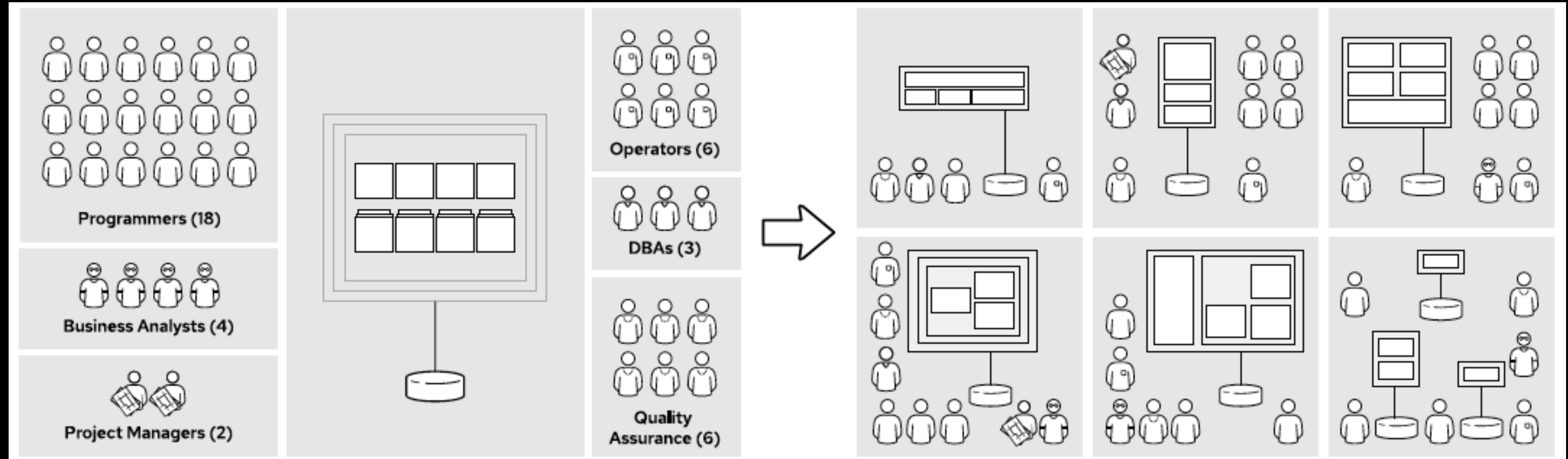


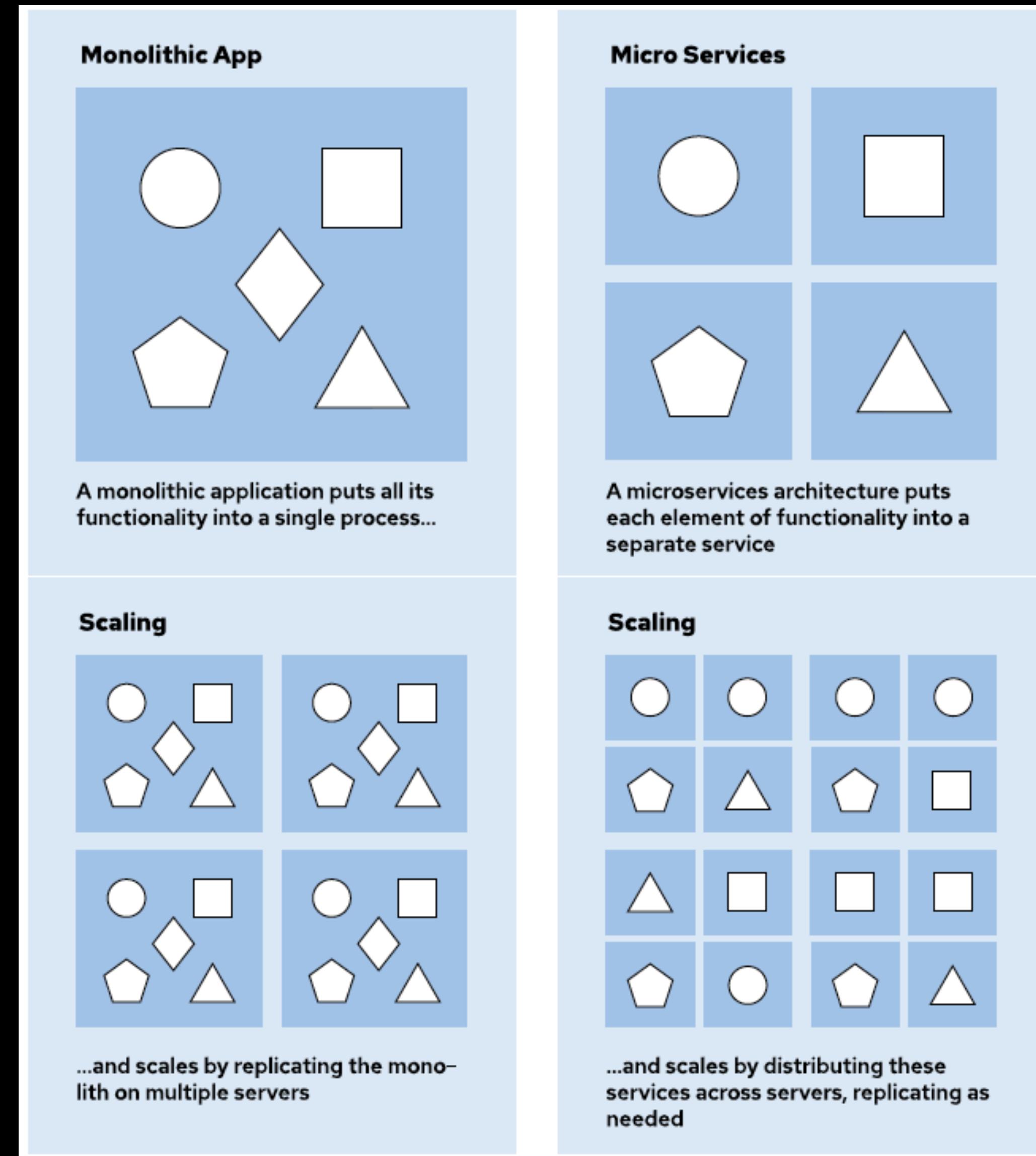
Desarrollo de micro servicios cloud-native con Quarkus

