

# Quarkus

---

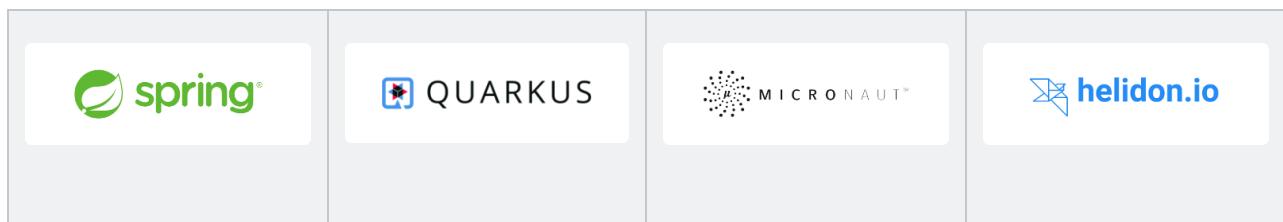
## Introduction

java application requires warm-up

Enemies of slow startup

- continuous delivery
- elasticity,scale on cloud:trends,people,reality

Microservices frameworks for java developers



Microservices are

Elastic

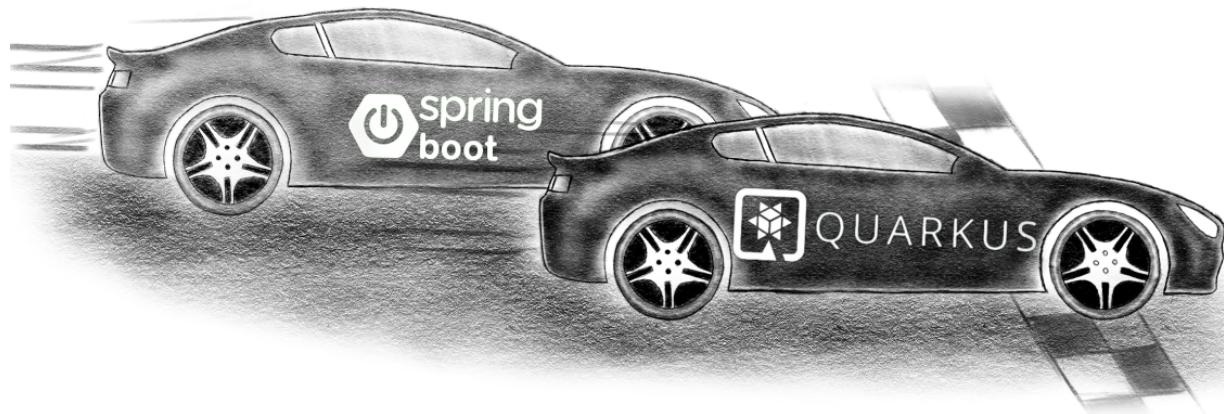
- Application start-up time
- Memory Usage

In terms of memory usage, the most interesting indicators are the **Resident Set Size (RSS)** and **Virtual memory Size**.

**RSS** is how much memory this process currently has in the main memory (RAM).

**VSZ** is how much virtual memory the process has in total. This includes all types of memory, both in RAM and swapped out.

---



---

## 1. Spring - Java Application

Startup Time	Memory Footprint
2.413s	real memory : 546.5 MB

---

What does a server / traditional java framework do at startup time? e.g spring framework

- Parse config files
- Classpath & classes scanning
  - For annotations, getters or other annotations

- Build framework metamodel objects
- Prepare reflection and build proxies
- **Start and open IO, threads etc**

How Quarkus work?

### **Move startup time activities to build time**

Build time benefits

- Do the work once, not at each start
- All the bootstrap classes are no longer loaded
- Less time to start, less memory used
- Less or no reflection nor dynamic proxy

## 2. Quarkus - JVM Application

<b>Startup Time</b>	<b>Memory Footprint</b>
0.862s	real memory : 322.4 MB

## 3. Quarkus - Native Application

<b>Startup Time</b>	<b>Memory Footprint</b>
0.019s	real memory : 19.6 MB

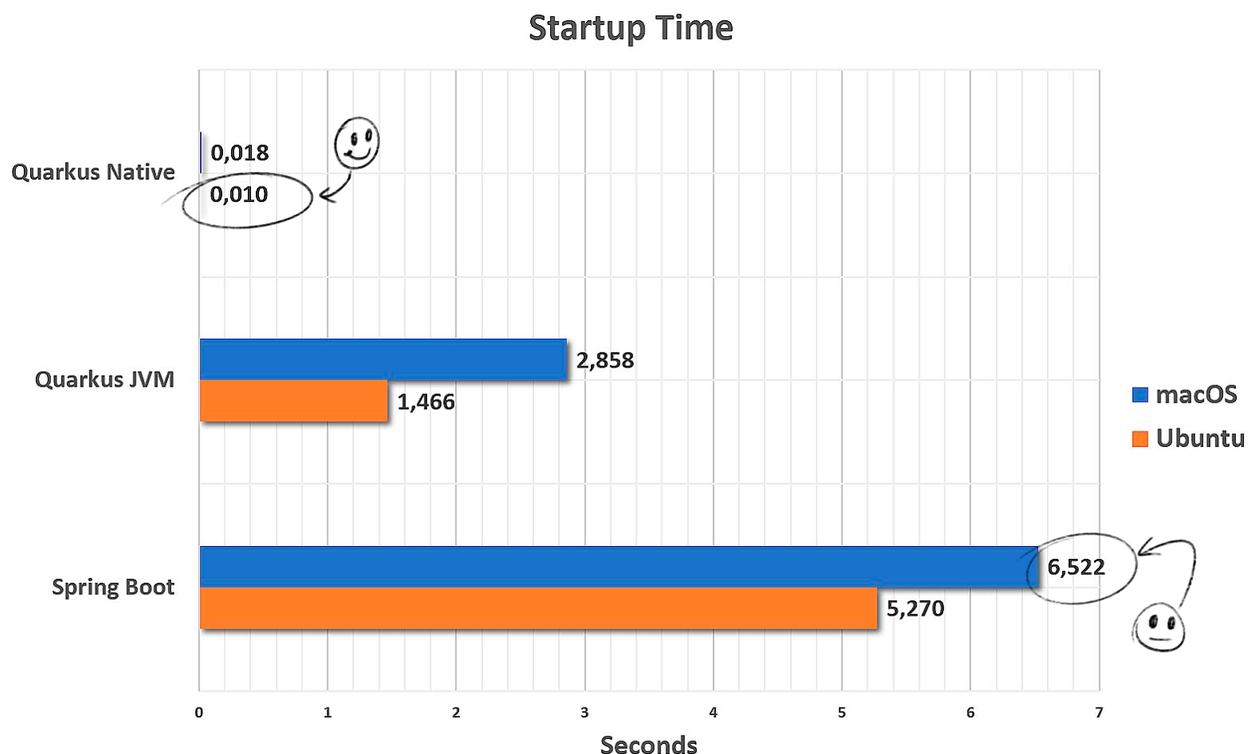
## 4. Quarkus - Java Application on docker container

