

---

# Novedades de la

# DEVNEXUS<sup>TM</sup>

Join the `<dev/>`olution

---

# Agenda

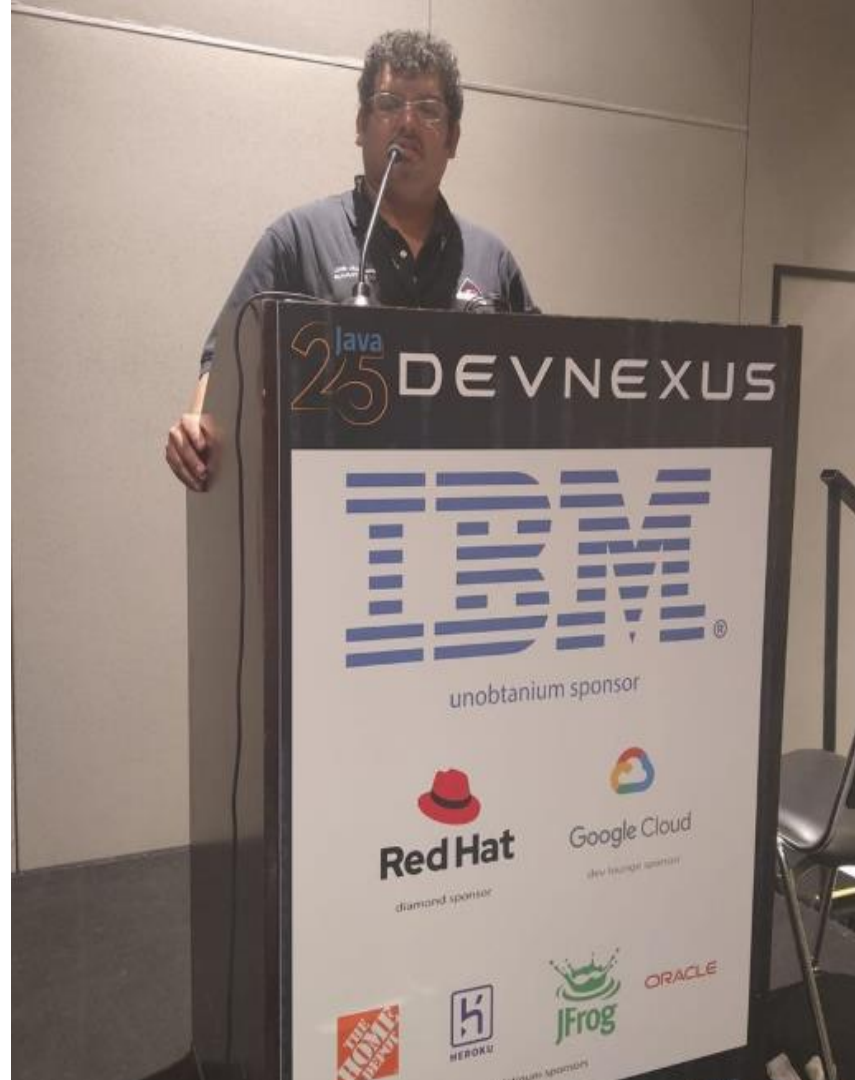
- Apache TomEE - TomiTribe
- Programacion Reactiva
- RSocket con Spring
- Quarkus
- Eclipse che
- OpenJDK y sus empresas soporte



# Presentación

Luis Rolando Ventocilla Santibañez

- Owner de JavaBsAs
- Desarrollador Java Senior en Red Link



# Apache TomEE

## MicroProfile and Jakarta EE on Tomcat



# Apache TomEE

El proyecto OpenEJB fue iniciado por Richard Monson-Haefel y **David Blevins** en 1999 como una implementación de código abierto de la especificación Enterprise JavaBeans. Blevins continuó desarrollando OpenEJB e integrando componentes de este proyecto con Apache Geronimo. En 2003, el componente OpenEJB se convirtió en un proyecto que opera bajo los auspicios de la Apache Software Foundation, en cuyo momento se reescribió con el objetivo de aprovechar Tomcat como un contenedor web incorporado. Se lanzó una versión beta de TomEE en octubre de 2011, y la primera versión lista para producción se envió en 2012.

