Red Hat build of Quarkus 3.20

Release Notes for Red Hat build of Quarkus 3.20

Last Updated: 2025-10-13

Red Hat build of Quarkus 3.20 Release Notes for Red Hat build of Quarkus 3.20

Legal Notice

Abstract

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

Table of Contents

PROVIDING FEEDBACK ON RED HAT BUILD OF QUARKUS DOCUMENTATION

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

Procedure

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

CHAPTER 1. RELEASE NOTES FOR RED HAT BUILD OF QUARKUS 3.20

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 3.20.

These include the following notable changes:

- Improved Jackson JSON processing with reflection-free deserialization
- OpenTelemetry logging support
- Quarkus REST clients updated to MicroProfile REST Client 4.0
- Switch to UBI 9 by default
- WebSockets Next API introduced

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 Jakarta REST.

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

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

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

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

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

Feature	Quarkus community version	Red Hat build of Quarkus version	Description
Access to the latest community features	Yes	No	With the Quarkus community version, you can access the latest feature developments. Red Hat does not release Red Hat build of Quarkus to correspond with every version that the community releases. The cadence of Red Hat build of Quarkus feature releases is approximately every six months.
Enterprise support from Red Hat	No	Yes	Red Hat provides enterprise support for Red Hat build of Quarkus only. To report issues about the Quarkus community version, see quarkusio/quarkus - Issues.
Access to long-term support	No	Yes	The lifecycle for a major release of Red Hat build of Quarkus is divided into two support phases; full support and maintenance support. For information about the product lifecycle, timelines, and support policies of Red Hat build of Quarkus, log in to the Red Hat Customer Portal and see the Product lifecycles and Red Hat build of Quarkus lifecycle and support policies Knowledge Base articles.
Common Vulnerabilities and Exposures (CVE) fixes and bug fixes backported to earlier releases	No	Yes	With Red Hat build of Quarkus, selected CVE fixes and bug fixes are regularly backported to supported streams. For more information about maintenance support, see Red Hat build of Quarkus lifecycle and support policies.
Tested and verified with Red Hat OpenShift Container Platform and Red Hat Enterprise Linux (RHEL)	No	Yes	Red Hat build of Quarkus is built, tested, and verified with Red Hat OpenShift Container Platform and RHEL. Red Hat provides both production and development support for supported configurations and tested integrations according to your subscription agreement. For more information, see Red Hat build of Quarkus supported configurations.
Built from source using secure build systems	No	Yes	In Red Hat build of Quarkus, the core platform and all supported extensions are provided by Red Hat using secure software delivery, which means that they are built from source, scanned for security issues, and with verified license usage.

Feature	Quarkus community version	Red Hat build of Quarkus version	Description
Access to support for JDK and Red Hat build of Quarkus Native builder distribution	No	Yes	Red Hat build of Quarkus supports certified OpenJDK builds and certified native executable builders. See admonition below. For more information, see Red Hat build of Quarkus supported configurations.

IMPORTANT

To build native Linux executables , Red Hat build of Quarkus supports using the Red Hat build of Quarkus Native Builder image (quarkus/mandrel-for-jdk-21-rhel8), which is based on GraalVM Mandrel.

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

1.3. NEW FEATURES, ENHANCEMENTS, AND TECHNICAL CHANGES

This section overviews the new features, enhancements, and technical changes introduced in Red Hat build of Quarkus 3.20.

1.3.1. Cloud

1.3.1.1. OpenShift and Kubernetes: Fabric8 Kubernetes Client upgraded to version 7.1

In Red Hat build of Quarkus 3.20, the following extensions upgrade the Fabric8 Kubernetes Client from version 6.13 to 7.1.

- **quarkus-openshift**
- **quarkus-openshift-client**
- **quarkus-kubernetes**
- **quarkus-kubernetes-client**

This upgrade brings several improvements to performance, compatibility, and testing.

Some key benefits of Fabric8 Kubernetes Client 7.1:

- Supports Kubernetes API v1.32 for better compatibility.
- Uses Vert.x instead of OkHttp for improved performance.
- Adds new tools for easier Custom Resource Definition (CRD) generation.

- Uses Vert.x-based **MockWebServer** for better testing.

This change introduces breaking changes. For details, see this release note in the "Changes that affect compatibility with earlier versions" section.

NOTE

Red Hat build of Quarkus 3.20 provides Developer Preview support for the **quarkus-kubernetes** extension, and does not currently support the **quarkus-kubernetes-client** extension. However, if you use these extensions, the described change might affect your migration.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.3.2. Core

1.3.2.1. Builder and runtime base images upgraded to UBI 9

Starting in Red Hat build of Quarkus 3.20, Red Hat builder and runtime base images use Red Hat Universal Base Image 9 (UBI 9) by default.

The following images are upgraded:

Table 1.1. Upgraded images

Images	UBI 8 version	UBI 9 version
Base images	• **registry.access.redhat.com/ubi8/ubi:8.10** • OpenJDK 17: **registry.access.redhat.com/ubi8/openjdk-17** • OpenJDK 21: **registry.access.redhat.com/ubi8/openjdk-21**	• **registry.access.redhat.com/ubi9/ubi:9.5** • OpenJDK 17: **registry.access.redhat.com/ubi9/openjdk-17** • OpenJDK 21: **registry.access.redhat.com/ubi9/openjdk-21**
Minimal images	• **registry.access.redhat.com/ubi8/ubi-minimal:8.10**	• **registry.access.redhat.com/ubi9-minimal:9.5**

Images	UBI 8 version	UBI 9 version
Runtimes images	• OpenJDK 17: **registry.access.redhat.com/ubi8/openjdk-17-runtime:1.21** • OpenJDK 21: **registry.access.redhat.com/ubi8/openjdk-21-runtime:1.21**	• OpenJDK 17: **registry.access.redhat.com/ubi9/openjdk-17-runtime:1.21** • OpenJDK 21: **registry.access.redhat.com/ubi9/openjdk-21-runtime:1.21**
Builder image (Mandrel)	• **registry.access.redhat.com/quarkus/mandrel-for-jdk-21-rhel8:23.1**	• No change. Red Hat uses a Mandrel builder image, which continues to be based on UBI 8. However, native executables produced on UBI8 can be run on UBI9. For more information, see the Note below.

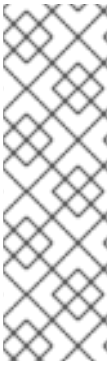
This upgrade introduces breaking changes. Builder and runtime base images upgraded to UBI 9 .

To upgrade to UBI 9 images manually, the following table outlines your options:

Table 1.2. Options for upgrading to UBI 9 manually

Images	Actions
Builder images	No action needed. Builder images are upgraded automatically in Red Hat build of Quarkus. However, you can also specify the builder image manually as follows: • **-Dquarkus.native.container-build=true** • **registry.access.redhat.com/quarkus/mandrel-for-jdk-21-rhel8:23.1**

Runtime images	Depending on the mode, apply the following settings:
	• JVM mode:
	  ◦ In your Dockerfile in the **src/main/docker/** directory, specify **registry.access.redhat.com/ubi9/openjdk-21-runtime:1.21** as the base image of your container.
	• Native mode:
	  ◦ To use UBI minimal, specify **registry.access.redhat.com/ubi9-minimal:9.5**.



NOTE

In Red Hat build of Quarkus native mode, if you use an in-container build to produce a native executable, you must use a Mandrel builder image. In Red Hat build of Quarkus 3.20, this Mandrel builder image is still based on UBI 8, however, native executables produced on UBI 8 can still be run on UBI 9.

When your native executable is created, you copy it into a container image, which in turn requires a runtime image as a base image. The preferred runtime image is the UBI 9-based runtime image: **registry.access.redhat.com/ubi9-minimal:9.5**.

If you encounter issues with UBI 9-based runtime images, you can manually switch back to UBI 8-based images. For details, see Builder and runtime base images upgraded to UBI 9 .

For more information, see the following resources:

- Red Hat Universal Base Image 9
- Quarkus Migration guide 3.19
- Creating a custom container image

1.3.2.2. Conditional extension dependencies in dev mode

With Red Hat build of Quarkus 3.20, extensions can now declare conditional dependencies that are activated only in development mode (dev mode) or when specific conditions are met, avoiding unnecessary dependencies in normal and test modes.

For more information, see the Dev mode-only extension dependencies section of the Quarkus "Conditional extension dependencies" guide.

1.3.2.3. Default locale configuration enhanced

Red Hat build of Quarkus 3.20 updates locale handling to align with Mandrel version 24.2 and later.

Applications now use a consistent default locale at run time, independent of the build system's locale. This change improves portability and predictability of locale-sensitive behavior across environments.

This change introduces breaking changes. For details, see this release note in the "Changes that affect compatibility with earlier versions" section.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.3.3. Data

1.3.3.1. Agroal: Database connection management in Dev UI enhanced

Starting in Red Hat build of Quarkus 3.20, the **quarkus-agroal** extension adds a dedicated **Agroal - Database connection pool** card and page for monitoring database connections to the Dev UI.

If your Quarkus application includes a JDBC driver extension that uses the **quarkus-agroal** extension, it is automatically included in your application.

In the Dev UI, you can favorite (star) the **Agroal - Database connection pool** card. You can also drag the **Database view** link from the card to the Dev UI menu. Clicking **Database view** opens the **Agroal - Database connection pool** page, where you can view all active datasources, including connection URLs.

This enhancement streamlines database monitoring and configuration within the Dev UI, improving debugging and database management.

1.3.3.2. Datasource: Validation of datasources during startup with ArC

In Red Hat build of Quarkus 3.20, the **quarkus-datasource** extension now validates datasource configurations during application startup by using the ArC-optimized dependency injection framework. This enhancement ensures that misconfigurations are detected early, preventing runtime failures and reducing the risk of traffic being routed to faulty instances.

Key changes in Red Hat build of Quarkus:

- If a datasource is injected into application beans, the application verifies its configuration at startup. Misconfigured datasources, such as those missing required properties, fail early instead of causing errors at first access.
- Datasources marked as inactive (**quarkus.datasource.active=false**) or missing required properties are not initialized, ensuring that only valid datasources are available to the application.
- Extensions such as Hibernate ORM now fail at startup if a required datasource is inactive, preventing applications from attempting to use unavailable resources.
- Datasources marked as inactive (**quarkus.datasource.active=false**) or missing required properties are excluded from health checks, ensuring that only relevant datasources contribute to diagnostics.

