Red Hat build of Quarkus 3.2

Release Notes for Red Hat build of Quarkus 3.2

Last Updated: 2024-04-04

Red Hat build of Quarkus 3.2 Release Notes for Red Hat build of Quarkus 3.2

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

MAKING OPEN SOURCE MORE INCLUSIVE 5

CHAPTER 1. RELEASE NOTES FOR RED HAT BUILD OF QUARKUS 3.2 6
1.1. ABOUT RED HAT BUILD OF QUARKUS 6
1.2. DIFFERENCES BETWEEN THE RED HAT BUILD OF QUARKUS COMMUNITY VERSION AND RED HAT BUILD OF QUARKUS 6
1.3. NEW FEATURES, ENHANCEMENTS, AND TECHNICAL CHANGES 8
1.3.1. Cloud 8
1.3.1.1. Cached section capabilities introduced in the Qute templating engine 8
1.3.1.2. Kubernetes client upgraded to version 6.7.2 8
1.3.2. Core 8
1.3.2.1. Build-time analytics (user telemetry) support 8
1.3.2.2. Infinispan annotation caching support 8
1.3.2.3. Management Network interface integration 9
1.3.2.4. Most of the quarkus-cache configurations are now runtime 9
1.3.2.5. Multiple SMTP mailer support 9
1.3.2.6. Revamp of the development UI 9
1.3.2.7. Scheduler programmatic API 9
1.3.2.8. Update tool integration 9
1.3.3. Data 10
1.3.3.1. Hibernate ORM extension now incorporates automated IN clause parameter padding 10
1.3.3.2. Hibernate ORM upgraded to version 6.2 10
1.3.3.3. Hibernate Search upgraded to version 6.2 10
1.3.3.4. Oracle JDBC driver upgraded to version 23.2.0.0 10
1.3.3.5. Reactive datasources now support CredentialsProvider values 10
1.3.4. Native 11
1.3.4.1. Red Had build of Red Hat build of Quarkus Native builder upgraded to version 23 11
1.3.5. Observability 11
1.3.5.1. OpenTelemetry SDK autoconfiguration 11
1.3.6. Security 11
1.3.6.1. Custom claim types in test dependencies now supported 11
1.3.6.2. OpenID Connect (OIDC) Front-channel Logout now supported 11
1.3.6.3. OpenID Connect token verification customization 11
1.3.6.4. Security annotations can be used as meta-annotations 11
1.3.6.5. Simplified OIDC multitenancy resolution for static tenants 12
1.3.6.6. SmallRye configuration properties expansion in @RolesAllowed 12
1.3.7. Standards 12
1.3.7.1. Eclipse MicroProfile 6 integration 12
1.3.7.2. Jakarta EE 10 integration 12
1.3.8. Tooling 12
1.3.8.1. Apache Maven version 3.9 supported 12
1.3.8.2. Deploy tool integration 12
1.3.8.3. Red Hat build of Quarkus CLI enhancements for building and pushing container images 13
1.3.8.3.1. Building a container image 13
1.3.8.3.2. Pushing a container image 13
1.3.9. Web 13
1.3.9.1. Federation support for SmallRye GraphQL 13
1.3.9.2. Filtering by named queries in REST Data with the Panache extension 13
1.3.9.3. gRPC exception handling 14
1.3.9.4. gRPC extension migration to Vert.x gRPC 14
1.3.9.5. Programmatic API to create Reactive REST clients 14

1.3.9.6. RESTEasy Reactive HTTP response headers and status codes can be customized 14
1.3.9.7. The @Encoded annotation on REST Client Reactive is now supported 15
1.4. SUPPORT AND COMPATIBILITY 15
1.4.1. Product updates and support lifecycle policy 15
1.4.2. Tested and verified environments 16
1.4.3. Development support 17
1.4.3.1. Development tools 18
1.5. DEPRECATED COMPONENTS AND FEATURES 18
1.5.1. Deprecation of Red Hat build of Quarkus support for Java 11 18
1.6. TECHNOLOGY PREVIEWS 18
1.6.1. Enhanced component testing 19
1.6.2. Hibernate Reactive upgraded to version 2 19
1.6.3. quarkus-opentelemetry-exporter-otlp merged into quarkus-opentelemetry 19
1.6.4. Support for storing transaction logs in a database 19
1.7. CHANGES THAT AFFECT COMPATIBILITY WITH EARLIER VERSIONS 20
1.7.1. Cloud 20
1.7.1.1. Upgrade to the Kubernetes client that is included with Red Hat build of Quarkus 20
1.7.1.2. Improved logic for generating TLS-based container ports 21
1.7.1.3. Removal of some Kubernetes and OpenShift properties 21
1.7.2. Core 21
1.7.2.1. Upgrade to Jandex 3 21
1.7.2.2. Migration path for users of Jandex API 22
1.7.2.3. Removal of io.quarkus.arc.config.ConfigProperties annotation 22
1.7.2.4. Interceptor binding annotations declared on private methods now generate build failures 22
1.7.2.5. Removal of the @AlternativePriority annotation 23
1.7.2.6. Testing changes: Fixation of the Mockito subclass mockmaker 23
1.7.2.7. Update to the minimum supported Maven version 24
1.7.2.8. Removal of quarkus-bootstrap-maven-plugin 24
1.7.2.9. Mutiny 2 moves to Java Flow 24
1.7.3. Data 24
1.7.3.1. Removal of Hibernate ORM with Panache methods 24
1.7.3.2. Enhancement in Hibernate ORM: Automated IN clause parameter padding 24
1.7.3.3. New dependency: Hibernate Reactive 2 and Hibernate ORM 6.2 25
1.7.3.4. Hibernate Search changes 25
1.7.3.5. Hibernate Validator - Validation.buildDefaultValidatorFactory() now returns a ValidatorFactory managed by Quarkus 26
1.7.3.6. Quartz jobs class name change 26
1.7.3.7. Deprecation of QuarkusTransaction.run and QuarkusTransaction.call methods 26
1.7.3.8. Renamed Narayana transaction manager property 28
1.7.4. Messaging 28
1.7.4.1. Removal of vertx-kafka-client dependency from SmallRye Reactive Messaging 28
1.7.5. Native 28
1.7.5.1. Native compilation - Native executables and .so files 28
1.7.5.2. Native Compilation - Work around missing CPU features 28
1.7.5.3. Testing changes: Removal of some annotations 29
1.7.6. Observability 29
1.7.6.1. Deprecated OpenTracing driver is replaced by OpenTelemetry 29
1.7.6.2. Default metrics format in Micrometer now aligned with Prometheus 30
1.7.6.3. Changes in the OpenTelemetry extension and removal of some sampler-related properties 30
1.7.7. Security 32
1.7.7.1. Removal of CORS filter default support for using a wildcard as an origin 32
1.7.7.2. OpenAPI CORS support change 33
1.7.7.3. Encryption of OIDC session cookie by default 33

