

Persistencia de Datos con Panache

Configurando persistencia

```
<dependencies>
  <!-- Hibernate ORM dependency -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-hibernate-orm</artifactId>
  </dependency>
</dependencies>
```

```
<dependencies>
  <!-- Panache dependency -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-hibernate-orm-panache</artifactId>
  </dependency>
</dependencies>
```

Agregando el driver

```
<dependencies>
  <!-- JDBC driver dependency -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-jdbc-postgresql</artifactId>
  </dependency>
</dependencies>
```

Configurando la base de datos

```
# datasource configuration
quarkus.datasource.username = myusername ❶
quarkus.datasource.password = mypassword ❷
quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/mydatabase ❸

# Optional configuration
quarkus.hibernate-orm.sql-load-script = META-INF/import-dev.sql ❹
quarkus.hibernate-orm.database.generation = drop-and-create ❺
```

Usando profiles: **dev** y **prod**

```
# production datasource configuration
%prod.quarkus.datasource.username = myusername
%prod.quarkus.datasource.password = mypassword
%prod.quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/mydatabase
%prod.quarkus.hibernate-orm.sql-load-script = no-file ❶
```

```
# development datasource configuration
%dev.quarkus.hibernate-orm.sql-load-script = META-INF/import-dev.sql
%dev.quarkus.hibernate-orm.database.generation = drop-and-create
%dev.quarkus.datasource.devservices.image-name = quay.io/example/postgres:14.1 ❷
```

Persistencia con Hibernate ORM

```
@ApplicationScoped
public class EmployeeService {
    @Inject
    EntityManager em; ❶

    @Transactional ❷
    public void createEmployee( String name ) {
        Employee employee = new Employee();
        employee.setName( name );
        em.persist( employee ); ❸
    }
}
```

Default:
REQUIRED

REQUIRES_NEW

MANDATORY

Persistencia con Hibernate ORM

```
@Transactional( TxType.REQUIRES_NEW )  
public void createEmployee( String name ) {  
    Employee employee = new Employee();  
    employee.setName( name );  
    em.persist( employee );  
}
```

Simplificando la persistencia con Panache

Patrón Repository

```
@Entity
public class Expense {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private BigDecimal amount;
    private String description;
    private LocalDateTime creationDate;

    // Getters and Setters
}
```

```
@ApplicationScoped
public class ExpenseRepository implements PanacheRepository<Expense> {
}
```

```
@Inject
ExpenseRepository expenseRepository;

Expense expense = new Expense();
expense.setName( "Hotel stay" );
expense.setAmount( 100 );
expense.setDescription( "Conference travel" );
expense.setCreationDate( LocalDateTime.now() );

expenseRepository.persist( expense );
```


Simplificando la persistencia con Panache

Active Record Pattern

```
@Entity
public class Expense extends PanacheEntity {
    public String name;
    public BigDecimal amount;
    public String description;
    public LocalDateTime creationDate;
}
```

```
Expense expense = new Expense();
expense.name = "Hotel stay";
expense.amount = 100;
expense.description "Conference travel";
expense.creationDate = LocalDateTime.now();

expense.persist();
```

```
@Entity
public class Expense extends PanacheEntity {
    public String name;
    public BigDecimal amount;
    public String description;
    public LocalDateTime creationDate;

    public static List<Expense> findCurrent(){
        return list( "creationDate", LocalDateTime.now() );
    }
}
```

Métodos provistos por Panache

```
// Persist the entity
instance.persist();

// check if the entity is persistent
instance.isPersistent()

// get a list of all entities
List<Entity> allEntitys = Entity.listAll();

// find a specific entity by ID
instance = Entity.findById( entityId );

// find a specific instance by ID via an Optional
Optional<Entity> optional = Entity.findByIdOptional( entityId );

// find all alive entities
List<Entity> aliveInstances = Entity.list( "alive", true );

// count all entities
long countAll = Entity.count();

// count all alive entities
long countAlive = Entity.count( "alive", true );

// delete all alive entities
Entity.delete( "alive", true );

// delete all entities
Entity.deleteAll();

// delete by id
boolean deleted = Entity.deleteById( entityId );

// set all alive entities as not alive
Entity.update( "alive = false where alive = ?1", true );
```

Paging y Sorting Query Results

Panache Paging Functions

Function	Usage
<code>page()</code>	Returns the current page.
<code>page(Page page)</code>	Sets and returns the specified page.
<code>page(int pageIndex, int pageSize)</code>	Shorthand for the previous method.
<code>pageCount()</code>	Returns the number of pages for the current page size.
<code>range(int startIndex, int lastIndex)</code>	Retrieves the results between <code>startIndex</code> and <code>lastIndex</code> .

```
@GET
public List<Example> list() {
    PanacheQuery<Example> exampleQuery = Example.findAll();
    return exampleQuery.page(Page.of(0, 10)).list();
}
```

Paging y Sorting Query Results

Panache Sorting Functions	
Function	Usage
by(String column)	Sorts by the given column in ascending order.
Function	Usage
by(String column, Direction direction)	Sorts by the column using the specified direction (Ascending or Descending).
by(String... columns)	Sorts by the given columns in the given order.
and(String column)	Adds an additional sorting column in ascending order.
and(String column, Direction direction)	Adds an additional sorting column with the given direction.
empty()	Returns an empty sorting instance.
ascending()	Sets the ascending order for the current sort columns.
ascending(String... columns)	Sets the columns to sort in ascending order.
descending()	Sets the descending order for the current sort columns.
descending(String... columns)	Sets the columns to sort in descending order.

