc143728400)

[What are Micro Services? 3](#_Toc143728401)

[Advantages 3](#_Toc143728402)

[Micro Services Architecture 3](#_Toc143728403)

[Micro-services features 4](#_Toc143728404)

[Micro-services components 4](#_Toc143728405)

[Management/Orchestration 4](#_Toc143728406)

[API Gateway 5](#_Toc143728407)

[Micro-service’s communication 5](#_Toc143728408)

[In Monolithic Application 5](#_Toc143728409)

[Micro Service Application 5](#_Toc143728410)

[Sample Application – Micro 5](#_Toc143728411)

[Req1 6](#_Toc143728412)

[Req 2 6](#_Toc143728413)

[MSA v/s SOA 7](#_Toc143728414)

[Micro services Monitoring 8](#_Toc143728415)

[Micro services Monitoring Tool 8](#_Toc143728416)

[Micro service Virtualization 8](#_Toc143728417)

[Components of Micro service 8](#_Toc143728418)

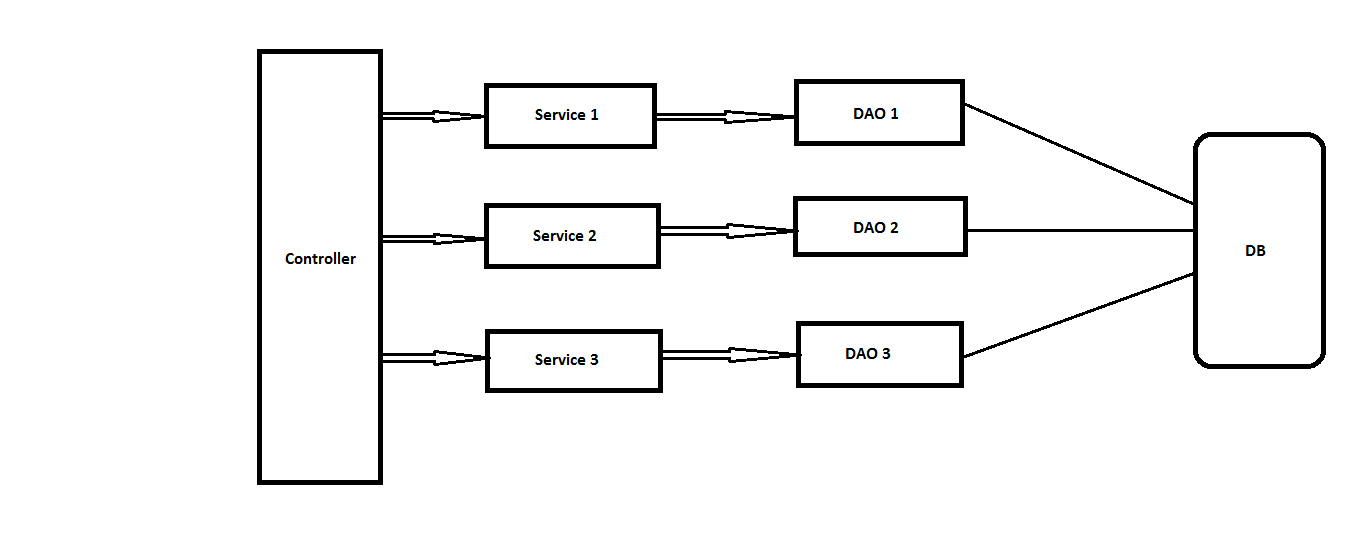
# Introduction

Before learning micro-services, let us understand the problem areas that software development teams faced that led to micro-service as a solution.

Before micro-services based application, monolithic applications were in rage.

# What is Monolithic Architecture?

Monolithic application has single code base with multiple modules



This application is bundled in a **single JAR files** and these JAR files are deployed on a Server

A monolithic application may exist on single or multiple servers.

