Enterprise Software Platforms

**PREPARED FOR**

A large Dutch banking organisation

**PREPARED BY**

Maarten Blömer

Nick Krijgsman

Faruk Aydin

Jursley Gonzalez

Vincent Andersen

# Table of Contents

[**Table of Contents**](#_z6lhuq9emniv) **1**

[**Case Study**](#_k6jx82pctwds) **3**

[Case goal:](#_bpxkra1gvikb) 3

[Case problems:](#_tz830mgo6nzk) 3

[Case questions:](#_3vn6lbsuph60) 3

[Brainstorming:](#_llbf4q7qnu37) 4

[Analyse and structure](#_9hr2189czuvy) 5

[Formulate learning objectives](#_rhp1wv31n7xy) 5

[**Independent study**](#_zc6h8lveooxw) **6**

[Quarkus(Faruk)](#_eaou76j8xww7) 6

[Cost of the product/License constraints](#_sef113z4ticn) 6

[Maturity of the product & Product roadmap quality](#_2u0ifd6pespc) 6

[Active community & support options](#_2ye0kd24i0k) 6

[Documentation quality](#_9yqdkuieleqe) 6

[Does it support the expected load (users, requests, data volume, etc.)](#_xslrh8nr6mkr) 7

[Ease of use](#_vhkc8jpd79r0) 7

[Ease of scalability(Kubernetes/Docker integration, usage of cloud services)](#_eqeyds531d5n) 7

[Security (Ease of security additions and own security)](#_ylrbh7uo46jb) 7

[Library support](#_lw2191gt9lww) 7

[Deployment speed](#_k7qivt97a01r) 7

[Java EE 8 (Jakarta EE 9)](#_97ho52d59gzk) 9

[Cost of the product/License constraints](#_pmy5zc64uz5i) 9

[Maturity of the product & Product roadmap quality](#_a0bo34phr0j1) 9

[Active community & support options](#_3jiemizdwhkf) 9

[Documentation quality](#_lsy32b92hw6) 9

[Does it support the expected load (users, requests, data volume, etc.)](#_94zjp4pjtqfs) 9

[Ease of use](#_ufmqraoqmmjl) 9

[Ease of scalability(Kubernetes/Docker integration, usage of cloud services)](#_r2m4m7mssmvv) 9

[Security (Ease of security additions and own security)](#_jku58b6gt1vd) 10

[Library support](#_sx2tjzcvoizi) 10

[Deployment speed](#_zv1lz2jw9vb) 10

[.NET 5 (Vincent)](#_v2xj76ghlwj7) 11

[Cost of the product/License constraints](#_x5f40km7399v) 11

[Maturity of the product & Product roadmap quality  
.NET 5](#_tampgopfwzf0) 11

[Active community & support options](#_sy0bo6giogvg) 11

[Documentation quality](#_ovowfx5jo720) 11

[Does it support the expected load (users, requests, data volume, etc.)](#_9raqbb7a667v) 12

[Ease of use](#_kq2nij44edqi) 12

[Ease of scalability(Kubernetes/Docker integration, usage of cloud services)](#_4xsgqv2k5ndm) 12

[Security (Ease of security additions and own security)](#_g735t5s5knbu) 12

[Library support](#_l9xh6hp7bxl) 12

[Deployment speed](#_qstfyqwa52on) 12

[Spring Boot (Nick)](#_qo5qk3cr0w3h) 13

[Cost of the product/License constraints (10)](#_iawno5pysmb7) 13

[Maturity of the product & Product roadmap quality](#_v9js8lodvann) 13

[Active community & support options](#_ecfarp6tk7gf) 13

[Documentation quality](#_72d8ugv2slgl) 13

[Does it support the expected load (users, requests, data volume, etc.)](#_azz0fl8bpuc5) 13

[Ease of use](#_xkmtbiyd7bzy) 13

[Ease of scalability(Kubernetes/Docker integration, usage of cloud services)](#_9ck0tl5f4qmt) 13

