

Reactor Netty Reference Guide

Stephane Maldini, Violeta Georgieva

Version 1.1.26

Table of Contents

1. About the Documentation	1
1.1. Latest Version and Copyright Notice	1
1.2. Contributing to the Documentation	1
1.3. Getting Help	1
2. Getting Started	2
2.1. Introducing Reactor Netty	2
2.2. Prerequisites	2
2.3. Understanding the BOM and versioning scheme	2
2.4. Getting Reactor Netty	3
2.5. Support and policies	6

Chapter 1. About the Documentation

This section provides a brief overview of **Reactor Netty** reference documentation. You do not need to read this guide in a linear fashion. Each piece stands on its own, though they often refer to other pieces.

1.1. Latest Version and Copyright Notice

The **Reactor Netty** reference guide is available as **HTML** documents. The latest copy is available at <https://projectreactor.io/docs/netty/release/reference/index.html>

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this **Copyright Notice**, whether distributed in print or electronically.

1.2. Contributing to the Documentation

The reference guide is written in **AsciiDoc**, and you can find its sources at <https://github.com/reactor/reactor-netty/tree/1.1.x/docs/asciidoc>.

If you have an improvement, we will be happy to get a pull request from you!

We recommend that you check out a local copy of the repository so that you can generate the documentation by using the **asciidocctor** Gradle task and checking the rendering. Some of the sections rely on included files, so **GitHub** rendering is not always complete.

1.3. Getting Help

There are several ways to reach out for help with **Reactor Netty**. You can:

- Get in touch with the community on [Gitter](#).
- Ask a question on stackoverflow.com at [reactor-netty](#).
- Report bugs in **GitHub** issues. The repository is the following: [reactor-netty](#).



All of **Reactor Netty** is open source, [including this documentation](#).

Chapter 2. Getting Started

This section contains information that should help you get going with **Reactor Netty**. It includes the following information:

- [Introducing Reactor Netty](#)
- [Prerequisites](#)
- [Understanding the BOM and versioning scheme](#)
- [Getting Reactor Netty](#)

2.1. Introducing Reactor Netty

Suited for Microservices Architecture, **Reactor Netty** offers backpressure-ready network engines for **HTTP** (including Websockets), **TCP**, and **UDP**.

2.2. Prerequisites

Reactor Netty runs on **Java 8** and above.

It has transitive dependencies on:

- Reactive Streams v1.0.4
- Reactor Core v3.x
- Netty v4.1.x

2.3. Understanding the BOM and versioning scheme

Reactor Netty is part of the **Project Reactor BOM** (since the **Aluminium** release train). This curated list groups artifacts that are meant to work well together, providing the relevant versions despite potentially divergent versioning schemes in these artifacts.



The versioning scheme has changed between 0.9.x and 1.0.x (Dysprosium and Europium).

Artifacts follow a versioning scheme of **MAJOR.MINOR.PATCH-QUALIFIER** while the BOM is versioned using a CalVer inspired scheme of **YYYY.MINOR.PATCH-QUALIFIER**, where:

- **MAJOR** is the current generation of Reactor, where each new generation can bring fundamental changes to the structure of the project (which might imply a more significant migration effort)
- **YYYY** is the year of the first GA release in a given release cycle (like 1.0.0 for 1.0.x)
- **.MINOR** is a 0-based number incrementing with each new release cycle
 - in the case of projects, it generally reflects wider changes and can indicate a moderate migration effort
 - in the case of the BOM it allows discerning between release cycles in case two get first

released the same year

- **.PATCH** is a 0-based number incrementing with each service release
- **-QUALIFIER** is a textual qualifier, which is omitted in the case of GA releases (see below)

The first release cycle to follow that convention is thus **2020.0.x**, codename **Europium**. The scheme uses the following qualifiers (note the use of dash separator), in order:

- **-M1..-M9**: milestones (we don't expect more than 9 per service release)
- **-RC1..-RC9**: release candidates (we don't expect more than 9 per service release)
- **-SNAPSHOT**: snapshots
- *no qualifier* for GA releases



Snapshots appear higher in the order above because, conceptually, they're always "the freshest pre-release" of any given PATCH. Even though the first deployed artifact of a PATCH cycle will always be a -SNAPSHOT, a similarly named but more up-to-date snapshot would also get released after eg. a milestone or between release candidates.

Each release cycle is also given a codename, in continuity with the previous codename-based scheme, which can be used to reference it more informally (like in discussions, blog posts, etc...). The codenames represent what would traditionally be the MAJOR.MINOR number. They (mostly) come from the [Periodic Table of Elements](#), in increasing alphabetical order.



Up until Dysprosium, the BOM was versioned using a release train scheme with a codename followed by a qualifier, and the qualifiers were slightly different. For example: Aluminium-RELEASE (first GA release, would now be something like YYYY.0.0), Bismuth-M1, Californium-SR1 (service release would now be something like YYYY.0.1), Dysprosium-RC1, Dysprosium-BUILD-SNAPSHOT (after each patch, we'd go back to the same snapshot version. would now be something like YYYY.0.X-SNAPSHOT so we get 1 snapshot per PATCH)

2.4. Getting Reactor Netty

As [mentioned earlier](#), the easiest way to use **Reactor Netty** in your core is to use the **BOM** and add the relevant dependencies to your project. Note that, when adding such a dependency, you must omit the version so that the version gets picked up from the **BOM**.

However, if you want to force the use of a specific artifact's version, you can specify it when adding your dependency as you usually would. You can also forego the **BOM** entirely and specify dependencies by their artifact versions.

2.4.1. Maven Installation

The **BOM** concept is natively supported by **Maven**. First, you need to import the **BOM** by adding the following snippet to your **pom.xml**. If the top section (**dependencyManagement**) already exists in your pom, add only the contents.

```

<dependencyManagement> ①
  <dependencies>
    <dependency>
      <groupId>io.projectreactor</groupId>
      <artifactId>reactor-bom</artifactId>
      <version>2022.0.22</version> ②
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

```

① Notice the `dependencyManagement` tag. This is in addition to the regular `dependencies` section.

② As of this writing, `2022.0.22` is the latest version of the `BOM`. Check for updates at <https://github.com/reactor/reactor/releases>.

Next, add your dependencies to the relevant reactor projects, as usual (except without a `<version>`). The following listing shows how to do so:

```

<dependencies>
  <dependency>
    <groupId>io.projectreactor.netty</groupId>
    <artifactId>reactor-netty-core</artifactId> ①
  </dependency> ②
</dependencies>
<dependencies>
  <dependency>
    <groupId>io.projectreactor.netty</groupId>
    <artifactId>reactor-netty-http</artifactId>
  </dependency>
</dependencies>

```

① Dependency on `Reactor Netty`

② No version tag here

2.4.2. Gradle Installation

The `BOM` concept is supported in Gradle since version 5. The following listing shows how to import the `BOM` and add a dependency to `Reactor Netty`:

```
dependencies {
    // import a BOM
    implementation platform('io.projectreactor:reactor-bom:2022.0.22') ①

    // define dependencies without versions
    implementation 'io.projectreactor.netty:reactor-netty-core' ②
    implementation 'io.projectreactor.netty:reactor-netty-http'
}
```

① As of this writing, 2022.0.22 is the latest version of the BOM. Check for updates at <https://github.com/reactor/reactor/releases>.

② There is no third : separated section for the version. It is taken from the BOM.

2.4.3. Milestones and Snapshots

Milestones and developer previews are distributed through the **Spring Milestones** repository rather than **Maven Central**. To add it to your build configuration file, use the following snippet:

Milestones in Maven

```
<repositories>
  <repository>
    <id>spring-milestones</id>
    <name>Spring Milestones Repository</name>
    <url>https://repo.spring.io/milestone</url>
  </repository>
</repositories>
```

For Gradle, use the following snippet:

Milestones in Gradle

```
repositories {
    maven { url 'https://repo.spring.io/milestone' }
    mavenCentral()
}
```

Similarly, snapshots are also available in a separate dedicated repository (for both Maven and Gradle):

-SNAPSHOTS in Maven

```
<repositories>
  <repository>
    <id>spring-snapshots</id>
    <name>Spring Snapshot Repository</name>
    <url>https://repo.spring.io/snapshot</url>
  </repository>
</repositories>
```

-SNAPSHOTS in Gradle

```
repositories {
  maven { url 'https://repo.spring.io/snapshot' }
  mavenCentral()
}
```

2.5. Support and policies

The entries below are mirroring <https://github.com/reactor/.github/blob/main/SUPPORT.adoc>

2.5.1. Do you have a question?



Search Stack Overflow first; discuss if necessary

If you're unsure why something isn't working or wondering if there is a better way of doing it please check on **Stack Overflow** first and if necessary start a discussion. Use relevant tags among the ones we monitor for that purpose:

- **reactor-netty** for specific reactor-netty questions
- **project-reactor** for generic reactor questions

If you prefer real-time discussion, we also have a few **Gitter channels**:

- **reactor** is the historic most active one, where most of the community can help
- **reactor-core** is intended for more advanced pinpointed discussions around the inner workings of the library
- **reactor-netty** is intended for netty-specific questions

Refer to each project's README for potential other sources of information.

