# Contributing guide

\*\*Want to contribute? Great!\*\*

We try to make it easy, and all contributions, even the smaller ones, are more than welcome.

This includes bug reports, fixes, documentation, examples...

But first, read this page (including the small print at the end).

## Legal

All original contributions to Quarkus are licensed under the

[ASL - Apache License](https://www.apache.org/licenses/LICENSE-2.0),

version 2.0 or later, or, if another license is specified as governing the file or directory being

modified, such other license.

All contributions are subject to the [Developer Certificate of Origin (DCO)](https://developercertificate.org/).

The DCO text is also included verbatim in the [dco.txt](dco.txt) file in the root directory of the repository.

## Reporting an issue

This project uses GitHub issues to manage the issues. Open an issue directly in GitHub.

If you believe you found a bug, and it's likely possible, please indicate a way to reproduce it, what you are seeing and what you would expect to see.

Don't forget to indicate your Quarkus, Java, Maven/Gradle and GraalVM version.

## Checking an issue is fixed in master

Sometimes a bug has been fixed in the `master` branch of Quarkus and you want to confirm it is fixed for your own application.

Testing the `master` branch is easy and you have two options:

\* either use the snapshots we publish daily on https://oss.sonatype.org/content/repositories/snapshots/

\* or build Quarkus all by yourself

This is a quick summary to get you to quickly test master.

If you are interested in having more details, refer to the [Build section](#build) and the [Usage section](#usage).

### Using snapshots

Snapshots are published daily so you will have to wait for a snapshot containing the commits you are interested in.

Then just add https://oss.sonatype.org/content/repositories/snapshots/ as a Maven repository \*\*and\*\* a plugin repository.

You can check the last publication date here: https://oss.sonatype.org/content/repositories/snapshots/io/quarkus/ .

### Building master

Just do the following:

```

git clone git@github.com:quarkusio/quarkus.git

cd quarkus

export MAVEN\_OPTS="-Xmx1563m"

./mvnw -Dquickly

```

Wait for a bit and you're done.

### Updating the version

Be careful, when using the `master` branch, you need to use the `quarkus-bom` instead of the `quarkus-universe-bom`.

Update both the versions of the `quarkus-bom` and the Quarkus Maven plugin to `999-SNAPSHOT`.

You can now test your application.

## Before you contribute

To contribute, use GitHub Pull Requests, from your \*\*own\*\* fork.

### Code reviews

All submissions, including submissions by project members, need to be reviewed before being merged.

### Coding Guidelines

\* We decided to disallow `@author` tags in the Javadoc: they are hard to maintain, especially in a very active project, and we use the Git history to track authorship. GitHub also has [this nice page with your contributions](https://github.com/quarkusio/quarkus/graphs/contributors). For each major Quarkus release, we also publish the list of contributors in the announcement post.

\* Commits should be atomic and semantic. Please properly squash your pull requests before submitting them. Fixup commits can be used temporarily during the review process but things should be squashed at the end to have meaningful commits.

We use merge commits so the GitHub Merge button cannot do that for us. If you don't know how to do that, just ask in your pull request, we will be happy to help!

### Continuous Integration

Because we are all humans, and to ensure Quarkus is stable for everyone, all changes must go through Quarkus continuous integration. Quarkus CI is based on GitHub Actions, which means that everyone has the ability to automatically execute CI in their forks as part of the process of making changes. We ask that all non-trivial changes go through this process, so that the contributor gets immediate feedback, while at the same time keeping our CI fast and healthy for everyone.

The process requires only one additional step to enable Actions on your fork (clicking the green button in the actions tab). [See the full video walkthrough](https://youtu.be/egqbx-Q-Cbg) for more details on how to do this.

### Tests and documentation are not optional

Don't forget to include tests in your pull requests.

Also don't forget the documentation (reference documentation, javadoc...).

Be sure to test your pull request in:

1. Java mode

2. Native mode

## Setup

If you have not done so on this machine, you need to:

\* Install Git and configure your GitHub access

\* Install Java SDK (OpenJDK recommended)

\* Install [GraalVM](https://quarkus.io/guides/building-native-image)

\* Install platform C developer tools:

\* Linux

\* Make sure headers are available on your system (you'll hit 'Basic header file missing (<zlib.h>)' error if they aren't).

\* On Fedora `sudo dnf install zlib-devel`

\* Otherwise `sudo apt-get install libz-dev`

\* macOS

\* `xcode-select --install`

\* Set `GRAALVM\_HOME` to your GraalVM Home directory e.g. `/opt/graalvm` on Linux or `$location/JDK/GraalVM/Contents/Home` on macOS

Docker is not strictly necessary: it is used to run the MariaDB and PostgreSQL tests which are not enabled by default. However it is a recommended install if you plan to work on Quarkus JPA support:

\* Check [the installation guide](https://docs.docker.com/install/), and [the MacOS installation guide](https://docs.docker.com/docker-for-mac/install/)

\* If you just install docker, be sure that your current user can run a container (no root required).

