

# Load Balancing In Microservices



Mesut Yakut · Follow

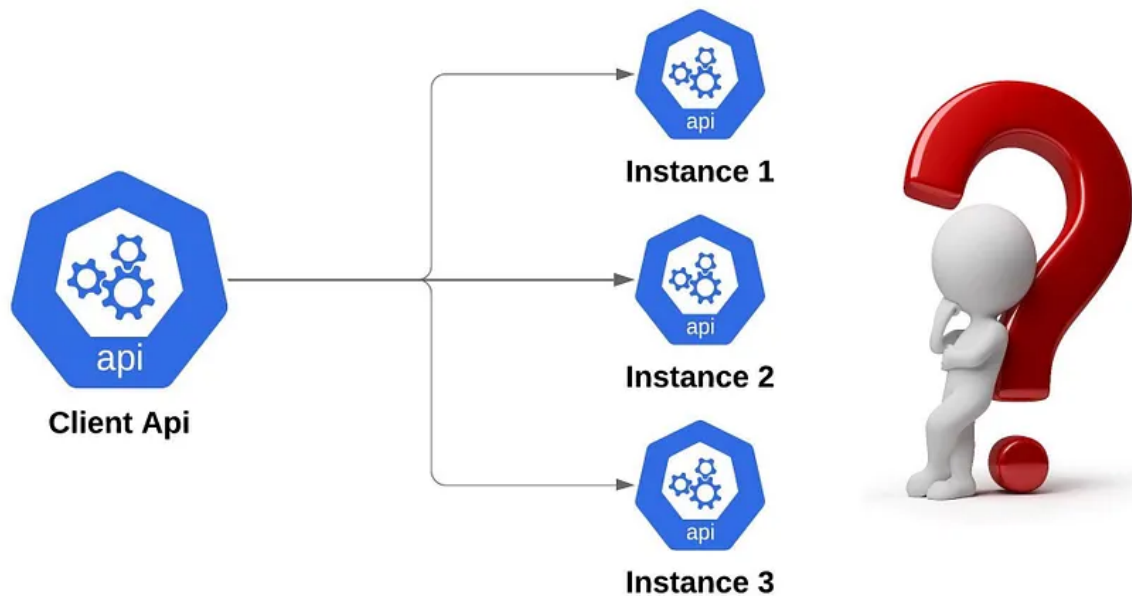
3 min read · May 11, 2020



Listen



Share



**H**i, In this post I will try to explain load balancing in microservices architecture. In which cases load balancing should be use and what are the load balancing types.

You may have chosen microservices architecture for many reasons, especially to take advantage of cloud technologies' features. Microservices architecture provides many advantages like distributed loosely coupled services, business or domain oriented development and agility for continuous delivery etc.

Everything is looking good at this point, *if all services working on one instance in your microservices environment.*

What about when you need to scale couple of services for handle more requests. Which Instance receives requests ? or How do clients know which instance they will send to

requests ?

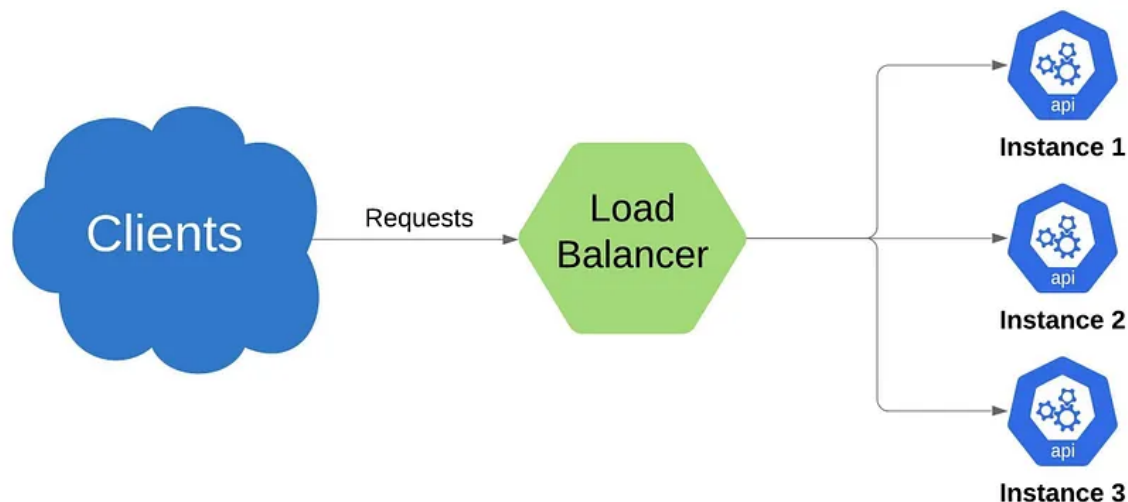
The answer to all these and similar questions: Load balancing

### Load Balancing:

Load balancing is the process of sharing, incoming network traffic in concurrent or discrete time between servers called a server farm or server pool. This sharing process can be evenly scale or can be performed according to certain rules. Rules like Round Robin, Least Connections etc.