[Security (Ease of security additions and own security)](#_y9e326xv3n1f) 14

[Library support](#_qrveemsscat9) 14

[Deployment speed](#_elpfd69oy6qp) 14

[JHipster](#_505gm0evad4t) 15

[Cost of the product/License constraints](#_6ln247hon8op) 15

[Maturity of the product & Product roadmap quality](#_qhfjb6yn8vzk) 15

[Active community & support options](#_2x4qd8mbeviz) 15

[Documentation quality](#_bfe45nygg76l) 15

[Does it support the expected load (users, requests, data volume, etc.)](#_m9d6qum00t5d) 15

[Ease of use](#_fqdbtvcxjaat) 15

[Ease of scalability(Kubernetes/Docker integration, usage of cloud services)](#_sedocs9utbwh) 16

[Security (Ease of security additions and own security)](#_md2oueb72uht) 16

[Library support](#_tnyfrc3fp6to) 17

[Deployment speed](#_43ndmkqrv6hj) 17

[**Matrix**](#_ms14khqko54c) **18**

[**Sources**](#_qwibiw7iqp2l) **20**

# Case Study

## Case goal:

Digital transformation in the form of new applications that enable customers to perform extensive self-service tasks.

## Case problems:

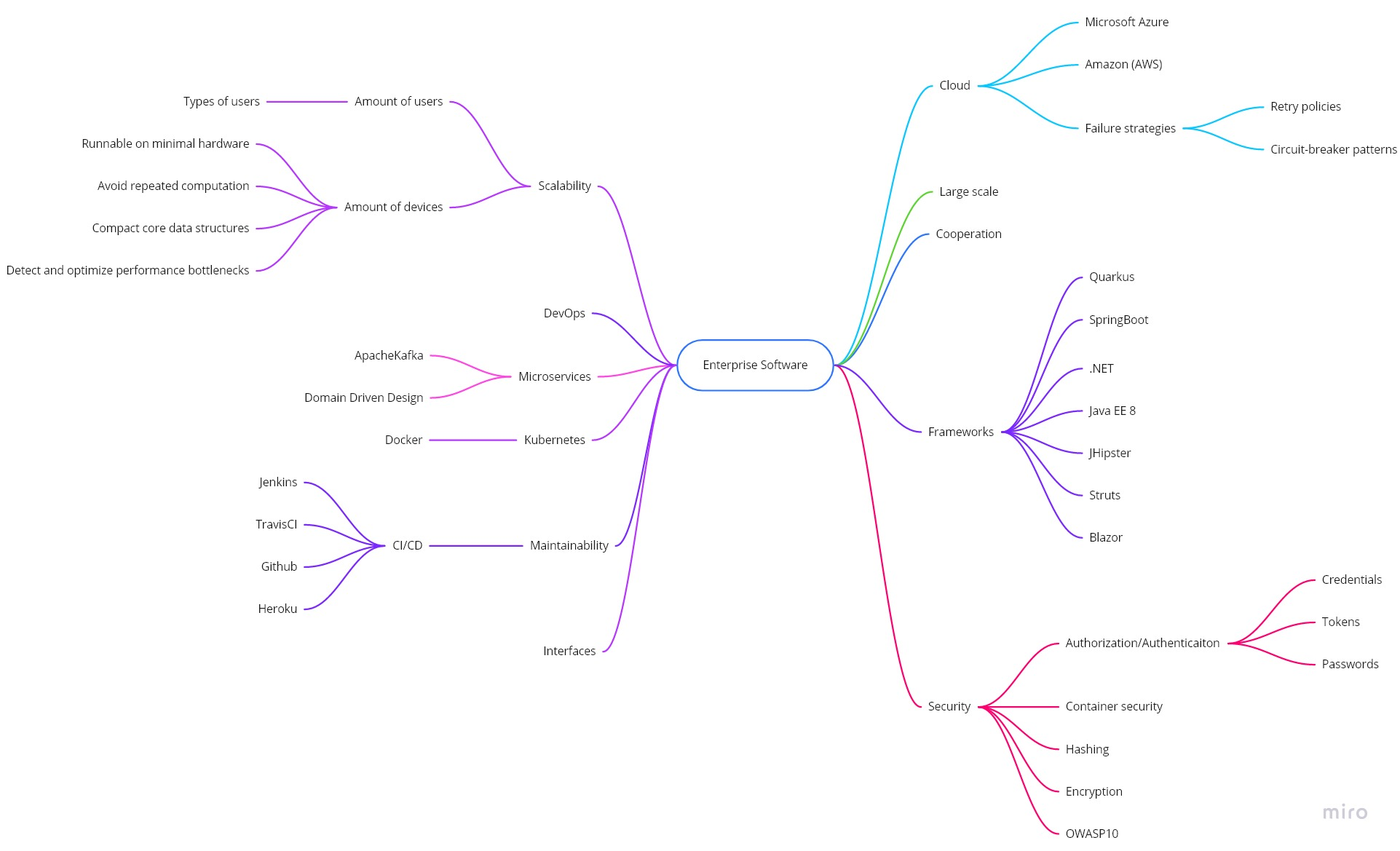
* New applications that enable customers to perform extensive self-service tasks.
* Extending current legacy backend services is hard because these are written in COBOL
* The COBOL engineers are already retired

