

Introducción a Quarkus

José Amadeo Díaz Díaz



Fundador de JoeDayz.pe



Java Champion



Miembro de @PeruJUG



[@jamdiazediaz](https://twitter.com/jamdiazediaz)

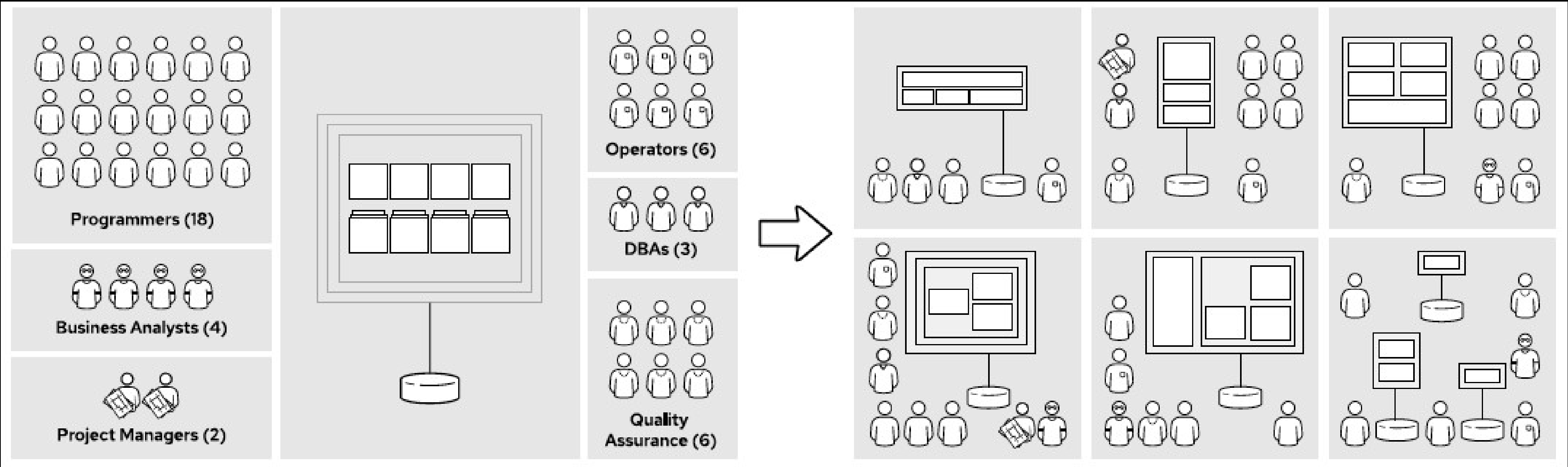


<https://github.com/joedayz>



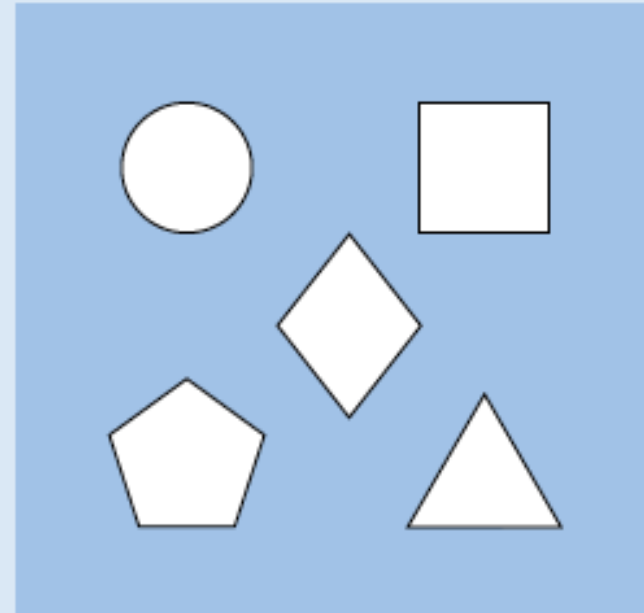
Micro servicios

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



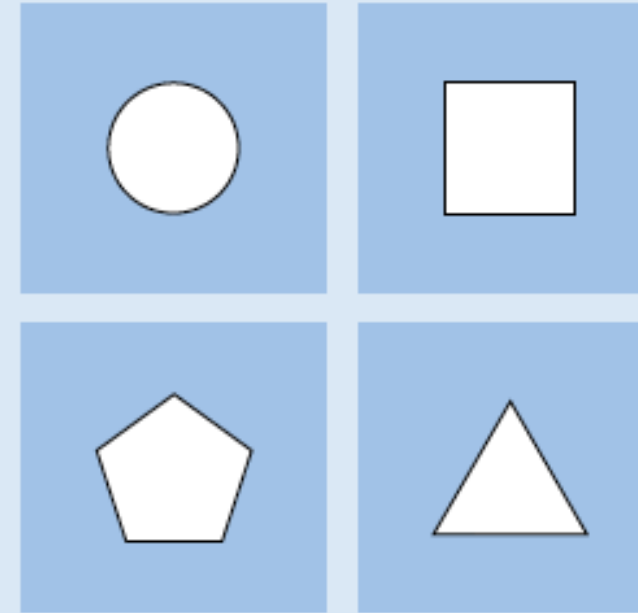
Comparando aplicaciones monolíticas vs micro servicios