Dos años después del anuncio de Apache TomEE en JavaOne 2011, varios creadores de Apache TomEE se unieron para formar Tomitribe, una compañía de soporte comercial dedicada a la comunidad de Apache TomEE y enfocada en promover valores de código abierto



# Características Apache TomEE

Basicamente ellos lo que ponen standard como bandera



# ¿Pero que es Tomcat + JAVAEE?



- Tomcat

## All Apache Components

- OpenJPA
- MyFaces
- OpenWebBans
- CXF and ActiveMQ

## Core Values

- Be Small
- Be Certified
- Be Tomcat

# ¿Pero que novedades en 2020?

Hoy hay algunas versiones:

1. TomEE plume
2. TomEE plus
3. TomEE webprofile
4. TomEE microprofile
5. OpenEJB Standalone





# Versiones de Apache TomEE

	Tomcat	TomEE	TomEE JAX-RS (~ Microprofile)	TomEE+	TomEE PluME	OpenEJB
Java Servlets	✓	✓	✓	✓	✓	
Java ServerPages (JSP)	✓	✓	✓	✓	✓	
Java ServerFaces (JSF)		✓	✓	✓	✓	
Java Transaction API (JTA)		✓	✓	✓	✓	✓
Java Persistence API (JPA)		✓	✓	✓	✓	✓
Java Contexts and Dependency Injection (CDI)		✓	✓	✓	✓	✓
Java Authentication and Authorization Service (JAAS)		✓	✓	✓	✓	✓
Java Authorization Contract for Containers (JACC)		✓	✓	✓	✓	✓
JavaMail API		✓	✓	✓	✓	✓
Bean Validation		✓	✓	✓	✓	✓
Enterprise JavaBeans		✓	✓	✓	✓	✓
Java API for RESTful Web Services (JAX-RS)			✓	✓	✓	✓
Java API for XML Web Services (JAX-WS)				✓	✓	✓
Java EE Connector Architecture				✓	✓	✓
Java Messaging Service (JMS)				✓	✓	✓
EclipseLink					✓	
Mojarra					✓	



# Programacion Reactiva

Un paradigma para operar y manejar streams de datos asíncronos; quizás sea esta la definición más extendida sobre lo que es programación reactiva.

Conceptos a destacar:

- Paradigma: define un marco de trabajo o forma de programar.
- Streams: flujo de datos.
- Datos asíncronos: no sabemos cuándo se producirán o cuándo llegarán a nuestro sistema



# Vamos a ver

- MicroProfile y Microservicios Reactivos
- RSocket con Spring
- Quarkus



# Emily Jiang



**Presento en la DevNexus su charla Reactive Microservice In Action**

<https://www.slideshare.net/EmilyJiang3/reactive-microserviceinactiondevnexus-228748403>



# Eclipse Microprofile



En su charla Emily Jiang nos presente como con MicroProfile nos puede ayudar antes ciertos escenarios cuando interactuamos dos microservicios con llamadas sincronicas y asincronicas

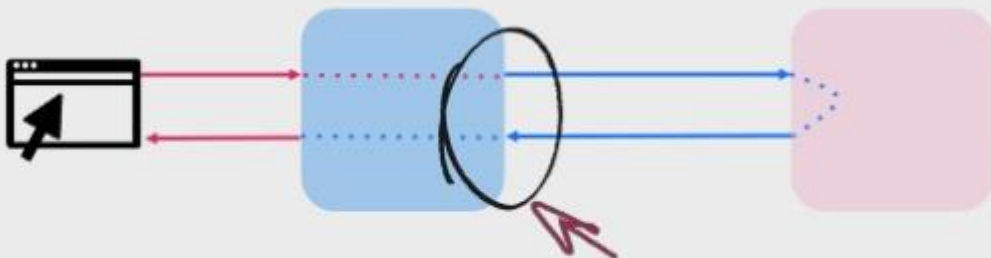
Two microservices...



