Red Hat build of Quarkus 1.11

Release Notes for Red Hat build of Quarkus 1.11

Last Updated: 2021-04-26

Red Hat build of Quarkus 1.11 Release Notes for Red Hat build of Quarkus 1.11

Legal Notice

Abstract

This document contains release notes for Red Hat build of Quarkus 1.11.

Table of Contents

PREFACE

These release notes list new features, features in technology preview, known issues, and issues fixed in Red Hat build of Quarkus 1.11.

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. RED HAT BUILD OF QUARKUS

Red Hat build of Quarkus is a Kubernetes-native Java stack that is optimized for use with containers and Red Hat OpenShift Container Platform. Quarkus is designed to work with popular Java standards, frameworks, and libraries such as Eclipse MicroProfile, Apache Kafka, RESTEasy (JAX-RS), Hibernate ORM (JPA), Spring, Infinispan, and Apache Camel.

The Quarkus dependency injection solution is based on CDI (contexts and dependency injection) and includes an extension framework to expand functionality and to configure, boot, and integrate a framework into your application.

Quarkus provides a container-first approach to building Java applications. This approach makes it much easier to build microservices-based applications written in Java as well as enabling those applications to invoke functions running on serverless computing frameworks. For this reason, Quarkus applications have small memory footprints and fast startup times.

CHAPTER 2. QUARKUS METERING LABELS FOR RED HAT OPENSHIFT

You can add metering labels to your Quarkus pods and check Red Hat subscription details with the OpenShift Metering Operator.

NOTE

Do not add metering labels to any pods that an operator deploys and manages.

Quarkus can use the following metering labels:

- **com.redhat.component-name: "Quarkus"**
- **com.redhat.component-type: application**
- **com.redhat.component-version: 1.11.6**
- **com.redhat.product-name: "Red_Hat_Runtimes"**
- **com.redhat.product-version: 2021-Q1**

Additional resources

- Configuring and using Metering in OpenShift Container Platform

CHAPTER 3. NEW AND CHANGED FEATURES

This section provides an overview of new features and changes introduced in Red Hat build of Quarkus 1.11.6.

3.1. THE CODE.QUARKUS.REDHAT.COM PROJECT GENERATOR

Red Hat introduces a new web-based project generator that you can use to create projects for applications based on the latest release of Red Hat build of Quarkus. The project generator provides several features that make developing new applications with Quarkus easier, such as:

- a web-browser-based interface for selecting extension that you want to add to your project.
- automatic generation of the project directory structure based on the build tool that you choose.
- automatic importing and configuration of the extensions that you select to use with your applications.
- automatic generation of starter code for your application.

You can access the project generator at code.quarkus.redhat.com. Note that Red Hat only provides support for creating Maven-based application projects with code.quarkus.redhat.com.

For more information on how to use code.quarkus.redhat.com, see *Creating a Quarkus Maven project using code.quarkus.redhat.com*.

3.2. USE OF OPENJDK 11 UNIVERSAL BASE IMAGE AS THE NEW DEFAULT BASE IMAGE FOR SOURCE-TO-IMAGE BUILDS

Red Hat build of Quarkus 1.11 supports the use of the OpenJDK 11 Universal Base Image for building and deploying applications to Red Hat OpenShift Container Platform using the Source-to-image (S2I) tool. You can download the latest version of the image from the Red Hat Ecosystem Catalog.

3.3. NEW MICROMETER METRICS EXTENSION FOR MONITORING YOUR QUARKUS APPLICATIONS WITH PROMETHEUS

Red Hat build of Quarkus 1.11 introduces a new extension for collecting runtime application metrics using the Micrometer library and Monitoring your application with Prometheus. The extension allows you to collect runtime application metrics from your applications and also from extensions that your application uses, that integrate with Micrometer (such as the Quarkus extensions for Apache Kafka, HTTP, Resteasy, and others) The Quarkus Micrometer Metrics is supported by Red Hat for use in production environments. See: *Collecting metrics in your Quarkus applications*

3.4. SUPPORT FOR MULTIPLE HIBERNATE ORM PERSISTENCE UNITS

In Red Hat build of Quarkus 1.11 you can define multiple data sources as persistence units when you use Hibernate ORM to manage data sources in your application. See the section about *Configuring multiple JDBC data sources* in *Configuring data sources in your Quarkus applications* for more details.