Micro servicios

Evolución de la estructura de un team de una arquitectura monolítica a micro servicios



Comparando aplicaciones monolíticas vs micro servicios



Comparación

Monolitos vs Micro servicios

- Modelar un simple dominio de negocio, haciendo este simple y pequeño
- Implementar almacenamiento y colaboración externa
- Independiente y pobre acoplamiento con otros servicios
- Son desarrollados, probados y desplegados de forma independiente
- Son Stateless
- Facilmente reemplazables y actualizables
- Escalados y desplegados de forma independiente respecto a otros servicios
- Tiene un contrato publicado de formato individual (API)

Principios de adopción exitosa

Micro servicios

- Reorganizando a DevOps
- Empaquetando el servicio como un contenedor
- Usando una infraestructura elástica
- Desplegando el servicio de forma independiente
- Integración continua y pipeline de despliegue

Complejidades de la arquitectura de Microservicios

Micro servicios

- Complejidad en la interacción
- Consistencia
- Administración distribuida
- Seguridad

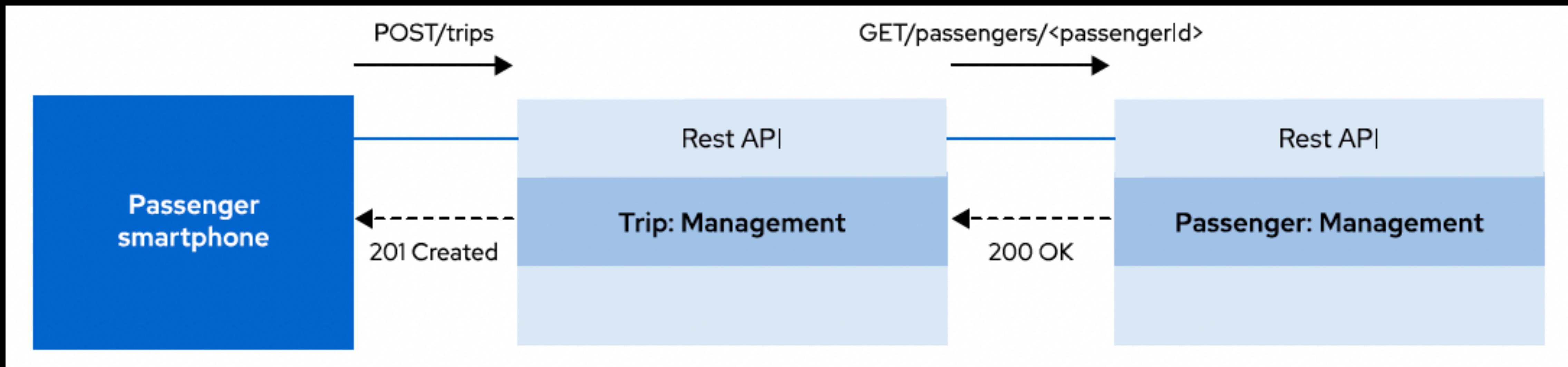
Principios de una Arquitectura Resiliente