## Case questions:

* How is the speed of deployment between different frameworks and does it matter?

## 

## Brainstorming:



## Analyse and structure

* Many different frameworks, CICD platforms, security options etc. to choose from and combine with each other.
* A banking application needs more focus on certain parts of the application such as security for both users and containers that communicate with each other.
* With the rising popularity of cryptocurrencies, the bank application may need to be scalable to accept transactions in these forms.

## Formulate learning objectives

The learning objectives for this study case is to understand different frameworks and compare them to each other.

* Quarkus (Faruk)
* Java EE 8 (Maarten)
* .NET (Vincent)
* Spring Boot (Nick)
* JHipster

# 

# Independent study

## Quarkus(Faruk)

### Cost of the product/License constraints

Quarkus is an open-source, free to use Java framework. All dependencies are available under the Apache Software License 2.0 or compatible license.

### Maturity of the product & Product roadmap quality

First release: November 6, 2019

Last release: February 11, 2021

Roadmap: <https://github.com/orgs/quarkusio/projects/5>

We can see that Quarkus has a lot of backlog items. These are potential features. They are also working on some fixes and requests for the next update.

### Active community & support options

Github stars: 7k <https://github.com/quarkusio/quarkus>

Twitter followers: 10,8k <https://twitter.com/quarkusio>

Quarkus has an active community. They indicate Vulnerabilities . They can also work together to solve problems. And the community is able to request new features. The accepted requests will be added to the roadmap.

### Documentation quality

There are a lot of options to get information from. First of all, the getting started page. This will help beginners to set up their environment. They also have guides on specific subjects. Some subjects are: Core, Security, Messaging, Serialization and a lot more. There is also a FaQ support for some common issues. And Quarkus is also giving free tutorials on YouTube.

### Does it support the expected load (users, requests, data volume, etc.)

### Ease of use

Quarkus was designed to be easy to use right from the start, with features that work well with little to no configuration. You can start writing code in 3 steps.

1. Select your IDE (IntelliJ IDEA, Eclipse, VSCode, Vim or Emacs)
2. You need a JDK 8 or 11+ (Optionally get GraalVM 20.3.1 for native compilation)
3. You need Apache Maven 3.6.2+ or Gradle.

Because there are lots of guides and tutorials you will learn quickly.

On <https://code.quarkus.io/> you can create a project with the dependencies you need.

### Ease of scalability(Kubernetes/Docker integration, usage of cloud services)

Quarkus was built around a container-first philosophy, meaning it’s optimized for lower memory usage and faster startup times. Quarkus builds applications to consume 1/10th the memory when compared to traditional Java, and has a faster startup time (as much as 300 times faster), both of which greatly reduce the cost of cloud resources.

