Red Hat build of Quarkus 3.2

Migrating applications to Red Hat build of Quarkus 3.2

Last Updated: 2024-04-04

Legal Notice

Abstract

This guide describes how to migrate applications from earlier versions of Red Hat build of Quarkus to the current version.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 4

CHAPTER 1. MIGRATING APPLICATIONS TO RED HAT BUILD OF QUARKUS 3.2 5
1.1. UPDATING PROJECTS TO THE LATEST RED HAT BUILD OF QUARKUS VERSION 5
1.1.1. Prerequisites 5
1.1.2. Procedure 5
1.2. CHANGES THAT AFFECT COMPATIBILITY WITH EARLIER VERSIONS 7
1.2.1. Cloud 7
1.2.1.1. Upgrade to the Kubernetes client that is included with Red Hat build of Quarkus 7
1.2.1.2. Improved logic for generating TLS-based container ports 7
1.2.1.3. Removal of some Kubernetes and OpenShift properties 7
1.2.2. Core 8
1.2.2.1. Upgrade to Jandex 3 8
1.2.2.2. Migration path for users of Jandex API 8
1.2.2.3. Removal of io.quarkus.arc.config.ConfigProperties annotation 9
1.2.2.4. Interceptor binding annotations declared on private methods now generate build failures 9
1.2.2.5. Removal of the @AlternativePriority annotation 9
1.2.2.6. Testing changes: Fixation of the Mockito subclass mockmaker 10
1.2.2.7. Update to the minimum supported Maven version 10
1.2.2.8. Removal of quarkus-bootstrap-maven-plugin 10
1.2.2.9. Mutiny 2 moves to Java Flow 10
1.2.3. Data 11
1.2.3.1. Removal of Hibernate ORM with Panache methods 11
1.2.3.2. Enhancement in Hibernate ORM: Automated IN clause parameter padding 11
1.2.3.3. New dependency: Hibernate Reactive 2 and Hibernate ORM 6.2 11
1.2.3.4. Hibernate Search changes 11
1.2.3.5. Hibernate Validator - Validation.buildDefaultValidatorFactory() now returns a ValidatorFactory managed by Quarkus 12
1.2.3.6. Quartz jobs class name change 12
1.2.3.7. Deprecation of QuarkusTransaction.run and QuarkusTransaction.call methods 13
1.2.3.8. Renamed Narayana transaction manager property 14
1.2.4. Messaging 14
1.2.4.1. Removal of vertx-kafka-client dependency from SmallRye Reactive Messaging 14
1.2.5. Native 14
1.2.5.1. Native compilation - Native executables and .so files 14
1.2.5.2. Native Compilation - Work around missing CPU features 15
1.2.5.3. Testing changes: Removal of some annotations 15
1.2.6. Observability 16
1.2.6.1. Deprecated OpenTracing driver is replaced by OpenTelemetry 16
1.2.6.2. Default metrics format in Micrometer now aligned with Prometheus 16
1.2.6.3. Changes in the OpenTelemetry extension and removal of some sampler-related properties 16
1.2.7. Security 19
1.2.7.1. Removal of CORS filter default support for using a wildcard as an origin 19
1.2.7.2. OpenAPI CORS support change 19
1.2.7.3. Encryption of OIDC session cookie by default 20
1.2.7.4. Default SameSite attribute set to Lax for OIDC session cookie 20
1.2.7.5. The OIDC ID token audience claim is verified by default 20
1.2.7.6. Removal of default password for the JWT key and keystore 21
1.2.8. Web 21
1.2.8.1. Changes to RESTEasy Reactive multipart 21
1.2.8.2. Enhanced JAXB extension control 22

1.3. ADDITIONAL RESOURCES 23

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. MIGRATING APPLICATIONS TO RED HAT BUILD OF QUARKUS 3.2

As an application developer, you can migrate applications that are based on earlier versions of Red Hat build of Quarkus to version 3.2 by using the Quarkus CLI's **update** command.

IMPORTANT

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the Quarkus CLI in production environments.

1.1. UPDATING PROJECTS TO THE LATEST RED HAT BUILD OF QUARKUS VERSION

You can update or upgrade your Red Hat build of Quarkus projects to the latest version by using an update command.

The **update** command primarily employs OpenRewrite recipes to automate updates for most project dependencies, source code, and documentation. Although these recipes perform many migration tasks, they do not cover all the tasks detailed in the migration guide.

Post-update, if expected updates are missing, consider the following reasons:

- The recipe applied by the **update** command might not include a migration task that your project requires.
- Your project might use an extension that is incompatible with the latest Red Hat build of Quarkus version.

IMPORTANT

For projects that use Hibernate ORM or Hibernate Reactive, review the Hibernate ORM 5 to 6 migration quick reference. The following update command covers only a subset of this guide.

1.1.1. Prerequisites

- Roughly 30 minutes
- An IDE
- JDK 11+ installed with **JAVA_HOME** configured appropriately
- Apache Maven 3.8.6 or later
- Optionally, the Red Hat build of Quarkus CLI if you want to use it
- Optionally Mandrel or GraalVM installed and configured appropriately if you want to build a native executable (or Docker if you use a native container build)
- A project based on Red Hat build of Quarkus version 2.13 or later.