1.7.7.4. Default SameSite attribute set to Lax for OIDC session cookie 34
1.7.7.5. The OIDC ID token audience claim is verified by default 34
1.7.7.6. Removal of default password for the JWT key and keystore 34
1.7.8. Web 34
1.7.8.1. Changes to RESTEasy Reactive multipart 34
1.7.8.2. Enhanced JAXB extension control 35
1.8. BUG FIXES 37
1.8.1. Security fixes resolved in Red Hat build of Quarkus 3.2.11 37
1.8.2. Security fixes resolved in Red Hat build of Quarkus 3.2.10 37
1.8.3. Security fixes resolved in Red Hat build of Quarkus 3.2.9.SP1 37
1.8.4. Security fixes resolved in Red Hat build of Quarkus 3.2.9 37
1.8.5. Security fixes resolved in Red Hat build of Quarkus 3.2.6 38
1.8.6. Other enhancements and bug fixes 38
1.9. KNOWN ISSUES 41
1.9.1. Using CDI interceptors to resolve multitenant OIDC configuration fails due to security fix in version 3.2.9.SP1 41
1.9.2. Podman 4.6 and later does not work with SELinux and Testcontainers library 41
1.9.3. Containers spawned by Testcontainers occasionally fail 42
1.9.4. HTTP/1.1 Upgrades to H2C fail under specific flow control conditions 42
1.9.5. Reactive Oracle datasource fails with specific Oracle JDBC driver versions 42
1.9.6. Community artifacts are used for native Vert.x dependencies on specific platforms 43
1.9.7. Red Hat build of Quarkus Kafka Streams are not supported on Windows due to a missing library 43
1.9.8. Community artifacts are used for some native dependencies on specific platforms 43
1.9.9. Dependency on org.apache.maven:maven:pom:3.6.3 might cause proxy issues 44
1.9.10. Build failure in the starter application generated by JBang with the Red Hat extension registry 44
1.10. ADVISORIES RELATED TO THIS RELEASE 44
1.10.1. Red Hat build of Quarkus 3.2.11 44
1.10.2. Red Hat build of Quarkus 3.2.10 44
1.10.3. Red Hat build of Quarkus 3.2.9.SP1 45
1.10.4. Red Hat build of Quarkus 3.2.9 45
1.10.5. Red Hat build of Quarkus 3.2.6 45
1.11. ADDITIONAL RESOURCES 45

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 3.2

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 3.2.

These include the following notable changes:

- Jakarta EE 10 integration
- Eclipse MicroProfile 6 integration
- Hibernate ORM upgraded to version 6.2
- Quarkus CLI enhancements for building and pushing container images
- Deprecation of Red Hat build of Quarkus support for Java 11

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 RED HAT BUILD OF 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.
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	Each feature release of Red Hat build of Quarkus is fully supported for approximately one year up until the next feature release. When a feature release is superseded by a new version, Red Hat continues to provide a further six months of maintenance support. For more information, see Support and compatibility.
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. In the Quarkus community version, CVEs, and bug fixes are typically made available in the latest release only.
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 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 provides an overview of the new features, enhancements, and technical changes introduced in Red Hat build of Quarkus 3.2.

1.3.1. Cloud

1.3.1.1. Cached section capabilities introduced in the Qute templating engine

In Red Hat build of Quarkus 3.2, the Qute templating engine is enhanced to provide the ability to cache those parts of a template that rarely change, which can help increase efficiency.

To use the cached sections feature, use the **quarkus-cache** extension, where **CacheSectionHelper** is registered and configured automatically.

For more information, see the Cached section part of the "Qute reference" guide.

1.3.1.2. Kubernetes client upgraded to version 6.7.2

The Kubernetes client included with Red Hat build of Quarkus has been upgraded from version 5.12 to 6.7.2. This upgrade offers enhanced features and improved support for developing cloud-native applications. For more information, see the Kubernetes client - Migration from 5.x to 6.x guide.

1.3.2. Core

1.3.2.1. Build-time analytics (user telemetry) support

Red Hat build of Quarkus 3.2 introduces a build-time analytics feature. This feature provides usage information about Red Hat build of Quarkus during the application's build time, but not during its run time.

The usage analytics report provides anonymous information, such as which operating systems, JAVA versions, build systems, and extensions are used. Usage analytics can help Red Hat better understand how Red Hat build of Quarkus is used and how it can be improved.

To opt-in, run Red Hat build of Quarkus in dev mode. The first time you do so, you are asked if you want to opt-in to contributing anonymous build-time data to the Quarkus community. This data will NOT be collected when you run a Red Hat build of Quarkus application in, for example, a production environment.

For more information, see the Quarkus usage analytics guide in the Quarkus community. For more information about what data is collected, see the Telemetry data collection notice.

1.3.2.2. Infinispan annotation caching support

The Red Hat build of Quarkus Infinispan extension now supports the declarative caching API, allowing annotation-based caching control in CDI-managed beans.

1.3.2.3. Management Network interface integration

The Management Network interface is a dedicated channel for managing and monitoring your applications, including providing endpoints for various management tasks such as health checks and metrics.

1.3.2.4. Most of the quarkus-cache configurations are now runtime

Most of the **quarkus-cache** extension configuration has been made runtime, allowing you to define the cache configuration at application startup. Certain configuration properties can be changed at runtime through API calls.

1.3.2.5. Multiple SMTP mailer support

Some applications require that emails be sent through different SMTP servers. In Red Hat build of Quarkus 3.2, you can now configure several mailers and send emails by using multiple SMTP servers.

For more information, see the Multiple mailer configuration section of the "Mailer reference" guide.

1.3.2.6. Revamp of the development UI

Red Hat build of Quarkus 3.2 introduces significant changes and enhancements to the development UI, including a graphical interface for streamlined management and monitoring of application components during development. This aids in efficient log navigation, metrics tracking, and endpoint management.

1.3.2.7. Scheduler programmatic API

With the Red Hat build of Quarkus 3.2 release, you can schedule jobs programmatically by using the new Scheduler programmatic API.

To schedule a job programmatically, you inject **io.quarkus.scheduler.Scheduler**. You can also remove jobs that are scheduled programmatically.

For more information, see the Programmatic scheduling section of the Quarkus “Scheduler reference” guide.

1.3.2.8. Update tool integration

The Red Hat build of Quarkus update tool simplifies the upgrade of your applications by automatically updating project dependencies, configurations, and code to match the latest Red Hat build of Quarkus version. It streamlines the migration process, ensuring compatibility and reducing the effort required to stay up-to-date.

To use the tool, run the **quarkus update** command in your project directory, following the interactive prompts to update your application.

IMPORTANT

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

For more information, see the Migrating applications to Red Hat build of Quarkus version 3.2 guide.

1.3.3. Data

1.3.3.1. Hibernate ORM extension now incorporates automated IN clause parameter padding

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

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

1.3.3.2. Hibernate ORM upgraded to version 6.2

Red Hat build of Quarkus now includes and supports Hibernate ORM version 6.2, therefore significantly upgrading the main persistence layer.

Hibernate ORM 6.2 brings many improvements and new features compared with version 5.6, but also some breaking changes.

For more information, see the following resources:

- Changes that affect compatibility with earlier versions
- Quarkus Migration Guide 3.0: Hibernate ORM 5 to 6 migration guide
- Quarkus Using Hibernate ORM and Jakarta Persistence guide

1.3.3.3. Hibernate Search upgraded to version 6.2

In Red Hat build of Quarkus 3.2, Hibernate Search has been upgraded to version 6.2.

Hibernate Search offers indexing and full-text search capabilities to your Red Hat build of Quarkus applications. Version 6.2 introduces enhancements, new features, and some notable changes to how Red Hat build of Quarkus applications handle default values for geo-point fields.

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

To learn more about what is new in Hibernate Search, see the Hibernate Search release notes.

1.3.3.4. Oracle JDBC driver upgraded to version 23.2.0.0

The Oracle JDBC driver has been upgraded to version 23.2. Customers using Oracle DB should note that older versions of the Oracle JDBC driver are not necessarily compatible with the latest Oracle DB release.

