Red Hat build of Quarkus 1.11

Configuring logging with Quarkus

Last Updated: 2021-06-24

Legal Notice

Abstract

Configure logging to collect information about events that occurred within your application.

Table of Contents

PREFACE

As an application developer, you can use logging to view and record messages about events that occur while your application is running. Log messages provide information that you can use to debug your application during development and testing, and to monitor your application in a production environment.

Prerequisites

- Have OpenJDK (JDK) 11 installed and the **JAVA_HOME** environment variable set to specify the location of the Java SDK.
  - Log in to the Red Hat Customer Portal to download Red Hat build of Open JDK from the Software Downloads page.
- Have Apache Maven 3.6.2 or higher installed.
  - Download Maven from the Apache Maven Project website.
- Have a Quarkus Maven project.
  - For information on how to create Quarkus applications with Maven, see Developing and compiling your Quarkus applications with Apache Maven.

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. JBOSS LOGMANAGER AND SUPPORTED LOGGING FRAMEWORKS

Quarkus uses the JBoss LogManager logging backend to collect and manage log data. You can use JBoss Logging to collect data about Quarkus internal events and also about the events within your application. You can configure logging behavior in your **application.properties** file.

JBoss LogManager supports several third-party logging APIs in addition to the JBoss Logging. JBoss LogManager merges the logs from all the supported logging APIs.

Supported APIs for logging:

- JDK JUL **java.util.logging**
- JBoss Logging
- SLF4J
- Apache Commons Logging
- Log4j 1.x
- Log4j 2

Quarkus handles all of its logging functionalities using JBoss Logging. When you use a library that relies on a different logging API, you need to exclude this library from the dependencies and configure JBoss Logging to use a logging adapter for the third-party API.

Additional resources

- Using logging adapters
- Adding Apache Log4j logging framework
- Example logging configuration

1.1. ADDING APACHE LOG4J LOGGING FRAMEWORK

Apache Log4j is a logging framework that includes a logging backend and a logging API. Since Quarkus uses the JBoss LogManager backend, you can add the **log4j2-jboss-logmanager** library to your project and use Log4j as a logging API. Adding the Log4j library routes the Log4j logs to the JBoss Log Manager. You do not need to include any Log4j dependencies.

Procedure

- Add the **log4j2-jboss-logmanager** library as a dependency of your project's **pom.xml** file:

pom.xml

```
<dependency>
  <groupId>org.jboss.logmanager</groupId>
  <artifactId>log4j2-jboss-logmanager</artifactId>
</dependency>
```

The **log4j2-jboss-logmanager** is the library for the Log4J version 2 API. If you want to use the legacy Log4J version 1 API then you must add the **log4j-jboss-logmanager** instead.

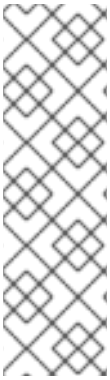
Additional resources

- About Log4j

1.2. USING LOGGING ADAPTERS

Quarkus relies on the JBoss Logging library for all the logging requirements.

When you are using libraries that have dependencies on other logging libraries, such as Apache Commons Logging, Log4j, or SLF4j, you must exclude those logging libraries from the dependencies and use one of the adapters provided by JBoss Logging. You do not need to add an adapter for libraries that are dependencies of Quarkus extensions.



NOTE

The third-party logging implementation is not included in the native executable and your application might fail to compile with an error message similar to the following:

```
Caused by java.lang.ClassNotFoundException:
org.apache.commons.logging.impl.LogFactoryImpl
```

You can prevent this error by configuring a JBoss Logging adapter for the third-party logging implementation that you use.