On Linux, check [the post-installation guide](https://docs.docker.com/install/linux/linux-postinstall/)

### IDE Config and Code Style

Quarkus has a strictly enforced code style. Code formatting is done by the Eclipse code formatter, using the config files

found in the `independent-projects/ide-config` directory. By default when you run `./mvnw install` the code will be formatted automatically.

When submitting a pull request the CI build will fail if running the formatter results in any code changes, so it is

recommended that you always run a full Maven build before submitting a pull request.

If you want to run the formatting without doing a full build, you can run `./mvnw process-sources`.

#### Eclipse Setup

Open the \*Preferences\* window, and then navigate to \_Java\_ -> \_Code Style\_ -> \_Formatter\_. Click \_Import\_ and then

select the `eclipse-format.xml` file in the `independent-projects/ide-config` directory.

Next navigate to \_Java\_ -> \_Code Style\_ -> \_Organize Imports\_. Click \_Import\_ and select the `eclipse.importorder` file.

#### IDEA Setup

Open the \_Preferences\_ window (or \_Settings\_ depending on your edition) , navigate to \_Plugins\_ and install the [Eclipse Code Formatter Plugin](https://plugins.jetbrains.com/plugin/6546-eclipse-code-formatter) from the Marketplace.

Restart your IDE, open the \*Preferences\* (or \*Settings\*) window again and navigate to \_Other Settings\_ -> \_Eclipse Code Formatter\_.

Select \_Use the Eclipse Code Formatter\_, then change the \_Eclipse Java Formatter Config File\_ to point to the

`eclipse-format.xml` file in the `independent-projects/ide-config` directory. Make sure the \_Optimize Imports\_ box is ticked, and

select the `eclipse.importorder` file as the import order config file.

## Build

\* Clone the repository: `git clone https://github.com/quarkusio/quarkus.git`

\* Navigate to the directory: `cd quarkus`

\* Set Maven heap to 1.5GB `export MAVEN\_OPTS="-Xmx1563m"`

\* Invoke `./mvnw -Dquickly` from the root directory

```bash

git clone https://github.com/quarkusio/quarkus.git

cd quarkus

export MAVEN\_OPTS="-Xmx1563m"

./mvnw -Dquickly

# Wait... success!

```

This build skipped all the tests, native-image builds, documentation generation etc. and used the Maven goals `clean install` by default.

For more details about `-Dquickly` have a look at the `quick-build` profile in `quarkus-parent` (root `pom.xml`).

Adding `-DskipTests=false -DskipITs=false` enables the tests.

It will take much longer to build but will give you more guarantees on your code.

You can build and test native images in the integration tests supporting it by using `./mvnw install -Dnative`.

By default the build will use the native image server. This speeds up the build, but can cause problems due to the cache

not being invalidated correctly in some cases. To run a build with a new instance of the server you can use

`./mvnw install -Dnative-image.new-server=true`.

### Workflow tips

Due to Quarkus being a large repository, having to rebuild the entire project every time a change is made isn't very productive.

The following Maven tips can vastly speed up development when working on a specific extension.

#### Building all modules of an extension

Let's say you want to make changes to the `Jackson` extension. This extension contains the `deployment`, `runtime` and `spi` modules

which can all be built by executing following command:

```

./mvnw install -f extensions/jackson/

```

This command uses the path of the extension on the filesystem to identify it. Moreover, Maven will automatically build all modules in that path recursively.

#### Building a single module of an extension

Let's say you want to make changes to the `deployment` module of the Jackson extension. There are two ways to accomplish this task as shown by the following commands:

```

./mvnw install -f extensions/jackson/deployment

```

or

```

./mvnw install --projects 'io.quarkus:quarkus-jackson-deployment'

```

In this command we use the groupId and artifactId of the module to identify it.

#### Running a single test

Often you need to run a single test from some Maven module. Say for example you want to run the `GreetingResourceTest` of the `resteasy-jackson` Quarkus integration test (which can be found [here](https://github.com/quarkusio/quarkus/blob/master/integration-tests/resteasy-jackson)).

One way to accomplish this is by executing the following command:

```

./mvnw test -f integration-tests/resteasy-jackson/ -Dtest=GreetingResourceTest

```

## Usage

After the build was successful, the artifacts are available in your local Maven repository.

To include them into your project a few things have to be changed.

