Red Hat build of Quarkus 3.8

Migrating applications to Red Hat build of Quarkus 3.8

Last Updated: 2024-05-09

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

PROVIDING FEEDBACK ON RED HAT BUILD OF QUARKUS DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

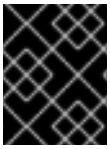
1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

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 3.8

As an application developer, you can migrate applications that are based on earlier versions of Red Hat build of Quarkus to version 3.8 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:3.8.4.redhat-00002: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 3.8 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 3.8.

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. Core

1.2.1.1. Changes in Stork load-balancer configuration

You can no longer use the previous configuration names **stork."service-name".load-balancer** and **quarkus.stork."service-name".load-balancer** for configuring the Stork load balancer. Instead, use **quarkus.stork."service-name".load-balancer.type** for configuration settings.

1.2.1.2. Dependency management update for OkHttp and Okio

OkHttp and Okio have been removed from the Quarkus Platform BOM, and their versions are no longer enforced, addressing issues related to outdated dependencies. This change affects test framework dependencies and streamlines runtime dependencies. Developers using these dependencies must now specify their versions in build files. Additionally, the **quarkus-test-infinispan-client** artifact has been removed due to the availability of robust Dev Services support for Infinispan.

1.2.1.3. Java version requirement update

Beginning with this version of Red Hat build of Quarkus, support for Java 11, deprecated in the previous version, has been removed. Java 21 is now the recommended version, although Java 17 is also supported.

1.2.1.4. JAXB limitations with collections in RESTEasy Reactive

In Red Hat build of Quarkus, using RESTEasy Reactive with Java Architecture for XML Binding (JAXB) does not support using collections, arrays, and maps as parameters or return types in REST methods. To overcome this limitation of JAXB, encapsulate these types within a class annotated with **@XmlRootElement**.

1.2.1.5. Mandatory specification of @StaticInitSafe at build time

During the static initialization phase, Red Hat build of Quarkus collects the configuration to inject in CDI beans. The collected values are then compared with their runtime initialization counterparts, and if a mismatch is detected, the application startup fails. With Red Hat build of Quarkus 3.8, you can now annotate configuration objects with the **@io.quarkus.runtime.annotations.StaticInitSafe** annotation to inform users that the injected configuration:

- is set at build time
- cannot be changed
- is safe to be used at runtime, instructing Red Hat build of Quarkus to not fail the startup on configuration mismatch

1.2.1.6. Qute: Isolated execution of tag templates by default

User tags in templates are now executed in isolation by default, restricting access to the calling template's context. This update can alter data handling within tag templates, potentially impacting their current functionality. To bypass this isolation and maintain access to the parent context, include **_isolated=false** or **_unisolated** in the tag call, for example, **# itemDetail item showImage=true _isolated=false**. This approach allows tags to access data from the parent context as before. This change minimizes unintended data exposure from the parent context to the tag, enhancing template data integrity. However, it might necessitate updates to existing templates reliant on shared context access, representing a notable change that could affect users unfamiliar with this isolation mechanism.

1.2.1.7. Qute: Resolving type pollution issues

ResultNode class is updated to be an abstract class, not an interface, and should not be user-implemented despite being in the public API. The **Qute** API now limits **CompletionStage** implementations to **java.util.concurrent.CompletableFuture** and **io.quarkus.qute.CompletedStage** by default, a restriction alterable with **-Dquarkus.qute.unrestricted-completion-stage-support=true**.

1.2.1.8. quarkus-rest-client extensions renamed to quarkus-resteasy-client

With Red Hat build of Quarkus 3.8, the following **quarkus-rest-client** extensions are renamed:

Old name	New name
quarkus-rest-client	**quarkus-resteasy-client**
quarkus-rest-client-mutiny	**quarkus-resteasy-client-mutiny**
quarkus-rest-client-jackson	**quarkus-resteasy-client-jackson**
quarkus-rest-client-jaxb	**quarkus-resteasy-client-jaxb**
quarkus-rest-client-jsonb	**quarkus-resteasy-client-jsonb**

1.2.1.9. Removing URI validation when @TestHTTPResource is injected

The **@TestHTTPResource** annotation now supports path parameters. Validation as a URI string is no longer applied due to non-compliance with the URI format.