For more information, see the Activate or deactivate datasources section of the "Configure data sources" guide.

1.3.4. Messaging

1.3.4.1. Messaging Kafka extension introduced with support for Kafka transactions

Red Hat build of Quarkus 3.20 introduces the **quarkus-messaging-kafka** extension, which adds support for Apache Kafka transactions through the SmallRye Reactive Messaging Kafka connector. This feature enables atomic writes to multiple Kafka topics and partitions, ensuring that either all records within a transaction are committed or none are, enhancing data consistency in event-driven applications.

This functionality is available as a Technology Preview feature.

For more information, see the Kafka Transactions section of the Quarkus "Apache Kafka Reference Guide" guide.

1.3.5. Observability

1.3.5.1. Dev UI: Extension developers can add log tabs to the Dev UI footer

Starting in Red Hat build of Quarkus 3.20, extension developers can create a custom log tab in the Dev UI footer and stream log data to it. This process is similar to creating a card or page in the Dev UI.

Log tabs provide continuous data for application developers to monitor and troubleshoot their applications without switching tools. Unlike pages, which disconnect from the back end when navigated away from, log tabs remain persistently connected.

For more information, see the Add a footer tab section of the Quarkus "Dev UI for extension developers" guide.

1.3.5.2. Grafana OTel LGTM dashboards

Red Hat build of Quarkus 3.20 provides multiple pre-configured Grafana dashboards when using LGTM Dev Services. These dashboards visualize application metrics, traces, and logs. They leverage OpenTelemetry for seamless data collection and presentation. Some dashboards display Micrometer data in multiple ways, including OpenTelemetry output. Each dashboard is tuned for a specific application setup.

The available dashboards are:

- **Quarkus Micrometer OpenTelemetry**: Used with the Micrometer and OpenTelemetry extension.
- **Quarkus Micrometer OTLP Registry**: Used with the Micrometer OTLP registry extension.
- **Quarkus Micrometer Prometheus Registry**: Used with the Micrometer Prometheus registry extension.
- **Quarkus OpenTelemetry Logging**: Displays logs from the OpenTelemetry extension.

Some dashboard panels may take a few minutes to display accurate data due to calculations over a sliding time window.

1.3.5.3. OpenTelemetry: Exclude specific URIs from OpenTelemetry tracing

Since this update, you can now exclude specific URIs from OpenTelemetry tracing using the **quarkus.otel.traces.suppress-application-uris** configuration property. This allows you to define a comma-separated list of URI patterns that the tracer should ignore.

For example, setting **quarkus.otel.traces.suppress-application-uris=trace,ping,people*** will turn off tracing for the **/trace** and **/ping** URIs, as well as any URI starting with **/people**, such as **/people/1** or **/people/1/cars**.

If you use a custom **quarkus.http.root-path**, ensure you include it in the configuration.

1.3.5.4. OpenTelemetry: Immediate export of spans and logs with SimpleSpanProcessor

In Red Hat build of Quarkus 3.20, the **quarkus-opentelemetry** extension adds support for the **SimpleSpanProcessor**, which immediately exports spans and logs as they finish. This behavior is particularly useful in serverless environments and short-lived applications.

To enable this feature, set the following configuration property to **true**:

```
quarkus.otel.simple=true
```

This configuration replaces the default **BatchSpanProcessor**, ensuring spans and logs are sent immediately instead of being buffered. This change is critical for Lambda functions and other short-lived processes because it ensures that telemetry data is exported before the application shuts down,

preventing data loss caused by batching delays.

1.3.5.5. OpenTelemetry: Logging support introduced

In Red Hat build of Quarkus 3.20, the **quarkus-opentelemetry** extension now supports OpenTelemetry (OTel) logging. You can use this capability to provide distributed logging for interactive web applications.

To enable this feature, set the following configuration property to **true**:

```
quarkus.otel.logs.enabled=true
```

For more information, see the Quarkus Using OpenTelemetry Logging guide.

1.3.5.6. OpenTelemetry: Metrics integration aligned with MicroProfile Telemetry 2.0

In Red Hat build of Quarkus 3.20, the **quarkus-opentelemetry** extension offers automatic instrumentation for JVM and HTTP server request metrics in alignment with the MicroProfile Telemetry 2.0 specification, provided that OpenTelemetry metrics are enabled.

OpenTelemetry metrics are disabled by default. To collect JVM and HTTP server request metrics, you must enable them by setting **quarkus.otel.metrics.enabled=true**.

The corresponding configuration properties are enabled by default and can be disabled by setting **quarkus.otel.instrument.jvm-metrics=false** or **quarkus.otel.instrument.http-server-metrics=false** in your configuration.

The Fault Tolerance with OpenTelemetry Metrics integration feature remains currently unsupported.

NOTE

If you also use the Micrometer extension, you should disable OpenTelemetry instrumentation to prevent potential conflicts.

Additionally, if you configure the **logging** exporter by setting **quarkus.otel.metrics.exporter=logging**, you might see OpenTelemetry internal tracing-related metrics in the logs. The **logging** exporter is intended for debugging purposes only and is not recommended for production environments.

1.3.6. Security

1.3.6.1. Authentication: Support for combining authentication mechanisms

Red Hat build of Quarkus 3.20 adds support for using multiple authentication mechanisms within a single request. You can now combine mechanisms such as mutual TLS (mTLS) and OpenID Connect (OIDC) bearer token authentication in the same authentication flow.

By default, authentication completes when the first **SecurityIdentity** is produced by one of the registered authentication mechanisms. To require all mechanisms to validate credentials, enable inclusive authentication by setting a configuration property.

This capability is beneficial when combining mutual TLS and bearer token authentication, such as in scenarios that require access token binding to a client certificate.

For more information and examples, see the Combining authentication mechanisms section of the "Security architecture" guide.

1.3.6.2. Authorization: Create custom security annotations by using @PermissionsAllowed

Red Hat build of Quarkus 3.20 adds support for using **@PermissionsAllowed** within meta-annotations so that you can create custom security annotations for more streamlined access control management.

This enhancement simplifies permission-based configuration, reduces repetitive security code, and improves maintainability. You can now define reusable security annotations that encapsulate **@PermissionsAllowed**, making permission handling more efficient and consistent.

For more information and examples, see the Create permission meta-annotations section of the "Authorization of web endpoints" guide.

1.3.6.3. Authorization: Define declarative permission checkers by using CDI bean methods

Starting in Red Hat build of Quarkus 3.20, you can use the **@PermissionChecker** annotation to define declarative permission checkers in CDI bean methods. This enhancement enables flexible and reusable authorization logic by allowing permission checks to be implemented separately from resource classes, reducing code duplication and improving maintainability.

These permission checkers work alongside **@PermissionsAllowed** by matching permission names based on string equality. They support general-purpose authorization checks by evaluating the **SecurityIdentity**, which represents the authenticated user, and the incoming HTTP request parameters. This approach allows for fine-grained access control. Checkers must be defined in normal-scoped or singleton CDI beans and return either a **boolean** or **Uni<Boolean>**, supporting both synchronous and reactive authorization checks.

This feature simplifies permission management by centralizing authorization logic, making security policies easier to enforce across multiple endpoints. Developers can create generalized permission checkers that validate multiple actions dynamically.

For more information, see the Create permission checkers section of the "Authorization of web endpoints" guide.

1.3.6.4. OIDC: Authenticate OIDC provider clients by using JWT bearer tokens from the filesystem

Starting in Red Hat build of Quarkus 3.20, with **quarkus-oidc**, you can configure OIDC to authenticate an OIDC provider client by loading a JWT bearer token from a filesystem.

This approach enables secure and automated authentication. It is especially useful in containerized environments such as OpenShift, where a service account token can be mounted as a file. Red Hat build of Quarkus automatically loads the token from the file and reloads it when the token expires.

For configuration details and usage examples, see the following resources:

- The JWT Bearer section of the "OpenID Connect (OIDC) client and token propagation" guide.
- The "Example: How JWT Bearer token can be used to authenticate client" in the OIDC provider client authentication section of the "OpenID Connect (OIDC) authentication" guide.

1.3.6.5. OIDC: Filter OIDC responses for custom processing

In Red Hat build of Quarkus 3.20, the following Quarkus extensions introduce OpenID Connect (OIDC) response filters.

- OpenID Connect (**quarkus-oidc**)
- OpenID Connect Client (**quarkus-oidc-client**)

Implement **OidcResponseFilter** to inspect response status, headers, and body, for logging or other actions.

By default, OIDC response filters apply globally. To target specific endpoints, such as the token endpoint, use the **@OidcEndpoint** annotation.

For more information, see the OIDC response filters section of the "OpenID Connect (OIDC) authentication" guide.

1.3.6.6. OIDC: mTLS-bound access tokens

Mutual TLS (mTLS) token binding enhances security by ensuring that access tokens are cryptographically bound to the client's mTLS authentication certificate.

This feature, based on RFC 8705, minimizes the risk of accepting stolen bearer access tokens by verifying that a thumbprint of the certificate used for authentication matches a certificate thumbprint contained in the token.

Starting in Red Hat build of Quarkus 3.20, the Quarkus OIDC extension, **quarkus-oidc**, now supports certificate-bound access tokens for endpoints secured with both OpenID Connect (OIDC) and mTLS.

To enable this feature, set the following configuration property:

```
quarkus.oidc.token.binding.certificate=true
```

For more information, see the Mutual TLS token binding section of the "OpenID Connect (OIDC) authentication" guide.

1.3.6.7. OIDC: OidcProviderClient injection and token revocation

Red Hat build of Quarkus 3.20 introduces support for injecting **OidcProviderClient**, allowing applications to interact with the OpenID Connect (OIDC) provider's UserInfo, token introspection, and revocation endpoints.

It enables developers to access UserInfo, token introspection, and revocation endpoints programmatically when needed. For example, an access token can be revoked after a user logs out.

For more information, see the Token revocation section of the OpenID Connect (OIDC) authentication guide.

1.3.6.8. OIDC: Programmatically creating OIDC tenants

Starting in Red Hat build of Quarkus 3.20, with **quarkus-oidc** you can programmatically define OpenID Connect (OIDC) tenants by using the **OidcTenantConfig** builder instead of configuring them in the **application.properties** file.

This approach provides greater flexibility than defining OIDC tenants in the configuration.

This approach is particularly useful for features that require static tenants, such as issuer-based and path-based tenant resolvers.

