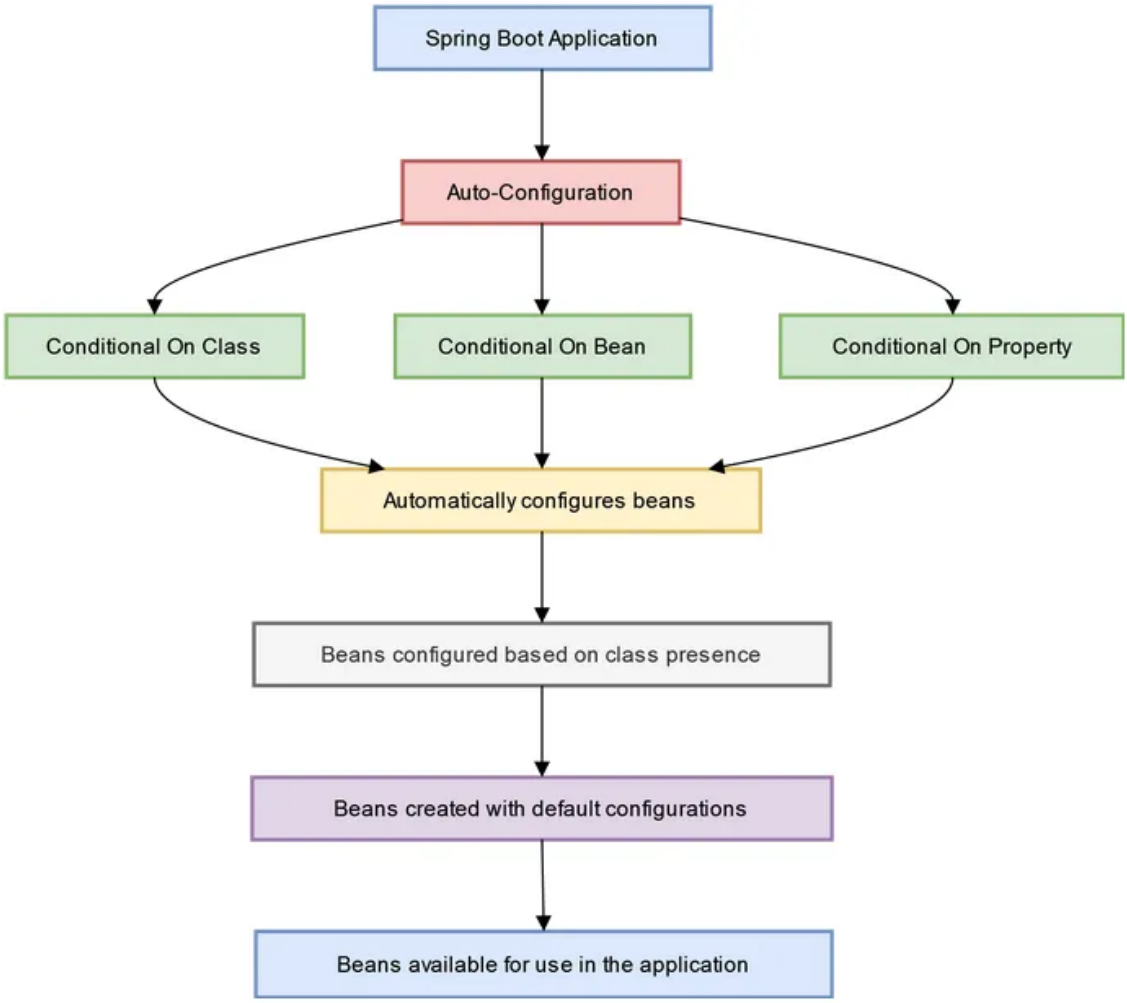
Member-only story

Difference between @SpringBootApplication and @EnableAutoConfiguration in Spring Boot

Understanding the key differences and Usage of `@SpringBootApplication` and `@EnableAutoConfiguration` in Spring Boot Applications

Soma · Follow
Published in Javarevisited
9 min read · Apr 13

Listen Share ••• More



Hello folks, If you are doing Java development then you may know that Spring Boot is one of the popular framework for building modern and scalable Java applications. It provides a wide range of features, including **auto-configuration**, that make Java development faster and more efficient

However, when working with Spring Boot, you may come across different annotations that seem similar in purpose, such as `@SpringBootApplication` and `@EnableAutoConfiguration`, while both are **used to enable auto-configuration feature of Spring Boot** there are some subtle difference between them, knowing which you can make best use of them.

