WILD CODE SCHOOL

Quarkus vs Spring Boot

When to use which Microservice Framework

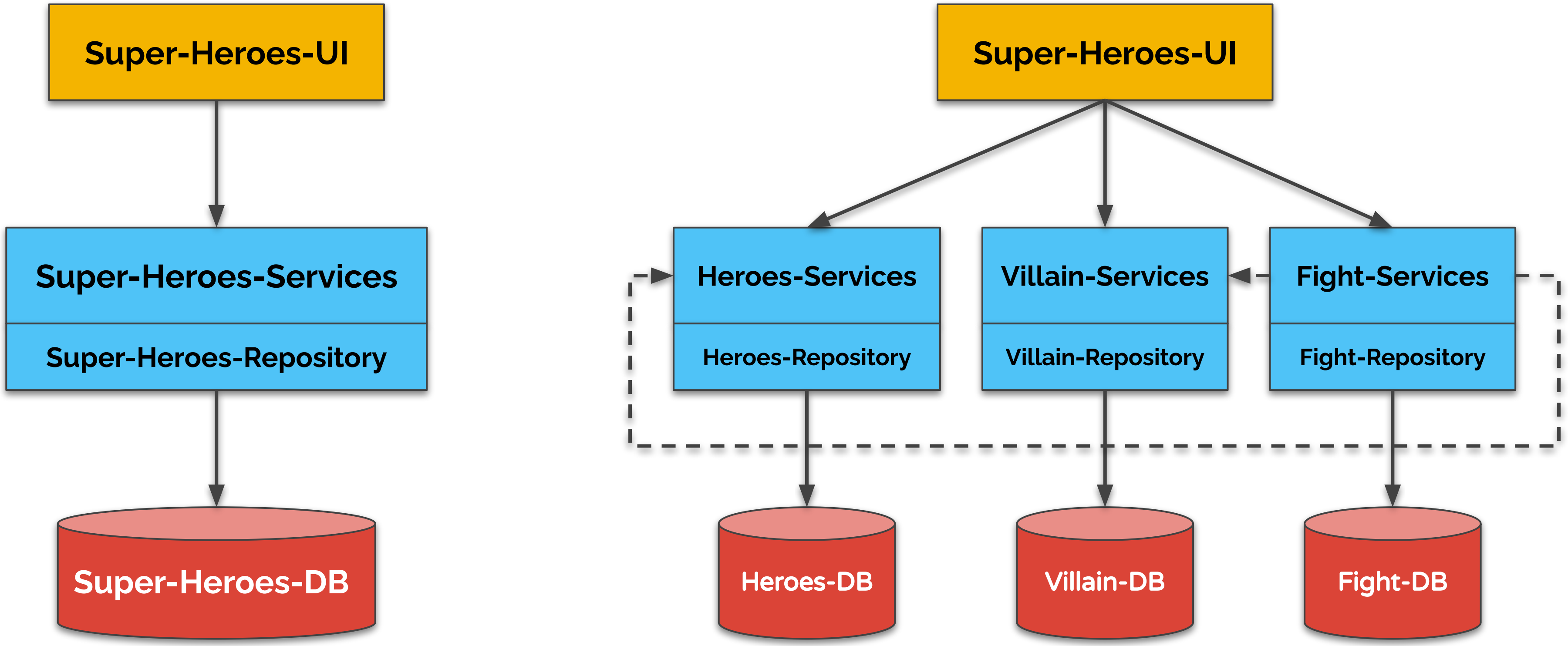
Won't be like this, promise!

https://blog.doubleslash.de/wp-content/uploads/2018/07/Nicht_mehr_zeitgem%C3%A4%C3%9Fe_Argumente_gegen%C3%BCber_JEE_und_Spring.jpg

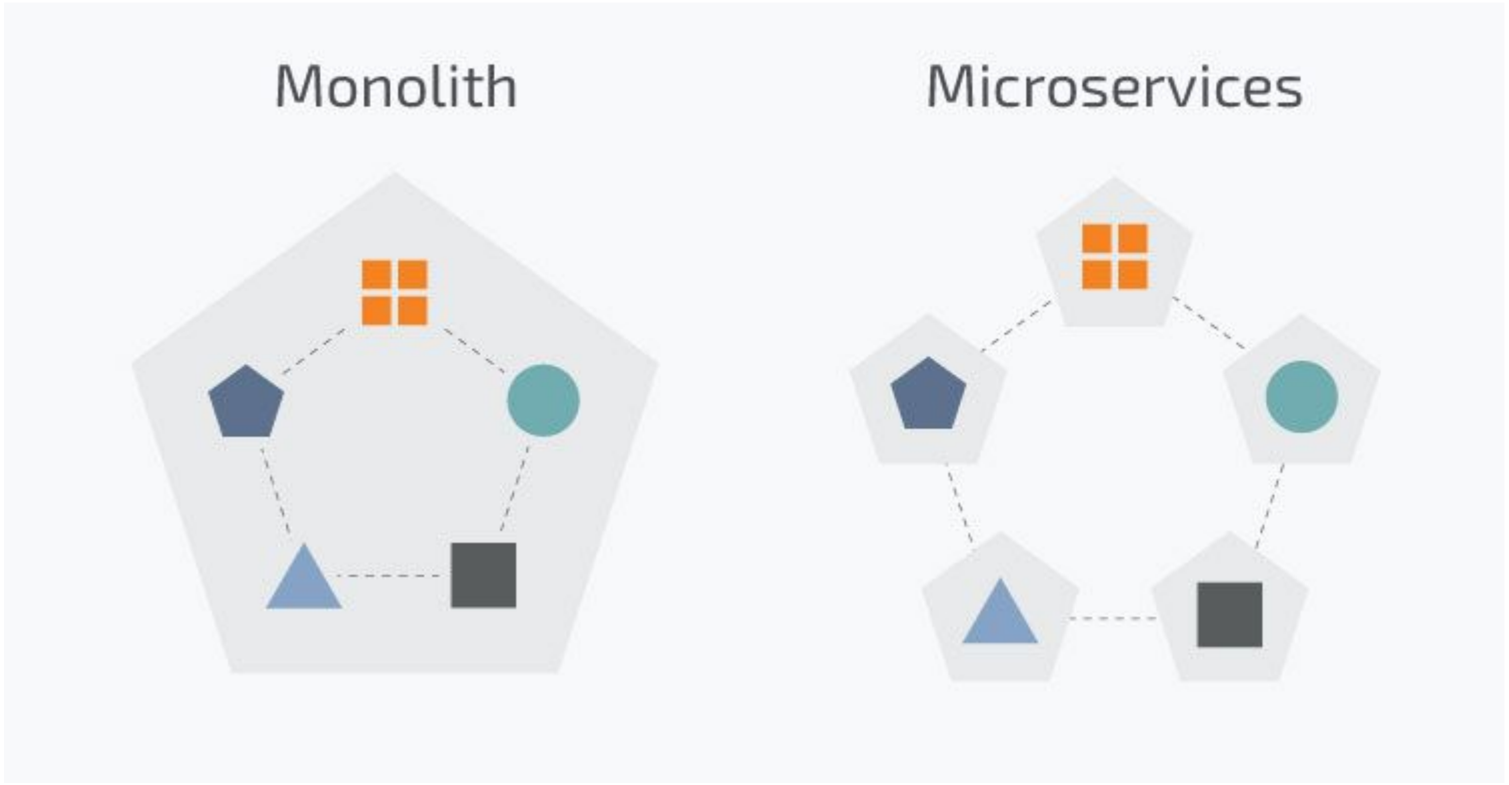
WILD
CODE
SCHOOL

- Why do we need Microservice Frameworks?
- A little History on Spring (Boot), J(2)EE and MicroProfile
- Demo: WeatherBot with Quarkus and Spring Boot
- How-to decide on (Microservice) Frameworks

Why do we need Microservice Frameworks?