<b>Startup Time</b>	<b>Memory Footprint</b>
1.865s	RSS - 93.86MiB

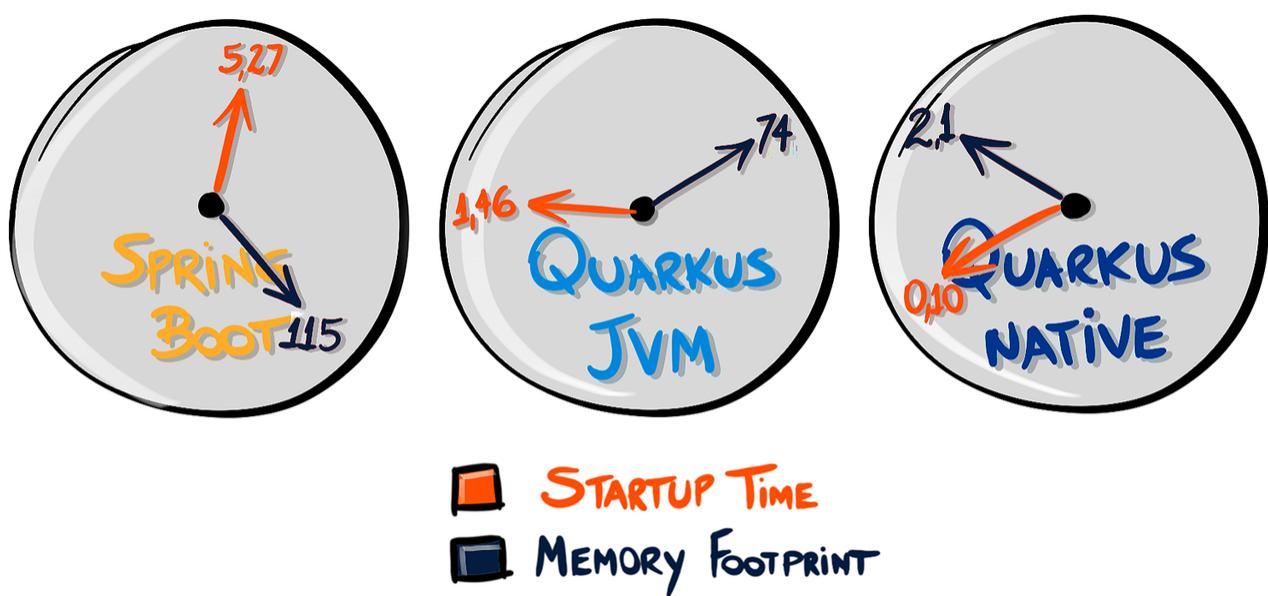
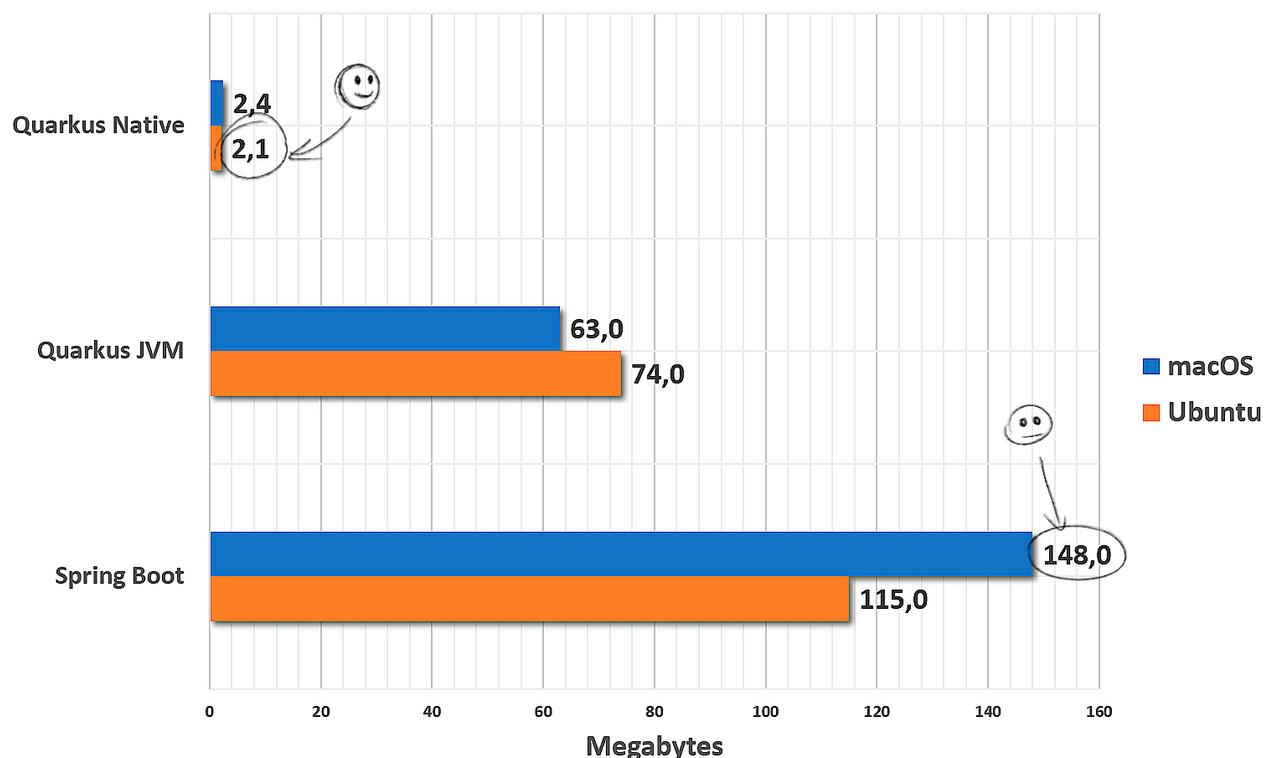
## 5. Quarkus - Native Application on docker container platform

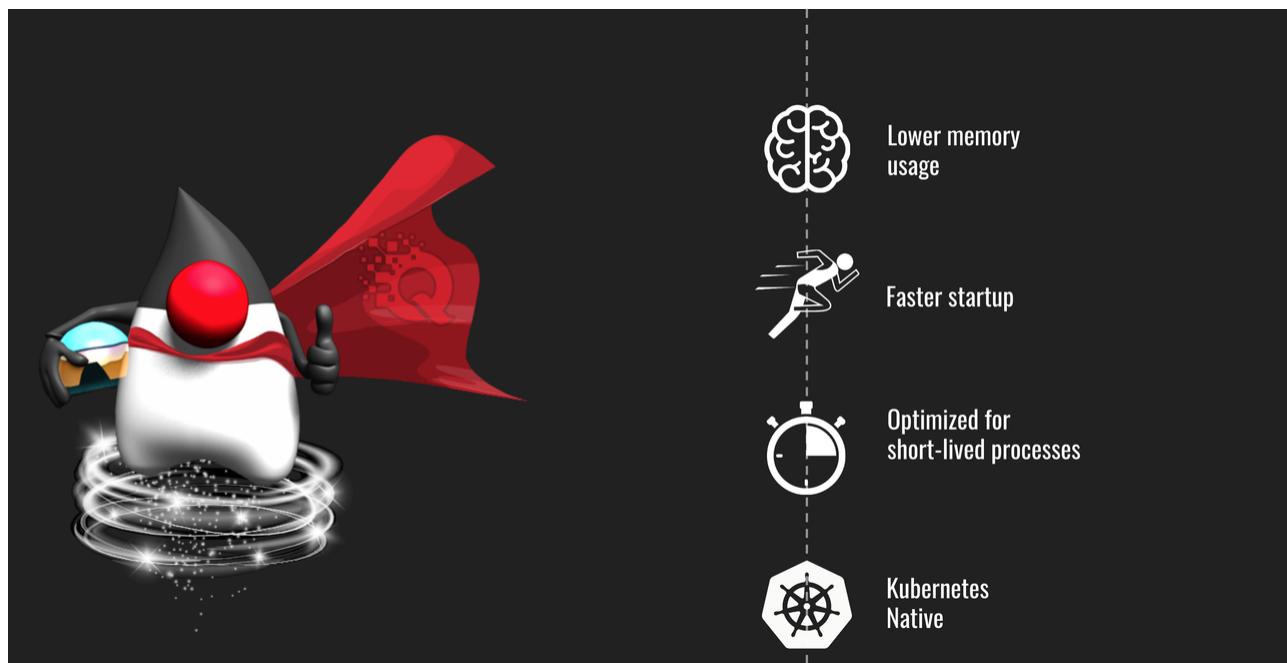
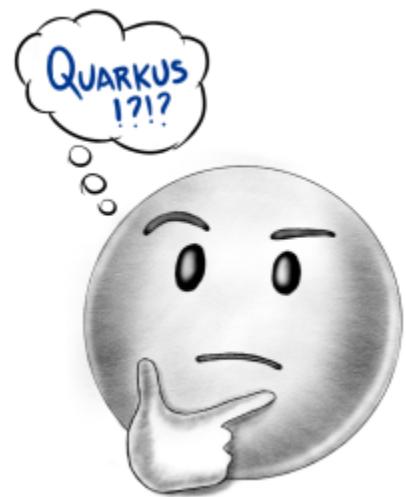
--	--

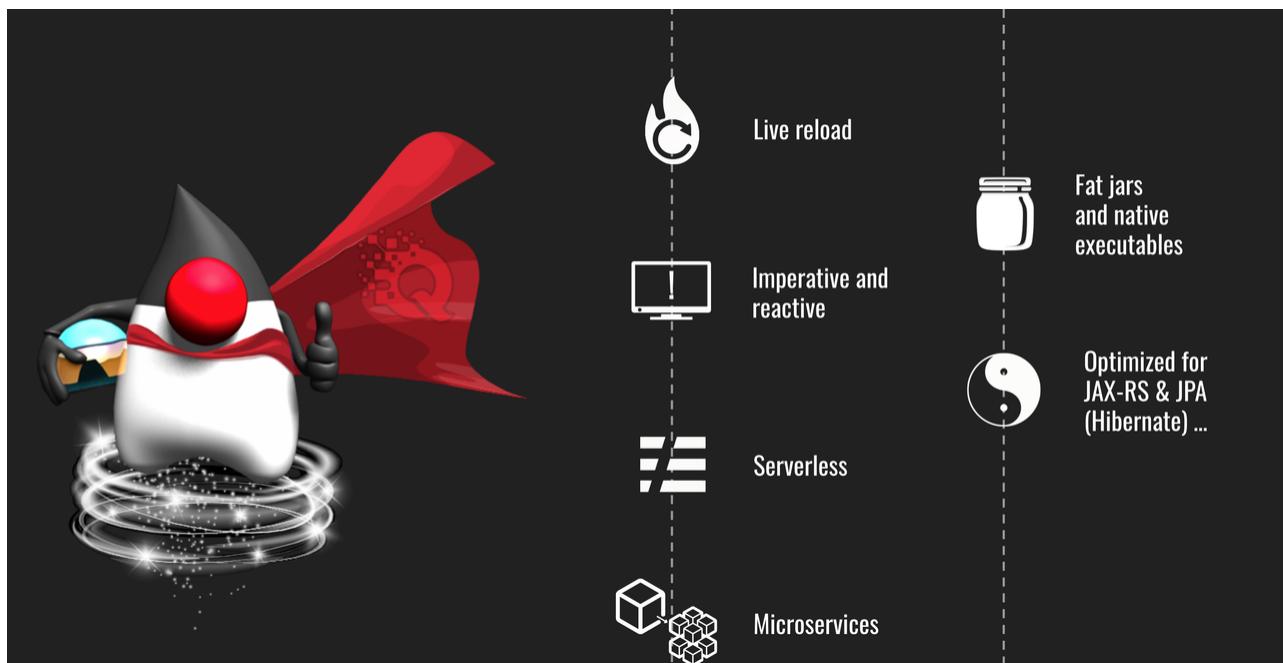
Startup Time	Memory Footprint
0.033s.	real memory : 4.2MiB