In this article, we will dive into the differences between these two annotations and their usage in Spring Boot applications. Understanding these differences will help you make informed decisions when configuring and bootstrapping their Spring Boot applications for optimal performance.

If you are preparing for Java and Spring Boot Interview then you have come to the right place. In the past, I have shared a lot of popular questions like difference between **RequestParam** and **PathVariable** annotations, difference between @`Contorller` and @`RestController` annotation, @Bean vs @Component annotation, and @Controller vs @Service @Repository and in this article, I am going to share another popular Spring Boot question, difference between @EnableAutoConfiguration and @SpringBootApplication.

`@EnableAutoConfiguration` annotation is used to enable Spring Boot's auto-configuration feature, which automatically configures the application based on the classpath dependencies and other properties. It scans the classpath for various configurations and automatically configures the beans and other components accordingly. For example, if you have Hibernate JAR file then it can configure Hibernate for you.

On the other hand, `@SpringBootApplication` is a combination of several other annotations, including `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`.

It is typically used to mark the main class of a Spring Boot application and is responsible for enabling the auto-configuration features provided by Spring Boot, setting up the application context, and scanning for Spring components.

By the way, if you are preparing for Java and Spring interview then In my earlier articles, I have also shared *21 Software Design Pattern questions*, 10 *Microservice Scenario based questions*, *20 SQL queries from Interviews*, ***50 Microservices questions***, *60 Tree Data Structure Questions*, 15 System Design Questions, and *35 Core Java Questions* and *21 Lambda and Stream questions* which you can use for your Java interview preparation.

In short, `@SpringBootApplication` is a higher-level annotation that includes `@EnableAutoConfiguration` along with other annotations, while `@EnableAutoConfiguration` is used specifically to enable the auto-configuration feature of Spring Boot.

Now that we know the basics, we will deep dive into each of them and understand them in bit more detail with code examples.

But, if you are not a Medium member yet then I highly recommend you to join Medium and read my other member only articles for your interview preparation. You can **join Medium here**

What is @EnableAutoConfiguration annotation? How does it work?

As I said, `@EnableAutoConfiguration` is an annotation in Spring Boot that enables automatic configuration of the application's beans and components based on classpath dependencies and other configuration settings.

When you annotate a class with `@EnableAutoConfiguration`, Spring Boot automatically performs the following tasks:

1. Scans the classpath for relevant Spring Boot starter dependencies and their configuration files.

2. Automatically configures beans and components based on the detected dependencies and their configuration.

3. Sets up sensible default configurations for various Spring Boot features, such as data sources, caching, logging, security, etc.

In other words, `@EnableAutoConfiguration` allows Spring Boot to automatically configure the application's beans and components based on the dependencies and their configuration, without having to explicitly define them in the application's code.

This **helps in reducing boilerplate configuration code** and makes it easier to quickly set up a Spring Boot application with sensible defaults.

