**Achieve best performance: Spring-Boot OR Node-JS?**

This article is based on a REAL use case scenario for a database intensive application, with focus on performance. The experimental results I’ve come across can help you choose one or the other.

Quite simply, the same application has been realized twice using different languages: JavaScript (Node-JS) and Java (Spring-Boot) with performance measuring executed in varied environments. All the code including deploy (docker-swarm) and testing environment (JMeter) is available at [link] – free for you to play with and customize.

Index:

1. Application and test description - what are we measuring and how
2. Results and conclusion

**1 – Application and tests description**

We will evaluate the performance of four REST JSON endpoints:

1. /find-all -- gets an increasing number of items from the DB
2. /save -- persists a new item on the DB
3. /get/{id} -- get an item from the DB
4. /cpu/{load} -- CPU intensive operation [https://gist.github.com/sqren/5083d73f184acae0c5b7]

To better simulate a real web application scenario, both applications includes:

* JWT validation for each endpoint
* Input DTO validation checks
* Log writing on file system

Load testing with Apache JMeter[link], simulating a discrete application load: 5 users simultaneously looping the four endpoint 20 times, for a total of 100 calls for each endpoint.   
Different tests have been run customizing the following parameters:

1. Percentage of CPU occupation
2. Application’s scaling (using docker swarm increasing replica-set)
3. Memory and CPU reservation for each application

Test metrics: average time response for each method. As a bonus I’ve measured up also container startup time and image build time with this useful script: [link]

Database RDBMS PostgreSQL with a very simple schema [link] <https://gist.github.com/GaetanoPiazzolla/5ee8164d349440cb1f30543940945774>

**2 – Results**

Here are proposed some graphics extrapolated from the complete results [https://github.com/GaetanoPiazzolla/spring-boot-vs-node-js/blob/main/test/results.xlsx] you can find in the repository.

1- The first graphic shows the total time of the JMETER execution percentage, in 4 different scenarios with increasing CPU operations. As it may seem obvious, NodeJS slows down rather a lot compared to Node. The more CPU intensive “blocking” operations are done on the container, the more Java benefits from thread usages, so performance degradation is mitigated.

We know this: time complexity can be reduced significantly by the use of an additional efficient data structure – Threads! This of course results in more memory usage in Java compared to JavaScript: I had to increase the memory of the containers when the spring application started using threads (starting from case 3).

2- What happens if we scale horizontally the application increasing container replicas?

As the NodeJS application suffers more from increasing CPU blocking operations, it benefits more from scaling up. We can see a decrease of almost 50% of time spent increasing just one container, from case 1 to case 2.

We can assume that NodeJS do scales better in this particular case, which is similar to a real case scenario in which there is a cpu intensive request and other database intensive requests.

To sum up, the choice of the better framework to pick in terms of performance is obviously guided by the kind of application you are building. From this simple test I’ve confirmed the following assumptions: If you expect a lot of blocking CPU calls, go with Spring. If you have low memory requirements, build the application using Node and If you have to access the persistence layer a lot, the best choice is Node.

Bonus:

Using this nice script <https://gist.github.com/GaetanoPiazzolla/956470742577f60425fa06784d39eb29> I’ve measured up building time of the containers:

* Spring: 2 minutes
* NodeJs: 6 seconds

It’s a really huge difference if you somehow need to build images very often.

When talking about readiness time of container (how much time we have to wait until the container is ready to answer REST calls)

* Spring: 20 seconds
* Node: 2 seconds

Node as good advantage also in this case.