What are Microservices?



https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

MONOLITHS

Strengths of the Monolithic Architecture

- **Less cross-cutting concerns**. Cross-cutting concerns are the concerns that affect the whole application such as **logging, handling, caching, and performance monitoring**. In a monolithic application, this area of functionality concerns only one application so it is easier to handle it.
- **Easier debugging and testing**. In contrast to the microservices architecture, monolithic applications are much **easier to debug and test**. Since a monolithic app is a single indivisible unit, you can **run end-to-end testing much faster.**
- **Simple to deploy**. Another advantage associated with the simplicity of monolithic apps is easier deployment. When it comes to monolithic applications, you do not have to handle many deployments – **just one file or directory**.
- **Simple to develop**. As long as the monolithic approach is a standard way of building applications, any engineering team has **the right knowledge and capabilities** to develop a monolithic application.

MONOLITHS

Weaknesses of the Monolithic Architecture

- **Understanding**. When a monolithic application scales up, it becomes **too complicated to understand**. Also, a complex system of code within one application is **hard to manage**.
- **Making changes**. It is **harder to implement changes** in such a large and complex application with highly tight coupling. Any code **change affects the whole system** so it has to be thoroughly coordinated. This makes the overall **development process much longer**.
- **Scalability**. You **cannot scale** components **independently**, only the whole application.
- **New technology barriers**. It is extremely problematic to apply a new technology in a monolithic application because then the **entire application has to be rewritten**.

Strengths of the Microservice Architecture

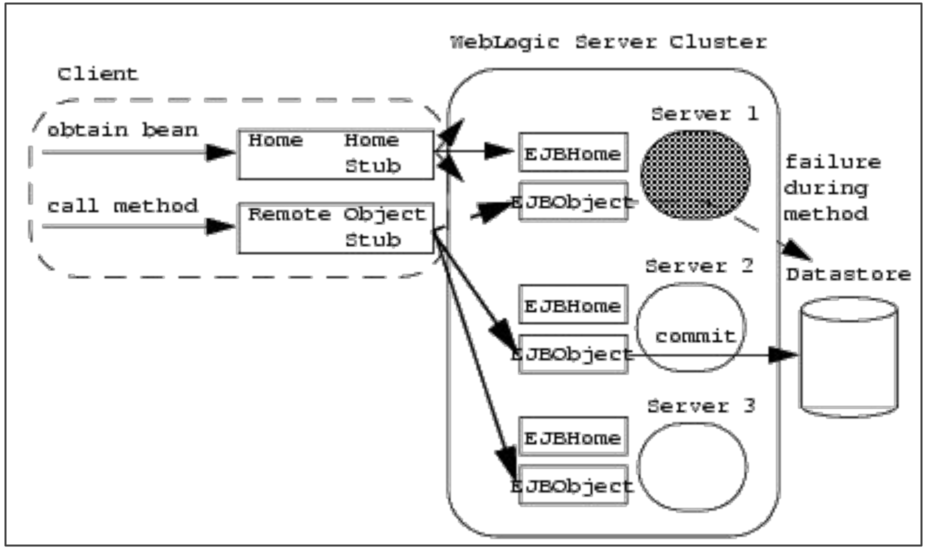
- **Independent components**. Firstly, all the services can be **deployed and updated independently**, which gives more flexibility. Secondly, a bug in one microservice has an impact only on a particular service and **does not influence the entire application**. Also, it is much **easier to add new features** to a microservice application than a monolithic one.
- **Easier understanding**. Split up into smaller and simpler components, a microservice application is **easier to understand and manage**. You just concentrate on a specific service that is related to a business goal you have.
- **Better scalability**. Another advantage of the microservices approach is that each element can be scaled independently. So the entire process is **more cost- and time-effective** than with monoliths when the whole application has to be scaled even if there is no need in it. In addition, every monolith has **limits in terms of scalability**, so the more users you acquire, the more problems you have with your monolith. Therefore, many companies, end up rebuilding their monolithic architectures.

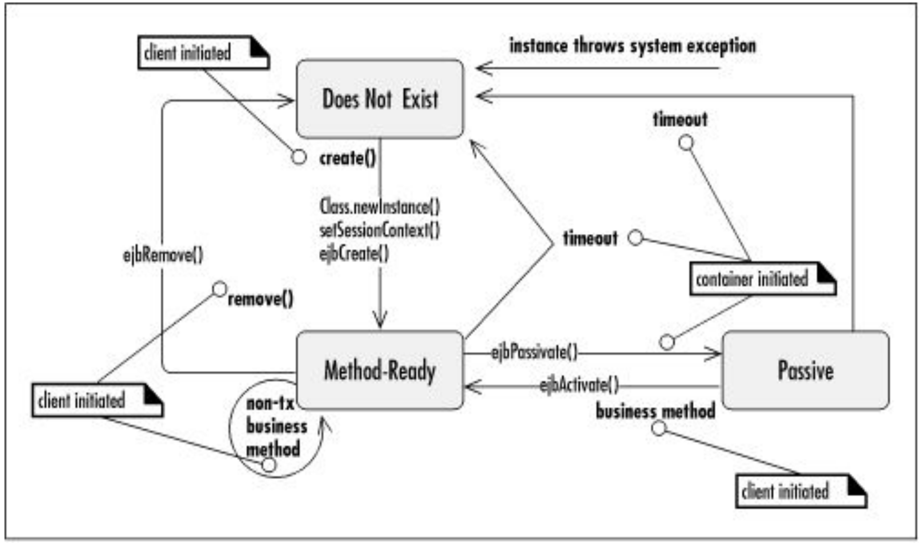
Weaknesses of the Microservice Architecture

- **Extra complexity**. Since a microservices architecture is a distributed system, you have to **choose and set up** the **connections between all the modules and databases**. Also, as long as such an application includes independent services, all of them have to be deployed independently.
- **System distribution**. A microservices architecture is a **complex system of multiple modules and databases** so all the connections have to be handled carefully.
- **Cross-cutting concerns**. When creating a microservices application, you will have to **deal with a number of cross-cutting concerns**. They include externalized configuration, logging, metrics, health checks, and others.
- **Testing**. A multitude of independently deployable components makes **testing** a microservices-based solution **much harder**.

