Red Hat build of Quarkus 1.11

Configuring data sources in your Quarkus applications

Last Updated: 2021-06-24

Legal Notice

Abstract

Configure a data source for your Quarkus application using extensions for built-in database drivers and connect your application to a relational database.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our technical content and encourage you to tell us what you think. If you'd like to add comments, provide insights, correct a typo, or even ask a question, you can do so directly in the documentation.

NOTE

You must have a Red Hat account and be logged in to the customer portal.

To submit documentation feedback from the customer portal, do the following:

1. Select the **Multi-page HTML** format.
2. Click the **Feedback** button at the top-right of the document.
3. Highlight the section of text where you want to provide feedback.
4. Click the **Add Feedback** dialog next to your highlighted text.
5. Enter your feedback in the text box on the right of the page and then click **Submit**.

We automatically create a tracking issue each time you submit feedback. Open the link that is displayed after you click **Submit** and start watching the issue or add more comments.

Thank you for the valuable feedback.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. INTRODUCTION TO USING DATA SOURCES WITH QUARKUS

When you want to add persistent data storage to your application, you must connect your application to a relational database. To achieve this, you use a data source that you connect to your application using a database driver. You can connect your Quarkus application to one or multiple data sources. You can use the data source management functionalities integrated into Quarkus to:

- configure your application to use one or multiple data sources
- obtain references to data sources in your code
- view and set pool tuning configuration properties

In Quarkus applications, you can use two types of database drivers to connect your application to a relational database. You can use multiple data sources of both types in one application simultaneously:

JDBC drivers

that use the standard JDBC API that provides database connectivity for Java-based application. Quarkus JDBC drivers manage database connections using Agroal, a fast, light-weight, and highly scalable database connection pool implementation that is integrated with other features of Quarkus, including security, transaction management and health checks.

Reactive drivers

that are based on the data source driver implementation in Eclipse Vert.x. Eclipse Vert.x reactive data source drivers provide the non-blocking and reactive network-related features of Quarkus and are suitable for applications that are designed to be highly scalable and event-driven.

You can configure both types of data source drivers using a set of unified and flexible configuration options that Quarkus provides.

1.1. SETTING THE DB-KIND PROPERTY FOR A DATA SOURCE

When you set the **db-kind** property in the configuration file of your application to match the type of your data source that you want to use, Quarkus automatically resolves the appropriate type of database driver. The following procedure demonstrates how you can set the **db-kind** property for a data source.

Prerequisites

- You have a Quarkus Maven project.

Procedure

1. Navigate to your Quarkus project directory.
2. In the **src/main/resources/application.properties** file, set the value of the **db-kind** property to match the type of the data source that you want to use. The following example uses **postgresql** as the data source type:

```
quarkus.datasource.db-kind=postgresql
```

Additional resources

- Driver extensions for Quarkus built-in databases

1.2. SETTING DATABASE CREDENTIALS

You can define credentials that your application uses to access your database. Defining access credentials for your database is optional. You can skip this procedure when configuring a data source for your application.

Prerequisites

- You have a Quarkus Maven project.
- You have set the **db-kind** property for your data source.
- Your Quarkus application is in JVM mode. This prerequisite applies when you use a JDBC driver that Quarkus does not support in native mode.

Procedure

1. Navigate to your Quarkus project directory.
2. In the **src/main/resources/application.properties** file, set the value of the **quarkus.datasource.username** property to match the username that your application uses to access the database:

```
quarkus.datasource.username=<username>
```

3. In the **src/main/resources/application.properties** file, set the value of the **quarkus.datasource.password** property to match the password that your application uses to access the database:

```
quarkus.datasource.password=<password>
```

Additional resources

- Setting the db-kind property for a data source

1.3. QUARKUS DRIVER EXTENSIONS FOR BUILT-IN DATABASES

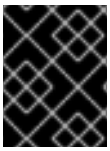
The following table provides an overview of Quarkus built-in databases and the extensions that you can use to connect your application to a relational database:

Table 1.1. Driver extensions for Quarkus built-in databases

Database built-in	db-kind	Agroal extension	Reactive extension
DB2	**db2**	**quarkus-jdbc-db2**	**quarkus-reactive-db2-client**
Derby	**derby**	**quarkus-jdbc-derby**	N/A
H2	**h2**	**quarkus-jdbc-h2**	N/A
MariaDB	**mariadb**	**quarkus-jdbc-mariadb**	**quarkus-reactive-mysql-client**

Database built-in	db-kind	Agroal extension	Reactive extension
Microsoft SQL Server	**mssql**	**quarkus-jdbc-mssql**	N/A
MySQL	**mysql**	**quarkus-jdbc-mysql**	**quarkus-reactive-mysql-client**
PostgreSQL	**postgresql**, **pgsql**, or **pg**	**quarkus-jdbc-postgresql**	**quarkus-reactive-pg-client**

You can configure H2 and Derby databases to run in embedded mode. The H2 and Derby driver extensions do not support compiling the embedded database engine into native executables.



IMPORTANT

This table includes supported and community artifacts. For a list of supported Maven artifacts, see the Red Hat build of Quarkus Component Details page.

When you use one of the built-in database kinds the JDBC driver resolves automatically to the following values:

Table 1.2. JDBC and XA drivers for Quarkus built-in databases

Database	JDBC driver	XA driver
DB2	**com.ibm.db2.jcc.DBDriver**	**com.ibm.db2.jcc.DB2XADataSource**
Derby	**org.apache.derby.jdbc.ClientDriver**	**org.apache.derby.jdbc.ClientXADataSource**
H2	**org.h2.Driver**	**org.h2.jdbcx.JdbcDataSource**
MariaDB	**org.mariadb.jdbc.Driver**	**org.mariadb.jdbc.MySQLDataSource**
Microsoft SQL Server	**com.microsoft.sqlserver.jdbc.SQLServerDriver**	**com.microsoft.sqlserver.jdbc.SQLServerXADataSource**
MySQL	**com.mysql.cj.jdbc.Driver**	**com.mysql.cj.jdbc.MysqlXADataSource**
PostgreSQL	**org.postgresql.Driver**	**org.postgresql.xa.PGXADataSource**

NOTE

You can configure H2 and Derby databases to run in embedded mode. The H2 and Derby driver extensions do not support compiling the embedded database engine into native executables.

Additional resources

- Red Hat build of Quarkus Supported Configurations page
- Red Hat build of Quarkus Component Details page

CHAPTER 2. JDBC DATA SOURCE CONFIGURATION

JDBC is the database connection API most commonly used in Java-based applications. You can use a JDBC data source driver to connect your application to a relational database.

To configure a JDBC data source, you must

- add the **quarkus-agroal** extension to your application
- add a **db-kind** extension to your application
- specify the JDBC URL that your application uses to access the data source

The following example shows how you can connect a **postgresql** data source to your application and specify the access credentials and the JDBC URL for the data source. For more information on how to specify the JDBC URL see Setting the JDBC URL of your data source .

Example of JDBC data source configuration

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=<your_username>
quarkus.datasource.password=<your_password>

quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/hibernate_orm_test
quarkus.datasource.jdbc.max-size=16
```

2.1. INSTALLING QUARKUS EXTENSIONS FOR JDBC DATA SOURCES

You must install the **quarkus-agroal** extension and a Quarkus JDBC database driver extension to configure a JDBC data source. The JDBC database driver that you add must match the type of JDBC database that you want to use.

The following procedure demonstrates how to install Quarkus extensions for JDBC data sources.

Prerequisites

- You have a Quarkus Maven project.
- You have set the **db-kind** property for your data source.

Procedure

1. Navigate to your Quarkus project directory.
2. Add the **quarkus-agroal** extension to your project:

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

3. Add the Quarkus extension for the appropriate type of relational database driver to your application:

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

   For example, to add a PostgreSQL database driver extension, use:

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

NOTE

If you are using Hibernate ORM, you do not need to add the Agroal extension dependency explicitly. Agroal is a transitive dependency of the Hibernate ORM extension. You must use a JDBC data source driver with Hibernate ORM.

Additional resources

- Driver extensions for Quarkus built-in databases
- Setting the db-kind property for a data source

2.2. SETTING THE JDBC URL OF YOUR DATA SOURCE

You must set the JDBC URL to complete the configuration of your JDBC data source. The following procedure demonstrates how to set the JDBC URL.

Prerequisites

- You have a Quarkus Maven project.
- You have added the corresponding data source driver to your application.
- You have set the **db-kind** property for your data source.

Procedure

1. Navigate to your Quarkus project directory.
2. Open the **src/main/resources/application.properties** file.
3. Set the value of the **quarkus.datasource.jdbc.url** property to match the JDBC URL for your data source:

   ```
   quarkus.datasource.jdbc.url=<JDBC_URL>
   ```

   For example, to set the JDBC URL of a PostgreSQL data source, use:

   ```
   quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/hibernate_orm_test
   ```

Additional resources

- Setting the **db-kind** property for a data source
- JDBC database connection URLs for databases with production and development support
- JDBC database connection URLs for databases with development support

2.3. OBTAINING A JDBC DATA SOURCE WITH HIBERNATE ORM

If you are using Hibernate ORM, your application automatically recognizes and connects to an available JDBC data source. To access the data source in your application code, you can obtain it as a CDI bean.

Prerequisites

- You have a Quarkus Maven project.
- You have installed Hibernate ORM.

Procedure

- To access the data source in your application code, add the **@Inject** annotation to the **dataSource** property:

```
import javax.sql.DataSource;

@Inject DataSource dataSource;
```

2.4. DISABLING A JDBC DATA SOURCE IN A SIMULTANEOUS CONFIGURATION

You can simultaneously configure a JDBC driver extension and a reactive driver extension for the same data source in your application. You can disable the JDBC data source driver in a simultaneous configuration, thus forcing your application to use the reactive data source driver.

Prerequisites

- You have a Quarkus Maven project.
- You have a JDBC data source driver and a reactive data source driver configured simultaneously in your application.

Procedure

1. Navigate to your Quarkus project directory.
2. Open the **src/main/resources/application.properties** file.
3. Set the **quarkus.datasource.jdbc** property to **false** to disable the JDBC data source:

```
quarkus.datasource.jdbc=false
```

2.5. CONFIGURING JDBC DRIVERS WITH NO BUILT-IN DRIVER EXTENSION

You can use JDBC drivers and databases that do not have built-in driver extensions. The following procedure demonstrates how to configure a JDBC driver with no built-in extension.

NOTE

You can use any JDBC driver while using your Quarkus application in JVM mode. The non-included JDBC drivers may not function properly when you compile your Quarkus as a native executable.

Prerequisites

- You have a Quarkus Maven project.
- Your Quarkus application is in JVM mode.
- You add the data source driver that you want to use as a dependency in the **pom.xml** file of your project.

Procedure

1. Navigate to your Quarkus project directory.
2. Open the **src/main/resources/application.properties** file.
3. Set the value of the **db-kind** property to **other**:

```
quarkus.datasource.db-kind=other
```

4. Set the value of the **quarkus.datasource.jdbc.driver** property to match the type of driver extension that you want to use:

```
quarkus.datasource.jdbc.driver=<driver>
```

5. Set the value of the **quarkus.datasource.jdbc.url** property to match the JDBC URL for your data source:

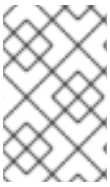
```
quarkus.datasource.jdbc.url=<JDBC_URL>
```

For example, Quarkus does not provide a built-in driver extension for the OpenTracing driver. Ensure that you add the **opentracing-jdbc** artifact to your **pom.xml** file, and set the following properties in your **application.properties** file to configure the OpenTracing driver:

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.jdbc.url=jdbc:tracing:postgresql://localhost:5432/mydatabase
quarkus.datasource.jdbc.driver=io.opentracing.contrib.jdbc.TracingDriver
quarkus.hibernate-orm.dialect=org.hibernate.dialect.PostgreSQLDialect
```

2.6. CONFIGURING MULTIPLE JDBC DATA SOURCES

You can configure multiple JDBC data sources for your Quarkus application with Agroal.