Micro servicios

- Diseño orientada a dominios y contextos acotados
 - Ventajas
 - Cambios en el modelo de dominio solo afecta a un numero limitado de servicios
 - Los servicios son autónomos
 - Desventajas
 - Se necesita un conocimiento experto para definir un contexto acotado
 - La complejidad se incrementa cuando se quiere mantener los contextos acotados consistentes
- Desplegar de forma independiente en un Runtime liviano
- Diseñara para fallar

Patrones y Buenas Prácticas para Micro servicios

Comunicación síncrona entre servicios



Comunicación sincrona entre servicios

Ventajas y Desventajas

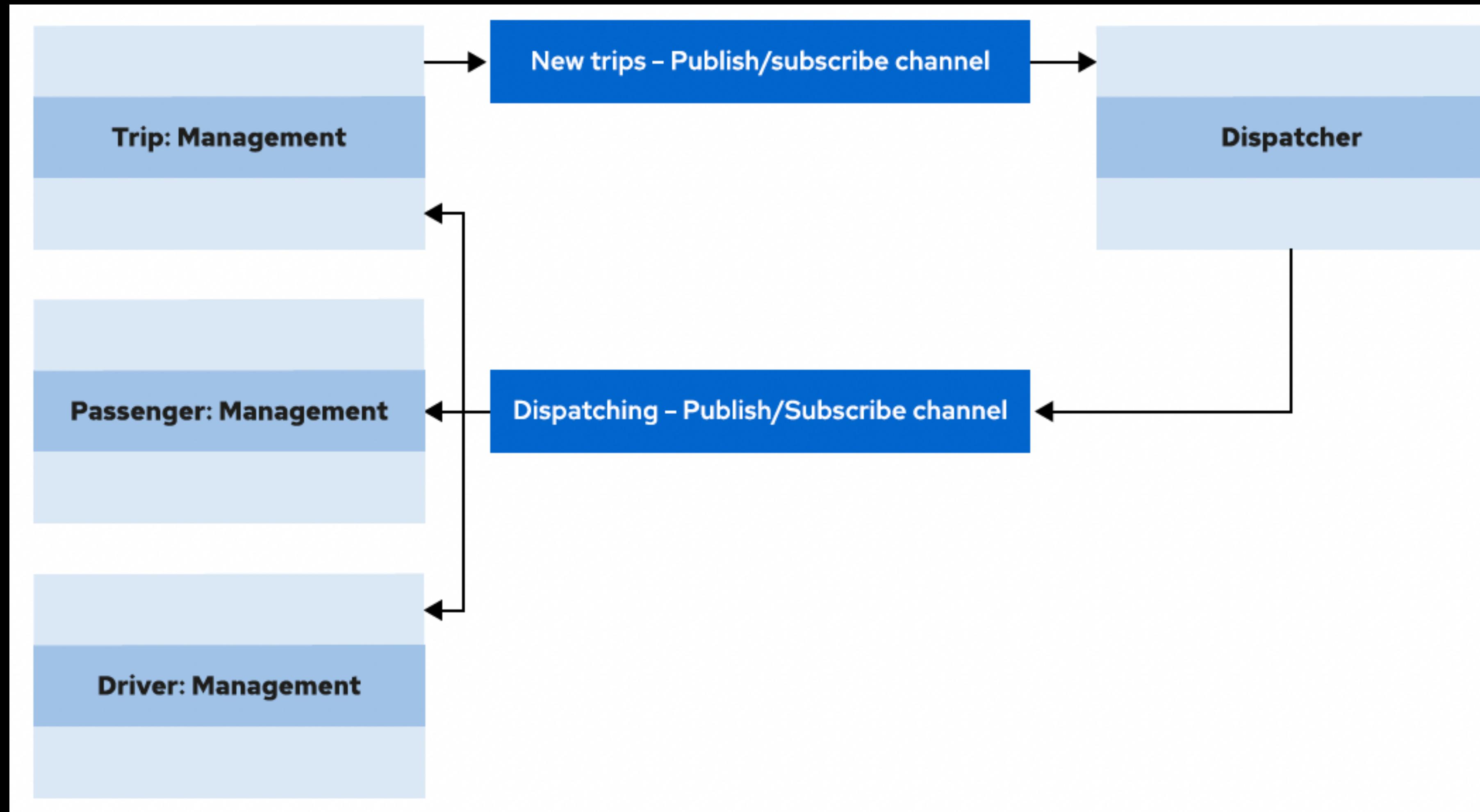
- **Ventajas**

- Fácil para programar y probar
- Proveer una mejor respuesta en tiempo real
- Firewall friendly, porque, este usa puertos estándar
- No hay necesidad por un broker como intermediario u otro software de integración