For more information, see the following resources in the "OpenID Connect (OIDC) authentication" guide:

- Programmatic OIDC start-up under "OIDC Bearer token authentication"
- Programmatic OIDC start-up under "OIDC authorization code flow mechanism for protecting web applications"
- Programmatic OIDC start-up for multiple tenants

1.3.6.9. OIDC Client: Dev Services for Keycloak no longer requires the OIDC extension

In Red Hat build of Quarkus 3.20, Dev Services for Keycloak can now start even when the **quarkus-oidc** extension is not present. This change enables you to test the **quarkus-oidc-client** extension in isolation, without requiring the full OIDC extension.

Additionally, Red Hat build of Quarkus now automatically configures key OIDC Client properties when Dev Services for Keycloak is enabled, including:

- **quarkus.oidc-client.auth-server-url**
- **quarkus.oidc-client.client-id**
- **quarkus.oidc-client.credentials.secret**

Previously, you had to set these properties manually.

These enhancements are backward compatible. Existing workflows and manual configurations continue to work as expected.

1.3.6.10. TLS Registry: Policy configuration for expired or not-yet-valid certificates

Red Hat build of Quarkus 3.20 now supports configuring policies for handling expired or not-yet-valid certificates during TLS handshakes across the certificate chain.

Previously, the trust store allowed such certificates without warnings, adhering to RFC 3280 and related specifications.

With this update, users can control the behavior by selecting one of the following options:

- IGNORE: Retains the previous behavior, allowing expired or not-yet-valid certificates without warnings.
- WARN: Logs a warning when such certificates are detected (new default).
- REJECT: Rejects the handshake if such certificates are present.

1.3.6.11. OIDC: Demonstrating Proof of Possession supported

In Red Hat build of Quarkus 3.20, the **quarkus-oidc** extension now supports Demonstrating Proof of Possession (DPoP) for access tokens.

This functionality is available as a Technology Preview feature.

For more information, see the following resources:

- The Demonstrating Proof of Possession (DPoP) section of the Red Hat build of Quarkus "OpenID Connect (OIDC) client and token propagation" guide.
- The RFC 9449: OAuth 2.0 Demonstrating Proof of Possession (DPoP) standard

1.3.6.12. TLS Registry: Encrypted PKCS#8 private keys in PEM keystores introduced

In Red Hat build of Quarkus 3.20, the **quarkus-tls-registry** extension introduces support for encrypted PKCS#8 private keys in PEM keystores.

This functionality is available as a Technology Preview feature.

It enables developers to secure private keys with AES-128-CBC encryption, improving application security.

To use this feature, set the **quarkus.tls.key-store.pem.password** property with the keystore password.

For more information, see the Management of security keys and certificates with the TLS Registry guide.

1.3.7. Tooling

1.3.7.1. Configure JVM options in extension metadata for applications running in Dev mode

With Red Hat build of Quarkus 3.20, extension authors can define JVM options for applications running in development mode. They can specify these options, such as **--enable-preview** or **--add-opens**, directly in the extension metadata, eliminating the need for them to be configured by application developers.

Extensions automatically apply these JVM options when running in development mode, ensuring a consistent environment for all application developers.

Application developers can turn off all extension-provided JVM options or selectively turn off options for specific extensions with the Quarkus Maven plugin. For more information, see the Extension provided Dev mode Java options section of the Quarkus "Quarkus and Maven" guide.

1.3.8. Web

1.3.8.1. Dev UI: HTTP access logs added to Dev UI

Starting in Red Hat build of Quarkus 3.20, the Dev UI footer includes an **HTTP** tab, which you can use to monitor incoming HTTP access log messages without switching tools or contexts.

For example, if you interact with your application's HTTP endpoints by using the **SmallRye OpenAPI - Swagger UI** page in the Dev UI, you can see the corresponding log messages in the **HTTP** tab in the Dev UI footer.

For more information, see the Configuring HTTP access logs section of the Quarkus "HTTP reference" guide.

1.3.8.2. REST: Java records as REST parameters

Red Hat build of Quarkus 3.20 introduces support for Java records as REST endpoint parameter containers, enabling concise and immutable data structures. Records can encapsulate queries, cookies, headers, forms, or multipart parameters, simplifying data handling in RESTful services.

Java records already work for JSON serialization and deserialization of REST bodies. However, **@BeanParam** does not yet support them because of the limitations of the Jakarta REST specification.

1.3.8.3. REST and RESTEasy: @AuthorizationPolicy annotation introduced

In Red Hat build of Quarkus 3.20, the **quarkus-rest** and **quarkus-resteasy** extensions add support for the new **@AuthorizationPolicy** annotation. You can use this annotation to bind a named **HttpSecurityPolicy** directly to a Jakarta REST endpoint. This feature provides an alternative to path-matching rules and enables more flexible authorization checks without modifying **SecurityIdentity** roles.

For more information, see the Custom HttpSecurityPolicy section of the "Authorization of web endpoints" guide.

1.3.8.4. REST Client: Dynamic per-invocation URLs with @Url annotation

In Red Hat build of Quarkus 3.20, the **quarkus-rest-client** extension adds support for the **@Url** annotation. You can now annotate a parameter in a REST client method to provide a URL at run time dynamically.

Typically, a REST client requires a base URL that is configured globally. However, some cases require setting the URL on a per-invocation basis. The **@Url** annotation supports this use case by enabling dynamic URL resolution for individual calls.

For more information, see the Dynamic base URLs section of the Quarkus "Using the REST Client" guide.

1.3.8.5. REST Client: MicroProfile REST Client updated to version 4.0

In Red Hat build of Quarkus 3.20, the **quarkus-rest-client** extension has been updated to MicroProfile REST Client 4.0. This update enhances how applications interact with RESTful services by providing a standardized, declarative API for creating and managing REST clients. It improves flexibility, performance, and security, while simplifying the integration of external REST services.

1.3.8.6. REST Jackson: Improved JSON processing with reflection-free deserialization

In Red Hat build of Quarkus 3.20, the **quarkus-rest-jackson** extension enhances its JSON processing capabilities by introducing reflection-free deserialization.

This enhancement complements reflection-free serialization, which was implemented in an earlier release.

By default, reflection-free JSON processing is disabled. To enable it, set the following configuration property to **true**:

```
quarkus.rest.jackson.optimization.enable-reflection-free-serializers=true
```

By eliminating reliance on reflection during deserialization, applications can achieve better performance and reduced memory consumption, particularly for native applications where reflection can introduce overhead. If you enable this feature, run tests to assess its effect on your applications.

For more information, see the JSON serialisation section of the Quarkus "Writing REST services with Quarkus REST" guide.

1.3.8.7. WebSockets Next extension introduced

Red Hat build of Quarkus 3.20 introduces support for the **quarkus-websockets-next** extension, which provides a modern, declarative API for defining WebSocket server and client endpoints.

This extension enables efficient, scalable real-time communication and integrates with Red Hat build of Quarkus's reactive architecture and networking layer.

Unlike the deprecated **quarkus-websockets** extension, WebSockets Next does not implement the Jakarta WebSocket specification.

Key capabilities introduced by WebSockets Next:

Improved security integration

WebSockets Next supports seamless integration with Quarkus Security and identity providers. Each WebSocket connection is associated with a **SecurityIdentity**, enabling early authorization during the HTTP upgrade process. Developers can secure endpoints using standard security annotations or permission checkers. For clients that support custom headers, bearer token authentication is fully supported. JavaScript clients, which cannot set custom authorization headers, can pass the token using the **Sec-WebSocket-Protocol** header as a workaround. Additionally, Red Hat build of Quarkus automatically closes WebSocket connections when an OIDC token expires.

For details, see the Security section of the "WebSockets Next reference" guide.

Structured message handling

The extension supports built-in JSON serialization and deserialization. Common types such as **String**, **JsonObject**, **JsonArray**, **Buffer**, and **byte[]** are sent without conversion. You can provide a custom codec to customize how messages are serialized or deserialized.

For more information, see the Serialization and deserialization section.

Observability integration

WebSockets Next integrates with both OpenTelemetry and Micrometer to enhance observability. If the OpenTelemetry extension is present, it automatically captures traces for opened and closed WebSocket connections and links them to the initial handshake span. When Micrometer support is enabled, WebSockets Next collects metrics for WebSocket activity.

For configuration details, see the Telemetry section of the WebSockets Next reference guide.

For more information, see the following resources:

- Getting started with WebSockets Next
- WebSockets Next reference 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 3.20 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 release or a minor release:

- A major release introduces new features and a support life cycle (full and maintenance) for a minimum of 3 years.
- A minor release introduces new features or changes that do not affect compatibility with earlier releases (breaking changes). The latest minor version is fully supported, and the previous version is considered in maintenance for six months, starting when a new minor version is released.

For more information, see Red Hat Life Cycle and Support Policies .

Red Hat build of Quarkus release version numbers are directly aligned with the Long-Term Support (LTS) versions of the Quarkus community project. For more information, see the Long-Term Support (LTS) for Quarkus blog post.