We generally discourage opening GitHub issues for questions, in favor of the two channels above.

2.5.2. Our policy on deprecations

When dealing with deprecations, given a version **A.B.C**, we'll ensure that:

- deprecations introduced in version **A.B.0** will be removed **no sooner than** version **A.B+1.0**
- deprecations introduced in version **A.B.1+** will be removed **no sooner than** version **A.B+2.0**
- we'll strive to mention the following in the deprecation javadoc:
 - target minimum version for removal
 - pointers to replacements for the deprecated method
 - version in which method was deprecated



This policy is officially in effect as of January 2021, for all modules in **2020.0** BOMs and newer release trains, as well as **Dysprosium** releases after **Dysprosium-SR15**.



Deprecation removal targets are not a hard commitment, and the deprecated methods **could live on further than these minimum target GA versions** (ie. only the most problematic deprecated methods will be removed aggressively).



That said, deprecated code that has outlived its minimum removal target version may be removed in any subsequent release (including patch releases, aka service releases) without further notice. So users should still strive to update their code as early as possible.

2.5.3. Support Timeline

Our GA release cadence is annual. The next release train is **2025**. The timeline is subject to change.

The following table summarises the support dates for each individual project followed by the BOM support.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
reactor-core				
3.7	2024-11-12	2026-08-31	2027-12-31	2024
3.6	2023-11-14	2025-08-31	2026-12-31	2023
3.5	2022-11-08	2024-08-31	2025-12-31	2022
3.4	2020-10-26	2024-08-31	2026-12-31	2020
reactor-netty				
1.2	2024-11-12	2026-08-31	2027-12-31	2024
1.1	2022-11-08	2025-08-31	2026-12-31	2022, 2023
1.0	2020-10-26	2024-08-31	2026-12-31	2020
reactor-kafka				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
1.3	2020-10-26	2026-08-31	2027-12-31	2020, 2022, 2023, 2024
reactor-pool				
1.1	2024-11-12	2026-08-31	2027-12-31	2024
1.0	2022-11-08	2025-08-31	2026-12-31	2022, 2023
0.2	2020-10-26	2024-08-31	2026-12-31	2020
reactor-addons				
3.5	2022-11-08	2026-08-31	2027-12-31	2022, 2023, 2024
3.4	2020-10-26	2024-08-31	2026-12-31	2020
reactor-kotlin-extensions				
1.2	2022-11-08	2026-08-31	2027-12-31	2022, 2023, 2024
1.1	2020-10-26	2024-08-31	2026-12-31	2020
reactor-rabbitmq				
1.5	2020-10-26	2024-08-31	2026-12-31	2020
reactor-bom				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
2024	2024-11-12	2026-08-31	2027-12-31 :leveloffset: 1 :leveloffset!: :leveloffset: 1 :sourcedir: ./../reactor-netty-core/src/main/java :examplesdir: ./../reactor-netty-examples/src/main/java/reactor/netty/examples/docume ntation/tcp/server :javadoc: https://projectreactor.io/docs/netty/1.1.26/api :nettyjavadoc: https://netty.io/4.1/api :wirelogger: reactor.netty.tcp.T cpServer = TCP Server Reactor Netty provides an easy to use and configure {javadoc}/reactor/ netty/tcp/TcpServe r.html[TcpServer]. It hides most of the Netty functionality that is needed to create a TCP server and adds Reactive Streams backpressure. == Starting and Stopping	=====
				9

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Callback	Description	<code>doOnBind</code>	Invoked when the server channel is about to bind.	<code>doOnBound</code>
Invoked when the server channel is bound.	<code>doOnChannelInit</code>	Invoked when initializing the channel.	<code>doOnConnection</code>	Invoked when a remote client is connected

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
doOnUnbound	Invoked when the server channel is unbound.	<p>=====</p> <p>The following example uses the <code>doOnConnection</code> and <code>doOnChannelInit</code> callbacks:</p> <p>====</p> <p>[source,java,indent=0] .{examplesdir}/lifecycle/Application.java ---- Unresolved directive in tcp-server.adoc - include:: {examplesdir}/lifecycle/Application.java [lines=18..39] ---- <1> <code>Netty</code> pipeline is extended with <code>ReadTimeoutHandler</code> when a remote client is connected. <2> <code>Netty</code> pipeline is extended with <code>LoggingHandler</code> when initializing the channel. ====</p> <p>== TCP-level Configurations</p> <p>This section describes three kinds of configuration that you can use at the TCP level:</p> <p>* [server-tcp-level-configurations-channel-options] *</p> <p>[server-tcp-level-configurations-</p>	=====	metric name

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
type	description	reactor.netty.tcp.server.connections.total	Gauge	The number of all opened connections. See [observability-metrics-connections-total]
reactor.netty.tcp.server.data.received	DistributionSummary	Amount of the data received, in bytes. See [observability-metrics-data-received]	reactor.netty.tcp.server.data.sent	DistributionSummary
Amount of the data sent, in bytes. See [observability-metrics-data-sent]	reactor.netty.tcp.server.errors	Counter	Number of errors that occurred. See [observability-metrics-errors-count]	reactor.netty.tcp.server.tls.handshake.time
Timer	Time spent for TLS handshake. See [observability-metrics-tls-handshake-time]	<p>=====</p> <p>These additional metrics are also available:</p> <p>ByteBufAllocator metrics</p> <p>[width="100%",options="header"]</p>	=====	metric name
type	description	reactor.netty.bytebuf.allocator.used.heap.memory	Gauge	The number of bytes reserved by heap buffer allocator. See [observability-metrics-used-heap-memory]
reactor.netty.bytebuf.allocator.used.direct.memory	Gauge	The number of bytes reserved by direct buffer allocator. See [observability-metrics-used-direct-memory]	reactor.netty.bytebuf.allocator.heap.arenas	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
The number of heap arenas (when PooledByteBufAllocator). See [observability-metrics-heap-arenas]	reactor.netty.bytebuf.allocator.direct.arenas	Gauge	The number of direct arenas (when PooledByteBufAllocator). See [observability-metrics-direct-arenas]	reactor.netty.bytebuf.allocator.threadlocal.caches
Gauge	The number of thread local caches (when PooledByteBufAllocator). See [observability-metrics-thread-local-caches]	reactor.netty.bytebuf.allocator.small.cache.size	Gauge	The size of the small cache (when PooledByteBufAllocator). See [observability-metrics-small-cache-size]
reactor.netty.bytebuf.allocator.normal.cache.size	Gauge	The size of the normal cache (when PooledByteBufAllocator). See [observability-metrics-normal-cache-size]	reactor.netty.bytebuf.allocator.chunk.size	Gauge
The chunk size for an arena (when PooledByteBufAllocator). See [observability-metrics-chunk-size]	reactor.netty.bytebuf.allocator.active.heap.memory	Gauge	The actual bytes consumed by in-use buffers allocated from heap buffer pools (when PooledByteBufAllocator). See [observability-metrics-active-heap-memory]	reactor.netty.bytebuf.allocator.active.direct.memory

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Gauge	The actual bytes consumed by in-use buffers allocated from direct buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-direct-memory]	===== <code>EventLoop</code> metrics [width="100%",options="header"]	=====	metric name
type	description	reactor.netty.eventloop.pending.tasks	Gauge	The number of tasks that are pending for processing on an event loop. See [observability-metrics-pending-tasks]

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<p>=====</p> <p>The following example enables that integration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/Application.java ---- Unresolved directive in tcp-server.adoc - include::{examplesdir}/metrics/Application.java[lines=18..32] ---- <1></pre> <p>Enables the built-in integration with Micrometer =====</p> <p>When TCP server metrics are needed for an integration with a system other than Micrometer or you want to provide your own integration with Micrometer, you can provide your own metrics recorder, as follows:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/custom/Application.java ---- Unresolved directive in tcp-server.adoc - include::{examplesdir}/metrics/custo</pre>	=====	contextual name	description	tls handshake

