Member-only story

How Pattern Matching of instanceof can help in Type Checking in Java? Example

Java 16 introduces pattern matching for the instanceof operator, making it possible to use instanceof in a more concise and readable way.

Soma · Follow
Published in Javarevisited
5 min read · Feb 19

Listen Share More

```
public static String describeShape(Object shape) {

 return shape switch {
 Circle c -> "Circle with radius " + c.getRadius();
 Rectangle r -> "Rectangle with width " + r.getWidth() + "
and height " + r.getHeight();
 Triangle t -> "Triangle with sides " + t.getSideA() + ", "
+ t.getSideB() + ", and " + t.getSideC();
 TriangleWithHeight tH -> "Triangle with base " +
tH.getBase() + " and height " + tH.getHeight();
 Square s -> "Square with side length " + s.getSideLength();
 };

}
```

Hello Java developers, if you follow new Java releases closely then you may know about new and exciting features they keep introduce every six month like Text

Blocks, Modules, Sealed Class, Records, Switch expression and now pattern matching for the `instanceof` operator.

The `instanceof` operator is not new in Java and its one of the lessor known and under rated tool to check the type of object in runtime but with new enhancement I believe it will get the popularity and love of Java developer which it should always get.

In past few articles I have been sharing my experience on Microservices like 50 Microservices Interview questions which I shared earlier as well my article about SAGA Design Pattern and Monolitihic vs Microservices architecture and in this article I will talk about instanceOf operator of Java programming language.

For those who don't know, Java 16 introduces **pattern matching for the instanceof operator,** making it possible to use instanceof in a more concise and readable way as shown below:

```
public static String powerOfInstanceOf(Object object) {

 if (object instanceof String s) {
 return "String of length " + s.length();
 }
 if (object instanceof Integer i) {
 return "Integer of value " + i;
 }
 return "Unknown object";
}
```

In this example, we have define a method `powerOfInstanceOfthat` takes an `Object` and returns a `String` describing the object. The method uses pattern matching with the `instanceof` operator to determine the type of the object.

If the object is a `String`, the pattern matching with `instanceof` binds the `String` variable `s` to the value of the object. If the object is an `Integer`, the pattern matching with `instanceof` binds the `Integer` variable `i` to the value of the object. If the object is neither a `String` nor an `Integer`, the method returns a default message.

Btw, if you are not a Medium member then I highly recommend you to join Medium and read great stories from great authors from real field. You can **join Medium here**

instanceOf Pattern Matching example with switch in Java

Note that the above example demonstrates the basic syntax of pattern matching with `instanceof` in Java.

Let's see one more example of pattern matching with the `instanceof` operator in Java 16, but before that let's define the class hierarchy which we will need for instanceof operator testing

This is an Abstract class Shape which represent an abstract shape, it could be triangle, rectangle, square or polygon or anything else.

```
public abstract class Shape {

}
```

Now let's define a couple of concrete classes to represent different types of Shape

Circle.java

This class represent a circle which extends Shape

```
public class Circle extends Shape {
 private double radius;

 public Circle(double radius) {
 this.radius = radius;
 }

 public double getRadius() {
 return radius;
```

```
  }
 }
```

Rectangle.java

This class represent a rectangle which extends Shape

```
public class Rectangle extends Shape {
 private double width;
 private double height;

 public Rectangle(double width, double height) {
 this.width = width;
 this.height = height;
 }

 public double getWidth() {
 return width;
 }

 public double getHeight() {
 return height;
 }
}
```

Triangle.java

This class represent a triangle which extends Shape

```
public class Triangle extends Shape {
 private double sideA;
 private double sideB;
 private double sideC;

 public Triangle(double sideA, double sideB, double sideC) {
 this.sideA = sideA;
 this.sideB = sideB;
 this.sideC = sideC;
 }

 public double getSideA() {
 return sideA;
 }
```

```
 public double getSideB() {
 return sideB;
 }

 public double getSideC() {
 return sideC;
 }
}
```