- **Desventajas**

- Soporta solo interacción al estilo request-y-response
- El cliente bloquea la lógica actual del proceso de negocio, mientras esperan por una respuesta
- Llamadas encadenadas entre servicios incrementan el tiempo para completar el request inicial
- Se necesita que tanto el cliente y el servicio este disponible para la que la comunicación completa se de por completo
- Fuerza al cliente conocer la ubicación del servicio, o usa un mecanismo de descubrimiento de servicio para localizar las instancias del servicio

Comunicación asíncrona basada en mensajes



Comunicación asincrona entre servicios

Ventajas y Desventajas

- **Ventajas**
 - Desacopla el cliente del servicio
 - Implementa message buffering
 - Habilita una interacción flexible entre el cliente y servicio
- **Desventajas**
 - Incrementa la complejidad operacional
 - Incrementa la complejidad de la implementación de las interacciones basadas en request-response

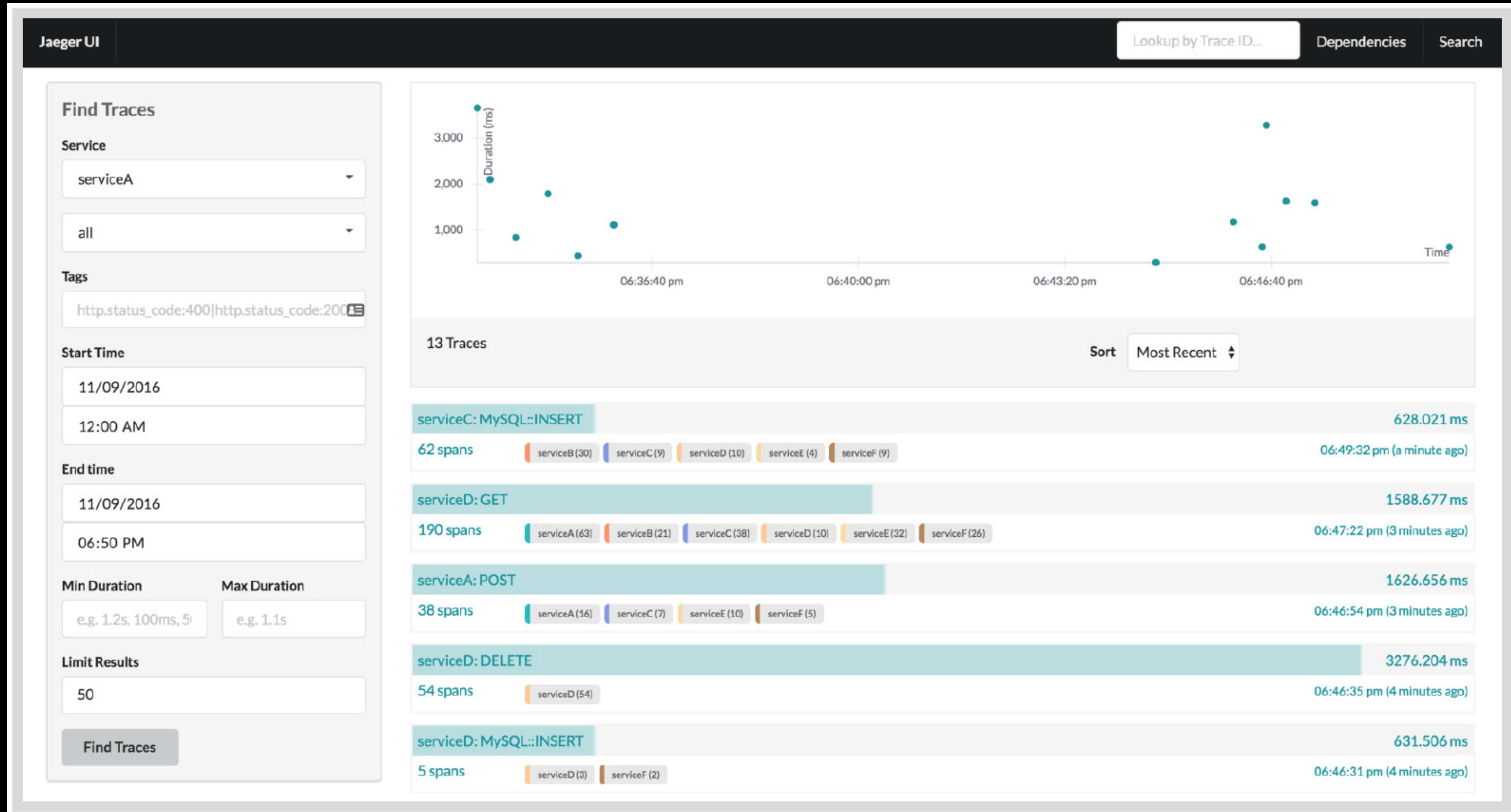
Diseñar para fallas

Tolerancia a fallos

- Time-out
- Retry
- Fallback
- Circuit Breaker

Diseñando para Observabilidad

Describiendo Tracing distribuido



El **OpenTracing API** es un estándar abierto que tu puedes usar para agregar tracing a tus aplicaciones distribuidas. Este soporta múltiples lenguajes como Java, JavaScript, y Go.

<https://opentracing.io>

Agregando **Métricas** a un micro servicio: Número de requests recibidos por un endpoint, el tiempo necesario para responder, o la cantidad de memoria usada por un servicio. En micro servicios se necesita coleccionar y centralizar estas métricas. Herramientas como **Prometheus** y **Grafana** nos proveen herramientas de visualización para mostrar mejor las métricas.

<https://prometheus.io>

<https://grafana.com>

Diseñando Seguridad

Micro servicios

- Single Sign-on
- Sesiones distribuidas
- Token del lado del cliente
- Token del lado del cliente con API Gateway

