

Quarkus - Using Flyway

Flyway is a popular database migration tool that is commonly used in JVM environments.

Quarkus provides first class support for using Flyway as will be explained in this guide.

Setting up support for Flyway

To start using Flyway with your project, you just need to:

- add your migrations to the `src/main/resources/db/migration` folder as you usually do with Flyway
- activate the `migrate-at-start` option to migrate the schema automatically or inject the `Flyway` object and run your migration as you normally do

In your `pom.xml`, add the following dependencies:

- the Flyway extension
- your JDBC driver extension (`quarkus-jdbc-postgresql`, `quarkus-jdbc-h2`, `quarkus-jdbc-mariadb`, ...)

```
<dependencies>
  <!-- Flyway specific dependencies -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-flyway</artifactId>
  </dependency>

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

Flyway support relies on the Quarkus datasource config. It can be customized for the default datasource as well as for every `named datasource`. First, you need to add the datasource config to the `application.properties` file in order to allow Flyway to manage the schema. Also, you can customize the Flyway behaviour by using the following properties:

`quarkus.flyway.migrate-at-start`

`true` to execute Flyway automatically when the application starts, `false` otherwise.

default: false

quarkus.flyway.validate-on-migrate

true to validate the applied migrations against the available ones, **false** otherwise.

default: true

quarkus.flyway.clean-at-start

true to execute Flyway clean command automatically when the application starts, **false** otherwise.

default: false

quarkus.flyway.locations

Comma-separated list of locations to scan recursively for migrations. The location type is determined by its prefix. Unprefixed locations or locations starting with `classpath:` point to a package on the classpath and may contain both SQL and Java-based migrations. Locations starting with `filesystem:` point to a directory on the filesystem, may only contain SQL migrations and are only scanned recursively down non-hidden directories.

default: classpath:db/migration

quarkus.flyway.connect-retries

The maximum number of retries when attempting to connect to the database. After each failed attempt, Flyway will wait 1 second before attempting to connect again, up to the maximum number of times specified by `connect-retries`.

default: 0

quarkus.flyway.schemas

Comma-separated case-sensitive list of schemas managed by Flyway. The first schema in the list will be automatically set as the default one during the migration. It will also be the one containing the schema history table.

default: <none>

quarkus.flyway.table

The name of Flyway's schema history table. By default (single-schema mode) the schema history table is placed in the default schema for the connection provided by the datasource. When the `quarkus.flyway.schemas` property is set (multi-schema mode), the schema history table is placed in the first schema of the list.

default: flyway_schema_history

quarkus.flyway.sql-migration-prefix

The file name prefix for versioned SQL migrations. Versioned SQL migrations have the following file name structure: `prefixVERSIONseparatorDESCRIPTIONsuffix`, which using the defaults translates to `V1.1__My_description.sql`

default: V

quarkus.flyway.repeatable-sql-migration-prefix

The file name prefix for repeatable SQL migrations. Repeatable SQL migrations have the following file name structure: `prefixSeparatorDESCRIPTIONsuffix`, which using the defaults translates to `R__My_description.sql`

default: R

quarkus.flyway.baseline-on-migrate

Whether to automatically call baseline when migrate is executed against a non-empty schema with no metadata table. This schema will then be baselined with the **baseline-version** before executing the migrations. Only migrations above **baseline-version** will then be applied. This is useful for initial Flyway production deployments on projects with an existing DB.

Be careful when enabling this as it removes the safety net that ensures Flyway does not migrate the wrong database in case of a configuration mistake!

default: false

quarkus.flyway.baseline-version

The version to tag an existing schema with when executing baseline

default: 1

quarkus.flyway.baseline-description

The description to tag an existing schema with when executing baseline

default: '<< Flyway Baseline >>'

The following is an example for the **application.properties** file:

```
# configure your datasource
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=sarah
quarkus.datasource.password=connor
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/mydata
base

# Flyway minimal config properties
quarkus.flyway.migrate-at-start=true

# Flyway optional config properties
# quarkus.flyway.baseline-on-migrate=true
# quarkus.flyway.baseline-version=1.0.0
# quarkus.flyway.baseline-description=Initial version
# quarkus.flyway.connect-retries=10
# quarkus.flyway.schemas=TEST_SCHEMA
# quarkus.flyway.table=flyway_quarkus_history
# quarkus.flyway.locations=db/location1,db/location2
# quarkus.flyway.sql-migration-prefix=X
# quarkus.flyway.repeatable-sql-migration-prefix=K
```

Add a SQL migration to the default folder following the Flyway naming conventions:
src/main/resources/db/migration/V1.0.0__Quarkus.sql

```
CREATE TABLE quarkus
(
  id INT,
  name VARCHAR(20)
);
INSERT INTO quarkus(id, name)
VALUES (1, 'QUARKED');
```

Now you can start your application and Quarkus will run the Flyway's migrate method according to your config:

```
@ApplicationScoped
public class MigrationService {
    // You can Inject the object if you want to use it manually
    @Inject
    Flyway flyway; ①

    public void checkMigration() {
        // This will print 1.0.0
        System.out.println(flyway.info().current().getVersion()
.toString());
    }
}
```

① Inject the Flyway object if you want to use it directly

Multiple datasources

Flyway can be configured for multiple datasources. The Flyway properties are prefixed exactly the same way as the named datasources, for example:

```

quarkus.datasource.db-kind=h2
quarkus.datasource.username=username-default
quarkus.datasource.jdbc.url=jdbc:h2:tcp://localhost/mem:default
quarkus.datasource.jdbc.min-size=3
quarkus.datasource.jdbc.max-size=13

quarkus.datasource.users.db-kind=h2
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

# Flyway configuration for the default datasource
quarkus.flyway.schemas=DEFAULT_TEST_SCHEMA
quarkus.flyway.locations=db/default/location1,db/default/location2
quarkus.flyway.migrate-at-start=true

# Flyway configuration for the "users" datasource
quarkus.flyway.users.schemas=USERS_TEST_SCHEMA
quarkus.flyway.users.locations=db/users/location1,db/users/location2
quarkus.flyway.users.migrate-at-start=true

# Flyway configuration for the "inventory" datasource
quarkus.flyway.inventory.schemas=INVENTORY_TEST_SCHEMA
quarkus.flyway.inventory.locations=db/inventory/location1,db/inventory/location2
quarkus.flyway.inventory.migrate-at-start=true

```

Notice there's an extra bit in the key. The syntax is as follows: `quarkus.flyway.[optional name.][datasource property]`.



Without configuration, Flyway is set up for every datasource using the default settings.

Using the Flyway object

In case you are interested in using the `Flyway` object directly, you can inject it as follows:



If you enabled the `quarkus.flyway.migrate-at-start` property, by the time you use the Flyway instance, Quarkus will already have run the migrate operation

```
@ApplicationScoped
public class MigrationService {
    // You can Inject the object if you want to use it manually
    @Inject
    Flyway flyway; ①

    @Inject
    @FlywayDataSource("inventory") ②
    Flyway flywayForInventory;

    @Inject
    @Named("flyway_users") ③
    Flyway flywayForUsers;

    public void checkMigration() {
        // Use the flyway instance manually
        flyway.clean(); ④
        flyway.migrate();
        // This will print 1.0.0
        System.out.println(flyway.info().current().getVersion()
            .toString());
    }
}
```

- ① Inject the Flyway object if you want to use it directly
- ② Inject Flyway for named datasources using the Quarkus `FlywayDataSource` qualifier
- ③ Inject Flyway for named datasources
- ④ Use the Flyway instance directly