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

IMPORTANT

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

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

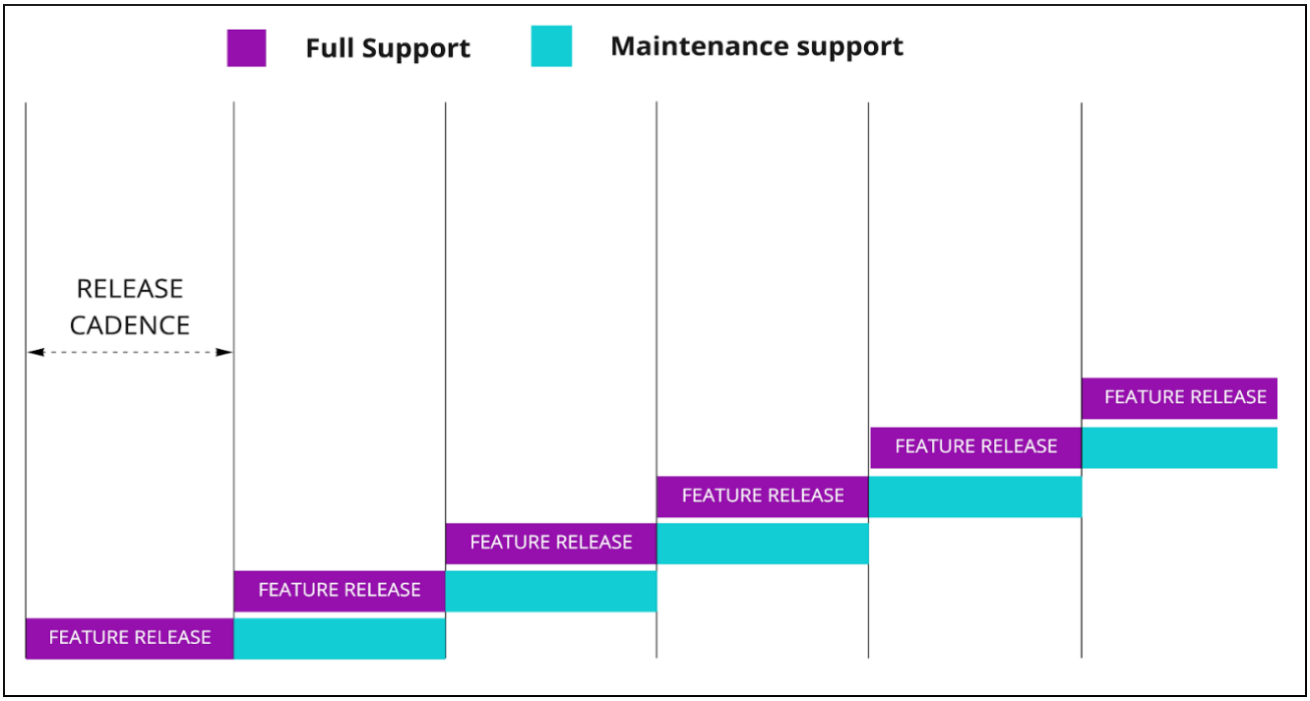


Figure 1. Feature release cadence and support lifecycle of Red Hat build of Quarkus

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

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

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

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

1.4.2. Tested and verified environments

Red Hat build of Quarkus 3.20 is available on the following versions of Red Hat OpenShift Container Platform: 4.19, 4.12, and Red Hat Enterprise Linux 8.10.

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 3.20. 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. Legacy configuration classes deprecated

In Red Hat build of Quarkus 3.20, legacy configuration classes based on **@ConfigRoot** and **@ConfigItem** have been deprecated. This deprecation prepares the migration to the **@ConfigMapping** framework, which provides a more consistent and type-safe configuration model.

The compatibility layer for existing configuration classes remains available in this release but is planned for removal in a future version.

Developers are encouraged to update their code to use **@ConfigMapping** interfaces to ensure long-term compatibility.

For more information, see the JIRA issue QDOCS-1150.

1.5.2. quarkus.http.cors property deprecated

In Red Hat build of Quarkus 3.20, the **quarkus.http.cors** configuration property has been deprecated. To enable Cross-Origin Resource Sharing (CORS), use the **quarkus.http.cors.enabled** property instead.

- Update your configuration as follows:

```
quarkus.http.cors.enabled=true
```

This change was introduced to improve compatibility with YAML configuration, where the previous structure required using special syntax such as ~ to assign a value to the root key, leading to confusing or error-prone setups.

For more information, see the Cross-Origin Resource Sharing (CORS) guide.

1.5.3. quarkus.log.*.json property deprecated

In Red Hat build of Quarkus 3.20, the **quarkus.log.*.json** configuration property has been deprecated. To enable JSON formatting for logging, use the **quarkus.log.*.json.enabled** property instead.

- Update your configuration as follows:

```
quarkus.log.console.json.enabled=true
```

This change was introduced to improve compatibility with YAML configuration, where the previous structure required using special syntax such as ~ to assign a value to the root key, leading to confusing or error-prone setups.

For more information, see the Migration Guide 3.19.

1.5.4. SmallRye Fault Tolerance version 6.7.0 deprecates first-generation programmatic API

In Red Hat build of Quarkus 3.20, the **quarkus-smallrye-fault-tolerance** extension includes SmallRye Fault Tolerance 6.7.0, which introduces no breaking changes but includes the following updates:

- The first version of the programmatic API (**FaultTolerance**, **@ApplyFaultTolerance**) is deprecated and planned for removal in SmallRye Fault Tolerance 7.0. The second version (**Guard**, **TypedGuard**, **@ApplyGuard**) serves as a replacement, but there are notable differences.
- Specification-defined configuration properties remain available, but Red Hat build of Quarkus now provides native configuration properties you can use instead.

For more information, see the SmallRye Fault Tolerance 6.7.0 release announcement, which includes links to the migration guides for the programmatic API and details about the new configuration properties. The Quarkus SmallRye Fault Tolerance guide provides a complete reference for these configuration properties.

1.5.5. WebSockets and WebSockets Client extensions deprecated

Red Hat build of Quarkus 3.20 deprecates the **quarkus-websockets** and **quarkus-websockets-client** extensions, which implement the Jakarta WebSocket specification.

Red Hat build of Quarkus plans to stop supporting these extensions in a future release.

To ensure compatibility with upcoming versions, migrate to the Quarkus WebSockets Next extension,**quarkus-websockets-next**, which offers a modern, more efficient WebSocket API.

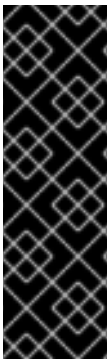
For more information, see the following Quarkus resources:

- Getting started with WebSockets Next
- WebSockets Next reference guide
- Quarkus WebSockets Next extension section of the "Release Notes for Red Hat build of Quarkus 3.20" guide

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.6. TECHNOLOGY PREVIEWS

This section lists features and extensions that are now available as a Technology Preview in Red Hat build of Quarkus 3.20.



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. New Technology Preview features

Red Hat build of Quarkus 3.20 introduces the following Technology Preview features:

- Messaging Kafka extension introduced with support for Kafka transactions
- OIDC: Demonstrating Proof of Possession supported
- TLS Registry: Encrypted PKCS#8 private keys in PEM keystores introduced

1.6.2. Continuing Technology Preview features

Red Hat build of Quarkus 3.20 continues to provide the following Technology Preview features that were introduced in Red Hat build of Quarkus 3.15:

- Cert-manager support and periodic reload of TLS certificates
- Generate reflection-free Jackson serializers
- Infinispan cache extension introduced
- JDK Flight Recorder extension introduced
- OpenTelemetry metrics data creation and reporting
- MongoDB tracing enhanced

1.7. CHANGES THAT AFFECT COMPATIBILITY WITH EARLIER VERSIONS

This section describes changes in Red Hat build of Quarkus 3.20 that affect the compatibility of applications built with earlier product versions.

Review these breaking changes and take the necessary steps to ensure that your applications continue functioning after updating them to Red Hat build of Quarkus 3.20.

To update your application projects to Red Hat build of Quarkus 3.20, see the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.1. Cloud

1.7.1.1. OpenShift and Kubernetes: Fabric8 Kubernetes Client upgraded to version 7.1

In Red Hat build of Quarkus 3.20, the following extensions upgrade the Fabric8 Kubernetes Client from version 6.13 to 7.1.

- **quarkus-openshift**
- **quarkus-openshift-client**
- **quarkus-kubernetes**
- **quarkus-kubernetes-client**

Breaking changes

The upgraded Fabric8 Kubernetes Client includes breaking changes affecting the **quarkus-kubernetes-client** and **quarkus-openshift-client** extensions.

Although the **quarkus-openshift** and **quarkus-kubernetes** extensions use Fabric8, their functionality remains unchanged, so you do not need to make any updates for them.

The **io.quarkus:quarkus-test-openshift-client** module has been removed as part of this upgrade. If your tests use this module, migrate to **io.quarkus:quarkus-test-kubernetes-client**, which offers equivalent functionality. For more information, see the OpenShift client section of the Quarkus "Kubernetes Client" guide.

The upgraded Fabric8 Kubernetes Client reorganized some model types and classes, moving some to different modules. To find their new locations, refer to the official Fabric8 Kubernetes Client: Migration from 6.x to 7.x guide.

Review your application for any affected dependencies, configurations, or API usages, and update them as needed. Then, thoroughly test your application to ensure full compatibility with Fabric8 7.1.

NOTE

Red Hat build of Quarkus 3.20 provides Developer Preview support for the **quarkus-kubernetes** extension, and does not currently support the **quarkus-kubernetes-client** extension. However, if you use these extensions, the described change might affect your migration.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.2. Compatibility

1.7.2.1. Removal of Reactive rename compatibility layer

In Red Hat build of Quarkus 3.15, many extensions and configuration properties were renamed as part of the rebranding of RESTEasy Reactive to Quarkus REST and Reactive Messaging to Quarkus Messaging. To support this transition, artifact relocations and configuration fallbacks were introduced.

In Red Hat build of Quarkus 3.20, these artifact relocations and configuration fallbacks have been removed. You must now use the new artifact names and configuration properties.

For more additional background information, see the RESTEasy Reactive extensions renamed to Quarkus REST section of the "Release Notes for Red Hat build of Quarkus 3.15" guide.

1.7.3. Core

1.7.3.1. Default locale configuration enhanced

Red Hat build of Quarkus 3.20 updates locale handling to align with Mandrel version 24.2 and later.

Breaking changes

In earlier releases, applications inherited the default locale from the build system.

To ensure consistent locale behavior across all applications, Red Hat build of Quarkus applies a standardized default locale strategy. Use the following table to determine how your application's behavior changes based on your locale configuration.

Application properties	Locales included in native executable	Default locale at run time
Neither **quarkus.locales** nor **quarkus.default-locale** is set	**en_US**	**en_US**
Only **quarkus.locales** is set	Locales listed in **quarkus.locales** and **en_US**	**en_US**
Only **quarkus.default-locale** is set	**en_US**	Value of **quarkus.default-locale**
Both **quarkus.locales** and **quarkus.default-locale** are set	Locales listed in **quarkus.locales** and **en_US**	Value of **quarkus.default-locale**

The **en_US** locale is always embedded in the native executable, regardless of the **quarkus.locales** configuration.

NOTE

If you set **quarkus.default-locale**, Red Hat build of Quarkus sets the **user.language** and **user.country** system properties at run time. For JDK 24 and later with GraalVM or Mandrel, you can also override these properties directly.