Monolithic App



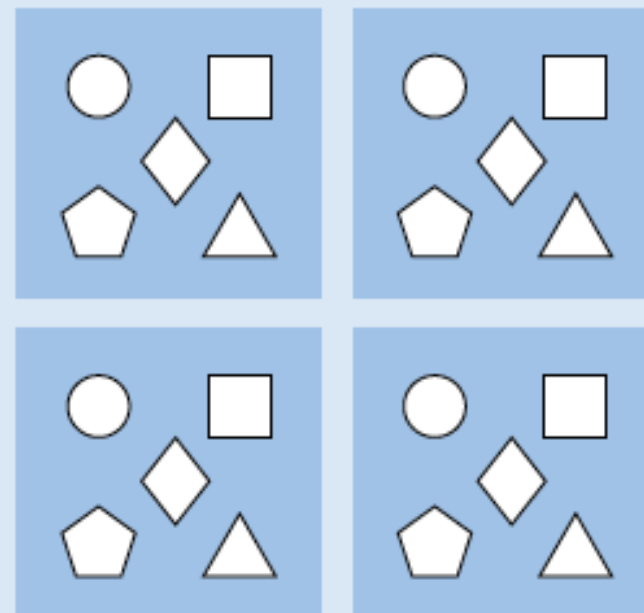
A monolithic application puts all its functionality into a single process...

Micro Services



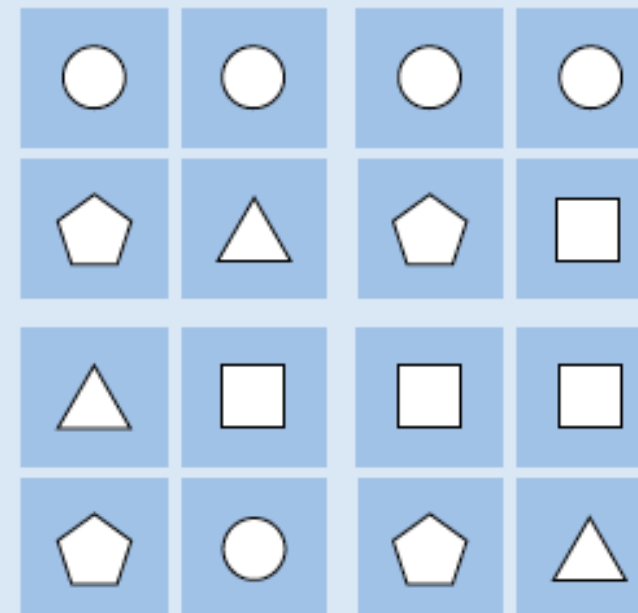
A microservices architecture puts each element of functionality into a separate service

Scaling



...and scales by replicating the monolith on multiple servers

Scaling



...and scales by distributing these services across servers, replicating as needed

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 formado individual (API)

Principios de adopción exitosa

Micro servicios

- Reorganizando a DevSecOps
- 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**

