Red Hat build of Quarkus 3.2

Logging configuration

Last Updated: 2024-04-04

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Read about the use of logging APIs in Red Hat build of Quarkus, configuring logging output, and using logging adapters for unified output.

Table of Contents

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. LOGGING CONFIGURATION

Read about the use of logging APIs in Quarkus, configuring logging output, and using logging adapters for unified output.

Quarkus uses the JBoss Log Manager logging backend for publishing application and framework logs. Quarkus supports the JBoss Logging API and multiple other logging APIs, seamlessly integrated with JBoss Log Manager. You can use any of the following APIs:

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

1.1. USE JBOSS LOGGING FOR APPLICATION LOGGING

When using the JBoss Logging API, your application requires no additional dependencies, as Quarkus automatically provides it.

An example of using the JBoss Logging API to log a message:

```
import org.jboss.logging.Logger;

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

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

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

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

NOTE

While JBoss Logging routes log messages into JBoss Log Manager directly, one of your libraries might rely on a different logging API. In such cases, you need to use a logging adapter to ensure that its log messages are routed to JBoss Log Manager as well.

1.2. GET AN APPLICATION LOGGER

In Quarkus, the most common ways to obtain an application logger are by:

- Declaring a logger field
- Simplified logging
- Injecting a configured logger

1.2.1. Declaring a logger field

With this classic approach, you use a specific API to obtain a logger instance, store it in a static field of a class, and call logging operations upon this instance.

The same flow can be applied with any of the supported logging APIs.

An example of storing a logger instance into a static field by using the JBoss Logging API:

```
package com.example;
import org.jboss.logging.Logger;

public class MyService {
    private static final Logger log = Logger.getLogger(MyService.class); 1

    public void doSomething() {
        log.info("It works!"); 2
    }
}
```

1 Define the logger field.

2 Invoke the desired logging methods on the **log** object.

1.2.2. Simplified logging

Quarkus simplifies logging by automatically adding logger fields to classes that use **io.quarkus.logging.Log**. This eliminates the need for repetitive boilerplate code and enhances logging setup convenience.

An example of simplified logging using static method calls:

```
package com.example;

import io.quarkus.logging.Log; 1

class MyService { 2
    public void doSomething() {
        Log.info("Simple!"); 3
    }
}
```

1 The **io.quarkus.logging.Log** class contains the same methods as JBoss Logging, except that they

are **static**.

2 Note that the class does not declare a logger field. This is because during application build, a **private static final org.jboss.logging.Logger** field is created automatically in each class that uses the **Log** API. The fully qualified name of the class that calls the **Log** methods is used as a logger name. In this example, the logger name would be **com.example.MyService**.

3 Finally, all calls to **Log** methods are rewritten to regular JBoss Logging calls on the logger field during the application build.

WARNING

Only use the **Log** API in application classes, not in external dependencies. **Log** method calls that are not processed by Quarkus at build time will throw an exception.

1.2.3. Injecting a configured logger

The injection of a configured **org.jboss.logging.Logger** logger instance with the **@Inject** annotation is another alternative to adding an application logger, but is applicable only to CDI beans.

You can use **@Inject Logger log**, where the logger gets named after the class you inject it to, or **@Inject @LoggerName("…") Logger log**, where the logger will receive the specified name. Once injected, you can use the **log** object to invoke logging methods.

An example of two different types of logger injection:

```
import org.jboss.logging.Logger;

@ApplicationScoped
class SimpleBean {

   @Inject
   Logger log; (1)

   @LoggerName("foo")
   Logger fooLog; (2)

   public void ping() {
     log.info("Simple!");
     fooLog.info("Goes to _foo_ logger!");
   }
}
```

1 The FQCN of the declaring class is used as a logger name, for example, **org.jboss.logging.Logger.getLogger(SimpleBean.class)** will be used.

2 In this case, the name *foo* is used as a logger name, for example,**org.jboss.logging.Logger.getLogger("foo")** will be used.

NOTE

The logger instances are cached internally. Therefore, when a logger is injected, for example, into a **@RequestScoped** bean, it is shared for all bean instances to avoid possible performance penalties associated with logger instantiation.

1.3. USE LOG LEVELS

Quarkus provides different log levels, which helps developers control the amount of information logged based on the severity of the events.

Table 1.1. Log levels used by Quarkus

OFF	A special level to use in configuration in order to turn off logging.
FATAL	A critical service failure or complete inability to service requests of any kind.
ERROR	A significant disruption in a request or the inability to service a request.
WARN	A non-critical service error or problem that may 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, useful for debugging.
TRACE	Messages that convey extra per-request debugging information that may be very high frequency.
ALL	A special level to use in configuration to turn on logging for all messages, including custom levels.

You can also configure the following levels for applications and libraries that use **java.util.logging**:

SEVERE	Same as **ERROR**.
WARNING	Same as **WARN**.
CONFIG	Service configuration information.
FINE	Same as **DEBUG**.
FINER	Same as **TRACE**.
FINEST	Increased debug output compared to **TRACE**, which might have a higher frequency.

Table 1.2. The mapping between the levels

Numerical level value	Standard level name	Equivalent **java.util.logging** (JUL) level name
1100	FATAL	Not applicable
1000	ERROR	SEVERE
900	WARN	WARNING
800	INFO	INFO
700	Not applicable	CONFIG
500	DEBUG	FINE
400	TRACE	FINER
300	Not applicable	FINEST

1.4. CONFIGURE THE LOG LEVEL, CATEGORY, AND FORMAT

JBoss Logging is built into Quarkus and provides unified configuration for all supported logging APIs.

Configure the runtime logging in the **application.properties** file.

An example of how you can set the default log level to **INFO** logging and include Hibernate **DEBUG** logs:

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

When you set the log level to below **DEBUG**, you must also adjust the minimum log level. This setting is either global, using the **quarkus.log.min-level** configuration property, or per category:

```
quarkus.log.category."org.hibernate".min-level=TRACE
```

This sets a floor level for which Quarkus needs to generate supporting code. The minimum log level must be set at build time so that Quarkus can open the door to optimization opportunities where logging on unusable levels can be elided.

An example from native execution:

Setting **INFO** as the minimum logging level sets lower-level checks, such as **isTraceEnabled**, to **false**. This identifies code like **if(logger.isDebug()) callMethod();** that will never be executed and mark it as "dead."

WARNING

If you add these properties on the command line, ensure the **"** character is escaped properly:

```
-Dquarkus.log.category.\"org.hibernate\".level=TRACE
```

All potential properties are listed in the logging configuration reference section.

1.4.1. Logging categories

Logging is configured on a per-category basis, with each category being configured independently. Configuration for a category applies recursively to all subcategories unless there is a more specific subcategory configuration.

The parent of all logging categories is called the "root category." As the ultimate parent, this category might contain a configuration that applies globally to all other categories. This includes the globally configured handlers and formatters.

Example 1.1. An example of a global configuration that applies to all categories:

```
quarkus.log.handlers=console,mylog
```

In this example, the root category is configured to use two handlers: **console** and **mylog**.

Example 1.2. An example of a per-category configuration:

```
quarkus.log.category."org.apache.kafka.clients".level=INFO
quarkus.log.category."org.apache.kafka.common.utils".level=INFO
```

This example shows how you can configure the minimal log level on the categories **org.apache.kafka.clients** and **org.apache.kafka.common.utils**.

For more information, see Logging configuration reference.

If you want to configure something extra for a specific category, create a named handler like **quarkus.log.handler.[console|file|syslog].<your-handler-name>.*** and set it up for that category by using **quarkus.log.category.<my-category>.handlers**.

An example use case can be a desire to use a different timestamp format for log messages which are saved to a file than the format used for other handlers.

For further demonstration, see the outputs of the Attaching named handlers to a category example.

Property Name	Default	Description
quarkus.log.category."<category-name>".level	**INFO** [a]	The level to use to configure the category named **<category-name>**. The quotes are necessary.
quarkus.log.category."<category-name>".min-level	**DEBUG**	The minimum logging level to use to configure the category named **<category-name>**. The quotes are necessary.
quarkus.log.category."<category-name>".use-parent-handlers	**true**	Specify whether this logger should send its output to its 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 may define customized default log levels for certain categories, in order to reduce log noise by default. Setting the log level in configuration will override any extension-defined log levels.

[b] By default, the configured category gets the same handlers attached as the one on the root logger.

NOTE

The **.** symbol separates the specific parts in the configuration property. The quotes in the property name are used as a required escape to keep category specifications, such as **quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE**, intact.

1.4.2. Root logger configuration

The root logger category is handled separately, and is configured by using the following properties:

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

- The parent category is examined if no level configuration exists for a given logger category.
- The root logger configuration is used if no specific configurations are provided for the category and any of its parent categories.

NOTE

Although the root logger's handlers are usually configured directly via **quarkus.log.console**, **quarkus.log.file** and **quarkus.log.syslog**, it can nonetheless have additional named handlers attached to it using the **quarkus.log.handlers** property.

1.5. LOGGING FORMAT

Quarkus uses a pattern-based logging formatter that generates human-readable text logs by default, but you can also configure the format for each log handler by using a dedicated property.

For the console handler, the property is **quarkus.log.console.format**.

The logging format string supports the following symbols:

Symbol	Summary	Description
%%	**%**	Renders a simple **%** character.
%c	Category	Renders the category name.
%C	Source class	Renders the source class name.footnote:calc[Format sequences which examine caller information may affect performance]
%d{xxx}	Date	Renders a date with the given date format string, which uses the syntax defined by **java.text.SimpleDateFormat**.
%e	Exception	Renders the thrown exception, if any.
%F	Source file	Renders the source file name.footnote:calc[]
%h	Host name	Renders the system simple host name.
%H	Qualified host name	Renders the system's fully qualified host name, which may be the same as the simple host name, depending on operating system configuration.
%i	Process ID	Render the current process PID.
%l	Source location	Renders the source location information, which includes source file name, line number, class name, and method name.footnote:calc[]
%L	Source line	Renders the source line number.footnote:calc[]
%m	Full Message	Renders the log message plus exception (if any).
%M	Source method	Renders the source method name.footnote:calc[]

Symbol	Summary	Description
%n	Newline	Renders the platform-specific line separator string.
%N	Process name	Render the name of the current process.
%p	Level	Render the log level of the message.
%r	Relative time	Render the time in milliseconds since the start of the application log.
%s	Simple message	Renders just the log message, with no exception trace.
%t	Thread name	Render the thread name.
%t{id}	Thread ID	Render the thread ID.
%z{<zone name>}	Time zone	Set the time zone of the output to **<zone name>**.
%X{<MDC property name>}	Mapped Diagnostic Context Value	Renders the value from Mapped Diagnostic Context
%X	Mapped Diagnostic Context Values	Renders all the values from Mapped Diagnostic Context in format {property.key=property.value}
%x	Nested Diagnostics context values	Renders all the values from Nested Diagnostics Context in format {value1.value2}

1.5.1. Alternative console logging formats

Changing the console log format is useful, for example, when the console output of the Quarkus application is captured by a service that processes and stores the log information for later analysis.

1.5.1.1. JSON logging format

The **quarkus-logging-json** extension may be employed to add support for the JSON logging format and its related configuration.

Add this extension to your build file as the following snippet illustrates:

- Using Maven:

```
<dependency>
  <groupId>io.quarkus</groupId>
```

```
    <artifactId>quarkus-logging-json</artifactId>
</dependency>
```

- Using Gradle:

```
implementation("io.quarkus:quarkus-logging-json")
```

By default, the presence of this extension replaces the output format configuration from the console configuration, and the format string and the color settings (if any) are ignored. The other console configuration items, including those controlling asynchronous logging and the log level, will continue to be applied.

For some, it will make sense to use humanly readable (unstructured) logging in dev mode and JSON logging (structured) in production mode. This can be achieved using different profiles, as shown in the following configuration.

Disable JSON logging in application.properties for dev and test mode

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

1.5.1.1.1. Configuration

Configure the JSON logging extension using supported properties to customize its behavior.

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

Console logging	Type	Default
quarkus.log.console.json Determine whether to enable the JSON console formatting extension, which disables "normal" console formatting. Environment variable: **QUARKUS_LOG_CONSOLE_JSON**	boolean	**true**
quarkus.log.console.json.pretty-print Enable "pretty printing" of the JSON record. Note that some JSON parsers will fail to read the pretty printed output. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_PRETTY_PRINT**	boolean	**false**
quarkus.log.console.json.date-format The date format to use. The special string "default" indicates that the default format should be used. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_DATE_FORMAT**	string	**default**

quarkus.log.console.json.record-delimiter The special end-of-record delimiter to be used. By default, newline is used. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_RECORD_DELIMITER**	string	
quarkus.log.console.json.zone-id The zone ID to use. The special string "default" indicates that the default zone should be used. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_ZONE_ID**	string	**default**
quarkus.log.console.json.exception-output-type The exception output type to specify. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_EXCEPTION_OUTPUT_TYPE**	**detailed**, **formatted**, **detailed-and-formatted**	**detailed**
quarkus.log.console.json.print-details Enable printing of more details in the log. Printing the details can be expensive as the values are retrieved from the caller. The details include the source class name, source file name, source method name, and source line number. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_PRINT_DETAILS**	boolean	**false**
quarkus.log.console.json.key-overrides Override keys with custom values. Omitting this value indicates that no key overrides will be applied. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_KEY_OVERRIDES**	string	
quarkus.log.console.json.excluded-keys Keys to be excluded from the JSON output. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_EXCLUDED_KEYS**	list of string	
quarkus.log.console.json.additional-field."field-name".value Additional field value. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_ADDITIONAL_FIELD__FIELD_NAME__VALUE**	string	required ❗

quarkus.log.console.json.additional-field."field-name".type Additional field type specification. Supported types: string, int, long String is the default if not specified. Environment variable: **QUARKUS_LOG_CONSOLE_JSON_ADDITIONAL_FIELD__FIELD_NAME__TYPE**	**string**, **int**, **long**	**string**
File logging	Type	Default
quarkus.log.file.json Determine whether to enable the JSON console formatting extension, which disables "normal" console formatting. Environment variable: **QUARKUS_LOG_FILE_JSON**	boolean	**true**
quarkus.log.file.json.pretty-print Enable "pretty printing" of the JSON record. Note that some JSON parsers will fail to read the pretty printed output. Environment variable: **QUARKUS_LOG_FILE_JSON_PRETTY_PRINT**	boolean	**false**
quarkus.log.file.json.date-format The date format to use. The special string "default" indicates that the default format should be used. Environment variable: **QUARKUS_LOG_FILE_JSON_DATE_FORMAT**	string	**default**
quarkus.log.file.json.record-delimiter The special end-of-record delimiter to be used. By default, newline is used. Environment variable: **QUARKUS_LOG_FILE_JSON_RECORD_DELIMITER**	string	
quarkus.log.file.json.zone-id The zone ID to use. The special string "default" indicates that the default zone should be used. Environment variable: **QUARKUS_LOG_FILE_JSON_ZONE_ID**	string	**default**
quarkus.log.file.json.exception-output-type The exception output type to specify. Environment variable: **QUARKUS_LOG_FILE_JSON_EXCEPTION_OUTPUT_TYPE**	**detailed**, **formatted**, **detailed-and-formatted**	**detailed**

quarkus.log.file.json.print-details Enable printing of more details in the log. Printing the details can be expensive as the values are retrieved from the caller. The details include the source class name, source file name, source method name, and source line number. Environment variable: **QUARKUS_LOG_FILE_JSON_PRINT_DETAILS**	boolean	**false**
quarkus.log.file.json.key-overrides Override keys with custom values. Omitting this value indicates that no key overrides will be applied. Environment variable: **QUARKUS_LOG_FILE_JSON_KEY_OVERRIDES**	string	
quarkus.log.file.json.excluded-keys Keys to be excluded from the JSON output. Environment variable: **QUARKUS_LOG_FILE_JSON_EXCLUDED_KEYS**	list of string	
quarkus.log.file.json.additional-field."field-name".value Additional field value. Environment variable: **QUARKUS_LOG_FILE_JSON_ADDITIONAL_FIELD__FIELD_NAME__VALUE**	string	required
quarkus.log.file.json.additional-field."field-name".type Additional field type specification. Supported types: string, int, long String is the default if not specified. Environment variable: **QUARKUS_LOG_FILE_JSON_ADDITIONAL_FIELD__FIELD_NAME__TYPE**	**string**, **int**, **long**	**string**
Syslog logging	Type	Default
quarkus.log.syslog.json Determine whether to enable the JSON console formatting extension, which disables "normal" console formatting. Environment variable: **QUARKUS_LOG_SYSLOG_JSON**	boolean	**true**
quarkus.log.syslog.json.pretty-print Enable "pretty printing" of the JSON record. Note that some JSON parsers will fail to read the pretty printed output. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_PRETTY_PRINT**	boolean	**false**

quarkus.log.syslog.json.date-format The date format to use. The special string "default" indicates that the default format should be used. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_DATE_FORMAT**	string	**defaul t**
quarkus.log.syslog.json.record-delimiter The special end-of-record delimiter to be used. By default, newline is used. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_RECORD_DELIMITER**	string	
quarkus.log.syslog.json.zone-id The zone ID to use. The special string "default" indicates that the default zone should be used. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_ZONE_ID**	string	**defaul t**
quarkus.log.syslog.json.exception-output-type The exception output type to specify. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_EXCEPTION_OUTPUT_TYPE**	**detailed**, **formatted**, **detailed-and-formatted**	**detailed**
quarkus.log.syslog.json.print-details Enable printing of more details in the log. Printing the details can be expensive as the values are retrieved from the caller. The details include the source class name, source file name, source method name, and source line number. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_PRINT_DETAILS**	boolean	**false**
quarkus.log.syslog.json.key-overrides Override keys with custom values. Omitting this value indicates that no key overrides will be applied. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_KEY_OVERRIDES**	string	

quarkus.log.syslog.json.excluded-keys Keys to be excluded from the JSON output. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_EXCLUDED_KEYS**	list of string	
quarkus.log.syslog.json.additional-field."field-name".value Additional field value. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_ADDITIONAL_FIELD__FIELD_NAME__VALUE**	string	required
quarkus.log.syslog.json.additional-field."field-name".type Additional field type specification. Supported types: string, int, long String is the default if not specified. Environment variable: **QUARKUS_LOG_SYSLOG_JSON_ADDITIONAL_FIELD__FIELD_NAME__TYPE**	**string**, **int**, **long**	**string**

WARNING

Enabling pretty printing might cause certain processors and JSON parsers to fail.

NOTE

Printing the details can be expensive as the values are retrieved from the caller. The details include the source class name, source file name, source method name, and source line number.

1.6. LOG HANDLERS

A log handler is a logging component responsible for the emission of log events to a recipient. Quarkus includes several different log handlers: **console**, **file**, and **syslog**.

The featured examples use **com.example** as a logging category.

1.6.1. Console log handler

The console log handler is enabled by default, and it directs all log events to the application's console, usually the system's **stdout**.

- A global configuration example:

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

- A per-category configuration example:

```
quarkus.log.handler.console.my-console-handler.format=%d{yyyy-MM-dd HH:mm:ss}
[com.example] %s%e%n

quarkus.log.category."com.example".handlers=my-console-handler
quarkus.log.category."com.example".use-parent-handlers=false
```

For details about its configuration, see the console logging configuration reference.

1.6.2. File log handler

To log events to a file on the application's host, use the Quarkus file log handler. The file log handler is disabled by default, so you must first enable it.

The Quarkus file log handler supports log file rotation. Log file rotation ensures effective log file management over time by maintaining a specified number of backup log files, while also keeping the primary log file up-to-date and manageable.

Log file rotation ensures effective log file management over time by maintaining a specified number of backup log files, while keeping the primary log file up-to-date and manageable.

- A global configuration example:

```
quarkus.log.file.enable=true
quarkus.log.file.path=application.log
quarkus.log.file.format=%d{yyyy-MM-dd HH:mm:ss} %-5p [%c] (%t) %s%e%n
```

- A per-category configuration example:

```
quarkus.log.handler.file.my-file-handler.enable=true
quarkus.log.handler.file.my-file-handler.path=application.log
quarkus.log.handler.file.my-file-handler.format=%d{yyyy-MM-dd HH:mm:ss} [com.example]
%s%e%n

quarkus.log.category."com.example".handlers=my-file-handler
quarkus.log.category."com.example".use-parent-handlers=false
```

For details about its configuration, see the file logging configuration reference.

1.6.3. Syslog log handler

The syslog handler in Quarkus follows the Syslog protocol, which is used to send log messages on UNIX-like systems. It utilizes the protocol defined in RFC 5424.

By default, the syslog handler is disabled. When enabled, it sends all log events to a syslog server, typically the local syslog server for the application.

- A global configuration example:

```
quarkus.log.syslog.enable=true
quarkus.log.syslog.app-name=my-application
quarkus.log.syslog.format=%d{yyyy-MM-dd HH:mm:ss} %-5p [%c] (%t) %s%e%n
```

- A per-category configuration example:

```
quarkus.log.handler.syslog.my-syslog-handler.enable=true
quarkus.log.handler.syslog.my-syslog-handler.app-name=my-application
quarkus.log.handler.syslog.my-syslog-handler.format=%d{yyyy-MM-dd HH:mm:ss}
[com.example] %s%e%n

quarkus.log.category."com.example".handlers=my-syslog-handler
quarkus.log.category."com.example".use-parent-handlers=false
```

For details about its configuration, see the Syslog logging configuration reference.

1.7. ADD A LOGGING FILTER TO YOUR LOG HANDLER

Log handlers, such as the console log handler, can be linked with a filter that determines whether a log record should be logged.

To register a logging filter:

1. Annotate a **final** class that implements **java.util.logging.Filter** with **@io.quarkus.logging.LoggingFilter**, and set the **name** property:

   An example of writing a filter:

```java
import io.quarkus.logging.LoggingFilter;
import java.util.logging.Filter;
import java.util.logging.LogRecord;

@LoggingFilter(name = "my-filter")
public final class TestFilter implements Filter {

    private final String part;

    public TestFilter(@ConfigProperty(name = "my-filter.part") String part) {
        this.part = part;
    }

    @Override
    public boolean isLoggable(LogRecord record) {
        return !record.getMessage().contains(part);
    }
}
```

   In this example, we exclude log records containing specific text from console logs. The specific text to filter on is not hard-coded; instead, it is read from the **my-filter.part** configuration property.

   An example of Configuring the filter in application.properties:

```
my-filter.part=TEST
```

2. Attach the filter to the corresponding handler using the **filter** configuration property, located in **application.properties**:

```
quarkus.log.console.filter=my-filter
```

1.8. EXAMPLES OF LOGGING CONFIGURATIONS

The following examples show some of the ways in which you can configure logging in Quarkus:

Console DEBUG logging except for Quarkus logs (INFO), no color, shortened time, shortened category prefixes

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

quarkus.log.category."io.quarkus".level=INFO
```

NOTE

If you add these properties in the command line, ensure **"** is escaped. For example, **-Dquarkus.log.category.\"io.quarkus\".level=DEBUG**.