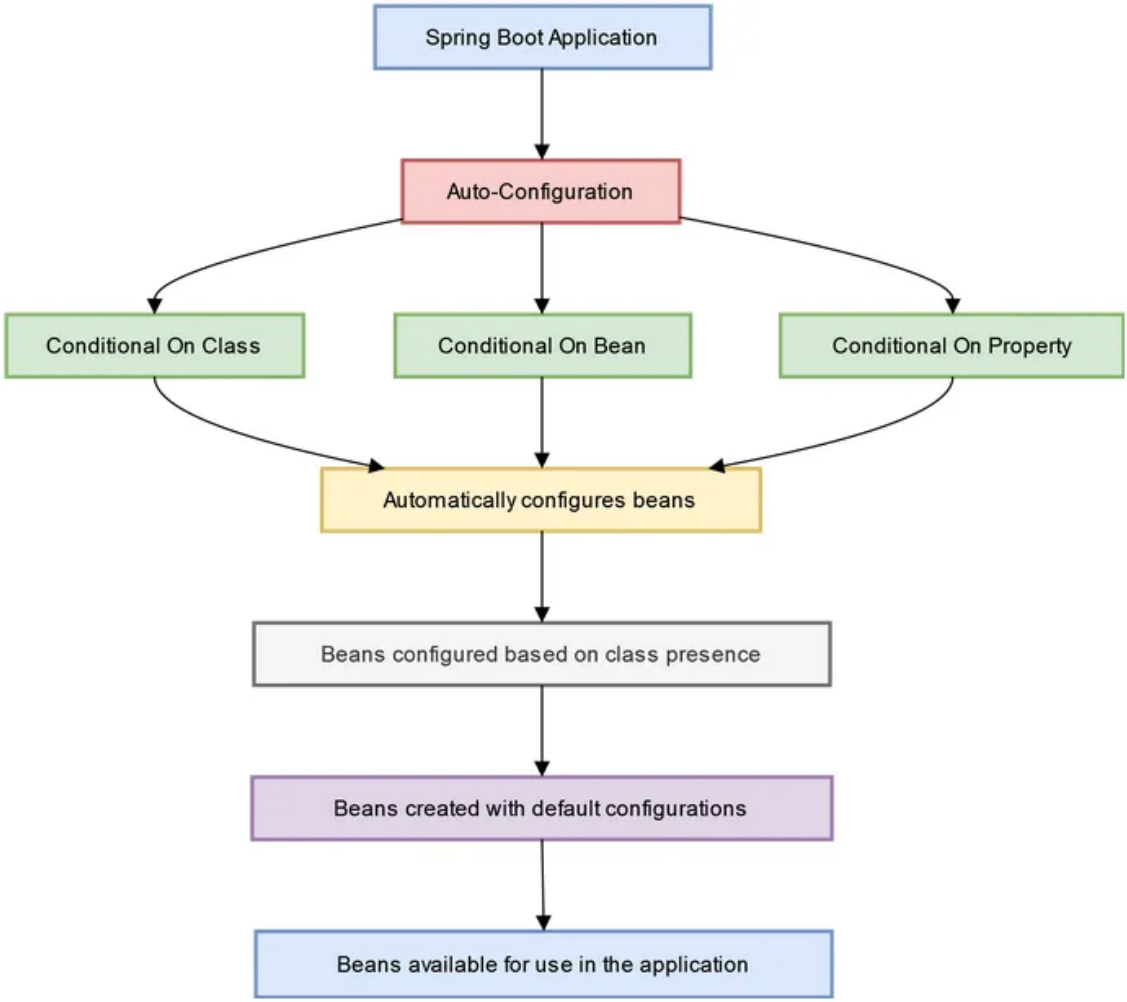
Here’s an example of using `@EnableAutoConfiguration` annotation in a Spring Boot application:

```
@EnableAutoConfiguration
public class PaymentServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(PaymentServiceApplication.class, args);
    }
}
```

In the above example, `@EnableAutoConfiguration` annotation is used to enable automatic configuration of the Spring Boot application. It scans the classpath dependencies and other settings to configure the application. The `PaymentServiceApplication` class is the main entry point of the PaymentService application, where the `main` method is defined to run the Spring Boot application.

And, if you are not familiar with how auto-configuration works in Spring Boot, here is a nice diagram to understand it, I am going to cover that topic soon, if you want me to cover it next, please leave a comment



What is @SpringBootApplication annotation? How does it work?

The `@SpringBootApplication` annotation is a meta-annotation in Spring Boot that combines three commonly used annotations: `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. It is typically **used to mark the main class of a Spring Boot application** and provides a convenient way to configure and bootstrap a Spring Boot application.

Here's a brief overview of how the `@SpringBootApplication` annotation works:

1. `@Configuration`: This annotation indicates that the class is a configuration class and can contain Spring bean definitions. It is used to define and configure beans that are managed by the Spring framework.

2. `@EnableAutoConfiguration`: This annotation enables automatic configuration of the Spring Boot application. It scans the classpath dependencies and other

settings to configure the application based on the detected technologies and available libraries.

3. `@ComponentScan` : This annotation *scans the package and its sub-packages for Spring components*, such as *beans, controllers, services, and repositories.* It allows Spring to automatically detect and register these components for use in the application.

By combining these three annotations, `@SpringBootApplication` simplifies the configuration and bootstrapping process of a Spring Boot application. It provides a convenient way to define beans, enable automatic configuration, and scan for components, making it easier to develop and deploy Spring Boot applications.

Here's an example of how you can use the `@SpringBootApplication` annotation with a Service class in a Spring Boot application:

```
@SpringBootApplication
public class MySpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(MySpringBootApplication.class, args);
    }

    // PaymentService class
    @Service
    public class PaymentService {
        // ... implementation of payment service ...
```

```
    }

    // ... other application-specific code here ...
}
```

In this example, `PaymentService` is a custom service class that can be used to implement payment-related functionality in the Spring Boot application. The `@Service` annotation is used to mark the `PaymentService` class as a Spring bean, which makes it eligible for dependency injection and other Spring-specific features.