1.2.1.10. Updates to GraalVM SDK 23.1.2 with dependency adjustments

The GraalVM SDK version has been updated to 23.1.2 in Red Hat build of Quarkus 3.8. Developers using extensions requiring GraalVM substitutions should switch from **org.graalvm.sdk:graal-sdk** to **org.graalvm.sdk:nativeimage** to access necessary classes. For those that use **org.graalvm.js:js**, replace this dependency with **org.graalvm.polyglot:js-community** for the community version. For the enterprise version, replace this dependency with **org.graalvm.polyglot:js**. The adjustment for the **graal-sdk** is automated with **quarkus update**. However, changes to the **js** dependency must be made manually. Even though it is highly unlikely, this change could affect users who depend on:

- **org.graalvm.sdk:collections**
- **org.graalvm.sdk:word**

1.2.1.11. Various adjustments to QuarkusComponentTest

In this release, **QuarkusComponentTest** has undergone several adjustments. It remains experimental and is not supported by Red Hat build of Quarkus. This experimental status indicates that the API might change at any time, reflecting feedback received.

The **QuarkusComponentTestExtension** is now immutable, requiring programmatic registration through the simplified constructor **QuarkusComponentTestExtension(Class...)** or the **QuarkusComponentTestExtension.builder()** method. The test instance lifecycle, either **Lifecycle#PER_METHOD** (default) or **Lifecycle#PER_CLASS**, dictates when the CDI container starts and stops; **PER_METHOD** starts the container before each test and stops it afterward, whereas **PER_CLASS** starts it before all tests and stops it after all tests. This represents a change from previous versions, where the container always started before and stopped after all tests.

1.2.2. Data

1.2.2.1. Hibernate ORM upgraded to 6.4

In Red Hat build of Quarkus 3.8, Hibernate Object-Relational Mapping (ORM) was upgraded to version 6.4 and introduced the following breaking changes:

- Compatibility with some older database versions is dropped. For more information about supported versions, see Supported dialects.
- Numeric literals are now interpreted as defined in Jakarta Persistence 3.2.

For more information, see the Hibernate ORM 6.4 migration guide.

1.2.2.2. Hibernate ORM database version verification at startup

When using Hibernate ORM on Red Hat build of Quarkus 3.8, you can verify the specified database version on application startup.

For Hibernate ORM to generate more efficient SQL and take advantage of more database features, you can set a specific database version for the Hibernate database in the **applications.properties** file.

For example: **quarkus.datasource.db-version = 14.0**

With this 3.8 release, on application startup, Red Hat build of Quarkus verifies the specified database version against the actual database version your application is connecting to, and throws an exception on startup in case of a mismatch.

For more information, see the Supported databases section of the Quarkus "Using Hibernate ORM and Jakarta persistence" guide.

1.2.2.3. Hibernate Search upgraded to 7.0

In Red Hat build of Quarkus 3.8, Hibernate Search was upgraded to version 7.0 and introduced the following breaking changes:

- The values accepted by the **quarkus.hibernate-search-orm.coordination.entity-mapping.outbox-event.uuid-type** and **quarkus.hibernate-search-orm.coordination.entity-mapping.agent.uuid-type** configuration properties changed:
  - **uuid-binary** is deprecated in favor of **binary**

- **uuid-char** is deprecated in favor of **char**

- The default value for the **quarkus.hibernate-search-orm.elasticsearch.query.shard-failure.ignore** property changed from **true** to **false**, meaning that Hibernate Search now throws an exception if at least one shard fails during a search operation. To get the previous behavior, set this configuration property to **true**.

NOTE

If you define multiple backends, you must set this configuration property for each Elasticsearch backend.

- The complement operator (~) in the regular expression predicate was removed with no alternative to replace it.

- Hibernate Search dependencies no longer have an **-orm6** suffix in their artifact ID; for example, applications now depend on the **hibernate-search-mapper-orm** module instead of **hibernate-search-mapper-orm-orm6**.

For more information, see the following resources:

- Hibernate Search documentation

- Hibernate Search 7.0.0.Final: Migration guide from 6.2

1.2.2.4. SQL Server Dev Services upgraded to 2022-latest

Dev Services for SQL Server updated its default image from **mcr.microsoft.com/mssql/server:2019-latest** to **mcr.microsoft.com/mssql/server:2022-latest**.

