Red Hat build of Quarkus 3.8

Release Notes for Red Hat build of Quarkus 3.8

Last Updated: 2024-07-23

Red Hat build of Quarkus 3.8 Release Notes for Red Hat build of Quarkus 3.8

Legal Notice

Abstract

Release notes provide information about new features, notable technical changes, features in technology preview, bug fixes, known issues, and related advisories.

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. RELEASE NOTES FOR RED HAT BUILD OF QUARKUS 3.8

Release notes provide information about new features, notable technical changes, features in technology preview, bug fixes, known issues, and related advisories for Red Hat build of Quarkus 3.8.

Information about upgrading and backward compatibility is also provided to help you make the transition from an earlier release.

1.1. ABOUT RED HAT BUILD OF QUARKUS

Red Hat build of Quarkus is a Kubernetes-native Java stack optimized for containers and Red Hat OpenShift Container Platform. Quarkus is designed to work with popular Java standards, frameworks, and libraries such as Eclipse MicroProfile, Eclipse Vert.x, Apache Camel, Apache Kafka, Hibernate ORM with Jakarta Persistence, and RESTEasy Reactive (Jakarta REST).

As a developer, you can choose the Java frameworks you want for your Java applications, which you can run in Java Virtual Machine (JVM) mode or compile and run in native mode. Quarkus provides a container-first approach to building Java applications. The container-first approach facilitates the containerization and efficient execution of microservices and functions. For this reason, Quarkus applications have a smaller memory footprint and faster startup times.

Quarkus also optimizes the application development process with capabilities such as unified configuration, automatic provisioning of unconfigured services, live coding, and continuous testing that gives you instant feedback on your code changes.

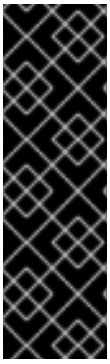
1.2. DIFFERENCES BETWEEN THE QUARKUS COMMUNITY VERSION AND RED HAT BUILD OF QUARKUS

As an application developer, you can access two different versions of Quarkus: the Quarkus community version and the productized version, Red Hat build of Quarkus.

The following table describes the differences between the Quarkus community version and Red Hat build of Quarkus.

Feature	Quarkus community version	Red Hat build of Quarkus version	Description
Access to the latest community features	Yes	No	With the Quarkus community version, you can access the latest feature developments. Red Hat does not release Red Hat build of Quarkus to correspond with every version that the community releases. The cadence of Red Hat build of Quarkus feature releases is approximately every six months.

Feature	Quarkus community version	Red Hat build of Quarkus version	Description
Enterprise support from Red Hat	No	Yes	Red Hat provides enterprise support for Red Hat build of Quarkus only. To report issues about the Quarkus community version, see quarkusio/quarkus - Issues.
Access to long-term support	No	Yes	The lifecycle for a major release of Red Hat build of Quarkus is divided into two support phases; full support and maintenance support. For information about the product lifecycle, timelines, and support policies of Red Hat build of Quarkus, log in to the Red Hat Customer Portal and see the Product lifecycles and Red Hat build of Quarkus lifecycle and support policies Knowledge Base articles.
Common Vulnerabilities and Exposures (CVE) fixes and bug fixes backported to earlier releases	No	Yes	With Red Hat build of Quarkus, selected CVE fixes and bug fixes are regularly backported to supported streams. For more information about maintenance support, see Red Hat build of Quarkus lifecycle and support policies.
Tested and verified with Red Hat OpenShift Container Platform and Red Hat Enterprise Linux (RHEL)	No	Yes	Red Hat build of Quarkus is built, tested, and verified with Red Hat OpenShift Container Platform and RHEL. Red Hat provides both production and development support for supported configurations and tested integrations according to your subscription agreement. For more information, see Red Hat build of Quarkus supported configurations.
Built from source using secure build systems	No	Yes	In Red Hat build of Quarkus, the core platform and all supported extensions are provided by Red Hat using secure software delivery, which means that they are built from source, scanned for security issues, and with verified license usage.
Access to support for JDK and Red Hat build of Quarkus Native builder distribution	No	Yes	Red Hat build of Quarkus supports certified OpenJDK builds and certified native executable builders. See admonition below. For more information, see Red Hat build of Quarkus supported configurations.



IMPORTANT

Red Hat build of Quarkus supports the building of native Linux executables by using a Red Hat build of Quarkus Native builder image, which is based on Mandrel and distributed by Red Hat.

For more information, see Compiling your Quarkus applications to native executables . Building native executables by using Oracle GraalVM Community Edition (CE), Mandrel community edition, or any other distributions of GraalVM is not supported for Red Hat build of Quarkus.

1.3. NEW FEATURES, ENHANCEMENTS, AND TECHNICAL CHANGES

This section overviews the new features, enhancements, and technical changes introduced in Red Hat build of Quarkus 3.8.

1.3.1. Core

1.3.1.1. Support for Java 21

Java 21 is now the recommended version, although Java 17 is also supported.

1.3.1.2. Support for Java 11 has been removed

With this 3.8 release, support for Java 11, which was deprecated in version 3.2, has been removed.

1.3.1.3. Red Hat build of Quarkus added support for virtual threads

The use of Red Hat build of Quarkus with Virtual Threads (VTs) brings the following:

- Enhances the management of concurrent tasks, improving scalability and resource efficiency.
- Enhances the imperative programming model by improving resource efficiency with virtual threads, which are inexpensive to block.
- Simplifies the concurrency model, streamlining code-base maintenance.
- Reduces thread context switching overhead, resulting in lower latency and higher throughput.
- Enables better multi-core processor utilization, allowing more concurrent tasks without heavy context-switching penalties.

NOTE

Virtual threads are supported only on Java 21 JVMs. For more information, see Virtual Threads section of the Oracle *Java Core Libraries Developer Guide* and the OpenJDK's JEP 444: Virtual Threads .

