Connecting Multiple Databases in Spring Boot

P Spriya · Follow
2 min read · Feb 6, 2024

To connect to two different databases in a Spring Boot application, you typically need to configure multiple `DataSource` beans and manage transactions accordingly. Here's a step-by-step guide:

1. Define Configuration Properties: Start by defining the database connection properties in your `application.properties` or `application.yml` file for each database. Specify different properties for each database, including URL, username, password, and driver class.

2. Configure DataSources: Create multiple DataSource configurations, each representing a connection to a different database. You can do this by creating separate configuration classes for each database or by configuring them in a single class.

3. Configure JPA or JDBC Templates: Once you have configured the DataSources, configure JPA repositories or JDBC templates for each database. Spring Boot auto-configuration may handle this for you based on the DataSource beans you've defined.

4. Manage Transactions: If you need transactions across multiple databases, you'll need to manage transaction synchronization manually or use JTA (Java Transaction API) for distributed transactions.

5. Here's an example demonstrating how to configure two DataSource beans in a Spring Boot application:

```
@Configuration
public class DataSourceConfig {

@Primary
@Bean(name = "firstDataSource")
@ConfigurationProperties(prefix = "spring.datasource.first")
public DataSource firstDataSource() {
return DataSourceBuilder.create().build();
}

@Bean(name = "secondDataSource")
@ConfigurationProperties(prefix = "spring.datasource.second")
public DataSource secondDataSource() {
return DataSourceBuilder.create().build();
}
}
```

In the above configuration:

- `firstDataSource` and `secondDataSource` represent two different database connections.
- `@ConfigurationProperties` is used to bind the properties from `application.properties` to the DataSource objects.

After configuring DataSources, you can use them in your repositories or JDBC templates by qualifying the `@Autowired` annotation with the DataSource name or by using `@Qualifier`:

```
@Autowired
@Qualifier("firstDataSource")
private DataSource firstDataSource;

@Autowired
@Qualifier("secondDataSource")
private DataSource secondDataSource;
```

Make sure to adjust your repository or template configurations accordingly to use the correct DataSource.

Finally, ensure proper transaction management for your use case. If you need transactions across multiple databases, you may need to implement custom transaction management logic or use JTA.

Making Spring Boot applications performant and resource-efficient

Jun 24 201 3

Sharat Naik

Connect and use multiple datasources in Spring boot Java

It becomes a little challenging when you have to configure multiple datasources in your database application. Here's an easy to follow...

Feb 3 28 2

Lists

Staff Picks
684 stories · 1124 saves

Stories to Help You Level-Up at Work
19 stories · 685 saves

Self-Improvement 101
20 stories · 2273 saves

Productivity 101
20 stories · 2004 saves

Vasko Jovanoski

Say Goodbye to Repetition: Building a Common Library in Spring Boot

Learn to create a shared library in Spring Boot, enhancing code reusability, simplifying maintenance, and reducing bugs in your...