Procedure

- Depending on the logging library that you are using, add one of the adapters to your **pom.xml** file:
  - Apache Commons Logging:

    pom.xml

    ```
    <dependency>
      <groupId>org.jboss.logging</groupId>
      <artifactId>commons-logging-jboss-logging</artifactId>
    </dependency>
    ```

  - Log4j:

    pom.xml

    ```
    <dependency>
      <groupId>org.jboss.logmanager</groupId>
      <artifactId>log4j-jboss-logmanager</artifactId>
    </dependency>
    ```

  - Log4j2:

    pom.xml

    -

```
<dependency>
    <groupId>org.jboss.logmanager</groupId>
    <artifactId>log4j2-jboss-logmanager</artifactId>
</dependency>
```

- SLF4j:

  pom.xml

```
<dependency>
    <groupId>org.jboss.slf4j</groupId>
    <artifactId>slf4j-jboss-logmanager</artifactId>
</dependency>
```

CHAPTER 2. ENABLING JBOSS LOGGING FOR YOUR APPLICATION

When you want to use JBoss Logging to collect application logs, you must add a logger to each class that you want to produce log. The following procedure demonstrates how you can add logging to your application programmatically using the API approach or declaratively using annotations.

Procedure

1. Depending on your application code, use one of the following approaches:

   a. Create an instance of **org.jboss.logging.Logger** and initialize it by calling the static method **Logger.getLogger(Class)** for each class:

   src/main/java/org/acme/ExampleResource.java

```
import org.jboss.logging.Logger; (1)

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

@Path("/hello")
public class ExampleResource {

    private static final Logger LOG = Logger.getLogger(ExampleResource.class); (2)

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        LOG.info("Hello"); (3)
        return "hello";
    }
}
```

   (1) Add the **org.jboss.logging.Logger** import statement for each class namespace that you want to use.

   (2) Add reference to the logger singleton for each class.

   (3) Set a logging level for the log message.

   b. Inject a configured **org.jboss.logging.Logger** instance in your beans and resource classes:

   src/main/java/org/acme/ExampleResource.java

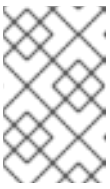
```
import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
```

```
import org.jboss.logging.Logger; (1)

import io.quarkus.arc.log.LoggerName;

@Path("/hello")
public class ExampleResource {

    @Inject
    Logger log; (2)

    @LoggerName("foo")
    Logger fooLog; (3)

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        log.info("Simple!");
        fooLog.info("Goes to foo logger!");
        return "hello";
    }
}
```

(1) Add the **org.jboss.logging.Logger** import statement for each class namespace that you want to use.

(2) The fully qualified class name of the declaring class is used as the logger name, which is equivalent to the initialization statement **org.jboss.logging.Logger.getLogger(ExampleResource.class)**.

(3) Set a name for the logger. In this example, the logger name is foo, which is equivalent to the initialization statement **org.jboss.logging.Logger.getLogger("foo")**.



NOTE

The logger instances are cached internally. A logger, that you inject into a bean, is shared for all bean instances to avoid the possible performance penalty associated with logger instantiation.

2. (Optional) Configure the logging output in your **application.properties** file:

src/main/resources/application.properties

```
<configuration_key>=<value>
```

For example, you can create a log file and print the output to a console and to the file:

src/main/resources/application.properties

```
quarkus.log.file.enable=true
quarkus.log.file.path=/tmp/trace.log
```

3. Run your application in development mode:

```
./mvnw quarkus:dev
```

4. Navigate to **http://localhost:8080/hello**.

5. Depending on your configuration, review the log messages on your terminal or in your log file. Example output for the **ExampleResource.class** with logging level set to **INFO**:

```
2021-05-21 15:38:39,751 INFO  [io.quarkus] (Quarkus Main Thread) my-project my-version
on JVM (powered by Quarkus 1.13.3.Final) started in 1.189s. Listening on:
http://localhost:8080
2021-05-21 15:38:39,765 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated.
Live Coding activated.
2021-05-21 15:38:39,766 INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi,
resteasy]
2021-05-21 15:38:58,790 INFO  [ExampleResource] (executor-thread-1) Hello
```

Additional resources

- JSON logging configuration properties

CHAPTER 3. SETTING RUNTIME CONFIGURATION

You can configure logging levels and logging categories settings in the **application.properties** file.

Logging categories are hierarchical. When you set a logging level for a category, the configuration applies to all sub-categories of that category.

There are two logging level settings: logging level and minimum logging level. The default logging level is **INFO**, the default minimum logging level is **DEBUG**. You can adjust both either globally, using the **quarkus.log.level** and **quarkus.log.min-level** property, or by category.