## Memory Footprint







---

## References

<https://www.graalvm.org/>

 GraalVM Community • [www.graalvm.org](http://www.graalvm.org)

<https://simply-how.com/quarkus-vs-spring-boot-production-performance>

<https://www.logicmonitor.com/blog/quarkus-vs-spring>

 Quarkus vs. Spring | LogicMonitor • www.logicmonitor.com

---

Quarkus is :

An Open Source  
stack to write Java apps



Cloud Native,

Microservices,

Serverless

---

## Quarkus core

---

1. Scaffolding a Quarkus Project with Maven

```
mvn io.quarkus:quarkus-maven-plugin:1.9.1.Final:create \
-DprojectGroupId=org.acme \
-DprojectArtifactId=getting-started \
-DclassName="org.acme.getting.started.GreetingResource" \
-Dpath="/hello"
cd getting-started
```

## 2. Scaffolding a Quarkus Project with Gradle

```
mvn io.quarkus:quarkus-maven-plugin:1.9.1.Final:create \
-DprojectGroupId=org.acme \
-DprojectArtifactId=getting-started \
-DclassName="org.acme.getting.started.GreetingResource" \
-Dpath="/hello"
-DbuildTool=gradle
cd getting-started
```

## 3. Scaffolding a Quarkus Project with the Quarkus Start Coding Website

<https://code.quarkus.io/>

## 4. Scaffolding a Quarkus Project with IDE

---

“Live Reloading with Dev Mode”

---

## Configuring quarkus with **Microprofile** config specification

- How to configure a Quarkus service
- How to inject configuration parameters in the service
- How to apply values depending on the environment
- How to correctly configure the logging system
- How to create customizations for the configuration system

1. Configuring the Application with Custom Properties
2. Configuring the Application with Custom Properties with YML

```
mvn quarkus:add-extension -Dextensions="config-yaml"
```

3. Accessing Configuration Properties Programmatically
4. Overwriting configuration values externally
  - a. environment variable ( 300 )
  - b. system property ( 400 )
5. configuring with profiles

Quarkus comes with three built-in profiles.

### **dev**

Activated when in development mode (i.e., quarkus:dev).

### **test**

Activated when running tests.

### **prod**

The default profile when not running in development or test mode; you don't need to set it in application.properties, as it is implicitly set

## 6. creating custom config sources

---

1. Changing Logger Configuration
2. Adding Application Logs
  - a. JDK java.util.Logging
  - b. Jboss Logging
  - c. SL4J
  - d. Apache common Logging
3. Advanced Logging for centralized Logging Tools i.e ELK tools

```
mvn quarkus:add-extension -Dextensions="logging-json"
```

```
mvn quarkus:add-extension -Dextensions="logging-gelf"
```

---

### Problem

You want to inject dependencies into your classes.

### Solution

Use Contexts and Dependency Injection (**CDI**)

---

## QUARKUS - CONTEXTS AND DEPENDENCY INJECTION

Quarkus DI solution is based on the [Contexts and Dependency Injection for Java 2.0](#) specification

<https://quarkus.io/guides/cdi>

 Quarkus - Introduction to Contexts and Dependency Injection • quarkus.io

- Injecting Dependencies
- Creating Factories
- Executing Object Life Cycle Events
- Executing Application Life Cycle Events
- Using a Named/Custom Qualifier
- Creating Interceptors

---

## Packaging Quarkus Applications

- How to package a Quarkus application for running in the JVM
- How to package a Quarkus application in a native executable
- How to containerize a Quarkus application

1. Running in Command Mode
2. Creating a Runnable JAR file

`./mvnw package`

3. Über-JAR Packaging

```
./mvnw package -Dquarkus.package.uber-jar=true
```

#### 4. "Building a Native Executable"

```
./mvnw package -Pnative
```

#### 5. Building a Docker Container for JAR File

```
mvn package
```

```
docker build -f src/main/docker/Dockerfile.jvm -t quarkus/quarkus-hello-service-jvm .
```

```
docker run -i --rm -p 8080:8080 quarkus/quarkus-hello-service-jvm
```

#### 6. Building a Docker Container for Native File

```
mvn package -Pnative -Dquarkus.native.container-build=true
```

```
docker build -f src/main/docker/Dockerfile.native -t nagabhushanam/quarkus-hello-service .
```

---

## Quarkus web

---

Quarkus integrates with RESTEasy, a JAX-RS implementation to define REST APIs.

- How to use JAX-RS for creating CRUD services

#### 1. Creating a Simple REST API Endpoint

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
```

```
public class GreetingResource {  
    @GET  
    @Produces(MediaType.TEXT_PLAIN)  
    public String hello() {  
        return "hello";  
    }  
    @POST  
    @Consumes(MediaType.TEXT_PLAIN)  
    public void create(String message) {  
        System.out.println("Create");  
    }  
  
    @PUT  
    @Consumes(MediaType.TEXT_PLAIN)  
    @Produces(MediaType.TEXT_PLAIN)  
    public String update(String message) {  
        System.out.println("Update");  
        return message;  
    }  
  
    @DELETE  
    public void delete() {  
        System.out.println("Delete");  
    }  
}
```

## 2. Extracting Request Parameters

- `@QueryParam` - to filter resources
- `@PathParam` - to identify resource
- `@HeaderParam`
- `@FormParam`
- `@CookieParam`
- `@MatrixParam`
- `@Context`

```
public static enum Order {
    desc, asc;
}

@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello(
    @Context UriInfo uriInfo,
    @QueryParam("order") Order order,
    @NotBlank @HeaderParam("authorization") String
authorization
) {
    //...
    return "...";
}
```

### 3. Using Semantic HTTP Response Status Codes

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

## 4. Binding HTTP Methods

## 5. Enabling Cross-Origin Resource Sharing (CORS)

```
quarkus.http.cors=true  
quarkus.http.cors.origins=http://example.com  
quarkus.http.cors.methods=GET,PUT,POST,DELETE  
quarkus.http.cors.headers=accept,authorization,content-type,x-requested-with
```

## 7. Marshalling/Unmarshalling JSON

<https://quarkus.io/guides/rest-json>

 Quarkus - Writing JSON REST Services • quarkus.io

## 8. Marshalling/Unmarshalling XML

## 9. Validating Input and Output Values

<https://quarkus.io/guides/validation>

 Quarkus - Validation with Hibernate Validator • quarkus.io

- 
- Configure datasources
  - Deal with transactions
  - Make use of the Panache API
  - Interact with NoSQL data stores

```
mvn quarkus:add-extension -Dextensions="jdbc-xxxx"
```

## 1. Defining a Datasource

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=<your username>
quarkus.datasource.password=<your password>

quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/hibernate_orm_test
quarkus.datasource.jdbc.max-size=16
```

## 2. Using Multiple Datasources

```
quarkus.datasource.driver=org.h2.Driver
quarkus.datasource.url=jdbc:h2:tcp://localhost/mem:default
quarkus.datasource.username=username-default
quarkus.datasource.min-size=3
quarkus.datasource.max-size=13

quarkus.datasource.users.driver=org.h2.Driver
quarkus.datasource.users.url=jdbc:h2:tcp://localhost/mem:users
quarkus.datasource.users.username=username1
```

```
quarkus.datasource.users.min-size=1  
quarkus.datasource.users.max-size=11  
  
quarkus.datasource.inventory.driver=org.h2.Driver  
quarkus.datasource.inventory.url=jdbc:h2:tcp://localhost/mem:inventory  
quarkus.datasource.inventory.username=username2  
quarkus.datasource.inventory.min-size=2  
quarkus.datasource.inventory.max-size=12
```

### 3. Adding Datasource Health Check

```
mvn quarkus:add-extension -Dextensions="quarkus-smallrye-health"
```

### 4. Defining Transaction Boundaries Declaratively

```
@javax.transaction.Transactional
```

### 5. Setting a Transaction Context

`@Transactional(REQUIRED) (default)`

Starts a transaction if none was started; otherwise, stays with the existing one

`@Transactional(REQUIRES_NEW)`

Starts a transaction if none was started; if an existing one was started, suspends it and starts a new one for the boundary of that method

`@Transactional(MANDATORY)`

Fails if no transaction was started; otherwise, works within the existing transaction

@Transactional(SUPPORTS)

If a transaction was started, joins it; otherwise, works with no transaction

@Transactional(NOT\_SUPPORTED)

If a transaction was started, suspends it and works with no transaction for the boundary of the method; otherwise, works with no transaction

@Transactional(NEVER)

If a transaction was started, raises an exception; otherwise, works with no transaction

## 6. Programmatic Transaction Control

```
@Inject  
UserTransaction userTransaction  
  
begin()  
commit()  
rollback()  
setRollbackOnly()  
getStatus()  
setTransactionTimeout(int)
```

