

# Quarkus - Micrometer Metrics

This guide demonstrates how your Quarkus application can utilize the Micrometer metrics library for runtime and application metrics.

## 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

## Architecture

In this example, we build a very simple microservice which offers one REST endpoint and that determines if a number is prime.

## 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 `micrometer-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:1.8.0.CR1:create \
  -DgroupId=org.acme \
  -DartifactId=micrometer-quickstart \
  -Dextensions="micrometer"
cd micrometer-quickstart
```

This command generates a Maven project, that imports the `micrometer` extension as a dependency.

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

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

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

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

You should also add a micrometer dependency for the registry of your choosing, e.g.

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

## Writing the application

The application consists of a single class that implements an algorithm for checking whether a number is prime. This algorithm is exposed over a REST interface. With the micrometer extension enabled, metrics for all http server requests are collected automatically.

We do want to add a few other metrics to demonstrate how those types work:

- A counter will be incremented for every prime number discovered
- A gauge will store the highest prime number discovered
- A timer will record the time spent testing if a number is prime.

```
package org.acme.micrometer;

import io.micrometer.core.instrument.MeterRegistry;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import java.util.concurrent.atomic.LongAccumulator;
import java.util.function.Supplier;

@Path("/")
public class PrimeNumberResource {

    private final LongAccumulator highestPrime = new
```

```

LongAccumulator(Long::max, 0);
private final MeterRegistry registry;

PrimeNumberResource(MeterRegistry registry) {
    this.registry = registry;

    // Create a gauge that uses the highestPrimeNumberSoFar
method
    // to obtain the highest observed prime number
    registry.gauge("prime.number.max", this,
        PrimeNumberResource::highestObservedPrimeNumber);
}

@GET
@Path("/{number}")
@Produces("text/plain")
public String checkIfPrime(@PathParam("number") int number) {
    if (number < 1) {
        return "Only natural numbers can be prime numbers.";
    }
    if (number == 1) {
        return "1 is not prime.";
    }
    if (number == 2) {
        return "2 is prime.";
    }
    if (number % 2 == 0) {
        return number + " is not prime, it is divisible by 2.";
    }

    Supplier<String> supplier = () -> {
        for (int i = 3; i < Math.floor(Math.sqrt(number)) + 1;
i = i + 2) {
            if (number % i == 0) {
                return number + " is not prime, is divisible by
" + i + ".";
            }
        }
        highestPrime.accumulate(number);
        return number + " is prime.";
    };

    return
registry.timer("prime.number.test").wrap(supplier).get();
}

/**
 * This method is called by the registered {@code
highest.prime.number} gauge.

```

```
    * @return the highest observed prime value
    */
    long highestObservedPrimeNumber() {
        return highestPrime.get();
    }
}
```

## Running and using the application

To run the microservice in dev mode, use `./mvnw clean compile quarkus:dev`

### Generate some values for the metrics

First, ask the endpoint whether some numbers are prime numbers.

```
curl localhost:8080/350
```

The application will respond that 350 is not a prime number because it can be divided by 2.

Now for some large prime number so that the test takes a bit more time:

```
curl localhost:8080/629521085409773
```

The application will respond that 629521085409773 is a prime number. If you want, try some more calls with numbers of your choice.

### Review the generated metrics

To view the metrics, execute `curl localhost:8080/metrics/`

Prometheus-formatted metrics will be returned in plain text in no particular order.

The application above has only one custom gauge that measures the time spent determining whether or not a number is prime. The micrometer extension enables additional system, jvm, and http metrics. A subset of the collected metrics are shown below.

```
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{method="GET",outcome="SUCCESS",s
tatus="200",uri="/{number}",} 4.0
http_server_requests_seconds_sum{method="GET",outcome="SUCCESS",sta
tus="200",uri="/{number}",} 0.041501773
# HELP http_server_requests_seconds_max
# TYPE http_server_requests_seconds_max gauge
```

```
http_server_requests_seconds_max{method="GET",outcome="SUCCESS",status="200",uri="/{number}",} 0.038066359
# HELP jvm_threads_peak_threads The peak live thread count since the Java virtual machine started or peak was reset
# TYPE jvm_threads_peak_threads gauge
jvm_threads_peak_threads 56.0
# HELP http_server_connections_seconds_max
# TYPE http_server_connections_seconds_max gauge
http_server_connections_seconds_max 0.102580737
# HELP http_server_connections_seconds
# TYPE http_server_connections_seconds summary
http_server_connections_seconds_active_count 5.0
http_server_connections_seconds_duration_sum 0.175032815
# HELP system_load_average_1m The sum of the number of runnable entities queued to available processors and the number of runnable entities running on the available processors averaged over a period of time
# TYPE system_load_average_1m gauge
system_load_average_1m 2.4638671875
# HELP http_server_bytes_written_max
# TYPE http_server_bytes_written_max gauge
http_server_bytes_written_max 39.0
# HELP http_server_bytes_written
# TYPE http_server_bytes_written summary
http_server_bytes_written_count 4.0
http_server_bytes_written_sum 99.0
# HELP jvm_classes_loaded_classes The number of classes that are currently loaded in the Java virtual machine
# TYPE jvm_classes_loaded_classes gauge
jvm_classes_loaded_classes 9341.0
# HELP prime_number_max
# TYPE prime_number_max gauge
prime_number_max 887.0
# HELP jvm_info JVM version info
# TYPE jvm_info gauge
jvm_info{runtime="OpenJDK Runtime Environment",vendor="Oracle Corporation",version="13.0.2+8",} 1.0
# HELP jvm_classes_unloaded_classes_total The total number of classes unloaded since the Java virtual machine has started execution
# TYPE jvm_classes_unloaded_classes_total counter
jvm_classes_unloaded_classes_total 28.0
# HELP prime_number_test_seconds
# TYPE prime_number_test_seconds summary
prime_number_test_seconds_count 3.0
prime_number_test_seconds_sum 1.94771E-4
# HELP prime_number_test_seconds_max
# TYPE prime_number_test_seconds_max gauge
prime_number_test_seconds_max 1.76162E-4
```