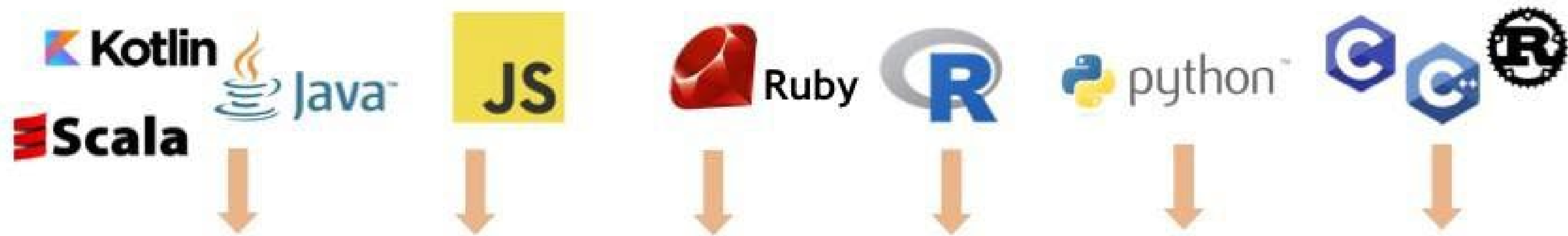
```
@GET
public List<Example> list() {
    return Example.findAll( Sort.by("sample") ).list();
}
```

Recursos

1. https://quarkus.io/version/2.13/guides/hibernate-orm#quarkus-hibernateorm_configuration
2. https://en.wikipedia.org/wiki/Jakarta_Persistence

Demo

Construyendo aplicaciones
nativas con Quarkus y GraalVM



Automatic transformation of interpreters to compiler

GraalVM™

Embeddable in native or managed applications



graalvm/mandrel



Mandrel is a downstream distribution of the GraalVM community edition. Mandrel's main goal is to provide a native-image release specifically...



0

Contributors



12

Issues



3

Discussions



378

Stars



15

Forks



mvn package -Pnative

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

```
quarkus.native.container-build=true
```

application.properties

```
quarkus.native.container-runtime=podman ❶  
quarkus.native.builder-image=registry.access.redhat.com/quarkus/mandrel-21-jdk17-  
rhel8:21.3 ❷
```

Limitaciones de aplicaciones nativas

- Inyectar recursos en tests de ejecutables nativos
- Registrarse para reflection (`@RegisterForReflection`)
- Modificadores de acceso privados
- Reflection en librerías de terceros

No podemos usar `@RegisterForReflection` con librerías de terceros que no son extensiones Quarkus. Más bien tenemos que crear este archivo en `src/main/resources/reflection-config.json`

```
[
  {
    "name" : "com.redhat.library.MyThirdPartyClass",
    "allDeclaredConstructors" : true,
    "allPublicConstructors" : true,
    "allDeclaredMethods" : true,
    "allPublicMethods" : true,
    "allDeclaredFields" : true,
    "allPublicFields" : true
  }
]
```

Y luego agregarlo en `application.properties`:

```
quarkus.native.additional-build-args =-H:ReflectionConfigurationFiles=reflection-  
config.json
```

Casos de uso para aplicaciones nativas

- Las aplicaciones nativas son ideales para entornos donde hay poca memoria, se necesita un tiempo de inicio rápido y performance inicial de CPU es clave.
 - Arquitecturas serverless
 - Aplicaciones con requerimientos de alta densidad de memoria
 - Despliegue en plataformas de orquestación
 - Aplicaciones de linea de comandos

Desventajas de aplicaciones nativas

- No Just-in-time (JIT) compilation
- Limitaciones en el código
- Lento build time
- Alto consumo de memoria en el proceso de build time

```
$ mvn package -Pnative -Dquarkus.native.native-image-xmx=6G
```

Desventajas de aplicaciones nativas

- Profiling y debugging no trabaja con Java tools
- El tamaño del ejecutable nativo es grande

Recurso s

1. <https://quarkus.io/version/2.13/guides/building-native-image>

Ejercicio: Construir aplicaciones
nativas con Quarkus y GraalVM

Laboratorio: Desarrollando cloud-native micro services con Quarkus

Gracias

www.joedayz.pe