## 7. Setting and Modifying a Transaction Timeout

```
@io.quarkus.narayana.jta.runtime.TransactionConfiguration
```

## 8. Persisting Data with Panache

<https://quarkus.io/guides/hibernate-orm-panache>

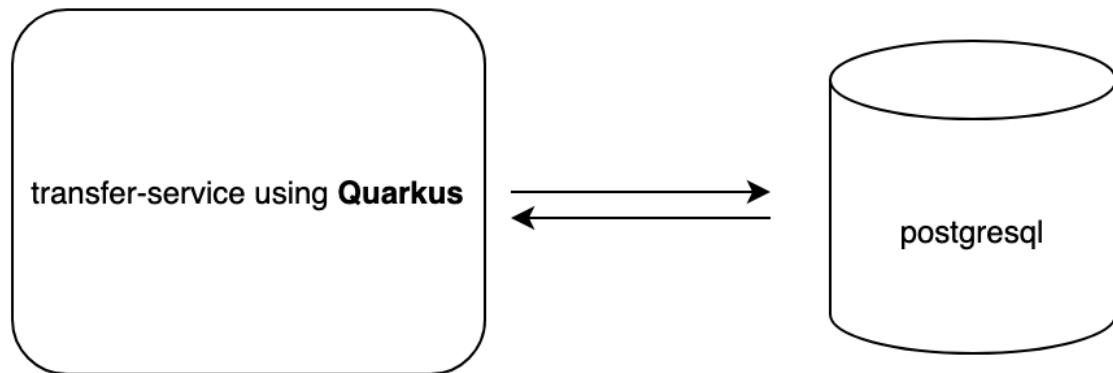
 Quarkus - Simplified Hibernate ORM with Panache • quarkus.io

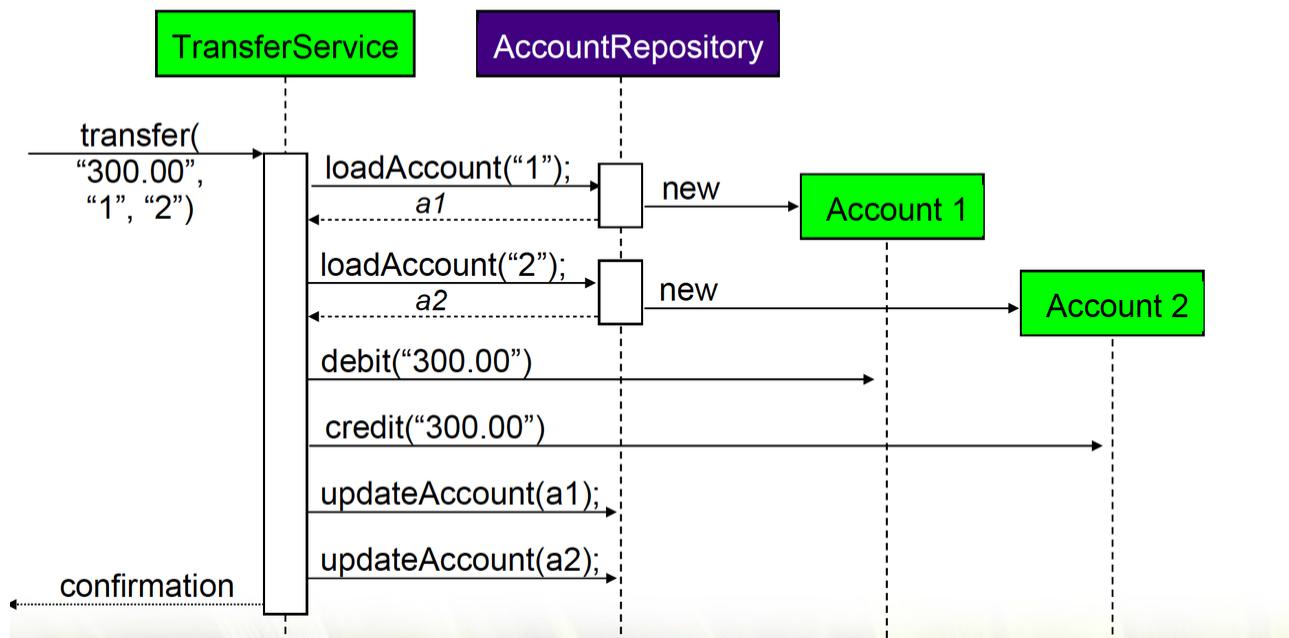
---

### Exercise

---

Ex. transfer-service api

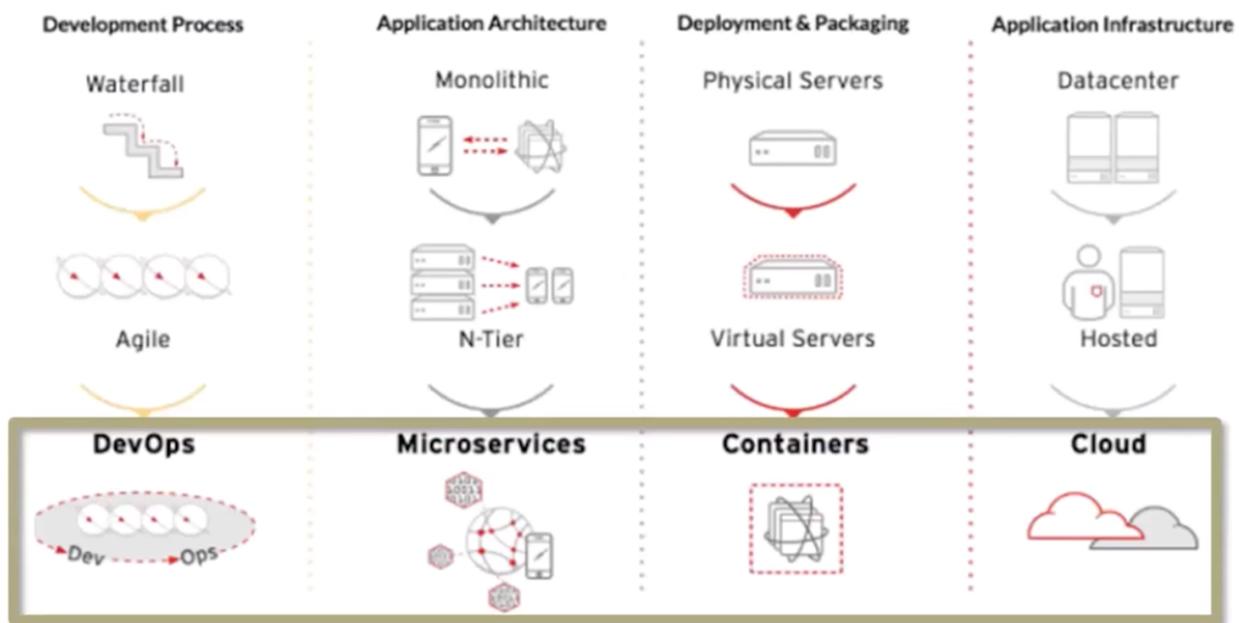




## Microservices

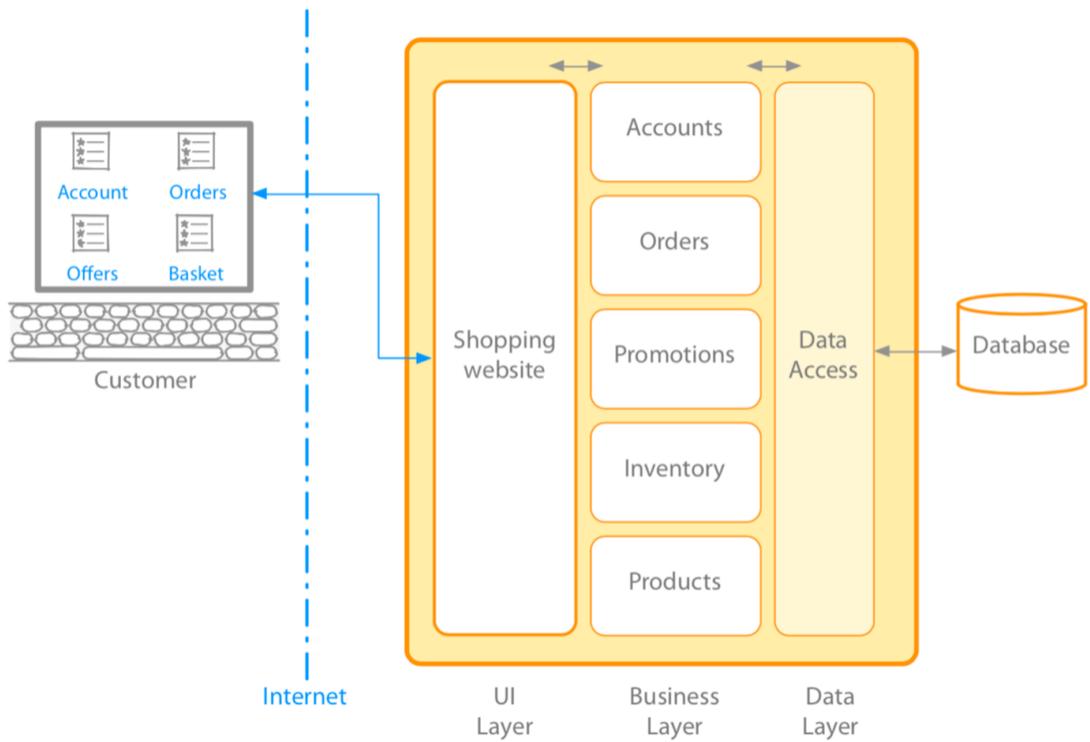
---

Software evolution



Monolithic architecture

Ex.



