

Quarkus - Quarkus Extension for Spring Scheduling API

While users are encouraged to use [regular Quarkus scheduler](#), Quarkus provides a compatibility layer for Spring Scheduled in the form of the [spring-scheduled](#) extension.

This guide explains how a Quarkus application can leverage the well known Spring Scheduled annotation to configure and schedule tasks.



This technology is considered preview.

In *preview*, backward compatibility and presence in the ecosystem is not guaranteed. Specific improvements might require to change configuration or APIs and plans to become *stable* are under way. Feedback is welcome on our [mailing list](#) or as issues in our [GitHub issue tracker](#).

For a full list of possible extension statuses, check our [FAQ entry](#).

Prerequisites

To complete this guide, you need:

- less than 15 minutes
- an IDE
- JDK 1.8+ installed with [JAVA_HOME](#) configured appropriately
- Apache Maven 3.6.3
- Some familiarity with the Spring Web extension

Solution

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can go right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an [archive](#).

The solution is located in the [spring-scheduled-quickstart](#) directory.

Creating the Maven project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.7.0.Final:create \
  -DprojectId=org.acme \
  -DprojectArtifactId=spring-scheduler-quickstart \
  -DclassName="org.acme.spring.scheduler.CountResource" \
  -Dpath="/count" \
  -Dextensions="spring-scheduled"
cd spring-scheduler-quickstart
```

This command generates a Maven project with a REST endpoint and adds the `spring-scheduled` extension.

If you already have your Quarkus project configured, you can add the `spring-scheduled` extension to your project by running the following command in your project base directory:

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

This will add the following to your `pom.xml`:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-spring-scheduled</artifactId>
</dependency>
```

Creating a scheduled job

In the `org.acme.spring.scheduler` package, create the `CounterBean` class, with the following content:

```

package org.acme.spring.scheduler;

import org.springframework.scheduling.annotation.Scheduled;

import java.util.concurrent.atomic.AtomicInteger;
import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped ①
public class CounterBean {

    private AtomicInteger counter = new AtomicInteger();

    public int get() { ②
        return counter.get();
    }

    @Scheduled(cron="*/5 * * * * ?") ③
    void cronJob() {
        counter.incrementAndGet(); ④
        System.out.println("Cron expression hardcoded");
    }

    @Scheduled(cron = "{cron.expr}") ⑤
    void cronJobWithExpressionInConfig() {
        counter.incrementAndGet();
        System.out.println("Cron expression configured in
application.properties");
    }

    @Scheduled(fixedRate = 1000) ⑥
    void jobAtFixedRate() {
        counter.incrementAndGet();
        System.out.println("Fixed Rate expression");
    }

    @Scheduled(fixedRateString = "${fixedRate.expr}") ⑦
    void jobAtFixedRateInConfig() {
        counter.incrementAndGet();
        System.out.println("Fixed Rate expression configured in
application.properties");
    }
}

```

1. Declare the bean in the *application* scope. Spring only detects @Scheduled annotations in beans.
2. The `get()` method allows retrieving the current value.
3. Use the Spring `@Scheduled` annotation with a cron-like expression to instruct Quarkus to schedule this method run. In this example we're scheduling a task to be executed at 10:15am every

day.

4. The code is pretty straightforward. Every day at 10:15am, the counter is incremented.
5. Define a job with a cron-like expression `cron.expr` which is configurable in `application.properties`.
6. Define a method to be executed at a fixed interval of time. The period is expressed in milliseconds.
7. Define a job to be executed at a fixed interval of time `fixedRate.expr` which is configurable in `application.properties`.

Updating the application configuration file

Edit the `application.properties` file and add the `cron.expr` and the `fixedRate.expr` configuration:

```
# The syntax used by Spring for cron expressions is the same as
which is used by regular Quarkus scheduler.
cron.expr=*/5 * * * * ?
fixedRate.expr=1000
```

Updating the resource and the test

Edit the `CountResource` class, and update the content to:

```
package org.acme.spring.scheduler;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/count")
public class CountResource {

    @Inject
    CounterBean counter;    ①

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "count: " + counter.get();    ②
    }
}
```

1. Inject the `CounterBean`
2. Send back the current counter value

We also need to update the tests. Edit the `CountResourceTest` class to match:

```
package org.acme.spring.scheduler;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.containsString;

import org.junit.jupiter.api.Test;

import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class CountResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
            .when().get("/count")
            .then()
            .statusCode(200)
            .body(containsString("count")); ①
    }
}
```

1. Ensure that the response contains `count`

Package and run the application

Run the application with: `./mvnw compile quarkus:dev`. In another terminal, run `curl localhost:8080/count` to check the counter value. After a few seconds, re-run `curl localhost:8080/count` to verify the counter has been incremented.

Observe the console to verify that the following messages has been displayed: - `Cron expression hardcoded` - `Cron expression configured in application.properties` - `Fixed Rate expression` - `Fixed Rate expression configured in application.properties` These messages indicate that the executions of methods annotated with `@Scheduled` have been triggered.

As usual, the application can be packaged using `./mvnw clean package` and executed using the `-runner.jar` file. You can also generate the native executable with `./mvnw clean package -Pnative`.

Using Property Expressions

Quarkus supports the use of property expressions in the `application.properties` file so to externalize the configuration of the tasks you should store the properties in the `application.properties` file and use the `fixedRateString`, `initialDelayString` params respectively.

Note that this configuration is a build time configuration, the property expression will be resolved at build time.

Unsupported Spring Scheduled functionalities

Quarkus currently only supports a subset of the functionalities that Spring `@Scheduled` provides with more features being planned. Currently, the `fixedDelay` and `fixedDelayString` parameters are not supported, in other words, `@Scheduled` methods are always executed independently.

Important Technical Note

Please note that the Spring support in Quarkus does not start a Spring Application Context nor are any Spring infrastructure classes run. Spring classes and annotations are only used for reading metadata and / or are used as user code method return types or parameter types. What that means for end users, is that adding arbitrary Spring libraries will not have any effect. Moreover Spring infrastructure classes (like `org.springframework.beans.factory.config.BeanPostProcessor` for example) will not be executed.

More Spring guides

Quarkus has more Spring compatibility features. See the following guides for more details:

- [Quarkus - Extension for Spring DI](#)
- [Quarkus - Extension for Spring Web](#)
- [Quarkus - Extension for Spring Data JPA](#)
- [Quarkus - Reading properties from Spring Cloud Config Server](#)
- [Quarkus - Extension for Spring Boot properties](#)
- [Quarkus - Extension for Spring Cache](#)
- [Quarkus - Extension for Spring Security](#)