Virtual thread Limitations:

- Libraries pinning the carrier thread might delay adoption until the Java ecosystem fully embraces virtual-thread compatibility.
- Lengthy computations require careful analysis to prevent monopolization of resources.

- Elasticity of the carrier thread pool could lead to increased memory consumption.
- The thread-local object polling pattern might significantly impact allocations and memory usage.
- Virtual threads do not inherently solve thread safety issues, requiring diligent management.

1.3.2. Data

1.3.2.1. Hibernate ORM upgraded to version 6.4

In Red Hat build of Quarkus 3.8, Hibernate Object-Relational Mapping (ORM) is upgraded to version 6.4.

For more information, see the following resources:

- Changes that affect compatibility with earlier versions
- Hibernate ORM documentation 6.4
- Quarkus Using Hibernate ORM and Jakarta Persistence guide

1.3.2.2. Hibernate Reactive operating alongside Agroal

In Red Hat build of Quarkus 3.8, Hibernate Reactive can co-exist alongside Agroal, which means you can use Flyway or Liquibase in your applications while using Hibernate Reactive as the Object-Relational Mapping (ORM).

NOTE

A limitation exists in Red Hat build of Quarkus whereby you cannot have both Hibernate ORM and Hibernate Reactive in the same application.

For more information, see the Quarkus Using Hibernate Reactive guide.

1.3.2.3. Hibernate Reactive upgraded to version 2.2

In Red Hat build of Quarkus 3.8, the Hibernate Reactive extension is upgraded to version 2.2, which is compatible with Hibernate ORM 6.4.0.

IMPORTANT

The Hibernate Reactive extension is available as a Technology Preview in Red Hat build of Quarkus 3.8.

For more information, see the Hibernate Reactive 2.2.0 documentation.

1.3.2.4. Hibernate Search upgraded to version 7.0

In Red Hat build of Quarkus 3.8, Hibernate Search is upgraded to version 7.0.

Hibernate Search offers indexing and full-text search capabilities to your Red Hat build of Quarkus applications. Version 7.0 introduces enhancements, new features, and some notable changes to the default configuration for geo-point fields.

For more details, see Changes that affect compatibility with earlier versions.

To learn more about Hibernate Search 7.0, see the following resources:

- Quarkus Hibernate Search guide
- Hibernate Search 7.0.1 reference documentation

1.3.2.5. New OpenSearch Dev Service

Red Hat build of Quarkus 3.8 introduces a new OpenSearch Dev Service.

When you use Hibernate Search, Dev Services defaults to starting Elasticsearch or OpenSearch based on your Hibernate Search configuration.

To configure Dev Services to use OpenSearch, specify the following setting:

quarkus.elasticsearch.devservices.distribution=opensearch

For more information, see the Configuring the image section of the Quarkus "Dev Services for Elasticsearch" guide.

1.3.3. Observability

1.3.3.1. Customization of Micrometer by using MeterRegistry

Red Hat build of Quarkus 3.8 introduces many ways to customize Micrometer by using the new **MeterRegistryCustomizer** interface implemented as Contexts and Dependency Injection (CDI) beans.

You can customize Micrometer in the following ways:

- By using **MeterFilter** instances to customize metrics emitted by **MeterRegistry** instances. The **Micrometer** extension detects **MeterFilter** CDI beans and uses them when initializing **MeterRegistry** instances.
- By using **HttpServerMetricsTagsContributor** for server HTTP requests. User code can contribute arbitrary tags based on the details of an HTTP request by providing CDI beans that implement **io.quarkus.micrometer.runtime.HttpServerMetricsTagsContributor**.
- By using **MeterRegistryCustomizer** for arbitrary customizations to meter registries. User code can change the configuration of any **MeterRegistry** that is activated by providing CDI beans that implement **io.quarkus.micrometer.runtime.MeterRegistryCustomizer**.

For more information, see the Customizing Micrometer section of the Quarkus “Micrometer Metrics” guide.

1.3.3.2. Micrometer @MeterTag supported

Micrometer defines two annotations, **@Counted** and **@Timed**, which you can add to methods.

With Red Hat build of Quarkus 3.8, Micrometer can add the **@MeterTag** annotation to parameters of methods annotated with **@Counted** and **@Timed**.

The **@MeterTag** annotation uses the **ValueResolver** or **ValueExpressionResolver** resolvers from the **io.micrometer.common.annotation** package to dynamically assign additional tag values to the method counter or timer.

For more information, see the Quarkus Micrometer Metrics guide.

1.3.3.3. Netty metrics supported in Micrometer

Red Hat build of Quarkus 3.8 introduces support for the gathering of Netty allocator metrics from the Micrometer metrics library.

The **quarkus.micrometer.binder.netty.enabled** property is introduced, which enables the gathering of Netty metrics if Micrometer support is also enabled.

Netty allocator metrics provide insights on memory allocation and usage within your Netty framework, which can help you to gain an understanding of the performance of your Red Hat build of Quarkus applications that use Netty.

The following metrics are gathered:

Metric	Description
netty.allocator.memory.used	Size, in bytes, of the memory that the allocator uses
netty.allocator.memory.pinned	Size, in bytes, of the memory that the allocated buffer uses.
netty.allocator.pooled.arenas	Number of arenas for a pooled allocator
netty.allocator.pooled.cache.size	Size, in bytes, of the cache for a pooled allocator
netty.allocator.pooled.threadlocal.caches	Number of **ThreadLocal** caches for a pooled allocator
netty.allocator.pooled.chunk.size	Size, in bytes, of memory chunks for a pooled allocator
netty.eventexecutor.tasks.pending	Number of pending tasks in the event executor

For more information about Micrometer, see the Quarkus Micrometer Metrics guide.

1.3.3.4. Replace OkHttp tracing gRPC exporter with Vert.x

In Red Hat build of Quarkus 3.8, the OpenTelemetry (OTel) extension, **quarkus-opentelemetry** is enhanced to replace the default OTel exporter with a Red Hat build of Quarkus implementation built on top of Vert.x.