### Monolithic architecture - pros

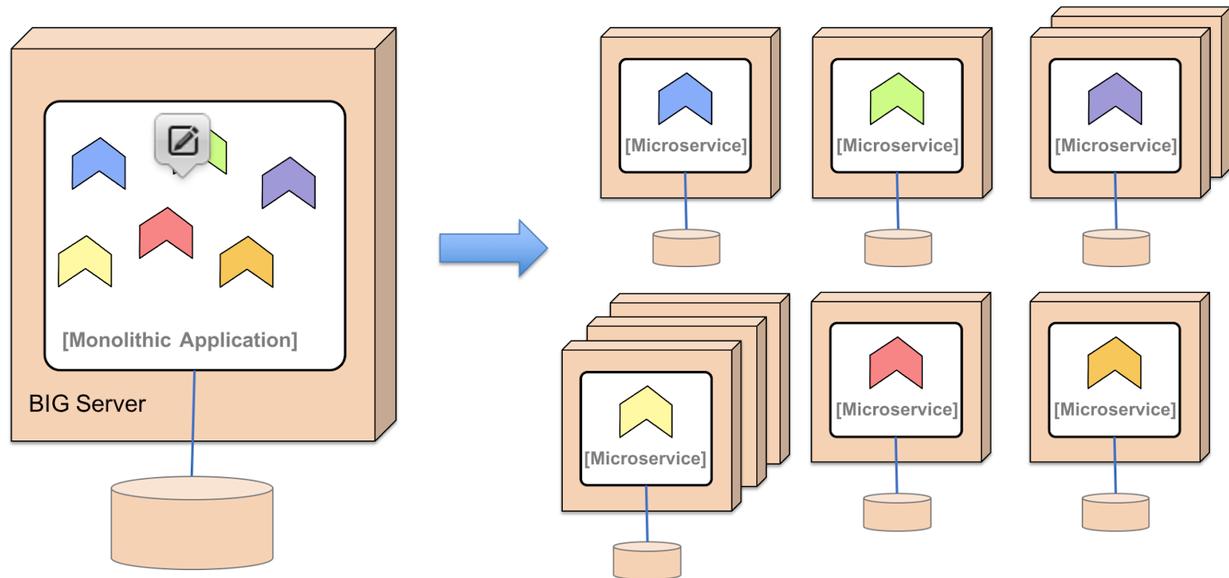
- Simpler development & deployment
- Fewer cross-cutting concerns
- Better performance

### Monolithic architecture - cons

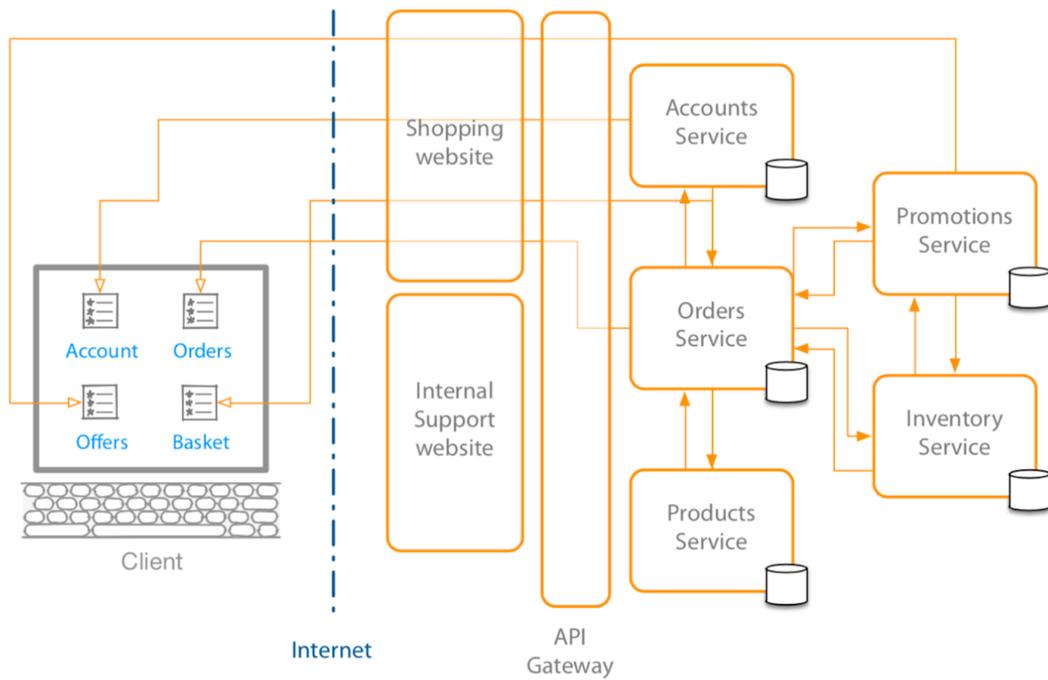
- Larger codebase
- Difficult to adopt new technologies
- Failure could affect whole system
- Scaling requires duplication of the whole
- Limited agility
  - every small update requires a full redeployment
  - all developers have to wait until it's done

- When several teams are working on the same project, agility can be reduced greatly.

## Enter Microservices



Ex.



## Microservice - characteristics

- Must conform to a shared-nothing architecture
- Must only communicate with well defined interfaces ( REST API )
- Must be deployed as separate runtime process e.g Docker
- Microservice instances are stateless

## Microservice - benefits

- Easy to develop,test and deploy
- Increased agility
- Ability to scale horizontally

## Challenges with microservices

- Synchronous communication
- Keeping the configuration up to date
- It's hard to track a request
- Analyzing the usage of hardware resources on a component level
- Management of many small components can become costly and error-prone

## The 8 fallacies of distributed computing

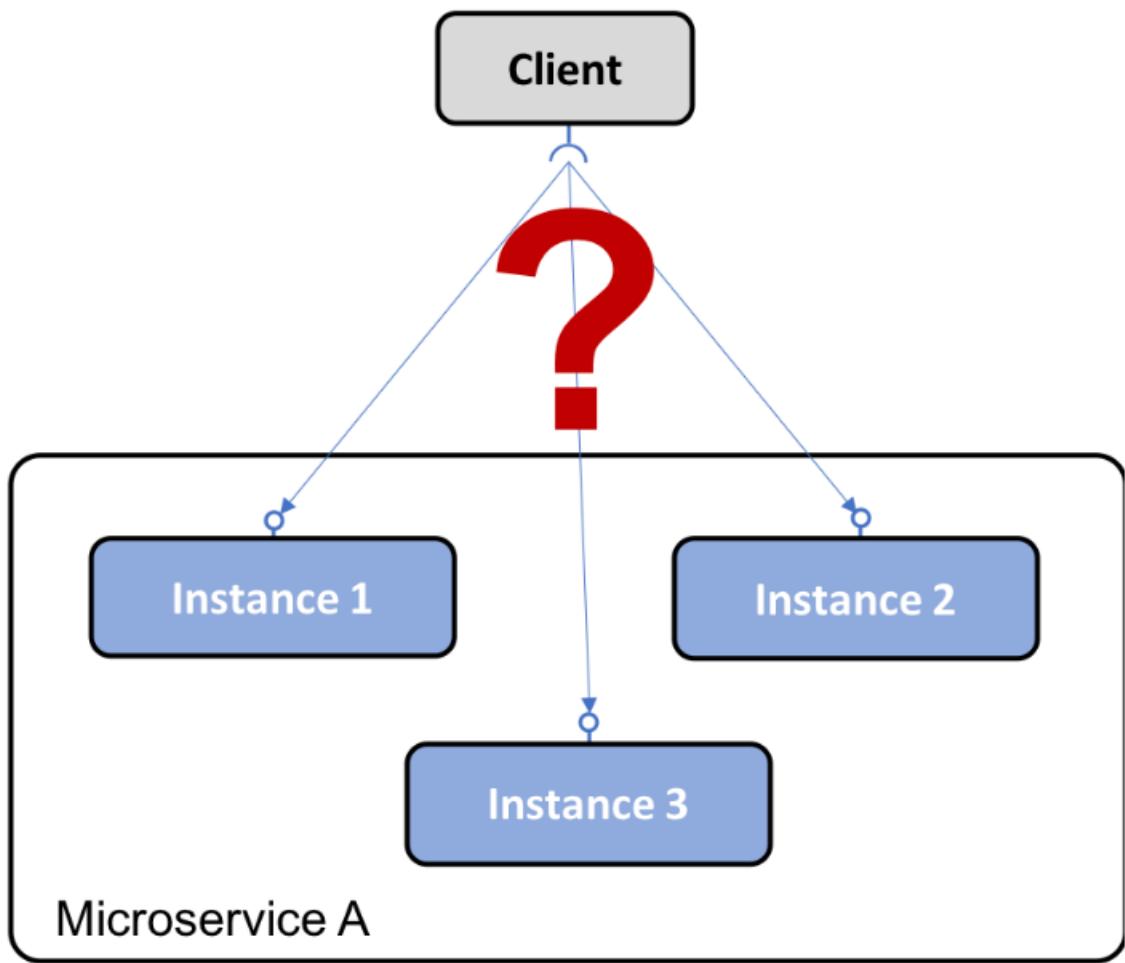
1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