TriangleWithHeight.java

This class represent a triangle with height which extends Shape

```
public class TriangleWithHeight extends Triangle {
 private double height;

 public TriangleWithHeight(double base, double height) {
 super(base, base, base);
 this.height = height;
 }

 public double getHeight() {
 return height;
 }
}
```

Square.java

This class represent a square which extends Shape

```
public class Square extends Shape {
 private double sideLength;
 public Square(double sideLength) {
 this.sideLength = sideLength;
 }
 public double getSideLength() {
 return sideLength;
 }
}
```

The above code is an example of creating a class hierarchy in Java.

And here is a code which uses `instaceof` pattern matching with switch expression to provide dynamic functionality based upon object.

```
public static String describeShape(Object shape) {

 return shape switch {
 Circle c -> "Circle with radius " + c.getRadius();
 Rectangle r -> "Rectangle with width " + r.getWidth() + " and height " + r.get
 Triangle t -> "Triangle with sides " + t.getSideA() + ", " + t.getSideB() + ",
 TriangleWithHeight tH -> "Triangle with base " + tH.getBase() + " and height "
 Square s -> "Square with side length " + s.getSideLength();
 };

}
```

can you use regular expression on instanceof operator

No, the `instanceof` operator in Java cannot be combined with regular expressions. The `instanceof` operator is used to check if an object is an instance of a specific class or one of its subclasses.

It does not have the ability to perform string matching or pattern matching, which are commonly performed using regular expressions.

Pros and cons of instanceof operator in Java:

Now, let' see pros and cons of using instanceof oprator in Java:

Pros of using `instanceof` operator:

1. Readability
   The `instanceof` operator is simple to understand and helps make the code more readable.

2. Type Checking
   It provides a way to check the type of an object at runtime, which can be useful in situations where the type of an object is not known at compile time.

3. Dynamic Dispatch
   The `instanceof` operator can be used in combination with polymorphism to

perform dynamic dispatch. This means that the correct method can be called at runtime, based on the actual type of the object.

Cons of using `instanceof` operator:

1. Coupling
   The `instanceof` operator can lead to tight coupling between different parts of the code, as it requires knowledge of the specific types that are being checked for.

2. Code Maintenance
   If new types are added to the code, the `instanceof` checks may need to be updated to account for the new types. This can lead to maintenance overhead and make the code more difficult to modify.

3. Performance
   The `instanceof` operator can be slower than alternatives, such as using the `getClass()` method, as it needs to traverse the class hierarchy to determine the type of the object.

That's all about **how to use instanceof operator in Java to check the type of object at runtime.** With new enhancement and pattern matching `instanceof` operator has now become a better and more powerful tool in Java and you should look at it now with a fresh perspective.

And, if you are not a Medium member then I highly recommend you to join Medium and read great stories from great authors from real field. You can **join Medium here**