The `MySpringBootApplication` class is the main class of the Spring Boot application, marked with the `@SpringBootApplication` annotation. The `main` method is the entry point of the application, which uses `SpringApplication.run()` to start the Spring Boot application.

The `PaymentService` class can be used as a regular Spring bean within the application, and its dependencies will be managed by the Spring framework.

What is difference between @EnableAutoConfigruation and @SpringBootApplication annotations?

As I said in above paragraph, `@EnableAutoConfiguration` and `@SpringBootApplication` are both annotations used in Spring Boot applications, but they have different purposes and usage.

`@EnableAutoConfiguration`:

This annotation is used to enable auto-configuration in a Spring Boot application. It allows Spring Boot to automatically configure the application's beans, based on the classpath and other configurations.

It scans the classpath for available configurations, and automatically configures the application with sensible defaults for various components, such as data sources, web servers, and message brokers, among others.

It is often used in combination with other annotations, such as `@Configuration` and `@ComponentScan`, to define additional configuration classes and customize the auto-configuration behavior.

`@SpringBootApplication`

This is a meta-annotation that combines multiple annotations, including `@EnableAutoConfiguration`, `@ComponentScan`, and `@Configuration`, into a single annotation.

It is typically used as the main annotation for the main class of a Spring Boot application. It enables auto-configuration, component scanning, and marks the class as a configuration class, allowing it to define beans and other application-specific configurations.

It provides a convenient way to bootstrap a Spring Boot application with sensible defaults and enables the application to start as a standalone, executable JAR file.

In summary, while `@EnableAutoConfiguration` is used to specifically enable auto-configuration in a Spring Boot application, `@SpringBootApplication` is a meta-annotation that combines multiple annotations, including `@EnableAutoConfiguration`, to provide a convenient way to bootstrap a Spring Boot application with sensible defaults and configuration capabilities.

Here is another nice diagram which shows how @SpringBootApplication annotation can enable auto-configuration in Spring boot.

Java and Spring Interview Preparation Material

Before any Java and Spring Developer interview, I always use to read the below resources

Grokking the Java Interview

Grokking the Java Interview: click here

I have personally bought these books to speed up my preparation.

You can get your sample copy here, check the content of it and go for it

Grokking the Java Interview [Free Sample Copy]: click here

> *If you want to prepare for the Spring Boot interview you follow this consolidated ebook, it also contains microservice questions from spring boot interviews.*

Grokking the Spring Boot Interview

You can get your copy here — **Grokking the Spring Boot Interview**

Conclusion

That's all about difference between EnableAutoConfiguration and Spring BootApplication annotation in Spring Boot. In conclusion, understanding the difference between `@EnableAutoConfiguration` and `@SpringBootApplication` is essential for developing Spring Boot applications effectively. While `@EnableAutoConfiguration` specifically enables auto-configuration in a Spring Boot application, on the other hand `@SpringBootApplication` is a meta-annotation that combines multiple annotations, including `@EnableAutoConfiguration`, to provide a convenient way to bootstrap a Spring Boot application with sensible defaults and configuration capabilities.

This concept is not just important for Spring Framework interview point of view but proper usage of these annotations can help streamline the configuration process and ensure optimal performance of your Spring Boot applications.

Whether you choose to use `@EnableAutoConfiguration` or `@SpringBootApplication`, understanding their purposes and usage will enable you to make informed decisions when developing Spring Boot applications.

This is one of the important questions for Spring Boot interviews and you should prepare for them. Additionally, you can also prepare Java Microservices Questions like *difference between API Gateway and Load Balancer*, *SAGA Pattern*, *how to manage*

transactions in Microservices, and *difference between SAGA and CQRS Pattern*, they are quite popular on interviews.