1.3.3.5. Reactive datasources now support CredentialsProvider values

Reactive datasources can now modify **CredentialsProvider** values, enhancing security and configurability. This allows real-time credential updates for authentication, ensuring data access security while maintaining application availability and minimizing operational disruptions.

1.3.4. Native

1.3.4.1. Red Had build of Red Hat build of Quarkus Native builder upgraded to version 23

Besides improved performance, version 23 brings improved generation of debug information, extended support for Java Flight Recorder (JFR) events, and experimental support for JFR event streams. It also introduces experimental support for Java Management Extensions (JMX).

Environment variables must now be passed to Mandrel through the new native-image option **-E<env-var-key>[=<env-var-value>]**.

Red Hat build of Quarkus Native builder now defaults to targeting **x86-64-v3**, the processor-specific application binary interface (psABI) on the AMD64 architecture, and introduces support for a new **-march** option for compiling to a more compatible native image for older architectures.

For more information, see the Work around missing CPU features article in the Red Hat build of Quarkus community "Native reference" guide.

1.3.5. Observability

1.3.5.1. OpenTelemetry SDK autoconfiguration

Red Hat build of Quarkus introduces OpenTelemetry SDK autoconfiguration, simplifying the integration of distributed tracing and observability. It automates OpenTelemetry SDK setup based on Red Hat build of Quarkus extensions, eliminating manual configuration and optimizing trace and metric collection.

1.3.6. Security

1.3.6.1. Custom claim types in test dependencies now supported

In Red Hat build of Quarkus 3.2, the **quarkus-test-security-jwt** and **quarkus-test-security-oidc** test dependencies are enhanced to support custom claim types.

With this update, you can improve the test coverage of applications that use custom JWT token claims.

1.3.6.2. OpenID Connect (OIDC) Front-channel Logout now supported

The inclusion of OIDC front-channel logout support in Red Hat build of Quarkus complements the already-supported OIDC back-channel logout, enabling the logout of users across multiple services in a distributed environment.

1.3.6.3. OpenID Connect token verification customization

Within Red Hat build of Quarkus 3.2, the option to tailor the OIDC token verification process is available. This customization permits the preprocessing of legacy token headers, commonly issued by OIDC providers like Microsoft Azure, prior to signature validation.

1.3.6.4. Security annotations can be used as meta-annotations

You can combine **@TestSecurity** and **@JwtSecurity** in a meta-annotation; for example:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.METHOD })
```

```
@TestSecurity(user = "userOidc", roles = "viewer")
@OidcSecurity(introspectionRequired = true,
    introspection = {
        @TokenIntrospection(key = "email", value = "user@gmail.com")
    }
)
public @interface TestSecurityMetaAnnotation {
}
```

This combination is useful if the same set of security settings are required in multiple test methods.

1.3.6.5. Simplified OIDC multitenancy resolution for static tenants

In an OIDC multitenancy setup where you set multiple tenant configurations in the **application.properties** file, you must specify how the tenant identifier gets resolved by registering the **TenantResolver** interface implementation.

Red Hat build of Quarkus 3.2 introduces a convention-based static tenant resolution, where the last path segment of the current HTTP request URL is used as a tenant identifier. For example, if the request URL ends with /**keycloak**, then a static tenant configuration whose tenant ID is **keycloak** is selected.

By using this option, you can reduce boilerplate code in simple multitenant configurations.

For more information, see the Configuring the application section of the Quarkus “Using OpenID Connect (OIDC) multitenancy” guide.

1.3.6.6. SmallRye configuration properties expansion in @RolesAllowed

The Red Hat build of Quarkus @RolesAllowed annotation supports dynamic role names through configuration properties, enhancing access control. This annotation restricts access based on SecurityIdentity (user roles), offering adaptable and configurable access control without code changes.

1.3.7. Standards

1.3.7.1. Eclipse MicroProfile 6 integration

Red Hat build of Quarkus 3.2 introduces integration of Eclipse MicroProfile 6, which enhances microservice development with up-to-date specifications for improved observability, OpenAPI, and JWT.

1.3.7.2. Jakarta EE 10 integration

Red Hat build of Quarkus 3.2 introduces the integration of Jakarta EE 10, which provides developers with access to the current APIs and specifications.

1.3.8. Tooling

1.3.8.1. Apache Maven version 3.9 supported

Red Hat build of Quarkus 3.2 adds support for Maven version 3.9 so that developers can use the latest Maven features. Maven version 3.8.6 or later remains supported.

1.3.8.2. Deploy tool integration

The **quarkus deploy** command in Quarkus facilitates deploying applications to various cloud platforms, containers, and serverless environments. It generates optimized container images and adapts the application to the target platform, ensuring efficient and reliable deployment.

To use the tool, run **quarkus deploy** followed by the desired deployment target and configuration options, allowing for seamless application deployment without manual configuration.

1.3.8.3. Red Hat build of Quarkus CLI enhancements for building and pushing container images

In Red Hat build of Quarkus 3.2, it is now easier to build and push container images. For more information, see the Container images section of the "Building Red Hat build of Quarkus apps with the **quarkus** command line interface" guide.

1.3.8.3.1. Building a container image

For example, you no longer need to adjust your **pom.xml** project configuration to build a docker image to add or remove container image extensions. Instead, you only need to run the following command:

```
quarkus image build docker
```

1.3.8.3.2. Pushing a container image

The **image push** command is similar to the **image build** command and provides some basic options to push images to a target container registry.

```
quarkus image push --registry=<image registry> --registry-username=<registry username> --registry-password-stdin
```

IMPORTANT

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

For a detailed list of the Red Hat build of Quarkus CLI image commands and how to use them, see the following resources:

- Building Red Hat build of Quarkus apps with the Red Hat build of Quarkus CLI
- Blog: Dev productivity - Red Hat build of Quarkus CLI

1.3.9. Web

1.3.9.1. Federation support for SmallRye GraphQL

Quarkus' SmallRye GraphQL now supports Apollo Federation 2 subgraph exposure, enabling federated GraphQL schema creation. This empowers unified GraphQL APIs by aggregating data from independently deployed GraphQL services, simplifying complex application development.

1.3.9.2. Filtering by named queries in REST Data with the Panache extension

Filtering by named queries in Red Hat build of Quarkus' REST Data with Panache extension streamlines data retrieval by applying predefined queries to REST endpoints, enhancing performance and code maintainability for efficient database interaction through REST APIs.

When listing entities, you can filter by a named query defined in your entity by the **@NamedQuery** annotation.

An example of the named query