In microservices architecture there are two type of load balancing; they are server side load balancing and client side load balancing. Let's take a closer look at them.

### Server Side Load Balancing



Server side load balancing is a classical load balancing. The traffic is distributed by a load distributor placed in front of the servers and distributed to the servers that will perform the main work equally or according to certain rules. As examples most common used server side load balancers nginx, netcaler etc.

Let's do an example of server side load balancing, in this example I will use nginx,

Firstly download nginx docker images and configure load balancing settings

```
docker run -p 80:80 -v ~/nginx/conf.d:/etc/nginx/conf.d -d nginx
```

next step is create nginx.conf in ~/nginx/conf.d directory and settings should be like below

```
1 upstream servers {  
2     server 192.168.1.54:8080;  
3     server 192.168.1.54:8081;  
4 }  
5 server {  
6     listen 80;  
7     location / {  
8         proxy_pass http://servers;  
9     }  
10 }
```

Nginx load balancing hosted with ❤️ by GitHub

[view raw](#)

and then restart nginx docker container

next step is our spring boot application. It will be a simple application which has just one endpoint

pom.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.7.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>

```

```
Spring boot demo pom.xml hosted with ❤ by GitHub view raw

1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.core.env.Environment;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RestController;
9
10 @SpringBootApplication
11 public class DemoApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(DemoApplication.class, args);
15     }
16
17 }
18
19 @RestController
20 @RequestMapping("/")
21 class DemoController {
22
23     private final Environment env;
24
25     DemoController(Environment env) {
26         this.env = env;
27     }
28
29     @GetMapping
30     public String sayHi() {
31         return "Hi from Demo application with port: " + env.getProperty("local.server.port");
32     }
33 }
```

Spring boot demo application class hosted with ❤ by GitHub [view raw](#)

application yaml:

```
1 server:
2   port: ${PORT:8080}
```

Spring boot demo application yaml hosted with ❤ by GitHub

[view raw](#)

let's start first instance of application

```
mvn spring-boot:run
```

for second instance we should change port;

```
export PORT=8081  
mvn spring-boot:run
```

Now we have two instances which are working on 8080 and 8081 ports

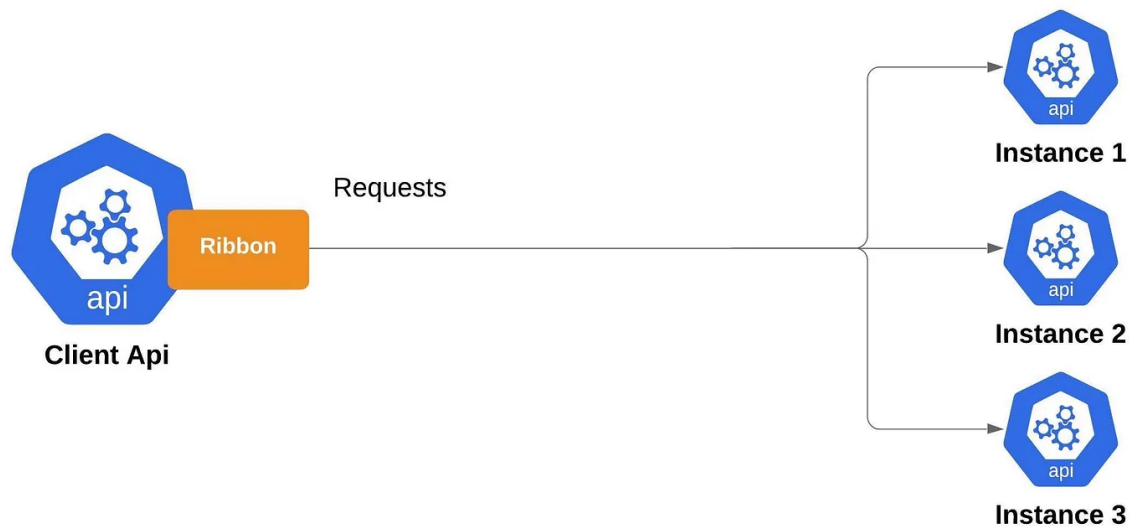
when you request to <http://localhost:80> it will return like this;

First call: Hi from Demo application with port: 8080

Secod call: Hi from Demo application with port: 8081

Now let's look at the second type load balancing, client side load balancing

### Client Side Load Balancing



In client side load balancing, client handles the load balancing. In this case client api should know all instance of server api addresses via hard coded or with a service registry. With this method you can escape bottlenecks and single point of failures. If you use service discovery you don't have to know any information about server api except api registered name, server registry mechanism will give all information about server api.


Let's do an example client side load balancing without registry.

*Firstly, I will use demo api like above and run two instances on 8080 and 8081.*

Next step will be create a client api which is using ribbon.

Our ribbon client api like below and simple;

pom.xml:

Open in app 

[Sign up](#)

[Sign In](#)



