JaCoCo Code Coverage with Spring Boot

5 min read · May 19, 2023

Truong Bui Follow

I'm pretty sure that each one of us has and will continue to write Unit Testing in our projects. Unit Testing is playing a very important role as we employ it to test every code snippet, function, method, and more.

Upon completing the writing of Unit Testing, it is essential to see how much the tests covered the code and identify areas that require additional testing. We do those things to make sure the application code is thoroughly tested and ready for deployment. Today I will introduce **JaCoCo**, a tool for generating code coverage reports for Java projects.

Let's get started

Add Plugins

We need to define JaCoCo maven plugin in pom.xml file.

```
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>0.8.8</version>
            </plugin>
        </plugins>
    </pluginManagement>
    <plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                </execution>
                <execution>
                    <id>report</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

For further details regarding JaCoCo Maven goals, you can refer to this resource: JaCoCo-Maven

Demo class

In this demonstration, we present a method for calculating shipping fees. The input parameter is the `weight`, which is used to determine the `shipping fee`.

```
public class ShippingService {
    public int calculateShippingFee(int weight) {
```

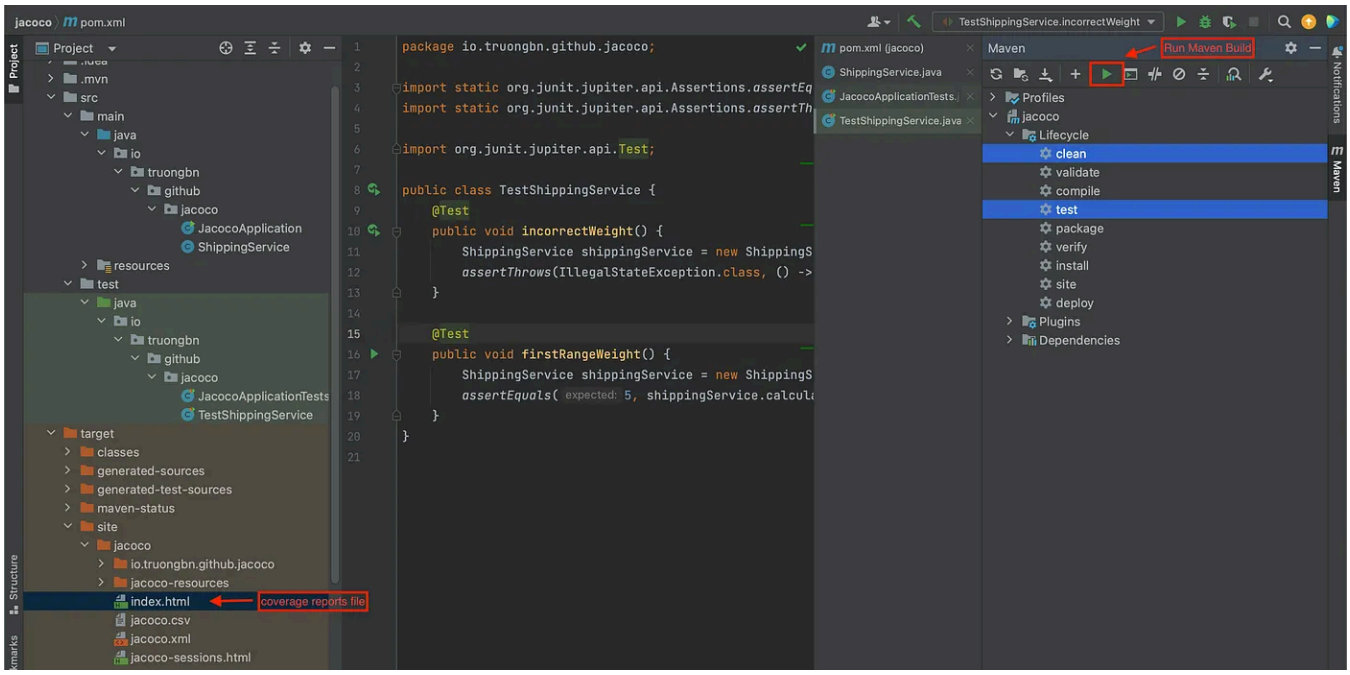
```
        if (weight <= 0) {
            throw new IllegalStateException("Please provide correct weight");
        }
        if (weight <= 2) {
            return 5;
        } else if (weight <= 5) {
            return 10;
        }
        return 15;
    }
}
```

Test class