```
@Entity
@NamedQuery(name = "Person.containsInName", query = "from Person where name like
CONCAT('%', CONCAT(:name, '%'))")
public class Person extends PanacheEntity {
  String name;
}
```

Next, you can set a query parameter **namedQuery** when listing the entities using the generated resource. Use the name of the desired query, such as calling **http://localhost:8080/people?namedQuery=Person.containsInName&name=ter**, which would retrieve all persons with names containing "ter".

1.3.9.3. gRPC exception handling

The gRPC exception handling facilitates more robust error management in gRPC services, enhancing application reliability and debugging of gRPC-based applications.

This feature enables passing the error message as a trailer. The gRPC client will receive a specific error message from the server in case of issues rather than a generic "server exception."

1.3.9.4. gRPC extension migration to Vert.x gRPC

The migration of the gRPC extension to Vert.x's implementation enhances alignment with the Vert.x ecosystem, offering an efficient way to develop a microservice by using gRPC communication.

This implementation allows a single HTTP server configuration that removes duplicity from your Red Hat build of Quarkus Security configuration.

1.3.9.5. Programmatic API to create Reactive REST clients

In previous releases, you could only create Reactive REST clients by configuring them in the **application.properties** file. This approach might have been problematic if you wanted to create dynamic clients.

With Red Hat build of Quarkus 3.2, you can now create Reactive REST clients programmatically by using the new Quarkus-specific API, **QuarkusRestClientBuilder**.

The **QuarkusRestClientBuilder** interface programmatically creates Reactive REST clients with additional configuration options.

For more information, see the Programmatic client creation with QuarkusRestClientBuilder section of the Quarkus "Using the REST Client" guide.

1.3.9.6. RESTEasy Reactive HTTP response headers and status codes can be customized

In Red Hat build of Quarkus 3.2, the RESTEasy Reactive client is enhanced to provide more flexibility when streaming responses.

With this update, you can customize HTTP headers, HTTP responses, and status codes.

For more information, see the Customizing headers and status section of the Quarkus “Writing REST services with RESTEasy Reactive” guide.

1.3.9.7. The @Encoded annotation on REST Client Reactive is now supported

Red Hat build of Quarkus 3.2 introduces support for the **@Encoded** annotation on REST Client Reactive. With this update, the **@Encoded** annotation impacts the decoding of parameters, such as the **PATH** and **QUERY** parameters.

For more information, see the following resources:

- Jakarta EE Platform API - Annotation Type Encoded
- Quarkus Using the REST Client guide

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 3.2 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. The version numbering of a Red Hat build of Quarkus feature release matches the Quarkus community version that it is based on. For more information, see the Long-Term Support (LTS) for Quarkus blog post.

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].

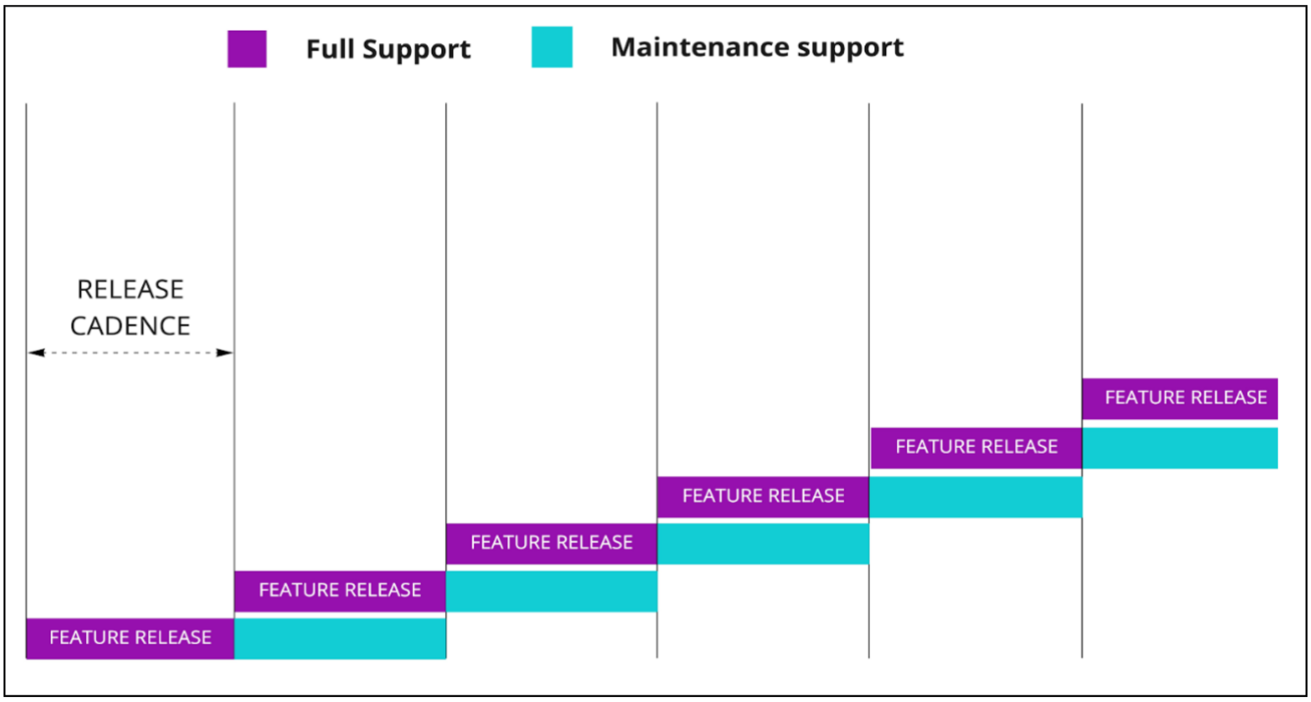

Figure 1. Feature release cadence and support lifecycle of Red Hat build of Quarkus

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.

1.4.2. Tested and verified environments

Red Hat build of Quarkus 3.2 is available on the following versions of Red Hat OpenShift Container Platform and Red Hat Enterprise Linux 8, with the listed supported installation container images.

Please note the **Tested** and **Supported** columns for each CPU architecture. To get the support status of versions in the following table whose deployment environments are not tested, see Support of Red Hat Middleware products and components on Red Hat OpenShift Container Platform in the Red Hat Knowledgebase.

The values captured in the following table represent the valid state at the time of release.

Table 1.1. Supported deployment environments for Red Hat build of Quarkus 3.2 on Red Hat OpenShift Container Platform and Red Hat Enterprise Linux

Platform	Architecture	Container Image / JVM	Tested	Supported

OpenShift Container Platform 4.10	AMD64 and Intel 64 (x86_64)	Red Hat build of OpenJDK 11 & 17	Yes	Yes
OpenShift Container Platform 4.10	IBM Power (ppc64le) and IBM Z (s390x)	Red Hat build of OpenJDK 11 & 17	No	Yes
OpenShift Container Platform 4.11	AMD64 and Intel 64 (x86_64)	Red Hat build of OpenJDK 11 & 17	No	Yes - See the Support article
OpenShift Container Platform 4.11	IBM Power (ppc64le) IBM Z (s390x)	Red Hat build of OpenJDK 11 & 17	No	Yes - See the Support article
OpenShift Container Platform 4.12	AMD64 and Intel 64 (x86_64)	Red Hat build of OpenJDK 11 & 17	No	Yes - See the Support article
OpenShift Container Platform 4.12	IBM Power (ppc64le) and IBM Z (s390x)	Red Hat build of OpenJDK 11 & 17	No	Yes - See the Support article
OpenShift Container Platform 4.13	AMD64 and Intel 64 (x86_64)	Red Hat build of OpenJDK 11 & 17	Yes	Yes
OpenShift Container Platform 4.13	IBM Power (ppc64le) and IBM Z (s390x)	Red Hat build of OpenJDK 11 & 17	Yes	Yes