```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>2.2.7.RELEASE</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.example</groupId>
12     <artifactId>ribbon-client</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>ribbon-client</name>
15     <description>Demo project for Spring Boot</description>
16
17     <properties>
18         <java.version>1.8</java.version>
19         <spring-cloud.version>Hoxton.SR4</spring-cloud.version>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter-web</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.cloud</groupId>
29             <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
30         </dependency>
31
32         <dependency>
33             <groupId>org.springframework.boot</groupId>
34             <artifactId>spring-boot-starter-test</artifactId>
35             <scope>test</scope>
36             <exclusions>
37                 <exclusion>
38                     <groupId>org.junit.vintage</groupId>
39                     <artifactId>junit-vintage-engine</artifactId>
40                 </exclusion>
41             </exclusions>
42         </dependency>
43     </dependencies>
44
45     <dependencyManagement>
46         <dependencies>
47             <dependency>
48                 <groupId>org.springframework.cloud</groupId>
49                 <artifactId>spring-cloud-dependencies</artifactId>

```



```
49         <artifactId>spring-cloud-dependencies</artifactId>
50         <version>${spring-cloud.version}</version>
51         <type>pom</type>

```

```

1  package com.example.ribbon.client;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.client.loadbalancer.LoadBalanced;
6  import org.springframework.context.annotation.Bean;
7  import org.springframework.web.bind.annotation.GetMapping;
8  import org.springframework.web.bind.annotation.RestController;
9  import org.springframework.web.client.RestTemplate;
10
11  @SpringBootApplication
12  public class RibbonClientApplication {
13
14      public static void main(String[] args) {
15          SpringApplication.run(RibbonClientApplication.class, args);
16      }
17
18      @LoadBalanced
19      @Bean
20      RestTemplate restTemplate() {
21          return new RestTemplate();
22      }
23
24  }
25
26  @RestController
27  class RibbonController {
28
29      private final RestTemplate restTemplate;
30
31      RibbonController(RestTemplate restTemplate) {
32          this.restTemplate = restTemplate;
33      }
34
35      @GetMapping
36      public String sayHi() {
37          return "Answer is: " + restTemplate.getForObject("http://say-hi/", String.class);
38      }
39  }
```

Spring boot simple ribbon application class hosted with ❤️ by GitHub

[view raw](#)

application yml:

```
1  server:
2    port: 9090
3
4  say-hi:
5    ribbon:
6      eureka:
7        enabled: false
8      listOfServers: localhost:8080,localhost:8081
9      ServerListRefreshInterval: 15000
```

Spring boot simple ribbon application yml hosted by GitHub

[view raw](#)

After running ribbon client api, go to <http://localhost:9090> to ribbon client endpoint. You will get these answers;

First call: Answer is: Hi from Demo application with port: 8080

Second call: Answer is: Hi from Demo application with port: 8081

As a conclusion, we have learnt what is load balancing and which load balancing types use in microservices architecture.

Have fun.

Load Balancing

Microservices

Spring Load Balancing

Ribbon

Nginx



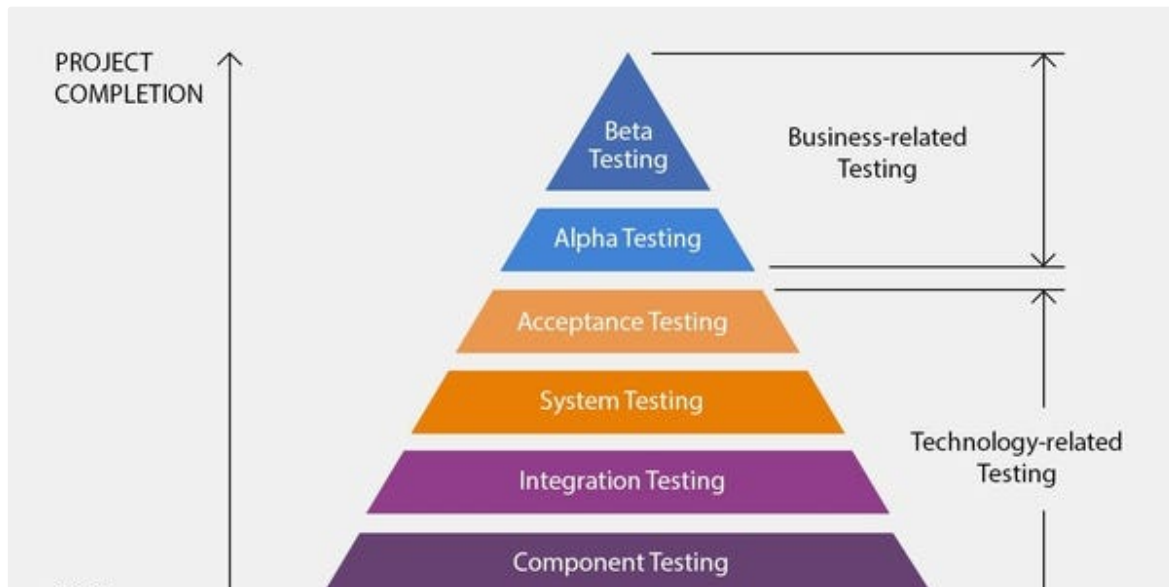
Follow

**Written by Mesut Yakut**

52 Followers

Cloud Native Technologist

### More from Mesut Yakut




 Mesut Yakut