https://ordina-jworks.github.io/conference/2018/06/26/Spring-IO-2018.html

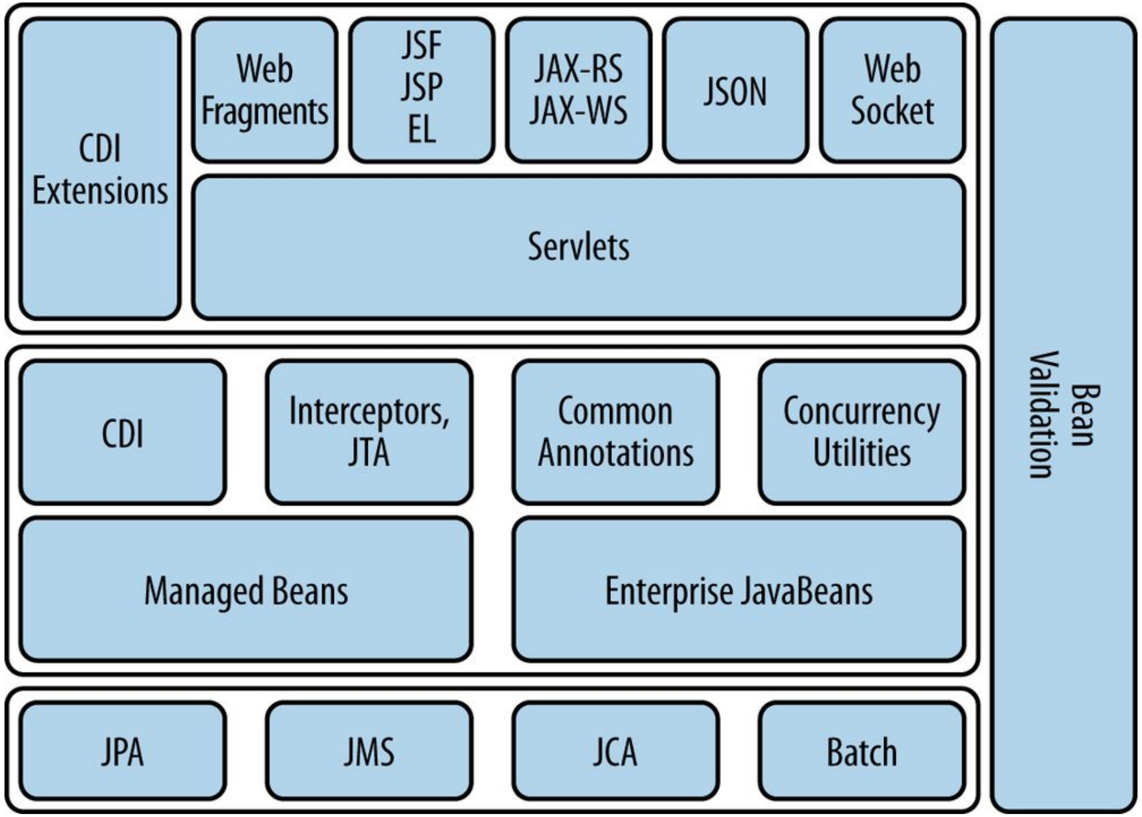
History of J(2)EE, Spring (Boot) and MicroProfile





http://www.mastertheboss.com/soa-cloud/quarkus/quarkus-vs-spring-boot-part-2/

Jakarta EE Technology Stack



taken from Java EE 7 Essentials

What is the “Spring Stack”?

Spring Framework, Spring Boot & Spring Cloud

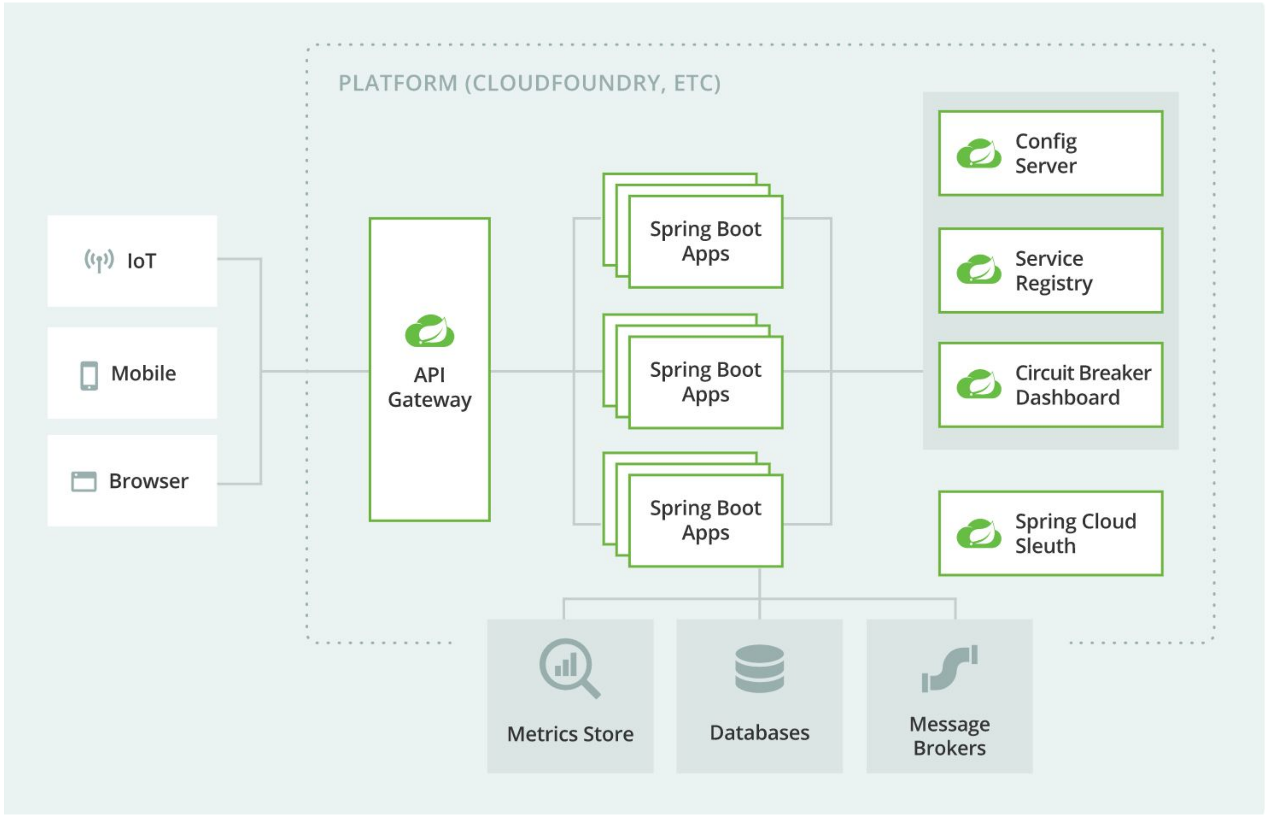
Spring Boot builds upon the **Spring Framework**, which is a popular, **open source, enterprise-level framework** for creating **standalone, production-grade applications for the JVM.**

It eases developing **web application and microservices** in Java, Kotlin, Scala or other JVM languages through **autoconfiguration, convention-over-configuration and cloud-readiness** by creating standalone applications with embedded runtime environments.