When you set the logging level below the minimum logging level, you must adjust the minimum logging level as well. Otherwise, the value of minimum logging level overrides the logging level.

Excessive logging has performance implications. You can adjust the minimum logging level to collect only relevant data about your application. Reducing log volume potentially optimize memory usage and improves the performance of your application. For example, in native execution the minimum level enables lower level checks (**isTraceEnabled**) to be folded to **false**, which results in dead code elimination.

Procedure

- Configure the logging in your **application.properties** file:

  src/main/resources/application.properties

  ```
  <configuration_key>=<value>
  ```

  The following example shows how to set the default logging level to **INFO** logging and include Hibernate **DEBUG** logs:

  src/main/resources/application.properties

  ```
  quarkus.log.level=INFO
  quarkus.log.category."org.hibernate".level=DEBUG
  ```

  NOTE

  When you set configuration properties via command line, make sure you escape **"**. For example **-Dquarkus.log.category.\"org.hibernate\".level=TRACE**.

Additional resources

- Logging levels
- Logging categories setting
- Example logging configuration
- Logging Configuration Reference - Quarkus Community Documentation

3.1. CONFIGURING LOGGING FORMAT

Quarkus uses a pattern-based logging formatter that generates human-readable text logs. The log entry displays the timestamp, the logging level, the class name, the thread ID and the message. You can customize the format for each log handler using a dedicated configuration property.

Prerequisites

- Have a Quarkus Maven project.

Procedure

- Set a value for the **quarkus.log.console.format** to configure the console handler:

  src/main/resources/application.properties

  ```
  quarkus.log.console.format=<logging_format_string>
  ```

  src/main/resources/application.properties

  ```
  quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
  ```

  This configuration results in the following log message formatting:

  ```
  14:11:07 INFO  [ExampleResource] (executor-thread-199) Hello
  ```

Additional resources

- Logging format strings
- Example logging configuration

3.1.1. Logging format strings

The following table shows the logging format string symbols that you can use to configure the format of log messages.

Table 3.1. Supported logging format symbols

Symbol	Summary	Description
%%	**%**	A simple **%** character.
%c	Category	The category name
%C	Source class	The source class name [a]
%d{xxx}	Date	Date with the given date format string, that follows the **java.text.SimpleDateFormat**
%e	Exception	The exception stack trace

Symbol	Summary	Description
%F	Source file	The source file name [a]
%h	Host name	The system simple host name
%H	Qualified host name	The fully qualified host name of the system. Depending on the OS configuration it might be the same as the simple host name.
%i	Process ID	The current process PID
%l	Source location	The source location (source file name, line number, class name, and method name) [a]
%L	Source line	The source line number [a]
%m	Full Message	The log message including exception trace
%M	Source method	The source method name [a]
%n	Newline	The platform-specific line separator string
%N	Process name	The name of the current process
%p	Level	The logging level of the message
%r	Relative time	The relative time in milliseconds since the start of the application log
%s	Simple message	The log message without exception trace
%t	Thread name	The thread name
%t{id}	Thread ID	The thread ID
%z{<zone name>}	Time zone	The time zone of the output in the **<zone name>** format
%X{<MDC property name>}	Mapped Diagnostics Context Value	The value from Mapped Diagnostics Context
%X	Mapped Diagnostics Context Values	All the values from Mapped Diagnostics Context in format {property.key=property.value}

Symbol	Summary	Description
%x	Nested Diagnostics context values	All the values from Nested Diagnostics Context in format {value1.value2}

[a] Format sequences that examine caller information might affect performance.

3.2. LOGGING CATEGORIES SETTING

You can use logging categories to organize log messages based on their severity or based on the component that they belong to. You can configure each category independently.

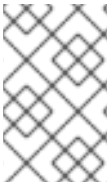
For every category the same settings applies to console, file, and syslog. You can override the settings by attaching one or more named handlers to a category.

Table 3.2. Logging categories configuration properties

Property Name	Default	Description
quarkus.log.category."<category-name>".level	**INFO** [a]	The level to configure the **<category-name>** category.
quarkus.log.category."<category-name>".min-level	**DEBUG**	The minimum logging level to configure the **<category-name>** category.
quarkus.log.category."<category-name>".use-parent-handlers	**true**	Enable the logger to send its output to the parent logger.
quarkus.log.category."<category-name>".handlers=[<handler>]	**empty** [b]	The names of the handlers that you want to attach to a specific category.