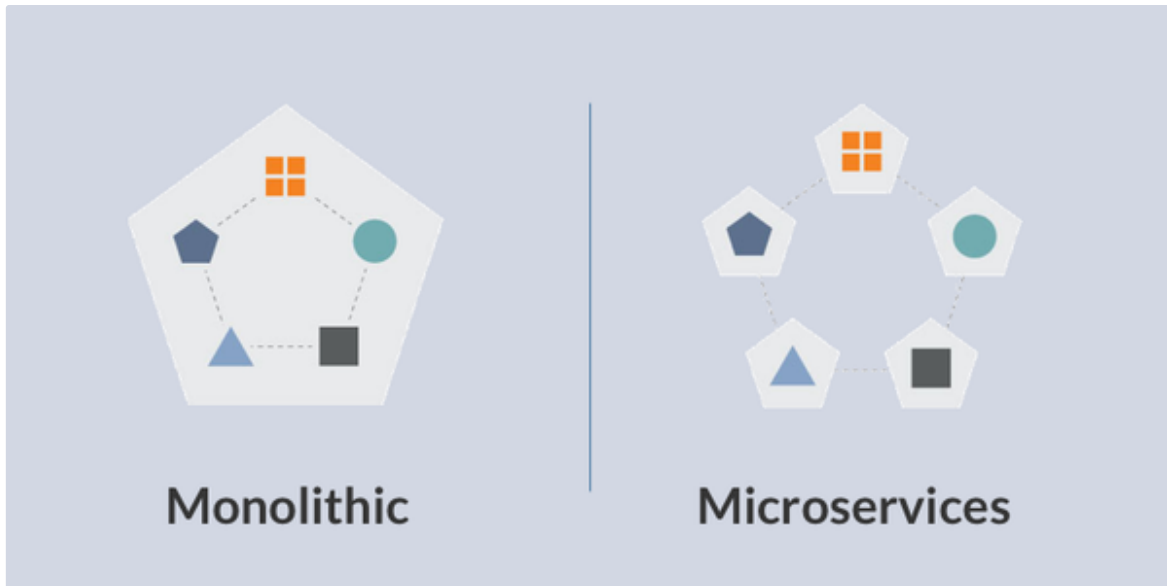
## Acceptance test with gauge and spring boot

Hi, in this article I will try to explain how to build an acceptance test with gauge and spring boot application. Acceptance test is last...

6 min read · May 5, 2020

 69  1





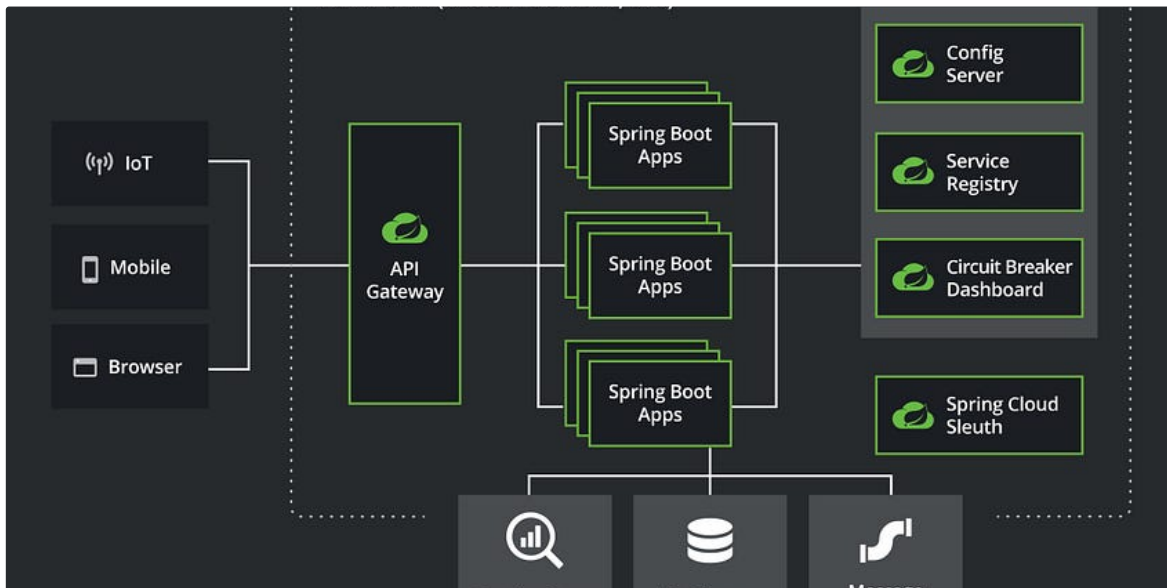
 Mesut Yakut

## Monolith to Microservices: Background Transformation

Recently, most of the big companies are moving from monolith architecture to microservices architecture. Some of them take this decision as...

4 min read · Feb 20, 2022

 27 



 Mesut Yakut

## Communication in Microservices Architecture

Anyone interested in cloud native, software architecture or who hurted from monolithic applications knows that the best solution of...

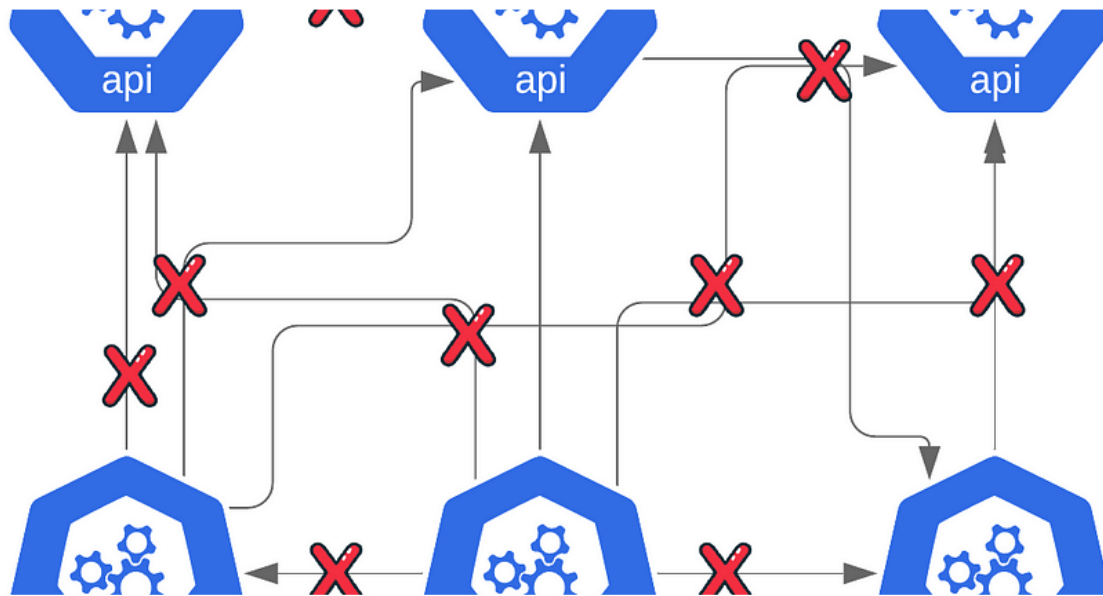
6 min read · Jun 11, 2020



62



1



Mesut Yakut