Spring Boot is the **most established and tested** Microservice framework in Java with the **biggest community** to support you when you are stuck, apart from **up-to-date and in-depth documentation**, there are also a lot of **resources available online** when it comes to learning Spring Boot.

Spring Cloud provides **many features out-of-the-box** which are needed in a Microservice architecture like **service discovery and load balancing**. It follows **Netflix Microservice architecture**, which is also the most common implementation.

Microservice Environment



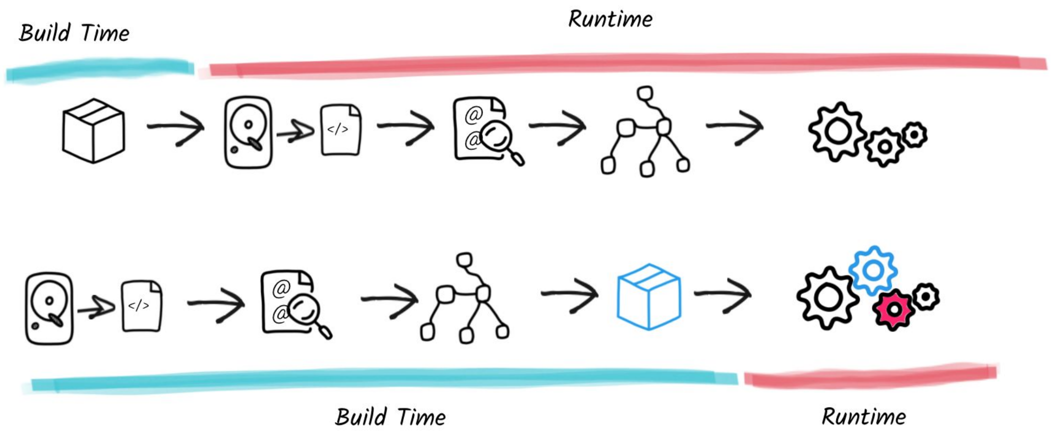
https://spring.io/microservices

What is Quarkus?

Quarkus

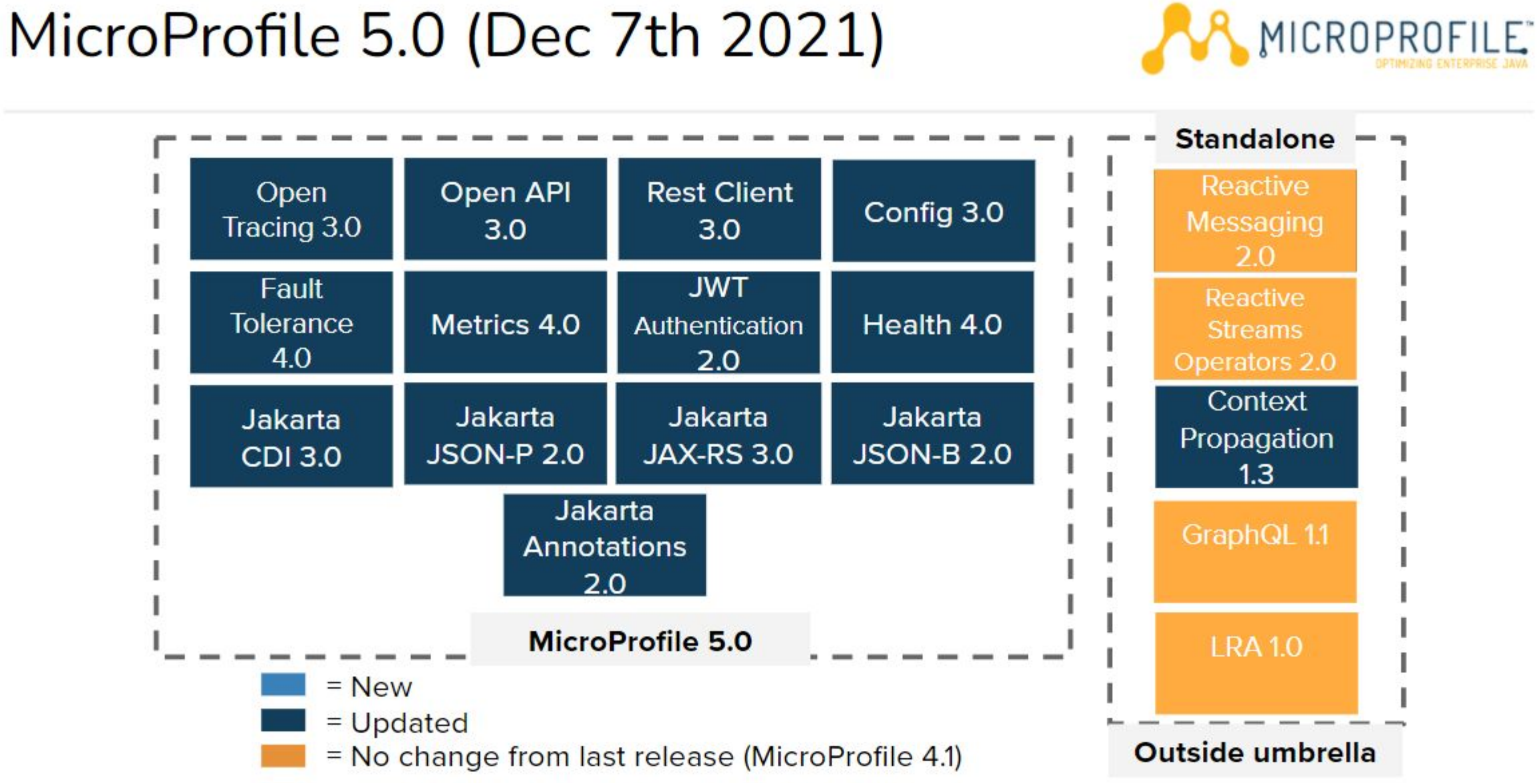
Quarkus is a **full-stack, Kubernetes-native Java framework** made for Java virtual machines (JVMs) and **native compilation**, **optimizing** Java specifically for **containers** and enabling it to become an effective platform for **serverless, cloud, and Kubernetes environments**.