# Eclipse MicroProfile

Ella realiza en vez de una llamada sincrónica, una llamada asíncrona

Asynchronous calls (because it's trendy)



```
CompletionStage<String> cs =  
    service.doSomething().thenApply(result -> {  
        //next instruction  
    })
```

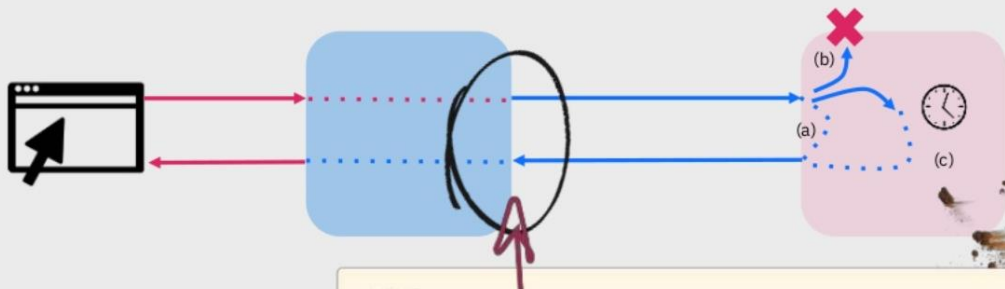


# Eclipse MicroProfile

Al hacer eso se encontró con diversos problemas:

- Demora mucho tiempo en ello → MicroProfile Tolerance

## MicroProfile Fault-Tolerance



```
1 @GET
2 @Path("/test/{param}")
3 @Asynchronous
4 @Fallback(fallbackMethod="myFallback")
5 public CompletionStage<String> invoke(@PathParam("param") String param) {
6     return service.doSomething(param);
7 }
8
9 public CompletionStage<String> myFallback(@PathParam("param") String param) {
10     return CompletableFuture.completedFuture("d'oh!");
11 }
```