File TRACE logging configuration

```
quarkus.log.file.enable=true
# Send output to a trace.log file under the /tmp directory
quarkus.log.file.path=/tmp/trace.log
quarkus.log.file.level=TRACE
quarkus.log.file.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
# Set 2 categories (io.quarkus.smallrye.jwt, io.undertow.request.security) to TRACE level
quarkus.log.min-level=TRACE
quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE
quarkus.log.category."io.undertow.request.security".level=TRACE
```

NOTE

As we don’t change the root logger, the console log will only contain **INFO** or higher level logs.

Named handlers attached to a category

```
# Send output to a trace.log file under the /tmp directory
quarkus.log.file.path=/tmp/trace.log
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
# Configure a named handler that logs to console
quarkus.log.handler.console."STRUCTURED_LOGGING".format=%e%n
# Configure a named handler that logs to file
quarkus.log.handler.file."STRUCTURED_LOGGING_FILE".enable=true
quarkus.log.handler.file."STRUCTURED_LOGGING_FILE".format=%e%n
# Configure the category and link the two named handlers to it
quarkus.log.category."io.quarkus.category".level=INFO
quarkus.log.category."io.quarkus.category".handlers=STRUCTURED_LOGGING,STRUCTURED_LO
GGING_FILE
```

Named handlers attached to the root logger

```
# configure a named file handler that sends the output to 'quarkus.log'
quarkus.log.handler.file.CONSOLE_MIRROR.enable=true
quarkus.log.handler.file.CONSOLE_MIRROR.path=quarkus.log
# attach the handler to the root logger
quarkus.log.handlers=CONSOLE_MIRROR
```

1.9. CENTRALIZED LOG MANAGEMENT

Use a centralized location to efficiently collect, store, and analyze log data from various components and instances of the application.

To send logs to a centralized tool such as Graylog, Logstash, or Fluentd, see the Quarkus Centralized log management guide.

1.10. CONFIGURE LOGGING FOR @QUARKUSTEST

Enable proper logging for **@QuarkusTest** by setting the **java.util.logging.manager** system property to **org.jboss.logmanager.LogManager**.

The system property must be set early on to be effective, so it is recommended to configure it in the build system.

Setting the java.util.logging.manager system property in the Maven Surefire plugin configuration

```
<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 Make sure the **org.jboss.logmanager.LogManager** is used.

2 Enable debug logging for all logging categories.

For Gradle, add the following configuration to the **build.gradle** file:

```
test {
  systemProperty "java.util.logging.manager", "org.jboss.logmanager.LogManager"
}
```

See also Running **@QuarkusTest** from an IDE .

1.11. USE OTHER LOGGING APIS

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

Suppose you use libraries that depend on other logging libraries, such as Apache Commons Logging, Log4j, or SLF4J. In that case, exclude them from the dependencies and use one of the JBoss Logging adapters.

This is especially important when building native executables, as you could encounter issues similar to the following when compiling the native executable:

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

The logging implementation is not included in the native executable, but you can resolve this issue using JBoss Logging adapters.

These adapters are available for popular open-source logging components, as explained in the next chapter.

1.11.1. Add a logging adapter to your application

For each logging API that is not **jboss-logging**:

1. Add a logging adapter library to ensure that messages logged through these APIs are routed to the JBoss Log Manager backend.

   NOTE

   This step is unnecessary for libraries that are dependencies of a Quarkus extension where the extension handles it automatically.

   - Apache Commons Logging:
     - Using Maven:

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

     - Using Gradle:

       ```
       implementation("org.jboss.logging:commons-logging-jboss-logging")
       ```

   - Log4j:
     - Using Maven:

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

- Using Gradle:

  ```
  implementation("org.jboss.logmanager:log4j-jboss-logmanager")
  ```

- Log4j 2:
  - Using Maven:

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

  - Using Gradle:

    ```
    implementation("org.jboss.logmanager:log4j2-jboss-logmanager")
    ```

    NOTE

    Do not include any Log4j dependencies because the **log4j2-jboss-logmanager** library contains all that is needed to use Log4j as a logging implementation.

- SLF4J:
  - Using Maven:

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

  - Using Gradle:

    ```
    implementation("org.jboss.slf4j:slf4j-jboss-logmanager")
    ```

2. Verify whether the logs generated by the added library adhere to the same format as the other Quarkus logs.

1.11.2. Use MDC to add contextual log information

Quarkus overrides the logging Mapped Diagnostic Context (MDC) to improve the compatibility with its reactive core.

1.11.2.1. Add and read MDC data

To add data to the MDC and extract it in your log output:

1. Use the **MDC** class to set the data.
2. Customize the log format to use **%X{mdc-key}**.

Let's consider the following code:

Example with JBoss Logging and io.quarkus.logging.Log

```
package me.sample;

import io.quarkus.logging.Log;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import org.jboss.logmanager.MDC;

import java.util.UUID;

@Path("/hello/jboss")
public class GreetingResourceJbossLogging {

    @GET
    @Path("/test")
    public String greeting() {
        MDC.put("request.id", UUID.randomUUID().toString());
        MDC.put("request.path", "/hello/test");
        Log.info("request received");
        return "hello world!";
    }
}
```

If you configure the log format with the following line:

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

You get messages containing the MDC data:

```
08:48:13 INFO request.id=c37a3a36-b7f6-4492-83a1-de41dbc26fe2 request.path=/hello/test
[me.sa.GreetingResourceJbossLogging] (executor-thread-1) request received
```

1.11.2.2. MDC and supported logging APIs

Depending on the API you use, the MDC class is slightly different. However, the APIs are very similar:

- Log4j 1 - **org.apache.log4j.MDC.put(key, value)**
- Log4j 2 - **org.apache.logging.log4j.ThreadContext.put(key, value)**
- SLF4J - **org.slf4j.MDC.put(key, value)**

1.11.2.3. MDC propagation

In Quarkus, the MDC provider has a specific implementation for handling the reactive context, ensuring that MDC data is propagated during reactive and asynchronous processing.

As a result, you can still access the MDC data in various scenarios:

- After asynchronous calls, for example, when a REST client returns a Uni.

- In code submitted to **org.eclipse.microprofile.context.ManagedExecutor**.
- In code executed with **vertx.executeBlocking()**.

NOTE

If applicable, MDC data is stored in a *duplicated context*, which is an isolated context for processing a single task (request).

1.12. LOGGING CONFIGURATION REFERENCE

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

Configuration property	Type	Default
quarkus.log.level The log level of the root category, which is used as the default log level for all categories. JBoss Logging supports Apache-style log levels: ● {@link org.jboss.logmanager.Level#FATAL} ● {@link org.jboss.logmanager.Level#ERROR} ● {@link org.jboss.logmanager.Level#WARN} ● {@link org.jboss.logmanager.Level#INFO} ● {@link org.jboss.logmanager.Level#DEBUG} ● {@link org.jboss.logmanager.Level#TRACE} In addition, it also supports the standard JDK log levels. Environment variable: **QUARKUS_LOG_LEVEL**	Level	**INFO**
quarkus.log.handlers The names of additional handlers to link to the root category. These handlers are defined in consoleHandlers, fileHandlers, or syslogHandlers. Environment variable: **QUARKUS_LOG_HANDLERS**	list of string	
Console logging	Type	Default
quarkus.log.console.enable If console logging should be enabled Environment variable: **QUARKUS_LOG_CONSOLE_ENABLE**	boolean	**true**

quarkus.log.console.stderr If console logging should go to **System#err** instead of **System#out**. Environment variable: **QUARKUS_LOG_CONSOLE_STDERR**	boolean	**false**
quarkus.log.console.format The log format. Note that this value is ignored if an extension is present that takes control of console formatting (e.g., an XML or JSON-format extension). Environment variable: **QUARKUS_LOG_CONSOLE_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.console.level The console log level. Environment variable: **QUARKUS_LOG_CONSOLE_LEVEL**	Level	**ALL**
quarkus.log.console.darken Specify how much the colors should be darkened. Note that this value is ignored if an extension is present that takes control of console formatting (e.g., an XML or JSON-format extension). Environment variable: **QUARKUS_LOG_CONSOLE_DARKEN**	int	**0**
quarkus.log.console.filter The name of the filter to link to the console handler. Environment variable: **QUARKUS_LOG_CONSOLE_FILTER**	string	
quarkus.log.console.async Indicates whether to log asynchronously Environment variable: **QUARKUS_LOG_CONSOLE_ASYNC**	boolean	**false**
quarkus.log.console.async.queue-length The queue length to use before flushing writing Environment variable: **QUARKUS_LOG_CONSOLE_ASYNC_QUEUE_LENGTH**	int	**512**

quarkus.log.console.async.overflow Determine whether to block the publisher (rather than drop the message) when the queue is full Environment variable: **QUARKUS_LOG_CONSOLE_ASYNC_OVERFLOW**	**block**, **discard**	**block**
File logging	Type	Default
quarkus.log.file.enable If file logging should be enabled Environment variable: **QUARKUS_LOG_FILE_ENABLE**	boolean	**false**
quarkus.log.file.format The log format Environment variable: **QUARKUS_LOG_FILE_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %h %N[%i] %-5p [%c{3.}] (%t) %s%e %n**
quarkus.log.file.level The level of logs to be written into the file. Environment variable: **QUARKUS_LOG_FILE_LEVEL**	Level	**ALL**
quarkus.log.file.path The name of the file in which logs will be written. Environment variable: **QUARKUS_LOG_FILE_PATH**	File	**quarkus.log**
quarkus.log.file.filter The name of the filter to link to the file handler. Environment variable: **QUARKUS_LOG_FILE_FILTER**	string	

quarkus.log.file.encoding The character encoding used Environment variable: **QUARKUS_LOG_FILE_ENCODING**	Charset	
quarkus.log.file.async Indicates whether to log asynchronously Environment variable: **QUARKUS_LOG_FILE_ASYNC**	boolean	**false**
quarkus.log.file.async.queue-length The queue length to use before flushing writing Environment variable: **QUARKUS_LOG_FILE_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.file.async.overflow Determine whether to block the publisher (rather than drop the message) when the queue is full Environment variable: **QUARKUS_LOG_FILE_ASYNC_OVERFLOW**	**block**, **discard**	**block**
quarkus.log.file.rotation.max-file-size The maximum log file size, after which a rotation is executed. Environment variable: **QUARKUS_LOG_FILE_ROTATION_MAX_FILE_SIZE**	MemorySize ?	**10M**
quarkus.log.file.rotation.max-backup-index The maximum number of backups to keep. Environment variable: **QUARKUS_LOG_FILE_ROTATION_MAX_BACKUP_INDEX**	int	**5**
quarkus.log.file.rotation.file-suffix The file handler rotation file suffix. When used, the file will be rotated based on its suffix. Example fileSuffix: .yyyy-MM-dd Note: If the suffix ends with .zip or .gz, the rotation file will also be compressed. Environment variable: **QUARKUS_LOG_FILE_ROTATION_FILE_SUFFIX**	string	
quarkus.log.file.rotation.rotate-on-boot Indicates whether to rotate log files on server initialization. You need to either set a **max-file-size** or configure a **file-suffix** for it to work. Environment variable: **QUARKUS_LOG_FILE_ROTATION_ROTATE_ON_BOOT**	boolean	**true**

Syslog logging	Type	Default
quarkus.log.syslog.enable If syslog logging should be enabled Environment variable: **QUARKUS_LOG_SYSLOG_ENABLE**	boolean	**false**
quarkus.log.syslog.endpoint The IP address and port of the Syslog server Environment variable: **QUARKUS_LOG_SYSLOG_ENDPOINT**	host:port	**localhost:514**