Quarkus

Microprofile

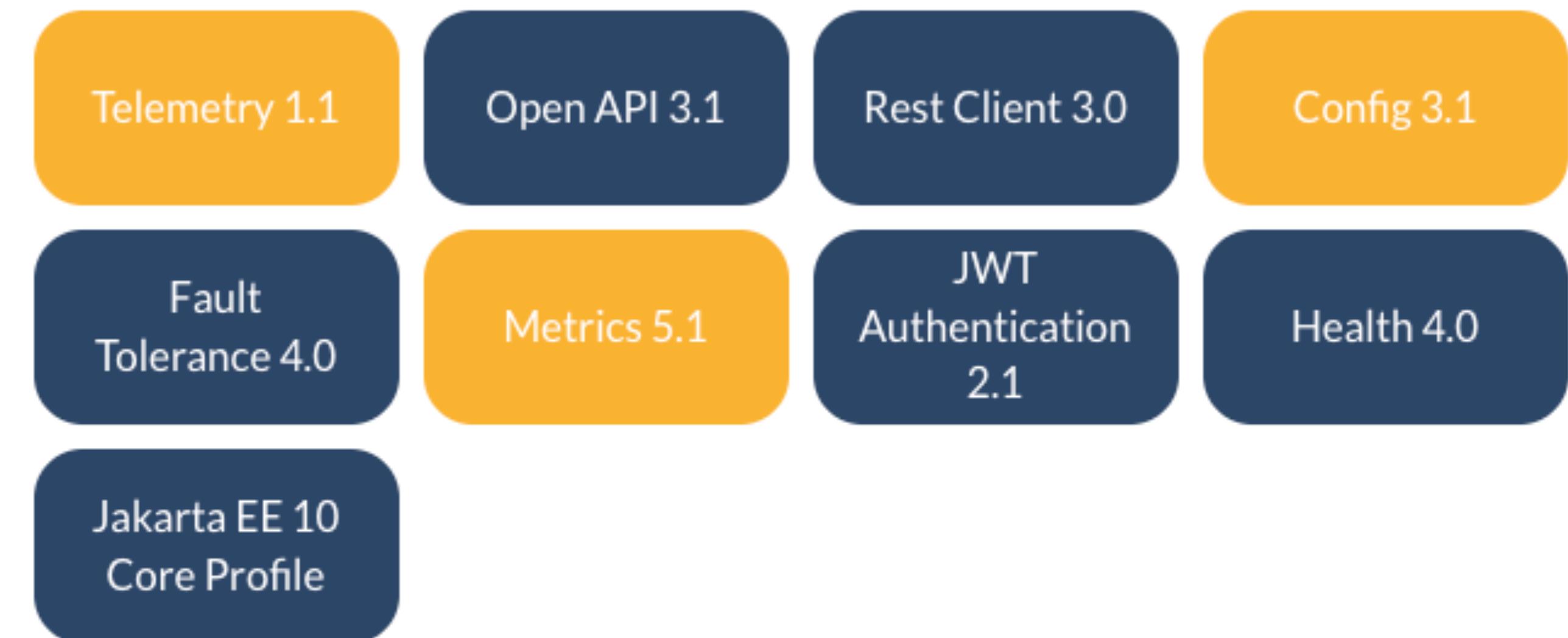
<https://micropatterns.io/>

- La industria comienza a girar hacia una arquitectura basada en microservicios para el desarrollo de nuevas aplicaciones, que también son en su mayoría aplicaciones nativas de la nube.
- La especificación MicroProfile es un joint-venture entre la fundación de Eclipse y otros vendors como Red Hat. La especificación define una plataforma que optimiza Java para una arquitectura basada en micro servicios y provee portabilidad para multiples runtimes.
- Proporciona un marco flexible que admite muchos de los patrones de diseño más comunes que ya utilizan los desarrolladores de Java que desarrollan microservicios en toda la industria.

MicroProfile 6.1

MicroProfile 6.1 adopts Jakarta EE 10 Core Profile. Jakarta EE 10 Core Profile defines a base profile to deliver the components used by MicroProfile as a single, complete package. In this release, MicroProfile Config, Metrics and Telemetry has minor updates.

MicroProfile 6.1 Presentation



New in MicroProfile
6.1



Updated in
MicroProfile 6.1



No changes since
MicroProfile 6.0

MicroProfile Starter



groupId *

artifactId *

MicroProfile Version

Java SE Version

Project Options

MicroProfile Runtime *

Examples for specifications [Select All](#) [Clear All](#)

Config Fault Tolerance
 JWT Auth Metrics
 Health Checks OpenAPI
 OpenTracing TypeSafe Rest Client

DOWNLOAD

MicroProfile Starter helps developers kickstart their microservices development journey, choosing the runtime they're most comfortable with from the list of available implementations for the MicroProfile version selected.

Try It Out!

MicroProfile Starter

```
@Path("/hello")
@RequestScoped
public class HelloService {

    @Inject
    @ConfigProperty(name = "greeting", defaultValue = "Welcome to MicroProfile")
    private String greeting;

    @GET
    @Path("{name}")
    public String sayHello(@PathParam("name") String name) {
        return greeting + " " + name + ", lets get started!";
    }
}
```

MicroProfile Starter helps developers kickstart their microservices development journey, choosing the runtime they're most comfortable with from the list of available implementations for the MicroProfile version selected.

Try It Out!

Microprofile

<https://microprofile.io/>