<https://www.rgoarchitects.com/Files/fallacies.pdf>

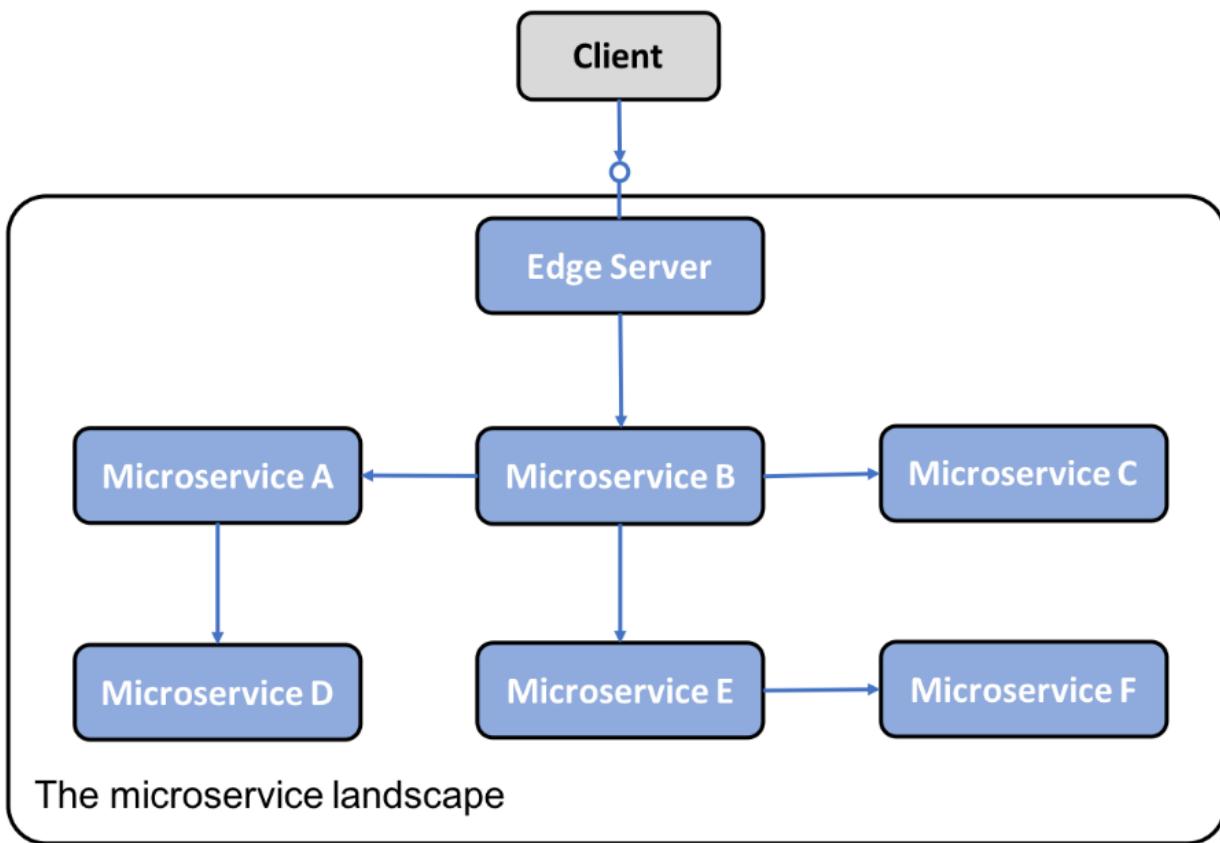
## Design patterns for microservices

- ▶ Service discovery
- ▶ Edge server
- ▶ Reactive microservices
- ▶ Central configuration
- ▶ Centralized log analysis
- ▶ Distributed tracing
- ▶ Circuit Breaker
- ▶ Control loop
- ▶ Centralized monitoring and alarms