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Information and time spent for TLS handshake. See [observability-spans-tls-handshake-span] .	<p>=====</p> <p>The following example enables that integration. This concrete example uses Brave and reports the information to Zipkin. See the Micrometer Tracing documentation for OpenTelemetry setup.</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/tracing/Application.java --- Unresolved directive in tcp-server.adoc - include::{examplesdir}/tracing/Application.java[lines=18..81] --- <1> Initializes Brave, Zipkin, and the Observation registry. <2> Enables the built-in integration with Micrometer. =====</pre> <p>The result in Zipkin looks like:</p> <p>image::images/tcp-server-tracing.png[]</p> <p>=== Access Current Observation Project Micrometer provides a library</p>	=====	Callback	Description
16	that assists with			

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<code>doAfterResolve</code>	Invoked after the remote address has been resolved successfully.	<code>doOnChannelInit</code>	Invoked when initializing the channel.	<code>doOnConnect</code>
Invoked when the channel is about to connect.	<code>doOnConnected</code>	Invoked after the channel has been connected.	<code>doOnDisconnected</code>	Invoked after the channel has been disconnected.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
doOnResolve	Invoked when the remote address is about to be resolved.	doOnResolveError	Invoked in case the remote address hasn't been resolved successfully.	<p>=====</p> <p>The following example uses the <code>doOnConnected</code> and <code>doOnChannelInit</code> callbacks:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/life cycle/Application.j ava ---- Unresolved directive in tcp- client.adoc - include::{example sdir}/lifecycle/App lication.java[lines= 18..41] ---- <1> Netty pipeline is extended with ReadTimeoutHandler when the channel has been connected. <2> Netty pipeline is extended with LoggingHandler when initializing the channel. =====</pre> <p>== TCP-level Configurations</p> <p>This section describes three kinds of configuration that you can use at the TCP level:</p> <p>* [client-tcp-level-configurations-channel-options] *</p> <p>[client-tcp-level-configurations-event-wire-logger]</p>
18				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
=====	Configuration name	Description	disposeInactivePoolsInBackground	When this option is enabled, connection pools are regularly checked in the background, and those that are empty and been inactive for a specified time become eligible for disposal. By default, this background disposal of inactive pools is disabled.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<code>disposeTimeout</code>	<p>When <code>ConnectionProvider#dispose()</code> or <code>ConnectionProvider#disposeLater()</code> is called, trigger a graceful shutdown for the connection pools, with this grace period timeout. From there on, all calls for acquiring a connection will fail fast with an exception. However, for the provided <code>Duration</code>, pending acquires will get a chance to be served. Note: The rejection of new acquires and the grace timer start immediately, irrespective of subscription to the <code>Mono</code> returned by <code>ConnectionProvider#disposeLater()</code>. Subsequent calls return the same <code>Mono</code>, effectively getting notifications from the first graceful shutdown call and ignoring subsequently provided timeouts. By default, <code>disposeTimeout</code> is not specified.</p>	<code>maxConnections</code>	The maximum number of connections (per connection pool) before start pending. Default to 2 * available number of processors (but with a minimum value of 16).	<code>metrics</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Enables/disables built-in integration with Micrometer. <code>ConnectionProvider</code> . <code>MeterRegistrar</code> can be provided for integration with another metrics system. By default, metrics are not enabled.	<code>pendingAcquireMaxCount</code>	The maximum number of extra attempts at acquiring a connection to keep in a pending queue. If -1 is specified, the pending queue does not have upper limit. Default to 2 * max connections.	<code>pendingAcquireTimeout</code>	The maximum time before which a pending acquire must complete, or a <code>TimeoutException</code> is thrown (resolution: ms). If -1 is specified, no such timeout is applied. Default: 45 seconds.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<p>=====</p> <p>If you need to disable the connection pool, you can apply the following configuration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/pool/Application.java ---- Unresolved directive in tcp-client-conn-provider.adoc - include::{examplesdir}/pool/Application.java[lines=18..35] ---- =====</pre> <p>=== Disposing Connection Pool</p> <p>- If you use the default <code>ConnectionProvider</code> provided by Reactor Netty, invoke <code>{javadoc}/reactor/netty/http/HttpResources.html[HttpResources]#disposeLoopsAndConnections/#disposeLoopsAndConnectionsLater</code> method.</p> <p>NOTE: Disposing <code>HttpResources</code> means that every client that is using it, will not be able to use it anymore!</p>		metric name	type	description
22				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
reactor.netty.connection.provider.total.connections	Gauge	The number of all connections, active or idle. See [observability-metrics-total-connections]	reactor.netty.connection.provider.active.connections	Gauge
The number of the connections that have been successfully acquired and are in active use. See [observability-metrics-active-connections]	reactor.netty.connection.provider.max.connections	Gauge	The maximum number of active connections that are allowed. See [observability-metrics-max-connections]	reactor.netty.connection.provider.idle.connections
Gauge	The number of the idle connections. See [observability-metrics-idle-connections]	reactor.netty.connection.provider.pending.connections	Gauge	The number of requests that are waiting for a connection. See [observability-metrics-pending-connections]
reactor.netty.connection.provider.pending.connections.time	Timer	Time spent in pending acquire a connection from the connection pool. See [observability-metrics-pending-connections-time]	reactor.netty.connection.provider.max.pending.connections	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<p>The maximum number of requests that will be queued while waiting for a ready connection. See [observability-metrics-max-pending-connections]</p>	<p>=====</p> <p>The following example enables that integration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/pool/metrics/Application.java --- Unresolved directive in tcp-client-conn-provider.adoc - include::{examplesdir}/pool/metrics/Application.java[lines=18..45] --- <1> Enables the built-in integration with Micrometer ===== == SSL and TLS When you need SSL or TLS, you can apply the following configuration. By default, if OpenSSL is available, the {nettyjavadoc}/io/netty/handler/ssl/SSLProvider.html#OPENSSL[SslProvider.OPENSSL] provider is used as a provider. Otherwise, the provider is {nettyjavadoc}/io/netty/handler/ssl/SSLProvider.html#JDK[SslProvider.JDK]. You can switch the provider by</pre>	<p>=====</p>	<p>metric name</p>	<p>type</p>
24				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
description	reactor.netty.tcp.client.data.received	DistributionSummary	Amount of the data received, in bytes. See [observability-metrics-data-received]	reactor.netty.tcp.client.data.sent
DistributionSummary	Amount of the data sent, in bytes. See [observability-metrics-data-sent]	reactor.netty.tcp.client.errors	Counter	Number of errors that occurred. See [observability-metrics-errors-count]
reactor.netty.tcp.client.tls.handshake.time	Timer	Time spent for TLS handshake. See [observability-metrics-tls-handshake-time]	reactor.netty.tcp.client.connect.time	Timer
Time spent for connecting to the remote address. See [observability-metrics-connect-time]	reactor.netty.tcp.client.address.resolver	Timer	Time spent for resolving the address. See [observability-metrics-hostname-resolution-time]	<p>=====</p> <p>These additional metrics are also available:</p> <p>Pooled ConnectionProvider metrics</p> <p>[width="100%",options="header"]</p>
=====	metric name	type	description	reactor.netty.connection.provider.total.connections
Gauge	The number of all connections, active or idle. See [observability-metrics-total-connections]	reactor.netty.connection.provider.active.connections	Gauge	The number of the connections that have been successfully acquired and are in active use. See [observability-metrics-active-connections]

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
reactor.netty.connection.provider.max.connections	Gauge	The maximum number of active connections that are allowed. See [observability-metrics-max-connections]	reactor.netty.connection.provider.idle.connections	Gauge
The number of the idle connections. See [observability-metrics-idle-connections]	reactor.netty.connection.provider.pending.connections	Gauge	The number of requests that are waiting for a connection. See [observability-metrics-pending-connections]	reactor.netty.connection.provider.pending.connections.time
Timer	Time spent in pending acquire a connection from the connection pool. See [observability-metrics-pending-connections-time]	reactor.netty.connection.provider.max.pending.connections	Gauge	The maximum number of requests that will be queued while waiting for a ready connection. See [observability-metrics-max-pending-connections]
=====	=====	metric name	type	description
<p>ByteBufAllocator metrics</p> <p>[width="100%",options="header"]</p>				
reactor.netty.bytebuf.allocator.used.heap.memory	Gauge	The number of bytes reserved by heap buffer allocator. See [observability-metrics-used-heap-memory]	reactor.netty.bytebuf.allocator.used.direct.memory	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
The number of bytes reserved by direct buffer allocator. See [observability-metrics-used-direct-memory]	reactor.netty.bytebuf.allocator.heap.arenas	Gauge	The number of heap arenas (when PooledByteBufAllocator). See [observability-metrics-heap-arenas]	reactor.netty.bytebuf.allocator.direct.arenas
Gauge	The number of direct arenas (when PooledByteBufAllocator). See [observability-metrics-direct-arenas]	reactor.netty.bytebuf.allocator.threadlocal.caches	Gauge	The number of thread local caches (when PooledByteBufAllocator). See [observability-metrics-thread-local-caches]
reactor.netty.bytebuf.allocator.small.cache.size	Gauge	The size of the small cache (when PooledByteBufAllocator). See [observability-metrics-small-cache-size]	reactor.netty.bytebuf.allocator.normal.cache.size	Gauge
The size of the normal cache (when PooledByteBufAllocator). See [observability-metrics-normal-cache-size]	reactor.netty.bytebuf.allocator.chunk.size	Gauge	The chunk size for an arena (when PooledByteBufAllocator). See [observability-metrics-chunk-size]	reactor.netty.bytebuf.allocator.active.heap.memory
Gauge	The actual bytes consumed by in-use buffers allocated from heap buffer pools (when PooledByteBufAllocator). See [observability-metrics-active-heap-memory]	reactor.netty.bytebuf.allocator.active.direct.memory	Gauge	The actual bytes consumed by in-use buffers allocated from direct buffer pools (when PooledByteBufAllocator). See [observability-metrics-active-direct-memory]