- Microprofile no es un estándar completo que requiera al Java Community Process (JCP). Estos estándares completos requieren años para completarse y son engorrosos de actualizar. Debido a que las aplicaciones de microservicios evolucionan constantemente, ningún estándar es realmente definitivo.
- La versión inicial de MicroProfile incluyó JAX-RS, CDI, y JSON-P desde Java EE, lo cual es lo mínimo requerido para construir un microservicio en Java. MicroProfile usa estas especificaciones porque son eficientes, hay familiaridad en su uso y son omnipresentes en el desarrollo Java.

MicroProfile platform specifications

Config	Externalizes application configuration
Fault Tolerance	Defines multiple strategies to improve application robustness
Health	Expresses application health to the underlying platform
JWT RBAC	Secures RESTful endpoints
Metrics	Exposes platform and application metrics
Open API	Java APIs for the OpenAPI specification that documents RESTful endpoints
OpenTelemetry	Defines behaviors and an API for accessing an OpenTelemetry-compliant Tracer object
REST Client	Type-safe invocation of REST endpoints

MicroProfile stand-alone specifications

GraphQL	Java API for the GraphQL query language
Reactive Streams operators	Provides asynchronous streaming to be able to stream data
Reactive Streams messaging	Provides asynchronous messaging support based on Reactive Streams

Quarkus

<https://quarkus.io/>

- Quarkus, es un runtime Java para micro servicios. Surgió como un esfuerzo colectivo de diferentes tecnologías que abarcan la implementación de MicroProfile. Se enfoca en factores claves como el startup time y performance, donde implementaciones anteriores como Thorntail fallaron para mejorar.
- Al brindar un proceso de compilación más completo, Quarkus brinda la mejor experiencia para micro servicios. Este proceso de compilación realiza actividades que normalmente se realizan durante el inicio o en tiempo de ejecución, incluida la configuración, el arranque y la preparación de reflection.

Quarkus

<https://quarkus.io/>

- Quarkus tiene muchas características que están relacionadas a ser cloud-native o container-native, soportando diferentes modelos de programación, siendo amigable para los desarrolladores y siguiendo los estándares o especificaciones.
- Quarkus es soportado por Red Hat. Este provee un conjunto de librerías que ya han sido probadas anteriormente en muchos proyectos de Red Hat para su usuario en entornos productivos. Seguridad y bug fixes son liberados constantemente y aplicados para brindar seguridad a los clientes empresariales.