1.1.2. Procedure

1. Create a working branch for your project by using your version control system.
2. To use the Red Hat build of Quarkus CLI in the next step, install the latest version of the Red Hat build of Quarkus CLI. Confirm the version number using **quarkus -v**.
3. Configure your extension registry client as described in the Configuring Red Hat build of Quarkus extension registry client section of the Quarkus "Getting Started" guide.
4. To update using the Red Hat build of Quarkus CLI, go to the project directory and update the project to the latest stream:

```
quarkus update
```

   Optional: By default, this command updates to the latest current version. To update to a specific stream instead of latest current version, add the **stream** option to this command followed by the version; for example: **--stream=3.2**

5. To update using Maven instead of the Red Hat build of Quarkus CLI, go to the project directory and update the project to the latest stream:
   a. Ensure that the Red Hat build of Quarkus Maven plugin version aligns with the latest supported Red Hat build of Quarkus version.
   b. Configure your project according to the guidelines provided in the Getting started with Quarkus guide.

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.2.11.Final-redhat-00001:update
```

      Optional: By default, this command updates to the latest current version. To update to a specific stream instead of latest current version, add the **stream** option to this command followed by the version; for example: **-Dstream=3.2**

6. Analyze the update command output for potential instructions and perform the suggested tasks if necessary.
7. Use a diff tool to inspect all changes.
8. Review the migration guide for items that were not updated by the update command. If your project has such items, implement the additional steps advised in these topics.
9. Ensure the project builds without errors, all tests pass, and the application functions as required before deploying to production.
10. Before deploying your updated Red Hat build of Quarkus application to production, ensure the following:
    - The project builds without errors.
    - All tests pass.
    - The application functions as required.

1.2. CHANGES THAT AFFECT COMPATIBILITY WITH EARLIER VERSIONS

This section describes changes in Red Hat build of Quarkus 3.2 that affect the compatibility of applications built with earlier product versions.

Review these breaking changes and take the steps required to ensure that your applications continue functioning after you update them to Red Hat build of Quarkus 3.2.

To automate many of these changes, use the **quarkus update** command to update your projects to the latest Red Hat build of Quarkus version.

1.2.1. Cloud

1.2.1.1. Upgrade to the Kubernetes client that is included with Red Hat build of Quarkus

The Kubernetes Client has been upgraded from 5.12 to 6.7.2. For more information, see the Kubernetes Client - Migration from 5.x to 6.x guide.

1.2.1.2. Improved logic for generating TLS-based container ports

Red Hat build of Quarkus 3.2 introduces changes in how the Kubernetes extension generates TLS-based container ports.

Earlier versions automatically added a container port named **https** to generated deployment resources. This approach posed problems, especially when SSL/TLS was not configured, rendering the port non-functional.

In 3.2 and later, the Kubernetes extension does not add a container port named **https** by default. The container port is only added if you take the following steps:

- You specify any relevant **quarkus.http.ssl.*** properties in your **application.properties** file.
- You set **quarkus.kubernetes.ports.https.tls=true** in your **application.properties** file.

1.2.1.3. Removal of some Kubernetes and OpenShift properties

With this 3.2 release, some previously deprecated Kubernetes and OpenShift-related properties have been removed. Replace them with their new counterparts.

Table 1.1. Removed properties and their new counterparts

Removed property	New property
quarkus.kubernetes.expose	**quarkus.kubernetes.ingress.expose**
quarkus.openshift.expose	**quarkus.openshift.route.expose**
quarkus.kubernetes.host	**quarkus.kubernetes.ingress.host**
quarkus.openshift.host	**quarkus.openshift.route.host**

Removed property	New property
quarkus.kubernetes.group	**quarkus.kubernetes.part-of**
quarkus.openshift.group	**quarkus.openshift.part-of**

Additionally, with this release, properties without the **quarkus.** prefix are ignored. For example, before this release, if you added a **kubernetes.name** property, it was mapped to **quarkus.kubernetes.name**. To avoid exceptions like **java.lang.ClassCastException** when upgrading from 2.16.0.Final to 2.16.1.Final #30850, this kind of mapping is no longer done.

As you continue your work with Kubernetes and OpenShift in the context of Quarkus, use the new properties and include the **quarkus.** prefix where needed.

1.2.2. Core

1.2.2.1. Upgrade to Jandex 3

With this 3.2 release, Jandex becomes part of the SmallRye project, consolidating all Jandex projects into a single repository: https://github.com/smallrye/jandex/. Consequently, a new release of the Jandex Maven plugin is delivered alongside the Jandex core.

This release also changes the Maven coordinates. Replace the old coordinates with the new ones.

Table 1.2. Old coordinates and their new counterparts

Old coordinates	New coordinates
org.jboss:jandex	**io.smallrye:jandex**
org.jboss.jandex:jandex-maven-plugin	**io.smallrye:jandex-maven-plugin**

If you use the Maven Enforcer plugin, configure it to ban any dependencies on **org.jboss:jandex**. An equivalent plugin is available for Gradle users.

1.2.2.2. Migration path for users of Jandex API

Jandex 3 contains many interesting features and improvements. These changes, unfortunately, required a few breaking changes. Here is the recommended migration path:

1. Upgrade to Jandex 2.4.3.Final. This version provides replacements for some methods that have changed in Jandex 3.0.0. For instance, instead of **ClassInfo.annotations()**, use **annotationsMap()**, and replace **MethodInfo.parameters()** with **parameterTypes()**. Stop using any methods that Jandex has marked as deprecated.

2. Ensure you do not use the return value of **Indexer.index()** or **indexClass()**.

3. If you compile your code against Jandex 2.4.3.Final, it can run against both 2.4.3.Final and 3.0.0. However, there are exceptions to this. If you implement the **IndexView** interface or, in some cases, rely on the **UnresolvedTypeVariable** class, it is not possible to keep the project compatible with both Jandex 2.4.3 and Jandex 3.

4. Upgrade to Jandex 3.0.0. If you implement the **IndexView** interface, ensure you implement the methods that have been added. And if you extensively use the Jandex **Type** hierarchy, verify if you need to handle **TypeVariableReference**, which is now used to represent recursive type variables.

Alongside this release, Jandex introduces a new documentation site. While it's a work in progress, it will become more comprehensive over time. You can also refer to the improved Jandex Javadoc for further information.

1.2.2.3. Removal of io.quarkus.arc.config.ConfigProperties annotation

With this 3.2 release, the previously deprecated **io.quarkus.arc.config.ConfigProperties** annotation has been removed.

Instead, use the **io.smallrye.config.ConfigMapping** annotation to inject multiple related configuration properties.

For more information, see the @ConfigMapping section of the "Mapping configuration to objects" guide.

1.2.2.4. Interceptor binding annotations declared on private methods now generate build failures

With this 3.2 release, declaring an interceptor binding annotation on a private method is not supported and triggers a build failure; for example:

```
jakarta.enterprise.inject.spi.DeploymentException: @Transactional does not affect method
com.acme.MyBean.myMethod() because the method is private. [...]
```

In earlier releases, declaring an interceptor binding annotation on a private method triggered only a warning in logs but was otherwise ignored.

This support change aims to prevent unintentional usage of interceptor annotations on private methods because they do not have any effect and can cause confusion.

To address this change, remove such annotations from private methods. If removing these annotations is not feasible, you can set the configuration property **quarkus.arc.fail-on-intercepted-private-method** to **false**. This setting reverts the system to its previous behavior, where only a warning is logged.

1.2.2.5. Removal of the @AlternativePriority annotation

This release removes the previously deprecated **@AlternativePriority** annotation. Replace it with both the **@Alternative** and **@Priority** annotations.

Example: Removed annotation

```
@AlternativePriority(1)
```

Example: Replacement annotations

```
@Alternative
@Priority(1)
```

Use **jakarta.annotation.Priority** with the **@Priority** annotation instead of **io.quarkus.arc.Priority**, which is deprecated and planned for removal in a future release. Both annotations perform identical functions.

1.2.2.6. Testing changes: Fixation of the Mockito subclass mockmaker

This release updates Mockito version 5.x. Notably, Mockito switched the default mockmaker to **inline** in its 5.0.0 release.

However, to preserve the mocking behavior Quarkus users are familiar with since Quarkus 1.x, and to prevent memory leaks for extensive test suites, Quarkus 3.0 fixes the mockmaker to **subclass** instead of **inline** until the latter is fully supported.

If you want to force the **inline** mockmaker, follow these steps:

1. Add the following exclusion to your **pom.xml**:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5-mockito</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-subclass</artifactId>
    </exclusion>
  </exclusions>
<dependency>
```

2. Add **mockito-core** to your dependencies.

3. Mockito 5.3 removed the **mockito-inline** artifact: you can remove it from your dependencies.

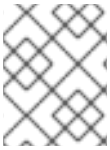
1.2.2.7. Update to the minimum supported Maven version

Quarkus has undergone a refactoring of its Maven plugins to support Maven 3.9. As a result, the minimum Maven version supported by Quarkus has been raised from 3.6.2 to 3.8.6 or later. Ensure your development environment is updated accordingly to benefit from the latest improvements and features.

1.2.2.8. Removal of quarkus-bootstrap-maven-plugin

With this 3.2 release, the previously-deprecated **io.quarkus:quarkus-bootstrap-maven-plugin** Maven plugin has been removed.

This plugin is for Quarkus extension development only. Therefore, if you are developing custom Quarkus extensions, you must change the artifact ID from **io.quarkus:quarkus-bootstrap-maven-plugin** to **io.quarkus:quarkus-extension-maven-plugin**.



NOTE

This change relates specifically to custom extension development. For standard application development, you use the **quarkus-maven-plugin** plugin.

1.2.2.9. Mutiny 2 moves to Java Flow

Mutiny is a reactive programming library, the versions 1.x of which were based on the **org.reactivestream** interfaces, whereas version 2 is based on **java.util.concurrent.Flow**. These APIs are identical, but the package name has changed.

Mutiny offers adapters to bridge between Mutiny 2 (Flow API) and other libraries with legacy reactive streams API.

1.2.3. Data

1.2.3.1. Removal of Hibernate ORM with Panache methods

With this 3.2 release, the following previously deprecated methods from Hibernate ORM with Panache and Hibernate ORM with Panache in Kotlin have been removed:

- **io.quarkus.hibernate.orm.panache.PanacheRepositoryBase#getEntityManager(Class<?> clazz)**
- **io.quarkus.hibernate.orm.panache.kotlin.PanacheRepositoryBase#getEntityManager(clazz: KClass<Any>)**

Instead, use the **Panache.getEntityManager(Class<?> clazz)** method.