Java Programming Coding Development Software Development

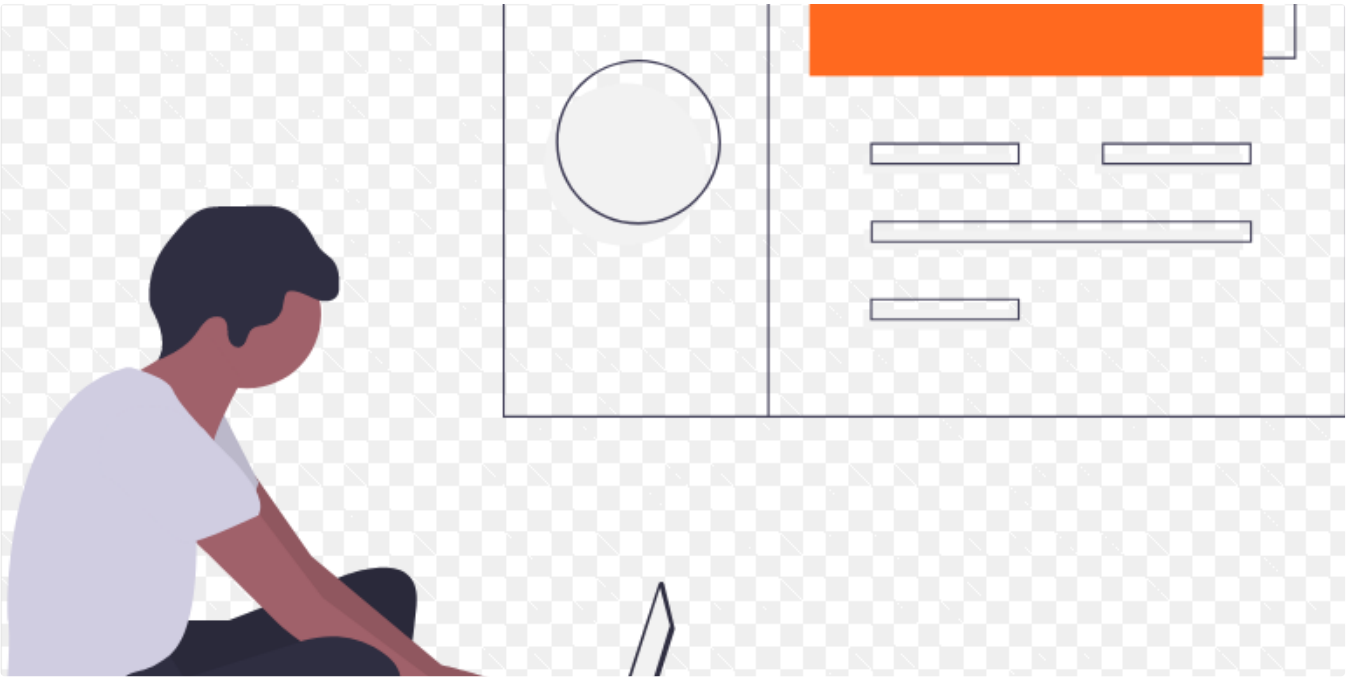
Follow

Written by Soma

4.2K Followers · Editor for Javarevisited

Java and React developer, Join Medium (my favorite Java subscription) using my link 👉 https://medium.com/@somasharma_81597/membership