### Security (Ease of security additions and own security)

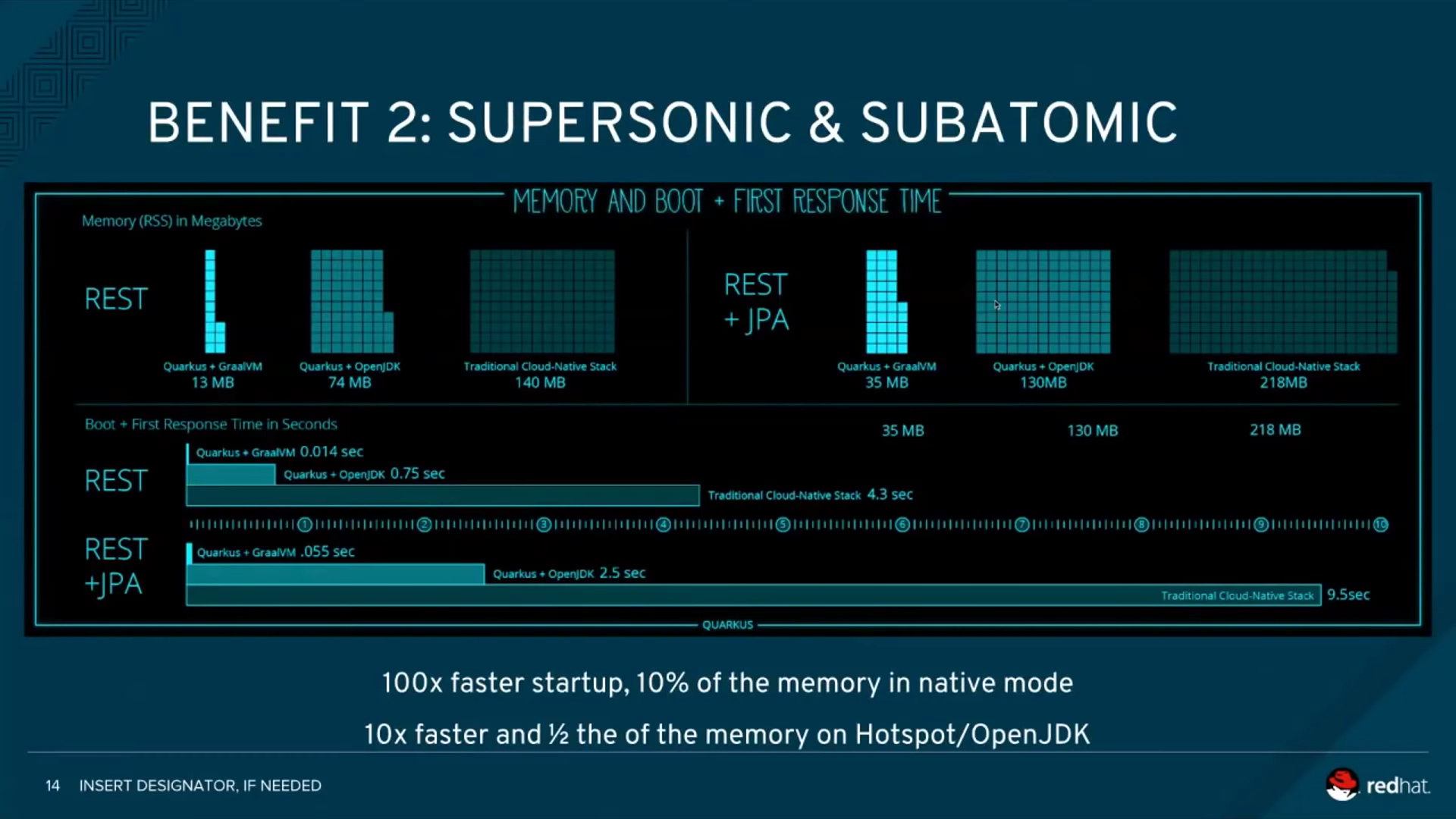
Quarkus Security provides the architecture, multiple authentication and authorization mechanisms, and other tools for the developers. The developer needs to make the decisions. He/She needs to know about the consequences of the libraries that are going to be used.