Container Native

<https://quarkus.io/>

- Fast startup
- Small memory footprint
- Smaller disk usage

Unificación de modelos de programación

<https://quarkus.io/>

- Imperativo
- Reactivo

Centrado en el Desarrollo

<https://quarkus.io/>

- Configuración de todas las tecnologías como dependencias en el mismo archivo de configuración
- Instant live reloading de cambios en modo desarrollador
- Un Marco obstinado que establece valores predeterminados utilizables
- Construcción nativa integrada para todas las bibliotecas incluidas

Librerías estándar

<https://quarkus.io/>

MicroProfile Specification	Project Name	Quarkus Dependency (Maven group:artifact)
Rest Client	SmallRye REST Client	io.quarkus:quarkus-smallrye-rest-client
Fault Tolerance	SmallRye Fault Tolerance	io.quarkus:quarkus-smallrye-fault-tolerance
Health Check	SmallRye Health	io.quarkus:quarkus-smallrye-health
Metrics	SmallRye Metrics	io.quarkus:quarkus-smallrye-metrics
JWT Security	SmallRye JWT	io.quarkus:quarkus-smallrye-jwt
OpenAPI	SmallRye OpenAPI	io.quarkus:quarkus-smallrye-openapi
OpenTelemetry	Quarkus OpenTelemetry	io.quarkus:quarkus-opentelemetry
Reactive Streams Operators	SmallRye Reactive Streams Operators	io.quarkus:quarkus-smallrye-reactive-streams-operators
Reactive Streams Messaging	SmallRye Reactive Messaging	io.quarkus:quarkus-smallrye-reactive-messaging

Usando el método Nativo

<https://quarkus.io/>

- Uno de los principales drivers en Quarkus es proveer el mejor soporte para Serverless, functions as a service (**FaaS**), y otras cargas de trabajo basadas en cloud. En un entorno de este tamaño, el uso de la memoria y startup time es super crítico.
- A pesar de las mejoras de rendimiento con respecto a los marcos de nube tradicionales, un tiempo de inicio de un segundo podría ser demasiado largo. Para esto, quarkus ofrece soporte total para la compilación anticipada (AoT) de **GraalVM**.

Usando el método Nativo

<https://quarkus.io/>

- GraalVM es una JVM especial de Oracle que ofrece compilación AoT para aplicaciones java ejecutables, así como otras características.
- La suposición del mundo cerrado significa que la JVM necesita saber en el momento de la compilación qué clases va a utilizar la aplicación. Para admitir la generación de ejecutables nativos, los desarrolladores deben cumplir con este supuesto. Quarkus logra esto proporcionando sus propias implementaciones de librerías y evaluando las clases requeridas en el momento de la compilación en lugar de en el tiempo de ejecución, a través de, reflexión.

