
Quarkus
Monitoring

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

public class MyClass {

   private static final Logger logger = Logger.getLogger(MyClass.class); (2)

   public List<Product> all() {
        logger.info("Listing all products"); (3)
        ...code omitted...
   }
}
```

1. Import the `Logger` class from JBoss Logging.
2. Get a logger instance. You must pass a class to the `getLogger` method. JBoss Logging uses the class name as the logger name, which in this example is `MyClass`. The logger name is useful to apply specific configurations to particular loggers.
3. Generate an informative log message.

The preceding code produces a log message similar to the following example:

```
2023-01-24 16:12:34,517 INFO  [com.red.tra.MyClass] (executor-thread-0) Listing
 all products
```

Red Hat Build of Quarkus uses JBoss LogManager

JBoss LogManager supports a number of facade APIs, such as *JBoss Logging*, *SLF4j*, or *Apache Commons Logging*, among others

Quarkus Log

```
import io.quarkus.logging.Log; (1)
...code omitted...

public class MyClass {

   public List<Product> all() {
        Log.info("Listing all products"); (2)
        ...code omitted...
   }
}
```

(1) The `io.quarkus.logging.Log` class exposes the same methods as `org.jboss.logging.Logger`, but as static methods.

(2) Generate a log message with level `INFO`. When you call this method, Quarkus internally calls the corresponding method from a `org.jboss.logging.Logger` instance with the same name as the class that calls the static method, which in this case is `MyClass`.

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

public class MyClass {
    @Inject
    Logger logger;

    ...code omitted...
}
```

Inyectar Logger

FATAL
For messages regarding failures that make the application crash or stop working.

ERROR
For messages regarding errors that cause undesired behaviors or issues when handling a request.

WARN
For messages regarding problems that do not require immediate action.

INFO
For messages regarding relevant information that is not related to errors or warnings. This is the default logging level in Red Hat Build of Quarkus.

DEBUG
For messages intended for debugging. Debug messages usually are more verbose.

Configurar el nivel de logger:

quarkus.log.level = DEBUG

quarkus.log.category."**org.example.myapp**".level = **WARNING**

quarkus.log.category."**org.apache.kafka.clients**".level = **DEBUG**

Customizing the Logging Format

```
quarkus.log.file.format = %d %-4p [%F] %m%n
```

The meaning of this format pattern is as follows:

- %d: Timestamp
- %p: Logging level. The -4 flag is for indentation
- %F: Source file
- %m: Logged message
- %n: New line

This format pattern produces logs similar to the following output:

```
2023-01-24 19:11:23,234 DEBUG [MyClass.java] Hello!
```

Generar log en un file

quarkus.log.file.enable = true
quarkus.log.file.path = /path/to/my/log_file

quarkus.log.file.rotation.rotate-on-boot → false

```
{
  "timestamp":"2023-01-26T18:02:30.606+01:00",
  "sequence":1932,
  "loggerClassName":"org.jboss.logging.Logger",
  "loggerName":"com.example.MyClass",
  "level":"DEBUG",
  "message":"Hello!",
  "threadName":"executor-thread-0",
  "threadId":166,
  "hostName":"myhost.example",
  "processName":"expenses-dev.jar",
  "processId":2982769,
  "customField": "customValue"
}
```


JSON Log

Debes instalar:
quarkus-logging-json

Con esa dependencia envías los logs a un Sistema centralizado:

- GrayLog
- LogStash
- Fluentd (EFK – ElasticSearch, FluentD, Kibana)

Log a un Sistema tercero

Debes instalar:
quarkus-logging-gelf