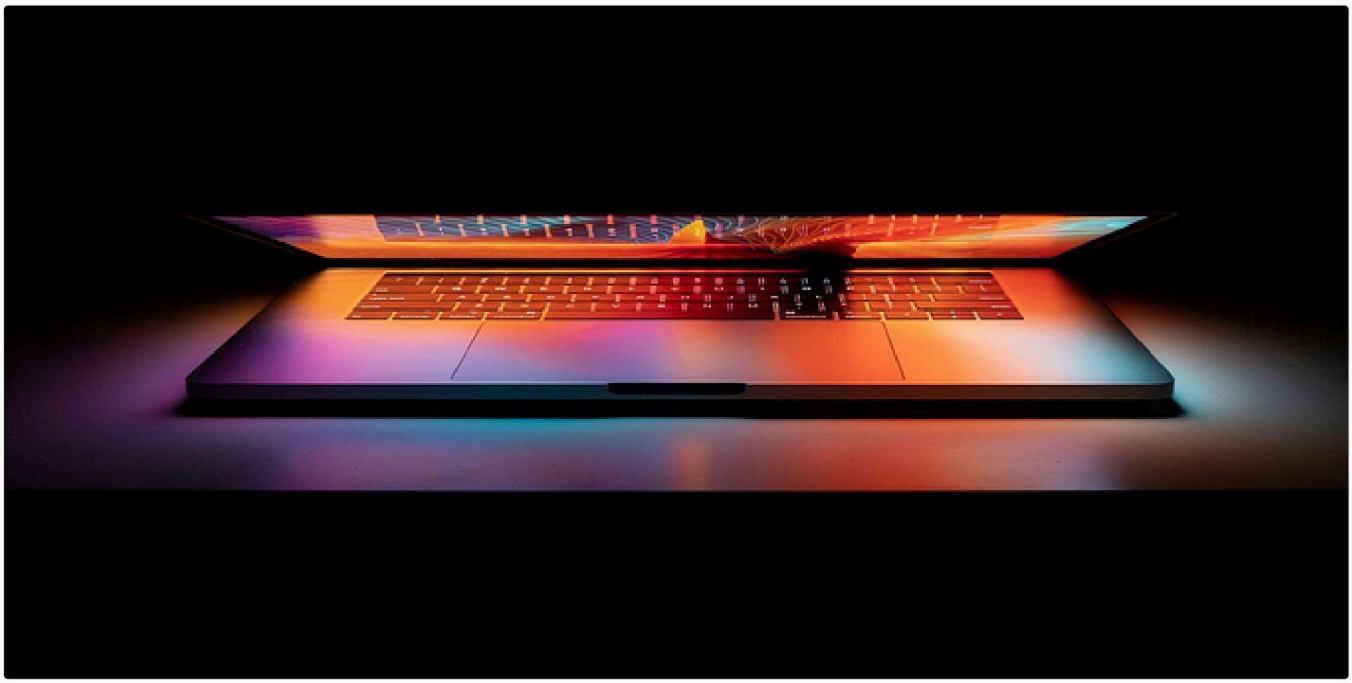
Jun 25 258 8

Oliver Foster

Spring Boot: How Many Requests Can Spring Boot Handle Simultaneously?

My article is open to everyone; non-member readers can click this link to read the full text.

Jun 17 840 12

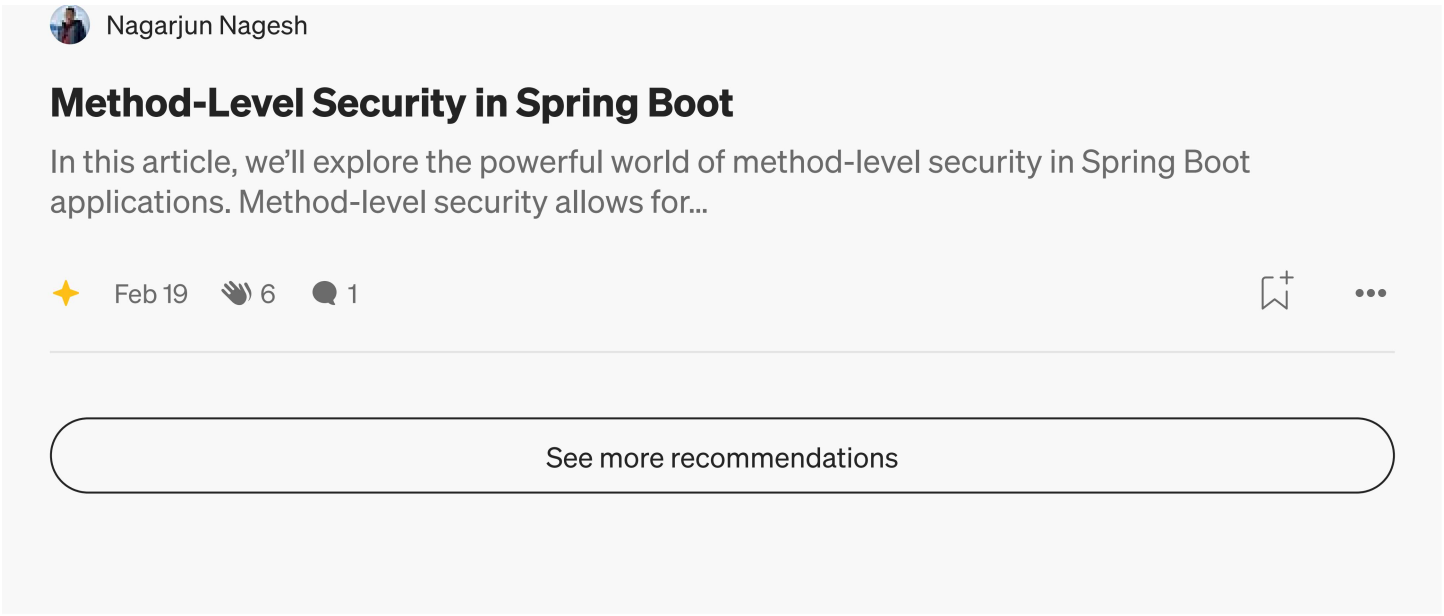


Mikael Svens

Why You Should Stop Using @Value Annotations In Spring (And Use This Instead)

If you have been working with Java and Spring Boot, I'm sure you have come across the @Value annotation. I'm here to show you an...

Mar 20 1.2K 27



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.