### Library support

Quarkus has a lot of libraries to use. It is easy to find and read about them on the guides page or YouTube.

### Deployment speed

Developer mode compiles in 1.5 seconds. Zero configuration and live reloading.



### 

## Java EE 8 (Jakarta EE 9)

### Cost of the product/License constraints

Now better known as Jakarta EE. Free version available. Licensed under Eclipse Public License or GNU General Public License.

### Maturity of the product & Product roadmap quality

Started under the name J2EE 1.2 in December 1999, and is now known under the name Jakarta EE 9 as of November 2020.  
Jakarta’s full roadmap can be viewed here: <https://eclipse-ee4j.github.io/jakartaee-platform/jakartaee9/JakartaEE9ReleasePlan>

### Active community & support options

Google groups community: <https://groups.google.com/g/jakartaee-ambassadors>

Twitter: <https://twitter.com/jee_ambassadors> (12.8k Followers)

Also a part of the Eclipse Foundation: <https://www.eclipse.org/> (330+ members)

Many other options: <https://jakarta.ee/connect/> (Socials, etc)

### Documentation quality

Very broad documentation hosted on their github page. <https://eclipse-ee4j.github.io/jakartaee-tutorial/toc.html> this page is also constantly being updated and worked on.

### Does it support the expected load (users, requests, data volume, etc.)

Has many APIs for things like security, authorization, authentication, mail, messaging, etc. Laying a solid foundation to support a big enterprise project.

### Ease of use

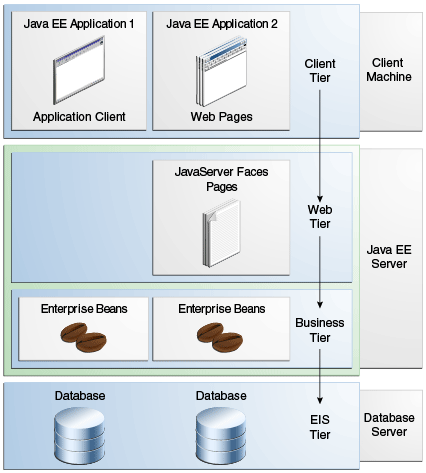
Compatibility with Eclipse GlassFish IDE. Required software: Any Java IDE, as per recommendation that would be Eclipse or NetBeans, Apache Maven is also required. There is a clickthrough tutorial on their github page that is easy to follow. <https://eclipse-ee4j.github.io/jakartaee-tutorial/usingexamples001.html#GEXAJ>

### Ease of scalability(Kubernetes/Docker integration, usage of cloud services)

Jakarta EE uses a native Java cloud service, together with WildFly you can run your Jakarta EE project in a docker or kubernetes container.

### Security (Ease of security additions and own security)

<https://eclipse-ee4j.github.io/jakartaee-tutorial/overview004.html#BNABA> Multiple security components in play already. Multilayer structure for applications.



For their web services support they use a lot of XML and SOAP transport protocols.

### Library support

Supports multiple Java libraries, can easily be implemented. Has a lot of built in things

### Deployment speed

Jakarta EE is mostly used for big monolithic enterprise structures thus it requires a bit more time to deploy compared to smaller projects using Spring boot.

## .NET 5 / .NET Core (Vincent)

### Cost of the product/License constraints

.NET 5 is part of the [open-source .NET platform](https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor#:~:text=Blazor%20is%20part%20of%20the,costs%2C%20including%20for%20commercial%20use.) which means that there are no fees or licensing costs, including for commercial use.

### 

### Maturity of the product & Product roadmap quality .NET 5

ASP.NET was initially released in January 5, 2002 with its stable release 4.8 in April 18, 2019. .NET 5 however is newer and more suitable for cloud-based web apps with its initial release with 1.0 in 27th June, 2016 and its latest stable release v5.0.2 on January 12, 2021. The current versions are planned to have long term support and the roadmap is available on their [github page](https://github.com/dotnet/core/blob/master/roadmap.md).