```
public class TestShippingService {
    @Test
    public void incorrectWeight() {
        ShippingService shippingService = new ShippingService();
        assertThrows(IllegalStateException.class, () -> shippingService.calcula
    }

    @Test
```


Testing



Go to Maven, select `clean` and `test` commands then `Run Maven Build`

Once the test has been completed, `target/site/jacoco/index.html` contains all output. Let's open that file in Browser to see the details.

jacoco

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
io.truongbn.github.jacoco		60%		50%	3	7	5	11	1	4	0	2
Total	12 of 30	60%	3 of 6	50%	3	7	5	11	1	4	0	2

You can see in the image, **jacoco** is the project name, `io.truongbn.github.jacoco` is the package. It's showing that code has been covered 60%, and branches have been covered 50%. For further details regarding Jacoco properties, you can refer to this resource: Coverage Counters.

Open the package up, we have 2 classes `ShippingService` and `JacocoApplication`. Let's focus on `ShippingService`, inside it, code has been covered 68%, and 50% of branches have been covered.

io.truongbn.github.jacoco

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ShippingService		68%		50%	2	5	3	8	0	2	0	1
JacocoApplication		37%		n/a	1	2	2	3	1	2	0	1
Total	12 of 30	60%	3 of 6	50%	3	7	5	11	1	4	0	2

Go inside `ShippingService` class, and open `calculateShippingfee(int)` method, we have below result.

ShippingService.java

```
1.  package io.truongbn.github.jacoco;
2.
3.  public class ShippingService {
4.      public int calculateShippingFee(int weight) {
5.          if (weight < 0) {
6.              throw new IllegalStateException("Please provide correct weight");
7.          }
8.          if (weight <= 2) {
9.              return 5;
10.         } else if (weight <= 5) {
11.             return 10;
12.         }
13.         return 15;
14.     }
15. }
```

Jacoco is pretty showing clearly the different levels of coverage here. It's using Diamond icons with different colors to represent code coverage for branches. And using Background colors to represent code coverage for lines.

- Green diamond means all branches have been covered.
- Yellow diamond means the code has been partially covered - some untested branches.
- Red diamond means no branches have been exercised during the test.

The same color code applies to the background color, but for lines coverage.

Let's add some more code to cover the partially covered branch.

```
@Test
public void secondRangeWeight() {
    ShippingService shippingService = new ShippingService();
    assertEquals(10, shippingService.calculateShippingFee(4));
}
```

`Run Maven Build` with `clean` and `test` commands again, open test coverage for `calculateShippingfee(int)` method in Browser again. We'll have below result.

ShippingService.java

```
1.  package io.truongbn.github.jacoco;
2.
3.  public class ShippingService {
4.      public int calculateShippingFee(int weight) {
5.          if (weight < 0) {
6.              throw new IllegalStateException("Please provide correct weight");
7.          }
8.          if (weight <= 2) {
9.              return 5;
10.         } else if (weight <= 5) {
11.             return 10;
12.         }
13.         return 15;
14.     }
15. }
```

You can see that the yellow diamond is still there. that means we haven't covered the scenario with weight greater than 5. Let's add one more test case.

```
@Test
public void lastRangeWeight() {
    ShippingService shippingService = new ShippingService();
    assertEquals(15, shippingService.calculateShippingFee(10));
}
```

Run it again, then open the report coverage in Browser.

ShippingService.java

```
1.  package io.truongbn.github.jacoco;
2.
3.  public class ShippingService {
4.      public int calculateShippingFee(int weight) {
5.          if (weight < 0) {
6.              throw new IllegalStateException("Please provide correct weight");
7.          }
8.          if (weight <= 2) {
9.              return 5;
10.         } else if (weight <= 5) {
11.             return 10;
12.         }
13.         return 15;
14.     }
15. }
```

Now you can see that all scenario has been completely covered.

Bonus: You might notice that the class JacocoApplication is not very critical to the coverage reports. In some cases, the coverage of such classes can skew the overall code coverage report. In order to avoid such irrelevant classes from affecting the code coverage, we can exclude them by using the Jacoco plugin.

