
quarkus
supersonic subatomic java

Agenda

- Que es Quarkus
- Primeros pasos
- Rest APIs
- Datos & ORM
- MicroProfile specs

caracteristicas

uso optimo de memoria

Plataforma optimizada para el desarrollo de aplicaciones java

300x

300 veces mas rápido que un microservicio normal

cambios en tiempo de ejecución

Optimización de tiempos de desarrollo

rápida inicialización

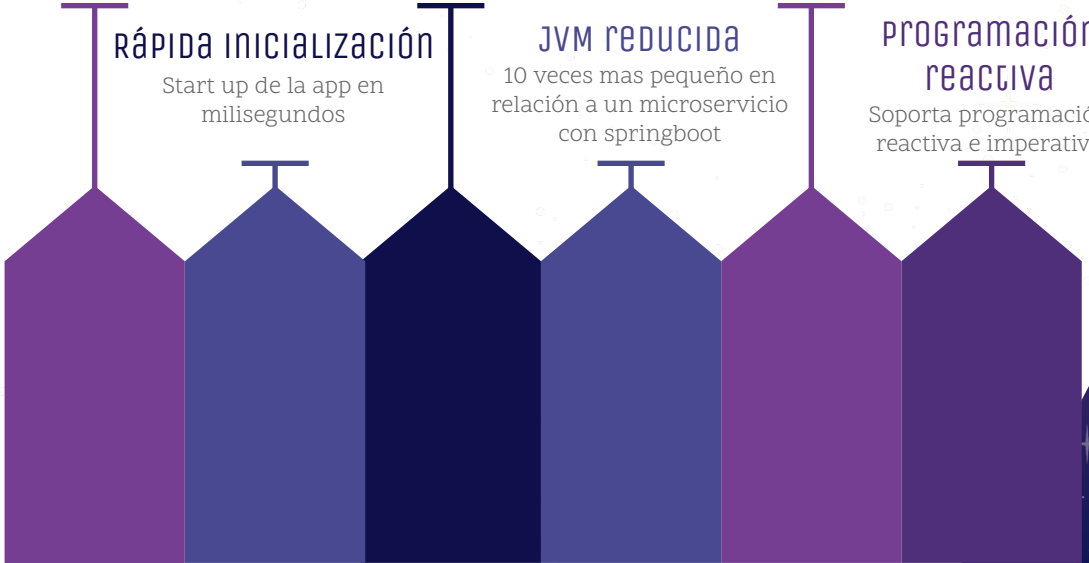
Start up de la app en milisegundos

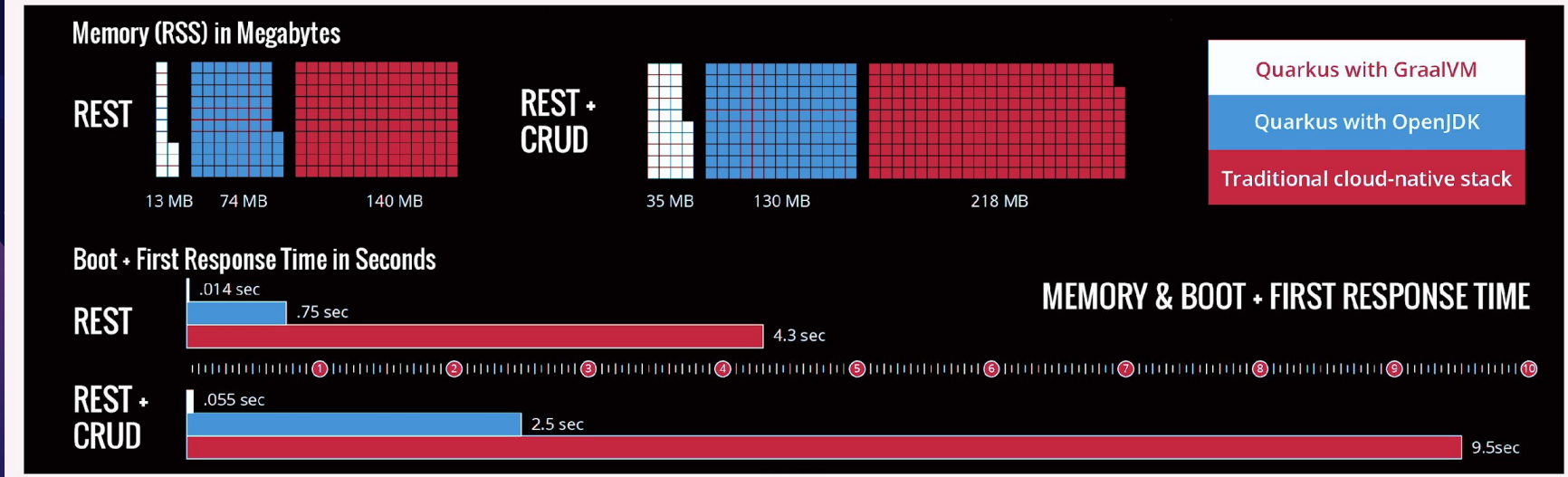
jvm reducida

10 veces mas pequeño en relación a un microservicio con springboot

programación reactiva

Soporta programación reactiva e imperativa




Memory (RSS) in Megabytes
REST
13 MB
74 MB
140 MB
REST + CRUD
35 MB
130 MB
218 MB
Quarkus with GraalVM
Quarkus with OpenJDK
Traditional cloud-native stack
Boot + First Response Time in Seconds
MEMORY & BOOT + FIRST RESPONSE TIME
REST
.014 sec
.75 sec
4.3 sec
REST + CRUD
.055 sec
2.5 sec
9.5sec

Manos a la obra !

primero proyecto

Cuenta con soporte por maven para facilitar la creación y mantenimiento de proyectos quarkus

```
--creación del proyecto

mvn io.quarkus:quarkus-maven-plugin:1.4.2.Final:create
-DprojectGroupId=com.example -DprojectArtifactId=quarkus-example
-DprojectVersion=1.0 -DclassName="com.jugbsas.controller.GreetingResource"

--compilación y ejecución
mvn compile quarkus:dev
```

Extensiones

Librarías optimizadas para performar según estándares de quarkus

```
mvn quarkus:list-extensions

[INFO] --- quarkus-maven-plugin:1.4.2.Final:list-extensions (default-cli)
Current Quarkus extensions available:
Debezium Quarkus Outbox                                 debezium-quarkus-outbox
Quarkus - Core                                          quarkus-core
JAXB                                                    quarkus-jaxb
quarkus-mongodb-panache
```

REST API

Por defecto el proyecto usa rest-easy como proveedor de apis rest.

```
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";
    }
}
```

REST API - POST

Para pasar una entidad JSON debemos agregar un plugin de parseo

```
mvn quarkus:add-extension
-Dextensions="quarkus-resteasy-jsonb,quarkus-hibernate-validator"
```

```
@POST
@Path("/meetup")
@Consumes(MediaType.APPLICATION_JSON)
public Response meetup(@Valid Meetup meetup) {
 logger.info(meetup);