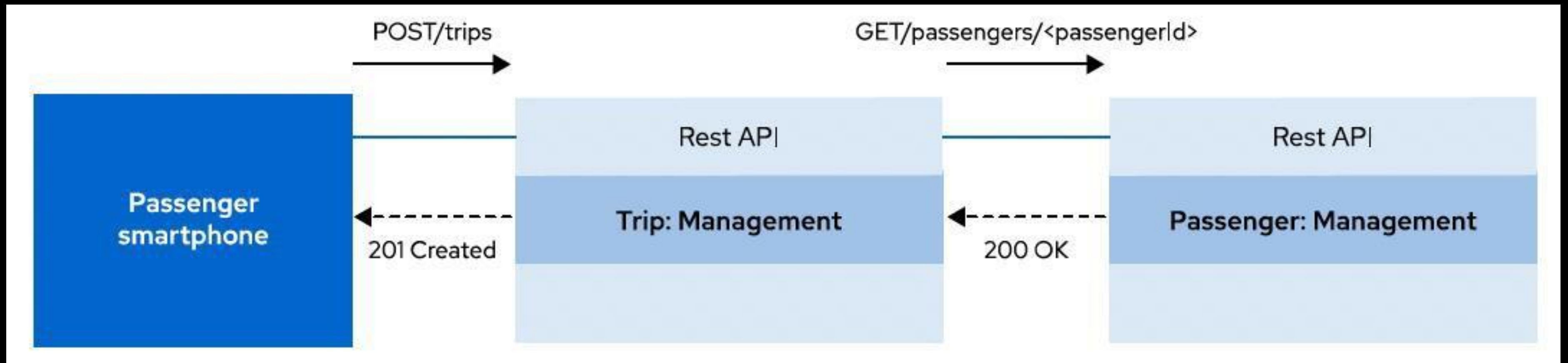
Micro servicios

- 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**
- Diseñar para fallar

<https://www.redhat.com/en/topics/microservices>

Patrones y Buenas Prácticas para Micro servicios

Comunicación síncrona entre servicios



Comunicación sincrónica entre servicios

Ventajas y Desventajas

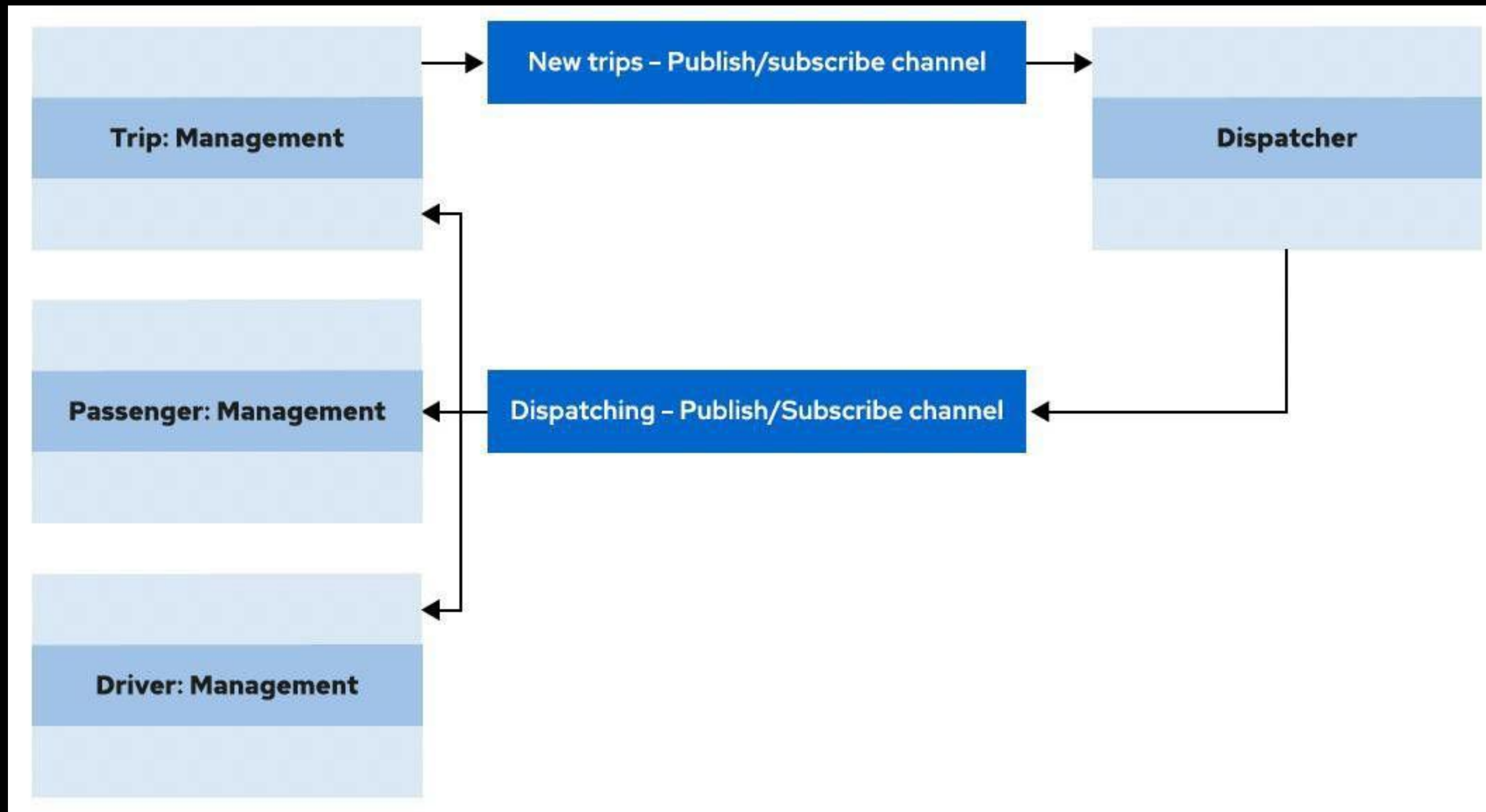
- **Ventajas**

- 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 asincrónica 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