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<div> <div>===== EventLoop metrics</div> <div>[width="100%",options="header"]</div> </div>	=====	metric name	type	description

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
reactor.netty.eventloop.pending.tasks	Gauge	The number of tasks that are pending for processing on an event loop. See [observability-metrics-pending-tasks]	<p>=====</p> <p>The following example enables that integration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/Application.java ---- Unresolved directive in tcp-client.adoc - include::{examplesdir}/metrics/Application.java[lines=18..34] ---- <1> Enables the built-in integration with Micrometer =====</pre> <p>When TCP client metrics are needed for an integration with a system other than Micrometer or you want to provide your own integration with Micrometer, you can provide your own metrics recorder, as follows:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/custom/Application.java ---- Unresolved directive in tcp-client.adoc - include::{examplesdir}/metrics/custo</pre>	=====

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
contextual name	description	hostname resolution	Information and time spent for resolving the address. See [observability-spans-hostname-resolution-span] .	connect

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Information and time spent for connecting to the remote address. See [observability-spans-connect-span] .	tls handshake	Information and time spent for TLS handshake. See [observability-spans-tls-handshake-span] .	<p>=====</p> <p>The following example enables that integration. This concrete example uses Brave and reports the information to Zipkin. See the Micrometer Tracing documentation for OpenTelemetry setup.</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/tracing/Application.java ---- Unresolved directive in tcp-client.adoc - include::{examplesdir}/tracing/Application.java[lines=18..81] ---- <1> Initializes Brave, Zipkin, and the Observation registry. <2> Enables the built-in integration with Micrometer. =====</pre> <p>The result in Zipkin looks like:</p> <p>image::images/tcp-client-tracing.png[]</p> <p>=== Access Current Observation Project Micrometer provides a library that assists with</p>	=====
				31

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Configuration name	Description	<code>bindAddressSupplier</code>	The supplier of the local address to bind to.	<code>cacheMaxTimeToLive</code>
The max time to live of the cached DNS resource records (resolution: seconds). If the time to live of the DNS resource record returned by the DNS server is greater than this max time to live, this resolver ignores the time to live from the DNS server and uses use this max time to live. Default to <code>Integer.MAX_VALUE</code> .	<code>cacheMinTimeToLive</code>	The min time to live of the cached DNS resource records (resolution: seconds). If the time to live of the DNS resource record returned by the DNS server is less than this min time to live, this resolver ignores the time to live from the DNS server and uses this min time to live. Default: 0.	<code>cacheNegativeTimeToLive</code>	The time to live of the cache for the failed DNS queries (resolution: seconds). Default: 0.
<code>completeOncePreferredResolved</code>	When this setting is enabled, the resolver notifies as soon as all queries for the preferred address type are complete. When this setting is disabled, the resolver notifies when all possible address types are complete. This configuration is applicable for <code>DnsNameResolver#resolveAll(String)</code> . By default, this setting is enabled.	<code>disableOptionalRecord</code>	Disables the automatic inclusion of an optional record that tries to give a hint to the remote DNS server about how much data the resolver can read per response. By default, this setting is enabled.	<code>disableRecursionDesired</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Specifies whether this resolver has to send a DNS query with the recursion desired (RD) flag set. By default, this setting is enabled.	<code>dnsAddressResolverGroupProvider</code>	Sets a custom function to create a <code>DnsAddressResolverGroup</code> given a <code>DnsNameResolverBuilder</code>	<code>hostsFileEntriesResolver</code>	Sets a custom <code>{nettyjavadoc}/io/netty/resolver/HostsFileEntriesResolver.html[HostsFileEntriesResolver]</code> to be used for hosts file entries. Default: <code>{nettyjavadoc}/io/netty/resolver/DefaultHostsFileEntriesResolver.html[DefaultHostsFileEntriesResolver]</code> .
<code>maxPayloadSize</code>	Sets the capacity of the datagram packet buffer (in bytes). Default: 4096.	<code>maxQueriesPerResolve</code>	Sets the maximum allowed number of DNS queries to send when resolving a host name. Default: 16.	<code>ndots</code>
Sets the number of dots that must appear in a name before an initial absolute query is made. Default: -1 (to determine the value from the OS on Unix or use a value of 1 otherwise).	<code>queryTimeout</code>	Sets the timeout of each DNS query performed by this resolver (resolution: milliseconds). Default: 5000.	<code>resolveCache</code>	The cache to use to store resolved DNS entries.
<code>resolvedAddressTypes</code>	The list of the protocol families of the resolved address.	<code>retryTcpOnTimeout</code>	Specifies whether this resolver will also fallback to TCP if a timeout is detected. By default, the resolver will only try to use TCP if the response is marked as truncated.	<code>roundRobinSelection</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Enables an {nettyjavadoc}/io/netty/resolver/AddressResolverGroup.html[AddressResolverGroup] of {nettyjavadoc}/io/netty/resolver/dns/DnsNameResolver.html[DnsNameResolver] that supports random selection of destination addresses if multiple are provided by the nameserver. See {nettyjavadoc}/io/netty/resolver/dns/RoundRobinDnsAddressResolverGroup.html[RoundRobinDnsAddressResolverGroup]. Default: {nettyjavadoc}/io/netty/resolver/dns/DnsAddressResolverGroup.html[DnsAddressResolverGroup]	runOn	Performs the communication with the DNS servers on the given {javadoc}/reactor/netty/resources/LoopResources.html[LoopResources]. By default, the LoopResources specified on the client level are used.	searchDomains	The list of search domains of the resolver. By default, the effective search domain list is populated by using the system DNS search domains.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
trace	A specific logger and log level to be used by this resolver when generating detailed trace information in case of resolution failure.	<p>=====</p> <p>Sometimes, you may want to switch to the JVM built-in resolver. To do so, you can configure the TcpClient as follows:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/resolver/custom/Application.java --- Unresolved directive in tcp-client.adoc - include::{examplesdir}/resolver/custom/Application.java[lines=18..37] --- <1> Sets the JVM built-in resolver. =====</pre> <p>:leveloffset: 3</p> <p>:leveloffset: 1</p> <p>:sourcedir:</p> <pre>./../reactor-netty-http/src/main/java :examplesdir: ./../reactor-netty-examples/src/main/java/reactor/netty/examples/documentation/http/server :javadoc: https://projectreactor.io/docs/netty/1.1.26/api :nettyjavadoc: https://netty.io/4.1/api :wirelogger:</pre>	=====	Configuration name

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Description	<code>baseDirectory</code>	Configures the directory where to store the data on the disk. Default to generated temp directory.	<code>charset</code>	Configures the <code>Charset</code> for the data. Default to <code>StandardCharsets.UTF_8</code> .
<code>maxInMemorySize</code>	Configures the maximum in-memory size per data i.e. the data is written on disk if the size is greater than <code>maxInMemorySize</code> , else it is in memory. If set to <code>-1</code> the entire contents is stored in memory. If set to <code>0</code> the entire contents is stored on disk. Default to <code>16kb</code> .	<code>maxSize</code>	Configures the maximum size per data. When the limit is reached, an exception is raised. If set to <code>-1</code> this means no limitation. Default to <code>-1</code> - unlimited.	<code>scheduler</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Configures the scheduler to be used for offloading disk operations in the decoding phase. Default to <code>Schedulers#boundedElastic()</code>	<code>streaming</code>	When set to <code>true</code> , the data is streamed directly from the parsed input buffer stream, which means it is not stored either in memory or file. When <code>false</code> , parts are backed by in-memory and/or file storage. Default to <code>false</code> . NOTE that with streaming enabled, the provided data might not be in a complete state i.e. <code>HttpData#isCompleted()</code> has to be checked. Also note that enabling this property effectively ignores <code>maxInMemorySize</code> , <code>baseDirectory</code> , and <code>scheduler</code> .	<p>=====</p> <p>==== Obtaining the Remote (Client) Address</p> <p>In addition to the metadata that you can obtain from the request, you can also receive the <code>host (server)</code> address, the <code>remote (client)</code> address and the <code>scheme</code>. Depending on the chosen factory method, you can retrieve the information directly from the channel or by using the <code>Forwarded</code> or <code>X-Forwarded- HTTP</code> request headers. The following example shows how to do so:</p> <p>====</p> <pre>[source,java,indent=0] .{examplesdir}/clientaddress/Application.java ---- Unresolved directive in http- server.adoc - include::{examples dir}/clientaddress /Application.java[lines=18..38] ---- <1> Specifies that the information about the connection is to be obtained from the Forwarded and X-Forwarded- HTTP request headers, if possible. <2></pre>	=====