## Advantages

* Simple to develop
* Simple to build and deploy
* Easier Testing and Error Tracing
* Problems of network latency are relatively less

## Disadvantages

* As project scales it becomes difficult to manage
* For single change re-deployment of whole application needed
* Difficult to adopt new technology for single functionality
* Single bug may down your whole application
* Tight coupling between components, as everything is in one application
* Less reusability
* Large code base; tough for developers and QA to understand the code and business knowledge
* More deployment and restart times

# What are Micro Services?

* Micro services are the small services that work together
* These smaller services communicate with each other directly using light weight protocols like HTTP

## Advantages

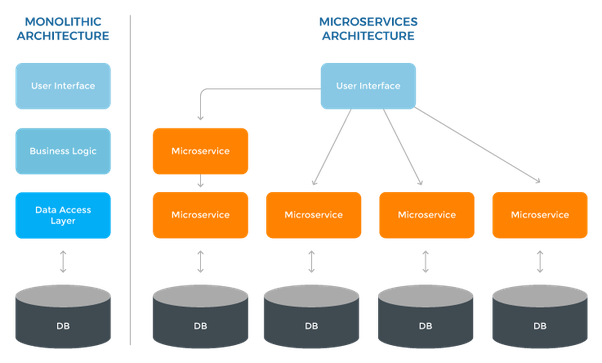
* Improved Scalability
* Less dependency
* Loosely coupled
* It is possible to change or upgrade each service individually rather than upgrading the entire application
* One service may be down but it will not impact others
* Programming Language and Technology Agnostic
* Easily use different technology for building different micro services
* Greater Business Agility and Support for DevOps

# Micro Services Architecture

Micro services architecture consists of a collection of small, autonomous services.

Each service is self-contained and should implement a single business capability within a bounded context.

A bounded context is a natural division within a business and provides an explicit boundary within which a domain model exists.



## Micro-services features

* Micro services are small, independent, and loosely coupled.
* A single small team of developers can write and maintain a service.
* Each service is a separate codebase, which can be managed by a small development team.
* Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
* Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
* Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
* Supports **polyglot programming**. For example, services don't need to share the same technology stack, libraries, or frameworks.

## Micro-services components



Besides for the services themselves, some other components appear in typical micro services architecture:

### ****Management/Orchestration****

This component is responsible for placing services on nodes, identifying failures, rebalancing services across nodes, and so forth. Typically this component is an off-the-shelf technology such as Kubernetes, rather than something custom built.

### API Gateway

The API gateway is the **single entry/end point** for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end.

#### Advantages of API gateway

* It decouples clients from services. Services can be versioned or refactored without needing to update all of the clients.
* Services can use messaging protocols that are not web friendly, such as AMQP.
* The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing.
* Out-of-the-box policies, like for throttling, caching, transformation, or validation.

## Micro-service’s communication

### In Monolithic Application

In a monolithic application running on a single process, components invoke one another using language-level method or function calls. These can be strongly coupled if you're creating objects with code (for example, new ClassName()), or can be invoked in a decoupled way if you're using Dependency Injection by referencing abstractions rather than concrete object instances. Either way, the objects are running within the same process. The biggest challenge when changing from a monolithic application to a micro-services based application lies in changing the communication mechanism.

### Micro Service Application

A micro services-based application is a distributed system running on multiple processes or services, usually even across multiple servers or hosts. Each service instance is typically a process.