### Active community & support options

.NET 5 features a long list of support options:

* Cross-platform compatibility
* Ability to host on Kestrel, IIS, HTTP.sys, Nginx, Apache and Docker
* Usage of blazor and razor for frontend development
* High performance and made for scalable systems

And many more functionalities which are viewable [here](https://docs.microsoft.com/en-us/dotnet/core/dotnet-five).

### 

The community behind .NET 5 is also very active with 31,753 questions tagged .net core on [stackoverflow](https://stackoverflow.com/questions/tagged/.net-core). This makes it easy to find already answered questions when faced with a problem.

### Documentation quality

.NET 5 is very well documented by [Microsoft themselves](https://docs.microsoft.com/en-us/dotnet/) varying from beginner tutorials to more advanced topics and sample code to go with all of them.

### Does it support the expected load (users, requests, data volume, etc.)

See “[Ease of scalability](#_4xsgqv2k5ndm)”.

Because of the reasons given for the ease of scalability, any expected load should be well handled while being generally cheaper in comparison to other frameworks.

### Ease of use

Since .NET 5 is based on C#, it offers easy, readable code that can be used all the way to the frontend. Using the given documentation by Microsoft and having prior knowledge of C#, beginning to use .NET 5 should be very easy.

In terms of support, Microsoft has many different tutorials which also cover topics like [Web Api’s](https://dotnet.microsoft.com/apps/aspnet/apis) but also short 15 minute startup projects to keep the barrier of entry low.

### Ease of scalability(Kubernetes/Docker integration, usage of cloud services)

Microsoft offers .NET 5 [tutorials](https://dotnet.microsoft.com/apps/aspnet/microservices) [,guides](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/) and documentation on how to approach a docker based application development with a microservices based architecture.  
.NET 5 ranks #2 at the time of writing in terms of performance on [Techempower.com](https://www.techempower.com/benchmarks/#hw=ph&test=plaintext) . Using this framework, systems run with a much lower number of servers/VM’s which also reduces cost in hosting per microservice which in turn makes it more scalable from a price/performance ratio.

### Security (Ease of security additions and own security)

.NET 5 provides many different tools and libraries for security but 3rd party identity services are also readily available to use. ASP.NET contains features for managing authentication, authorization, data protection, HTTPS enforcement, app secrets, XSRF/CSRF prevention, and CORS management but development of these options is entirely up to the developer.

### 

### Library support

.NET 5 supports a wide variety of existing libraries that are supported within the .NET landscape.

### Deployment speed

According to the benchmarks performed in [this microsoft devlog](https://devblogs.microsoft.com/aspnet/grpc-performance-improvements-in-net-5/), .NET 5 is very performative compared to other web frameworks which implies a much better cost/performance ratio and deployment speed.

## Spring Boot (Nick)

### Cost of the product/License constraints

Spring boot is open source, which is free to use. Even the Spring Tools Suite for Eclipse is free.

### Maturity of the product & Product roadmap quality

First release: 1-4-2014.

Spring boot is still active on maintaining and developing their open source code. More info can be found at their [GitHub page](https://github.com/spring-projects/spring-boot).

### Active community & support options

The Spring Boot community is an active community. The last posted issues are posted today (22-2-2021) and the latest release was on 18-2-2021.

### Documentation quality

There is so much documentation on Spring Boot. The Spring Boot site and the GitHub page hold a lot of information. There are also a lot of people that have written blogs or articles on Spring Boot, it being tutorials or comparisons with other frameworks. Even on [StackOverflow](https://stackoverflow.com/) Spring Boot is a popular framework. With almost 100k results on the tag “spring-boot” and another 200k results on the tag “spring”.

### Does it support the expected load (users, requests, data volume, etc.)

Spring Boot is a solid framework that can be used for large enterprise solutions.

### Ease of use

Spring boot is a very easy to use framework with a flat learning curve. There is a lot of documentation to help the developer create and maintain a Spring Boot application. For example, on <https://start.spring.io/> you can create a Spring Boot application with all the dependencies you need.