Quarkus is backed by Redhat and it's quickly gaining ground for creating **high-performance, scalable Java applications**. One of the key features of Quarkus based applications is **fast boot time**.



https://quarkus.io/quarkus-workshops/super-heroes/#_welcome

MicroProfile Spec

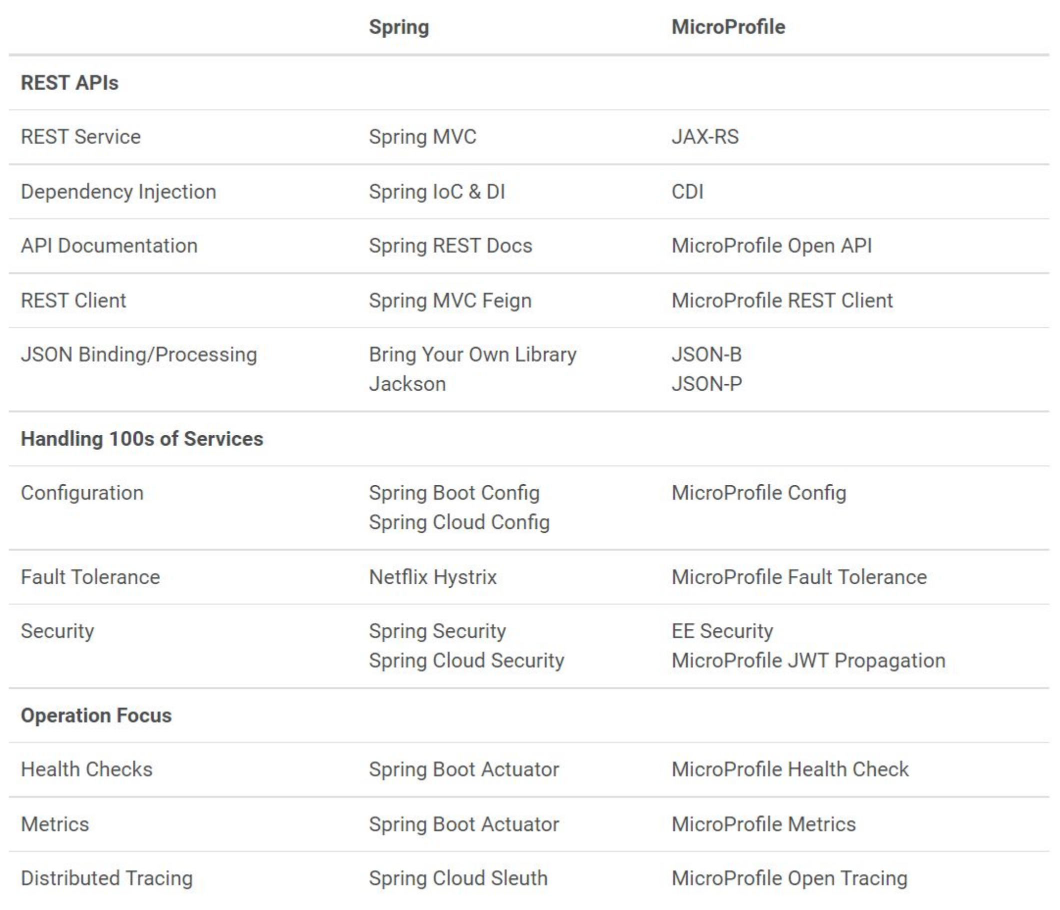


MicroProfile 5.0 (Dec 7th 2021)

🟦 = New
⬛ = Updated
🟧 = No change from last release (MicroProfile 4.1)

https://docs.google.com/presentation/d/1PfXEOJzy8v2kGRudvknPox5elRasewv3MBeV0P4Rk6M/edit#slide=id.ga6bdad56ea_0_7

Spring Submodules vs MicroProfile/JEE Specs



	Spring	MicroProfile
REST APIs		
REST Service	Spring MVC	JAX-RS
Dependency Injection	Spring IoC & DI	CDI
API Documentation	Spring REST Docs	MicroProfile Open API
REST Client	Spring MVC Feign	MicroProfile REST Client
JSON Binding/Processing	Bring Your Own Library Jackson	JSON-B JSON-P
Handling 100s of Services		
Configuration	Spring Boot Config Spring Cloud Config	MicroProfile Config
Fault Tolerance	Netflix Hystrix	MicroProfile Fault Tolerance
Security	Spring Security Spring Cloud Security	EE Security MicroProfile JWT Propagation
Operation Focus		
Health Checks	Spring Boot Actuator	MicroProfile Health Check
Metrics	Spring Boot Actuator	MicroProfile Metrics
Distributed Tracing	Spring Cloud Sleuth	MicroProfile Open Tracing

What is Cloud Native?

❖ **Externalise configuration:** Cloud native microservices must be configurable. When changing configuration, the microservcies should not be repackaged. **Build once, configure everywhere.** This is also highly recommended by the 12-factor methodology.

❖ **RESTful: Cloud Native microservices must be stateless.** The state should be stored in the database, not in the memory. The microservice itself should be treated **as cattle, not pet**. In this case, when a microservice container is not responding, the **underline cloud infrastructure can replace it without losing anything**.

❖ **Fault Tolerance:** Cloud Native microservices should be resilient. **It should function under whatever the situation is.** For example, even if its downstream service is out of service.

❖ **Discoverable:** Cloud Native microservices should be able to **advertise itself** for what function it provides or its capability so that **clients can invoke the services**.

❖ **Secure:** Security is extremely important in cloud native microservices. In a monolith application, only a small portion of the functions was exposed while **majority functions were hidden**. When moving to cloud native, every single service has a unique identity. You will need to **ensure that only authenticated and authorised users** can invoke the relevant microservices. For instance, if you deploy a microservice of salary query, you might only want a caller with the access right of "Manager" and "Owner" access this service.

https://www.eclipse.org/community/eclipse_newsletter/2019/september/microprofile.php

What is Cloud Native?

