

What is Quarkus?

Quarkus (<https://quarkus.io/>) is a Kubernetes Native Java stack tailored for GraalVM & OpenJDK HotSpot, crafted from the best of breed Java libraries and standards. Also focused on developer experience, making things just work with little to no configuration and allowing to do live coding.

Getting Started

Quarkus comes with a Maven archetype to scaffold a very simple starting project.

```
mvn io.quarkus:quarkus-maven-plugin:0.13.3:create \
  -DprojectId=org.acme \
  -DprojectArtifactId=getting-started \
  -DclassName="org.acme.quickstart.GreetingResource" \
  -Dpath="/hello"
```

This creates a simple JAX-RS resource called `GreetingResource`.

```
@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}
```

Extensions

Quarkus comes with extensions to integrate with some libraries such as JSON-B, Camel or MicroProfile libraries. To list all available extensions just run:

```
./mvnw quarkus:list-extensions
```

And to register the extensions into build tool:

```
./mvnw quarkus:add-extension -Dextensions=""
```

Adding Configuration Parameters

To add configuration to your application, Quarkus relies on MicroProfile Config spec (<https://github.com/eclipse/microprofile-config>).

```
@ConfigProperty(name = "greetings.message")
String message;
```

```
@ConfigProperty(name = "greetings.message",
                 defaultValue = "Hello")
String messageWithDefault;
```

```
@ConfigProperty(name = "greetings.message")
Optional<String> optionalMessage;
```

Properties can be set as environment variable, system property or in `src/main/resources/application.properties`.

```
greetings.message = Hello World
```

Injection

Quarkus is based on CDI to implement injection of code.

```
@ApplicationScoped
public class GreetingService {

    public String message(String message) {
        return message.toUpperCase();
    }
}
```

Scope annotation is mandatory to make the bean discoverable by CDI.

```
@Inject
GreetingService greetingService;
```

Taule CDI?

JSON Marshalling/Unmarshalling

To work with JSON-B you need to add a dependency:

```
./mvnw quarkus:add-extension
-Dextensions="io.quarkus:quarkus-resteasy-jsonb"
```

Any POJO is marshalled/unmarshalled automatically.

```
public class Sauce {
    private String name;
    private long scovilleHeatUnits;

    // getter/setters
}
```

JSON equivalent:

```
{
    "name": "Blair's Ultra Death",
    "scovilleHeatUnits": 1100000
}
```

In a POST endpoint example:

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public Response create(Sauce sauce) {
    // Create Sauce
    return Response.created(URI.create(sauce.getId()))
        .build();
}
```

Persistence

Quarkus works with JPA(Hibernate) as persistence solution. But also provides an Active Record pattern (https://en.wikipedia.org/wiki/Active_record_pattern) implementation under Panache project.

To use database access you also need to add Quarkus JDBC drivers instead of the original ones. At this time H2, MariaDB and PostgreSQL drivers are supported.

```
./mvnw quarkus:add-extension
-Dextensions="io.quarkus:quarkus-hibernate-orm-panache,
io.quarkus:quarkus-jdbc-mariadb"
```

```
@Entity
public class Developer extends PanacheEntity {

    // id field is implicit

    public String name;
}
```

And configuration in

src/main/resources/application.properties:

```
quarkus.datasource.url=jdbc:mariadb://localhost:3306/mydb
quarkus.datasource.driver=org.mariadb.jdbc.Driver
quarkus.datasource.username=developer
quarkus.datasource.password=developer
quarkus.hibernate-orm.database.generation=update
```

Database operations:

// Insert
Developer developer = new Developer();
developer.name = "Alex";
developer.persist();

// Find All
Developer.findAll().list();

// Find By Query
Developer.find("name", "Alex").firstResult();

// Delete
Developer developer = new Developer();
developer.id = 1;
developer.delete();

// Delete By Query
long numberOfDeleted = Developer.delete("name", "Alex");

Remember to annotate methods with @Transactional annotation to make changes persisted in database.

Clarification about queries

Static Methods

Field	Parameters	Return
findById	Object	Returns object or null if not found.
find	String, [Object..., Map<String, Object>, Parameters]	Lists of entities meeting given query with parameters set.

Field	Parameters	Return
find	String, Sort, [Object..., Map<String, Object>, Parameters]	Lists of entities meeting given query with parameters set sorted by Sort attribute/s.
findAll		Finds all entities.
findAll	Sort	Finds all entities sorted by Sort attribute/s.
stream	String, [Object..., Map<String, Object>, Parameters]	java.util.stream.Stream of entities meeting given query with parameters set.
stream	String, Sort, [Object..., Map<String, Object>, Parameters]	java.util.stream.Stream of entities meeting given query with parameters set sorted by Sort attribute/s.
streamAll		java.util.stream.Stream of all entities.
streamAll	Sort	java.util.stream.Stream of all entities sorted by Sort attribute/s.
count		`Number of entities.
count	String, [Object..., Map<String, Object>, Parameters]	Number of entities meeting given query with parameters set.
deleteAll		Number of deleted entities.

Field	Parameters	Return
delete	String, [Object..., Map<String, Object>, Parameters]	Number of deleted entities meeting given query with parameters set.
persist	[Iterable, Steram, Object...]	

Rest Client

Quarkus implements MicroProfile Rest Client (https://github.com/eclipse/microprofile-rest-client) spec:

./mvnw quarkus:add-extension
-Dextensions="io.quarkus:quarkus-smallrye-rest-client"

To get content from http://worldclockapi.com/api/json/cet/now you need to create a service interface:

@Path("/api") @RegisterRestClient public interface WorldClockService {


 @GET @Path("/json/cet/now")
 @Produces(javax.ws.rs.core.MediaType.APPLICATION_JSON)
 WorldClock getNow();
}

And configure the hostname at application.properties:

org.acme.quickstart.WorldClockService/mp-rest/url=http://worldclockapi.com

xx

Authors :

@alexsotob

undefined

v0.13.3