Review your application's locale settings in the configuration properties and, if necessary, update the settings to avoid unexpected changes in behavior.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.3.2. SmallRye Fault Tolerance: Fallback and BeforeRetry methods now defined at build time

In Red Hat build of Quarkus 3.20, the configuration properties for **@Fallback.fallbackMethod()** and **@BeforeRetry.methodName()** are now resolved at build time and cannot be changed at runtime. The affected configuration properties are:

- **<class_name>/<method_name>/Fallback/fallbackMethod**
- **<class_name>/Fallback/fallbackMethod**
- **Fallback/fallbackMethod**
- **<class_name>/<method_name>/BeforeRetry/methodName**
- **<class_name>/BeforeRetry/methodName**
- **BeforeRetry/methodName**

This change ensures proper reflection configuration and compatibility with native image compilation.

1.7.3.3. SmallRye Fault Tolerance version 6.7.0 deprecates first-generation programmatic API

In Red Hat build of Quarkus 3.20, the **quarkus-smallrye-fault-tolerance** extension includes SmallRye Fault Tolerance 6.7.0, which introduces no breaking changes but includes the following updates:

- The first version of the programmatic API (**FaultTolerance**, **@ApplyFaultTolerance**) is deprecated and planned for removal in SmallRye Fault Tolerance 7.0. The second version (**Guard**, **TypedGuard**, **@ApplyGuard**) serves as a replacement, but there are notable differences.
- Specification-defined configuration properties remain available, but Red Hat build of Quarkus now provides native configuration properties you can use instead.

For more information, see the SmallRye Fault Tolerance 6.7.0 release announcement, which includes links to the migration guides for the programmatic API and details about the new configuration properties. The Quarkus SmallRye Fault Tolerance guide provides a complete reference for these configuration properties.

1.7.3.4. Scheduler methods now require a started scheduler

Starting in Red Hat build of Quarkus 3.20, the behavior of methods in **io.quarkus.scheduler.Scheduler** has changed.

When the scheduler is not started, almost all methods now throw an **UnsupportedOperationException**.

To verify whether the scheduler is running, you can use a new method, **Scheduler#isStarted()**.

This change affects scheduler instances coming from both the **quarkus-scheduler** and **quarkus-quartz** extensions.

1.7.3.5. Management interface now listens on localhost in development and test modes

Starting in Red Hat build of Quarkus 3.20, when you use development and test modes, the management interface listens on the localhost interface by default instead of on **0.0.0.0**. This change aligns with the behavior of the main interface.

On Windows with Windows Subsystem for Linux (WSL), both the management and main interfaces continue to listen on **0.0.0.0**.

For more information, see the Quarkus Management interface reference guide.

1.7.3.6. Migration to @ConfigMapping and deprecation of configuration classes

In Red Hat build of Quarkus 3.20, configuration has migrated to the **@ConfigMapping** framework. This change deprecates the legacy configuration classes in favor of a unified configuration model based on interfaces and method signatures.

A compatibility layer remains in place for certain commonly used configuration classes, but it is planned for removal in a future release.

1.7.3.7. Builder and runtime base images upgraded to UBI 9

Starting in Red Hat build of Quarkus 3.20, Red Hat builder and runtime base images are upgraded to use Red Hat Universal Base Image 9 (UBI 9).

When upgrading from UBI 8 to UBI 9, be aware that changes in system dependencies, native image builds, or package and GNU C Library (glibc) version updates might affect your application's behavior or runtime environment. Review and update your configurations as needed.

If you encounter issues with UBI 9, you can manually switch back to using UBI 8:

Table 1.3. Switching back to UBI 8

Images	Actions
Builder images	Set the builder image manually as follows: • **-Dquarkus.native.container-build=true** • **registry.access.redhat.com/quarkus/mandrel-for-jdk-21-rhel8:23.1**

Images	Actions

Runtime images	Depending on the mode, apply the following settings:
	• JVM mode:
	  ◦ In your Dockerfile in the **src/main/docker/** directory, specify **registry.access.redhat.com/ubi8/openjdk-21** as the base image of your container.
	• Native mode:
	  ◦ To use UBI minimal, specify **registry.access.redhat.com/ubi8/ubi-minimal:8.10**.

For more information, see the following resources:

- Red Hat Universal Base Image 9
- Quarkus Migration guide 3.19
- Creating a custom container image

1.7.4. Data

1.7.4.1. Datasource: Removal of implicit default URL for reactive SQL datasources

In earlier versions of Red Hat build of Quarkus, when Dev Services were disabled or running in production mode, the **quarkus.datasource.reactive.url** and **quarkus.datasource.<datasource-name>.reactive.url** properties were implicitly set to an undocumented default, targeting **localhost** with a database-specific port.

Starting in Red Hat build of Quarkus 3.20, these properties no longer have a default value when Dev Services are disabled or the application is running in production mode. If the property is not set, the corresponding datasource is deactivated. If the application attempts to use a deactivated datasource, it will fail at startup. For details, see the Datasource usage fails fast if datasource is deactivated or has no URL set release note.

If your application requires an active datasource and you want it to connect to a database on **localhost**, you must now set the **quarkus.datasource.reactive.url** or **quarkus.datasource.<datasource-name>.reactive.url** property explicitly. For example:

```
quarkus.datasource.reactive.url=postgresql://localhost:5432/mydatabase
```

In earlier versions, this configuration was set implicitly. Starting with this release, you must define the URL to activate the datasource.

1.7.4.2. Datasources without a URL no longer contribute to a health check

Previously, when Dev Services were disabled or in production mode, Red Hat build of Quarkus contributed a health check for datasources even if **quarkus.datasource.jdbc.url**, **quarkus.datasource.<datasource-name>.jdbc.url**, **quarkus.datasource.reactive.url**, or **quarkus.datasource.**

<datasource-name>.reactive.url properties were not set. This caused unreliable health checks that always succeeded for JDBC datasources and almost always failed for reactive datasources.

Starting with Red Hat build of Quarkus 3.20, datasources without a URL no longer contribute a health check. To ensure a health check is available, explicitly set the **quarkus.datasource.jdbc.url**, **quarkus.datasource.<datasource-name>.jdbc.url**, **quarkus.datasource.reactive.url**, or **quarkus.datasource.<datasource-name>.reactive.url** property.

1.7.4.3. Datasource usage fails fast if datasource is deactivated or has no URL set

In Red Hat build of Quarkus 3.20, applications start successfully even if a datasource is deactivated with **quarkus.datasource.active=false** or lacks a URL. This behavior often leads to runtime failures when the datasource is first accessed, especially when Dev Services are disabled or in production mode.

With this update, applications fail to start if Red Hat build of Quarkus detects that a datasource is used but is either deactivated or missing a URL.

Red Hat build of Quarkus now enforces stricter validation during startup to catch these issues early:

- **Static CDI Injection**: If a datasource is injected statically using **@Inject DataSource** or **@Inject Pool**, the application will fail to start with a clear and actionable error message.
- **Dynamic Retrieval**: Datasources retrieved dynamically, such as by using **Arc.container().instance()** or **@Inject Instance<DataSource>**, will not be detected during startup. However, retrieving these beans at runtime will throw an explicit exception with actionable guidance.

The same validation applies to the Flyway and Liquibase extensions for their **Flyway** and **LiquibaseFactory** CDI beans.

NOTE

Red Hat build of Quarkus 3.20 does not currently support the Flyway and Liquibase extensions. However, if you use these extensions, the described change might affect your migration.

Impact on Applications

- If your application uses a datasource, Flyway, or LiquibaseFactory bean that may be deactivated or lack a URL, you could encounter a startup failure like the following:

```
io.quarkus.arc.InactiveBeanException: Bean is not active: SYNTHETIC bean
[class=io.agroal.api.AgroalDataSource, id=sqqLi56D50iCdXmOjyjPSAxbLu0]
Reason: Datasource' <default>' was deactivated automatically because its URL was not set.
```

- To activate the datasource, set configuration property **quarkus.datasource.jdbc.url**.
  For more information, see the Configure data sources guide.

How to resolve if the datasource should be active

Ensure that all required configuration properties are set:

1. Set **quarkus.datasource.jdbc.url** or the appropriate configuration for the datasource type to ensure it is properly activated.

How to resolve if the datasource might be inactive

Adjust your code or configuration to handle inactive datasources gracefully:

1. Deactivate unused extensions: If an extension depends on a datasource that might not be active, deactivate the extension explicitly by setting its **active** configuration property to **false**. For example:
   - **quarkus.hibernate-search-standalone.active=false**
   - **quarkus.hibernate-search-orm.active=false**
   - **quarkus.hibernate-orm.active=false**
   - **quarkus.flyway.active=false**
   - **quarkus.datasource.active=false**
2. Inject dynamically: Instead of using static CDI injection, use **@Inject InjectableInstance<DataSource>** to check if the datasource bean is active before using it.

   ```
   if (ds.getHandle().getBean().isActive()) {
   DataSource dataSource = ds.get();
   // Use the datasource
   }
   ```

   These changes improve application reliability by detecting misconfigured datasources early, preventing unexpected runtime errors.

1.7.4.4. Flyway version 11 removes cleanOnValidationError

In Red Hat build of Quarkus 3.20, the **quarkus-flyway** extension upgrades Flyway to version 11, which removes the **cleanOnValidationError** configuration parameter. Additionally, calling **Flyway.validate()** no longer cleans on validation error.

To mitigate this change, Red Hat build of Quarkus 3.20 introduces the **quarkus.flyway.validate-at-start.clean-on-validation-error** configuration property, which provides behavior similar to **cleanOnValidationError**, but applies only when the application starts.

Workaround: If you require the previous behavior of **cleanOnValidationError** in explicit calls to **Flyway.validate()**, consider catching validation errors in your application and triggering a database cleanup explicitly using **Flyway.clean()**.

NOTE

Red Hat build of Quarkus 3.20 does not currently support the Flyway extension. However, if you use this extension, the described change might affect your migration.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

For more information, see the Quarkus Using Flyway guide.

1.7.4.5. IBM Db2 driver and container image upgraded to version 12

In Red Hat build of Quarkus 3.20, the IBM Db2 driver and container image that Dev Services uses have been upgraded to version 12.

The automated update described in the Migrating applications to Red Hat build of Quarkus 3.20 guide upgrades the IBM Db2 driver and container image to version 12.

Breaking changes

This upgrade introduces breaking changes to the license registration process and connectivity configuration.