1.2.3.2. Enhancement in Hibernate ORM: Automated IN clause parameter padding

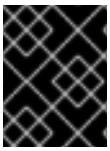
With this 3.2 release, the Hibernate Object-Relational Mapping (ORM) extension has been changed to incorporate automatic **IN** clause parameter padding as a default setting. This improvement augments the caching efficiency for queries that incorporate **IN** clauses.

To revert to the previous functionality and deactivate this feature, you can set the property value of **quarkus.hibernate-orm.query.in-clause-parameter-padding** to **false**.

1.2.3.3. New dependency: Hibernate Reactive 2 and Hibernate ORM 6.2

With this 3.2 release, Quarkus depends on the Hibernate Reactive 2 extension instead of Hibernate Reactive 1. This change implies several changes in behavior and database schema expectations that are incompatible with earlier versions.

Most of the changes are related to Hibernate Reactive 2 depending on Hibernate ORM 6.2 instead of Hibernate ORM 5.6.



IMPORTANT

The Hibernate Reactive 2 extension is available as a Technology Preview in Red Hat build of Quarkus 3.2.

For more information, see the following resources:

- Migration Guide 3.0: Hibernate Reactive
- Hibernate Reactive: 2.0 series
- Migration Guide 3.0: Hibernate ORM 5 to 6 migration

1.2.3.4. Hibernate Search changes

Changes in the defaults for projectable and sortable on GeoPoint fields

With this 3.2 release, Hibernate Search 6.2 changes how defaults are handled for **GeoPoint** fields.

Suppose your Hibernate Search mapping includes **GeoPoint** fields that use the default value for the **projectable** option and either the default value or **Sortable.NO** for the **sortable** option. In that case, Elasticsearch schema validation fails on startup because of missing doc values on those fields.

To prevent that failure, complete either of the following steps:

- Revert to the previous defaults by adding **projectable = Projectable.NO** to the mapping annotation of relevant **GeoPoint** fields.
- Recreate your Elasticsearch indexes and reindex your database. The easiest way to do so is to use the **MassIndexer** with **dropAndCreateSchemaOnStart(true)**.

For more information, see the Data format and schema changes section of the "Hibernate Search 6.2.1.Final: Migration Guide from 6.1".

Deprecated or renamed configuration properties

With this 3.2 release, the **quarkus.hibernate-search-orm.automatic-indexing.synchronization.strategy** property is deprecated and is planned for removal in a future version. Use the **quarkus.hibernate-search-orm.indexing.plan.synchronization.strategy** property instead.

Also, the **quarkus.hibernate-search-orm.automatic-indexing.enable-dirty-check** property is deprecated and is planned for removal in a future version. There is no alternative to replace it. After the removal, it is planned that Search will always trigger reindexing after a transaction modifies an object's field. That is, if a transaction makes the fields "dirty."

For more information, see the Configuration changes section of the "Hibernate Search 6.2.1.Final: Migration Guide from 6.1".

1.2.3.5. Hibernate Validator - Validation.buildDefaultValidatorFactory() now returns a ValidatorFactory managed by Quarkus

With this 3.2 release, Quarkus doesn't support the manual creation of **ValidatorFactory** instances. Instead, you must use the **Validation.buildDefaultValidatorFactory()** method, which returns **ValidatorFactory** instances managed by Quarkus that you inject through Context and Dependency Injection (CDI). The main reason for this change is that a **ValidatorFactory** must be carefully crafted to work in native executables. Before this release, you could still manually create a **ValidatorFactory** instance and handle it yourself if you could make it work. This change aims to improve the compatibility with components creating their own **ValidatorFactory**.

For more information, see the following resources:

- Hibernate Validator extension and CDI section of the "Validation with Hibernate Validator" guide.
- ValidatorFactory and native executables section of the "Validation with Hibernate Validator" guide.
- Obtaining a Validator instance of the "Hibernate Validator 8.0.0.Final - Jakarta Bean Validation Reference Implementation: Reference Guide."

1.2.3.6. Quartz jobs class name change

If you are storing jobs for the Quartz extension in a database by using Java Database Connectivity (JDBC), run the following query to update the job class name in your **JOB_DETAILS** table:

```
UPDATE JOB_DETAILS SET JOB_CLASS_NAME =
'io.quarkus.quartz.runtime.QuartzSchedulerImpl$InvokerJob' WHERE JOB_CLASS_NAME =
'io.quarkus.quartz.runtime.QuartzScheduler$InvokerJob';
```

1.2.3.7. Deprecation of QuarkusTransaction.run and QuarkusTransaction.call methods

The **QuarkusTransaction.run** and **QuarkusTransaction.call** methods have been deprecated in favor of new, more explicit methods.