3.5. SUPPORT FOR SAVING GENERATED OPENAPI SCHEMAS

Red Hat build of Quarkus 1.11 introduces support for saving OpenAPI schemas generated for your applications with the Quarkus Smallrye OpenAPI extension. You can set the value of the **quarkus-**

smallrye-openapi_quarkus.smallrye-openapi.store-schema-directory to the path of the directory in which the YAML and JSON files that contain the OpenAPI schemas are saved when you compile your application. For example:

application.properties

```
quarkus-smallrye-openapi_quarkus.smallrye-openapi.store-schema-
directory=/path/to/schema/directory
```

3.6. ARC CONTEXT AND DEPENDENCY INJECTION SUPPORT IN THE QUARKUS QUARTZ EXTENSION

You can now use the Quarkus Quartz extension to schedule periodic tasks that rely on Context and Dependency Injection. See the community documentation for the Quarkus Quartz extension for details.

3.7. SMALLRYE REACTIVE MESSAGING UPGRADE TO VERSION 2.7.1

In Red Hat build of Quarkus 1.11, the SmallRye Reactive Messaging API used in the Quarkus Reactive Messaging Extensions for AMQP and Apache Kafka has been upgraded to version 2.7.1. See SmallRye Reactive Messaging API documentation for more details.

3.8. MUTINY REACTIVE API UPGRADE TO VERSION 0.12.5

In Red Hat build of Quarkus 1.11, the Mutiny event-driven library used in reactive extensions for Quarkus has been upgraded to version 0.12.5.

3.9. SUPPORT FOR BEAN VALIDATION IN REACTIVE ROUTES

The Reactive Routes Extensions in Red Hat build of Quarkus 1.11 supports constraint validation for Java beans.

3.10. CHANGE TO JACKSON AS THE DEFAULT JSON SERIALIZATION AND DESERIALIZATION TOOL FOR QUARKUS REST APPLICATIONS

WARNING

This change might break the REST endpoints in your application when you upgrade your application from Red Hat build of Quarkus 1.7 to Red Hat build of Quarkus 1.11. Update your application code to ensure that object mapping works correctly after you upgrade your applications.

In the Red Hat build of Quarkus 1.11 release, Jackson is set as the default **ObjectMapper** tool used by the the Quarkus REST JSON Extension. You can inject Jackson in your the REST Controller class of your application using Context and Dependency Injection to provide support for converting your REST application data to and from the JSON format. For more details about breaking changes caused by

disabling the **DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES** feature in Jackson in Red Hat build of Quarkus 1.11, see *Upgrading your applications from Red Hat build of Quarkus 1.7 to Red Hat build of Quarkus 1.11*

3.11. NEW OPTION FOR ENABLING NON-APPLICATION USER INTERFACES WHEN STARTING YOUR APPLICATION IN PRODUCTION MODE

In Red Hat build of Quarkus 1.11, you can specify the **-Dquarkus.<ui-name>.always-include=true** to enable user interfaces that are part of the JAR when you start the application in Production mode. The option is available for the following interfaces:

- Swagger UI
- OpenAPI
- SmallRye Health UI
- GraphQL UI

For example, when you create a JAR for a REST application that contains a SwaggerUI interface, this interface is disabled by default when you start the application. You can append the **-Dquarkus.swagger-ui.always-include=true** option to the start command to enable the interface when you start the application:

```
java -jar -Dquarkus.swagger-ui.enable=true target/<application-name>-1.0.0-SNAPSHOT-runner.jar
```

Note, that you must replace **<application-name>** with the name of your JAR.

3.12. QUARKUS REST CLIENT SECURITY UPDATE TO RESOLVE CVE-2020-25633

The **quarkus-rest-client** extension available in Red Hat build of Quarkus 1.11 is affected by a change in the handling of **WebApplicationException** by MicroProfile REST Client and JAX-RS client that is introduced as part of an update that resolves the CVE-2020-25633 security issue.

As a result of the security update, RESTEasy version 4.5.9. changes the way **Response** is handled when a Client application returns a **WebApplicationException**. Before the 4.5.9 update, the **Response** that was sent from the remote server form the local domain contained sensitive information about the remote server (for example, cookies) that an unauthorized party could potentially gain access to.

The RESTEasy 4.5.9 update changes the way the **Response** contents are treated. When the local server receives the **Response** RESTEasy removes all the sensitive content before storing the response, but retains a way for the local server to access the original content of the **Response** if necessary.

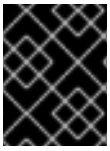
This change to exception handling makes it possible to avoid the security risks associated with storing sensitive content on the local server, but still ensures that Clients using RESTEasy version 4.5.9 remain compatible with the JAX-RS specification.