 return Response.ok().build();
}
```

Datos & ORM

Proporciona un framework llamado **Panache** basado en hibernate que se basa en patrones de persistencia abstrayendo lógica común de manejo de datos:

- Active record
- DAO

Datos & ORM

Tenemos que definir el uso de un jdbc de acuerdo a la BBDD y la extensión de panache

```
mvn quarkus:add-extension
-Dextensions="quarkus-jdbc-h2,quarkus-hibernate-orm-panache"
```

Configuración de application properties

```
quarkus.datasource.db-kind=h2
quarkus.datasource.jdbc.url =jdbc:h2:mem:meetup
quarkus.datasource.username=jugbsas
quarkus.datasource.password=
quarkus.hibernate-orm.dialect = org.hibernate.dialect.H2Dialect
quarkus.hibernate-orm.database.generation=drop-and-create
quarkus.hibernate-orm.log.sql=true
```

Panache DAO

Implementando una interfaz del framework se puede implementar un DAO sencillamente

```
@ApplicationScoped
public class MeetupDAO implements PanacheRepository<Meetup> {

    @Transactional
    public void create(Meetup meetup) {
       persist(meetup);
    }

    @Transactional
    public List<Meetup> find(String name) {
       return findAll().list();
    }

}
```

Panache Active record

Podemos tener un objeto que persista su estado

```
@Entity(name = "MEETUP")
public class Meetup extends PanacheEntity implements Serializable {

@Id
private Long id;

}
```

Panache

Ejemplo de uso en un servicio rest

```
@POST
@Path("/meetup")
@Consumes(MediaType.APPLICATION_JSON)
public Response meetup(@Valid Meetup meetup){
  logger.info(meetup);
  //meetupDAO.create(meetup);
  meetup.persist();
  return Response.ok().build();
}
}
```

```
@GET
@Path("/meetup")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public List<Meetup> meetup() {
    return Meetup.findAll().list();
    //return meetupDAO.find();
}}
```

Inyección de dependencias

Implementa la inyección de dependencias según el standard de CDI.

```
@ApplicationScopped
@RequestScoped
@Inject

@ConfigProperty(name = "greetings.message", defaultValue = "no greeting").
```

Como lo hace?

Procesamiento de metadatos en tiempo de compilación

Facilita el start up del proyecto y las deducciones en tiempo de ejecución

Bajo uso del proceso de reflexión

Optimización del running del programa

GraalVM

Totalmente compatible con la nueva VM nativa

THanKS!