```
<plugins>
    <plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <configuration>
            <excludes>
                <exclude>io/truongbn/github/jacoco/JacocoApplication.class</exc
            </excludes>
        </configuration>
        ...
    </plugin>
</plugins>
```

For further details regarding JaCoCo excludes, you can refer to this resource: <excludes>

Now let's say we're using CI/CD for deploying the code, we probably like to verify how much percentage of lines of code coverage or percentage of code coverage has been done, etc. To do that we need to add one more execution in the Jacoco plugin.

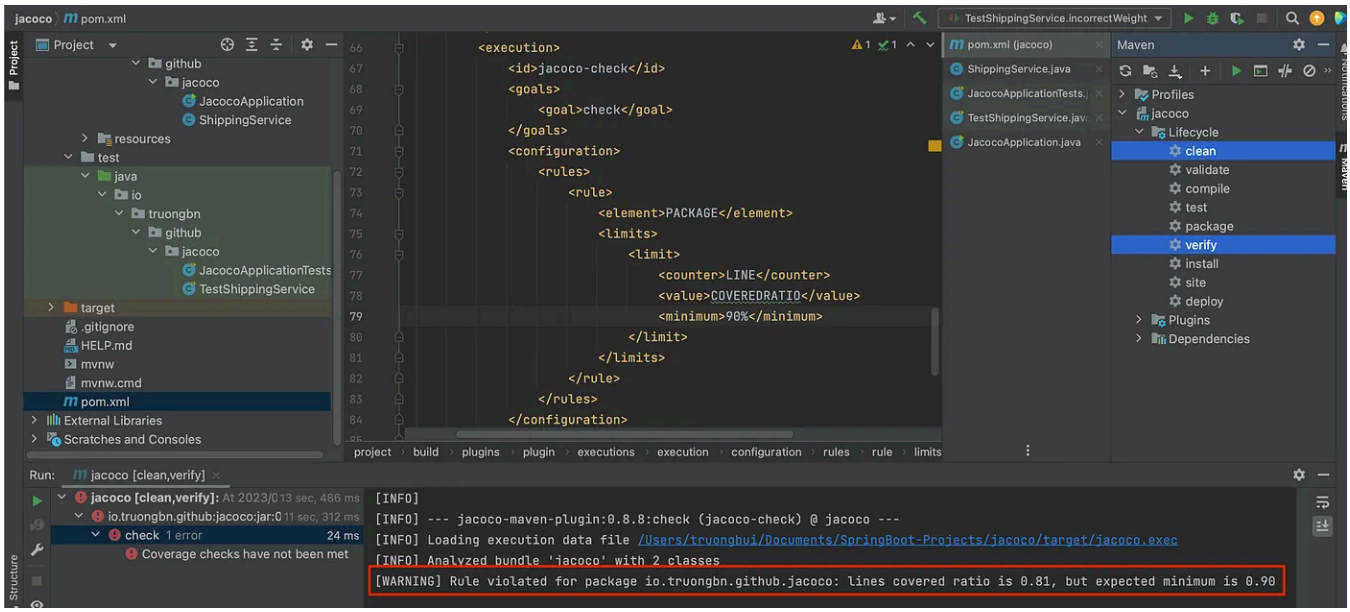
```
<execution>
    <id>jacoco-check</id>
    <goals>
        <goal>check</goal>
    </goals>
    <configuration>
        <rules>
            <rule>
                <element>PACKAGE</element>
                <limits>
                    <limit>
                        <counter>LINE</counter>
                        <value>COVEREDRATIO</value>
                        <minimum>90%</minimum>
                    </limit>
                </limits>
            </rule>
```