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**

Join Medium with my referral link - Soma

Read this and every story from thousands of great writers on Medium). Read more from great writers on Medium like...

medium.com

Other Spring Framework articles you may like

Top 20 Spring Framework and Spring Boot Scenario based Questions for Java Developers

Mastering Practical Application of Spring and Spring Boot with These Real-World Interview Questions for Java Developers

medium.com

25 Spring Framework Interview Questions for 1 to 3 Years Experienced Java Programmers

Preparing for Java Interviews? Don't forget to prepare for Spring framework questions, here are the popular Spring...

medium.com

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...

medium.com

Java | Spring Boot | Programming | Development | Software Development

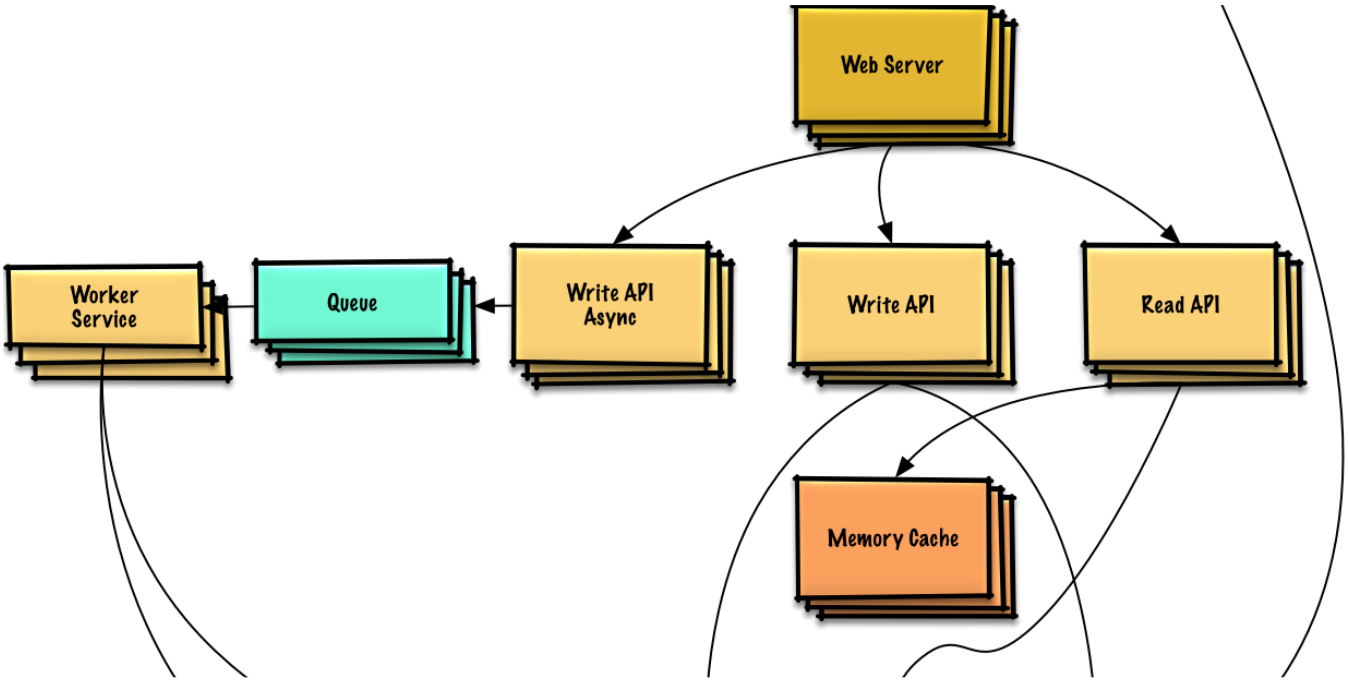
Follow

Written by Soma

4.1K 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

10 System Design Topics You Must Prepare for Tech Interviews

These are the 10 topics every developer must prepare for tech interviews to get ahead of 99% of candidates.

· 7 min read · Jul 2

149

javinpaul in Javarevisited

10 Best Backend Frameworks for Web Development in 2023

These are the best backend development frameworks for web development in Java, Ruby, Python, JavaScript, PHP, Scala, and Golang in 2023

9 min read · Mar 9, 2022

398 5

javinpaul in Javarevisited

10 Best Books for Data Structure and Algorithms for Beginners in Java, C/C++, and Python

Algorithms are language agnostic, and any programmer worth their salt should be able to convert them to code in their programming language...