[a] Some extensions define customized default logging levels for certain categories to reduce log noise. Setting the logging level in configuration overrides any extension-defined logging levels.

[b] By default, the configured category inherits all the attached handlers from the root logger category.



NOTE

You must place logging category names inside double quotes (**"**) when using them in the names of properties to escape the periods (**.**) that are typically part of the category names.

Additional resources

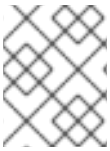
- Example logging configuration
- Logging levels

3.3. LOGGING LEVELS

You can use logging levels to categorize logs by severity or by their impact on the health and stability of your Quarkus application. Logging levels let you filter the critical events from the events that are purely informative.

Table 3.3. Quarkus supports the following logging levels:

Logging level	Description
OFF	Special level to turn off logging.
FATAL	A critical service failure or an inability to complete a service request.
ERROR	A significant disruption in a request or the inability to service a request.
WARN	A non-critical service error or a problem that might not require immediate correction.
INFO	Service lifecycle events or important related very-low-frequency information.
DEBUG	Messages that convey extra information regarding lifecycle or non-request-bound events which may be helpful for debugging.
TRACE	Messages that convey extra per-request debugging information that may be very high frequency.
ALL	Special level for all messages including custom levels.



NOTE

Additionally, you can use the logging level names described by the **java.util.logging** package.

Additional resources

- API documentation for the **java.util.logging** package

3.4. ROOT LOGGER CONFIGURATION

The root logger category is at the top of the logger hierarchy. The root logger captures all log messages of the specified logging level or higher that are sent to the server and are not captured by a logging category. The root logger category is configured at the top level of the logging configuration.

Table 3.4. Root logger configuration properties

Property Name	Default	Description
quarkus.log.level	**INFO**	The default logging level for every logging category.
quarkus.log.min-level	**DEBUG**	The default minimum logging level for every logging category.

3.5. QUARKUS LOG HANDLERS

A log handler is a logging component that sends log events to a recipient. Quarkus includes the following log handlers:

Console log handler

The console log handler is enabled by default. It outputs all log events to the console of your application (typically to the system's **stdout**).

File log handler

The file log handler is disabled by default. It outputs all log events to a file on the application's host. The file log handler supports log file rotation.

Syslog log handler

Syslog is a protocol for sending log messages on Unix-like systems. The specifications of the syslog protocol are defined in RFC 5424.

The syslog handler sends all log events to a syslog server (by default, the syslog server runs on the same host as the application). The syslog handler is disabled by default.

Additional resources

- Console Logging Configuration Properties - Quarkus Community Documentation
- File Logging Configuration Properties - Quarkus Community Documentation
- Syslog Logging Configuration Properties - Quarkus Community Documentation

3.6. EXAMPLE LOGGING CONFIGURATION

This section shows examples of how you can configure logging for your Quarkus project.

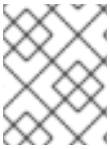
src/main/resources/application.properties

```
# Format log messages to have shorter time and shorter category prefixes.
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n

# Remove color from log messages.
quarkus.log.console.color=false

# Enable console DEBUG logging with the exception of Quarkus logs that have a logging level set to INFO.
quarkus.log.console.level=DEBUG
quarkus.log.category."io.quarkus".level=INFO
```

src/main/resources/application.properties

```
# Enable file logging and set a path to the log file.
quarkus.log.file.enable=true
quarkus.log.file.path=/tmp/trace.log

# Enable TRACE log messages in a log file.
quarkus.log.file.level=TRACE

# Set a format for the log file output.
```

```
quarkus.log.file.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n

# Set logging level to TRACE for specific categories.
quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE
quarkus.log.category."io.undertow.request.security".level=TRACE
```



NOTE

By default, the root logger level is set to **INFO**. When you want to collect logs for lower levels, such as **DEBUG** or **TRACE**, you need to change the root logger configuration.