This eliminates the dependency on the OkHttp library. The exporter continues to be automatically wired with Contexts and Dependency Injection (CDI), so the **quarkus.otel.traces.exporter** property defaults to **cdi**.

For more information, see the Quarkus Using OpenTelemetry guide.

1.3.4. Security

1.3.4.1. Ability to split OIDC session cookies that exceed 4KB

With Red Hat build of Quarkus 3.8, you can split an OpenID Connect (OIDC) session cookie into smaller cookies if its content size exceeds 4KB.

Typically, a session cookie, which is encrypted by default, comprises a concatenation of three tokens, namely, ID, access, and refresh tokens. If its size is greater than 4KB, some browsers might be unable to handle it.

With this update, a session cookie exceeding 4KB in size is automatically split into multiple chunks.

For more information, see the Quarkus OIDC authorization code flow mechanism for protecting web applications guide.

1.3.4.2. Create OIDC SecurityIdentity instance after the HTTP request is complete

With Red Hat build of Quarkus 3.8, you can create OIDC **SecurityIdentity** instances for authentication purposes after a HTTP request completes.

With this version, the **quarkus-oidc** extension includes the **io.quarkus.oidc.TenantIdentityProvider** interface, which you can inject and call to convert a token to a **SecurityIdentity** instance after an HTTP request completes.

For more information, see the following Quarkus resources:

- OIDC authorization code flow mechanism for protecting web applications guide.
- Authentication after an HTTP request has completed section of the "OIDC bearer token authentication" guide.

1.3.4.3. Customization of OIDC JavaScript request checks

Red Hat build of Quarkus 3.8 introduces the OIDC **JavaScriptRequestChecker** bean that you can use to customize JavaScript request checks.

If you use single-page applications (SPAs) and JavaScript APIs such as **Fetch** or **XMLHttpRequest**(XHR) with Red Hat build of Quarkus web applications, you must set a header in the browser script to identify the request as a JavaScript request. However, the script engine can also set an engine-specific request header itself.

With this update, you can now register a custom **io.quarkus.oidc.JavaScriptRequestChecker** bean, which informs Red Hat build of Quarkus if the current request is a JavaScript request, thereby helping to avoid the creation of redundant headers.

1.3.4.4. Delayed OIDC JWK resolution now supported

Red Hat build of Quarkus 3.8 introduces support for delayed OIDC JSON Web Key (JWK) resolution, and you can now resolve keys the moment a token is available.

This release adds the **quarkus.oidc.jwks.resolve-early** configuration property. By default, this property is set to **true**, which means that JWK keys are resolved the moment you establish an OIDC provider connection.

However, you can set it to **false**, enabling the delayed resolution of keys simultaneously to token verification. The delayed JWK resolution uses the current token instead of the read-once approach at initialization time. For example, the token might provide information on how to resolve keys correctly.

1.3.4.5. 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**.

1.3.4.6. HTTP Permissions and Roles moved to runtime configuration

Red Hat build of Quarkus has updated to allow 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.

1.3.4.7. Mapping OIDC scope attribute to SecurityIdentity permissions in Bearer token authentication

If you use Bearer token authentication in Red Hat build of Quarkus, you can map **SecurityIdentity** roles from the verified JWT access tokens. Red Hat build of Quarkus 3.8 introduces the ability to map the OIDC scope parameter to permissions on the **SecurityIdentity** object.

For example, you can use **@PermissionAllowed("orders_read")** to request that JWT tokens have a **scope** claim with an **orders_read** value.

For more information, see the Quarkus OIDC Bearer token authentication guide.

1.3.4.8. Observing security events by using CDI

With Red Hat build of Quarkus 3.8, you can use Context and Dependency Injection (CDI) to observe authentication and authorization security events.

The CDI observers can be either synchronous or asynchronous and reporting of the following security events is supported:

- **io.quarkus.security.spi.runtime.AuthenticationFailureEvent**
- **io.quarkus.security.spi.runtime.AuthenticationSuccessEvent**
- **io.quarkus.security.spi.runtime.AuthorizationFailureEvent**
- **io.quarkus.security.spi.runtime.AuthorizationSuccessEvent**
- **io.quarkus.oidc.SecurityEvent**

For more information, see the Observe security events section of the Quarkus “Security tips and tricks” guide.

1.3.4.9. OIDC authorization code flow nonce supported

Red Hat build of Quarkus 3.8 introduces support for an OpenID Connect (OIDC) authorization code flow nonce feature.

When the OIDC authorization server issues an ID token in response to an authorization request, the ID token includes a **nonce** claim that must match the nonce authentication request query parameter. This feature helps to mitigate replay attacks by ensuring that the ID token is returned in response to the original authorization request and is not a replayed response.

1.3.4.10. OIDC request filters supported

With Red Hat build of Quarkus 3.8, you can customize OIDC client requests made by either **quarkus-oidc-client** or **quarkus-oidc** extensions to update or add new request headers by registering one or more **OidcRequestFilter** implementations.

For example, an OIDC request filter can analyze the request body and add its digest as a new header value.

For more information, see the OIDC request filters section of the Quarkus "OIDC authorization code flow mechanism for protecting web applications" guide.

1.3.4.11. New OIDC @TenantFeature annotation introduced to bind OIDC features to tenants

In Red Hat build of Quarkus 3.8, a new **@TenantFeature** annotation is introduced to bind OpenID Connect (OIDC) features to OIDC tenants.

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

1.3.4.12. OIDC token propagation supported

Red Hat build of Quarkus 3.8 introduces support for OIDC token propagation.

With this update, Red Hat build of Quarkus endpoints use REST clients to propagate the incoming OIDC access tokens to other secure endpoints that expect access tokens.