NOTE

When you use the Hibernate ORM extension, you can add multiple persistent data storage units to your application. For each persistence unit, you can point to the data source of your choice.

Prerequisites

- You have a Quarkus Maven project.
- You have multiple JDBC data sources.

Procedure

1. Navigate to your Quarkus project directory.
2. In the **src/main/resources/application.properties** file, set the following properties for each of your data sources:
   The following example shows the complete configuration for 3 JDBC data sources:

```
quarkus.datasource.db-kind=h2 (1)
quarkus.datasource.username=username-default (2)
quarkus.datasource.jdbc.url=jdbc:h2:tcp://localhost/mem:default (3)
quarkus.datasource.jdbc.min-size=3 (4)
quarkus.datasource.jdbc.max-size=13 (5)

quarkus.datasource.users.db-kind=h2 (6)
quarkus.datasource.users.username=username1
quarkus.datasource.users.jdbc.url=jdbc:h2:tcp://localhost/mem:users
quarkus.datasource.users.jdbc.min-size=1
quarkus.datasource.users.jdbc.max-size=11

quarkus.datasource.inventory.db-kind=h2
quarkus.datasource.inventory.username=username2
quarkus.datasource.inventory.jdbc.url=jdbc:h2:tcp://localhost/mem:inventory
quarkus.datasource.inventory.jdbc.min-size=2
quarkus.datasource.inventory.jdbc.max-size=12
```

(1) The type of the data source that you want to use.

(2) The username for the data source.

(3) The JDBC connection URL of the data source.

(4) The minimum number of connections that are maintained in the connection pool of the data source.

(5) The maximum number of connections that are maintained in the connection pool of the data source.

(6) If you choose **users** as the name of the data source, the fully qualified reference of your data source is **quarkus.datasource.users**

3. Inject multiple data sources into by adding the **@Inject** annotation to each one of the class properties. When you configure multiple data sources for your application, ensure that you add the **io.quarkus.agroal.DataSource** qualifier with the name of the data source as the value to each **DataSource** class:

```
import javax.inject.Inject;
import io.agroal.api.AgroalDataSource;
```

```
import io.quarkus.agroal.DataSource;

@Inject
AgroalDataSource defaultDataSource;

@Inject
@DataSource("users")
AgroalDataSource usersDataSource;

@Inject
@DataSource("inventory")
AgroalDataSource inventoryDataSource;
```

Additional resources

- Setting the db-kind property for a data source
- Setting database credentials
- Setting up multiple persistence units

CHAPTER 3. REACTIVE DATA SOURCE CONFIGURATION

You can use a reactive data source driver to connect your application to a relational database.

3.1. SETTING THE REACTIVE DATA SOURCE CONNECTION URL

You must configure the connection URL for the reactive data source to complete configuration of your data source.

Prerequisites

- You have a Quarkus Maven project.
- You have added a reactive data source driver to your application.

Procedure

1. Navigate to your Quarkus project directory.
2. In the **src/main/resources/application.properties** file, set the value of the **quarkus.datasource.reactive.url** property to match the connection URL of your reactive data source:

```
quarkus.datasource.reactive.url=<datasource_URL>
```

   For example, to set the reactive data source connection URL for the PostgreSQL data source:

```
quarkus.datasource.reactive.url=postgresql:///your_database
```

3. Optionally, you can set the maximum number of connections in the connection pool of your data source to improve the performance of your application:

```
quarkus.datasource.reactive.max-size=20
```

   For example, to add a **postgresql** data source with a reactive data source driver:

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=<your_username>
quarkus.datasource.password=<your_password>

quarkus.datasource.reactive.url=postgresql://localhost:5432/quarkus_test_database
quarkus.datasource.reactive.max-size=20
```

Additional resources

- Driver extensions for Quarkus built-in databases
- JDBC database connection URLs for databases with production and development support
- JDBC database connection URLs for databases with development support

3.2. DISABLING A REACTIVE DATA SOURCE IN A SIMULTANEOUS CONFIGURATION

You can simultaneously configure a reactive driver extension and a JDBC driver extension for the same data source in your application. You can disable the reactive data source driver in a simultaneous configuration, thus forcing your application to use the JDBC data source driver.

Prerequisites

- You have a Quarkus Maven project.
- You have a JDBC data source driver and a reactive data source driver configured simultaneously in your application.

Procedure

1. Navigate to your Quarkus project directory.
2. Open the **src/main/resources/application.properties** file.
3. Set the **quarkus.datasource.reactive** property to **false** to disable the reactive data source:

```
quarkus.datasource.reactive=false
```

CHAPTER 4. DATA SOURCE MANAGEMENT

You can enable additional features through application properties to manage data sources.

4.1. DATA SOURCE HEALTH CHECK

External systems (like Kubernetes) perform health checks on your data sources to determine if your application is able to respond to requests.

If you are using the **quarkus-smallrye-health** extension, the **quarkus-agroal** and reactive client extensions automatically add a readiness health check to validate the data source.

To view information about the data source validation status, access the **/health/ready** endpoint of your application. If you have multiple data sources, the application checks all data sources and their statuses will be listed as **DOWN** on the endpoint if there is a data source validation failure.

You can disable this behavior using the **quarkus.datasource.health.enabled** property.

4.2. DATA SOURCE METRICS

When you use either the **quarkus-smallrye-metrics** or the **quarkus-micrometer** extension, metrics emitted by the **quarkus-agroal** extension are exposed on the metrics endpoint **/q/metrics**.

To enable metrics for all data sources, you can set the **quarkus.datasource.jdbc.enable-metrics** property to **true**. To disable metrics for all data sources, set **quarkus.datasource.jdbc.enable-metrics** property to **false**.

When you have data source metrics enabled (**quarkus.datasource.jdbc.enable-metrics=true**), you can disable metrics for a specific data source by setting **quarkus.datasource.<datasource_name>.jdbc.enable-metrics** property for a named data source to **false**.

When you have data source metrics disabled (**quarkus.datasource.jdbc.enable-metrics=false**), you can enable metrics for a specific datasource by setting **quarkus.datasource.<datasource_name>.jdbc.enable-metrics** property for a named data source to **true**.

You can use this feature to access the collected metrics programmatically. The collected metrics are available after calling the **dataSource.getMetrics()** method on an injected **AgroalDataSource** instance. If collection of metrics is disabled for the data source, all values are zero.

4.3. NARAYANA TRANSACTION MANAGER INTEGRATION

Narayana transaction manager integration is automatically added when the **quarkus-narayana-jta** extension is available.

You can override this by setting the **quarkus.datasource.jdbc.transactions** configuration property.

4.4. TEST WITH IN-MEMORY DATABASES

Use the database you intend to use in production, container technologies made it simple to achieve. It is possible to run integration tests using the JVM powered databases as well. H2 and Derby databases are commonly used in embedded mode to run integration tests.

The embedded engine will function properly in JVM mode but is unable to compile into a native executable. Quarkus does not support embedding the entire database engine into a native executable.

If you want to run integration tests in both JVM and/or native executables, you can add either **@QuarkusTestResource**(**H2DatabaseTestResource.class**) or **@QuarkusTestResource**(**DerbyDatabaseTestResource.class**) on any class in your integration tests. The test suite can now start and stop the embedded database as a separate process as necessary to run your tests.

@QuarkusTestResource(**H2DatabaseTestResource.class**) and **@QuarkusTestResource**(**DerbyDatabaseTestResource.class**) are provided by the artifacts having Maven coordinates **io.quarkus:quarkus-test-h2** and **io.quarkus:quarkus-test-derby**, respectively for H2 and Derby.

The following example demonstrates how to use the helper for H2 databases:

```
package my.app.integrationtests.db;

import io.quarkus.test.common.QuarkusTestResource;
import io.quarkus.test.h2.H2DatabaseTestResource;