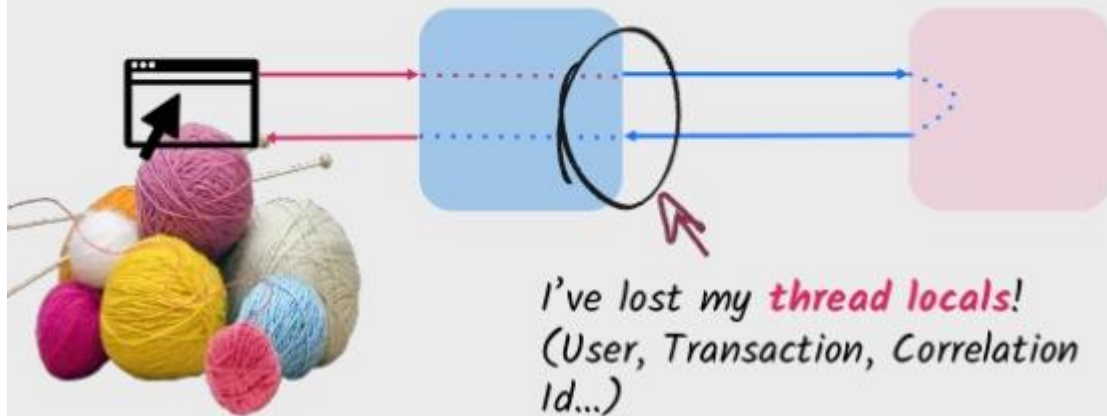


# Eclipse MicroProfile

En el caso cuando llamamos sincronicamente a un microservicio mediante threads

- Pierdes tus thread locales

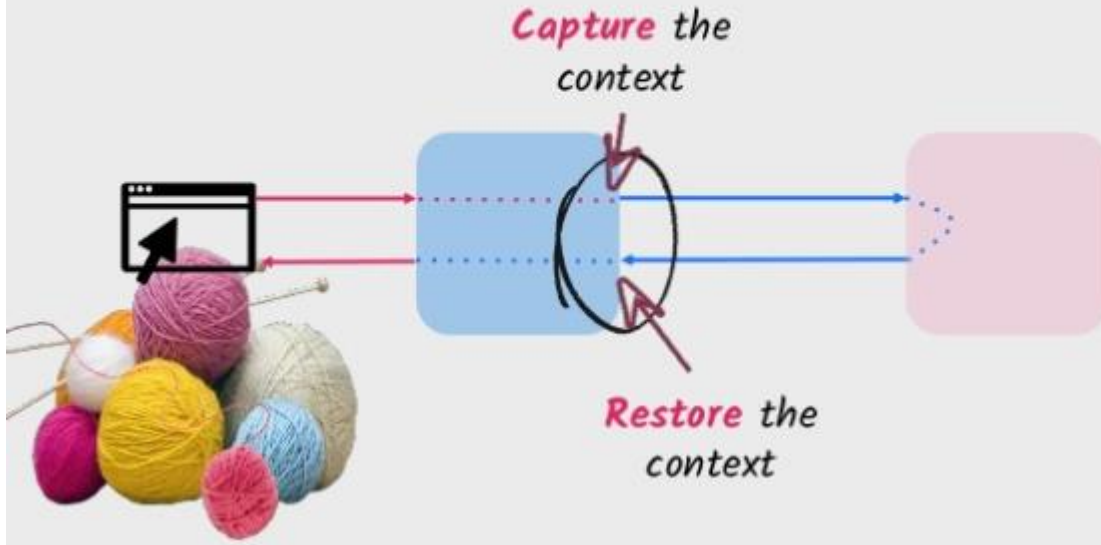
Sorry, I'm out of context...





# Eclipse Microprofile

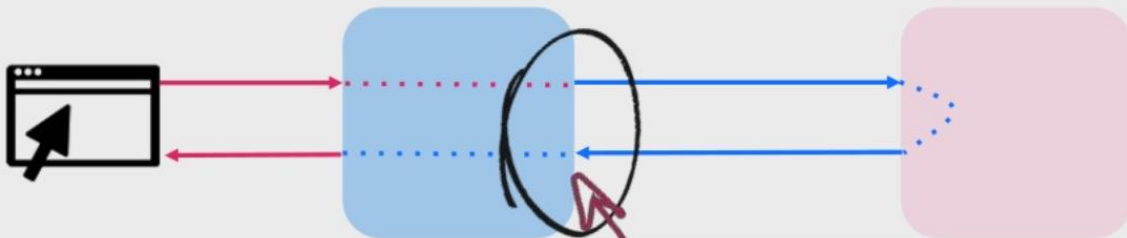
## MicroProfile Context Propagation



# Eclipse MicroProfile

## MicroProfile Context Propagation

Clip slide



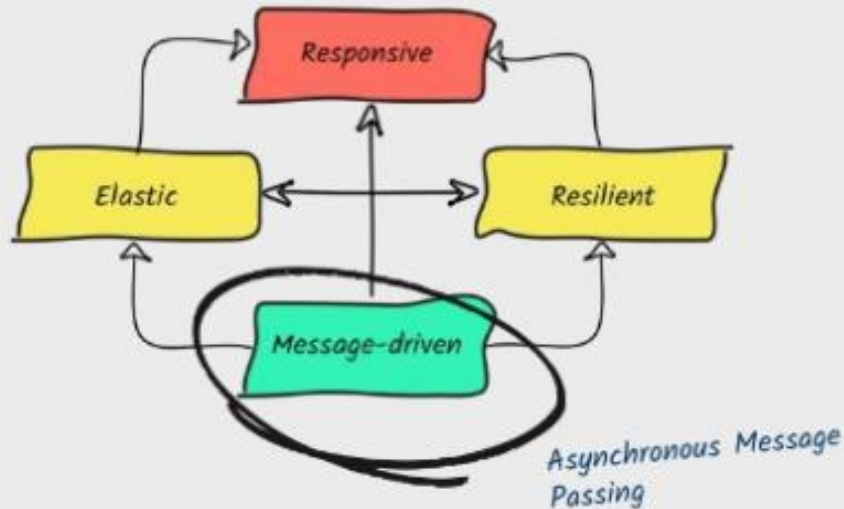
```
1 @Inject
2 ThreadContext threadContext;
3
4 @GET
5 @Path("/test/{param}")
6 public CompletionStage<Response> invoke(@PathParam("param") String param,
7     @Context UriInfo uriInfo) {
8
9     return threadContext.withContextCapture(service.doSomething())
10        .thenApply(body ->
11            ok(body)
12                .header("X-Served-From", uriInfo.getAbsolutePath().toString())
13                .build());
14
15 }
```