### Ease of scalability(Kubernetes/Docker integration, usage of cloud services)

While it is easy to scale up on a Spring Boot application, it is not the fastest on startup and it is quite heavy on the memory. This is one of the flaws of working with a Java based framework.

Spring has thought of cloud solutions and has project examples and integration so that the applications can be easily deployed with Kubernetes or other cloud solutions.

### Security (Ease of security additions and own security)

Spring Boot has developed its own security support called “Spring Boot Security”. It is a dependency that can be added to a Spring Boot application. Developers are free to customize all kinds of security options.

### Library support

Spring Boot has support for all your needs, see <https://spring.io/projects/spring-boot> for more information.

### Deployment speed

Spring boot is not the fastest on startup and it is quite heavy on the memory. This is one of the flaws of working with a Java based framework.

## JHipster (Jursley)

### Cost of the product/License constraints

JHipster is a free and open-source application generator written in java. Making JHipster have no costs to be used.

### Maturity of the product & Product roadmap quality

Initial release:

* 21 October, 2013

Stable release:

* version 6.8.0 on March 8, 2020

They don’t have specific roadmaps to showcase their plans. Because it being a open source project with multiple contributors. They mostly have to do lists in the issues section where you can see what they’re goals are for the next version. <https://github.com/jhipster/generator-jhipster/issues/10958>

### Active community & support options

Questions on how to use JHipster can be posted on Stack Overflow with the “jhipster” tag.

Reporting issues or feature requests can be submitted on their GitHub issues page.

JHipster has the option to contact their main contributors and other users through a chat room on Gitter.im.

### Documentation quality

Documentation on JHipster is minimal on their own website, because of the use of multiple frameworks, apis and microservices. The current documentation is on how to setup the different types of technology stacks with reference to the documentation of the used technology.

### Does it support the expected load (users, requests, data volume, etc.)

### Ease of use

Technologies used by JHipster have their default configuration and best practices used as much as possible, making it easier to understand how JHipster works when the default configurations are the same as in the documentation.

JHipster has many options when generating a project, they only add a option if the implementation of the technology is complex and imply configuring or coding several components. The setup of a JHipster project is straight forward, you can create a technology stack by adding the options that you want to be able to use during development.

JHipster is compatible with known IDE’s like Eclipse, Microsoft Visual Studio and Atom.

### Ease of scalability(Kubernetes/Docker integration, usage of cloud services)

JHipster’s first question when generating a project is if you want to generate a monolithic or microservice architecture.

The foundation of microservices is built on the cloud and virtualization, both of which include scalability as a core feature. Virtualization enables servers to be created and provisioned at the push of a button as demand spikes, and simply turned off and deleted when the need subsides.

By using the microservices architecture it gives a JHipster project the option to only scale a microservice that has an increase or decrease in load.

### Security (Ease of security additions and own security)

JHipster makes use of Spring Security. Spring Security is an authentication and access-control framework and can be easily included in a Spring Boot application, in this case a JHipster application.

It focuses and covers the following topics:

1. Use HTTPS in production
2. Test your dependencies and find Spring Boot vulnerabilities
3. Enable CSRF protection
4. Use a content security policy for Spring Boot XSS protection
5. Use OpenID Connect for authentication
6. Use password hashing
7. Use the latest releases
8. Store secrets securely
9. Pen test your app
10. Have your security team do a code reviews

It gives developers the option to easily add security options and customize these.

### Library support

JHipster has full on library support as an application generator with different options. It is possible to add your own options if these are not included in the default configurations

### Deployment speed

A JHipster application can be deployed anywhere a Spring Boot application can be deployed. To build an optimized JavaScript client you can invoke this by using webpack. This will increase the build speed time which will result in added time to the build step during deployment.