1.3.4.13. Role mappings for client certificates

Red Hat build of Quarkus 3.8 now supports mapping the Common Name (CN) attribute from a client's X.509 certificate to roles when using the Mutual TLS (mTLS) authentication mechanism. This functionality is activated under specific conditions:

This functionality is activated under specific conditions:

- If the mTLS authentication mechanism is enabled with either **quarkus.http.ssl.client-auth=required** or **quarkus.http.ssl.client-auth=request**
- The **application.properties** file references a role mappings file with the **quarkus.http.auth.certificate-role-properties** property.

The role mapping file is expected to have the **CN=role1,role,...,roleN** format and encoded by using UTF-8.

1.3.4.14. Support for token verification with an inlined certificate chain

Red Hat build of Quarkus 3.8 introduces the verification of OIDC bearer access tokens by using the X.509 certificate chain that is inlined in the token.

This means that you validate the certificate chain before you extract a public key from the leaf certificate. The leaf certificate refers to an X.509 certificate that is positioned at the end of a certificate chain. To verify the token's signature, you use this public key.

1.3.5. Tooling

1.3.5.1. Expanded update capability with OpenRewrite

quarkus update now supports OpenRewrite recipes for external Red Hat build of Quarkus extensions, expanding its capabilities beyond built-in extensions only. New recipes have been introduced, enhancing migration support for external extensions.

Be aware that Red Hat provides development support for using Quarkus development tools, including the Quarkus CLI to prototype, develop, test, and deploy Red Hat build of Quarkus applications. Red Hat does not support using Quarkus development tools in production environments.

1.3.6. Web

1.3.6.1. Enhanced /info endpoint through CDI integration

Applications using **quarkus-info** can now enrich the **/info** endpoint with additional data through CDI integration. This feature enhances the ability to customize and extend application diagnostics and metadata visibility.

NOTE

For more information, see the Quarkus community's CDI integration guide.

1.3.6.2. Improvements to the SSE support in REST Client Reactive

With Red Hat build of Quarkus 3.8, the REST Client's Server-Sent Events (SSE) capabilities are enhanced, 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.

1.3.6.3. ObjectMapper customization in REST Client Reactive Jackson

With Red Hat build of Quarkus 3.8, you can customize the **ObjectMapper** when using the **rest-client-reactive-jackson** extension. You can add the custom **ObjectMapper**, that only the client uses, by using the annotation **@ClientObjectMapper**.

IMPORTANT

For any customization action where you want to inherit the default settings, you must never modify the default object mapper **defaultObjectMapper**. You must create a copy instead. The **defaultObjectMapper** is the instance of **ObjectMapper** that Red Hat build of Quarkus itself configures, makes available as a CDI bean, and which RESTEasy Reactive and REST Client, among other applications, use by default.

For more information, see the Customizing the ObjectMapper in REST Client Reactive Jackson section of the Quarkus "Using the REST client" guide.

1.3.6.4. Path parameter support in @TestHTTPResource

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

1.4. SUPPORT AND COMPATIBILITY

You can find detailed information about the supported configurations and artifacts that are compatible with Red Hat build of Quarkus 3.8 and the high-level support lifecycle policy on the Red Hat Customer Support portal as follows:

- For a list of supported configurations, OpenJDK versions, and tested integrations, see Red Hat build of Quarkus Supported configurations.
- For a list of the supported Maven artifacts, extensions, and BOMs for Red Hat build of Quarkus, see Red Hat build of Quarkus Component details .
- For general availability, full support, and maintenance support dates for all Red Hat products, see Red Hat Application Services Product Update and Support Policy .

1.4.1. Product updates and support lifecycle policy

In Red Hat build of Quarkus, a feature release can be either a major or a minor release that introduces new features or support. Red Hat build of Quarkus release version numbers are directly aligned with the Long-Term Support (LTS) versions of the Quarkus community project. For more information, see the Long-Term Support (LTS) for Quarkus blog post.

The version numbering of a Red Hat build of Quarkus feature release matches the Quarkus community version on which it is based.

IMPORTANT

Red Hat does not release a productized version of Quarkus for every version the community releases. The cadence of the Red Hat build of Quarkus feature releases is about every six months.

Red Hat build of Quarkus provides full support for a feature release right up until the release of a subsequent version. When a feature release is superseded by a new version, Red Hat continues to provide a further six months of maintenance support for the release, as outlined in the following support lifecycle chart [Fig. 1].