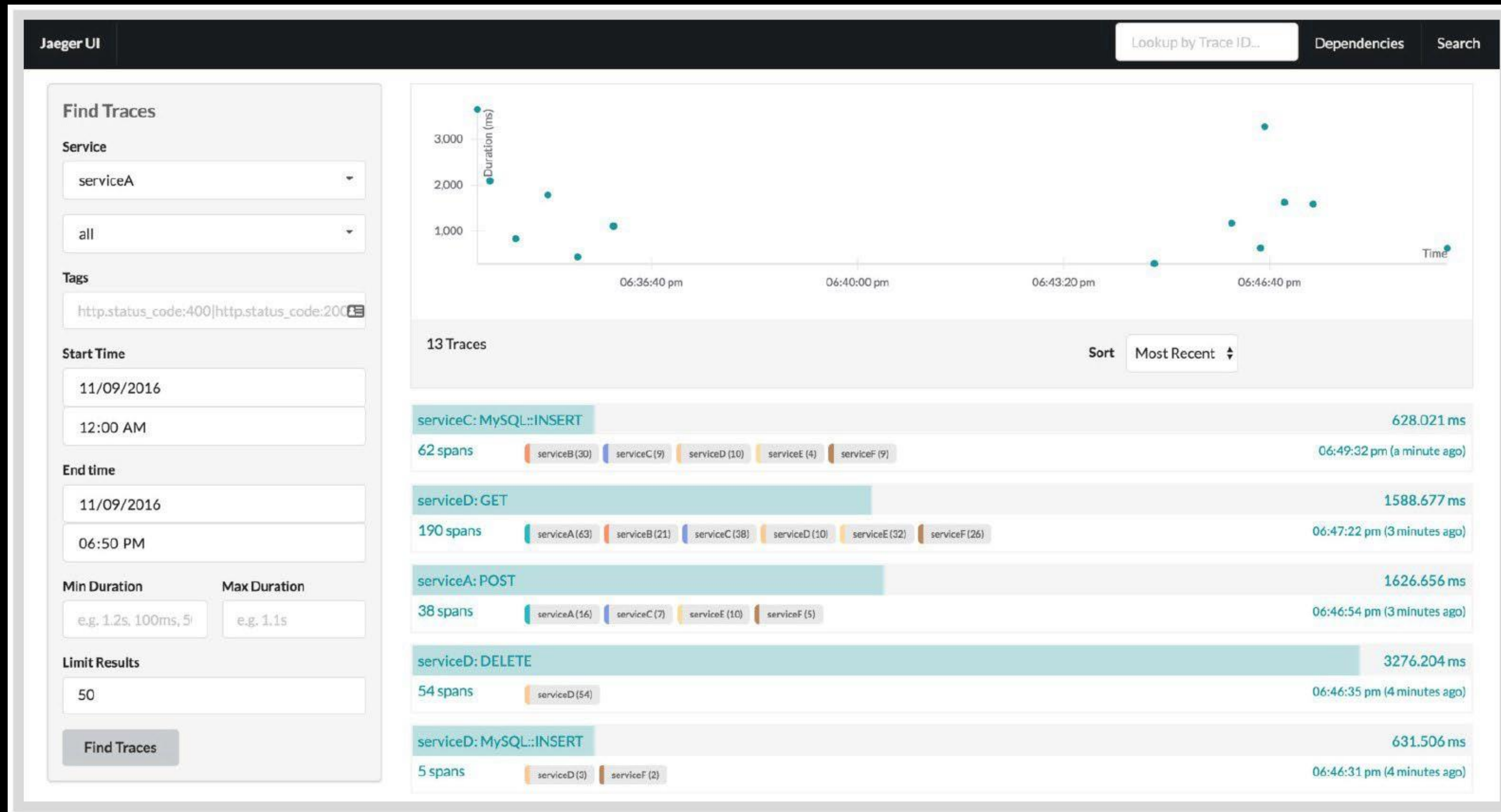
Diseñar para fallas

Tolerancia a fallos

- Time-out
- Retry
- Fallback
- Circuit Breaker

Diseñando para Observabilidad

Describiendo Tracing distribuido



DEPRECADO!

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

<https://opentracing.io>

El OpenTelemetry (quarkus-opentelemetry)

<https://opentelemetry.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://microprofile.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 múltiples 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 7.1

MicroProfile 7.1 was released on June 17th, 2025. This release updates two component specifications: MicroProfile Telemetry and MicroProfile OpenAPI.

[MicroProfile 7.1 Presentation](#)