```
            </rules>
        </configuration>
    </execution>
```

In this execution, we're adding one rule. The rule is that for the PACKAGE, the count should be LINE and the LINE coverage minimum should be 90%

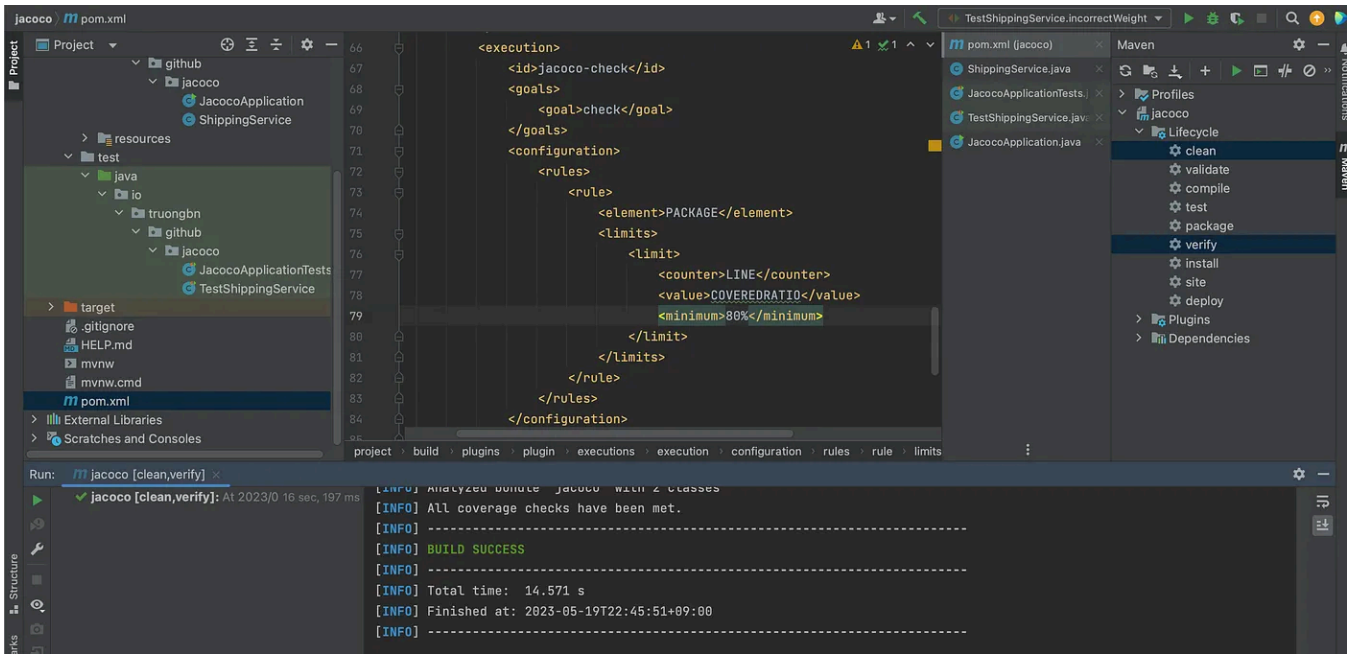
For further details regarding JaCoCo rules, you can refer to this resource: jacoco:check

Go to Maven, select `clean` and `verify` commands then `Run Maven Build` to check it.



You can see it failed. the reason clearly showed that the rule is violated "lines covered ratio is 0.81, but expected minimum is 0.90".

Now let's update LINE coverage minimum is 80%, run it again.



It's **BUILD SUCCESS** 😎