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Callback	Description	<code>doOnBind</code>	Invoked when the server channel is about to bind.	<code>doOnBound</code>
Invoked when the server channel is bound.	<code>doOnChannelInit</code>	Invoked when initializing the channel.	<code>doOnConnection</code>	Invoked when a remote client is connected

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
doOnUnbound	Invoked when the server channel is unbound.	<p>=====</p> <p>The following example uses the <code>doOnConnection</code> and <code>doOnChannelInit</code> callbacks:</p> <p>====</p> <pre>[source,java,indent=0] .{examplesdir}/lifecycle/Application.java ---- Unresolved directive in http-server.adoc - include::{examplesdir}/lifecycle/Application.java[lines=18..39] ---- <1> Netty pipeline is extended with ReadTimeoutHandler when a remote client is connected. <2> Netty pipeline is extended with LoggingHandler when initializing the channel. =====</pre> <p>== TCP-level Configuration</p> <p>When you need to change configuration on the TCP level, you can use the following snippet to extend the default <code>TCP</code> server configuration:</p> <p>====</p> <pre>[source,java,indent=0]</pre>	=====	metric name
		<pre>[source,java,indent=0]</pre>		

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
type	description	reactor.netty.http.server.streams.active	Gauge	The number of active HTTP/2 streams. See [observability-metrics-streams-active]
reactor.netty.http.server.connections.active	Gauge	The number of http connections currently processing requests. See [observability-metrics-connections-active]	reactor.netty.http.server.connections.total	Gauge
The number of all opened connections. See [observability-metrics-connections-total]	reactor.netty.http.server.data.received	DistributionSummary	Amount of the data received, in bytes. See [observability-metrics-data-received]	reactor.netty.http.server.data.sent
DistributionSummary	Amount of the data sent, in bytes. See [observability-metrics-data-sent]	reactor.netty.http.server.errors	Counter	Number of errors that occurred. See [observability-metrics-errors-count]
reactor.netty.http.server.data.received.time	Timer	Time spent in consuming incoming data. See [observability-metrics-http-server-data-received-time]	reactor.netty.http.server.data.sent.time	Timer

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Time spent in sending outgoing data. See [observability-metrics-http-server-data-sent-time]	reactor.netty.http.server.response.time	Timer	Total time for the request/response See [observability-metrics-http-server-response-time]	<p>=====</p> <p>These additional metrics are also available:</p> <p>ByteBufAllocator metrics</p> <p>[width="100%",options="header"]</p>
=====	metric name	type	description	reactor.netty.bytebuf.allocator.used.heap.memory
Gauge	The number of bytes reserved by heap buffer allocator. See [observability-metrics-used-heap-memory]	reactor.netty.bytebuf.allocator.used.direct.memory	Gauge	The number of bytes reserved by direct buffer allocator. See [observability-metrics-used-direct-memory]
reactor.netty.bytebuf.allocator.heap.arenas	Gauge	The number of heap arenas (when PooledByteBufAllocator). See [observability-metrics-heap-arenas]	reactor.netty.bytebuf.allocator.direct.arenas	Gauge
The number of direct arenas (when PooledByteBufAllocator). See [observability-metrics-direct-arenas]	reactor.netty.bytebuf.allocator.threadlocal.caches	Gauge	The number of thread local caches (when PooledByteBufAllocator). See [observability-metrics-thread-local-caches]	reactor.netty.bytebuf.allocator.small.cache.size

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Gauge	The size of the small cache (when <code>PooledByteBufAllocator</code>). See [observability-metrics-small-cache-size]	reactor.netty.bytebuf.allocator.normal.cache.size	Gauge	The size of the normal cache (when <code>PooledByteBufAllocator</code>). See [observability-metrics-normal-cache-size]
reactor.netty.bytebuf.allocator.chunk.size	Gauge	The chunk size for an arena (when <code>PooledByteBufAllocator</code>). See [observability-metrics-chunk-size]	reactor.netty.bytebuf.allocator.active.heap.memory	Gauge
The actual bytes consumed by in-use buffers allocated from heap buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-heap-memory]	reactor.netty.bytebuf.allocator.active.direct.memory	Gauge	The actual bytes consumed by in-use buffers allocated from direct buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-direct-memory]	===== <code>EventLoop</code> metrics [width="100%",options="header"]
=====	metric name	type	description	reactor.netty.eventloop.pending.tasks

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Gauge	The number of tasks that are pending for processing on an event loop. See [observability-metrics-pending-tasks]	<p>=====</p> <p>The following example enables that integration:</p> <p>====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/Application.java ---- Unresolved directive in http-server.adoc - include:::{examplesdir}/metrics/Application.java[lines=18..52] ---- <1> Applies upper limit for the meters with URI tag <2> Templated URIs will be used as an URI tag value when possible <3> Enables the built-in integration with Micrometer =====</pre> <p>NOTE: In order to avoid a memory and CPU overhead of the enabled metrics, it is important to convert the real URIs to templated URIs when possible. Without a conversion to a template-like form, each distinct URI leads to the creation of a distinct tag, which takes a lot of memory for the</p>	=====	contextual name
		memory for the		

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
description	<HTTP METHOD>_<URI>	Information and total time for the request. See [observability-spans-http-server-response-span] .	<p>=====</p> <p>The following example enables that integration. This concrete example uses Brave and reports the information to Zipkin. See the Micrometer Tracing documentation for OpenTelemetry setup.</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/tracing/Application.java ---- Unresolved directive in http-server.adoc - include::{examplesdir}/tracing/Application.java[lines=18..91] ---- <1> Initializes Brave, Zipkin, and the Observation registry. <2> Templated URIs are used as an URI tag value when possible. <3> Enables the built-in integration with Micrometer. =====</pre> <p>The result in Zipkin looks like:</p> <p>image::images/http-server-tracing.png[]</p> <p>=== Access Current Observation</p>	302
44				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
303	307	<p>308`. When it is 303 status code, GET method is used for the redirect. * followRedirect(BiPredicate<HttpClientRequest, HttpClientResponse>): Enables auto-redirect support if the supplied predicate matches.</p> <p>The following example uses followRedirect(true):</p> <p>====</p> <p>[source,java,indent=0] .{examplesdir}/redirect/Application.java ---- Unresolved directive in http-client.adoc - include:::{examplesdir}/redirect/Application.java[lines=18..32] ---- =====</p> <p>== Consuming Data</p> <p>To receive data from a given HTTP endpoint, you can use one of the methods from {javadoc}/reactor/netty/http/client/HttpClient.ResponseReceiver.html[HttpClient.ResponseReceiver]. The following example uses the responseContent method:</p>	=====	Callback
				45

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Description	<code>doAfterRequest</code>	Invoked when the request has been sent.	<code>doAfterResolve</code>	Invoked after the remote address has been resolved successfully.
<code>doAfterResponseSuccess</code>	Invoked after the response has been fully received.	<code>doOnChannelInit</code>	Invoked when initializing the channel.	<code>doOnConnect</code>
Invoked when the channel is about to connect.	<code>doOnConnected</code>	Invoked after the channel has been connected.	<code>doOnDisconnected</code>	Invoked after the channel has been disconnected.
<code>doOnError</code>	Invoked when the request has not been sent and when the response has not been fully received.	<code>doOnRedirect</code>	Invoked when the response headers have been received, and the request is about to be redirected.	<code>doOnRequest</code>
Invoked when the request is about to be sent.	<code>doOnRequestError</code>	Invoked when the request has not been sent.	<code>doOnResolve</code>	Invoked when the remote address is about to be resolved.
<code>doOnResolveError</code>	Invoked in case the remote address hasn't been resolved successfully.	<code>doOnResponse</code>	Invoked after the response headers have been received.	<code>doOnResponseError</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<p>Invoked when the response has not been fully received.</p>	<p>=====</p> <p>The following example uses the <code>doOnConnected</code> and <code>doOnChannelInit</code> callbacks:</p> <p>=====</p> <p>[source,java,indent=0]</p> <p> <code> .{examplesdir}/lifecycle/Application.java --- Unresolved directive in http-client.adoc - include::{examplesdir}/lifecycle/Application.java[lines= 18..39] --- <1> Netty pipeline is extended with ReadTimeoutHandler when the channel has been connected. <2> Netty pipeline is extended with LoggingHandler when initializing the channel. ===== == TCP-level Configuration When you need configurations on a TCP level, you can use the following snippet to extend the default TCP client configuration (add an option, bind address etc.): ===== [source,java,indent=0] </code> </p>	=====	Configuration name	Description
	<p>=====</p> <p>[source,java,indent=0]</p>			