For details about the change in storing **Response** contents in RESTEasy version 4.5.9, see the section about **WebApplicationException** handling in the RESTEasy 4.5.9 documentation.

3.13. UPGRADE OF THE DEFAULT MANDREL BASE IMAGE FOR COMPILING NATIVE EXECUTABLES TO VERSION 20.3

In Red Hat build of Quarkus 1.11, the default base image for compiling native executables is upgraded to Mandrel 20.3. As a result, the default value of the **quarkus.native.builder-image** configuration property for compiling Quarkus applications to native executables changes to **quay.io/quarkus/ubi-quarkus-mandrel:20.3-java11**.

3.14. QUARKUS KUBERNETES CLIENT UPGRADED TO VERSION 5.X



IMPORTANT

Red Hat does not provide support for using the Quarkus Kubernetes Client in production environments.

Red Hat build of Quarkus 1.11 includes a new version of Quarkus Kubernetes Client. If you are using the Quarkus Kubernetes Client version 4.x in a development environment and want to upgrade your existing applications to version 5.x, see the Kubernetes Client Migration Guide .

3.15. QUARKUS DEV UI

IMPORTANT

Red Hat does not provide support for using the Quarkus Dev UI in production environments.

Red Hat build of Quarkus 1.11 uses includes the first release of Quarkus Dev UI. Dev UI is an experimental interface that you can use to:

- view a list of extension that you are currently using in your application,
- see the status of the extensions in your application
- access the documentation for the extensions that you are using in your application.

You can access Dev UI when you start your application in Development mode and navigate to **localhost:8080/q/dev** in your web browser.

CHAPTER 4. UPGRADING YOUR APPLICATIONS FROM RED HAT BUILD OF QUARKUS 1.7 TO RED HAT BUILD OF QUARKUS 1.11

This section provides an overview of breaking changes that you must address when you upgrade your applications from Red Hat build of Quarkus 1.7 to Red Hat build of Quarkus 1.11.

4.1. CHANGE OF CONFIGURATION PROPERTIES FOR THE QUARKUS QUARTZ EXTENSION

The release of Red Hat build of Quarkus 1.11 introduces several changes to the configuration properties available for the Quarkus Quartz extension.

New configuration properties introduced in Red Hat build of Quarkus 1.11

- **quarkus.quartz.cluster-checkin-interval**
- **quarkus.quartz.instance-name**
- **quarkus.quartz.start-mode**

Configuration properties removed in Red Hat build of Quarkus 1.11

- **quarkus.quartz.force-start**

Table 4.1. Configuration properties renamed in Red Hat build of Quarkus 1.11

Property name in Quarkus 1.7	Property name in Quarkus 1.11
quarkus.quartz.triggerListener."namedTriggerListener".class	quarkus.quartz.trigger-listeners."listener-name".class
quarkus.quartz.triggerListener."namedTriggerListener"	quarkus.quartz.trigger-listeners."listener-name".properties
quarkus.quartz.jobListener."namedJobListener".class	quarkus.quartz.job-listeners."listener-name".class
quarkus.quartz.jobListener."namedJobListener"	quarkus.quartz.job-listeners."listener-name".properties
quarkus.quartz.plugin."namedPlugin".class	quarkus.quartz.plugins."plugin-name".class
quarkus.quartz.plugin."namedPlugin"	quarkus.quartz.plugins."plugin-name".properties

4.2. CHANGE OF NAMING STRATEGY FOR SPRING BOOT CONFIGURATION PROPERTIES

In Red Hat build of Quarkus 1.11 the naming strategy used for Spring Boot configuration properties that contain a combination of uppercase and lowercase characters in Quarkus application is no longer set to **verbatim**.

Instead you can set the **quarkus.arc.config-properties-default-naming-strategy** property to one of the following values in the **application.properties** file of your project:

from-config

the naming convention is specified in your application configuration

verbatim

the name of the configuration property matches the name of the field or method to which the property applies

kebab

the name of the configuration property is uses lowercase characters with spaces replaced by hyphens. For example: **application-name**

If you do not set the **quarkus.arc.config-properties-default-naming-strategy** property for your application, **kebab** is used as the default value.

If you are using Spring Boot configuration properties that are formatted according to the **verbatim** naming strategy in your application, ensure that you make one of the following changes:

- Set the value of the **quarkus.arc.config-properties-default-naming-strategy** to **verbatim** in the **application.properties** or **application.yml** file of your project. For example:

  application.properties

  ```
  quarkus.arc.config-properties-default-naming-strategy=verbatim
  ```

- Convert then names of configuration properties that you use in application to match the **kebab** naming strategy.