Review the new license requirements, configuration steps, and SSL/TLS settings detailed on IBM's Features with behavior changes when upgrading to the Db2 Connect 12.1 driver for upgrading to Db2 Connect 12.1.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.4.6. Hibernate ORM Bean Validation contributes to DDL by default

In Red Hat build of Quarkus 3.20, the default behavior of Hibernate ORM's Bean Validation integration has changed. Previously, validation constraints were not considered during Data Definition Language (DDL) generation. With this release, Hibernate ORM includes applicable validation constraints in DDL by default, ensuring that database schemas align more closely with application-level constraints.

- To revert to the previous behavior where validation constraints do not influence DDL generation, set the **quarkus.hibernate-orm.validation.mode** configuration property to **callback**:

```
quarkus.hibernate-orm.validation.mode=callback
```

The **quarkus.hibernate-orm.validation.enabled** property has also been deprecated.

- To disable Bean Validation integration, use the following setting:

```
quarkus.hibernate-orm.validation.mode=none
```

For more information about these changes and guidance on migrating your application, see the Migration Guide 3.19, or the following resources:

- Validation modes and Hibernate Validator integration
- Apply validation modes to the Hibernate Reactive session factory config - GitHub PR

1.7.5. Logging

1.7.5.1. quarkus.log.*.json property deprecated

In Red Hat build of Quarkus 3.20, the **quarkus.log.*.json** configuration property has been deprecated. To enable JSON formatting for logging, use the **quarkus.log.*.json.enabled** property instead.

- Update your configuration as follows:

```
quarkus.log.console.json.enabled=true
```

This change was introduced to improve compatibility with YAML configuration, where the previous structure required using special syntax such as ~ to assign a value to the root key, leading to confusing or error-prone setups.

For more information, see the Migration Guide 3.19.

1.7.6. Observability

1.7.6.1. OpenTelemetry: Database incubating values moved

In Red Hat build of Quarkus 3.20, the **quarkus-opentelemetry** extension introduces breaking changes because some database incubating values, such as those related to Redis, were moved to a different package.

OpenTelemetry does not maintain a list of changes for database semantic conventions because they are not yet stable.

This change might require manual intervention, if you use manual instrumentation. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.6.2. SmallRye OpenTracing: Extension and related JDBC tracing configuration removed

In Red Hat build of Quarkus 3.20, the previously deprecated **quarkus-smallrye-opentracing** extension and related configuration property **quarkus.datasource.jdbc.tracing** have been removed.

The **quarkus-opentelemetry** extension is now the preferred tracing solution. To enable JDBC tracing with OpenTelemetry, use the following configuration property:

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

To migrate, replace the OpenTracing extension with OpenTelemetry and update your configuration accordingly.

1.7.7. Security

1.7.7.1. OIDC Client: Behavior changed if client does not have a URL

Starting in Red Hat build of Quarkus 3.20, if you use the **quarkus-oidc-client** extension but do not configure the OpenID Connect (OIDC) client to point to a specific URL, Dev Services for Keycloak automatically starts in dev mode.

To disable this behavior, add the following property to your **application.properties** file:

```
quarkus.keycloak.devservices.enabled=false
```

1.7.7.2. Security WebAuthn: Reimplementation by using WebAuthn4J

In Red Hat build of Quarkus 3.20, the **quarkus-security-webauthn** extension has been reimplemented by using WebAuthn4J to enhance security, align with industry standards, and improve long-term maintainability.

As a result, this update is not compatible with previous versions of the extension.

IMPORTANT

This release note is provided for users of the **quarkus-security-webauthn** extension. Although Red Hat build of Quarkus 3.20 does not yet support this extension, be aware that this change may impact your migration.

To transition smoothly to the new implementation, use the following information.

userName changes

- All **userName** references have been replaced with **username**.

Authenticator class changes

- The **Authenticator** class (from Vert.x) is no longer used and has been replaced functionally with **WebAuthnCredentialRecord**. This new class holds similar data, but as a WebAuthn4J subtype, requires different methods for accessing content:
  - **WebAuthnCredentialRecord.getRequiredPersistedData()** returns a **RequiredPersistedData** record with all necessary persistence data, simplifying storage management.
  - **WebAuthnCredentialRecord.fromRequiredPersistedData(RequiredPersistedData)**, a static method, reconstructs a **WebAuthnCredentialRecord** from stored data.
- If your application stores **Authenticator** data in JPA entities or database tables, you must migrate to the new **WebAuthnCredentialRecord** format. If you encounter issues, report them in the project repository.

WebAuthnUserProvider class changes

- **findWebAuthnCredentialsByUserName()** is now **findByUsername()**.
- **findWebAuthnCredentialsByCredID()** is now **findByCredentialId()**.
- **updateOrStoreWebAuthnCredentials()** has been split into the following:
  - **update(String credentialId, long counter)**
  - **store(WebAuthnCredentialRecord credentialRecord)**

Default endpoint changes

- **/q/webauthn/login** is now **/q/webauthn/login-options-challenge**.
- **/q/webauthn/register** is now **/q/webauthn/register-options-challenge**.
- **/q/webauthn/callback** has been split into the following endpoints, which are turned off by default for security reasons:
  - **/q/webauthn/login**
    To enable, set the **quarkus.webauthn.enable-login-endpoint** property.
  - **/q/webauthn/register**

To enable, set the **quarkus.webauthn.enable-registration-endpoint** property.

- The new **/q/webauthn/register** endpoint requires a **username** query parameter.
- **/.well-known/webauthn** has been added to return the list of allowed related origins.
- The user name for **login** and **login-options-challenge** is now optional.
- The **/q/webauthn/login-options-challenge** and **/q/webauthn/register** endpoints have moved from **POST** to **GET** methods and now accept parameters as query parameters instead of JSON bodies.

WebAuthnSecurity class changes

- Two methods have been added:
  - **getLoginOptionsChallenge()**
  - **getRegisterOptionsChallenge()**
- The **username** parameter for **login()** and **loginOptionsChallenge()** is now optional.
- The **register()** method now requires a **username** parameter due to the removal of the username cookie.

Configuration changes

- **quarkus.webauthn.require-resident-key** (boolean, default: **false**) has been replaced by **quarkus.webauthn.resident-key** (data enumeration, default: **REQUIRED**).
- **quarkus.webauthn.challenge-username-cookie-name** setting has been removed along with its associated cookie.
- The following new configuration settings have been added:
  - **quarkus.webauthn.load-metadata** (boolean, default: **false**) controls the loading of Fast Identity Online (FIDO) metadata.
  - **quarkus.webauthn.user-presence-required** (boolean, default: **true**) specifies whether user presence is required.
- **quarkus.webauthn.user-verification** now defaults to **REQUIRED** instead of **DISCOURAGED**.
- **quarkus.webauthn.timeout** now defaults to **5 minutes** instead of **1 minute**, in accordance with the WebAuthn standard.
- **quarkus.webauthn.pub-key-cred-params** is now **quarkus.webauthn.public-key-credential-parameters**.
- **quarkus.webauthn.origin** is now **quarkus.webauthn.origins** (plural) and now supports multiple origins, in accordance with the WebAuthn standard.
- The following new boolean configuration options have been added:
  - **quarkus.webauthn.enable-registration-endpoint** (boolean, default: **false**) enables the default registration endpoint.

- **quarkus.webauthn.enable-login-endpoint** (boolean, default: **false**) enables the default login endpoint.

WebAuthn credential verification changes

- WebAuthn credential attestation verification might behave differently for **quarkus.security.webauthn.attestation** settings other than **NONE** (the default).

quarkus-test-security-webauthn test module changes

- The **WebAuthnHardware** constructor now requires a **URL** parameter to represent the endpoint location. You can get this URL from your test classes by using **@TestHTTPResource URL url**.
- **WebAuthnEndpointHelper.invokeRegistration()** is now **obtainRegistrationChallenge()**.
- **WebAuthnEndpointHelper.invokeLogin()** is now **obtainLoginChallenge()** and accepts an optional **username** parameter.
- **WebAuthnEndpointHelper.invokeCallback()** has been split into the following:
  - **WebAuthnEndpointHelper.invokeRegistration()**, which requires a **username** parameter.
  - **WebAuthnEndpointHelper.invokeLogin()**, which accepts an optional **username** parameter.

JavaScript library changes

- **registerOnly()** is now **registerClientSteps()**.
- **loginOnly()** is now **loginClientSteps()**.
- **login()** and **loginClientSteps()** now accept an optional **user** parameter.
- The constructor parameter **registerPath** is now **registerOptionsChallengePath** (default: **/q/webauthn/register-options-challenge**).
- The constructor parameter **loginPath** is now **loginOptionsChallengePath** (default: **/q/webauthn/login-options-challenge**).
- The constructor parameter **callbackPath** has been split into the following:
  - **registerPath** (default: **/q/webauthn/register**)
  - **loginPath** (default: **/q/webauthn/login**)
- The constructor now accepts a **csrf** option of type **JsonObject** with two keys:
  - **header**, which specifies the header name used to include the Cross-Site Request Forgery Prevention (CSRF) token.
  - **value**, which specifies the CSRF token value.
    This update improves the security of custom endpoints protected by the **quarkus-rest-csrf** extension. For more information, see the Quarkus Cross-Site Request Forgery Prevention guide.

The status of this feature is downgraded from **preview** to **experimental** to allow for further stabilization.

For more information, see the Quarkus Using Security with WebAuthn guide.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.8. Tooling

1.7.8.1. @WithTestResource flaws fixed: restrictToAnnotatedClass replaced by scope

Earlier, Red Hat build of Quarkus 3.15 shipped with a version of **@WithTestResource** that was flawed and was thus not announced.

In Red Hat build of Quarkus 3.20, the flaws have been fixed, resulting in a breaking change. In this release, the earlier **restrictToAnnotatedClass** field has been replaced by **scope**.

1.7.8.2. JUnit 5 Mockito: Default mocking strategy changed to inline

In earlier releases, the **quarkus-junit5-mockito** dependency was configured to create mock objects by using the **subclass** mocking strategy.

Starting in Red Hat build of Quarkus 3.20, the dependency is now configured to use an **inline** mocking strategy by default.

1.7.8.3. Quarkus Test Framework JUnit 5 Mockito version alignment

In Red Hat build of Quarkus 3.20.0, the **quarkus-junit5-mockito** dependency was temporarily non-functional due to an intentional mismatch in its dependency versions. The version of Mockito in use required JUnit 5.11 or later, but the platform included JUnit 5.10.