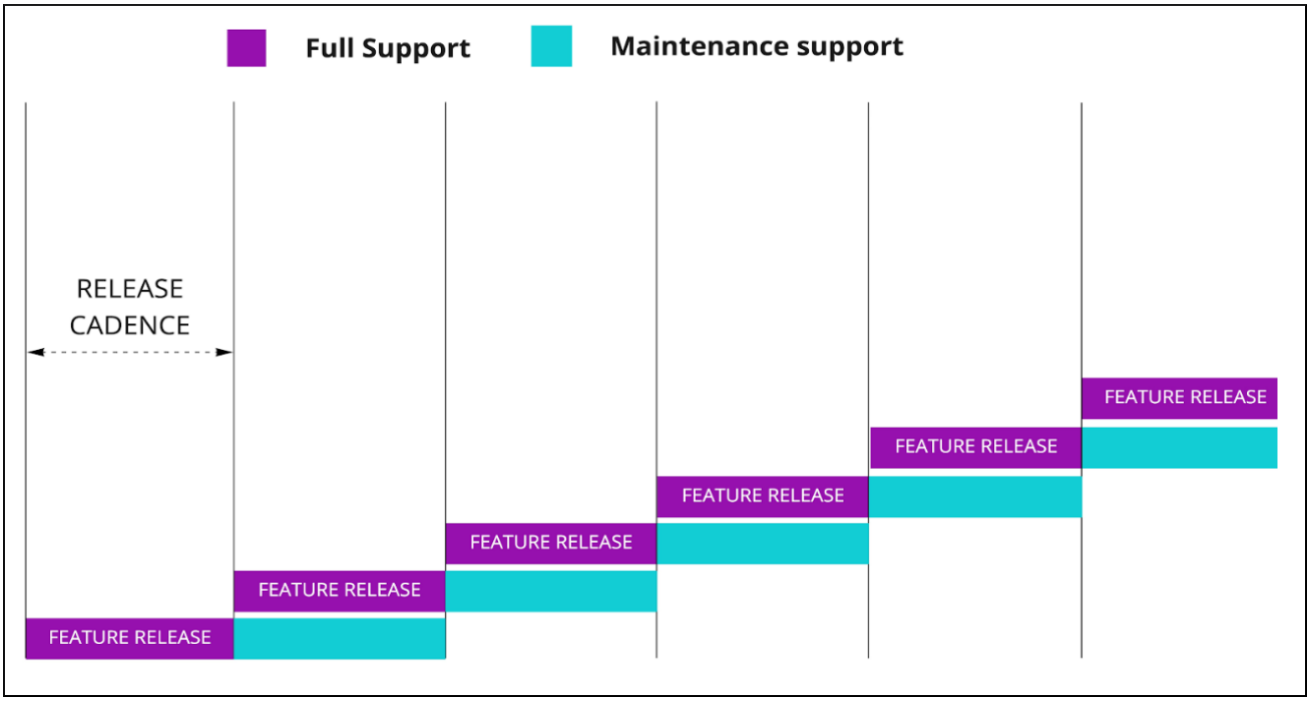


Figure 1. Feature release cadence and support lifecycle of Red Hat build of Quarkus

During the full support phase and maintenance support phase of a release, Red Hat also provides 'service-pack (SP)' updates and 'micro' releases to fix bugs and Common Vulnerabilities and Exposures (CVE).

New features in subsequent feature releases of Red Hat build of Quarkus can introduce enhancements, innovations, and changes to dependencies in the underlying technologies or platforms. For a detailed summary of what is new or changed in a successive feature release, see New features, enhancements, and technical changes.

While most of the features of Red Hat build of Quarkus continue to work as expected after you upgrade to the latest release, there might be some specific scenarios where you need to change your existing applications or do some extra configuration to your environment or dependencies. Therefore, before upgrading Red Hat build of Quarkus to the latest release, always review the Changes that affect compatibility with earlier versions and Deprecated components and features sections of the release notes.

For detailed information about the product lifecycle, timelines, and support policies of Red Hat build of Quarkus, log in to the Red Hat Customer Portal and see the Knowledgebase article, Red Hat build of Quarkus lifecycle and support policies.

1.4.2. Tested and verified environments

Red Hat build of Quarkus 3.8 is available on the following versions of Red Hat OpenShift Container Platform: 4.15, 4.12, and Red Hat Enterprise Linux 8.9.

For a list of supported configurations, log in to the Red Hat Customer Portal and see the Knowledgebase solution Red Hat build of Quarkus Supported configurations .

1.4.3. Development support

Red Hat provides development support for the following Red Hat build of Quarkus features, plugins, extensions, and dependencies:

Features

- Continuous Testing
- Dev Services
- Dev UI
- Local development mode
- Remote development mode

Plugins

- Maven Protocol Buffers Plugin

1.4.3.1. Development tools

Red Hat provides development support for using Quarkus development tools, including the Quarkus CLI and the Maven and Gradle plugins, to prototype, develop, test, and deploy Red Hat build of Quarkus applications.

Red Hat does not support using Quarkus development tools in production environments. For more information, see the Red Hat Knowledgebase article Development Support Scope of Coverage.

1.5. DEPRECATED COMPONENTS AND FEATURES

The components and features listed in this section are deprecated with Red Hat build of Quarkus 3.8. They are included and supported in this release. However, no enhancements will be made to these components and features, and they might be removed in the future.

For a list of the components and features that are deprecated in this release, log in to the Red Hat Customer Portal and view the Red Hat build of Quarkus Component details page.

1.5.1. Deprecation of DeploymentConfig

With Red Hat build of Quarkus 3.8, the **DeploymentConfig** object, deprecated in OpenShift, is also deprecated in Red Hat build of Quarkus. Now, **Deployment** is the default and preferred deployment kind for the **quarkus-openshift** extension.

If you redeploy applications that you deployed before by using **DeploymentConfig**, by default, those applications use **Deployment** but do not remove the previous **DeploymentConfig**. This leads to a deployment of both new and old applications, so, you must remove the **DeploymentConfig** manually.

However, if you want to continue to use **DeploymentConfig**, it is still possible to do so by explicitly setting **quarkus.openshift.deployment-kind** to **DeploymentConfig**.

For more information, see Deploying your Red Hat build of Quarkus applications to OpenShift Container Platform.

1.5.2. Deprecation of OpenShift Service Binding Operator

The OpenShift Service Binding Operator is deprecated in OpenShift Container Platform (OCP) 4.13 and later and is planned to be removed in a future OCP release.

1.5.3. Deprecation of quarkus-reactive-routes

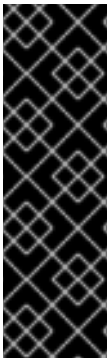
Starting with version 2.13, **quarkus-reactive-routes** is deprecated and is planned to be removed in a future version. SmallRye JWT no longer contains **quarkus-reactive-routes**; thus, its automatic inclusion is discontinued. To maintain functionality, add **quarkus-reactive-routes** to your build configurations.

1.5.4. Discontinuation of quarkus-test-infinispan-client artifact

The **quarkus-test-infinispan-client** artifact has been discontinued and is no longer part of Red Hat build of Quarkus. This change follows its redundancy, as it was not used outside the Quarkus core repository and had been replaced by Dev Services for Infinispan.

1.6. TECHNOLOGY PREVIEWS

This section lists features and extensions that are now available as a Technology Preview in Red Hat build of Quarkus 3.8.



IMPORTANT

Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat recommends that you do not use them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about Red Hat Technology Preview features, see Technology Preview Features Scope.

1.6.1. Hibernate Search management endpoint introduced

Red Hat build of Quarkus 3.8 exposes an HTTP management endpoint for Hibernate Search as a Technology Preview feature.