```

# HELP http_server_bytes_read
# TYPE http_server_bytes_read summary
http_server_bytes_read_count 4.0
http_server_bytes_read_sum 0.0
# HELP http_server_bytes_read_max
# TYPE http_server_bytes_read_max gauge
http_server_bytes_read_max 0.0
# HELP jvm_threads_states_threads The current number of threads
having NEW state
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{state="runnable",} 37.0
jvm_threads_states_threads{state="blocked",} 0.0
jvm_threads_states_threads{state="waiting",} 15.0
jvm_threads_states_threads{state="timed-waiting",} 4.0
jvm_threads_states_threads{state="new",} 0.0
jvm_threads_states_threads{state="terminated",} 0.0
# HELP jvm_buffer_memory_used_bytes An estimate of the memory that
the Java virtual machine is using for this buffer pool
# TYPE jvm_buffer_memory_used_bytes gauge
jvm_buffer_memory_used_bytes{id="mapped",} 0.0
jvm_buffer_memory_used_bytes{id="direct",} 149521.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes
that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'profiled
nmethods'",} 1.1403264E7
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",}
4194304.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",}
9.2274688E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",}
4.9803264E7
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-
nmethods'",} 2555904.0
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",}
6.9206016E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class
Space",} 6815744.0
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-
profiled nmethods'",} 2555904.0

```

Note that metrics appear lazily, you often won't see any data for your endpoint until something tries to access it, etc.

## Configuration Reference

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

Configuration property	Type	Default
<p> <code>quarkus.micrometer.enabled</code></p> <p>Micrometer metrics support. Micrometer metrics support is enabled by default.</p>	boolean	<code>true</code>
<p> <code>quarkus.micrometer.registry-enabled-default</code></p> <p>Micrometer MeterRegistry discovery. Micrometer MeterRegistry implementations discovered on the classpath will be enabled automatically by default.</p>	boolean	<code>true</code>
<p> <code>quarkus.micrometer.binder-enabled-default</code></p> <p>Micrometer MeterBinder discovery. Micrometer MeterBinder implementations discovered on the classpath will be enabled automatically by default.</p>	boolean	<code>true</code>
<p> <code>quarkus.micrometer.binder.vertx.enabled</code></p> <p>Vert.x metrics support. Support for Vert.x metrics will be enabled if micrometer support is enabled, Vert.x MetricsOptions is on the classpath and either this value is true, or this value is unset and <code>quarkus.micrometer.binder-enabled-default</code> is true.</p>	boolean	
<p> <code>quarkus.micrometer.binder.mp-metrics.enabled</code></p> <p>Microprofile Metrics support. Support for Microprofile metrics will be enabled if micrometer support is enabled, and this value is true. You need to also include the microprofile api jar to your dependencies: <code>&lt;dependency&gt;&lt;groupId&gt;org.eclipse.microprofile.metrics&lt;/groupId&gt; &lt;artifactId&gt;microprofile-metrics-api&lt;/artifactId&gt; &lt;/dependency&gt;</code></p>	boolean	
<p> <code>quarkus.micrometer.binder.jvm</code></p> <p>Micrometer JVM metrics support. Micrometer JVM metrics support is enabled by default.</p>	boolean	<code>true</code>
<p> <code>quarkus.micrometer.binder.system</code></p> <p>Micrometer System metrics support. Micrometer System metrics support is enabled by default.</p>	boolean	<code>true</code>

<p> <code>quarkus.micrometer.export.datadog.enabled</code></p> <p>Support for export to Datadog Support for Datadog will be enabled if micrometer support is enabled, the DatadogMeterRegistry is on the classpath and either this value is true, or this value is unset and <code>quarkus.micrometer.registry-enabled-default</code> is true.</p>	boolean	
<p> <code>quarkus.micrometer.export.jmx.enabled</code></p> <p>Support for export to JMX Support for JMX will be enabled if micrometer support is enabled, the JmxMeterRegistry is on the classpath and either this value is true, or this value is unset and <code>quarkus.micrometer.registry-enabled-default</code> is true.</p>	boolean	
<p> <code>quarkus.micrometer.export.prometheus.enabled</code></p> <p>Support for export to Prometheus. Support for Prometheus will be enabled if micrometer support is enabled, the PrometheusMeterRegistry is on the classpath and either this value is true, or this value is unset and <code>quarkus.micrometer.registry-enabled-default</code> is true.</p>	boolean	
<p> <code>quarkus.micrometer.export.prometheus.path</code></p> <p>The path for the prometheus metrics endpoint (produces text/plain). The default value is <code>/metrics</code>.</p>	string	<code>/metrics</code>