Red Hat Enterprise Linux 8	AMD64 and Intel 64 (x86_64)	Red Hat build of OpenJDK 11 & 17	Yes	Yes
Red Hat Enterprise Linux 8	AMD64 and Intel 64 (x86_64)	Eclipse Temurin OpenJDK 11 & 17	Yes	Yes

- 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 3.2. 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 Red Hat build of Quarkus support for Java 11

In Red Hat build of Quarkus 3.2, support for Java 11 is deprecated and is planned to be removed in a future release. Although Red Hat build of Quarkus 3.2 still supports Java 11 as the minimal version, start using Java 17 instead.

1.6. TECHNOLOGY PREVIEWS

This section lists features and extensions that are now available as a Technology Preview in Red Hat build of Quarkus 3.2.

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. Enhanced component testing

Red Hat build of Quarkus 3.2 introduces a JUnit extension named **QuarkusComponentTestExtension** as a Technology Preview feature.

This new extension aims to help ease testing of CDI components and mocking of their dependencies and is available in the **quarkus-junit5-component** dependency.

For more information, see the Testing components section of the Red Hat build of Quarkus "Testing your application" guide.

1.6.2. Hibernate Reactive upgraded to version 2

With this 3.2 release, Red Hat build of Quarkus depends on the Hibernate Reactive 2 extension instead of Hibernate Reactive 1. This change implies several changes in behavior and database schema expectations that are incompatible with earlier versions. Most of the changes are related to Hibernate Reactive 2 depending on Hibernate ORM 6.2 instead of Hibernate ORM 5.6.

1.6.3. quarkus-opentelemetry-exporter-otlp merged into quarkus-opentelemetry

The **quarkus-opentelemetry-exporter-otlp** extension is part of the **quarkus-opentelemetry** extension. This unified extension provides OpenTelemetry Protocol (OTLP) exporter functionality without additional setup, streamlining OTLP exporter usage.

1.6.4. Support for storing transaction logs in a database

With Red Hat build of Quarkus 3.2, for cloud environments where persistent storage is unavailable, such as when application containers cannot use persistent volumes, you can configure the transaction management to store transaction logs in a database by using a Java Database Connectivity (JDBC) datasource.

IMPORTANT

This configuration is only relevant for *Jakarta Transactions* transactions.

While there are several benefits to using a database to store transaction logs, you might notice a reduction in performance compared with using the file system to store the logs.

However, in cloud-native apps, it is important to assess transactions after careful evaluation. The **narayana-jta** extension, which manages these transactions, requires stable storage, a unique reusable node identifier, and a steady IP address to work correctly. While the JDBC object store provides stable storage, users must still plan how to meet the other two requirements.

To store transaction logs by using a JDBC datasource, configure the **quarkus.transacion-manager.object-store.<property>** properties, where **<property>** can be any of the following options:

- **type** (*string*): Configure this property to **jdbc** to enable usage of a Red Hat build of Quarkus JDBC datasource for storing transaction logs. The default value is **file-system**.
- **datasource** (*string*): Specify the name of the datasource for the transaction log storage. If no value is provided for the **datasource** property, Red Hat build of Quarkus uses the default datasource.
- **create-table** (*boolean*): When set to **true**, the transaction log table gets automatically created if it does not already exist. The default value is **false**.

- **drop-table** (*boolean*): When set to **true**, the tables are dropped on startup if they already exist. The default value is **false**.
- **table-prefix** (string): Specify the prefix for a related table name. The default value is **quarkus_**.

Also consider the following points:

- You can manually create the transaction log table during the initial setup by setting the **create-table** property to **true**.
- JDBC data sources and ActiveMQ Artemis allow the enlistment and automatic registration of **XAResourceRecovery** instances.
  However, be aware that the following points are not included in Red Hat build of Quarkus's support for storing transaction logs in a database:
  - JDBC datasources is part of the **quarkus-agroal** extension and requires that the following application property is set as shown: **quarkus.datasource.jdbc.transactions=XA**.
  - ActiveMQ Artemis (community client) is part of **quarkus-pooled-jms** extension and requires that the following application property is set as shown: **quarkus.pooled-jms.transaction=XA**.
    For more information, see CEQ-4878.
- To ensure data protection in case of application crashes or failures, enable the transaction crash recovery with the **quarkus.transaction-manager.enable-recovery=true** configuration.

NOTE

To work around the current known issue of Agroal having a different view on running transaction checks, set the datasource transaction type for the datasource responsible for writing the transaction logs to **disabled**:

```
quarkus.datasource.<TX_LOG>.jdbc.transactions=disabled
```

This example uses TX_LOG as the datasource name.

1.7. CHANGES THAT AFFECT COMPATIBILITY WITH EARLIER VERSIONS

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

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

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

1.7.1. Cloud

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

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

1.7.1.2. Improved logic for generating TLS-based container ports

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

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

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

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

1.7.1.3. Removal of some Kubernetes and OpenShift properties

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

Table 1.2. Removed properties and their new counterparts

Removed property	New property
quarkus.kubernetes.expose	**quarkus.kubernetes.ingress.expose**
quarkus.openshift.expose	**quarkus.openshift.route.expose**
quarkus.kubernetes.host	**quarkus.kubernetes.ingress.host**
quarkus.openshift.host	**quarkus.openshift.route.host**
quarkus.kubernetes.group	**quarkus.kubernetes.part-of**
quarkus.openshift.group	**quarkus.openshift.part-of**

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

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

1.7.2. Core

1.7.2.1. Upgrade to Jandex 3

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

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

Table 1.3. Old coordinates and their new counterparts

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

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

1.7.2.2. Migration path for users of Jandex API

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.7.2.5. Removal of the @AlternativePriority annotation

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

Example: Removed annotation

```
@AlternativePriority(1)
```

Example: Replacement annotations

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

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

1.7.2.6. Testing changes: Fixation of the Mockito subclass mockmaker

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

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

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

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

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

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

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

1.7.2.7. Update to the minimum supported Maven version

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

1.7.2.8. Removal of quarkus-bootstrap-maven-plugin

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

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

NOTE

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

1.7.2.9. Mutiny 2 moves to Java Flow

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

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

1.7.3. Data

1.7.3.1. Removal of Hibernate ORM with Panache methods

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

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

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

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

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

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

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

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

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

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

IMPORTANT

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

For more information, see the following resources:

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

1.7.3.4. Hibernate Search changes

Changes in the defaults for **projectable** and **sortable** on **GeoPoint** fields

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

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

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

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

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

Deprecated or renamed configuration properties

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

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

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

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

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

For more information, see the following resources:

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

1.7.3.6. Quartz jobs class name change

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

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

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

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

Update code that relies on these deprecated methods as follows:

Before

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

After

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

Before

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

After

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

Before

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

After

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

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

1.7.3.8. Renamed Narayana transaction manager property

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

1.7.4. Messaging

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

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

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

1.7.5. Native

1.7.5.1. Native compilation - Native executables and .so files

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

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

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

NOTE

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

1.7.5.2. Native Compilation - Work around missing CPU features

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

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

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

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

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

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

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

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

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

NOTE

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

1.7.5.3. Testing changes: Removal of some annotations

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

Table 1.4. Removed annotations and their new counterparts

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

The replacement annotations are functionally equivalent to the removed ones.

1.7.6. Observability

1.7.6.1. Deprecated OpenTracing driver is replaced by OpenTelemetry

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

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

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

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

1.7.6.2. Default metrics format in Micrometer now aligned with Prometheus

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

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

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

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

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

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

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

Table 1.5. Deprecated properties and their new counterparts

Deprecated properties	New properties
quarkus.opentelemetry.enabled	**quarkus.otel.enabled**
quarkus.opentelemetry.tracer.enabled	**quarkus.otel.traces.enabled**
quarkus.opentelemetry.propagators	**quarkus.otel.propagators**
quarkus.opentelemetry.tracer.suppress-non-application-uris	**quarkus.otel.traces.suppress-non-application-uris**
quarkus.opentelemetry.tracer.include-static-resources	**quarkus.otel.traces.include-static-resources**
quarkus.opentelemetry.tracer.sampler	**quarkus.otel.traces.sampler**

Deprecated properties	New properties
quarkus.opentelemetry.tracer.sampler.ratio	**quarkus.otel.traces.sampler.arg**
quarkus.opentelemetry.tracer.exporter.otlp.enabled	**quarkus.otel.exporter.otlp.enabled**
quarkus.opentelemetry.tracer.exporter.otlp.headers	**quarkus.otel.exporter.otlp.traces.headers**
quarkus.opentelemetry.tracer.exporter.otlp.endpoint	**quarkus.otel.exporter.otlp.traces.legacy-endpoint**

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

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

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

Table 1.6. Removed sampler property values and their new counterparts

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

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

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

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

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

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

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

OpenTelemetry upgrades

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

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

Table 1.7. Deprecated properties and their new counterparts

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

JDBC tracing activation

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

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

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

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

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

1.7.7. Security

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

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

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

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

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

WARNING

Use this setting with caution to maintain optimal system security.

1.7.7.2. OpenAPI CORS support change

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

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

1.7.7.3. Encryption of OIDC session cookie by default

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.7.8. Web

1.7.8.1. Changes to RESTEasy Reactive multipart

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

- Before this release, you could catch all file uploads regardless of the parameter name using the syntax: **@RestForm List<FileUpload> all**, but this was ambiguous and not intuitive. Now, this form only fetches parameters named **all**, just like for every other form element of other types, and you must use the following form to catch every parameter regardless of its name: **@RestForm(FileUpload.ALL) List<FileUpload> all**.

- Multipart form parameter support has been added to **@BeanParam**. The **@MultipartForm** annotation is now deprecated. Use **@BeanParam** instead of **@MultipartForm**.
- The **@BeanParam** is now optional and implicit for any non-annotated method parameter with fields annotated with any **@Rest*** or **@*Param** annotations.
- Multipart elements are no longer limited to being encapsulated inside **@MultipartForm**-annotated classes: they can be used as method endpoint parameters and endpoint class fields.
- Multipart elements now default to the **@PartType(MediaType.TEXT_PLAIN)** MIME type unless they are of type **FileUpload**, **Path**, **File**, **byte[]**, or **InputStream**.
- Multipart elements of the **MediaType.TEXT_PLAIN** MIME type are now deserialized using the regular **ParamConverter** infrastructure. Before this release, deserialization used **MessageBodyReader**.
- Multipart elements of the **FileUpload**, **Path**, **File**, **byte[]**, or **InputStream** types are special-cased and deserialized by the RESTEasy Reactive extension, not by the **MessageBodyReader** or **ParamConverter** classes.
- Multipart elements of other explicitly set MIME types still use the appropriate **MessageBodyReader** infrastructure.
- Multipart elements can now be wrapped in **List** to obtain all values of the part with the same name.
- Any client call that includes the **@RestForm** or **@FormParam** parameters defaults to the **MediaType.APPLICATION_FORM_URLENCODED** content type unless they are of the **File**, **Path**, **Buffer**, **Multi<Byte>**, or **byte[]** types, in which case it defaults to the **MediaType.MULTIPART_FORM_DATA** content type.
- Class **org.jboss.resteasy.reactive.server.core.multipart.MultipartFormDataOutput** has been moved to **org.jboss.resteasy.reactive.server.multipart.MultipartFormDataOutput**.
- Class **org.jboss.resteasy.reactive.server.core.multipart.PartItem** has been moved to **org.jboss.resteasy.reactive.server.multipart.PartItem**.
- Class **org.jboss.resteasy.reactive.server.core.multipart.FormData.FormValue** has been moved to **org.jboss.resteasy.reactive.server.multipart.FormValue**.
- The REST Client no longer uses the server-specific **MessageBodyReader** and **MessageBodyWriter** classes associated with Jackson. Before this release, the REST Client unintentionally used those classes. The result is that applications that use both **quarkus-resteasy-reactive-jackson** and **quarkus-rest-client-reactive** extensions must now include the **quarkus-rest-client-reactive-jackson** extension.

1.7.8.2. Enhanced JAXB extension control

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

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

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

```java
package org.acme.one;

import jakarta.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Model {
    private String name1;

    public String getName1() {
        return name1;
    }

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

package org.acme.two;

import jakarta.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Model {
    private String name2;

    public String getName2() {
        return name2;
    }

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

To activate this feature, add the following property:

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

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

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

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

Or you can exclude all the classes under a package:

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

1.8. BUG FIXES

Red Hat build of Quarkus 3.2 enhances stability and resolves critical bugs, ensuring optimal performance and security. To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.2.11.

1.8.1. Security fixes resolved in Red Hat build of Quarkus 3.2.11

- CVE-2024-1597 **org.postgresql/postgresql**: **pgjdbc**: PostgreSQL JDBC Driver vulnerability allows SQL injection with PreferQueryMode=SIMPLE
- CVE-2024-1979 **io.quarkus/quarkus-kubernetes-deployment**: Potential information leakage via annotations
- CVE-2024-1726 **io.quarkus.resteasy.reactive/resteasy-reactive**: Delayed security checks on certain inherited endpoints in RESTEasy Reactive could lead to denial of service
- CVE-2024-25710 **org.apache.commons/commons-compress**: Infinite loop denial of service with corrupted DUMP file
- CVE-2024-26308 **org.apache.commons/commons-compress**: OutOfMemoryError caused by unpacking a malformed Pack200 file
- CVE-2024-1300 **io.vertx:vertx-core**: Memory leak in TCP servers with TLS and SNI enabled
- CVE-2024-1023 **io.vertx:vertx-core**: Memory leak from Netty FastThreadLocal data structures usage in Vert.x

1.8.2. Security fixes resolved in Red Hat build of Quarkus 3.2.10

- CVE-2023-22102 **mysql/mysql-connector-java**: Connector/J unspecified vulnerability (CPU October 2023)
- CVE-2023-48795 **org.apache.sshd/sshd-core**: **ssh**: Prefix truncation attack on Binary Packet Protocol (BPP)
- CVE-2023-4043 **org.eclipse.parsson/parsson**: Denial of Service due to large number parsing

1.8.3. Security fixes resolved in Red Hat build of Quarkus 3.2.9.SP1

- CVE-2023-5675 **io.quarkus.resteasy.reactive/resteasy-reactive**: **quarkus**: Authorization flaw in Quarkus RestEasy Reactive and Classic when "quarkus.security.jaxrs.deny-unannotated-endpoints" or "quarkus.security.jaxrs.default-roles-allowed" properties are used
- CVE-2023-6267 **io.quarkus/quarkus-resteasy**: **quarkus**: JSON payload getting processed prior to security checks when REST resources are used with annotations

1.8.4. Security fixes resolved in Red Hat build of Quarkus 3.2.9

- CVE-2023-43642: **snappy-java**: Missing upper bound check on chunk length in snappy-java can lead to Denial of Service (DoS) impact
- CVE-2023-39410: **avro**: **apache-avro**: Apache Avro Java SDK: Memory when deserializing untrusted data in Avro Java SDK