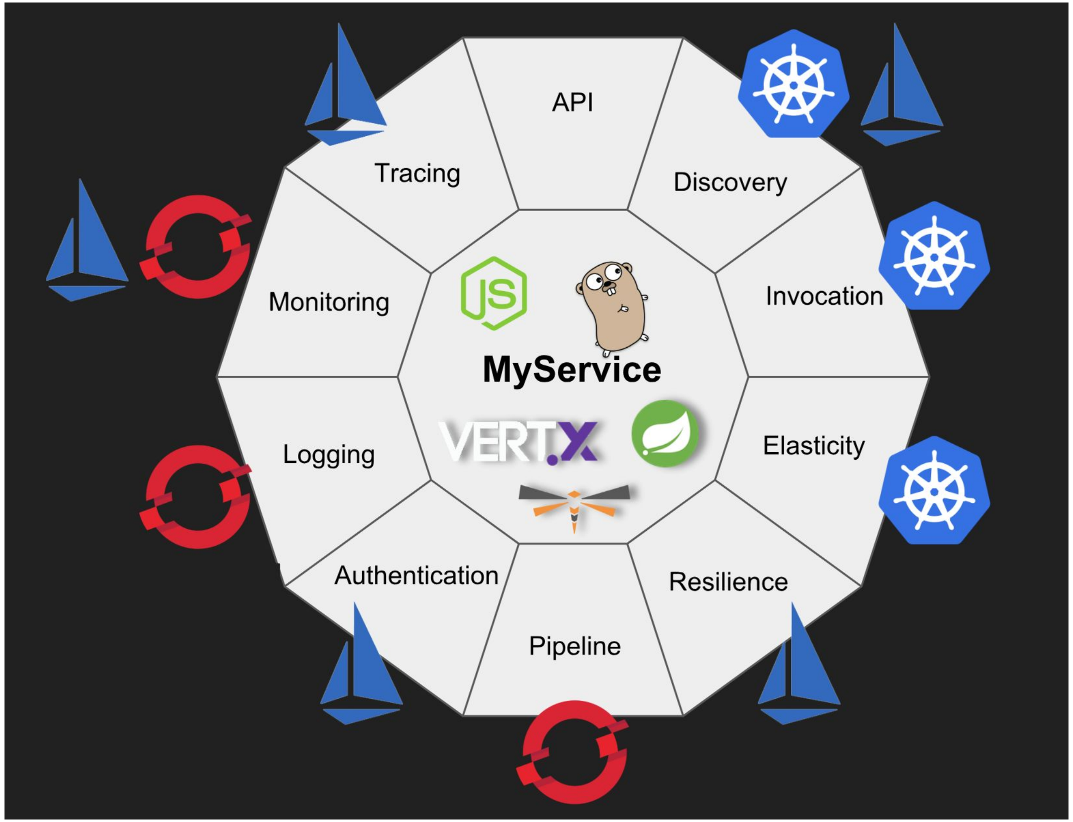
- **Monitorable:** Once a cloud native microservice is running in the cloud, it is essential to **emit some metrics** so that DevOps can **monitor it in order to find out how fast it responds** and how much loads it is taking, etc.

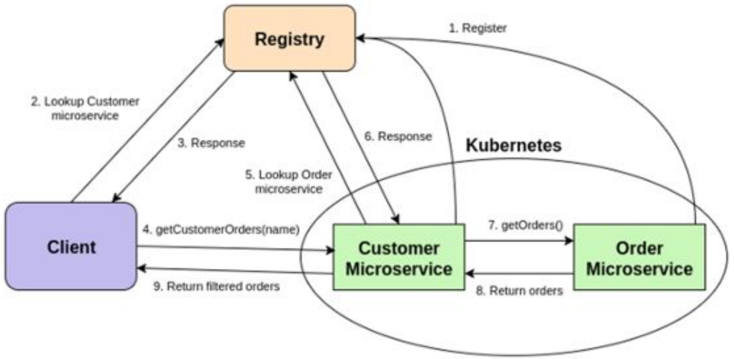
- **Traceable:** A cloud native microservice mostly needs to talk to other services, which then talks to different services. If you try to draw an invocation chain, you might end up a->b->d->f->e and so on. Once a service is not functioning, **you need to figure out which one is faulty**. Without the ability to be able to visualise the invocation chain, it will be very hard to identify the faulty service. In order to build the invocation chain, **microservices need to be able to propagate the correlation id**. With this, some tools such as ZipKin or Jaeger can build a trace span to illustrate each service status.

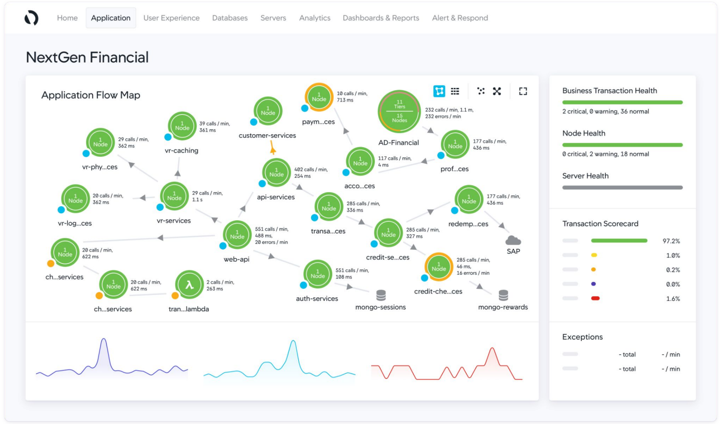
- **Ability to communicate with the Cloud:** A cloud native microservice needs to be able to **communicate with cloud infrastructure about its healthy status**. It should be able to tell the infrastructure whether it is ready to receive the request or whether it is still alive. If it is not alive, cloud infrastructure, e.g. Kubernetes, **can easily replace it**. If it is not ready, e.g. its database connection is not ready, **cloud infrastructure will not route requests to this service**.

https://www.eclipse.org/community/eclipse_newsletter/2019/september/microprofile.php

What is Cloud Native?