<p> <code>quarkus.micrometer.export.stackdriver.enabled</code></p> <p>Support for export to Stackdriver. &lt;p&gt; Support for Stackdriver will be enabled if micrometer support is enabled, the StackdriverMeterRegistry is on the classpath and either this value is true, or this value is unset and <code>{@code quarkus.micrometer.registry-enabled-default}</code> is true.</p> <div data-bbox="209 1512 276 1574" style="border: 1px solid #000; border-radius: 50%; width: 30px; height: 30px; display: flex; align-items: center; justify-content: center; margin: 10px 0;">  </div> <p>Stackdriver libraries do not yet support running in native mode. The Stackdriver MeterRegistry will be automatically disabled for native builds.</p> <p>See <a href="https://github.com/grpc/grpc-java/issues/5460">https://github.com/grpc/grpc-java/issues/5460</a></p>	boolean	
<p><code>quarkus.micrometer.binder.vertx.match-patterns</code></p> <p>Comma-separated case-sensitive list of regular expressions defining Paths that should be matched and used as tags. By default, the first path segment will be used. This default behavior will also apply to any URI path that is found (2xx or 5xx) but doesn't match elements in this list.</p>	list of string	

<p><code>quarkus.micrometer.binder.vertx.ignore-patterns</code></p> <p>Comma-separated case-sensitive list of regular expressions defining Paths that should be ignored / not measured.</p>	list of string									
<p><code>quarkus.micrometer.export.datadog</code></p> <p>Datadog MeterRegistry configuration &lt;p&gt; A property source for configuration of the Datadog MeterRegistry to push metrics using the Datadog API, see <a href="https://micrometer.io/docs/registry/datadog">https://micrometer.io/docs/registry/datadog</a>.</p> <p>Available values:</p> <table border="1" data-bbox="164 645 1169 1167"> <thead> <tr> <th>Property=Default</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>apiKey=YOUR_KEY</code></td> <td>Define the key used to push data using the Datadog API</td> </tr> <tr> <td><code>publish=true</code></td> <td>By default, gathered metrics will be published to Datadog when the MeterRegistry is enabled. Use this attribute to selectively disable publication of metrics in some environments.</td> </tr> <tr> <td><code>step=1m</code></td> <td>The interval at which metrics are sent to Datadog. The default is 1 minute.</td> </tr> </tbody> </table> <p>Other micrometer configuration attributes can also be specified.</p>	Property=Default	Description	<code>apiKey=YOUR_KEY</code>	Define the key used to push data using the Datadog API	<code>publish=true</code>	By default, gathered metrics will be published to Datadog when the MeterRegistry is enabled. Use this attribute to selectively disable publication of metrics in some environments.	<code>step=1m</code>	The interval at which metrics are sent to Datadog. The default is 1 minute.	Map<String, String>	required 
Property=Default	Description									
<code>apiKey=YOUR_KEY</code>	Define the key used to push data using the Datadog API									
<code>publish=true</code>	By default, gathered metrics will be published to Datadog when the MeterRegistry is enabled. Use this attribute to selectively disable publication of metrics in some environments.									
<code>step=1m</code>	The interval at which metrics are sent to Datadog. The default is 1 minute.									
<p><code>quarkus.micrometer.export.jmx</code></p> <p>JMX registry configuration properties. &lt;p&gt; A property source for configuration of the JMX MeterRegistry, see <a href="https://micrometer.io/docs/registry/jmx">https://micrometer.io/docs/registry/jmx</a>.</p>	Map<String, String>	required 								
<p><code>quarkus.micrometer.export.prometheus</code></p> <p>Prometheus registry configuration properties. &lt;p&gt; A property source for configuration of the Prometheus MeterRegistry, see <a href="https://micrometer.io/docs/registry/prometheus">https://micrometer.io/docs/registry/prometheus</a></p>	Map<String, String>	required 								

`quarkus.micrometer.export.stackdriver`

Stackdriver registry configuration properties. <p> A property source for configuration of the Stackdriver MeterRegistry, see <https://micrometer.io/docs/registry/stackdriver>.

Available values:

Property=Default	Description
<code>project-id=MY_PROJECT_ID</code>	Define the project id used to push data to Stackdriver Monitoring
<code>publish=true</code>	By default, gathered metrics will be published to Datadog when the MeterRegistry is enabled. Use this attribute to selectively disable publication of metrics in some environments.
<code>step=1m</code>	The interval at which metrics are sent to Stackdriver Monitoring. The default is 1 minute.

Map<String, String>

required



Other micrometer configuration attributes can also be specified.