1.8.5. Security fixes resolved in Red Hat build of Quarkus 3.2.6

- CVE-2023-33202: **bcpkix**: **bc-java**: Out of memory while parsing ASN.1 crafted data in **org.bouncycastle.openssl.PEMParser** class
- CVE-2023-4853: **quarkus-http**: quarkus: HTTP security policy bypass
- CVE-2023-44487: **netty-codec-http2**: HTTP/2: Multiple HTTP/2 enabled web servers are vulnerable to a DDoS attack (Rapid Reset Attack)

1.8.6. Other enhancements and bug fixes

- QUARKUS-3964 Fix tracing protocol configuration to only allow GRPC
- QUARKUS-3963 Handle generic types for ParamConverter in REST Client
- QUARKUS-3962 Never register server-specific providers in REST Client (fixed)
- QUARKUS-3960 Register methods of RESTeasy Reactive parameter containers for reflection
- QUARKUS-3959 Use an empty string in an SSE event when there is no data
- QUARKUS-3958 Update the Infinispan client intelligence section documentation
- QUARKUS-3956 Update the keycloak-admin-client extension to recognize the quarkus.tls.trust-all property
- QUARKUS-3955 Always run a JPA password action
- QUARKUS-3954 Reactive REST Client: check for ClientRequestFilter when skipping @Provider auto-discovery
- QUARKUS-3950 Fix various minor issues in the quarkus update command
- QUARKUS-3949 Fix Panache bytecode enhancement for @Embeddable records
- QUARKUS-3948 Save pathParamValues encoded and perform decoding when requested
- QUARKUS-3947 Fix != expression in @PreAuthorize check
- QUARKUS-3945 Support using commas to add extensions with CLI
- QUARKUS-3943 Fixes stork path param resolution in REST Client
- QUARKUS-3941 Do not expand config properties for Gradle Workers
- QUARKUS-3940 Verify duplicated context handling when caching a Uni
- QUARKUS-3939 Always set ssl and alpn for non-plain-text with Vert.x gRPC channel
- QUARKUS-3851 Upgrade to Hibernate ORM 6.2.18.Final

- QUARKUS-3841 Hibernate issue with @OneToMany mappedBy association(HHH-16593)
- QUARKUS-3791 Jandex indexing throws an NPE with the latest Oracle driver
- QUARKUS-3779 [GSS](3.2.z) RESTEASY-3380 - Source references exposed in RESTEasy error response
- QUARKUS-3757 Unfiltered traces from the management interface
- QUARKUS-3420 Duplicate artifacts brought in by extraneous io.quarkus in ER4
- QUARKUS-3273 Duplicated artifacts in Ghost
- QUARKUS-3598 Version alignment with Red Hat Build of Apache Camel for Red Hat build of Quarkus 3.2.0
- QUARKUS-3586 Automate step of creating depstobuild.txt
- QUARKUS-3476 quarkus-bom-deps-to-build.txt not delivered with 2.13.8.SP3.CR1 and 3.2.9.CR1
- QUARKUS-3761 Hibernate Reactive doesn’t work with Red Hat build of Quarkus 3.2.9.CR1 but works with upstream release
- QUARKUS-3759 Missing Sources for mvnpm/importmap in 3.2.9
- QUARKUS-3764 Red Hat build of Quarkus 3.2.9.CR1 contains 2 Red Hat build of Quarkus BOMs, one of them has 555 missing dependencies in Maven repo zip
- QUARKUS-3439 Red Hat build of Quarkus create app with gradle causing unresolved netty dependencies
- QUARKUS-1481 Platform source zips contain only quarkus source
- QUARKUS-3758 Duplicate Pom for io.github.crac:org-crac and Jboss Threads in 3.2.9
- QUARKUS-3377 support quarkus-keycloak-authorization again in 3.2.z
- QUARKUS-3597 Productize Red Hat build of Quarkus JOSDK extensions 6.3.3
- QUARKUS-3424 Increase in number of duplicate artifacts with no direct dependency lineage to platform boms/supported extensions
- QUARKUS-3582 Red Hat build of Quarkus 3.2: move start-stop metrics and tech empower jobs to JDK17 in performance labs
- QUARKUS-3570 Adding the JUL URL to the Logging guide update
- QUARKUS-3571 Make hibernate reactive status clear in docs
- QUARKUS-3546 Fix handling of HTTP/2 H2 empty frames in RestEasy Reactive
- QUARKUS-3564 Remove update guide from docs yml
- QUARKUS-3565 Enhancements to Configuration section of the Logging guide
- QUARKUS-3566 Applying the QE feedback for the Logging guide

- QUARKUS-3567 Doc link fixes & enhancements to Bearer token authentication tutorial
- QUARKUS-3572 Fix doc link Asciidoc change link to xref where applicable
- QUARKUS-3573 Config doc - Avoid processing methods if not @ConfigMapping
- QUARKUS-3662 Tiny grammar tweaks for the Authorization of web endpoints guide
- QUARKUS-3563 Fix title of upx.adoc
- QUARKUS-3569 Remove 'Security vulnerability detection' topic from downstream doc list
- QUARKUS-3568 Additional review and application of QE feedback to the Datasource guide
- QUARKUS-3339 Vert.x SQL client hangs when it inserts null or empty string into Oracle DB
- QUARKUS-3367 HTTP/1.1 upgrade to H2C cannot process fully request entity with a size greater than the initial window size
- QUARKUS-3669 Bump Keycloak version to 22.0.6
- QUARKUS-3670 Vert.x: fix NPE in ForwardedProxyHandler
- QUARKUS-3668 Fix dead link in infinispan-client-reference.adoc
- QUARKUS-3671 Fix quarkus update regression on extensions
- QUARKUS-3672 Take @ConstrainedTo into account for interceptors
- QUARKUS-3680 Let custom OIDC token propagation filters customize the exchange status
- QUARKUS-3679 Update Vert.x version to 4.4.6
- QUARKUS-3663 Tiny Vale tweaks for Datasource and Logging guide
- QUARKUS-3664 Duplicate Authorization Bearer Header Fix
- QUARKUS-3666 Fixing Db2 Driver typo
- QUARKUS-3675 Make the ZSTD Substitutions more robust
- QUARKUS-3677 Fix deployer detection in quarkus-maven-plugin
- QUARKUS-3676 Fix handling of HTTP/2 H2 empty frames in RestEasy Reactive
- QUARKUS-3665 More reliable test setup in integration-tests/hibernate-orm-tenancy/datasource
- QUARKUS-3674 QuarkusSecurityTestExtension after each call should not be made for tests without @TestSecurity
- QUARKUS-3673 Dev UI: Fix height in Rest Client
- QUARKUS-3667 Fix assertions in Hibernate ORM 5.6 compatibility tests
- QUARKUS-3678 ArC: fix PreDestroy callback support for decorators
- QUARKUS-3691 Prepare for ORM update

- QUARKUS-3689 Fix issue in Java migration in dev-mode

1.9. KNOWN ISSUES

Review the following known issues for insights into Red Hat build of Quarkus 3.2 limitations and workarounds.

1.9.1. Using CDI interceptors to resolve multitenant OIDC configuration fails due to security fix in version 3.2.9.SP1

The security fix implemented in Red Hat build of Quarkus version 3.2.9.SP1 to address CVE-2023-6267 introduced a breaking change.

This breaking change is relevant only when using multiple OIDC providers with RestEasy Classic and occurs if you use Context and Dependency Injection (CDI) interceptors to programmatically resolve OIDC tenant configuration identifiers.