To resolve the issue, Red Hat build of Quarkus 3.20.1 upgrades JUnit to version 5.12.1. This update aligns the versions of JUnit and Mockito and restores compatibility for the **quarkus-junit5-mockito** dependency.

The impact of this update is expected to be minimal. Testing in the community and Camel Quarkus has not revealed any regressions. For more information, see the related issue: https://github.com/quarkusio/quarkus/issues/46858.

NOTE

The quarkus-junit5-mockito extension is not supported or tested with the Red Hat build of Quarkus. It is mentioned here for awareness in case users have included it in their application dependencies.

Workaround

If you experience compatibility issues with JUnit 5.12.1, you can override the JUnit version in your Maven project to use the version from the Red Hat build of Quarkus 3.20.0.

To ensure consistent alignment across all JUnit artifacts, import the JUnit BOM version used in 3.20.0 as shown:

```
<dependencyManagement>
```

```
  <dependencies>
    <dependency>
      <groupId>org.junit</groupId>
      <artifactId>junit-bom</artifactId>
      <version>5.10.5</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

NOTE

The **junit-bom** is not a supported test library of Red Hat build of Quarkus 3.20.0. If you choose to use the BOM to align JUnit dependencies, use the upstream version from the JUnit project.

1.7.9. Web

1.7.9.1. HTTP compression now includes application/json and application/xhtml+xml by default

If HTTP compression is enabled, the **quarkus.http.compress-media-types** configuration property defines the list of media types to compress. In Red Hat build of Quarkus 3.20, the default value of this property has changed. Newly compressed media types now include **application/json** and **application/xhtml+xml**.

1.7.9.2. REST Client: Stricter configuration to optimize lookups

In Red Hat build of Quarkus 3.20, the REST Client Configuration became more strict to reduce the number of lookups and combinations required to retrieve the configuration:

- The MicroProfile Rest Client config style **[Simple Class Name]/mp-rest** does not work anymore. The specification does not specify this style. The **FQCN/mp-rest** style continues to function as expected and as specified by the MicroProfile Rest Client.
- Only REST Clients discovered by Red Hat build of Quarkus are loaded by the **RestClientsConfig** and **RestClientsBuildTimeConfig**.
- The **RestClientsConfig#clients** and **RestClientsBuildTimeConfig#client** maps always use the Fully Qualified Collection Name (FQCN) of the REST Client interface. Before, it would use all the keys found, even if related to the same REST Client duplicating entries.
- Removed legacy configuration names **quarkus.rest.client.max-redirects** and **quarkus.rest.client.multipart-post-encoder-mode**.

1.7.9.3. Qute: Default character escaping in JSON templates

Starting in Red Hat build of Quarkus 3.20, the **quarkus-qute** extension automatically escapes double quotes (**"**), backslashes (\), and control characters (**U+0000** through **U+001F**) in JSON templates if a corresponding template variant is set. The extension automatically assigns a variant to templates located in the **src/main/resources/templates** directory.

By default, **java.net.URLConnection#getFileNameMap()** determines the content type of a template file. You can define additional suffix-to-content-type mappings by using the **quarkus.qute.content-types** configuration.

To render the unescaped value, use the **raw** or **safe** properties, which are implemented as extension methods of **java.lang.Object**, or wrap the string value in the **io.quarkus.qute.RawString** class.

For more information, see the Character escapes section of the Quarkus "Qute reference" guide.

1.7.9.4. REST Jackson: ObjectMapperCustomizer behavior changed for default JSON processor instance

In Red Hat build of Quarkus 3.20, with the **quarkus-rest-jackson** extension, only **ObjectMapperCustomizer** beans without qualifiers are applied to the default **ObjectMapper** instance of the Jackson JSON processor.

In earlier versions, all customizer beans, including those with custom qualifiers, were applied to the default **ObjectMapper** instance.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

For more information, see the following resources:

- The Improved Jackson JSON processing with reflection-free deserialization section of the Release Notes for Red Hat build of Quarkus 3.20.
- The Configuring JSON support section of the Quarkus "Writing JSON REST Services" guide.

1.7.9.5. REST: Empty query parameters now handled as null or empty collection

In earlier releases, with the **quarkus-rest** extension, query parameters with empty values, such as **?foo=**, were deserialized as follows:

- **@RestQuery String foo** produced an empty string (**""**).
- **@RestQuery List<String> foo** produced a collection containing a single empty string.

In Red Hat build of Quarkus 3.20, this behavior has changed:

- **@RestQuery String foo** is now deserialized as **null**.
- **@RestQuery List<String> foo** is now deserialized as an empty collection.

Update your code accordingly.

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.7.9.6. WebSockets and WebSockets Client extensions deprecated

Red Hat build of Quarkus 3.20 deprecates the **quarkus-websockets** and **quarkus-websockets-client** extensions, which implement the Jakarta WebSocket specification.

Red Hat build of Quarkus plans to stop supporting these extensions in a future release.

To ensure compatibility with upcoming versions, migrate to the Quarkus WebSockets Next extension,**quarkus-websockets-next**, which offers a modern, more efficient WebSocket API.

For more information, see the following Quarkus resources:

- Getting started with WebSockets Next
- WebSockets Next reference guide
- Quarkus WebSockets Next extension section of the "Release Notes for Red Hat build of Quarkus 3.20" guide

This change requires manual intervention. It is not included in the automated update process described in the Migrating applications to Red Hat build of Quarkus 3.20 guide.

1.8. KNOWN ISSUES

Review the following known issues for insights into Red Hat build of Quarkus 3.20 limitations and workarounds.

1.8.1. FIPS: Known limitations when testing in FIPS-enabled environments

In Red Hat build of Quarkus 3.20, testing has been conducted to evaluate how Quarkus applications behave in environments with FIPS (Federal Information Processing Standards) mode enabled.

While testing generally completes successfully, some technology integrations and native image configurations currently present limitations that prevent validation or compilation.

Testing in FIPS-enabled environments has shown successful results for several key components, including OpenID Connect (OIDC), supported databases, caching, messaging with Kafka in non-native modes, and OpenTelemetry. However, some technology integrations and native configurations cannot currently be verified in these environments.

These findings do not indicate official support by Red Hat build of Quarkus or its components for FIPS.

Scenarios that cannot be verified

The following technology integrations or configurations are currently not verifiable in FIPS-enabled environments:

- MariaDB 11.x
- Infinispan client extension in both JVM and native mode with Mandrel 23.0 and 23.1
- Apache Kafka using SCRAM and OAUTHBEARER SASL mechanisms in native mode
- JDBC MSSQL driver in native mode
- DB2
- Reactive MSSQL client in native mode
- Native image compilation using Red Hat Mandrel 23.1 builder image

Notable related issues

The following public Jira tickets provide additional details on the limitations encountered:

- QUARKUS-5984: MariaDB 11.x fails to connect in FIPS-enabled environments due to compatibility issues.
- QUARKUS-2036: Infinispan client extension lacks support for FIPS-enabled environments.
- QUARKUS-2984: Native builds using JDBC MSSQL and Reactive MSSQL clients fail on RHEL 8 with FIPS enabled.
- QUARKUS-5232: SASL SCRAM mechanism is unusable in native mode within FIPS-enabled environments.
- QUARKUS-5233: SASL OAUTHBEARER mechanism is unusable in native mode within FIPS-enabled environments.
- MANDREL-254: Mandrel 23.1 builder image requires rework to support FIPS-enabled environments.
- QUARKUS-4387: Quarkus Reactive MySQL client is not supported in FIPS-enabled environment.
- QUARKUS-4612: Infinispan client extension fails on FIPS with native Mandrel 23.0 and 23.1.

Workaround

No workaround is currently available.

This release note is intended to preemptively disclose known challenges while broader compatibility testing continues.

1.8.2. Kafka Streams extension missing a native library on Microsoft Windows

Applications that use the **quarkus-kafka-streams** extension on Microsoft fail at runtime due to a missing native library, **librocksdbjni-win64.dll**.

During startup, this issue throws the following error:

```
java.lang.RuntimeException: librocksdbjni-win64.dll was not found inside JAR
```

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

No workaround is available at this time.

Example error

```
13:07:08,118 INFO  [app] ERROR: Failed to start application (with profile [prod])
13:07:08,118 INFO  [app] java.lang.RuntimeException: Failed to start quarkus
13:07:08,118 INFO  [app]  at io.quarkus.runner.ApplicationImpl.doStart(Unknown Source)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Application.start(Application.java:101)
13:07:08,118 INFO  [app]  at
io.quarkus.runtime.ApplicationLifecycleManager.run(ApplicationLifecycleManager.java:111)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:71)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:44)
13:07:08,118 INFO  [app]  at io.quarkus.runtime.Quarkus.run(Quarkus.java:124)
13:07:08,118 INFO  [app]  at io.quarkus.runner.GeneratedMain.main(Unknown Source)
13:07:08,118 INFO  [app] Caused by: java.lang.ExceptionInInitializerError
```

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

For more information, see QUARKUS-3434.

1.8.3. Missing native library for Snappy on Windows

In Red Hat build of Quarkus 3.20, running an application that uses the Snappy compression library on Windows produces an error due to a missing native library.

When attempting to compress data with Snappy in Windows environments, users might encounter an error message similar to the following example.

Example error message

```
...
org.eclipse.microprofile.reactive.messaging.Message$5@1e8dc267 from channel 'test' was not sent
to Kafka topic 'test' - nacking message: org.apache.kafka.common.KafkaException:
org.xerial.snappy.SnappyError: [FAILED_TO_LOAD_NATIVE_LIBRARY] no native library is found for
os.name=Windows and os.arch=x86_64
...
```

Workaround: No workaround is available at this time. This issue is planned to be resolved in a future release.

For more information, see QUARKUS-5983.

1.8.4. Native image build intermittently fails with unreachable type errors on Mandrel 23.1

In Red Hat build of Quarkus 3.20, building a native image with the Mandrel 23.1 runtime might fail intermittently due to heap snapshot verification errors.

The build process fails with an error indicating that a type was not marked as reachable:

```
AnalysisType<... reachable: false>
```

These errors are not consistent and can occur across different applications. They have not been reliably reproduced in a controlled environment.

Example error 1

```
com.oracle.graal.pointsto.util.AnalysisError: The heap snapshot verifier discovered a type not marked as reachable:
AnalysisType<VMOption$Origin[] -> HotSpotType<[Lcom/sun/management/VMOption$Origin;, resolved>, allocated: false, inHeap: false, reachable: false>
 at com.oracle.graal.pointsto.heap.HeapSnapshotVerifier$ScanningObserver.ensureTypeScanned(HeapSnapshotVerifier.java:332)
 ...
```

Example error 2