New in MicroProfile 7.1



Updated in MicroProfile 7.1




No changes since MicroProfile 7.0

start.microprofile.io

☆

Relaunch to update

All Bookmarks

MICROPROFILE™
OPTIMIZING ENTERPRISE JAVA

See The Code

MicroProfile Starter

Generate MicroProfile Maven Project with Examples

groupId *

com.example

artifactId *

demo

MicroProfile Version

Java SE Version

Project Options

Build Tool

☒ Maven

☐ Gradle

MicroProfile Runtime *

Examples for specifications

DOWNLOAD

MicroProfile Version

▼

5.0	🔗
4.1	🔗
4.0	🔗
3.3	🔗
3.2	🔗
3.0	🔗
2.2	🔗
2.1	🔗
2.0	🔗
1.4	🔗
1.3	🔗
1.2	🔗

KumuluzEE 🔗

WildFly 🔗

WildFly Swarm 🔗

Thorntail V2 🔗

Helidon 🔗

Payara Micro 🔗

Quarkus 🔗

Apache TomEE 8.0.0-M3 🔗

Open Liberty 🔗

groupId *

com.example

artifactId *

demo

MicroProfile Version

MP 3.2

Java SE Version

Java 8

Project Options

Build Tool

☒ Maven

☐ Gradle

MicroProfile Runtime *

Quarkus

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

☐ Config 🔗

☐ Fault Tolerance 🔗

☐ JWT Auth 🔗

☐ Metrics 🔗

☐ Health 🔗

☐ OpenAPI 🔗

☐ OpenTracing 🔗

☐ Rest Client 🔗

DOWNLOAD

```
@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

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 microservicios 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 cómo el **startup time y performance**, donde implementaciones anteriores cómo **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	<code>io.quarkus:quarkus-smallrye-rest-client</code>
Fault Tolerance	SmallRye Fault Tolerance	<code>io.quarkus:quarkus-smallrye-fault-tolerance</code>
Health Check	SmallRye Health	<code>io.quarkus:quarkus-smallrye-health</code>
Metrics	SmallRye Metrics	<code>io.quarkus:quarkus-smallrye-metrics</code>
JWT Security	SmallRye JWT	<code>io.quarkus:quarkus-smallrye-jwt</code>
OpenAPI	SmallRye OpenAPI	<code>io.quarkus:quarkus-smallrye-openapi</code>
OpenTelemetry	Quarkus OpenTelemetry	<code>io.quarkus:quarkus-opentelemetry</code>
Reactive Streams Operators	SmallRye Reactive Streams Operators	<code>io.quarkus:quarkus-smallrye-reactive-streams-operators</code>
Reactive Streams Messaging	SmallRye Reactive Messaging	<code>io.quarkus:quarkus-smallrye-reactive-messaging</code>

Comparativa con Spring Framework

MicroProfile Specification	Quarkus / SmallRye Project	Spring Framework / Spring Boot Competencia 
Rest Client	SmallRye REST Client (<code>quarkus-smallrye-rest-client</code>)	Spring WebClient (en <code>spring-webflux</code>) o RestTemplate (en <code>spring-web</code>)
Fault Tolerance	SmallRye Fault Tolerance (<code>quarkus-smallrye-fault-tolerance</code>)	Spring Retry + Resilience4j (Hystrix antes, ya deprecated)
Health Check	SmallRye Health (<code>quarkus-smallrye-health</code>)	Spring Boot Actuator – Health Indicators
Metrics	SmallRye Metrics (<code>quarkus-smallrye-metrics</code>)	Spring Boot Actuator – Micrometer Metrics
JWT Security	SmallRye JWT (<code>quarkus-smallrye-jwt</code>)	Spring Security – JWT (con <code>spring-security-oauth2-resource-server</code>)
OpenAPI	SmallRye OpenAPI (<code>quarkus-smallrye-openapi</code>)	Springdoc OpenAPI (<code>springdoc-openapi-ui</code>)
OpenTelemetry	Quarkus OpenTelemetry (<code>quarkus-opentelemetry</code>)	Spring Observability + Micrometer Tracing + OpenTelemetry
Reactive Streams Operators	SmallRye Reactive Streams Operators (<code>quarkus-smallrye-reactive-streams-operators</code>)	Project Reactor (en el núcleo de Spring WebFlux)
Reactive Streams Messaging	SmallRye Reactive Messaging (<code>quarkus-smallrye-reactive-messaging</code>)	Spring Cloud Stream (con binders como Kafka, RabbitMQ, etc.)

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**

Spring Framework vs GraalVM

◆ 2019

• Spring Experimenta con GraalVM:

Empiezan los primeros experimentos con compilación nativa usando GraalVM.

- Pivotal (ahora VMware Tanzu) lanza **Spring Native Experimental**.
- Se probaba con **Spring Boot 2.3+**, pero era muy limitado (no todo funcionaba).



◆ 2020 – 2021

• Spring Native Project (separado de Spring Boot).

- Aparece un proyecto aparte (`spring-native`) para agregar soporte experimental a GraalVM.
- Usaba `spring-graalvm-native` y dependía mucho de **SubstrateVM** (el motor nativo de GraalVM).
- Requería configurar **hints** (`@NativeHint`) y complicadas reflexiones manuales.
- Compatible con **Spring Boot 2.4 y 2.5** en adelante.



◆ 2022

• Spring Boot 2.7 + Spring Native 0.12.x:

- Última versión experimental.
- Se trabajaba mucho con AOT (Ahead of Time processing).
- El feedback de la comunidad mostró la necesidad de integrar esto directamente en el core de Spring.

Spring Framework vs GraalVM

◆ 2023

- Spring Boot 3.0 (noviembre 2022):
 - Gran salto: Integración oficial y estable de GraalVM Native Images.
 - Ya no se usa el proyecto separado `spring-native`.
 - Introducción del AOT Engine en Spring Framework 6.
 - Uso de `mvn -Pnative native:compile` o `./gradlew nativeCompile` para generar ejecutables nativos.
 - Requiere GraalVM 22.x o superior y Java 17+.



◆ 2024

- Spring Boot 3.2 y 3.3:
 - Mejoras en tiempos de arranque y soporte más automático (menos hints manuales).
 - Integración con **Buildpacks** (Paketo) para generar imágenes nativas en contenedores sin que el desarrollador se preocupe de la instalación local de GraalVM.
 - Ejemplo:

```
bash
```

```
./mvnw spring-boot:build-image -Dspring-boot.build-image.native=true
```

Genera una imagen Docker nativa lista para producción.

Spring Framework vs GraalVM

◆ 2025 (Hoy)

- Spring Boot 3.3.x (última estable, que tú ya usas 😊).
 - Soporte **out-of-the-box** para GraalVM Native Image.
 - Funciona con **Java 21** y **GraalVM 24.x**.
 - Se recomienda usar **Spring AOT Engine** en lugar de configuraciones manuales.
 - La comunidad está reportando aplicaciones con **arranque en <100ms** y **uso de memoria muy bajo**, ideales para serverless, microservicios y apps cloud-native.

¿Qué es un Buildpack?

<https://spring.io/>

- Es una herramienta que **convierte tu aplicación en una imagen de contenedor lista para ejecutarse** sin que tengas que escribir un Dockerfile.
- Fue creado inicialmente por **Heroku** (2011), luego Cloud Foundry lo adoptó y ahora existe un estándar llamado **Cloud Native Buildpacks (CNB)** impulsado por la **CNCF** (Cloud Native Computing Foundation)

¿Cómo funciona el Buildpack?

<https://spring.io/>

- **Detect** → El buildpack detecta el tipo de aplicación (Java, Node.js, Python, etc).
- **Build** → Instala dependencias, empaqueta la aplicación y la configura
- **Export** → Genera una imagen de contenedor (ej. OCI/Docker image)

```
bash

./mvnw spring-boot:build-image
```

Ese comando:

- Usa el buildpack oficial de **Paketo** (para Java)
- Empaqueta tu aplicación **Spring Boot** dentro de una imagen optimizada (sin Dockerfile)
- Si agregas `-Dspring-boot.build-image.native=true`, además la compila con **GraalVM Native Image**.

Recursos

1. MicroProfile Working Group Charter: <https://www.eclipse.org/org/workinggroups/microprofile-charter.php>
2. Documentación de Quarkus: https://docs.redhat.com/en/documentation/red_hat_build_of_quarkus/3.8
3. Microprofile: <https://microprofile.io>
4. Quarkus (Upstream open source project): <https://quarkus.io>
5. SmallRye project: <https://smallrye.io>



[PROJECTS](#) [COMMUNITY](#) [BLOG](#) [DOCUMENTATION](#)

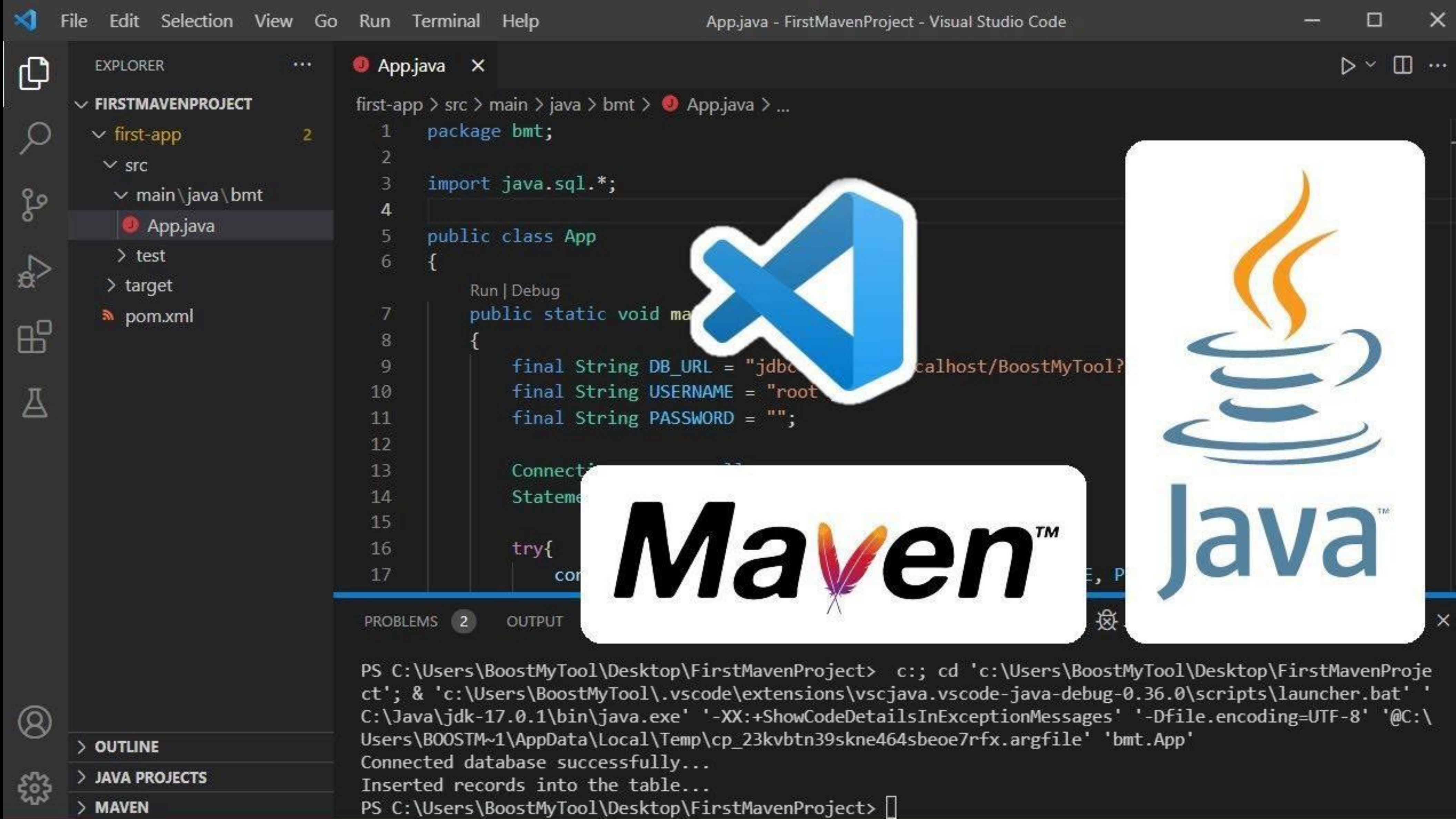


Helping Java Developers deliver for tomorrow

APIs and implementations tailored for Cloud development,
including, but not limited to, [Eclipse MicroProfile](#)



Entorno de desarrollo



GraalVM™



**Docker
Desktop**





Alibaba Cloud



Azure



Google Cloud