## Loosely Coupling in Microservices Architecture

The concept of dependency, gained new meanings over time with the development and change of the software industry. In this article, we...

3 min read · Dec 22, 2020

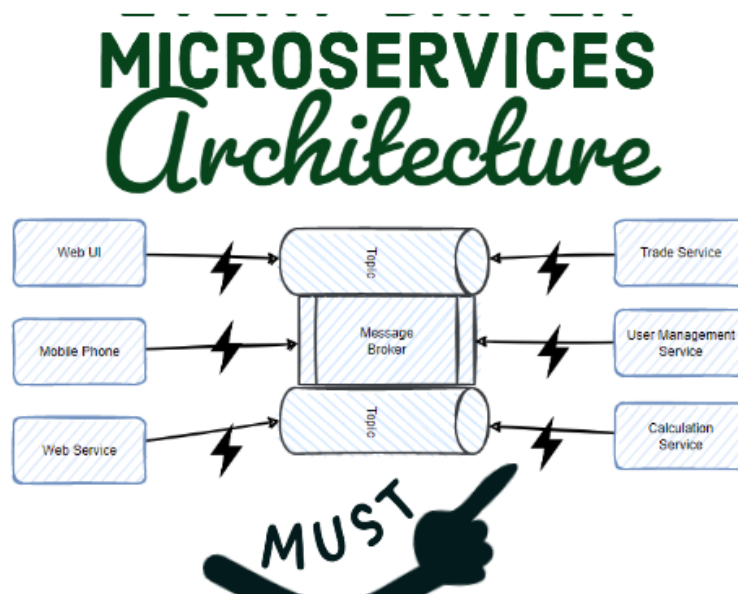


113



See all from Mesut Yakut

Recommended from Medium



Farhad Malik in FinTechExplained

## What Is Event-Driven Microservices Architecture?

Designing Modern Event-Driven Microservices Applications With Kafka And Docker Containers Suitable For All Levels

★ · 13 min read · Dec 6, 2022

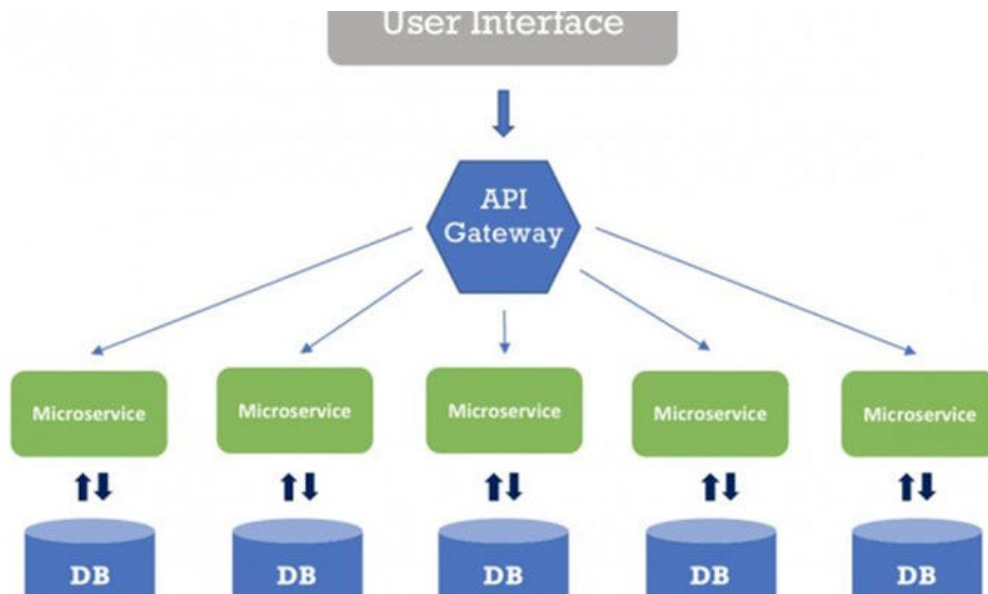


123



2





Soma in Java revisited

## 50 Microservices Design and Architecture Interview Questions for Experienced Java Programmers

Preparing for Senior Java developer role where Microservices skill is required? Here are 50 questions which you should know before going...