More from Soma and Javarevisited



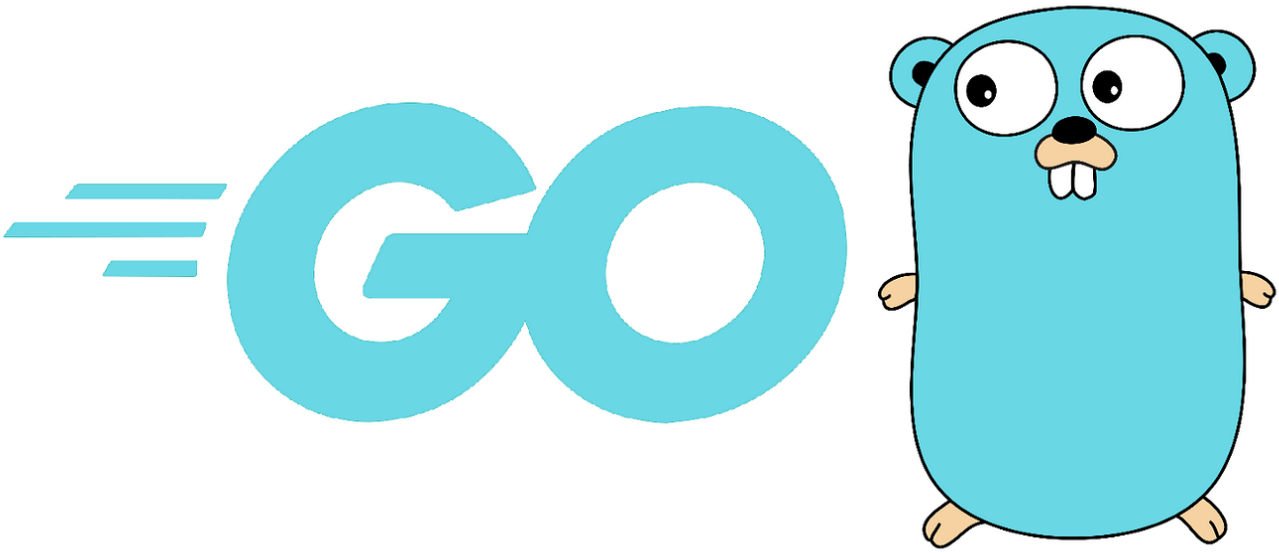
Soma in Javarevisited

Difference between @RequestMapping and @GetMapping Annotations in Spring MVC Web Framework

Hello friends , if you are preparing for Java Developer and Spring Developer interviews then you must prepare for questions like...

✨ · 6 min read · Aug 19

49



javinpaul in Javarevisited

10 Projects You Can Build to Learn Golang in 2023

My favorite Golang Projects with courses for beginners and experienced developers