```
quarkus create app org.acme:getting-started \
    --extension='rest'
cd getting-started
```

```
mvn io.quarkus.platform:quarkus-maven-plugin:3.16.1:create \
    -DprojectId=org.acme \
    -DprojectId=getting-started \
    -Dextensions='rest'
cd getting-started
```


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

CONFIGURE YOUR APPLICATION

Group

Artifact

Build Tool

 MORE OPTIONS

 0

Generate your application (`\n` + `\r`)

Search



Search across all available extensions: rest, hibernate, web, data...

[View the full list of available extensions](#)

Start from an extensions preset



REST service



REST service with database



Event driven service with
Kafka



Command-line tool



Web app with Model-View-
Controller



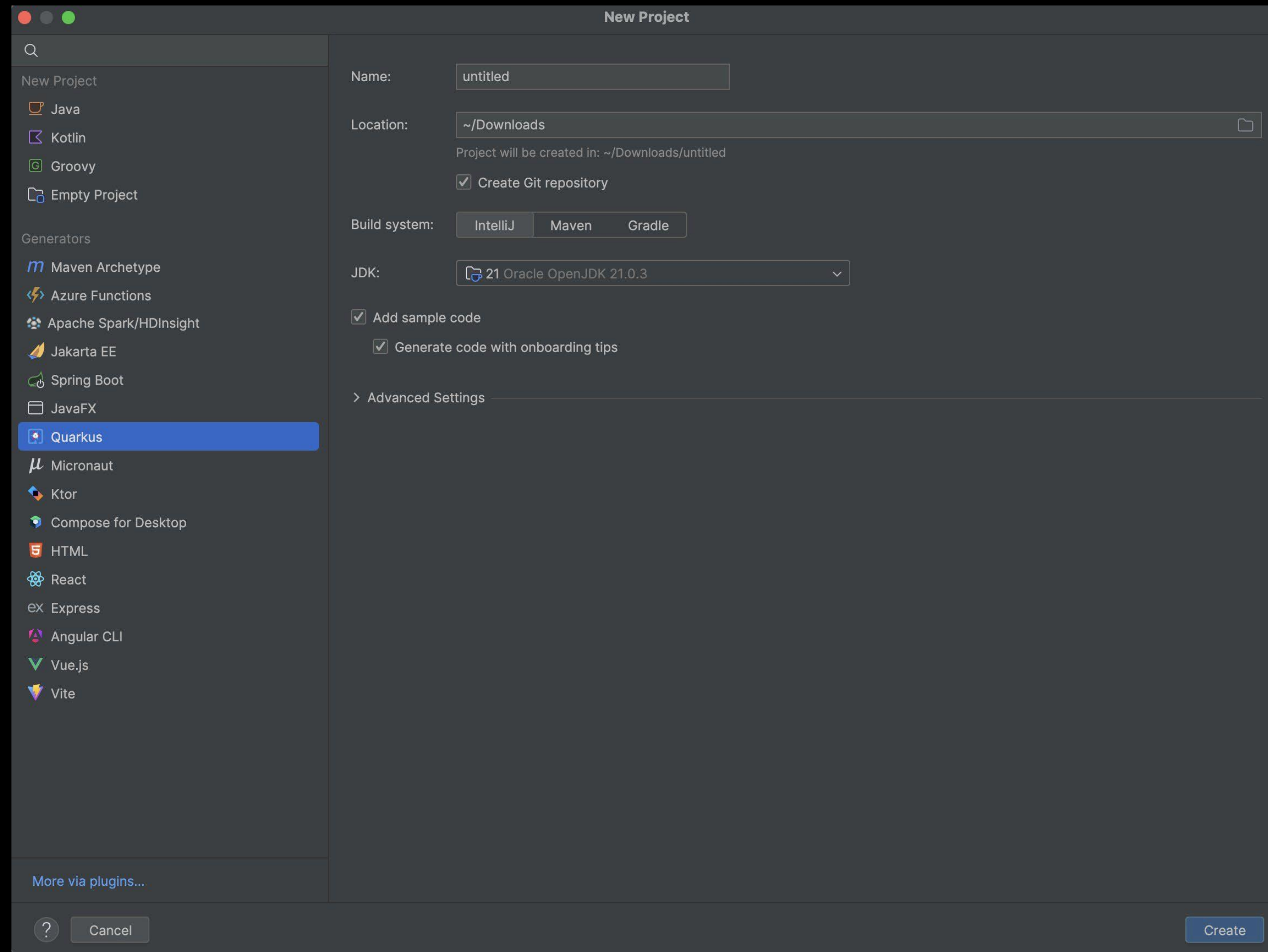
Web app with NPM UI



Web app with ServerSide
Rendering



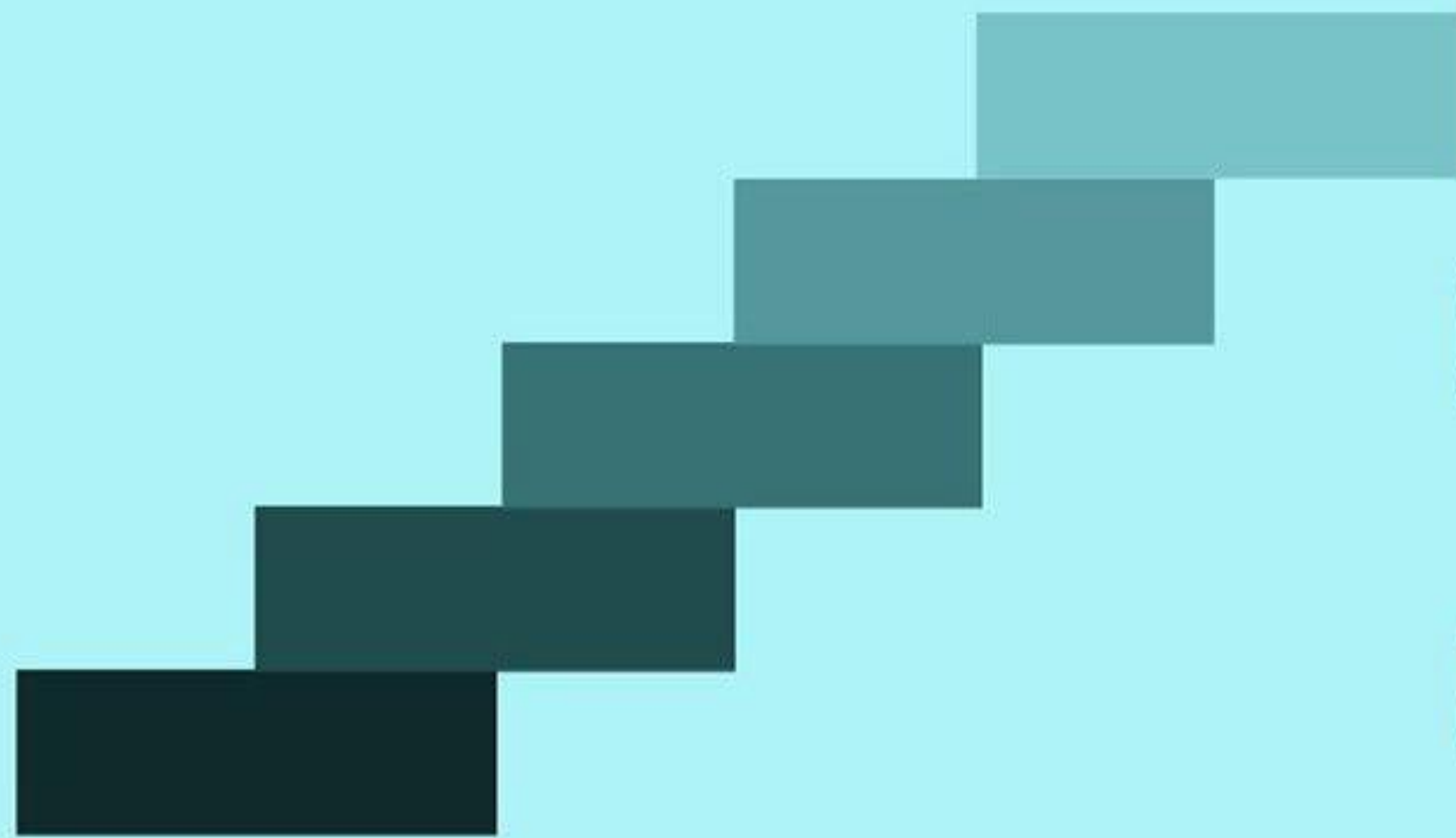
AI Infused service



<https://joedayz.pe/jetbrains>

Recursos

1. Adoptium.net: <https://adoptium.net/temurin/releases/>
2. Maven: <https://maven.apache.org/>
3. GraalVM: <https://www.graalvm.org/>
4. Podman: <https://podman.io>
5. VSCodium: <https://vscodium.com/>
6. Iniciar con Quarkus: <https://code.quarkus.io/>



NEXT
STEPS

Gracias

www.joedayz.pe