Update code that relies on these deprecated methods as follows:

Before

```
QuarkusTransaction.run(() -> { ... });
QuarkusTransaction.call(() -> { ... });
```

After

```
QuarkusTransaction.requiringNew().run(() -> { ... });
QuarkusTransaction.requiringNew().call(() -> { ... });
```

Before

```
QuarkusTransaction.run(QuarkusTransaction.runOptions()
        .semantic(RunOptions.Semantic.REQUIRED),
        () -> { ... });
QuarkusTransaction.call(QuarkusTransaction.runOptions()
        .semantic(RunOptions.Semantic.REQUIRED),
        () -> { ... });
```

After

```
QuarkusTransaction.joiningExisting().run(() -> { ... });
QuarkusTransaction.joiningExisting().call(() -> { ... });
```

Before

```
QuarkusTransaction.run(QuarkusTransaction.runOptions()
        .timeout(10)
        .exceptionHandler((throwable) -> {
            if (throwable instanceof SomeException) {
                return RunOptions.ExceptionResult.COMMIT;
            }
            return RunOptions.ExceptionResult.ROLLBACK;
        }),
        () -> { ... });
QuarkusTransaction.call(QuarkusTransaction.runOptions()
        .timeout(10)
        .exceptionHandler((throwable) -> {
```

```
            if (throwable instanceof SomeException) {
              return RunOptions.ExceptionResult.COMMIT;
            }
            return RunOptions.ExceptionResult.ROLLBACK;
        }),
        () -> { ... });
```

After

```
QuarkusTransaction.requiringNew()
        .timeout(10)
        .exceptionHandler((throwable) -> {
            if (throwable instanceof SomeException) {
              return RunOptions.ExceptionResult.COMMIT;
            }
            return RunOptions.ExceptionResult.ROLLBACK;
        })
        .run(() -> { ... });
QuarkusTransaction.requiringNew()
        .timeout(10)
        .exceptionHandler((throwable) -> {
            if (throwable instanceof SomeException) {
              return RunOptions.ExceptionResult.COMMIT;
            }
            return RunOptions.ExceptionResult.ROLLBACK;
        })
        .call(() -> { ... });
```

For more information, see the Programmatic Approach section of the "Using transactions in Quarkus" guide.

1.2.3.8. Renamed Narayana transaction manager property

With this 3.2 release, the **quarkus.transaction-manager.object-store-directory** configuration property is renamed to **quarkus.transaction-manager.object-store.directory**. Update your configuration by replacing the old property name with the new one.

1.2.4. Messaging

1.2.4.1. Removal of vertx-kafka-client dependency from SmallRye Reactive Messaging

This release removes the previously deprecated **vertx-kafka-client** dependency for the **smallrye-reactive-messaging-kafka** extension. Although it wasn't used for client implementations, **vertx-kafka-client** provided default Kafka Serialization and Deserialization (SerDes) for **io.vertx.core.buffer.Buffer**, **io.vertx.core.json.JsonObject**, and **io.vertx.core.json.JsonArray** types from the **io.vertx.kafka.client.serialization** package.

If you require this dependency, you can get SerDes for the mentioned types from the **io.quarkus.kafka.client.serialization** package.

1.2.5. Native

1.2.5.1. Native compilation - Native executables and .so files

With this 3.2 release, changes in GraalVM/Mandrel affect the use of extensions reliant on **.so** files, such as the Java Abstract Window Toolkit (AWT) extension.

When using these extensions, you must add or copy the corresponding **.so** files to the native container; for example:

```
COPY --chown=1001:root target/*.so /work/
COPY --chown=1001:root target/*-runner /work/application
```

NOTE

In this context, the AWT extension provides headless server-side image processing capabilities, not GUI capabilities.

1.2.5.2. Native Compilation - Work around missing CPU features

With this 3.2 release, if you build native executables on recent machines and run them on older machines, you might encounter the following failure when starting the application:

```
The current machine does not support all of the following CPU features that are required by the image: [CX8, CMOV, FXSR, MMX, SSE, SSE2, SSE3, SSSE3, SSE4_1, SSE4_2, POPCNT, LZCNT, AVX, AVX2, BMI1, BMI2, FMA].
Please rebuild the executable with an appropriate setting of the -march option.
```

This error message means that the native compilation used more advanced instruction sets that are unsupported by the CPU running the application. To work around that issue, add the following line to the **application.properties** file:

```
quarkus.native.additional-build-args=-march=compatibility
```

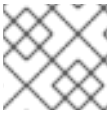
Then, rebuild your native executable. This setting forces the native compilation to use an older instruction set, increasing the chance of compatibility but decreasing optimization.

To explicitly define the target architecture, run **native-image -march=list** to get a list of supported configurations. Then, specify a target architecture; for example:

```
quarkus.native.additional-build-args=-march=x86-64-v4
```

If you are experiencing this problem with older AMD64 hosts, try **-march=x86-64-v2** before using **-march=compatibility**.

The GraalVM documentation for Native Image Build Options states that "[the **-march** parameter generates] instructions for a specific machine type. [This parameter] defaults to **x86-64-v3** on AMD64 and **armv8-a** on AArch64. Use **-march=compatibility** for best compatibility, or **-march=native** for best performance if a native executable is deployed on the same machine or on a machine with the same CPU features. To list all available machine types, use **-march=list**."



NOTE

The **-march** parameter is available only in GraalVM 23 and later.

1.2.5.3. Testing changes: Removal of some annotations

With this 3.2 release, the previously deprecated **@io.quarkus.test.junit.NativeImageTest** and **@io.quarkus.test.junit.DisabledOnNativeImageTest** annotations have been rimage::images/ref_changes-that-affect-backward-compatibility-88d2f.png[]. Replace them with their new counterparts.