Before this fix, CDI interceptors ran before authentication checks. After introducing the fix, authentication occurs before CDI interceptors are triggered. Therefore, using CDI interceptors to resolve multiple OIDC provider configuration identifiers no longer works. RestEasy Reactive applications are not affected.

Workaround: Use the **quarkus.oidc.TenantResolver** method to resolve the current OIDC configuration tenant ID.

For more information, see the Resolving tenant identifiers with annotations section of the Quarkus “Using OpenID Connect (OIDC) multitenancy” guide.

1.9.2. Podman 4.6 and later does not work with SELinux and Testcontainers library

The **Ryuk** container, which is essential to the **testcontainers** library used during dev mode, cannot be started when using Podman 4.6 or later. Specifically, these issues manifest when using **SELinux** and prevent the **Ryuk** container from starting successfully.

Here are the specific issues and corresponding workarounds:

- **Connection to Docker daemon socket fails** By default, an error occurs stating, **Permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock**."
    - Workaround: Update the **containers.conf** file to include **label=false**.
- **SELinux and containers configuration mismatch** If **SELinux** is enabled on the operating system but disabled in the **containers.conf** file, an **InternalServerErrorException** occurs.
    - Workaround: Run **sudo setenforce 0** to disable **SELinux**.
- **Unresolved Oracle Cloud Infrastructure (OCI) permission error** An error message appears stating, **OCI permission denied**. This issue has no workaround.

Given these issues, consider the following options:

- Refrain from using **Ryuk** containers until this known issue is resolved.
- Use an earlier version of Podman, such as version 4.5.x, that is compatible with Red Hat Enterprise Linux 8 and later.

For more information, see the following resources:

- QUARKUS-3451 - Podman 4.6 and newer does not work properly with SELinux and test-containers
- testcontainers/ryuk
- GitHub issue #20206: Ryuk container cannot be started on podman 4.6.2

1.9.3. Containers spawned by Testcontainers occasionally fail

Containers spawned by the **testcontainers** library for dev mode continuous testing occasionally fail with a **Broken pipeline** error.

Workaround: To work around the issue, restart dev mode. This issue does not affect production mode.

For more information, see QUARKUS-3448 - Broken pipe when creating containers with Podman.

1.9.4. HTTP/1.1 Upgrades to H2C fail under specific flow control conditions

When upgrading an HTTP/1.1 connection to H2C, the server does not account for inbound HTTP messages in the H2 flow controller. This results in unprocessed messages when the window size reaches zero.

A fix for this issue is planned for an upcoming release.

Workaround: No workaround is available at this time. Until this issue has been fixed, refrain from upgrading HTTP/1.1 connections to H2C if the message payload size exceeds the flow control window size.

For more information, see the following resources:

- Quarkus GitHub issue #35180 - Server fails receiving large data over http/2
- Vert.x Pull Request #4802 - HTTP/1.1 upgrade to H2C cannot process fully request entity with a size greater than the initial window size
- CEQ-7160 - CXF - Netty Http2Exception: Flow control window exceeded for stream: 0 when sending a ~64 KiB attachment
- QUARKUS-3367 - HTTP/1.1 upgrade to H2C cannot process fully request entity with a size greater than the initial window size.

1.9.5. Reactive Oracle datasource fails with specific Oracle JDBC driver versions

The Reactive Oracle datasource relies on Oracle's Java™ database connectivity (JDBC) driverReactive extensions. A bug exists in Oracle JDBC driver versions **23.2** and **21.11** that causes the failure of the application to receive any response under the following conditions:

- You use Reactive extensions to run an **UPDATE** or **INSERT** query that produces an error such as a constraint violation.
- You enable generated keys retrieval.

NOTE

Oracle might not support using the Oracle JDBC driver v21.10.0.0 with an Oracle 23 database.

Workarounds:

- Change the Oracle JDBC driver version in your **pom.xml** file or equivalent configuration to **com.oracle.database.jdbc:ojdbc11:21.10.0.0**.
- Avoid running queries that require generated key retrieval. For example, load sequence values before running **INSERT** queries.

For more information, see QUARKUS-3339 - Vertx SQL client hangs when it inserts a null or empty string into Oracle DB.

1.9.6. Community artifacts are used for native Vert.x dependencies on specific platforms

Applications that use the Vert.x extension on newly supported platforms, such as Linux on **aarch64** and Windows on **x86-64**, inadvertently download Quarkus community versions of **com.aayushatharva.brotli4j** artifacts rather than the ones built and provided by Red Hat. This issue has no functional impact.

A fix for this issue is planned for an upcoming release.

Workaround: No workaround is available at this time.

For more information, see QUARKUS-3314 - com.aayushatharva.brotli4j:native-linux-aarch64 and native-windows-x86_64 are not productized.

1.9.7. Red Hat build of Quarkus Kafka Streams are not supported on Windows due to a missing library

Kafka Streams fails to load RocksDB on Windows operating systems because the **librocksdbjni-win64.dll** native library is not in the Red Hat build of Quarkus.

Workaround: There is no workaround for running Quarkus Kafka Streams on Windows. Use non-Windows operating systems until a fix is available or it is confirmed that this extension will not be supported on Windows.

NOTE

Kafka Streams is a Technology Preview feature.

For more information, see QUARKUS-3434 - Ghost: Quarkus Kafka Streams not supported on Windows due to missing librocksdbjni-win64.dll

1.9.8. Community artifacts are used for some native dependencies on specific platforms

Red Hat build of Quarkus has native libraries for **x86_64** architectures for the following components.

- **io.netty.netty-transport-native-epoll**

- **io.netty.netty-transport-native-unix-common**
- **com.aayushatharva.brotli4j**

However, Red Hat build of Quarkus lacks native libraries for these components on the **ppc64le** and **s390x** architectures. Instead, it downloads the Quarkus community versions of the artifacts rather than the ones built and provided by Red Hat. This issue has no functional impact.

Workaround: No workaround is available at this time.

For more information, see the following resources:

- QUARKUS-3434 - Ghost: Quarkus Kafka Streams not supported on Windows due to missing librocksdbjni-win64.dll

1.9.9. 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

1.9.10. Build failure in the starter application generated by JBang with the Red Hat extension registry

Building the starter application generated by JBang with Red Hat extension registry might result in an unspecified error when **postBuild()** runs:

```
[jbang] [ERROR] Issue running postBuild()
dev.jbang.cli.ExitException: Issue running postBuild()
```

Red Hat build of Quarkus does not support this JBang scenario or development tooling.

Workaround: No workaround is available at this time.

For more information, see QUARKUS-3371 - Application created with jbang can not be built

1.10. ADVISORIES RELATED TO THIS RELEASE

Before you start using and deploying Red Hat build of Quarkus 3.2.11, review the advisories about enhancements, bug fixes, and CVE fixes for other technologies and services related to the release.

1.10.1. Red Hat build of Quarkus 3.2.11

- RHSA-2024:129589

1.10.2. Red Hat build of Quarkus 3.2.10

- RHSA-2024:0722

1.10.3. Red Hat build of Quarkus 3.2.9.SP1

- RHSA-2024:0495

1.10.4. Red Hat build of Quarkus 3.2.9

- RHEA-2023:7612

1.10.5. Red Hat build of Quarkus 3.2.6

- RHEA-2023:5416

1.11. ADDITIONAL RESOURCES

- Migrating applications to Red Hat build of Quarkus version 3.2 guide.
- Getting Started with Red Hat build of Quarkus

Revised on 2024-04-04 12:02:13 UTC