Users preferring the previous version can specify an alternative by using the config property detailed in the References section in the Red Hat build of Quarkus "Configure data sources" guide.

1.2.2.5. Upgrade to Flyway adds additional dependency for Oracle users

In Red Hat build of Quarkus 3.8, the Flyway extension is upgraded to Flyway 9.20.0, which delivers an additional dependency, **flyway-database-oracle**, for Oracle users.

Oracle users must update the **pom.xml** file to include the **flyway-database-oracle** dependency. To do so, do the following:

```
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-database-oracle</artifactId>
</dependency>
```

For more information, see the Quarkus Using Flyway guide.

1.2.3. Native

1.2.3.1. Strimzi OAuth support issue in the Kafka extension

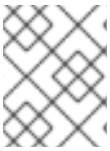
The Kafka extension's Strimzi OAuth support in **quarkus-bom** now uses **io.strimzi:strimzi-kafka-oauth** version 0.14.0, introducing a known issue that leads to native build failures. The error, **Substitution**

target for `io.smallrye.reactive.kafka.graal.Target_com_jayway_jsonpath_internal_DefaultsImpl is not loaded can be bypassed by adding **io.strimzi:kafka-oauth-common** to your project's classpath.

1.2.4. Observability

1.2.4.1. @AddingSpanAttributes annotation added

When using Opentelemetry (oTel) instrumentation with Red Hat build of Quarkus 3.8, you can now annotate a method in any Context Dependency Injection (CDI)-aware bean by using the **io.opentelemetry.instrumentation.annotations.AddingSpanAttributes** annotation, which does not create a new span but adds annotated method parameters to attributes in the current span.



NOTE

If you mistakenly annotate a method with both **@AddingSpanAttributes** and **@WithSpan** annotations, the **@WithSpan** annotation takes precedence.

For more information, see the CDI section of the Quarkus "Using OpenTelemetry" guide.

1.2.4.2. quarkus-smallrye-metrics extension no longer supported

With Red Hat build of Quarkus 3.8, the **quarkus-smallrye-metrics** extension is no longer supported. Now, it is available as a community extension only. Its use in production environments is discouraged.

From Red Hat build of Quarkus 3.8, **quarkus-smallrye-metrics** is replaced by the fully supported **quarkus-micrometer** extension.

1.2.4.3. quarkus-smallrye-opentracing extension no longer supported

With Red Hat build of Quarkus 3.8, SmallRye OpenTracing is no longer supported. To continue using distributed tracing, migrate your applications to SmallRye OpenTelemetry, which is now fully supported with this release and no longer a Technology Preview feature. If you still need to use **quarkus-smallrye-opentracing**, adjust your application to use the extensions from Quarkiverse by updating the **groupId** and specifying the version manually.

1.2.4.4. Refactoring of Scheduler and OpenTelemetry Tracing extensions

In Red Hat build of Quarkus 3.8, integration of OpenTelemetry Tracing and the **quarkus-scheduler** extension has been refactored.

Before this update, only **@Scheduled** methods had a new **io.opentelemetry.api.trace.Span** class, which is associated automatically when you enable tracing. That is, when the **quarkus.scheduler.tracing.enabled** configuration property is set to **true**, and the **quarkus-opentelemetry** extension is available.

With this 3.8 release, all scheduled jobs, including those that are scheduled programmatically, have a Span associated automatically when tracing is enabled. The unique job identifier for each scheduled method is either generated, is specified by setting the **io.quarkus.scheduler.Scheduled#identity** attribute or with the **JobDefinition** method. Before this update, span names followed the **<simpleclassname>.<methodName>** format.

For more information, see the following Quarkus resources:

- Scheduler reference

- Using OpenTelemetry

1.2.5. Security

1.2.5.1. Enhanced Security with mTLS and HTTP Restrictions

When mTLS client authentication (**quarkus.http.ssl.client-auth**) is set to **required**, Red Hat build of Quarkus automatically disables plain HTTP ports to ensure that only secure HTTPS requests are accepted. To enable plain HTTP, configure **quarkus.http.ssl.client-auth** to **request** or set both **quarkus.http.ssl.client-auth=required** and **quarkus.http.insecure-requests=enabled**.