src/main/resources/application.properties

```
# Set path to the log file.
quarkus.log.file.path=/tmp/trace.log

# Configure console format.
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n

# Configure a console log handler.
quarkus.log.handler.console."STRUCTURED_LOGGING".format=%e%n

# Configure a file log handler.
quarkus.log.handler.file."STRUCTURED_LOGGING_FILE".enable=true
quarkus.log.handler.file."STRUCTURED_LOGGING_FILE".format=%e%n

# Configure the category and associate it with the two named handlers.
quarkus.log.category."io.quarkus.category".level=INFO
quarkus.log.category."io.quarkus.category".handlers=STRUCTURED_LOGGING,STRUCTURED_LO
GGING_FILE
```

Additional resources

- Logging levels
- Logging categories setting
- Logging Configuration Reference - Quarkus Community Documentation

CHAPTER 4. CONFIGURING JSON LOGGING FORMAT

You can change the output format of the console log to JSON to make it easier to process and store the log information for later analysis.

To configure the JSON logging format, you need to add the **quarkus-logging-json** extension to your Quarkus project. The **quarkus-logging-json** extension replaces the output format configuration from the console configuration. The console configuration items such as the format string and the color settings will be ignored. Other console configuration items, including those controlling asynchronous logging and the logging level, continue to be applied.

Procedure

1. Add the **quarkus-logging-json** extension to the **pom.xml** file of your application:

   pom.xml

   ```
   <dependencies>
     <!-- ... your other dependencies are here ... -->
     <dependency>
       <groupId>io.quarkus</groupId>
       <artifactId>quarkus-logging-json</artifactId>
     </dependency>
   </dependencies>
   ```

2. (Optional) Set a profile-specific configuration for JSON logging in your **application.properties** file:

   src/main/resources/application.properties

   ```
   %<profile>.<configuration_key>=<value>
   ```

   The following example shows how you can disable JSON logging for the development and test profiles:

   src/main/resources/application.properties

   ```
   %dev.quarkus.log.console.json=false
   %test.quarkus.log.console.json=false
   ```

Additional resources

- JSON Configuration Properties - Quarkus Community Documentation

4.1. JSON LOGGING CONFIGURATION PROPERTIES

You can configure the JSON logging extension with the following configuration properties:

Table 4.1. JSON configuration properties

Configuration property	Description	Type	Default
quarkus.log.console.json	Enable the JSON console formatting extension.	boolean	**true**
quarkus.log.console.json.pretty-print	Enable pretty printing of the JSON record. [a]	boolean	**false**
quarkus.log.console.json.date-format	The format for dates. The **default** string sets the default format to be used.	string	**default**
quarkus.log.console.json.record-delimiter	Special end-of-record delimiter. By default, newline is used as delimiter.	string	
quarkus.log.console.json.zone-id	The ID for zone. The **default** string sets the default zone to be used.	string	**default**
quarkus.log.console.json.exception-output-type	The output type for exception.	**detailed, formatted, detailed-and-formatted**	**detailed**
quarkus.log.console.json.print-details	Enable detailed printing of the logs. The details include the source class name, source file name, source method name, and source line number. [b]	boolean	**false**

[a] Some processors and JSON parsers might fail to read pretty printed output.

[b] Printing the details can be expensive as the values are retrieved from the caller.

CHAPTER 5. CONFIGURING LOGGING FOR @QUARKUSTEST

If you want to configure logging for your **@QuarkusTest**, you need to set up the **maven-surefire-plugin** accordingly. You must specify the **LogManager** using the **java.util.logging.manager** system property.

Procedure

- Set the **LogManager** using the **java.util.logging.manager** system property:

  pom.xml

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemPropertyVariables>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
(1)
          <quarkus.log.level>DEBUG</quarkus.log.level> (2)
          <maven.home>${maven.home}</maven.home>
        </systemPropertyVariables>
      </configuration>
    </plugin>
  </plugins>
</build>
```

(1) Add the **org.jboss.logmanager.LogManager**.

(2) Enable debug logging for all logging categories.

CHAPTER 6. ADDITIONAL RESOURCES

- Logging Configuration Reference - Quarkus Community Documentation
- Centralized Log Management - Quarkus Community Documentation

Revised on 2021-06-24 08:58:41 UTC