# Reactive Microservices

## Reactive Systems

<http://www.reactivemanifesto.org/>



# Reactive Microservices

Para que la interacción entre dos microservicios sea reactiva tiene que cumplir el manifiesto con estas 4 premisas:

- **Responsivos:** El [sistema](#) responde a tiempo en la medida de lo posible.
- **Resilientes:** El sistema permanece responsivo frente a [fallos](#).
- **Elásticos:** El sistema se mantiene responsivo bajo variaciones en la carga de trabajo.
- **Orientados a Mensajes:** Los Sistemas Reactivos confían en el [intercambio de mensajes asíncrono](#) para establecer fronteras entre componentes, lo que asegura bajo acoplamiento, aislamiento y [transparencia de ubicación](#).

Si se cumple estas premisas aseguras que tu sistema sea Reactivo.



# RSocket con Spring

RSocket es un binario protocolo para transportar flujo de datos como:

- request / response (stream 1)
- request / stream (finite stream of many)
- fire and forget (no response)
- channel (bi directional channel)

Fuente: <https://www.baeldung.com/spring-boot-rsocket>





# QUARKUS

- El nombre de dicho framework es **Quarkus**, que llega junto con **Supersonic Subatomic Java**. [Quarkus es un framework](#) nativo de Java para Kubernetes diseñado para GraalVM y HotSpot, creado a partir de las mejores librerías y estándares Java del mercado.

Entre **las cualidades** que ofrece Quarkus (pruebas basadas en plataformas con Red Hat):

- Arranque rápido**, en algunas decenas de milisegundos, que permite el escalado automático de microservicios en contenedores y Kubernetes, así como la ejecución inmediata de FaaS.
- La **utilización mínima de memoria** ayuda a optimizar la densidad de contenedores en despliegues de arquitectura de microservicios que quieren múltiples contenedores.
- Menor tamaño de aplicación de del contendores.**





# QUARKUS

- Dota de un modelo **reactivo e imperativo** unificado para que los desarrolladores Java se sientan familiarizados.
- Los desarrolladores podrán disfrutar de **configuración unificada** en un solo fichero de propiedades, cero configuraciones, recarga en directo en un abrir y cerrar de ojos, código simplificado para el 80% de los usos comunes y flexible para el 20%, sin generación de ejecutables nativos molestos.
- Contarás con las **mejores bibliotecas y estándares**.
- **Soluciones eficaces** para ejecutar Java en microservicios, serverless, nube, contenedores, Kubernetes, FaaS, etc.





La Fundación Eclipse cree que en el sector de los entornos de desarrollo aún hay sitio para la innovación, y lo quieren demostrar con el que han calificado como su IDE de nueva generación. Se trata de [Eclipse Che](#), una herramienta multiplataforma que **se basa en la nube** para que los desarrolladores puedan trabajar de manera local o remota.