11 min read · Aug 7

98 1

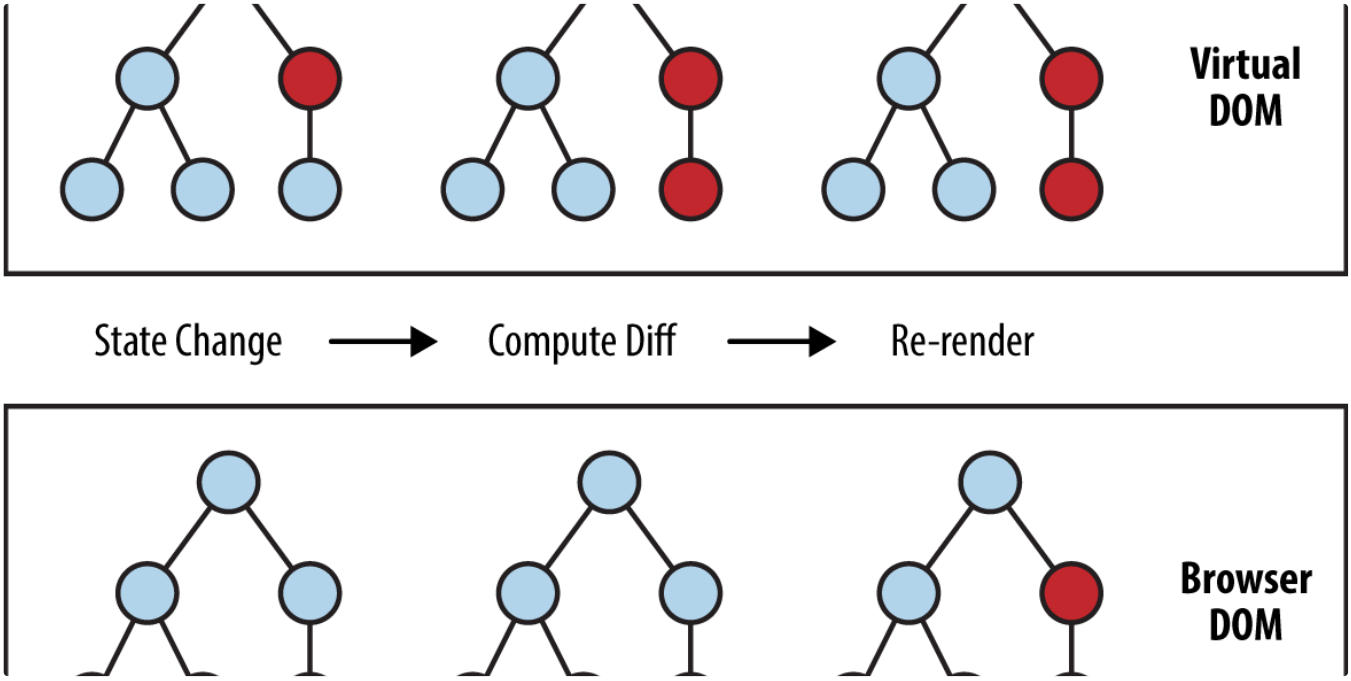
Sivaram Rasathurai in Javarevisited

10 Java Stream Tips—Must Read

Java Stream API is like a Swiss Army knife for Java developers—it's versatile, compact, and can handle a wide variety of tasks with ease.

4 min read · Mar 1

294



Soma in JavaScript in Plain English

10 React Features That Make Frontend Development Easier

Few reasons why React.js is considered one of the best frontend library of our time

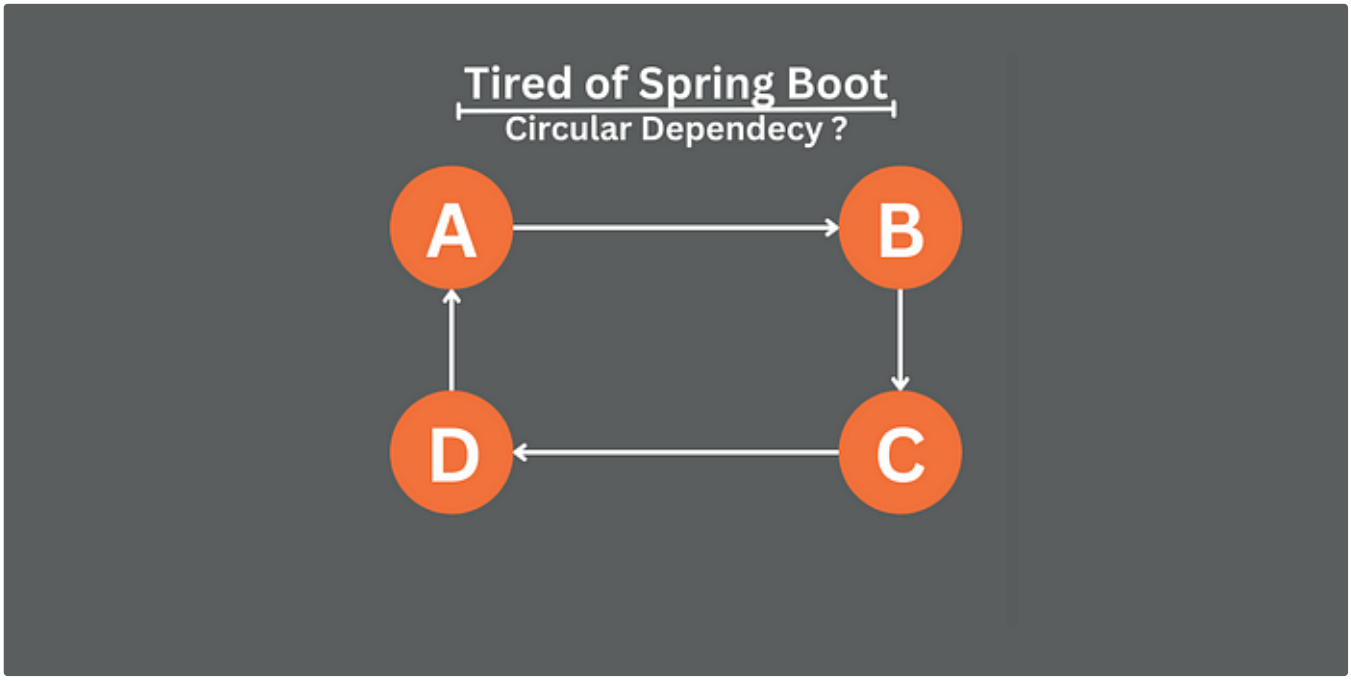
· 7 min read · Jul 27

155

See all from Soma

See all from Javarevisited

Recommended from Medium



karthik jeyapal

Circular Dependency in Spring Boot: How to Detect and Fix It

In Spring Boot, circular dependency occurs when two or more beans depend on each other. This can happen when a bean requires another bean...