1.2.5.2. JWT extension removes unnecessary Reactive Routes dependency

The JWT extension no longer transitively depends on the Reactive Routes extension. If your application uses both JWT and Reactive Routes features but does not declare an explicit dependency on Reactive Routes, you must add this dependency.

1.2.5.3. Keycloak Authorization dropped the keycloak-adapter-core dependency

The **quarkus-keycloak-authorization** extension no longer includes the **org.keycloak:keycloak-adapter-core** dependency due to its update to Keycloak 22.0.0 and its irrelevance to the extension's functionality. In future Keycloak versions, it is planned to remove the Keycloak Java adapters code. If your application requires this dependency, manually add it to your project's **pom.xml**.

1.2.5.4. Using CDI interceptors to resolve OIDC tenants in RESTEasy Classic no longer supported

You can no longer use Context and Dependency Injection (CDI) annotations and interceptors to resolve tenant OIDC configuration for RESTEasy Classic applications.

Due to security checks that are enforced before CDI interceptors and checks requiring authentication are triggered, using CDI interceptors to resolve multiple OIDC provider configuration identifiers no longer works.

Use **@Tenant** annotation or custom **io.quarkus.oidc.TenantResolver** instead.

For more information, see the Resolve with annotations section of the Quarkus "Using OIDC multitenancy guide".

1.2.5.5. Using OIDC @Tenant annotation to bind OIDC features to tenants no longer possible

In Red Hat build of Quarkus 3.8, you must now use the **quarkus.oidc.TenantFeature** annotation instead of **quarkus.oidc.Tenant** to bind OpenID Connect (OIDC) features to OIDC tenants.

The **quarkus.oidc.Tenant** annotation is now used for resolving tenant configuration.

1.2.5.6. Security profile flexibility enhancement

Red Hat build of Quarkus 3.8 allows runtime configuration of HTTP permissions and roles, enabling flexible security settings across profiles. This resolves the issue of native executables locking to build-time security configurations. Security can now be dynamically adjusted per profile, applicable in both JVM and native modes.

1.2.6. Standards

1.2.6.1. Correction in GraphQL directive application

The application of annotation-based GraphQL directives has been corrected to ensure they are only applied to the schema element types for which they are declared.

For example, if a directive was declared to apply to the GraphQL element type FIELD but was erroneously applied to a different element type, it was still visible in the schema on the element where it should not be applicable, leading to an invalid schema. This was now corrected, and directives have their usage checked against their applicability declaration.

If you had directives applied incorrectly in this way, they will no longer appear in the schema, and Red Hat build of Quarkus 3.8 will log a warning during the build.

1.2.7. OpenAPI standardizes content type defaults for POJOs and primitives

This change has standardized the default content type for generating OpenAPI documentation when a **@ContentType** annotation is not provided. Previously, the default content type varied across different extensions, such as RestEasy Reactive, RestEasy Classic, Spring Web, and OpenAPI. For instance, OpenAPI always used JSON as the default, whereas RestEasy used JSON for object types and text for primitive types. Now, all extensions have adopted uniform default settings, ensuring consistency:

- **Primitive types** are now uniformly set to **text/plain**.
- **Complex POJO (Plain Old Java Object) types**default to **application/json**.

This unification ensures that while the behavior across extensions is consistent, it differentiates appropriately based on the type of data, with primitives using **text/plain** and POJOs using **application/json**. This approach does not imply that the same content type is used for all Java types but rather that all extensions now handle content types in the same manner, tailored to the nature of the data.

1.2.8. Web

1.2.8.1. Improved SSE handling in REST Client

Red Hat build of Quarkus 3.8 has enhanced its REST Client's Server-Sent Events (SSE) capabilities, enabling complete event returns and filtering. These updates and new descriptions in REST Client provide developers with increased control and flexibility in managing real-time data streams.

1.2.8.2. Manual addition of the Reactive Routes dependency

Until version 3.8, the Red Hat build of Quarkus SmallRye JWT automatically incorporated **quarkus-reactive-routes**, a feature discontinued from version 3.8 onwards. To ensure continued functionality, manually add **quarkus-reactive-routes** as a dependency in your build configuration.

1.3. ADDITIONAL RESOURCES

- Release notes for Red Hat build of Quarkus version 3.2