quarkus.log.syslog.app-name The app name used when formatting the message in RFC5424 format Environment variable: **QUARKUS_LOG_SYSLOG_APP_NAME**	string	
quarkus.log.syslog.hostname The name of the host the messages are being sent from Environment variable: **QUARKUS_LOG_SYSLOG_HOSTNAME**	string	

quarkus.log.syslog.facility Sets the facility used when calculating the priority of the message as defined by RFC-5424 and RFC-3164 Environment variable: **QUARKUS_LOG_SYSLOG_FACILITY**	**kernel**, **user-level**, **mail-system**, **system-daemons**, **security**, **syslogd**, **line-printer**, **network-news**, **uucp**, **clock-daemon**, **security2**, **ftp-daemon**, **ntp**, **log-audit**, **log-alert**, **clock-daemon2**, **local-use-0**, **local-use-1**, **local-use-2**, **local-use-3**, **local-use-4**, **local-use-5**, **local-use-6**, **local-use-7**	**user-level**

quarkus.log.syslog.syslog-type Set the **SyslogType syslog type** this handler should use to format the message sent Environment variable: **QUARKUS_LOG_SYSLOG_SYSLOG_TYPE**	**rfc5424**, **rfc3164**	**rfc5424**
quarkus.log.syslog.protocol Sets the protocol used to connect to the Syslog server Environment variable: **QUARKUS_LOG_SYSLOG_PROTOCOL**	**tcp**, **udp**, **ssl-tcp**	**tcp**
quarkus.log.syslog.use-counting-framing If enabled, the message being sent is prefixed with the size of the message Environment variable: **QUARKUS_LOG_SYSLOG_USE_COUNTING_FRAMING**	boolean	**false**
quarkus.log.syslog.truncate Set to **true** to truncate the message if it exceeds maximum length Environment variable: **QUARKUS_LOG_SYSLOG_TRUNCATE**	boolean	**true**
quarkus.log.syslog.block-on-reconnect Enables or disables blocking when attempting to reconnect a **org.jboss.logmanager.handlers.SyslogHandler.Protocol#TCP TCP** or **org.jboss.logmanager.handlers.SyslogHandler.Protocol#SSL_TCP SSL TCP** protocol Environment variable: **QUARKUS_LOG_SYSLOG_BLOCK_ON_RECONNECT**	boolean	**false**
quarkus.log.syslog.format The log message format Environment variable: **QUARKUS_LOG_SYSLOG_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.syslog.level The log level specifying what message levels will be logged by the Syslog logger Environment variable: **QUARKUS_LOG_SYSLOG_LEVEL**	Level	**ALL**

quarkus.log.syslog.filter The name of the filter to link to the file handler. Environment variable: **QUARKUS_LOG_SYSLOG_FILTER**	string	
quarkus.log.syslog.async Indicates whether to log asynchronously Environment variable: **QUARKUS_LOG_SYSLOG_ASYNC**	boolean	**false**
quarkus.log.syslog.async.queue-length The queue length to use before flushing writing Environment variable: **QUARKUS_LOG_SYSLOG_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.syslog.async.overflow Determine whether to block the publisher (rather than drop the message) when the queue is full Environment variable: **QUARKUS_LOG_SYSLOG_ASYNC_OVERFLOW**	**block**, **discard**	**block**
Logging categories	**Type**	**Default**
quarkus.log.category."categories".level The log level for this category. Note that to get log levels below **INFO**, the minimum level build-time configuration option also needs to be adjusted. Environment variable: **QUARKUS_LOG_CATEGORY__CATEGORIES__LEVEL**	InheritableLevel	**inherit**
quarkus.log.category."categories".handlers The names of the handlers to link to this category. Environment variable: **QUARKUS_LOG_CATEGORY__CATEGORIES__HANDLERS**	list of string	
quarkus.log.category."categories".use-parent-handlers Specify whether this logger should send its output to its parent Logger Environment variable: **QUARKUS_LOG_CATEGORY__CATEGORIES__USE_PARENT_HANDLERS**	boolean	**true**
Console handlers	**Type**	**Default**

quarkus.log.handler.console."console-handlers".enable If console logging should be enabled Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ENABLE**	boolean	**true**
quarkus.log.handler.console."console-handlers".stderr If console logging should go to **System#err** instead of **System#out**. Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__STDERR**	boolean	**false**
quarkus.log.handler.console."console-handlers".format The log format. Note that this value is ignored if an extension is present that takes control of console formatting (e.g., an XML or JSON-format extension). Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.handler.console."console-handlers".level The console log level. Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__LEVEL**	Level	**ALL**
quarkus.log.handler.console."console-handlers".darken Specify how much the colors should be darkened. Note that this value is ignored if an extension is present that takes control of console formatting (e.g., an XML or JSON-format extension). Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__DARKEN**	int	**0**
quarkus.log.handler.console."console-handlers".filter The name of the filter to link to the console handler. Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__FILTER**	string	

quarkus.log.handler.console."console-handlers".async Indicates whether to log asynchronously Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC**	boolean	**false**
quarkus.log.handler.console."console-handlers".async.queue-length The queue length to use before flushing writing Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.console."console-handlers".async.overflow Determine whether to block the publisher (rather than drop the message) when the queue is full Environment variable: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC_OVERFLOW**	**block**, **discard**	**block**
File handlers	Type	Default
quarkus.log.handler.file."file-handlers".enable If file logging should be enabled Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ENABLE**	boolean	**false**
quarkus.log.handler.file."file-handlers".format The log format Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %h %N[%i] %-5p [%c{3.}] (%t) %s%e %n**

quarkus.log.handler.file."file-handlers".level The level of logs to be written into the file. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__LEVEL**	Level	**ALL**
quarkus.log.handler.file."file-handlers".path The name of the file in which logs will be written. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__PATH**	File	**quarkus.log**