Table 1.3. Removed annotations and their new counterparts

Removed annotations	New annotations
@io.quarkus.test.junit.NativeImageTest	**@io.quarkus.test.junit.QuarkusIntegrationTest**
@io.quarkus.test.junit.DisabledOnNativeImageTest	**@io.quarkus.test.junit.DisabledOnIntegrationTest**

The replacement annotations are functionally equivalent to the removed ones.

1.2.6. Observability

1.2.6.1. Deprecated OpenTracing driver is replaced by OpenTelemetry

With this 3.2 release, support for the OpenTracing driver has been deprecated. Removal of the OpenTracing driver is planned for a future Quarkus release.

With this 3.2 release, the SmallRye GraphQL extension has replaced its OpenTracing integration with OpenTelemetry. As a result, when using OpenTracing, the extension no longer generates spans for GraphQL operations.

Also, with this release, the **quarkus.smallrye-graphql.tracing.enabled** configuration property is obsolete and has been removed. Instead, the SmallRye GraphQL extension automatically produces spans when the OpenTelemetry extension is present.

Update your Quarkus applications to use OpenTelemetry so that they remain compatible with future Quarkus releases.

1.2.6.2. Default metrics format in Micrometer now aligned with Prometheus

With this 3.2 release, the Micrometer extension exports metrics in the **application/openmetrics-text** format by default, in line with the Prometheus standard. This change helps make your data easier to read and interpret.

To you get metrics in the earlier format, you can change the **Accept** request header to **text/plain. For example, with the `curl** command:

```
curl -H "Accept: text/plain" localhost:8080/q/metrics/
```

1.2.6.3. Changes in the OpenTelemetry extension and removal of some sampler-related properties

With this 3.2 release, the OpenTelemetry (OTel) extension has significant improvements. Before this release, the OpenTelemetry SDK (OTel SDK) was created at build time and had limited configuration options; most notably, it could not be disabled at run time. Now, it offers enhanced flexibility. It can be

disabled at run time by setting **quarkus.otel.sdk.disabled=true**.

After some preparatory steps at build time, the OTel SDK is configured at run time using the OTel auto-configuration feature. This feature supports some of the properties defined in the Java OpenTelemetry SDK. For more information, see the OpenTelemetry SDK Autoconfigure reference.

The OpenTelemetry extension is compatible with earlier versions. Most properties have been deprecated but still function alongside the new ones until they are removed in a future release. You can replace the deprecated properties with new ones.

Table 1.4. Deprecated properties and their new counterparts

Deprecated properties	New properties
quarkus.opentelemetry.enabled	**quarkus.otel.enabled**
quarkus.opentelemetry.tracer.enabled	**quarkus.otel.traces.enabled**
quarkus.opentelemetry.propagators	**quarkus.otel.propagators**
quarkus.opentelemetry.tracer.suppress-non-application-uris	**quarkus.otel.traces.suppress-non-application-uris**
quarkus.opentelemetry.tracer.include-static-resources	**quarkus.otel.traces.include-static-resources**
quarkus.opentelemetry.tracer.sampler	**quarkus.otel.traces.sampler**
quarkus.opentelemetry.tracer.sampler.ratio	**quarkus.otel.traces.sampler.arg**
quarkus.opentelemetry.tracer.exporter.otlp.enabled	**quarkus.otel.exporter.otlp.enabled**
quarkus.opentelemetry.tracer.exporter.otlp.headers	**quarkus.otel.exporter.otlp.traces.headers**
quarkus.opentelemetry.tracer.exporter.otlp.endpoint	**quarkus.otel.exporter.otlp.traces.legacy-endpoint**

With this 3.2 release, some of the old **quarkus.opentelemetry.tracer.sampler**-related property values have been removed.

If the sampler is parent based, there is no need to set the now-dropped **quarkus.opentelemetry.tracer.sampler.parent-based** property.

Replace the following **quarkus.opentelemetry.tracer.sampler** values with new ones:

Table 1.5. Removed sampler property values and their new counterparts

Old value	New value	New value if parent-based
on	**always_on**	**parentbased_always_on**
off	**always_off**	**parentbased_always_off**
ratio	**traceidratio**	**parentbased_traceidratio**

Many new properties are now available. For more information, see the Quarkus Using OpenTelemetry guide.

Quarkus allowed the Context and Dependency Injection (CDI) configuration of many classes: **IdGenerator**, **Resource** attributes, **Sampler**, and **SpanProcessor**. This is a feature not available in standard OTel, but it's still provided here for convenience. However, the CDI creation of the **SpanProcessor** through the **LateBoundBatchSpanProcessor** is now deprecated. If there's a need to override or customize it, feedback is appreciated. The processor will continue to be used for supporting earlier versions, but soon the standard exports bundled with the OTel SDK will be used. This means the default exporter uses the following configuration:

```
quarkus.otel.traces.exporter=cdi
```

As a preview, the stock OTLP exporter is now available by setting:

```
quarkus.otel.traces.exporter=otlp
```

Additional configurations of the OTel SDK are now available, using the standard Service Provider Interface (SPI) hooks for **Sampler** and **SpanExporter**. The remaining SPIs are also accessible, although compatibility validation through testing is still required. For more information, see the updated OpenTelemetry Guide.