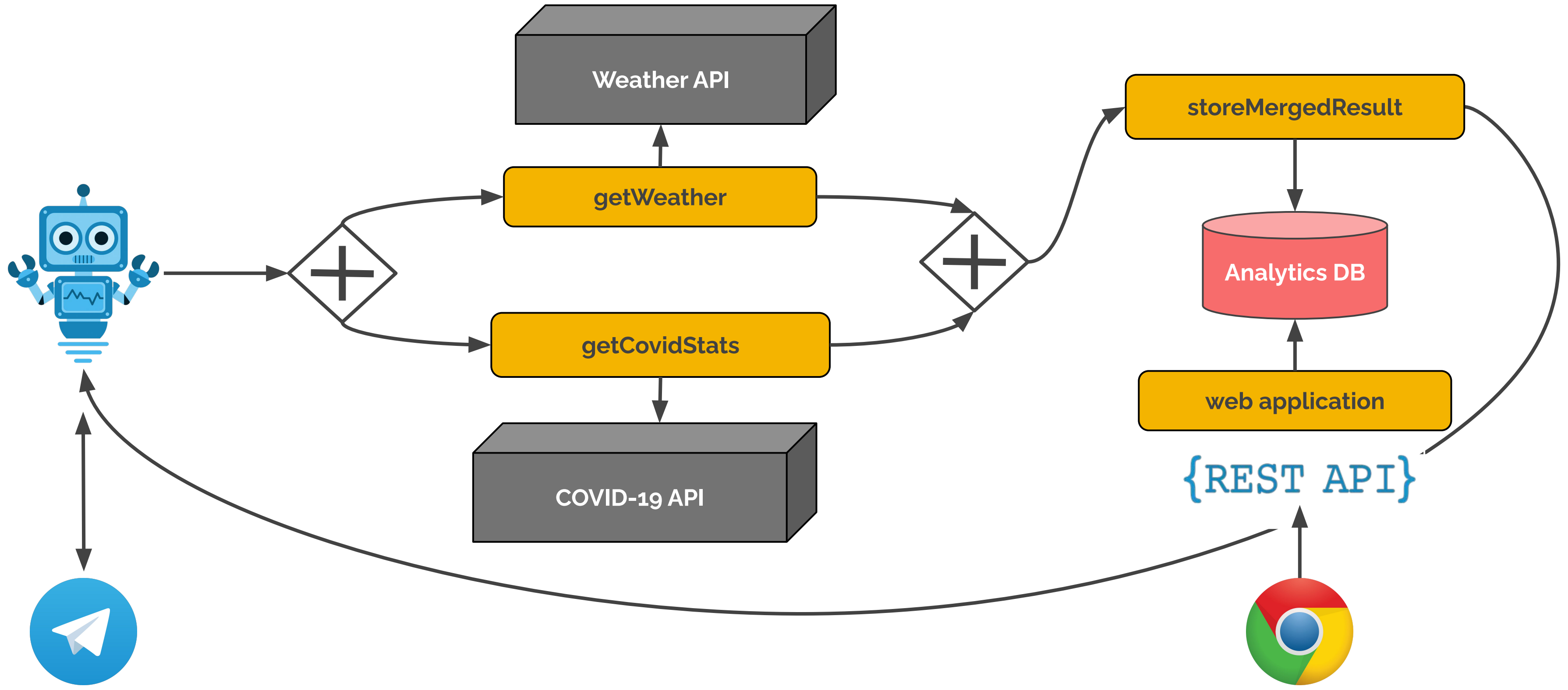






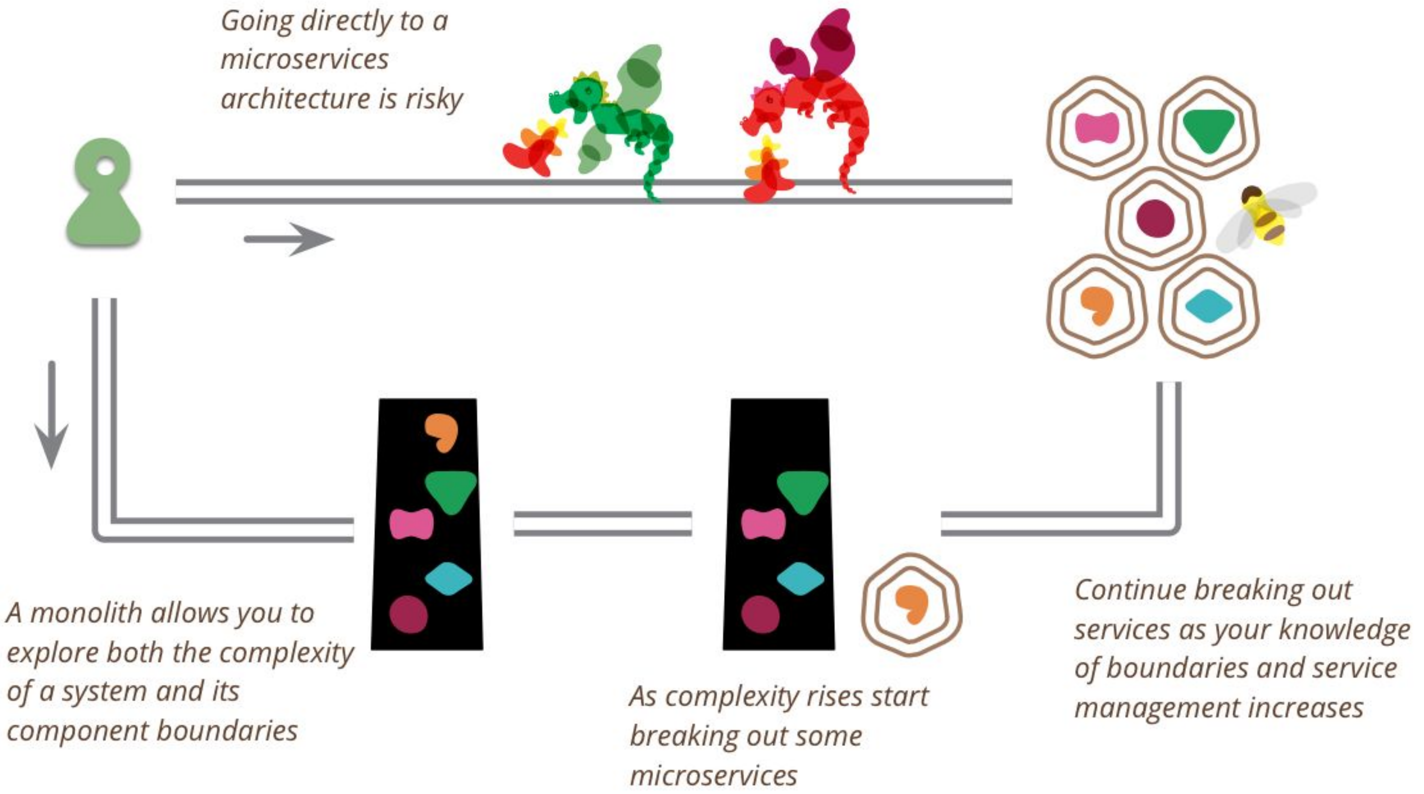
https://developers.redhat.com/blog/2018/06/28/why-kubernetes-is-the-new-application-server#are_application_servers_dead_
https://www.eclipse.org/community/eclipse_newsletter/2017/september/article5.php

Demo: Weather and COVID-19 Telegram Bot



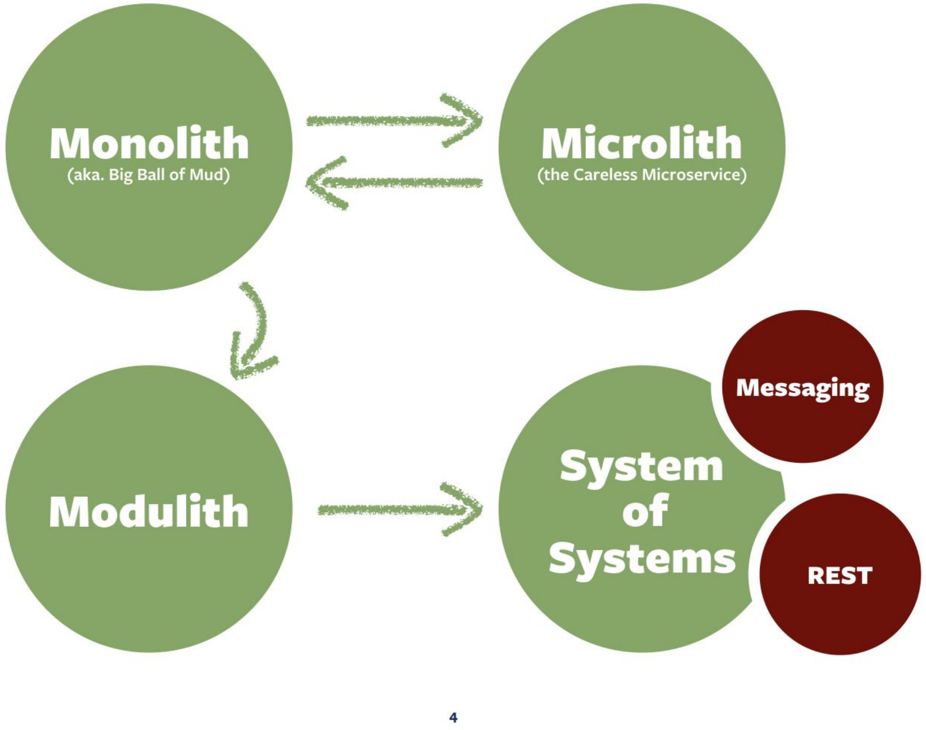
https://technology.amis.nl/software-development/java/apache-camel-and-spring-boot-calling-multiple-services-in-parallel-and-merging-results/

How to choose the best Architecture?