With this feature, you can trigger mass indexing of data and other maintenance tasks. By default, this endpoint is not enabled. To enable it, set the following configuration properties to **true**:

- **quarkus.management.enabled=true**
- **quarkus.hibernate-search-orm.management.enabled=true**

Red Hat build of Quarkus exposes the management endpoint under **/q/hibernate-search/reindex** on the management interface, which is exposed on port 9000 by default.

For more information, see the following resources:

- Quarkus Hibernate Search guide
- Quarkus Management interface reference guide

1.6.2. List of extensions that are in technology preview

- RESTEasy Reactive JAXB, **quarkus-resteasy-reactive-jaxb**. JAXB serialization support for RESTEasy Reactive. This extension is not compatible with the quarkus-resteasy extension, or any of the extensions that depend on it.

- SmallRye Stork, **quarkus-smallrye-stork**. SmallRye Stork is a dynamic service discovery and selection framework for locating and selecting service instances.
- Elasticsearch REST client, **quarkus-elasticsearch-rest-client**. Connect to an Elasticsearch cluster using the REST low level client.
- Hibernate Reactive, **quarkus-hibernate-reactive**. A reactive API for Hibernate ORM, supporting non-blocking database drivers and a reactive style of interaction with the database.
- MongoDB client, **quarkus-mongodb-client**. Connect to MongoDB in either imperative or reactive style.
- Reactive MS SQL client, **quarkus-reactive-mssql-client**. Connect to the Microsoft SQL Server database using the reactive pattern.
- Reactive Oracle client, **quarkus-reactive-oracle-client**. Connect to the Oracle database using the reactive pattern.
- Apache Kafka Streams, **quarkus-kafka-streams**. Implement stream processing applications based on Apache Kafka.
- Kubernetes Service Binding, **quarkus-kubernetes-service-binding**. Read runtime configuration based on the Kubernetes Service Binding Specification.
- OpenShift Client, **quarkus-openshift-client**. Interact with OpenShift and develop OpenShift Operators.
- OpenID Connect Token Propagation, **quarkus-oidc-token-propagation**. Use a Jakarta REST Client filter to propagate an incoming Bearer access token or token acquired from Authorization Code Flow as an HTTP Authorization Bearer token.
- OpenID Connect Token Propagation Reactive, **quarkus-oidc-token-propagation-reactive**. Use Reactive REST Client to propagate an incoming Bearer access token or a token acquired from Authorization Code Flow as an HTTP Authorization Bearer token.

1.7. 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.

1.7.1. Core

1.7.1.1. 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.

1.7.1.2. 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.

1.7.1.3. 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.

1.7.1.4. 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**.

1.7.1.5. 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

1.7.1.6. 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.

1.7.1.7. 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**.

1.7.1.8. 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**

1.7.1.9. 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.

1.7.1.10. 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**

1.7.1.11. 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.

1.7.2. Data

1.7.2.1. 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.

1.7.2.2. 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

1.7.2.3. 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.

1.7.2.4. 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.

1.7.3. Native

1.7.3.1. 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.

1.7.4. Observability

1.7.4.1. @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.

1.7.4.2. 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.

1.7.4.3. 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.

1.7.4.4. 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

1.7.5. Security

1.7.5.1. 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**.

1.7.5.2. 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.

1.7.5.3. 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**.

1.7.5.4. 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".

1.7.5.5. 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.

1.7.5.6. 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.

1.7.6. Standards

1.7.6.1. 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.

1.7.7. 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.

1.7.8. Web

1.7.8.1. 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.

1.7.8.2. 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.

1.8. KNOWN ISSUES

Review the following known issues for insights into Red Hat build of Quarkus 3.8 limitations and workarounds.

1.8.1. Infinispan client extension does not work on FIPS and Native Mandrel 23.1

In native mode, while using the native builder container for Red Hat build of Quarkus 3.8 from **registry.redhat.io/quarkus/mandrel-for-jdk-21-rhel8**, the Red Hat build of Quarkus Infinispan client extension does not work on Federal Information Processing Standards (FIPS)-enabled systems.

Workaround: Avoid using native mode with this native image builder. No workaround is available at this time.

1.8.2. quarkus-container-image-jib extension still uses OpenJDK image version 1.18

The **quarkus-container-image-jib** extension was not updated to use OpenJDK image version 1.19 and still uses OpenJDK image version 1.18.

Workaround: It is planned to be updated in a future release of Red Hat build of Quarkus.

1.8.3. Native build failures with Strimzi OAuth client update to 0.14.0

The Strimzi OAuth Client encounters a known issue due to an update of the **io.strimzi:strimzi-kafka-oauth** dependency to 0.14.0, leading to native build failures indicated by the following error: **Substitution target for io.smallrye.reactive.kafka.graal.Target_com_jayway_jsonpath_internal_DefaultsImpl is not loaded.**

Workaround: To workaround this issue, include the **io.strimzi:kafka-oauth-common** dependency in the classpath.

1.8.4. Missing native library for the Kafka Streams extension on AArch64

Applications that use the **quarkus-kafka-streams** extension have runtime failures on AArch64 systems due to the absence of the native library **librocksdbjni-linux-AArch64.so**. This issue throws a **java.lang.RuntimeException: librocksdbjni-linux-AArch64.so was not found inside JAR** error during application startup. This error prevents the successful initialization of the RocksDB component, which is crucial for Kafka Streams applications.

Workaround: No workaround is available at this time.

