

# Quarkus - Scheduling Periodic Tasks

Modern applications often need to run specific tasks periodically. In this guide, you learn how to schedule periodic tasks.



If you need a clustered scheduler use the [Quartz extension](#).

## Prerequisites

To complete this guide, you need:

- less than 10 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.5.3+

## Architecture

In this guide, we create a straightforward application accessible using HTTP to get the current value of a counter. This counter is periodically (every 10 seconds) incremented.



## 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 `scheduler-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:1.3.0.CR1:create \  
  -DprojectId=org.acme \  
  -DprojectId=scheduler-quickstart \  
  -DclassName="org.acme.scheduler.CountResource" \  
  -Dpath="/count" \  
  -Dextensions="scheduler" \  
cd scheduler-quickstart
```

It generates:

- the Maven structure
- a landing page accessible on <http://localhost:8080>
- example `Dockerfile` files for both `native` and `jvm` modes
- the application configuration file
- an `org.acme.scheduler.CountResource` resource
- an associated test

The Maven project also imports the Quarkus scheduler extension.

## Creating a scheduled job

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

```

package org.acme.scheduler;

import java.util.concurrent.atomic.AtomicInteger;
import javax.enterprise.context.ApplicationScoped;
import io.quarkus.scheduler.Scheduled;
import io.quarkus.scheduler.ScheduledExecution;

@ApplicationScoped ①
public class CounterBean {

    private AtomicInteger counter = new AtomicInteger();

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

    @Scheduled(every="10s") ③
    void increment() {
        counter.incrementAndGet(); ④
    }

    @Scheduled(cron="0 15 10 * * ?") ⑤
    void cronJob(ScheduledExecution execution) {
        counter.incrementAndGet();
        System.out.println(execution.getScheduledFireTime());
    }

    @Scheduled(cron = "{cron.expr}") ⑥
    void cronJobWithExpressionInConfig() {
        counter.incrementAndGet();
        System.out.println("Cron expression configured in
application.properties");
    }
}

```

1. Declare the bean in the *application* scope
2. The `get()` method allows retrieving the current value.
3. Use the `@Scheduled` annotation to instruct Quarkus to run this method every 10 seconds provided a worker thread is available (Quarkus is using 10 worker threads for the scheduler). If it is not available the method invocation should be re-scheduled by default i.e it should be invoked as soon as possible. The invocation of the scheduled method does not depend on the status or result of the previous invocation.
4. The code is pretty straightforward. Every 10 seconds, the counter is incremented.
5. Define a job with a cron-like expression. The annotated method is executed at 10:15am every day.
6. Define a job with a cron-like expression `cron.expr` which is configurable in `application.properties`.

# Updating the application configuration file

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

```
# By default, the syntax used for cron expressions is based on
Quartz - http://www.quartz-scheduler.org/documentation/quartz-
2.3.0/tutorials/crontrigger.html
# You can change the syntax using the following property:
# quarkus.scheduler.cron-type=unix
cron.expr=*/5 * * * * ?
```

# Updating the resource and the test

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

```
package org.acme.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.scheduler;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

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

@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 message `Cron expression configured in application.properties` has been displayed indicating that the cron job using an expression configured in `application.properties` has 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`.

## Scheduler Configuration Reference

 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
------------------------	------	---------

 `quarkus.scheduler.cron-type`

The syntax used in CRON expressions.

`cron4j,`  
`quartz,`  
`unix,`  
`spring`

`quartz`