@QuarkusTestResource(H2DatabaseTestResource.class)
public class TestResources {
}
```

This allows you to test your application even when it is compiled into a native executable, while the database will run in the JVM as usual.

Connect to it using:

```
quarkus.datasource.db-kind=h2
quarkus.datasource.jdbc.url=jdbc:h2:tcp://localhost/mem:test
```

APPENDIX A. JDBC DATABASE CONNECTION URLS FOR DATABASES WITH PRODUCTION AND DEVELOPMENT SUPPORT

You can use a JDBC database connection URL to connect to a relational database. Each database has its own format of the database URL that contains information about the database itself and other configuration properties.

The following examples show you JDBC database connection URLs for databases that have been tested with Red Hat build of Quarkus. For more information about supported databases and versions, see the Red Hat build of Quarkus Supported Configurations page.

A.1. POSTGRESQL DATABASE URL EXAMPLE

PostgreSQL runs only as a server. You must specify connection details or use the default values.

Configuration options

```
jdbc:postgresql:[//][host][:port][/database][?key=value…]
```

Example

```
jdbc:postgresql://localhost/test
```

Additional resources

- PostgreSQL documentation

A.2. MARIADB DATABASE URL EXAMPLE

MariaDB only runs as a server. You must specify connection details or use the default values.

Configuration options

```
jdbc:mariadb:[replication:|failover:|sequential:|aurora:]//<hostDescription>[,<hostDescription>…
]/[database][?<key1>=<value1>[&<key2>=<value2>]] hostDescription:: <host>[:<portnumber>] or
address=(host=<host>)[(port=<portnumber>)][(type=(master|slave))]
```

Example

```
jdbc:mariadb://localhost:3306/test
```

Additional resources

- MariaDB documentation

A.3. MYSQL DATABASE URL EXAMPLE

MySQL runs only as a server. You must specify connection details or use the default values.

Configuration options

```
jdbc:mysql:[replication:|failover:|sequential:|aurora:]//<hostDescription>[,<hostDescription>…
]/[database][?<key1>=<value1>[&<key2>=<value2>]] hostDescription:: <host>[:<portnumber>] or
address=(host=<host>)[(port=<portnumber>)][(type=(master|slave))]
```

Example

```
jdbc:mysql://localhost:3306/test
```

Additional resources

- MySQL documentation

A.4. MICROSOFT SQL SERVER DATABASE URL EXAMPLE

Microsoft SQL Server uses default system values unless you specify different values. The Microsoft SQL Server JDBC drivers function the same way as other drivers. The connection URL should be in the following format:

Configuration options

```
jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]
```

Example

```
jdbc:sqlserver://localhost:1433;databaseName=AdventureWorks
```

Additional resources

- Microsoft SQL Server documentation

APPENDIX B. JDBC DATABASE CONNECTION URLS FOR DATABASES WITH DEVELOPMENT SUPPORT

You can use a JDBC database connection URL to connect to a relational database. Each database has its own format of the database URL that contains information about the database itself and other configuration properties.

The following examples show you JDBC database connection URLs for databases with development support. For more information about development support, see the Development Support Scope of Coverage page.

B.1. DERBY DATABASE URL EXAMPLE

Derby is an embedded database that can run as a server, based on a file, or live in memory.

Configuration options

```
jdbc:derby:[//serverName[:portNumber]/][memory:]databaseName[;property=value[;property=value]]
```

Example

```
jdbc:derby://localhost:1527/myDB, jdbc:derby:memory:myDB;create=true
```

Additional resources

- Derby documentation

B.2. H2 DATABASE URL EXAMPLE

H2 is an embedded database that can run as a server, based on a file, or live completely in memory.

Configuration options

```
jdbc:h2:{ {.|mem:}[name] | [file:]fileName | {tcp|ssl}:[//]server[:port][,server2[:port]]/name }
[;key=value…]
```

Example

```
jdbc:h2:tcp://localhost/~/test, jdbc:h2:mem:myDB
```

Additional resources

- H2 database documentation

Revised on 2021-06-24 09:07:31 UTC