OpenTelemetry upgrades

OpenTelemetry (OTel) 1.23.1 introduced breaking changes, including the following items:

- HTTP span names are now **"{http.method} {http.route}"** instead of just **"{http.route}"**.
- All methods in all **Getter** classes in **instrumentation-api-semconv** have been renamed to use the **get()** naming scheme.
- Semantic convention changes:

Table 1.6. Deprecated properties and their new counterparts

Deprecated properties	New properties
messaging.destination_kind	**messaging.destination.kind**
messaging.destination	**messaging.destination.name**
messaging.consumer_id	**messaging.consumer.id**

Deprecated properties	New properties
messaging.kafka.consumer_group	**messaging.kafka.consumer.group**

JDBC tracing activation

Before this release, to activate Java Database Connectivity (JDBC) tracing, you used the following configuration:

```
quarkus.datasource.jdbc.url=jdbc:otel:postgresql://localhost:5432/mydatabase
# use the 'OpenTelemetryDriver' instead of the one for your database
quarkus.datasource.jdbc.driver=io.opentelemetry.instrumentation.jdbc.OpenTelemetryDriver
```

With this 3.2 release, you can use a much simpler configuration:

```
quarkus.datasource.jdbc.telemetry=true
```

With this configuration, you do not need to change the database URL or declare a different driver.

1.2.7. Security

1.2.7.1. Removal of CORS filter default support for using a wildcard as an origin

The default behavior of the cross-origin resource sharing (CORS) filter has significantly changed. In earlier releases, when the CORS filter was enabled, it supported all origins by default. With this 3.2 release, support for all origins is no longer enabled by default. Now, if you want to permit all origins, you must explicitly configure it to do so.

After a thorough evaluation, if you determine that all origins require support, configure the system in the following manner:

```
quarkus.http.cors=true
quarkus.http.cors.origins=/.*/
```

Same-origin requests receive support without needing the **quarkus.http.cors.origins** configuration. Therefore, adjusting the **quarkus.http.cors.origins** becomes essential only when you allow trusted third-party origin requests. In such situations, enabling all origins might pose unnecessary risks.

WARNING

Use this setting with caution to maintain optimal system security.

1.2.7.2. OpenAPI CORS support change

With this 3.2 release, OpenAPI has changed its cross-origin resource sharing (CORS) settings and no longer enables wildcard (*) origin support by default. This change helps to prevent potential leakage of OpenAPI documents, enhancing the overall security of your applications.

Although you can enable wildcard origin support in dev mode , it is crucial to consider the potential security implications. Avoid enabling all origins in a production environment because it exposes your applications to security threats. Ensure your CORS settings align with your production environment's recommended security best practices.

1.2.7.3. Encryption of OIDC session cookie by default

With this 3.2 release, the OpenID Connect (OIDC) session cookie, created after the completion of an OIDC Authorization Code Flow, is encrypted by default. In most scenarios, you are unlikely to notice this change.

However, if the **mTLS** or **private_key_jwt** authentication methods - where the OIDC client private key signs a JSON Web Token (JWT) - are used between Quarkus and the OIDC Provider, an in-memory encryption key gets generated. This key generation can result in some pods failing to decrypt the session cookie, especially in applications dealing with many requests. This situation can arise when a pod attempting to decrypt the cookie isn't the one that encrypted it.

If such issues occur, register an encryption secret of 32 characters; for example:

```
quarkus.oidc.token-state-manager.encryption-secret=eUk1p7UB3nFiXZGUXi0uph1Y9p34YhBU
```

An encrypted session cookie can exceed 4096-bytes, which can cause some browsers to ignore it. If this occurs, try one or more of the following steps:

- Set **quarkus.oidc.token-state-manager.split-tokens=true** to store ID, access, and refresh tokens in separate cookies.
- Set **quarkus.oidc.token-state-manager.strategy=id-refresh-tokens** if there's no need to use the access token as a source of roles to request **UserInfo** or propagate it to downstream services.
- Register a custom **quarkus.oidc.TokenStateManager** Context and Dependency Injection (CDI) bean with the alternative priority set to **1**.

If application users access the Quarkus application from within a trusted network, disable the session cookie encryption by applying the following configuration:

```
quarkus.oidc.token-state-manager.encryption-required=false
```

1.2.7.4. Default SameSite attribute set to Lax for OIDC session cookie

With this 3.2 release, for the Quarkus OpenID Connect (OIDC) extension, the session cookie **SameSite** attribute is set to **Lax** by default.

In some earlier releases of Quarkus, the OIDC session cookie **SameSite** attribute was set to **Strict** by default. This setting introduced unpredictability in how different browsers handled the session cookie.

1.2.7.5. The OIDC ID token audience claim is verified by default

With this 3.2 release, the OpenID Connect (OIDC) ID token **aud** (audience) claim is verified by default. This claim must equal the value of the configured **quarkus.oidc.client-id** property, as required by the OIDC specification.

To override the expected ID token audience value, set the **quarkus.oidc.token.audience** configuration property. If you deal with a noncompliant OIDC provider that does not set an ID token **aud** claim, you can set **quarkus.oidc.token.audience** to **any**.

WARNING

Setting **quarkus.oidc.token.audience** to **any** reduces the security of your 3.2 application.