# Eclipse Che

Eclipse Che

Dashboard

Get Started

Workspaces (1)

Stacks

Factories

Administration

RECENT WORKSPACES

Create Workspace









java-gradle-1bw94

Luis Ventocilla

Getting Started with Eclipse Che Hosted by Red Hat

Create a Custom Workspace

CREATE & OPEN

 <b>C/C++</b> Stack with C/C++ and Clang 8	 <b>.NET Core</b> Stack with .Net 2.2	 <b>ASP.NET Core Web Application</b> Stack for developing ASP.NET Core Web Application	 <b>Go</b> Stack with Go 1.12.10
 <b>Java Gradle</b> Java Stack with OpenJDK 11 and Gradle 5.2.1	 <b>Java Maven</b> Java Stack with OpenJDK 11 and Maven 3.6.0	 <b>Java with Spring Boot and MongoDB</b> Java stack with OpenJDK 8, MongoDB and Spring Boot Guestbook demo application	 <b>Java with Spring Boot and MySQL</b> Java stack with OpenJDK 8, MySQL and Spring Boot Petclinic demo application





# Eclipse Che

Eclipse Che

Dashboard

Get Started

Workspaces (1)

Stacks

Factories

Administration

RECENT WORKSPACES

Create Workspace

java-gradle-1bw94

Loading...

```
pulling image "quay.io/eclipse/che-plugin-metadata-broker:v3.1.1"
Successfully pulled image "quay.io/eclipse/che-plugin-metadata-broker:v3.1.1"
Created container
Started container
Starting plugin metadata broker
List of plugins and editors to install
- redhat/java11/latest - Java Linting, Intellisense, formatting, refactoring, Maven/Gradle support and more...
- redhat-developer/che-workspace-telemetry-woopra-backend/0.0.1 - Telemetry plugin to send information to Woopra
- eclipse/che-machine-exec-plugin/7.9.1 - Che Plug-in with che-machine-exec service to provide creation terminal or tasks for Eclipse CHE workspace containers.
- eclipse/che-theia/7.9.1 - Eclipse Theia
All plugin metadata has been successfully processed
pulling image "quay.io/eclipse/che-theia-endpoint-runtime-binary:7.9.1"
Successfully pulled image "quay.io/eclipse/che-theia-endpoint-runtime-binary:7.9.1"
Created container
```





# Eclipse Che

The screenshot displays the Eclipse Che IDE interface. On the left is the Explorer view showing a project structure with folders like 'theia', 'console-java-simple', 'gradle', 'settings', 'bin', 'src', 'main', 'java', 'org', 'eclipse', 'che', 'examples', 'test', and files like '.classpath', '.codenvy.json', '.gitignore', '.project', 'build.gradle', 'LICENSE', 'pom.xml', and 'README.md'. The main editor shows a file named 'HelloWorld.java' with the following content:

```
1  /*
2   * Copyright (c) Red Hat, Inc. All rights reserved.
3   * Licensed under the MIT License. See LICENSE in the project root for license information.
4   */
5  package org.eclipse.che.examples;
6
7  public class HelloWorld {
8      Run | Debug
9      public static void main(String... args) {
10         String a = "Che";
11         System.out.println("Hello World " + a + "!");
12     }
13 }
```

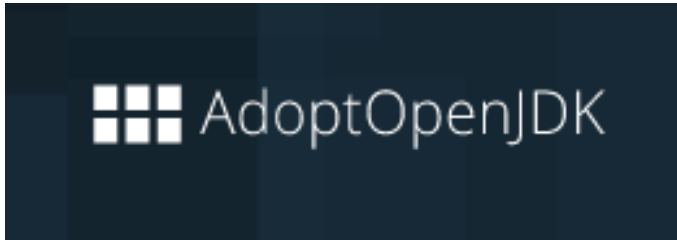
Below the editor is the 'Problems' and 'Java Process Console' view. The console output shows the command executed and the result:

```
/projects $ /usr/lib/jvm/default-jvm/bin/java -Dfile.encoding=UTF-8 @/tmp/cp_bdtv0hf5odvs9kczv2w6o289o.argfile org.eclipse.che.ex
amples.HelloWorld
Hello World Che!
/projects $
```



# Open JDK y sus amigos

## OpenJDK



# Muchas Gracias!!!