Example java.lang.RuntimeException: librocksdbjni-linux-AArch64.so error

```
09:32:54,059 INFO  [app] ERROR: Failed to start application (with profile [prod])
09:32:54,059 INFO  [app] java.lang.RuntimeException: Failed to start quarkus
09:32:54,060 INFO  [app]  at io.quarkus.runner.ApplicationImpl.doStart(Unknown Source)
09:32:54,060 INFO  [app]  at io.quarkus.runtime.Application.start(Application.java:101)
09:32:54,060 INFO  [app]  at
io.quarkus.runtime.ApplicationLifecycleManager.run(ApplicationLifecycleManager.java:111)
09:32:54,061 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:71)
09:32:54,061 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:44)
09:32:54,061 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:124)
09:32:54,062 INFO  [app]  at io.quarkus.runner.GeneratedMain.main(Unknown Source)
09:32:54,062 INFO  [app] Caused by: java.lang.ExceptionInInitializerError
09:32:54,063 INFO  [app]  at
io.quarkus.kafka.streams.runtime.KafkaStreamsRecorder.loadRocksDb(KafkaStreamsRecorder.java:14
)
09:32:54,063 INFO  [app]  at
io.quarkus.deployment.steps.KafkaStreamsProcessor$loadRocksDb1611413226.deploy_0(Unknown
Source)
09:32:54,063 INFO  [app]  at
io.quarkus.deployment.steps.KafkaStreamsProcessor$loadRocksDb1611413226.deploy(Unknown
Source)
09:32:54,064 INFO  [app]  ... 7 more
09:32:54,064 INFO  [app] Caused by: java.lang.RuntimeException: librocksdbjni-linux-AArch64.so
was not found inside JAR.
09:32:54,065 INFO  [app]  at
org.rocksdb.NativeLibraryLoader.loadLibraryFromJarToTemp(NativeLibraryLoader.java:118)
09:32:54,065 INFO  [app]  at
org.rocksdb.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.java:102)
09:32:54,065 INFO  [app]  at
org.rocksdb.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:82)
09:32:54,066 INFO  [app]  at org.rocksdb.RocksDB.loadLibrary(RocksDB.java:70)
09:32:54,066 INFO  [app]  at org.rocksdb.RocksDB.<clinit>(RocksDB.java:39)
09:32:54,067 INFO  [app]  ... 10 more
```

1.8.5. Missing native library for the Kafka Streams extension on Microsoft Windows

Applications that use the **quarkus-kafka-streams** extension on Microsoft Windows have runtime failures due to the absence of the native library **librocksdbjni-win64.dll**. This issue throws in a **java.lang.RuntimeException: librocksdbjni-win64.dll was not found inside JAR** error during the application startup process.

This error prevents the successful initialization of the RocksDB component, which is crucial for Kafka Streams applications.

Workaround: No workaround is available at this time.

Example java.lang.RuntimeException: librocksdbjni-win64.dll error

```
13:07:08,118 INFO  [app] ERROR: Failed to start application (with profile [prod])
13:07:08,118 INFO  [app] java.lang.RuntimeException: Failed to start quarkus
13:07:08,118 INFO  [app]  at io.quarkus.runner.ApplicationImpl.doStart(Unknown Source)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Application.start(Application.java:101)
13:07:08,118 INFO  [app]  at
io.quarkus.runtime.ApplicationLifecycleManager.run(ApplicationLifecycleManager.java:111)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:71)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:44)
```

```
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:124)
13:07:08,118 INFO  [app]  at io.quarkus.runner.GeneratedMain.main(Unknown Source)
13:07:08,118 INFO  [app] Caused by: java.lang.ExceptionInInitializerError
13:07:08,118 INFO  [app]  at
io.quarkus.kafka.streams.runtime.KafkaStreamsRecorder.loadRocksDb(KafkaStreamsRecorder.java:14
)
13:07:08,118 INFO  [app]  at
io.quarkus.deployment.steps.KafkaStreamsProcessor$loadRocksDb1611413226.deploy_0(Unknown
Source)
13:07:08,118 INFO  [app]  at
io.quarkus.deployment.steps.KafkaStreamsProcessor$loadRocksDb1611413226.deploy(Unknown
Source)
13:07:08,118 INFO  [app]  ... 11 more
13:07:08,118 INFO  [app] Caused by: java.lang.RuntimeException: librocksdbjni-win64.dll was not
found inside JAR.
13:07:08,118 INFO  [app]  at
org.rocksdb.NativeLibraryLoader.loadLibraryFromJarToTemp(NativeLibraryLoader.java:118)
13:07:08,118 INFO  [app]  at
org.rocksdb.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.java:102)
13:07:08,118 INFO  [app]  at
org.rocksdb.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:82)
13:07:08,118 INFO  [app]  at org.rocksdb.RocksDB.loadLibrary(RocksDB.java:70)
13:07:08,118 INFO  [app]  at org.rocksdb.RocksDB.<clinit>(RocksDB.java:39)
13:07:08,118 INFO  [app]  ... 14 more
```

1.8.6. Clarification on missing Vert.x classes during native builds

During native builds, developers might get **java.lang.ClassNotFoundException** errors for Vert.x classes such as **io.vertx.core.http.impl.Http1xServerResponse** and **io.vertx.core.parsetools.impl.RecordParserImpl**. These errors occur when building applications, including those that use the **quarkus-qpid-jms** extension, without including Vert.x as a direct or transitive dependency.

It is crucial to clarify that **quarkus-qpid-jms** does not use Vert.x directly. The issue arises from the **quarkus-netty** extension, which **quarkus-qpid-jms** uses. The **quarkus-netty** extension is responsible for registering these Vert.x classes for runtime initialization during native builds, without verifying their presence. This leads to the noted exceptions when there is no other extension in the build that introduces Vert.x.

These **ClassNotFoundException** errors are logged during the build process but do not impact the functionality of the applications. They are a result of the native build process and the way dependencies are handled within Red Hat build of Quarkus, specifically through the **quarkus-netty** module.