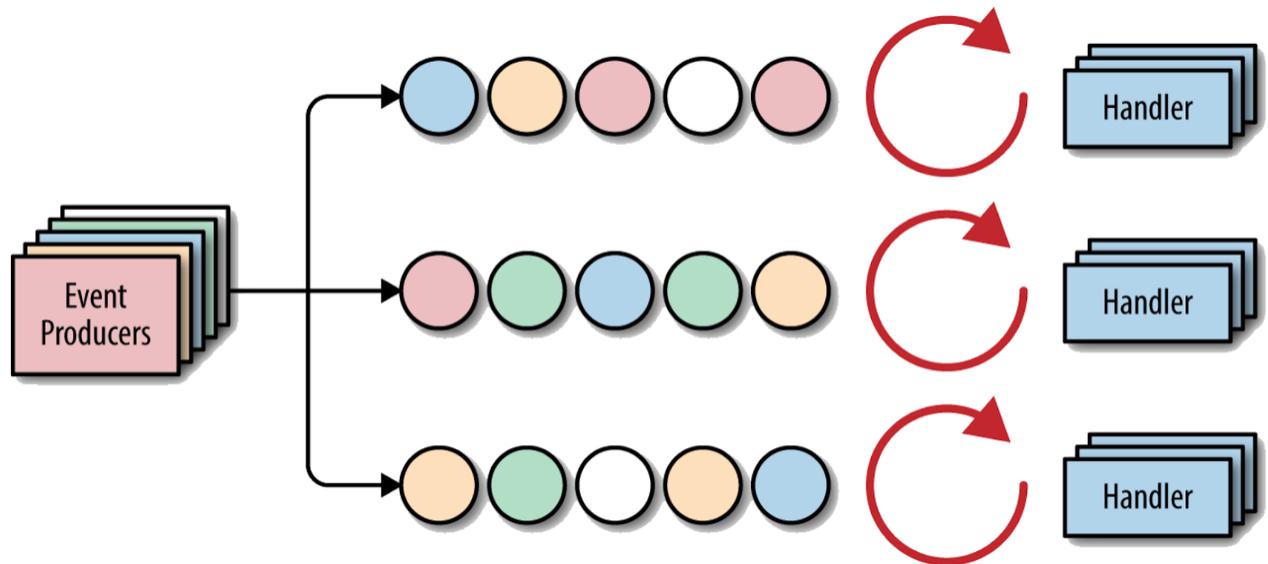
## 1. Service Discovery



2. Edge ( Gateway/Load Balancer) Server



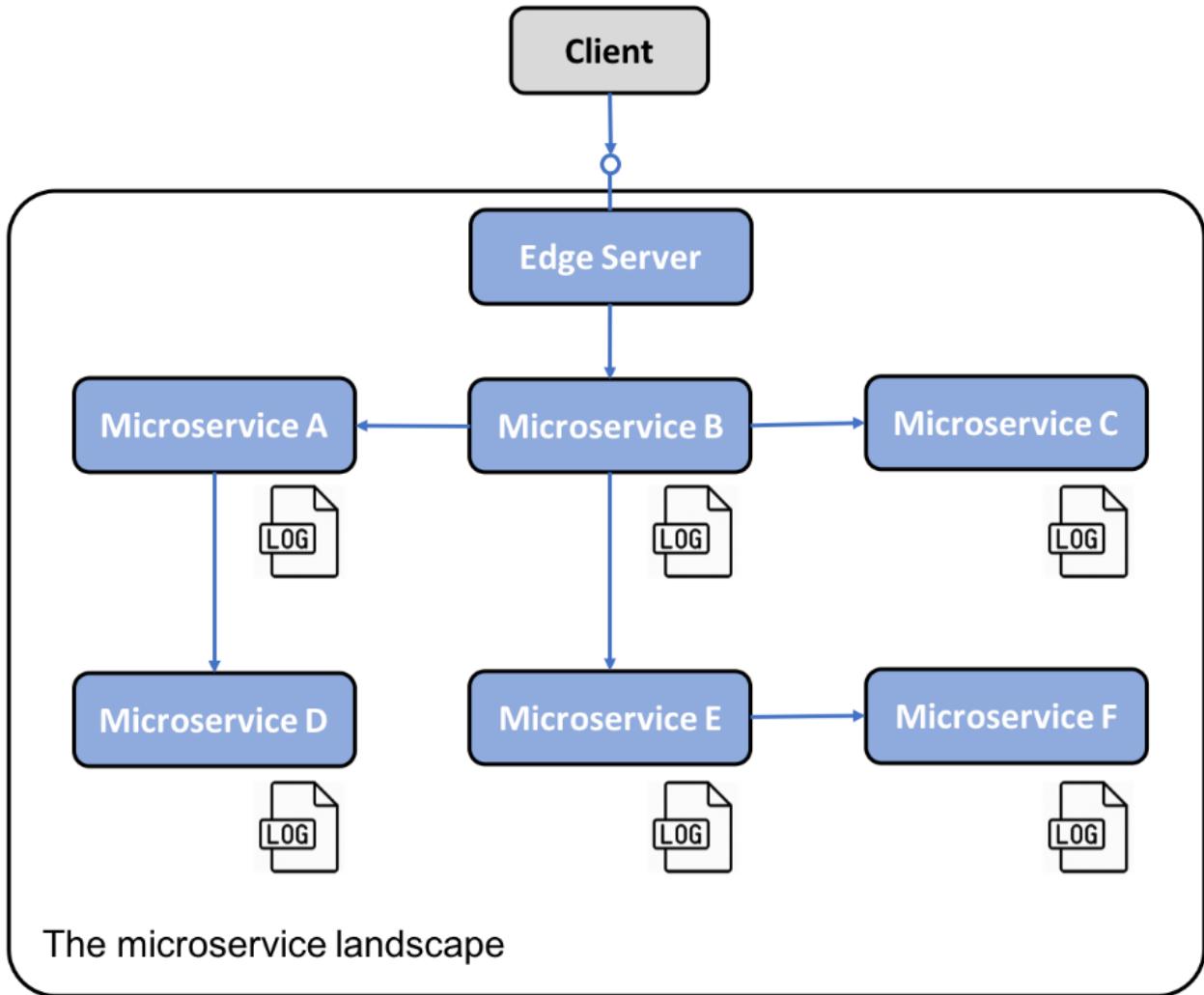
### 3. Reactive Microservices



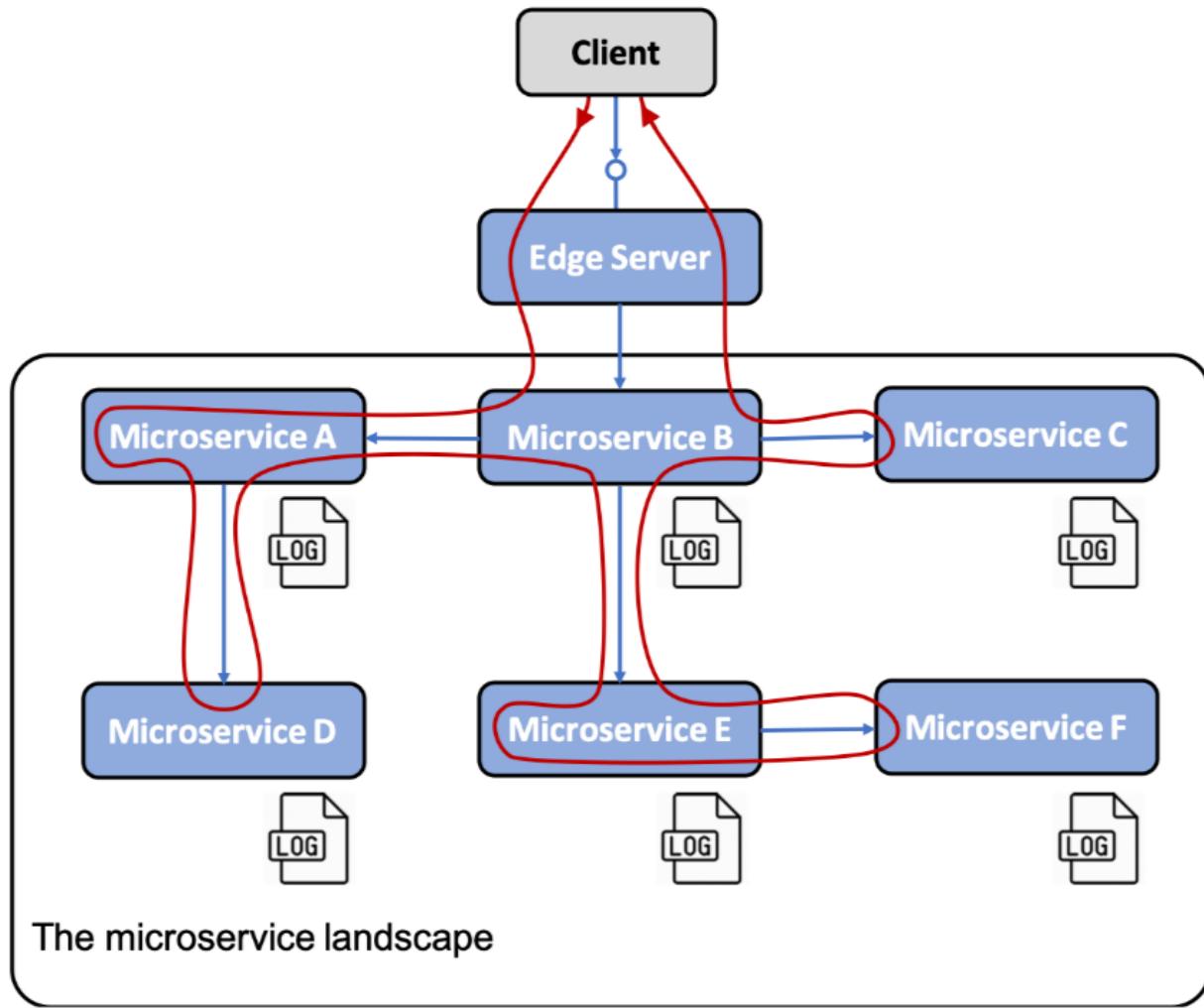
#### 4. Central Configuration

Add a new component, **a configuration server**, to the system landscape to store the configuration of all the microservices.

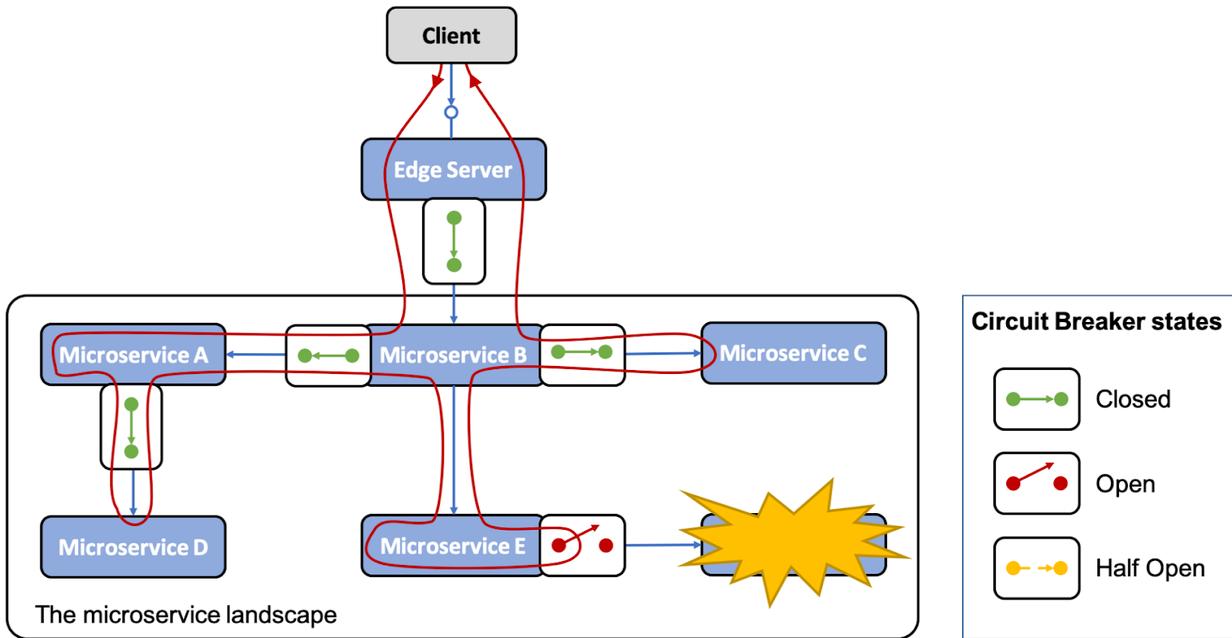
#### 5. Centralized Log Analysis



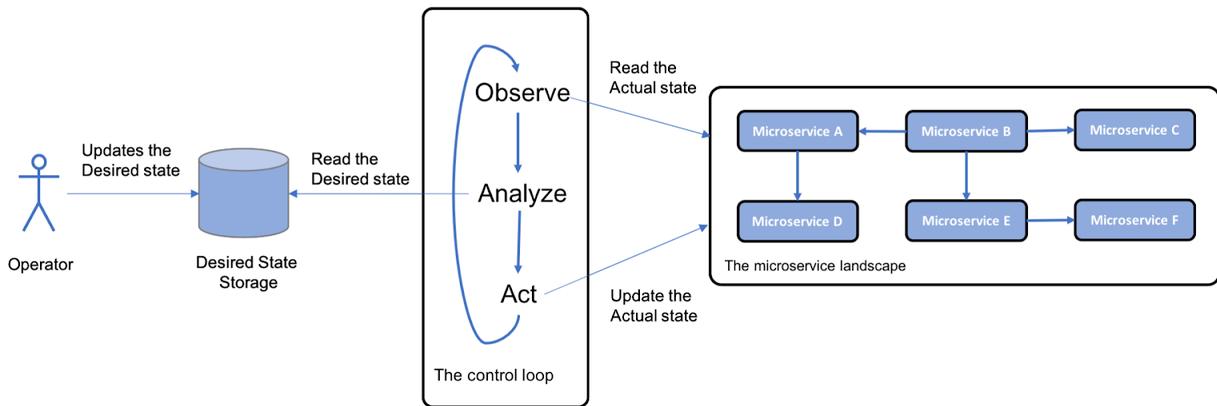
## 6. Distributed Tracing



## 7. Circuit Breaker



## 8. Control Loop



## 9. Centralized monitoring & alarms



<https://docs.google.com/presentation/d/1B7a5dZhB7ERQsqESvKRskukPIEB6pG5HcCIVOa3BPFw/edit>

## Eclipse MicroProfile specifications

<https://projects.eclipse.org/projects/technology.microprofile>

<https://download.eclipse.org/microprofile/microprofile-2.2/microprofile-spec-2.2.html>

## Quarkus Workshop ( super-heroes )

## workshop - superheroes

Welcome to Super Heroes Fight!

Astérix



⚡ 9



Dexterity, Intelligence, Jump, Peak Human Condition, Reflexes,  
Stamina, Super Speed, Super Strength

NEW FIGHTERS

FIGHT !