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<code>disposeInactivePoolsInBackground</code>	<p>When this option is enabled, connection pools are regularly checked in the background, and those that are empty and been inactive for a specified time become eligible for disposal. Connection pool is considered empty when there are no active connections, idle connections and pending acquisitions. By default, this background disposal of inactive pools is disabled.</p>	<code>disposeTimeout</code>	<p>When <code>ConnectionProvider#dispose()</code> or <code>ConnectionProvider#disposeLater()</code> is called, trigger a graceful shutdown for the connection pools, with this grace period timeout. From there on, all calls for acquiring a connection will fail fast with an exception. However, for the provided <code>Duration</code>, pending acquires will get a chance to be served. Note: The rejection of new acquires and the grace timer start immediately, irrespective of subscription to the <code>Mono</code> returned by <code>ConnectionProvider#disposeLater()</code>. Subsequent calls return the same <code>Mono</code>, effectively getting notifications from the first graceful shutdown call and ignoring subsequently provided timeouts. By default, dispose timeout is not specified.</p>	<code>evictInBackground</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
When this option is enabled, each connection pool regularly checks for connections that are eligible for removal according to eviction criteria like <code>maxIdleTime</code> . By default, this background eviction is disabled.	<code>fifo</code>	Configure the connection pool so that if there are idle connections (i.e. pool is under-utilized), the next acquire operation will get the <code>Least Recently Used</code> connection (LRU, i.e. the connection that was released first among the current idle connections). Default leasing strategy.	<code>lifo</code>	Configure the connection pool so that if there are idle connections (i.e. pool is under-utilized), the next acquire operation will get the <code>Most Recently Used</code> connection (MRU, i.e. the connection that was released last among the current idle connections).
<code>maxConnections</code>	The maximum number of connections (per connection pool) before start pending. Default to 2 * available number of processors (but with a minimum value of 16).	<code>maxIdleTime</code>	The time after which the channel is eligible to be closed when idle (resolution: ms). Default: max idle time is not specified.	<code>maxLifeTime</code>
The total life time after which the channel is eligible to be closed (resolution: ms). Default: max life time is not specified.	<code>metrics</code>	Enables/disables built-in integration with Micrometer. <code>ConnectionProvider.MeterRegistrar</code> can be provided for integration with another metrics system. By default, metrics are not enabled.	<code>pendingAcquireMaxCount</code>	The maximum number of extra attempts at acquiring a connection to keep in a pending queue. If -1 is specified, the pending queue does not have upper limit. Default to 2 * max connections.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
pendingAcquireTimeout	The maximum time before which a pending acquire must complete, or a TimeoutException is thrown (resolution: ms). If -1 is specified, no such timeout is applied. Default: 45 seconds.	<p>=====</p> <p>NOTE: When you expect a high load, be cautious with a connection pool with a very high value for maximum connections. You might experience <code>reactor.netty.http.client.PrematureCloseException</code> exception with a root cause "Connect Timeout" due to too many concurrent connections opened/acquired.</p> <p>If you need to disable the connection pool, you can apply the following configuration:</p> <p>====</p> <pre>[source,java,indent=0] .{examplesdir}/pool/Application.java ---- Unresolved directive in http-client-conn-provider.adoc - include:::{examplesdir}/pool/Application.java[lines=18..49] ---- =====</pre> <p>=== Disposing Connection Pool</p> <p>- If you use the default</p>	=====	metric name
50				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
type	description	reactor.netty.connection.provider.total.connections	Gauge	The number of all connections, active or idle. See [observability-metrics-total-connections]
reactor.netty.connection.provider.active.connections	Gauge	The number of the connections that have been successfully acquired and are in active use. See [observability-metrics-active-connections]	reactor.netty.connection.provider.max.connections	Gauge
The maximum number of active connections that are allowed. See [observability-metrics-max-connections]	reactor.netty.connection.provider.idle.connections	Gauge	The number of the idle connections. See [observability-metrics-idle-connections]	reactor.netty.connection.provider.pending.connections
Gauge	The number of requests that are waiting for a connection. See [observability-metrics-pending-connections]	reactor.netty.connection.provider.pending.connections.time	Timer	Time spent in pending acquire a connection from the connection pool. See [observability-metrics-pending-connections-time]
reactor.netty.connection.provider.max.pending.connections	Gauge	The maximum number of requests that will be queued while waiting for a ready connection. See [observability-metrics-max-pending-connections]	===== The following table provides information for the HTTP client metrics when it is configured to serve HTTP/2 traffic: [width="100%",options="header"]	=====

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
metric name	type	description	reactor.netty.connection.provider.active.streams	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
The number of the active HTTP/2 streams. See [observability-metrics-active-streams]	reactor.netty.connection.provider.pending.streams	Gauge	The number of requests that are waiting for opening HTTP/2 stream. See [observability-metrics-pending-streams]	<p>=====</p> <p>The following example enables that integration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/pool/metrics/Application.java ---- Unresolved directive in http-client-connection.provider.adoc - include::{examplesdir}/pool/metrics/Application.java[lines=18..45] ---- <1> Enables the built-in integration with Micrometer =====</pre> <p>== SSL and TLS</p> <p>When you need SSL or TLS, you can apply the configuration shown in the next example. By default, if OpenSSL is available, a <code>{nettyjavadoc}/io/netty/handler/ssl/SSLProvider.html#OPENSSL[SslProvider.OPENSSL]</code> provider is used as a provider. Otherwise, a <code>{nettyjavadoc}/io/netty/handler/ssl/SSLProvider.html#JDK[SslProvider.JDK]</code> provider is used</p> <p>You can switch the provider by using⁵³</p>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
=====	metric name	type	description	reactor.netty.http.client.data.received
DistributionSummary	Amount of the data received, in bytes. See [observability-metrics-data-received]	reactor.netty.http.client.data.sent	DistributionSummary	Amount of the data sent, in bytes. See [observability-metrics-data-sent]
reactor.netty.http.client.errors	Counter	Number of errors that occurred. See [observability-metrics-errors-count]	reactor.netty.http.client.tls.handshake.time	Timer
Time spent for TLS handshake. See [observability-metrics-tls-handshake-time]	reactor.netty.http.client.connect.time	Timer	Time spent for connecting to the remote address. See [observability-metrics-connect-time]	reactor.netty.http.client.address.resolver
Timer	Time spent for resolving the address. See [observability-metrics-hostname-resolution-time]	reactor.netty.http.client.data.received.time	Timer	Time spent in consuming incoming data. See [observability-metrics-http-client-data-received-time]
reactor.netty.http.client.data.sent.time	Timer	Time spent in sending outgoing data. See [observability-metrics-http-client-data-sent-time]	reactor.netty.http.client.response.time	Timer

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Total time for the request/response See [observability-metrics-http-client-response-time]	===== These additional metrics are also available: Pooled ConnectionProvider metrics [width="100%",options="header"]	=====	metric name	type
description	reactor.netty.connection.provider.total.connections	Gauge	The number of all connections, active or idle. See [observability-metrics-total-connections]	reactor.netty.connection.provider.active.connections
Gauge	The number of the connections that have been successfully acquired and are in active use. See [observability-metrics-active-connections]	reactor.netty.connection.provider.max.connections	Gauge	The maximum number of active connections that are allowed. See [observability-metrics-max-connections]
reactor.netty.connection.provider.idle.connections	Gauge	The number of the idle connections. See [observability-metrics-idle-connections]	reactor.netty.connection.provider.pending.connections	Gauge
The number of requests that are waiting for a connection. See [observability-metrics-pending-connections]	reactor.netty.connection.provider.pending.connections.time	Timer	Time spent in pending acquire a connection from the connection pool. See [observability-metrics-pending-connections-time]	reactor.netty.connection.provider.max.pending.connections