7 min read · Apr 16

4

Eray Araz

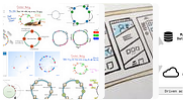
Spring Webhook

Introduction

3 min read · Apr 2

109 1

Lists

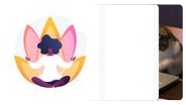


General Coding Knowledge

20 stories · 257 saves

It's never too late or early to start something

15 stories · 93 saves

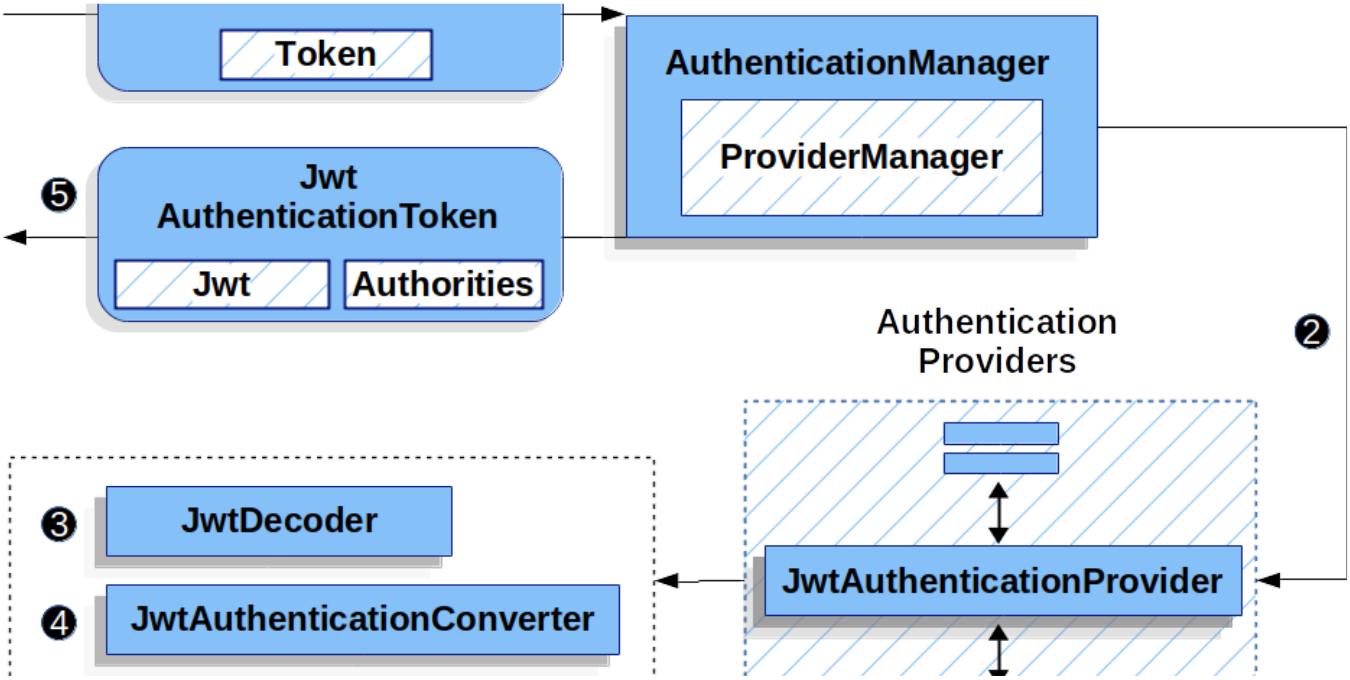


Stories to Help You Grow as a Software Developer

19 stories · 309 saves

Coding & Development

11 stories · 129 saves



Alper Kurtoglu

Spring OAuth2 with OIDC: Moving from ‘Scope’ to ‘Roles’

In this tutorial, I will discuss how to use authorities instead of scopes as roles in Spring OAuth2.