We have just conducted a brief demonstration to observe the behavior of JaCoCo, hope it's working as expected you guys!

Completed source code can be found in this GitHub repository: https://github.com/buingoctruong/springboot-jacoco

Happy learning!

Thank you for reading!

Spring Boot | Jacoco | Test Coverage | Code Quality | Code Coverage Tools

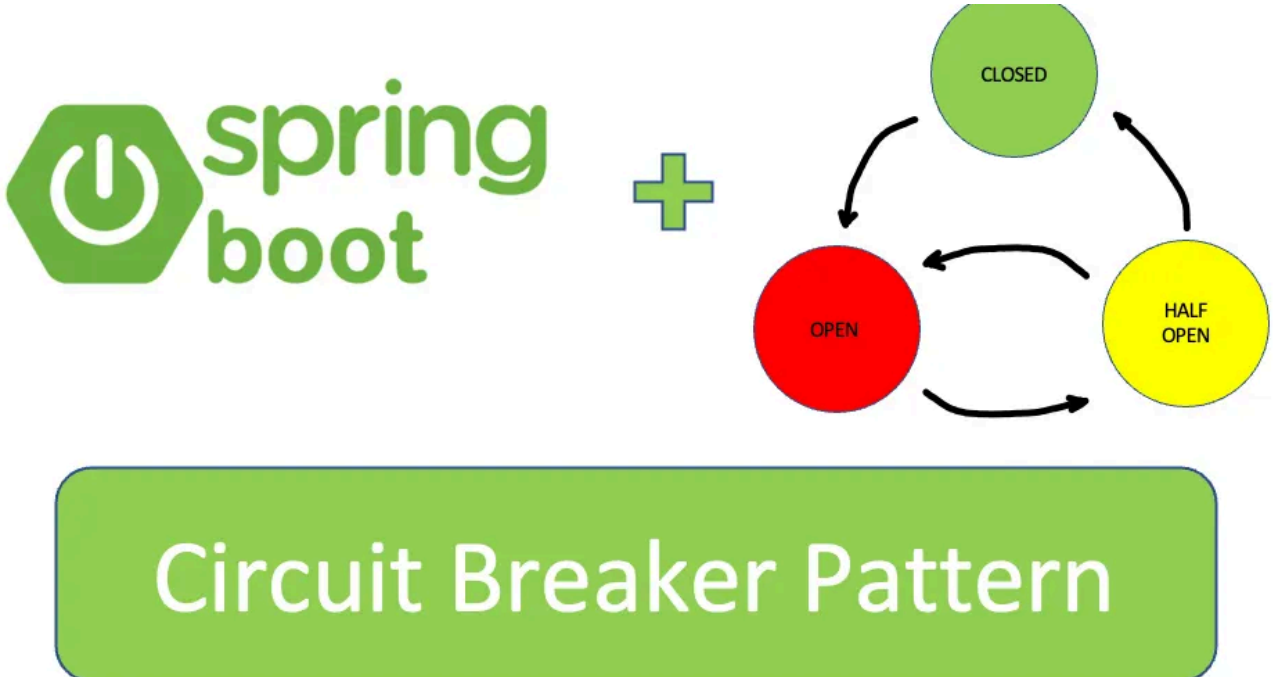
Follow

Written by Truong Bui

659 followers · 14 following

Software Engineer - Dreamer - Writer - Reader

More from Truong Bui



Truong Bui

Circuit Breaker Pattern in Spring Boot

Part of the Resilience4J Article Series: If you haven't read my other articles yet, please refer to the following links: 1. MicroService...