🌟 · 11 min read · Jan 14

344 5

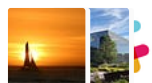


### Lists



#### Staff Picks

300 stories · 62 saves



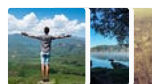
#### Stories to Help You Level-Up at Work

19 stories · 17 saves



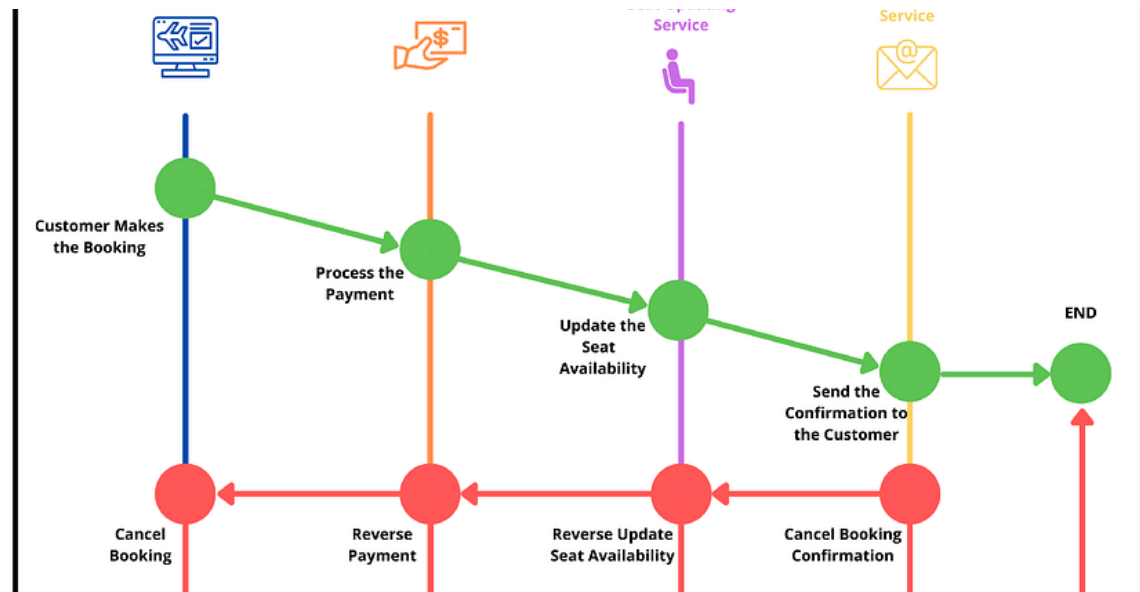
#### Self-Improvement 101

20 stories · 39 saves



#### Productivity 101

20 stories · 39 saves



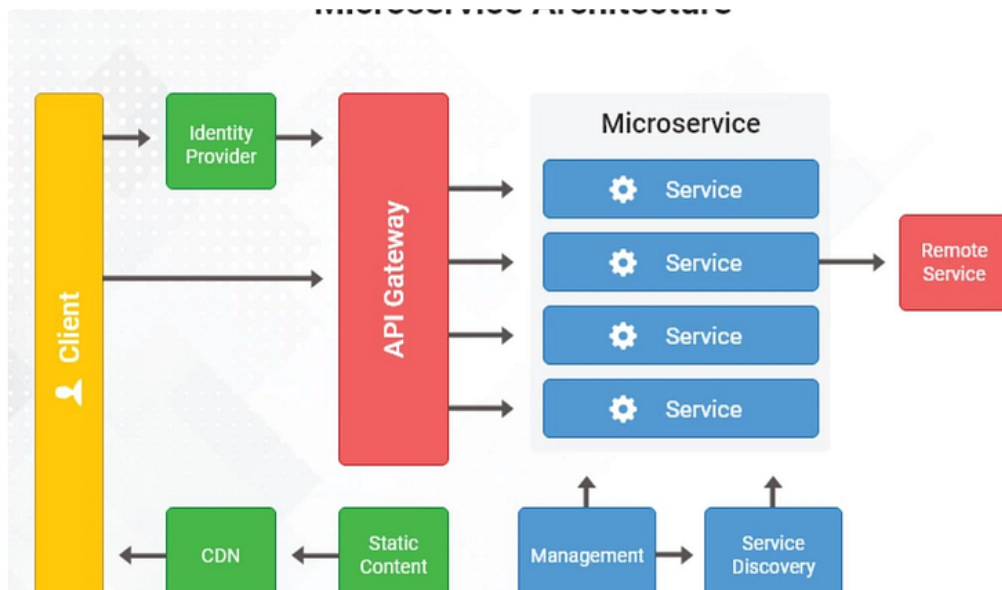
Soma in Java revisited

## What is SAGA Pattern in Microservice Architecture? Which Problem does it solve?

SAGA is an essential Micro service Pattern which solves the problem of maintaining data consistency in distributed system

★ · 6 min read · Jan 31

315 1







Dineshchandr - A Top writer in Technology in Javarevisited

## Do you know about Microservices and their Design Patterns?

Hello everyone. In this article, we are going to see about Microservice Architecture and how it is different from a Monolithic application...