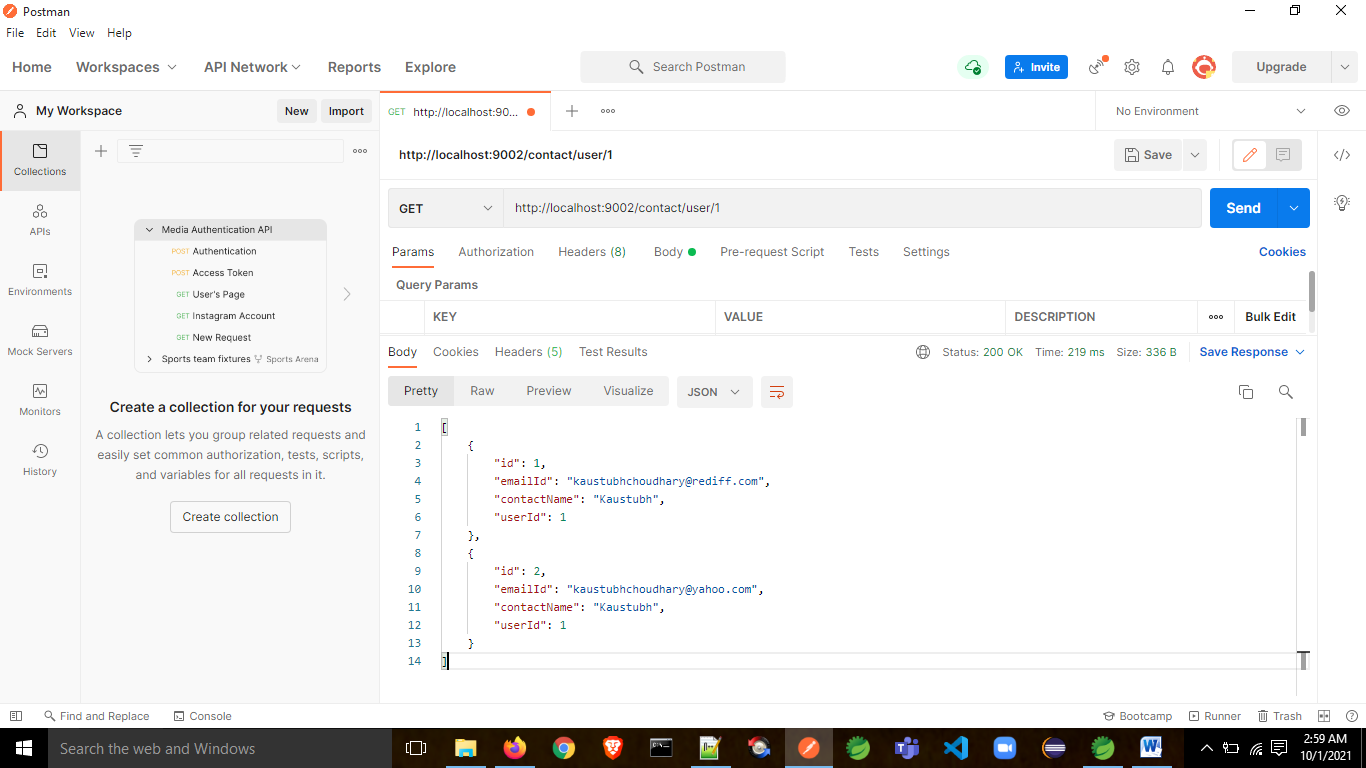
Therefore, services must interact using an inter-process communication protocol such as HTTP, AMQP, or a binary protocol like TCP, depending on the nature of each service.

The two commonly used protocols are **HTTP request/response** with **resource APIs** (when querying most of all), and **lightweight asynchronous messaging** when communicating updates across multiple micro services.