3 min read · Aug 17

83

```
lass java.lang.Number
assed: 2

h exit code 0
```

Chunks

Deep Dive into Java Generics

As a Java engineer, we may have worked with Generics at some point without understanding fully how they work. Generics was introduced in...

5 min read · Aug 20

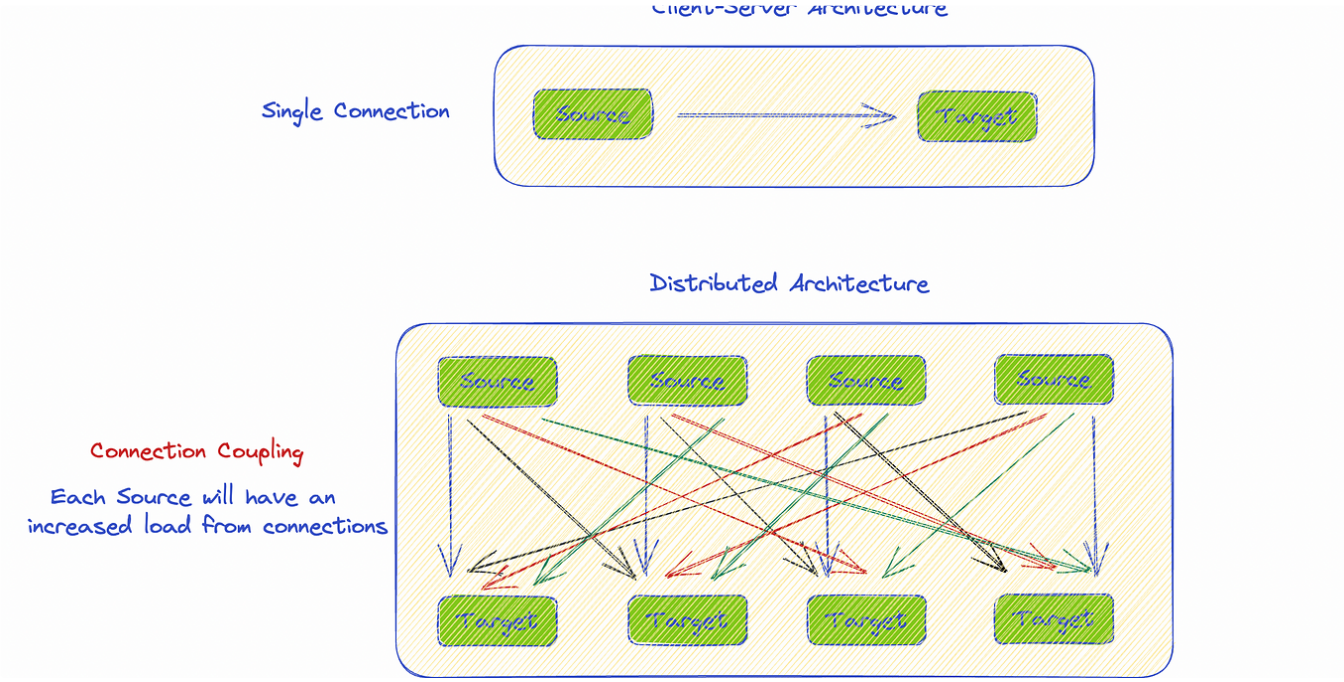
3

Ionut Anghel

REST Endpoint Best Practices Every Developer Should Know

5 min read · Mar 18

79 1



Akshat Vashishtha

Kafka with Spring Boot using docker-compose

Prior understanding to Kafka, we should understand the problem Kafka try to solve. In simple Client-Server architecture source machine...

4 min read · May 9

39

See more recommendations