#### With Maven

\*pom.xml\*

```

<properties>

<quarkus-plugin.version>999-SNAPSHOT</quarkus-plugin.version>

<quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>

<quarkus.platform.group-id>io.quarkus</quarkus.platform.group-id>

<quarkus.platform.version>999-SNAPSHOT</quarkus.platform.version>

.

.

.

</properties>

```

#### With Gradle

\*gradle.properties\*

```

quarkusPlatformArtifactId=quarkus-bom

quarkusPluginVersion=999-SNAPSHOT

quarkusPlatformVersion=999-SNAPSHOT

quarkusPlatformGroupId=io.quarkus

```

\*settings.gradle\*

```

pluginManagement {

repositories {

mavenLocal() // add mavenLocal() to first position

jcenter()

mavenCentral()

gradlePluginPortal()

}

.

.

.

}

```

\*build.gradle\*

```

repositories {

mavenLocal() // add mavenLocal() to first position

jcenter()

mavenCentral()

}

```

### MicroProfile TCK's

Quarkus has a TCK module in `tcks` where all the MicroProfile TCK's are set up for you to run if you wish. These

include tests to areas like Config, JWT Authentication, Fault Tolerance, Health Checks, Metrics, OpenAPI, OpenTracing,

REST Client, Reactive Messaging and Context Propagation.

The TCK module is not part of the main Maven reactor build, but you can enable it and run the TCK tests by activating

the Maven Profile `-Ptcks`. If your work is related to any of these areas, running the TCK's is highly recommended to

make sure you are not breaking the project. The TCK's will also run on any Pull Request.

You can either run all of the TCK's or just a subset by executing `mvn verify` in the `tcks` module root or each of

the submodules. If you wish to run a particular test, you can use Maven `-Dtest=` property with the fully qualified

name of the test class and optionally the method name by using

`mvn verify -Dtest=fully.qualified.test.class.name#methodName`.

### Test Coverage

Quarkus uses Jacoco to generate test coverage. If you would like to generate the report run `mvn install -Ptest-coverage`,

then change into the `coverage-report` directory and run `mvn package`. The code coverage report will be generated in

`target/site/jacoco/`.

This currently does not work on Windows as it uses a shell script to copy all the classes and files into the code coverage

module.

## Extensions

### Descriptions

Extensions descriptions (in the `runtime/pom.xml` description or in the YAML `quarkus-extension.yaml`)

are used to describe the extension and are visible in https://code.quarkus.io.

Try and pay attention to it.

Here are a few recommendation guidelines:

- keep it relatively short so that no hover is required to read it

- describe the function over the technology

- use an action / verb to start the sentence

- do no conjugate the action verb (`Connect foo`, not `Connects foo` nor `Connecting foo`)

- connectors (JDBC / reactive) etc tend to start with Connect

- do not mention `Quarkus`

- do not mention `extension`

- avoid repeating the extension name

Bad examples and the corresponding good example:

- "AWS Lambda" (use "Write AWS Lambda functions")

- "Extension for building container images with Docker" (use "Build container images with Docker")

- "PostgreSQL database connector" (use "Connect to the PostgreSQL database via JDBC")

- "Asynchronous messaging for Reactive Streams" (use "Produce and consume messages and implement event driven and data streaming applications")

## The small print

This project is an open source project, please act responsibly, be nice, polite and enjoy!

## Frequently Asked Questions

\* The Maven build fails with `OutOfMemoryException`

Set Maven options to use 1.5GB of heap: `export MAVEN\_OPTS="-Xmx1563m"`.

\* IntelliJ fails to import Quarkus Maven project with `java.lang.OutOfMemoryError: GC overhead limit exceeded`

In IntelliJ IDEA (version older than `2019.2`) if you see problems in the Maven view claiming `java.lang.OutOfMemoryError: GC overhead limit exceeded` that means the project import failed.

To fix the issue, you need to update the Maven importing settings:

`Build, Execution, Deployment` > `Build Tools`> `Maven` > `Importing` > `VM options for importer`

To import Quarkus you need to define the JVM Max Heap Size (E.g. `-Xmx1g`)

\*\*Note\*\* As for now, we can't provide a unique Max Heap Size value. We have been reported to require from 768M to more than 3G to import Quarkus properly.

\* Build hangs with DevMojoIT running infinitely

```

./mvnw clean install

# Wait...

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.192 s - in io.quarkus.maven.it.GenerateConfigIT

[INFO] Running io.quarkus.maven.it.DevMojoIT

```

DevMojoIT require a few minutes to run but anything more than that is not expected. Make sure that nothing is running on 8080.

\* The native integration test for my extension didn't run in the CI

In the interest of speeding up CI, the native build stage `run\_native\_tests\_stage` have been split into multiple steps.

This means that each new extension needs to be configured explicitly in `azure-pipelines.yml` to have it's integration tests run in native mode