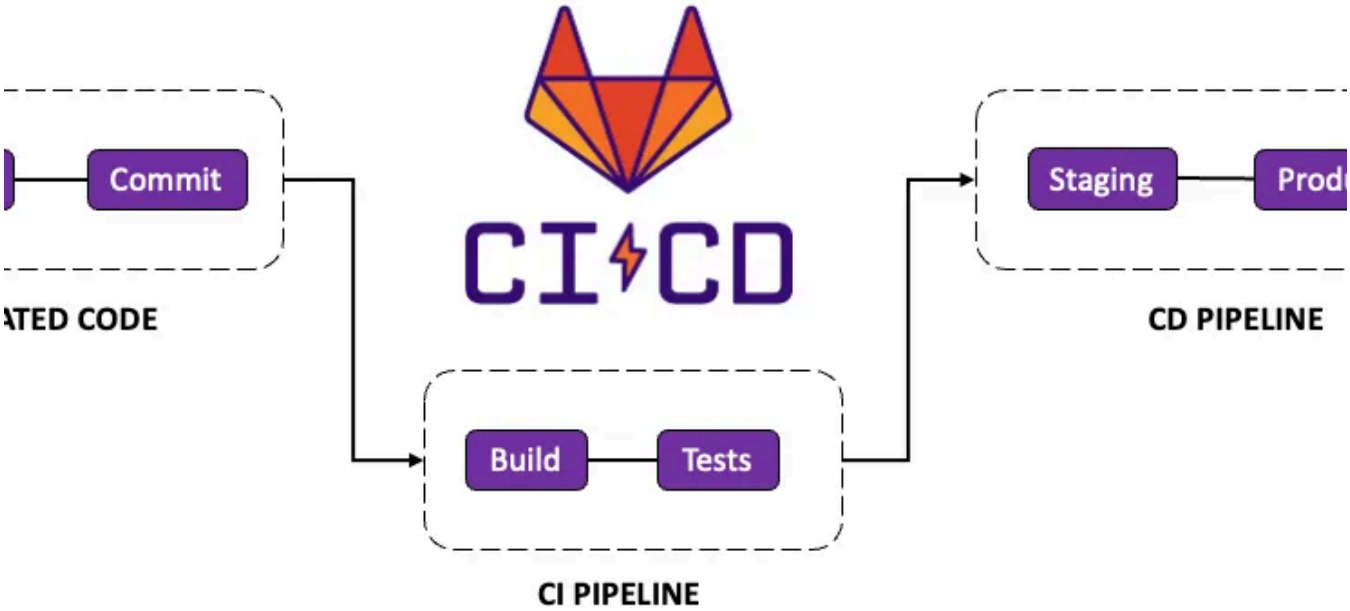
Apr 18, 2023 201 5

Truong Bui

MicroService Patterns: Rate Limiting with Spring Boot

Part of the Resilience4J Article Series: If you haven't read my other articles yet, please refer to the following links: 1. Circuit Breaker...

Aug 13, 2023 128 3



Truong Bui

Exploring GitLab CI/CD: A Practical Guide

Recently, in my self-directed learning journey, GitLab CI/CD caught my interest, prompting me to plan to write an article about it.

Jan 3, 2024 29 2

Truong Bui

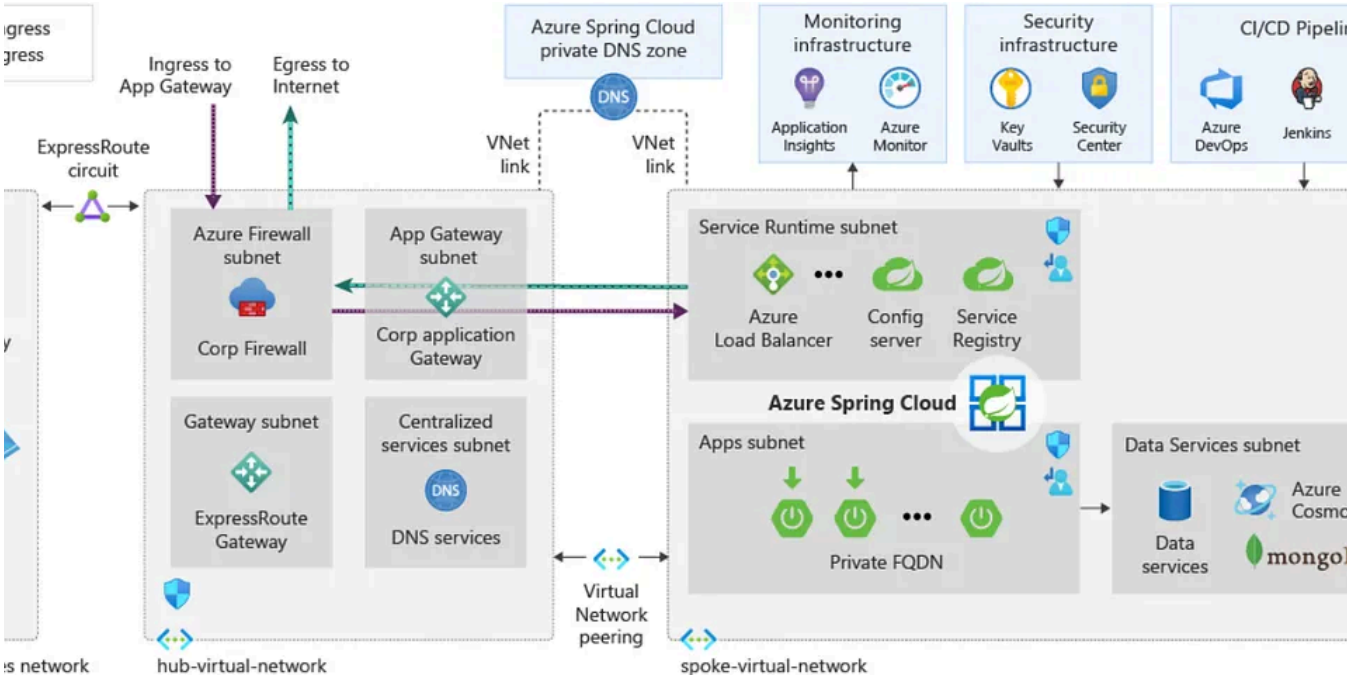
Structured Logging With Elastic Common Schema (ECS)