# Matrix

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Evaluation Matrix** | **Weight** | **Quarkus** | **Jakarta EE 9** | **.NET 5** | **Spring Boot** | **JHipster** |
| **Cost/License constraints** | 0.05 | **0.5** (10) | **0.4** (8) | **0.5** (10) | **0.5** (10) | **0.4** (8) |
| **Maturity of the product** | 0.05 | **0.25** (5) | **0.5** (10) | **0.45** (9) | **0.4** (8) | **0.45** (9) |
| **Active community** | 0.05 | **0.35** (7) | **0.35** (7) | **0.35** (7) | **0.4** (8) | **0.35** (7) |
| **Documentation quality** | 0.05 | **0.5** (10) | **0.45** (9) | **0.45** (9) | **0.45** (9) | **0.15** (3) |
| **Load support (users, requests, data volume, etc.)** | 0.2 | **1.6** (8) | **1.6** (8) | **1.8** (9) | **1.6** (8) | **1.6** (8) |
| **Ease of use** | 0.05 | **0.45** (9) | **0.3** (6) | **0.45** (9) | **0.45** (9) | **0.45** (9) |
| **Ease of Scalability** | 0.15 | **1.35** (9) | **1.2** (8) | **1.35** (9) | **1.05** (7) | **1.35** (9) |
| **Security** | 0.2 | **1.4** (7) | **1.6** (8) | **1.6** (8) | **1.6** (8) | **1.6** (8) |
| **Library support** | 0.05 | **0.45** (9) | **0.35** (7) | **0.45** (9) | **0.45** (9) | **0.45** (9) |
| **Deployment speed** | 0.15 | **1.2** (8) | **0.75** (5) | **1.05** (7) | **0.75** (5) | **0.9** (6) |
| Total score: | 1 | **8.05** (82) | **7.5** (76) | **8.45** (86) | **7,65** (81) | **7.7** (78) |

**Advice**

Based on the [Matrix](#_ms14khqko54c) alone which scores the following frameworks as such:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Framework** | Quarkus | Jakarta EE 9 | **.NET 5** | Spring Boot | JHipster |
| **Score** | 82 | 76 | **86** | 81 | 76 |

We can conclude that .NET 5 is the leading framework over the runner up Quarkus, which only lost points over the maturity of the product.

Compared to the other frameworks, the .NET boot up times are relatively faster. This is why we advise the .NET framework over some of the Java based frameworks like Spring Boot and Jakarta EE. The reasoning behind not choosing JHipster is that it is a tool to initialize a standard application that is based on Spring Boot. It has a default configuration, in the banking world this is not feasible. Quarkus is an up and coming framework with lots of support, but due to the low maturity level it is prone to have major issues and changes. In the banking world is it essential to be reliable and robust. For this reason we wouldn't advise Quarkus.

.NET has long term support from Microsoft with close integration with Microsoft Azure in order for an easy, robust and reliable solution to the given problem.

In terms of security, .NET features many tools and libraries such as NWebsec which contains libraries that work together to remove version headers, control cache headers, stop potentially dangerous redirects, and set important security headers.   
On top of this, .NET provides built-in identity providers and you can easily manage app secrets, which are a way to store and use confidential information without having to expose it in the code.

For data protection .NET provides a data protection stack which contains an easy to use cryptographic API the developers can use to protect data, including key management and rotation.

With a great focus on security, .NET offers many different and reliable tools that can improve the security of the application and the data it contains. It has great starting points and support from the company itself, ranging from guides to an active community, so that developers can easily integrate into the workflow and deliver the best solution.

# Sources

|  |  |
| --- | --- |
| **Framework** | **Source** |
| JHipster | https://www.jhipster.tech/ |
| JHipster | https://github.com/jhipster/generator-jhipster/blob/main/CONTRIBUTING.md#question |
| Spring Boot | https://medium.com/better-programming/which-java-microservice-framework-should-you-choose-in-2020-4e306a478e58 |
| JHipster | https://www.jhipster.tech/creating-an-app/ |
| Microservices | https://techbeacon.com/app-dev-testing/challenges-scaling-microservices |
| Spring Boot security | https://snyk.io/blog/spring-boot-security-best-practices/ |
| Spring Boot | https://spring.io/blog/2020/06/10/the-path-towards-spring-boot-native-applications |
| Spring Boot | https://github.com/spring-projects/spring-boot |
| Quarkus | https://quarkus.io/ |
| Quarkus | https://github.com/quarkusio |