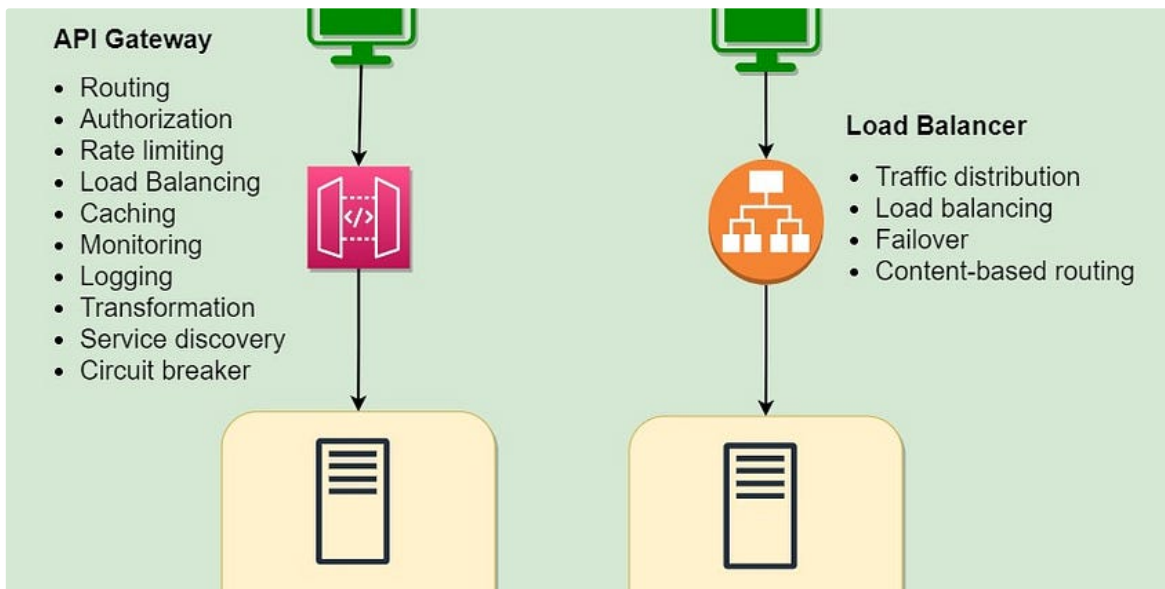
★ · 8 min read · Feb 3



420



2



Arslan Ahmad in Level Up Coding

## System Design Interview Basics: Difference Between API Gateway and Load Balancer

Often, we come across software architectural components that are part of every system design and feel as though we don't have much...

★ · 7 min read · Dec 9, 2022

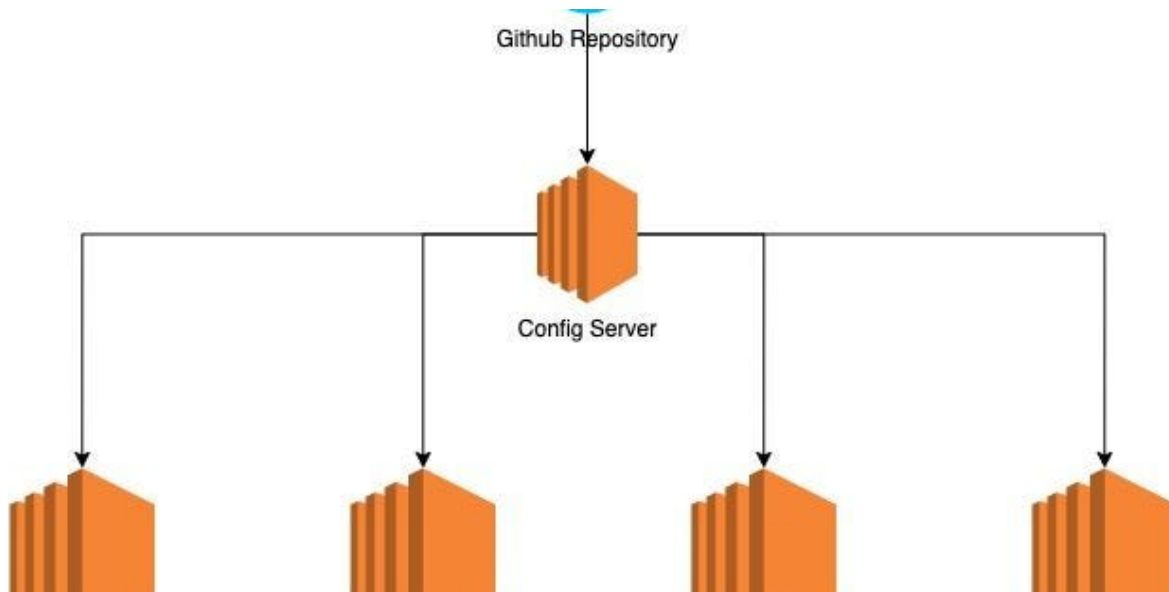


787



6





 Dineshchandgr - A Top writer in Technology in Javarevisited

## Why do we need to externalize the configuration in Microservices Architecture?

Hello everyone. In this article, we are going to see what is a configuration and look at the problems in maintaining configuration for a...

★ · 7 min read · Dec 28, 2022

 154  4



See more recommendations