12 min read · Mar 13, 2020

1.2K 1

Soma in Javarevisited

50+ Spring Boot Interview Questions and Answers for Java Programmers

These are 50+ Spring Boot Interview Questions and answers to Crack your Next Java Developer interview

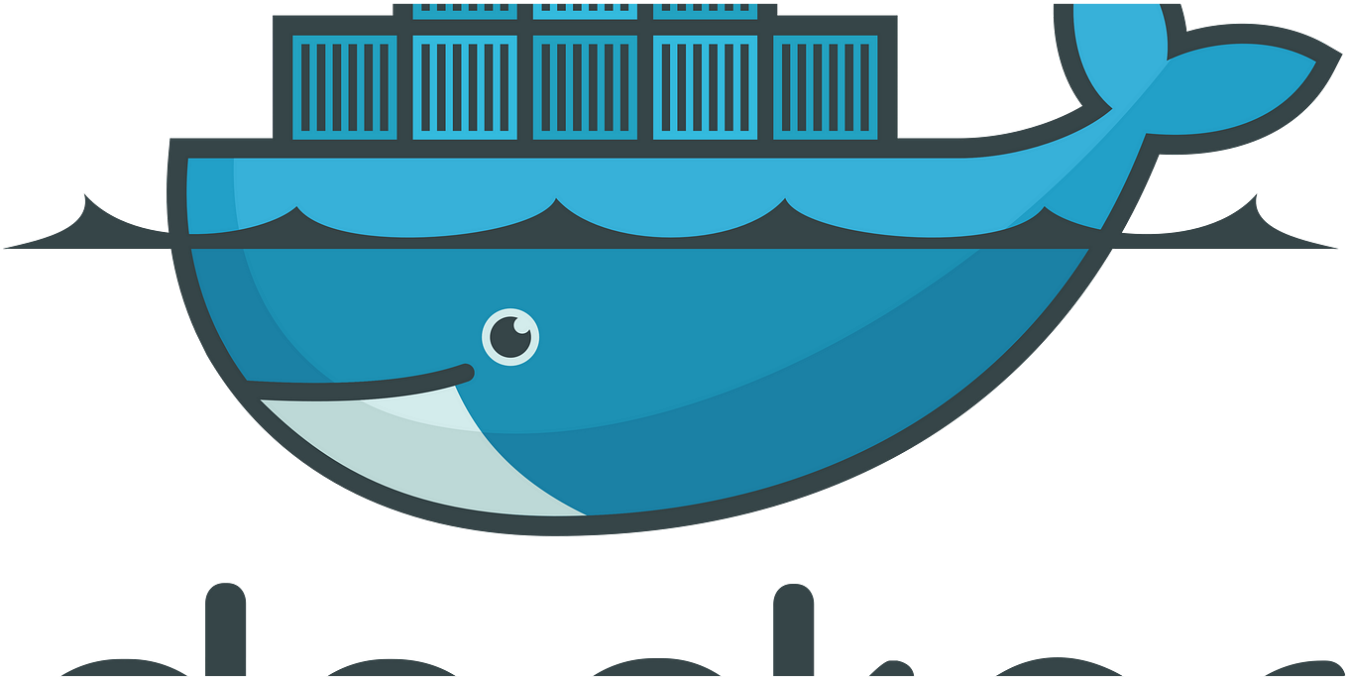
· 19 min read · Apr 24

182

See all from Soma

See all from Javarevisited

Recommended from Medium



EazyPeazyGeeky in Stackademic

How to Install and Use Docker on Ubuntu 22.04

Docker is a platform that allows you to easily create, deploy, and run applications in containers. Containers are lightweight, portable...

2 min read · Jul 30

Hamza Nassour in Javarevisited

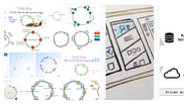
What Happens Internally When You Start A Spring Boot Application(Part1)

ApplicationContext creation/registration , AutoConfiguration .. .

4 min read · Feb 19

94

Lists



General Coding Knowledge

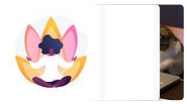
20 stories · 200 saves

It's never too late or early to start something

13 stories · 68 saves

Coding & Development

11 stories · 101 saves



Stories to Help You Grow as a Software Developer

19 stories · 268 saves

Syed Habib Ullah

Top Most Asked Java Interview Questions at Accenture, Infosys, Capgemini, Wipro, Cognizant...

As Java is one of the most popular programming languages used in software development today, it's no surprise that Java skill is highly...

9 min read · Apr 1

30 2

The Tech Insider

Liquibase: An Open-Source Solution for Database Versioning and Change Control

Simplifying Database Schema Management

3 min read · May 3

1

Daniel Zielinski

Senior Java Software Developer Interview Questions—part 1

1. Anemic Model vs Rich Model ?

· 3 min read · Feb 18

17