Example **java.lang.ClassNotFoundException** error

```
java.lang.ClassNotFoundException: io.vertx.core.http.impl.Http1xServerResponse
 at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
 at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
 at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:526)
 at
org.graalvm.nativeimage.builder/com.oracle.svm.hosted.NativeImageClassLoader.loadClass(NativeIma
geClassLoader.java:652)
 ... (further stack trace details)
java.lang.ClassNotFoundException: io.vertx.core.parsetools.impl.RecordParserImpl
 at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
```

```
at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:526)
at
org.graalvm.nativeimage.builder/com.oracle.svm.hosted.NativeImageClassLoader.loadClass(NativeIma
geClassLoader.java:652)
... (further stack trace details)
```

Workaround: To prevent these log entries and ensure all dependencies are properly recognized, you can optionally add the **quarkus-vertx** extension to your project.

1.8.7. AArch64 support limitations in JVM mode testing on OpenShift

The testing pipeline for JVM mode on Red Hat OpenShift Container Platform with AArch64, operational since Red Hat build of Quarkus 3.2, has a few known limitations around AArch64:

- Red Hat Serverless is not supported on AArch64. A feature request for support of Red Hat Serverless on OpenShift clusters running on the AArch64 architecture is tracked in SRVCOM-2472, with plans to include support in Serverless 1.33.
- Red Hat AMQ Streams is not supported on AArch64. Because AMQ Streams is not yet supported on AArch64, the support for this integration has not been tested yet. This issue is currently not tracked in Red Hat's issue management system.
- Red Hat Single Sign-On is not supported on AArch64. Because Red Hat Single Sign-On and Red Hat build of Keycloak are not supported on AArch64 yet, integration with Red Hat build of Quarkus applications has not been tested yet.
- Service Binding is not supported on AArch64. Because the bound services supported in the Technology Preview of service binding integration with Red Hat build of Quarkus are not supported on AArch64 yet, this integration has not been tested yet. Additionally, the OpenShift Service Binding Operator is deprecated in OpenShift Container Platform (OCP) 4.13 and later and is planned to be removed in a future OCP release.

AArch64 support is limited to the Red Hat Universal Base Image (UBI) containers and does not extend to bare-metal environments.

Workaround: No workarounds are available at this time.

1.8.8. Dependency on org.apache.maven:maven:pom:3.6.3 might cause proxy issues

The dependency on **org.apache.maven:maven:pom:3.6.3** might be resolved when using certain Quarkus extensions. This is not specific to the Gradle plugin but impacts any project with **io.smallrye:smallrye-parent:pom:37** in its parent Project Object Model (POM) hierarchy. This dependency can cause build failures for environments behind a proxy that restricts access to **org.apache.maven** artifacts with version 3.6.x. None of the binary packages from Maven 3.6.3 are downloaded as dependencies of the Quarkus core framework or supported Quarkus extensions.

Workaround: No workaround is available at this time.

For more information, see QUARKUS-1025 - Gradle plugin drags in maven core 3.6.x .

1.9. UPDATES FOR RED HAT BUILD OF QUARKUS 3.8.5 SP1

Red Hat build of Quarkus 3.8 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.8.5.SP1.

1.9.1. Bug fixes

To view the issues that have been resolved for this release, see Red Hat build of Quarkus 3.8.5 SP1 bug fixes.

1.9.2. Advisories

Before you start using and deploying Red Hat build of Quarkus 3.8.5.SP1, review the following advisory related to the release.

- RHBA-2024:4723

1.10. UPDATES FOR RED HAT BUILD OF QUARKUS 3.8.5

Red Hat build of Quarkus 3.8 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.8.5.SP1.

1.10.1. Bug fixes

To view the issues that have been resolved for this release, see Red Hat build of Quarkus 3.8.5 bug fixes .

1.10.2. Security fixes

- CVE-2024-34447 **org.bouncycastle-bctls**: **org.bouncycastle**: Use of incorrectly-resolved name or reference
- CVE-2024-30171 **org.bouncycastle-bcprov-jdk18on**: BouncyCastle vulnerable to a timing variant of Bleichenbacher (Marvin Attack)
- CVE-2024-29857 **org.bouncycastle:bcprov-jdk18on**: **org.bouncycastle**: Importing an EC certificate with crafted F2m parameters might lead to Denial of Service
- CVE-2024-30172 **org.bouncycastle-bcprov-jdk18on**: Infinite loop in ED25519 verification in the ScalarUtil class

1.10.3. Advisories

Before you start using and deploying Red Hat build of Quarkus 3.8.5, review the following advisory related to the release.

- RHSA-2024:4326

1.11. UPDATES FOR RED HAT BUILD OF QUARKUS 3.8.4

Red Hat build of Quarkus 3.8 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.8.5.SP1.

1.11.1. Bug fixes

To view the issues that have been resolved for this release, see Red Hat build of Quarkus 3.8.4 bug fixes .

1.11.2. Security fixes

- CVE-2024-2700 **io.quarkus/quarkus-core**: Leak of local configuration properties into Quarkus applications
- CVE-2024-29025 **io.netty/netty-codec-http**: Allocation of Resources Without Limits or Throttling

1.11.3. Advisories

Before you start using and deploying Red Hat build of Quarkus 3.8.4, review the following advisory related to the release.

- RHSA-2024:2106

1.12. UPDATES FOR RED HAT BUILD OF QUARKUS 3.8.3

Red Hat build of Quarkus 3.8 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.8.5.SP1.

1.12.1. Bug fixes

- QUARKUS-2289 Unable to mount file into container on MacOS
- QUARKUS-3206 RabbitMQ TCP connections fail to close while using the Dev Mode Live-Reload feature of Quarkus
- QUARKUS-3804 Regression in 2.13.9 when using domain socket in Vert.x
- QUARKUS-4061 Quarkus maven plugin creates projects with non-relocated dependencies
- QUARKUS-4065 HTTP endpoint returns empty body when using RHBQ, but not upstream Quarkus

1.12.2. Advisories

Before you start using and deploying Red Hat build of Quarkus 3.8.3, review the following advisory related to the release.

- RHEA-2024:2057

1.13. ADDITIONAL RESOURCES

- Migrating applications to Red Hat build of Quarkus 3.8 guide.

- Getting Started with Red Hat build of Quarkus

Revised on 2024-07-23 13:28:17 UTC