quarkus.log.handler.file."file-handlers".filter The name of the filter to link to the file handler. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__FILTER**	string	
quarkus.log.handler.file."file-handlers".encoding The character encoding used Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ENCODING**	Charset	
quarkus.log.handler.file."file-handlers".async Indicates whether to log asynchronously Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC**	boolean	**false**
quarkus.log.handler.file."file-handlers".async.queue-length The queue length to use before flushing writing Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.file."file-handlers".async.overflow Determine whether to block the publisher (rather than drop the message) when the queue is full Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC_OVERFLOW**	**block**, **discard**	**block**

quarkus.log.handler.file."file-handlers".rotation.max-file-size The maximum log file size, after which a rotation is executed. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_MAX_FILE_SIZE**	MemorySize ?	**10M**
quarkus.log.handler.file."file-handlers".rotation.max-backup-index The maximum number of backups to keep. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_MAX_BACKUP_INDEX**	int	**5**
quarkus.log.handler.file."file-handlers".rotation.file-suffix The file handler rotation file suffix. When used, the file will be rotated based on its suffix. Example fileSuffix: .yyyy-MM-dd Note: If the suffix ends with .zip or .gz, the rotation file will also be compressed. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_FILE_SUFFIX**	string	
quarkus.log.handler.file."file-handlers".rotation.rotate-on-boot Indicates whether to rotate log files on server initialization. You need to either set a **max-file-size** or configure a **file-suffix** for it to work. Environment variable: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_ROTATE_ON_BOOT**	boolean	**true**
Syslog handlers	**Type**	**Default**
quarkus.log.handler.syslog."syslog-handlers".enable If syslog logging should be enabled Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ENABLE**	boolean	**false**
quarkus.log.handler.syslog."syslog-handlers".endpoint The IP address and port of the Syslog server Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ENDPOINT**	host:port	**localhost:514**

quarkus.log.handler.syslog."syslog-handlers".app-name The app name used when formatting the message in RFC5424 format Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__APP_NAME**	string	
quarkus.log.handler.syslog."syslog-handlers".hostname The name of the host the messages are being sent from Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__HOSTNAME**	string	

quarkus.log.handler.syslog."syslog-handlers".facility Sets the facility used when calculating the priority of the message as defined by RFC-5424 and RFC-3164 Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FACILITY**	**kernel**, **user-level**, **mail-system**, **system-daemons**, **security**, **syslogd**, **line-printer**, **network-news**, **uucp**, **clock-daemon**, **security2**, **ftp-daemon**, **ntp**, **log-audit**, **log-alert**, **clock-daemon2**, **local-use-0**, **local-use-1**, **local-use-2**, **local-use-3**, **local-use-4**, **local-use-5**, **local-use-6**, **local-use-7**	**user-level**

quarkus.log.handler.syslog."syslog-handlers".syslog-type Set the **SyslogType syslog type** this handler should use to format the message sent Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__SYSLOG_TYPE**	**rfc5424**, **rfc3164**	**rfc5424**
quarkus.log.handler.syslog."syslog-handlers".protocol Sets the protocol used to connect to the Syslog server Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__PROTOCOL**	**tcp**, **udp**, **ssl-tcp**	**tcp**
quarkus.log.handler.syslog."syslog-handlers".use-counting-framing If enabled, the message being sent is prefixed with the size of the message Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__USE_COUNTING_FRAMING**	boolean	**false**
quarkus.log.handler.syslog."syslog-handlers".truncate Set to **true** to truncate the message if it exceeds maximum length Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__TRUNCATE**	boolean	**true**
quarkus.log.handler.syslog."syslog-handlers".block-on-reconnect Enables or disables blocking when attempting to reconnect a **org.jboss.logmanager.handlers.SyslogHandler.Protocol#TCP TCP** or **org.jboss.logmanager.handlers.SyslogHandler.Protocol#SSL_TCP SSL TCP** protocol Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__BLOCK_ON_RECONNECT**	boolean	**false**

quarkus.log.handler.syslog."syslog-handlers".format The log message format Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.handler.syslog."syslog-handlers".level The log level specifying what message levels will be logged by the Syslog logger Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__LEVEL**	Level	**ALL**
quarkus.log.handler.syslog."syslog-handlers".filter The name of the filter to link to the file handler. Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FILTER**	string	
quarkus.log.handler.syslog."syslog-handlers".async Indicates whether to log asynchronously Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC**	boolean	**false**
quarkus.log.handler.syslog."syslog-handlers".async.queue-length The queue length to use before flushing writing Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.syslog."syslog-handlers".async.overflow Determine whether to block the publisher (rather than drop the message) when the queue is full Environment variable: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC_OVERFLOW**	**block**, **discard**	**block**
Log cleanup filters - internal use	Type	Default

quarkus.log.filter."filters".if-starts-with The message prefix to match Environment variable: **QUARKUS_LOG_FILTER__FILTERS__IF_STARTS_WITH**	list of string	**inherit**
quarkus.log.filter."filters".target-level The new log level for the filtered message. Defaults to DEBUG. Environment variable: **QUARKUS_LOG_FILTER__FILTERS__TARGET_LEVEL**	Level	**DEBUG**

ABOUT THE MEMORYSIZE FORMAT

A size configuration option recognises string in this format (shown as a regular expression): **[0-9]+[KkMmGgTtPpEeZzYy]?**. If no suffix is given, assume bytes.