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Gauge	The maximum number of requests that will be queued while waiting for a ready connection. See [observability-metrics-max-pending-connections]	<p>=====</p> <p>The following table provides information for the HTTP client metrics when it is configured to serve HTTP/2 traffic:</p> <p>[width="100%",options="header"]</p>	=====	metric name
type	description	reactor.netty.connection.provider.active.streams	Gauge	The number of the active HTTP/2 streams. See [observability-metrics-active-streams]
reactor.netty.connection.provider.pending.streams	Gauge	The number of requests that are waiting for opening HTTP/2 stream. See [observability-metrics-pending-streams]	<p>=====</p> <p>ByteBufAllocator metrics</p> <p>[width="100%",options="header"]</p>	=====
metric name	type	description	reactor.netty.bytebuf.allocator.used.heap.memory	Gauge
The number of bytes reserved by heap buffer allocator. See [observability-metrics-used-heap-memory]	reactor.netty.bytebuf.allocator.used.direct.memory	Gauge	The number of bytes reserved by direct buffer allocator. See [observability-metrics-used-direct-memory]	reactor.netty.bytebuf.allocator.heap.arenas

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Gauge	The number of heap arenas (when <code>PooledByteBufAllocator</code>). See [observability-metrics-heap-arenas]	<code>reactor.netty.bytebuf.allocator.direct.arenas</code>	Gauge	The number of direct arenas (when <code>PooledByteBufAllocator</code>). See [observability-metrics-direct-arenas]
<code>reactor.netty.bytebuf.allocator.threadlocal.caches</code>	Gauge	The number of thread local caches (when <code>PooledByteBufAllocator</code>). See [observability-metrics-thread-local-caches]	<code>reactor.netty.bytebuf.allocator.small.cache.size</code>	Gauge
The size of the small cache (when <code>PooledByteBufAllocator</code>). See [observability-metrics-small-cache-size]	<code>reactor.netty.bytebuf.allocator.normal.cache.size</code>	Gauge	The size of the normal cache (when <code>PooledByteBufAllocator</code>). See [observability-metrics-normal-cache-size]	<code>reactor.netty.bytebuf.allocator.chunk.size</code>
Gauge	The chunk size for an arena (when <code>PooledByteBufAllocator</code>). See [observability-metrics-chunk-size]	<code>reactor.netty.bytebuf.allocator.active.heap.memory</code>	Gauge	The actual bytes consumed by in-use buffers allocated from heap buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-heap-memory]

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
reactor.netty.bytebuf allocator.active.direct.memory	Gauge	The actual bytes consumed by in-use buffers allocated from direct buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-direct-memory]	===== <code>EventLoop</code> metrics [width="100%",options="header"]	=====
metric name	type	description	reactor.netty.eventloop.pending.tasks	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<p>The number of tasks that are pending for processing on an event loop. See [observability-metrics-pending-tasks]</p>	<p>=====</p> <p>The following example enables that integration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/Application.java ---- Unresolved directive in http-client.adoc - include::{examplesdir}/metrics/Application.java[lines=18..51] ---- <1> Applies upper limit for the meters with URI tag <2> Templated URIs will be used as a URI tag value when possible <3> Enables the built-in integration with Micrometer =====</pre> <p>NOTE: In order to avoid a memory and CPU overhead of the enabled metrics, it is important to convert the real URIs to templated URIs when possible. Without a conversion to a template-like form, each distinct URI leads to the creation of a distinct tag, which takes a lot of memory for the metrics.</p>	=====	contextual name	description
				59

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
HTTP <HTTP METHOD>	Information and total time for the request. See [observability-spans-http-client-response-span] .	hostname resolution	Information and time spent for resolving the address. See [observability-spans-hostname-resolution-span] .	connect

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Information and time spent for connecting to the remote address. See [observability-spans-connect-span] .	tls handshake	Information and time spent for TLS handshake. See [observability-spans-tls-handshake-span] .	<p>=====</p> <p>The following example enables that integration. This concrete example uses Brave and reports the information to Zipkin. See the Micrometer Tracing documentation for OpenTelemetry setup.</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/tracing/Application.java ---- Unresolved directive in http-client.adoc - include::{examplesdir}/tracing/Application.java[lines=18..90] ---- <1> Initializes Brave, Zipkin, and the Observation registry. <2> Templated URIs are used as an URI tag value when possible. <3> Enables the built-in integration with Micrometer. =====</pre> <p>The result in Zipkin looks like:</p> <p>image::images/http-client-tracing.png[]</p> <p>=== Access Current Observation</p>	=====
				61

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Configuration name	Description	<code>bindAddressSupplier</code>	The supplier of the local address to bind to.	<code>cacheMaxTimeToLive</code>
The max time to live of the cached DNS resource records (resolution: seconds). If the time to live of the DNS resource record returned by the DNS server is greater than this max time to live, this resolver ignores the time to live from the DNS server and uses this max time to live. Default to <code>Integer.MAX_VALUE</code> .	<code>cacheMinTimeToLive</code>	The min time to live of the cached DNS resource records (resolution: seconds). If the time to live of the DNS resource record returned by the DNS server is less than this min time to live, this resolver ignores the time to live from the DNS server and uses this min time to live. Default: 0.	<code>cacheNegativeTimeToLive</code>	The time to live of the cache for the failed DNS queries (resolution: seconds). Default: 0.
<code>completeOncePreferredResolved</code>	When this setting is enabled, the resolver notifies as soon as all queries for the preferred address type are complete. When this setting is disabled, the resolver notifies when all possible address types are complete. This configuration is applicable for <code>DnsNameResolver#resolveAll(String)</code> . By default, this setting is enabled.	<code>disableOptionalRecord</code>	Disables the automatic inclusion of an optional record that tries to give a hint to the remote DNS server about how much data the resolver can read per response. By default, this setting is enabled.	<code>disableRecursionDesired</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Specifies whether this resolver has to send a DNS query with the recursion desired (RD) flag set. By default, this setting is enabled.	<code>DnsAddressResolverGroupProvider</code>	Sets a custom function to create a <code>DnsAddressResolverGroup</code> given a <code>DnsNameResolverBuilder</code>	<code>hostsFileEntriesResolver</code>	Sets a custom <code>{nettyjavadoc}/io/netty/resolver/HostsFileEntriesResolver.html[HostsFileEntriesResolver]</code> to be used for hosts file entries. Default: <code>{nettyjavadoc}/io/netty/resolver/DefaultHostsFileEntriesResolver.html[DefaultHostsFileEntriesResolver]</code> .
<code>maxPayloadSize</code>	Sets the capacity of the datagram packet buffer (in bytes). Default: 4096.	<code>maxQueriesPerResolve</code>	Sets the maximum allowed number of DNS queries to send when resolving a host name. Default: 16.	<code>ndots</code>
Sets the number of dots that must appear in a name before an initial absolute query is made. Default: -1 (to determine the value from the OS on Unix or use a value of 1 otherwise).	<code>queryTimeout</code>	Sets the timeout of each DNS query performed by this resolver (resolution: milliseconds). Default: 5000.	<code>resolveCache</code>	The cache to use to store resolved DNS entries.
<code>resolvedAddressTypes</code>	The list of the protocol families of the resolved address.	<code>retryTcpOnTimeout</code>	Specifies whether this resolver will also fallback to TCP if a timeout is detected. By default, the resolver will only try to use TCP if the response is marked as truncated.	<code>roundRobinSelection</code>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Enables an {nettyjavadoc}/io/netty/resolver/AddressResolverGroup.html[AddressResolverGroup] of {nettyjavadoc}/io/netty/resolver/dns/DnsNameResolver.html[DnsNameResolver] that supports random selection of destination addresses if multiple are provided by the nameserver. See {nettyjavadoc}/io/netty/resolver/dns/RoundRobinDnsAddressResolverGroup.html[RoundRobinDnsAddressResolverGroup]. Default: {nettyjavadoc}/io/netty/resolver/dns/DnsAddressResolverGroup.html[DnsAddressResolverGroup]	runOn	Performs the communication with the DNS servers on the given {javadoc}/reactor/netty/resources/LoopResources.html[LoopResources]. By default, the LoopResources specified on the client level are used.	searchDomains	The list of search domains of the resolver. By default, the effective search domain list is populated by using the system DNS search domains.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
trace	A specific logger and log level to be used by this resolver when generating detailed trace information in case of resolution failure.	<p>=====</p> <p>Sometimes, you may want to switch to the JVM built-in resolver. To do so, you can configure the <code>HttpClient</code> as follows:</p> <p>=====</p> <p>[source,java,indent=0] .{examplesdir}/resolver/custom/Application.java ---- Unresolved directive in http-client.adoc - include::{examplesdir}/resolver/custom/Application.java[lines=18..38] ---- <1> Sets the JVM built-in resolver.</p> <p>=====</p> <p>== Timeout Configuration This section describes various timeout configuration options that can be used in <code>HttpClient</code>. Configuring a proper timeout may improve or solve issues in the communication process. The configuration options can be grouped as follows:</p>	=====	Callback
		* [connection-		