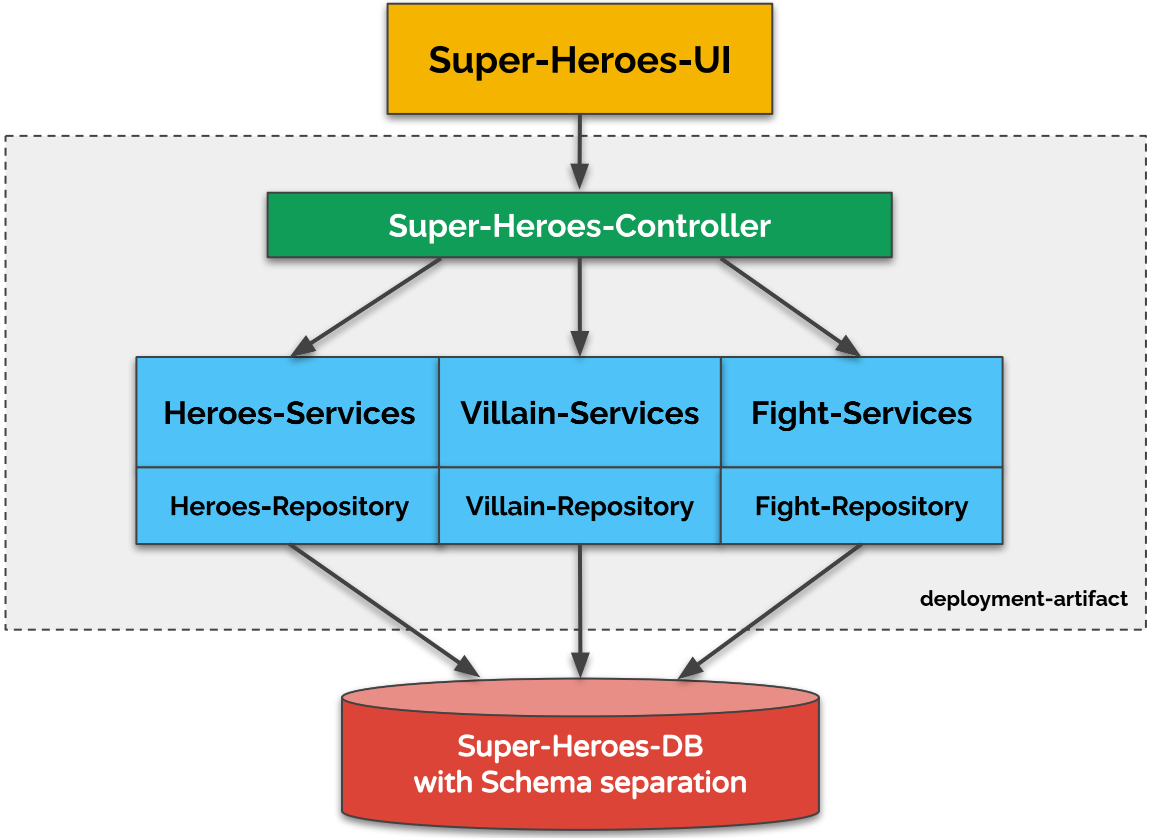
Going directly to a microservices architecture is risky
A monolith allows you to explore both the complexity of a system and its component boundaries
As complexity rises start breaking out some microservices
Continue breaking out services as your knowledge of boundaries and service management increases

Why do we need Microservices and Cloud Native?



https://ordina-jworks.github.io/conference/2018/06/26/Spring-IO-2018.html

The Modulith: Clean, Modular Architecture



Comparison between Quarkus and Spring Boot

Basis	Quarkus	Spring Boot
Microservices	Embraces the **MicroProfile API**, which is driven by an highly active and responsive community - **more innovative** and **purely developer-driven.**	Provides its **own modules** to develop modern Microservices architectures with the same goals.
Front-end development	At the moment includes basic built-in front-end options (Servlet) and the new **Qute template engine**.	Based on the **solid foundation of Spring MVC** and includes **mature templates (eg. Thymeleaf)**
Maturity	It is a **relatively new framework**, although derived from production ready **Java Enterprise API**	**Mature, open-source, feature-rich framework** with **excellent documentation and a huge community**
Data persistence	Uses frameworks familiars to developers (Hibernate ORM) which can be further be accelerated with **Panache and include reactive API. Focus on innovation.**	Based on **Spring Data abstraction. Focus on maturity**
Dependency Injection	Uses **CDI**. At the moment, only a **subset of the CDI features** is implemented	Uses its **robust Dependency injection Container.**
Speed	**Best overall application performance (JVM and Native).**	The **abstraction built on the top** of Spring makes it **generally slower** than projects derived from Java Enterprise API.

from: http://www.mastertheboss.com/soa-cloud/quarkus/quarkus-vs-spring-boot-part-2/

Links and other information

About Microservices

- ❖ Criteria for Usage: https://dwmkerr.com/the-death-of-microservice-madness-in-2018/

Comparison of Spring Boot and Quarkus

- ❖ Which Framework is Right for You: https://rollbar.com/blog/quarkus-vs-spring-boot/
- ❖ With Code and Benchmarks: http://code-addict.pl/quarkus-vs-spring/

Quarkus Persistence with Panache

- ❖ Panache Guide: https://quarkus.io/guides/hibernate-orm-panache
- ❖ Panache ORM Guide: https://fullstackcode.dev/2021/10/17/quarkus-simplified-hibernate-orm-with-panache/

Spring Data (JPA)

- ❖ Accessing Data with Spring JPA: https://spring.io/guides/gs/accessing-data-jpa/

Links and other information

Frameworks and Services used in Demo

- **REST CountryInformation**: https://restcountries.com/#api-endpoints-v3-code
- **REST COVID 19**: https://documenter.getpostman.com/view/10808728/SzS8rjbc#4b88f773-be9b-484f-b521-bb58dda0315c
- **Online JSON Formatter**: https://jsonformatter.curiousconcept.com/#
- **Jayway JSONPath**: https://www.baeldung.com/guide-to-jayway-jsonpath
- **Unirest HTTP Client**: https://www.baeldung.com/unirest

History

- **“War” between JEE and Spring:** https://www.coherentsolutions.com/blog/the-on-going-war-between-jee-and-spring-framework/
- **Current State of Jakarta EE:** https://jakarta.ee/release/9.1/