Usando el método Nativo

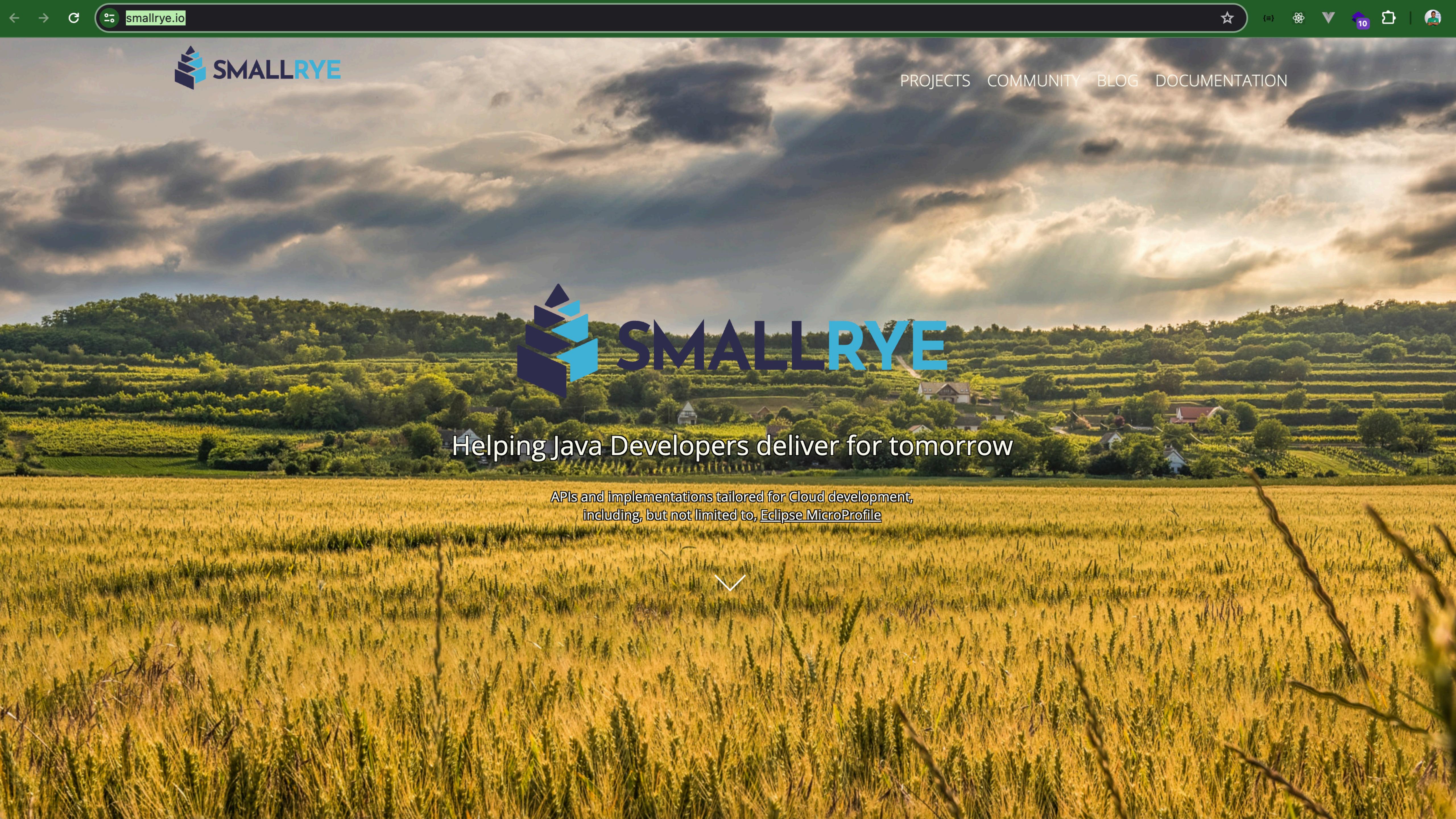
<https://quarkus.io/>

- Durante la compilación AoT, GraalVM elimina todas las clases innecesarias de la aplicación, librerías dependientes y JVM.
- Red Hat tiene una distribución llamada Mandrel de GraalVM que se enfoca en generar imágenes nativas para aplicaciones.
- Este soporte nativo de AoT y GraalVM usando Mandrel hace a Quarkus una excelente elección para despliegues cloud.

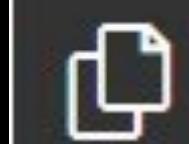
Ventajas de Quarkus

<https://quarkus.io/>

- Comparado a otros frameworks cloud-native basados en Java, Quarkus ofrece los siguientes beneficios:
 - Rápido tiempo de respuesta en general
 - Bajo uso de memoria
 - Alto consumo en rutas reactivas
 - Soporte completo a contenedores
 - Compliance con MicroProfile



Entorno de desarrollo



EXPLORER

...

FIRSTMAVENPROJECT

first-app

2

src

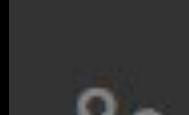
main\java\bmt

App.java

> test

> target

pom.xml



App.java X

1

package bmt;

2

import java.sql.*;

3

public class App

4

{

Run | Debug

5 public static void main

6

{ final String DB_URL = "jdbc:mysql://localhost:3306/BoostMyTool?"; final String USERNAME = "root"; final String PASSWORD = ""; }

7

8

9

10

11

12

13

14

15

16

17



PROBLEMS 2

OUTPUT

X

X

```
PS C:\Users\BoostMyTool\Desktop\FirstMavenProject> cd 'c:\Users\BoostMyTool\Desktop\FirstMavenProject'; & 'c:\Users\BoostMyTool\.vscode\extensions\vscjava.vscode-java-debug-0.36.0\scripts\launcher.bat' 'C:\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-Dfile.encoding=UTF-8' '@C:\Users\BOOSTM~1\AppData\Local\Temp\cp_23kvbtn39skne464sbeoe7rfx.argfile' 'bmt.App'  
Connected database successfully...  
Inserted records into the table...  
PS C:\Users\BoostMyTool\Desktop\FirstMavenProject> 
```



Java™



GraalVM™





NEXT STEPS