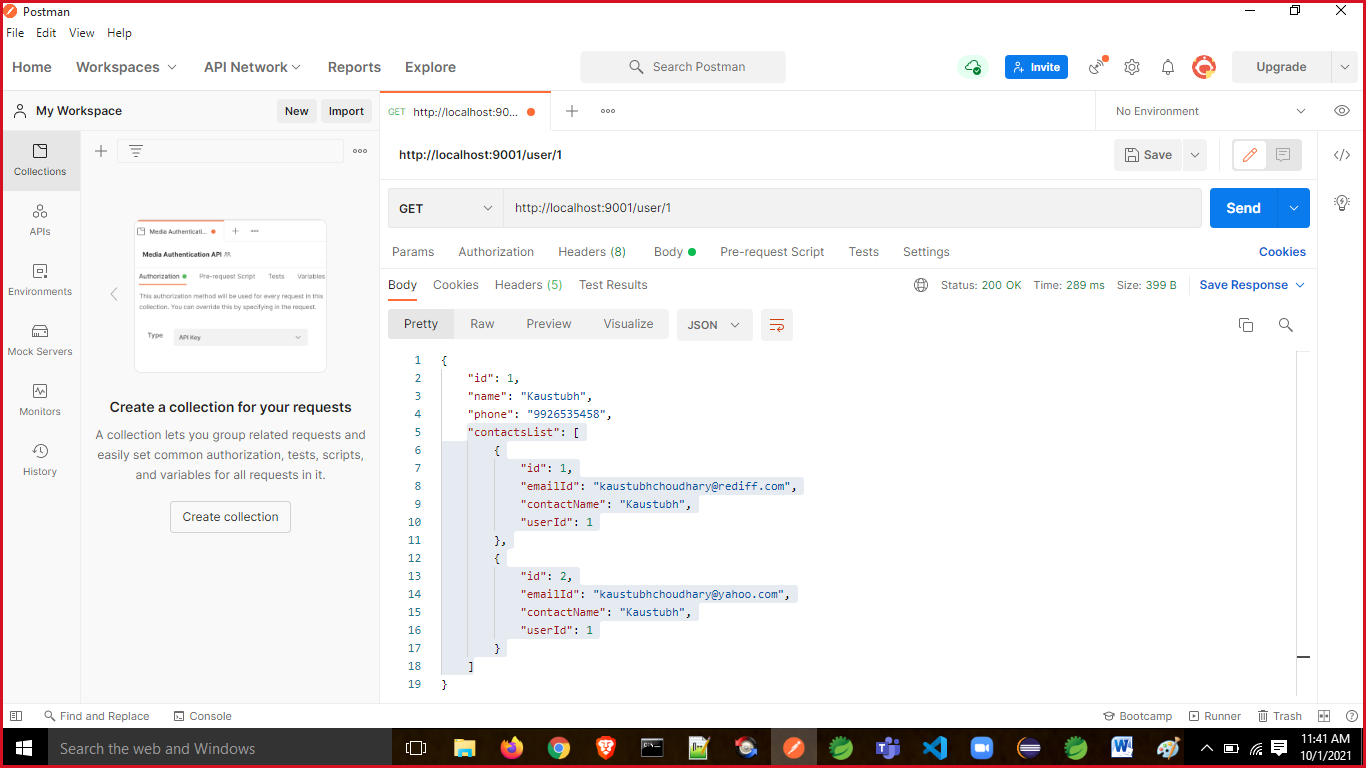
# Sample Application – Micro

* The Application folder **micro** contains 2 modules and 1 Eureka Server (to register the other services)
* Run Eureka Service 1st and then both the modules one by one
* Open Postman and make a Get Request for URL: <http://localhost:9002/contact/user/1>
* Make another request for URL: <http://localhost:9001/user/1>

### Req1



### Req 2



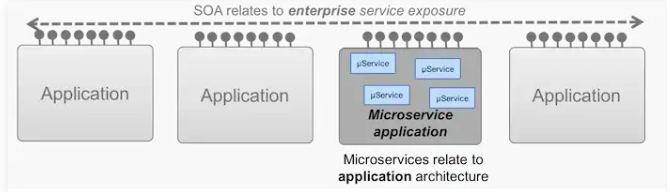
# MSA v/s SOA

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Micro Service Architecture (MSA)** | **Service-Oriented Architecture (SOA)** |
| 1. | Micro services uses lightweight protocols such as REST, and HTTP, etc. | SOA supports multi-message protocols. |
| 2. | It focuses on decoupling/loose coupling. | It focuses on application service reusability. |
| 3. | It uses a simple messaging system for communication. | It uses Enterprise Service Bus (ESB) for communication. |
| 4. | Micro services follow "share as little as possible" architecture approach. | SOA follows "share as much as possible architecture" approach. |
| 5. | Micro services are much better in fault tolerance in comparison to SOA. | SOA is not better in fault tolerance in comparison to MSA. |
| 6. | Each micro service has an independent database. | SOA services share the whole data storage. |
| 7. | MSA used modern relational databases. | SOA used traditional relational databases. |
| 8. | MSA tries to minimize sharing through bounded context (the coupling of components and its data as a single unit with minimal dependencies). | SOA enhances component sharing. |
| 9. | It is better suited for the smaller and well portioned, web-based system. | It is better for a large and complex business application environment. |

The main difference between SOA and micro services: **Scope**

The main distinction between the two approaches comes down to **scope**.

* To put it simply, service-oriented architecture (SOA) has an enterprise scope
* While the micro-services architecture has an application scope.



# Micro services Monitoring

Monitoring is the control system of the micro services. As the micro services are more complex and harder to understand its performance and troubleshoot the problems. Given the vivid changes to software delivery, it is required to monitor the service. There are **five** principles of monitoring micro services, as follows:

* Monitor container and what's inside them.
* Alert on service performance.
* Monitor services that are elastic and multi-location.
* Monitor APIs.
* Monitor the organizational structure.

These principles allow us to address technological changes associated with the microservices and organizational changes related to them.

### Micro services Monitoring Tool

There are three monitoring tools are as follows:

1. Hystrix dashboard
2. Eureka admin dashboard
3. Spring boot admin dashboard

## Micro service Virtualization

Micro services virtualization is the method to simulate the behavior of specific components in various component-based applications like cloud-based application, SOA, and API driven architecture. Service virtualization also reduces cost and save time. By combining service virtualization, an organization can develop the application which can be delivered from various locations and dissimilar environments.

## Components of Micro service

There are the following components of micro services:

* Spring Cloud Config Server
* Netflix Eureka Naming Server
* Hystrix Server
* Netflix Zulu API Gateway Server
* Netflix Ribbon
* Zipkin Distributed Tracing Server