```
AnalysisType<MemoryType[] -> HotSpotType<[Ljava/lang/management/MemoryType;, resolved>, allocated: false, inHeap: false, reachable: false>
AnalysisType<HotSpotDiagnosticMXBean$ThreadDumpFormat[] -> HotSpotType<[Lcom/sun/management/HotSpotDiagnosticMXBean$ThreadDumpFormat;, resolved>, allocated: false, inHeap: false, reachable: false>
```

Example error 3

```
AnalysisType<MemoryType[] -> HotSpotType<[Ljava/lang/management/MemoryType;, resolved>, allocated: false, inHeap: false, reachable: false>
```

Workaround: Retry the native image build. Because the issue is intermittent, the build is likely to succeed on subsequent attempts.

This issue is planned to be resolved in a future release.

For more information, see MANDREL-332.

1.8.5. OpenTelemetry: Dev mode fails to reload when tracing is disabled and metrics are enabled

In Red Hat build of Quarkus 3.20, an application that uses the **quarkus-opentelemetry** extension does not restart in Dev mode if the following properties are set in the **application.properties** file:

- **quarkus.otel.traces.enabled=false**
- **quarkus.otel.metrics.enabled=true**

During the reload attempt, Quarkus throws an **UnsatisfiedResolutionException**:

Example error message

```
jakarta.enterprise.inject.UnsatisfiedResolutionException: No bean found for required type [class io.quarkus.opentelemetry.runtime.tracing.DelayedAttributes] and qualifiers [[@jakarta.enterprise.inject.Any()]]
```

Workaround

No workaround is available at this time. This issue is planned to be resolved in a future release.

1.8.6. Quarkus CLI cannot discover the TLS registry CLI plugin

Red Hat build of Quarkus 3.15 introduced the **quarkus-tls-registry-cli** plugin, which enables TLS registry support for the Quarkus CLI.

However, development tools do not currently discover Quarkus CLI plugins that are hosted on **maven.repository.redhat.com**. As a result, the Quarkus CLI cannot resolve the TLS registry CLI plugin by default.

When this issue occurs, the CLI returns an error similar to the following output:

```
[jbang] [ERROR] Could not resolve dependencies: The following artifacts could not be resolved:
io.quarkus:quarkus-tls-registry-cli:jar:3.20.3.redhat-00006 (absent): Could not find artifact
io.quarkus:quarkus-tls-registry-cli:jar:3.20.3.redhat-00006 in central
(https://repo1.maven.org/maven2/)
```

Workaround: To enable the Quarkus CLI to resolve the plugin, configure JBang to use the Red Hat Maven repository at **https://maven.repository.redhat.com/ga/**.

You can use one of the following methods:

- Create a **jbang.properties** file with the following content:

  ```
  run.repos=central,https://maven.repository.redhat.com/ga/
  ```

  To apply this configuration locally, place the file in the root directory of your project. To apply the configuration globally, place the file in the **~/.jbang** directory.

- If JBang is already installed, configure the repository globally by running the following command:

  ```
  jbang config set run.repos central,https://maven.repository.redhat.com/ga/
  ```

For more information, see QUARKUS-5183.

1.8.7. Quarkus CLI updates only the Red Hat build of Quarkus platform version

In Red Hat build of Quarkus 3.20, running the update command updates only the BOMs included in the Red Hat build of Quarkus platform, **com.redhat.quarkus.platform**. These can include BOMs such as **quarkus-camel-bom** and **quarkus-cxf-bom** if part of the release, **quarkus-qpid-jms-bom**, and **quarkus-operator-sdk-bom**.

However, the command does not update BOM versions tied to the upstream community release. If your project includes both Red Hat build of Quarkus and upstream BOMs, this issue can lead to potentially incompatible combinations of Quarkus and upstream BOMs.

Example update command

```
$ mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.20.3.redhat-00006:update -
Dmaven.repo.local=<path-to-local-repo>
```

Example pom.xml file with issue

```
<properties>
  <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
  <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
  <quarkus.platform.version>3.20.3.redhat-00003</quarkus.platform.version>
</properties>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>${quarkus.platform.artifact-id}</artifactId>
      <version>${quarkus.platform.version}</version>
    </dependency>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-amazon-services-bom</artifactId>
      <version>3.2.12.Final</version> (1)
    </dependency>
  </dependencies>
</dependencyManagement>
```

(1) Version not updated.

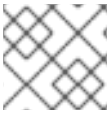
Workaround: Update the version numbers manually.

For more information, see QUARKUS-5185.

1.8.8. RSA cipher initialization triggers NPE in native mode with mandrel-for-jdk-21-rhel8:23.1 and FIPS enabled

In Red Hat build of Quarkus 3.20, using the **quarkus/mandrel-for-jdk-21-rhel8:23.1** Red Hat build of Quarkus Native Builder image to initialize an RSA cipher in native mode within a FIPS-enabled environment produces a **NullPointerException** (NPE) error.

This issue occurs only in native mode with FIPS enabled; it does not affect native mode with FIPS disabled. It also does not impact JVM mode when using Red Hat build of OpenJDK with FIPS enabled.



NOTE

Native mode is not supported in a FIPS-enabled environment.

Example error message

```
2024-10-17 10:45:01,931 ERROR [io.qua.ver.htt.run.QuarkusErrorHandler] (executor-thread-1)
HTTP Request to /repro failed, error id: 9b1f5dbb-058b-4c9b-9377-f3acc0a6cba5-1:
java.lang.RuntimeException: java.lang.NullPointerException
    at org.acme.ReproResource.init(ReproResource.java:38)
...
```

Workaround:

1. If present, remove the following property from the **application.properties** file:

```
quarkus.security.security-providers=SunPKCS11
```

2. Add the following property to the **application.properties** file to initialize the **ReproResource** class at runtime:

```
quarkus.native.additional-build-args=--initialize-at-run-time=org.acme.ReproResource
```

For more information, see MANDREL-245.

1.8.9. WebSockets Next: Connection error metrics not incremented on open event exceptions

In Red Hat build of Quarkus 3.20, with the **websockets-next** extension, when a **@WebSocket** endpoint throws an exception in the **@OnOpen** lifecycle method, the following metrics are not incremented:

- **quarkus_websockets_server_connections_opened_errors_total{uri=${ENDPOINT}}**
- **quarkus_websockets_client_connections_opened_errors_total{uri=${ENDPOINT}}**

This behavior affects both server-side and client-side WebSocket connection metrics.

Workaround: No workaround is available at this time. This issue is planned to be resolved in a future release.

For more information, see QUARKUS-5977.

1.9. RED HAT BUILD OF QUARKUS 3.20.3

Red Hat build of Quarkus 3.20 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.20.3.

1.9.1. Bug fixes

To view the list of issues resolved in this release, see the Red Hat build of Quarkus 3.20.3 bug fixes .

Among these fixes, note the following change that might affect compatibility: JUnit 5 and Mockito version alignment in the Quarkus test framework, listed in the "Changes that affect compatibility with earlier versions" section.

1.9.2. Security fixes

- CVE-2025-58056 **netty-codec-http**: Netty is vulnerable to request smuggling due to incorrect parsing of chunk extensions
- CVE-2025-58057 **netty-codec**: Netty's BrotliDecoder is vulnerable to DoS via zip bomb style

1.9.3. Advisory

Before you start using and deploying Red Hat build of Quarkus 3.20.3, review the following advisory related to the release.

- RHSA-2025:17563

1.10. RED HAT BUILD OF QUARKUS 3.20.2 SP1

Red Hat build of Quarkus 3.20 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.20.3 SP1.

1.10.1. Bug fixes

To view the list of issues resolved in this release, see Red Hat build of Quarkus 3.20.2 SP1 bug fixes .

1.10.2. Security fix

- CVE-2025-55163 **netty-codec-http2**: Netty MadeYouReset HTTP/2 DDoS Vulnerability

1.10.3. Advisory

Before you start using and deploying Red Hat build of Quarkus 3.20.2.SP1, review the following advisory related to the release.

- RHSA-2025:14008

1.11. RED HAT BUILD OF QUARKUS 3.20.2

Red Hat build of Quarkus 3.20 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.20.3.

1.11.1. Bug fixes

To view the list of issues resolved in this release, see the Red Hat build of Quarkus 3.20.2 bug fixes .

Among these fixes, note the following change that might affect compatibility: JUnit 5 and Mockito version alignment in the Quarkus test framework, listed in the "Changes that affect compatibility with earlier versions" section.

1.11.2. Security fixes

- CVE-2025-49146 **quarkus-bom**: pgjdbc insecure authentication in channel binding
- CVE-2025-49574 **quarkus-bom**: Quarkus potential data leak

1.11.3. Advisory

Before you start using and deploying Red Hat build of Quarkus 3.20.2, review the following advisory related to the release.

- RHSA-2025:13010

1.12. RED HAT BUILD OF QUARKUS 3.20.1

Red Hat build of Quarkus 3.20 provides increased stability and includes fixes to bugs that have a significant impact on users.

To get the latest fixes for Red Hat build of Quarkus, ensure you are using the latest available version, which is 3.20.3.

1.12.1. Bug fixes

To view the list of issues resolved in this release, see the Red Hat build of Quarkus 3.20.1 bug fixes .

Among these fixes, note the following change that might affect compatibility: JUnit 5 and Mockito version alignment in the Quarkus test framework, listed in the "Changes that affect compatibility with earlier versions" section.

1.12.2. Security fixes

- CVE-2025-24970 **io.netty/netty-handler**: SslHandler doesn't correctly validate packets which can lead to native crash when using native SSLEngine

1.12.3. Advisory

Before you start using and deploying Red Hat build of Quarkus 3.20.1, review the following advisory related to the release.

- RHSA-2025:8258

1.13. ADVISORY FOR RED HAT BUILD OF QUARKUS 3.20.0

Before you start using and deploying Red Hat build of Quarkus 3.20.0, review the following advisory related to the release.

- RHEA-2025:4162

1.14. ADDITIONAL RESOURCES

- Migrating applications to Red Hat build of Quarkus 3.20 guide.
- Getting Started with Red Hat build of Quarkus

Revised on 2025-10-13 09:43:09 UTC