Match

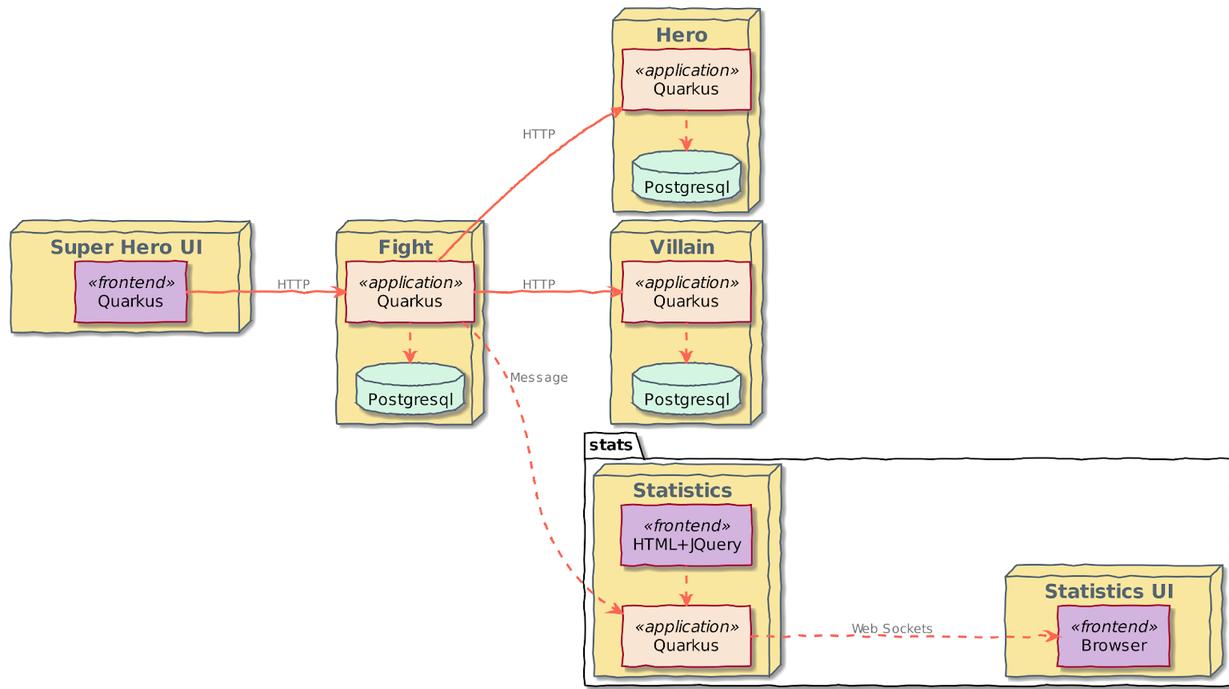
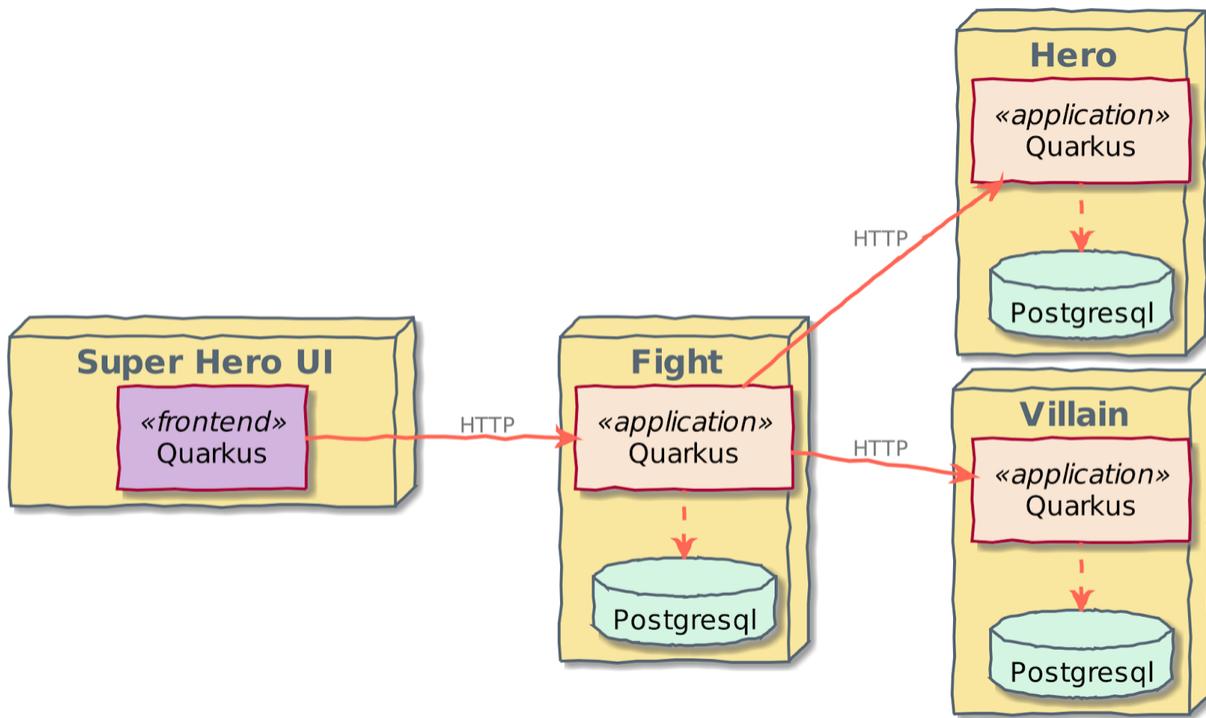


⚡ 14



Accelerated Healing, Durability, Energy Absorption, Energy  
Blasts, Enhanced Hearing, Flight, Invulnerability, Jump, Super  
Breath, Super Speed, Super Strength, Telekinesis, Vision - Heat,  
Vision - Telescopic, Vision - X-Ray

ID	Fight Date	Winner	Loser
10	Oct 14, 2019, 11:04:22 AM	Black Canary	Superman
9	Oct 14, 2019, 11:04:22 AM	Tanker Bug	Shuri
8	Oct 14, 2019, 11:04:22 AM	Moondragon	Darth Plagueis
7	Oct 14, 2019, 11:04:22 AM	The Eraser	Gandalf The White



## Super heroes

What you are going to learn:

- What is Quarkus and how you can use it
- How to build an HTTP endpoint (REST API) with Quarkus
- How to access a database
- How you can use Swagger and OpenAPI
- How you test your microservice
- How you improve the resilience of your service
- How to build event-driven and reactive microservices with Kafka
- How to build native executable
- How to extend Quarkus with extensions
- And many more

Ready? Here we go!

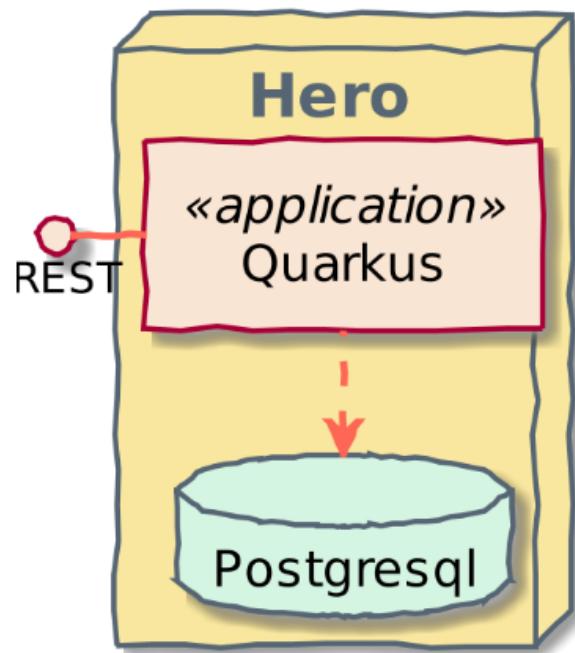
---

## Software Requirements

This workshop will make use of the following software, tools, frameworks that you will need to install and know (more or less) how it works:

- Any IDE you feel comfortable with (eg. IntelliJ IDEA, Eclipse IDE, VS Code..)
  - JDK 8 / 11
  - GraalVM 19.2.1 or 19.3.1
  - Maven 3.6.x
  - Docker
  - **cURL (or any other command line HTTP client)**
  - **Node JS (optional, only if you are in a *frontend* mood)**
- 

## Creating a REST/HTTP Microservice



you learn:

- how to create a new Quarkus application
  - how to implement REST API using JAX-RS
  - how to compose your application using CDI beans
  - how to access your database using Hibernate with Panache
  - how to use transactions
  - how to enable OpenAPI and Swagger-UI
- 

## Quarkus - Custom Extension

Most of the Quarkus magic happens inside extensions.

The goal of an extension is to compute ***just enough bytecode*** to start the services that the application requires, and drop everything else.

So, when writing an extension, you need to distinguish the action that:

- Can be done at build time
- Must be done at runtime

Because of this distinction, extensions are divided into 2 parts: a build time augmentation and a runtime.

The augmentation part is responsible for all the metadata processing, annotation scanning, XML parsing...The output of this augmentation is **recorded bytecode**, which, then, is executed at runtime to instantiate the relevant services.

## The extension framework

To build an extension, Quarkus provides a framework to:

- read configuration from the `application.properties` file and map it to objects,
- read metadata from classes without having to load them, this includes classpath and annotation scanning,
- generate bytecode if needed (for proxies for instance),
- pass sensible defaults to the application,
- make the application compatible with GraalVM (resources, reflection, substitutions),
- implement hot-reload

## Eclipse Vertx

## Eclipse Vertx with Quarkus

# **Serverless application with Quarkus**