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
Description	doOnBind	Invoked when the server channel is about to bind.	doOnBound	Invoked when the server channel is bound.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<code>doOnChannelInit</code>	Invoked when initializing the channel.	<code>doOnUnbound</code>	Invoked when the server channel is unbound.	<p>=====</p> <p>The following example uses the <code>doOnBound</code> and <code>doOnChannelInit</code> callbacks:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/life cycle/Application.j ava ---- Unresolved directive in udp- server.adoc - include::{example sdir}/lifecycle/App lication.java[lines= 18..39] ---- <1> Netty pipeline is extended with LineBasedFrameDeco der when the server channel is bound. <2> Netty pipeline is extended with LoggingHandler when initializing the channel. =====</pre> <p>== Connection Configuration</p> <p>This section describes three kinds of configuration that you can use at the UDP level:</p> <p>* [server-udp-connection-configurations-channel-options] *</p>
				<p>[server-udp-connection-</p>

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
=====	metric name	type	description	reactor.netty.udp.server.data.received
DistributionSummary	Amount of the data received, in bytes. See [observability-metrics-data-received]	reactor.netty.udp.server.data.sent	DistributionSummary	Amount of the data sent, in bytes. See [observability-metrics-data-sent]
reactor.netty.udp.server.errors	Counter	Number of errors that occurred. See [observability-metrics-errors-count]	===== <p>These additional metrics are also available:</p> <p>ByteBufAllocator metrics</p> <p>[width="100%",options="header"]</p>	=====
metric name	type	description	reactor.netty.bytebuf.allocator.used.heap.memory	Gauge
The number of bytes reserved by heap buffer allocator. See [observability-metrics-used-heap-memory]	reactor.netty.bytebuf.allocator.used.direct.memory	Gauge	The number of bytes reserved by direct buffer allocator. See [observability-metrics-used-direct-memory]	reactor.netty.bytebuf.allocator.heap.arenas
Gauge	The number of heap arenas (when PooledByteBufAllocator). See [observability-metrics-heap-arenas]	reactor.netty.bytebuf.allocator.direct.arenas	Gauge	The number of direct arenas (when PooledByteBufAllocator). See [observability-metrics-direct-arenas]

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
reactor.netty.bytebuf allocator.threadlocal.caches	Gauge	The number of thread local caches (when PooledByteBufAllocator). See [observability-metrics-thread-local-caches]	reactor.netty.bytebuf allocator.small.cache.size	Gauge
The size of the small cache (when PooledByteBufAllocator). See [observability-metrics-small-cache-size]	reactor.netty.bytebuf allocator.normal.cache.size	Gauge	The size of the normal cache (when PooledByteBufAllocator). See [observability-metrics-normal-cache-size]	reactor.netty.bytebuf allocator.chunk.size
Gauge	The chunk size for an arena (when PooledByteBufAllocator). See [observability-metrics-chunk-size]	reactor.netty.bytebuf allocator.active.heap.memory	Gauge	The actual bytes consumed by in-use buffers allocated from heap buffer pools (when PooledByteBufAllocator). See [observability-metrics-active-heap-memory]
reactor.netty.bytebuf allocator.active.direct.memory	Gauge	The actual bytes consumed by in-use buffers allocated from direct buffer pools (when PooledByteBufAllocator). See [observability-metrics-active-direct-memory]	===== EventLoop metrics [width="100%",options="header"]	=====
metric name	type	description	reactor.netty.eventloop.pending.tasks	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<p>The number of tasks that are pending for processing on an event loop. See [observability-metrics-pending-tasks]</p>	<p>=====</p> <p>The following example enables that integration:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/Application.java ---- Unresolved directive in udp-server.adoc - include::{examplesdir}/metrics/Application.java[lines=18..34] ---- <1> Enables the built-in integration with Micrometer =====</pre> <p>When UDP server metrics are needed for an integration with a system other than Micrometer or you want to provide your own integration with Micrometer, you can provide your own metrics recorder, as follows:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/metrics/custom/Application.java ---- Unresolved directive in udp-server.adoc - include::{examplesdir}/metrics/custo</pre>	=====	Callback	Description
70				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<code>doAfterResolve</code>	Invoked after the remote address has been resolved successfully.	<code>doOnChannelInit</code>	Invoked when initializing the channel.	<code>doOnConnect</code>
Invoked when the channel is about to connect.	<code>doOnConnected</code>	Invoked after the channel has been connected.	<code>doOnDisconnected</code>	Invoked after the channel has been disconnected.

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
<code>doOnResolve</code>	Invoked when the remote address is about to be resolved.	<code>doOnResolveError</code>	Invoked in case the remote address hasn't been resolved successfully.	<p>=====</p> <p>The following example uses the <code>doOnConnected</code> and <code>doOnChannelInit</code> callbacks:</p> <p>=====</p> <pre>[source,java,indent=0] .{examplesdir}/life cycle/Application.j ava ---- Unresolved directive in udp- client.adoc - include::{example sdir}/lifecycle/App lication.java[lines= 18..40] ---- <1> Netty pipeline is extended with LineBasedFrameDeco der when the channel has been connected. <2> Netty pipeline is extended with LoggingHandler when initializing the channel. =====</pre> <p>== Connection Configuration</p> <p>This section describes three kinds of configuration that you can use at the UDP level:</p> <p>* [client-udp-connection-configurations-channel-options] *</p> <p>[client-udp-connection-</p>
72				

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
=====	metric name	type	description	reactor.netty.udp.c lient.data.received
DistributionSummary	Amount of the data received, in bytes. See [observability-metrics-data-received]	reactor.netty.udp.c lient.data.sent	DistributionSummary	Amount of the data sent, in bytes. See [observability-metrics-data-sent]
reactor.netty.udp.c lient.errors	Counter	Number of errors that occurred. See [observability-metrics-errors-count]	reactor.netty.udp.c lient.connect.time	Timer
Time spent for connecting to the remote address. See [observability-metrics-connect-time]	reactor.netty.udp.c lient.address.resolver	Timer	Time spent for resolving the address. See [observability-metrics-hostname-resolution-time]	===== <p>These additional metrics are also available:</p> <p>ByteBufAllocator metrics</p> <p>[width="100%",options="header"]</p>
=====	metric name	type	description	reactor.netty.byte buf.allocator.used. heap.memory
Gauge	The number of bytes reserved by heap buffer allocator. See [observability-metrics-used-heap-memory]	reactor.netty.byte buf.allocator.used. direct.memory	Gauge	The number of bytes reserved by direct buffer allocator. See [observability-metrics-used-direct-memory]
reactor.netty.byte buf.allocator.heap. arenas	Gauge	The number of heap arenas (when PooledByteBufAllocator). See [observability-metrics-heap-arenas]	reactor.netty.byte buf.allocator.direc t.arenas	Gauge

Version	Initial OSS Release	OSS Support End	Commercial Support (*) End	Published in BOM
The number of direct arenas (when <code>PooledByteBufAllocator</code>). See [observability-metrics-direct-arenas]	reactor.netty.bytebuf.allocator.threadlocal.caches	Gauge	The number of thread local caches (when <code>PooledByteBufAllocator</code>). See [observability-metrics-thread-local-caches]	reactor.netty.bytebuf.allocator.small.cache.size
Gauge	The size of the small cache (when <code>PooledByteBufAllocator</code>). See [observability-metrics-small-cache-size]	reactor.netty.bytebuf.allocator.normal.cache.size	Gauge	The size of the normal cache (when <code>PooledByteBufAllocator</code>). See [observability-metrics-normal-cache-size]
reactor.netty.bytebuf.allocator.chunk.size	Gauge	The chunk size for an arena (when <code>PooledByteBufAllocator</code>). See [observability-metrics-chunk-size]	reactor.netty.bytebuf.allocator.active.heap.memory	Gauge
The actual bytes consumed by in-use buffers allocated from heap buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-heap-memory]	reactor.netty.bytebuf.allocator.active.direct.memory	Gauge	The actual bytes consumed by in-use buffers allocated from direct buffer pools (when <code>PooledByteBufAllocator</code>). See [observability-metrics-active-direct-memory]	===== <code>EventLoop</code> metrics [width="100%",options="header"]
=====	metric name	type	description	reactor.netty.eventloop.pending.tasks