1.2.7.6. Removal of default password for the JWT key and keystore

Before this release, Quarkus used **password** as the default password for the JSON Web Token (JWT) key and keystore. With this 3.2 release, this default value has been removed.

If you are still using the default password, set a new value to replace **password** for the following properties in the **application.properties** file:

```
quarkus.oidc-client.credentials.jwt.key-store-password=password
quarkus.oidc-client.credentials.jwt.key-password=password
```

1.2.8. Web

1.2.8.1. Changes to RESTEasy Reactive multipart

With this 3.2 release, the following changes impact multipart support in RESTEasy Reactive:

- Before this release, you could catch all file uploads regardless of the parameter name using the syntax: **@RestForm List<FileUpload> all**, but this was ambiguous and not intuitive. Now, this form only fetches parameters named **all**, just like for every other form element of other types, and you must use the following form to catch every parameter regardless of its name: **@RestForm(FileUpload.ALL) List<FileUpload> all**.
- Multipart form parameter support has been added to **@BeanParam**. The **@MultipartForm** annotation is now deprecated. Use **@BeanParam** instead of **@MultipartForm**.
- The **@BeanParam** is now optional and implicit for any non-annotated method parameter with fields annotated with any **@Rest*** or **@*Param** annotations.
- Multipart elements are no longer limited to being encapsulated inside **@MultipartForm**-annotated classes: they can be used as method endpoint parameters and endpoint class fields.
- Multipart elements now default to the **@PartType(MediaType.TEXT_PLAIN)** MIME type unless they are of type **FileUpload**, **Path**, **File**, **byte[]**, or **InputStream**.
- Multipart elements of the **MediaType.TEXT_PLAIN** MIME type are now deserialized using the regular **ParamConverter** infrastructure. Before this release, deserialization used **MessageBodyReader**.

- Multipart elements of the **FileUpload**, **Path**, **File**, **byte[]**, or **InputStream** types are special-cased and deserialized by the RESTEasy Reactive extension, not by the **MessageBodyReader** or **ParamConverter** classes.
- Multipart elements of other explicitly set MIME types still use the appropriate **MessageBodyReader** infrastructure.
- Multipart elements can now be wrapped in **List** to obtain all values of the part with the same name.
- Any client call that includes the **@RestForm** or **@FormParam** parameters defaults to the **MediaType.APPLICATION_FORM_URLENCODED** content type unless they are of the **File**, **Path**, **Buffer**, **Multi<Byte>**, or **byte[]** types, in which case it defaults to the **MediaType.MULTIPART_FORM_DATA** content type.
- Class **org.jboss.resteasy.reactive.server.core.multipart.MultipartFormDataOutput** has been moved to **org.jboss.resteasy.reactive.server.multipart.MultipartFormDataOutput**.
- Class **org.jboss.resteasy.reactive.server.core.multipart.PartItem** has been moved to **org.jboss.resteasy.reactive.server.multipart.PartItem**.
- Class **org.jboss.resteasy.reactive.server.core.multipart.FormData.FormValue** has been moved to **org.jboss.resteasy.reactive.server.multipart.FormValue**.
- The REST Client no longer uses the server-specific **MessageBodyReader** and **MessageBodyWriter** classes associated with Jackson. Before this release, the REST Client unintentionally used those classes. The result is that applications that use both **quarkus-resteasy-reactive-jackson** and **quarkus-rest-client-reactive** extensions must now include the **quarkus-rest-client-reactive-jackson** extension.

1.2.8.2. Enhanced JAXB extension control

The JAXB extension detects classes that use JAXB annotations and registers them into the default **JAXBContext** instance. Before this release, any issues or conflicts between the classes and JAXB triggered a JAXB exception at runtime, providing a detailed description to help troubleshoot the problem. However, you could preemptively tackle these conflicts during the build stage.

This release adds a feature that can validate the **JAXBContext** instance at build time so that you can detect and fix JAXB errors early in the development cycle.

For example, as shown in the following code block, binding both classes to the default **JAXBContext** instance would inevitably lead to a JAXB exception. This is because the classes share the identical name, **Model**, despite existing in different packages. This concurrent naming creates a conflict, leading to the exception.

```
package org.acme.one;

import jakarta.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Model {
   private String name1;

   public String getName1() {
      return name1;
   }
```

```
    public void setName1(String name1) {
        this.name1 = name1;
    }
}

package org.acme.two;

import jakarta.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Model {
    private String name2;

    public String getName2() {
        return name2;
    }

    public void setName2(String name2) {
        this.name2 = name2;
    }
}
```

To activate this feature, add the following property:

```
quarkus.jaxb.validate-jaxb-context=true
```

Additionally, this release adds the **quarkus.jaxb.exclude-classes** property. With this property, you can specify classes to exclude from binding to the **JAXBContext**. You can provide a comma-separated list of fully qualified class names or a list of packages.

For example, to resolve the conflict in the preceding example, you can exclude one or both of the classes:

```
quarkus.jaxb.exclude-classes=org.acme.one.Model,org.acme.two.Model
```

Or you can exclude all the classes under a package:

```
quarkus.jaxb.exclude-classes=org.acme.*
```

1.3. ADDITIONAL RESOURCES

- Release notes for Red Hat build of Quarkus version 3.2