As you may already know logging is crucial in the software development lifecycle. It serves multiple purposes, including monitoring the...

Nov 27, 2023 25

See all from Truong Bui

Recommended from Medium

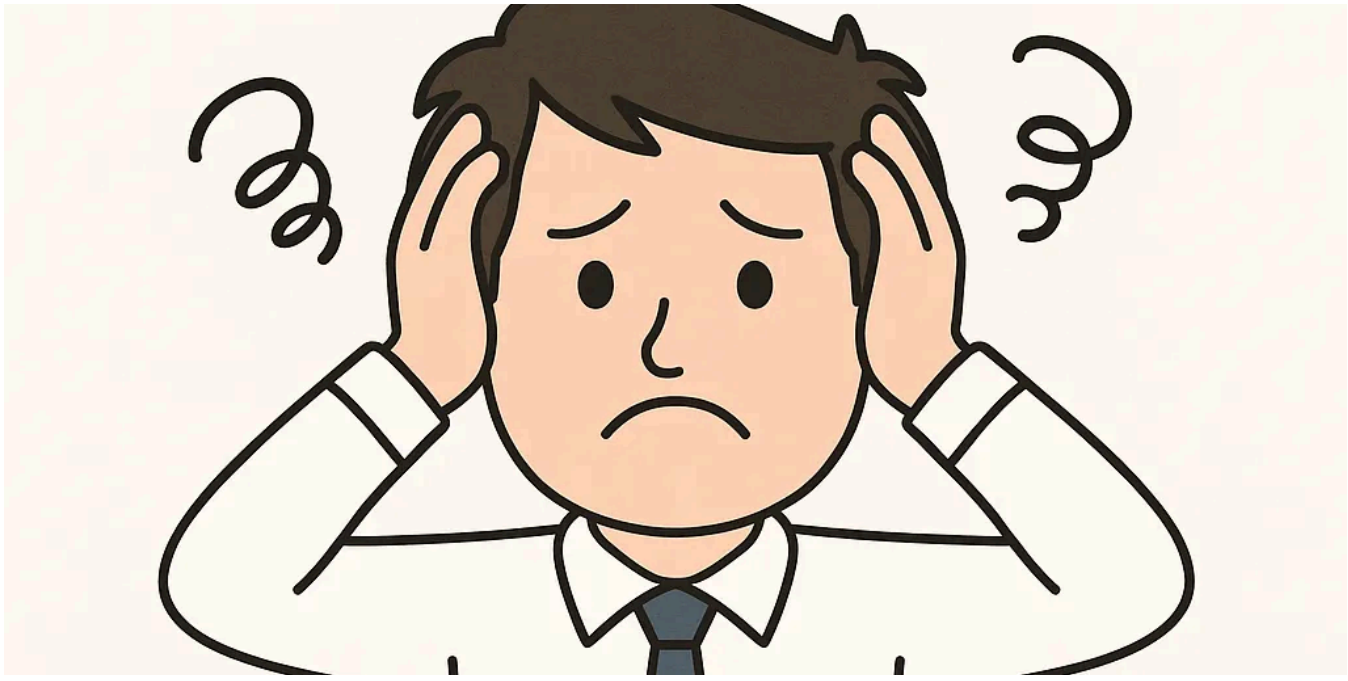


Jaytech

Game-Changing Java & Spring Boot Trends for 2026

Most important Java & Spring Boot trends for 2026 🚀

Jan 4 · 403 · 5

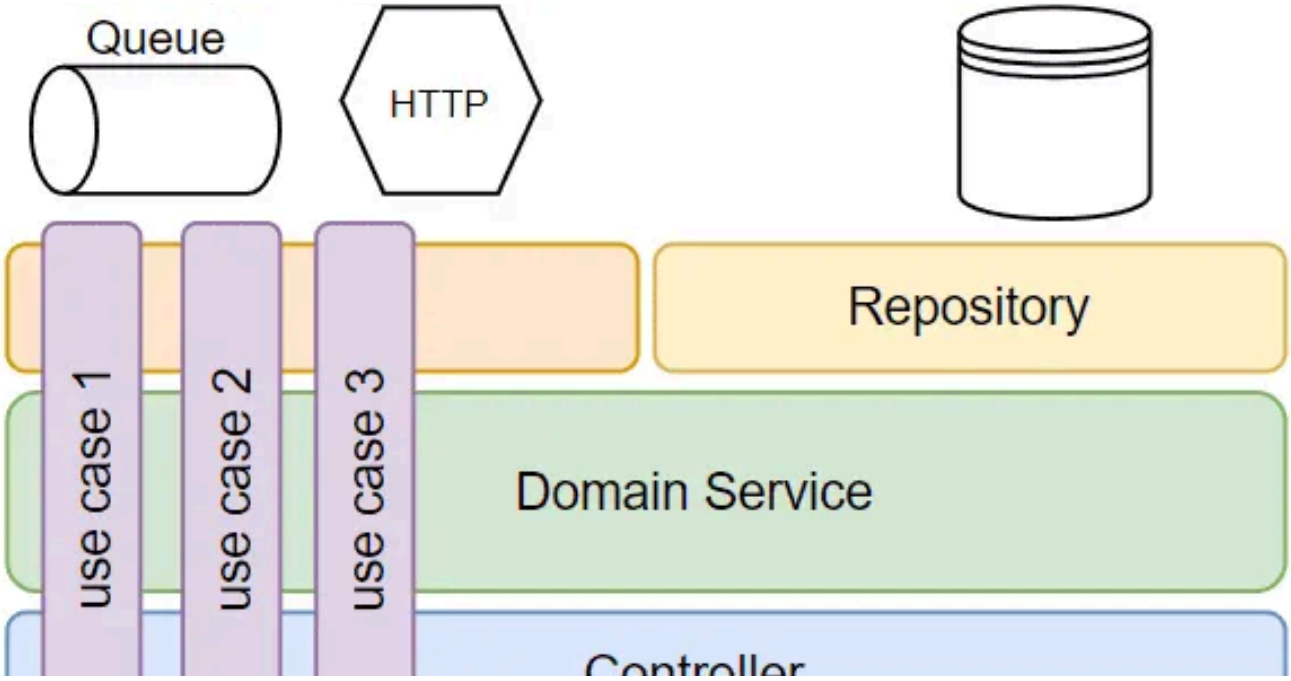


Umesh Kumar Yadav

Token, Session, Cookie, JWT, OAuth2—I can't tell the difference!

Recently, I've noticed that some people easily confuse the concepts of Token, Session, Cookie, JWT, and OAuth2.

Dec 10, 2025 179 2

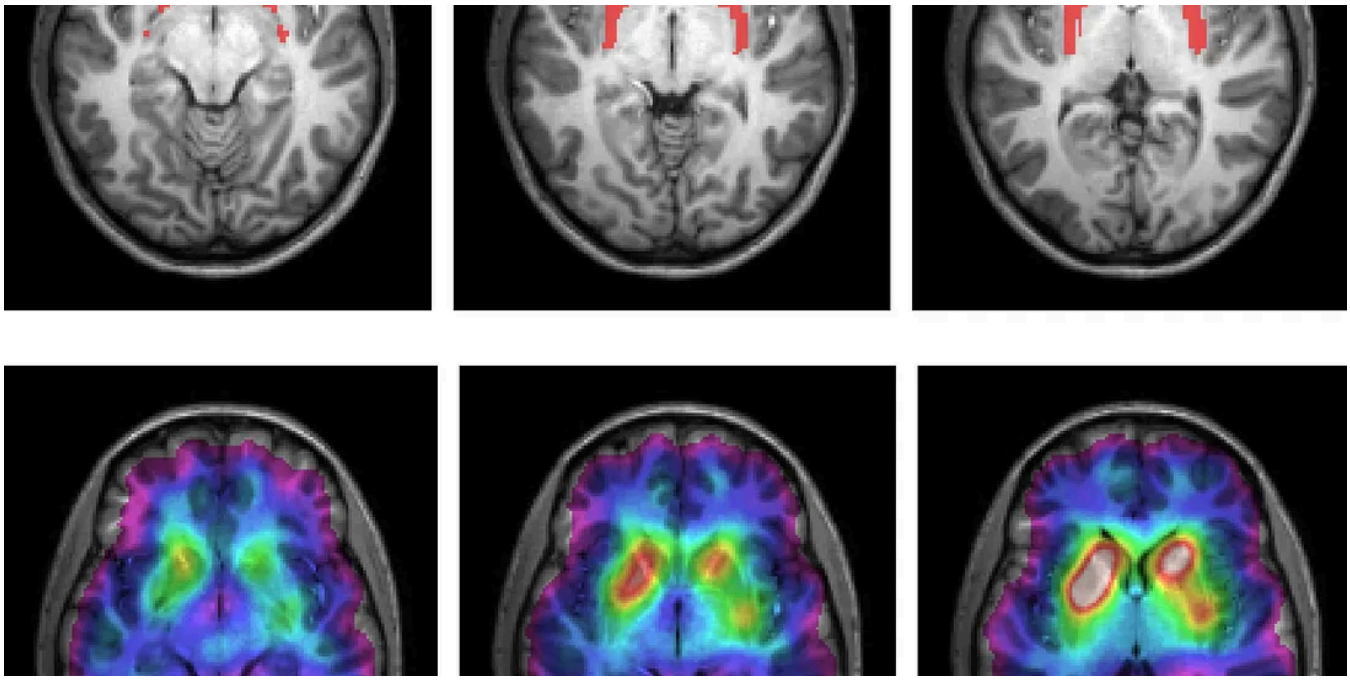


Reyanshicodes

Spring Boot Developers, It’s Time to Delete Your Service Layer

Let’s be honest: every Spring Boot project looks like déjà vu.

Oct 5, 2025 549 93



In Write A Catalyst by Dr. Patricia Schmidt

As a Neuroscientist, I Quit These 5 Morning Habits That Destroy Your Brain

Most people do #1 within 10 minutes of waking (and it sabotages your entire day)

Jan 15 20K 345

Joe Njenga

I Tried New Claude Code Ollama Workflow (It's Wild & Free)

Claude Code now works with Ollama, which takes the game to the next level for developers who want to work locally or need flexible model...

Jan 19 1.3K 19

Code With Sunil | Code Smarter, not harder

Understanding SSL, TLS and HTTPS: The Ultimate Guide

If you are not a Member — Read for free here