4.3. REMOVAL OF SUPPORT FOR THE QUARKUS.DATASOURCE.URL AND QUARKUS.DATASOURCE.DRIVER DATA SOURCE CONFIGURATION PROPERTIES

Red Hat build of Quarkus 1.11 no longer supports properties for configuring the connection URLs and drivers of data sources introduced in Red Hat build of Quarkus 1.3.

You must replace the unsupported configuration properties with properties compatible with Quarkus 1.11 to ensure that your data source configuration works properly when you upgrade your application to Quarkus 1.11.

See *Configuring data sources in your Quarkus applications* for details on how to configure your data sources in Quarkus 1.11 applications.

4.4. CHANGE OF DEFAULT MEDIA TYPE TO JSON FOR QUARKUS APPLICATIONS

WARNING

This change might break the REST endpoints in your application when you upgrade your application from Red Hat build of Quarkus 1.7 to Red Hat build of Quarkus 1.11. Update the format of the return types used by your the REST endpoint in your application to ensure that your REST endpoints continue to work correctly after your upgrade your application.

In the Red Hat build of Quarkus 1.11 release, the default format for serializing application data is changed to JSON.

You can disable the default use of the JSON as content-type format for REST endpoints in your application, and use annotations to explictly enable the use of JSON only for interfaces that you want to use it for:

1. Set the value of the **quarkus.resteasy-json.default-json** property to **false** in the **application.properties** file of your application.

   application.properties

   ```
   quarkus.resteasy-json.default-json=false
   ```

2. Add the **@Produces(MediaType.APPLICATION_JSON)** and **@Consumes(MediaType.APPLICATION_JSON)** annotations to the REST endpoints of your applications for which you want to use JSON as the content type format.

4.5. THE FAIL_ON_UNKNOWN_PROPERTIES FEATURE IS DISABLED IN JACKSON BY DEFAULT

In Red Hat build of Quarkus 1.11, Jackson is configured to ignore unknown properties by having the **DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES** disabled by default to avoid failures while attempting to deserialize JSON data objects containing properties not recognized by your application.

Set the value of the **quarkus.jackson.fail-on-unknown-properties** to **true** in the **application.properties** file of your application to enable the **FAIL_ON_UNKNOWN_PROPERTIES** feature. You must enable this feature separately for each class of your applications:

application.properties

```
<fully-qualified-class-name>.quarkus.jackson.fail-on-unknown-properties=true
```

See the Quarkus REST JSON extension guide for more details about using and customizing Jackson in your {PrductName} REST application.

4.6. CHANGES TO CONFIGURING THE MINIMUM LOGGING LEVEL AT THE DEBUG AND TRACE LEVELS

In Quarkus 1.11, you must set the minimum logging level when you configure your logger to log messages at the **DEBUG** and **TRACE** level. You must set the minimum log level before you start your application. If you do not set the minimum logging level in your logger configuration, messages at the **DEBUG** and **TRACE** log level will not appear in your log output. For example, if you configure your logger to log at **TRACE** level, you must minimum log level to **TRACE**. Otherwise, messages at **TRACE** level will not appear in your log output.

You can set the minimum log level for a specific category in the **application.properties** file of your application. **application.properties**. For example, to enable logging at the **TRACE** level for the **io.quarkus.smallrye.jwt** and **io.undertow.request.security** categories in your logger configuration, you can set the following properties

application.properties

```
quarkus.log.file.enable=true
# Send output to a trace.log file under the /tmp directory
quarkus.log.file.path=/tmp/trace.log
quarkus.log.file.level=TRACE
quarkus.log.file.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
# Set 2 categories (io.quarkus.smallrye.jwt, io.undertow.request.security) to TRACE level
quarkus.log.min-level=TRACE
quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE
quarkus.log.category."io.undertow.request.security".level=TRACE
```

4.7. CHANGES TO THE INTERNAL STRUCTURE OF THE RED HAT BUILD OF QUARKUS BOM

In Red Hat build of Quarkus 1.11, the **com.redhat.quarkus:quarkus-universe-bom** no longer directly contains the **goupId**, **artifactID** and **version** of all extensions and dependencies. Instead, the **com.redhat.quarkus:quarkus-universe-bom** imports the **com.redhat.quarkus quarkus-product-bom** that contains the dependency declarations for all Red Hat build of Quarkus extensions supported by Red Hat and the **io.quarkus:quarkus-universe-bom** that contains all the community Quarkus extensions.

This change does not impact the way that you can use the Red Hat build of Quarkus BOM in your Maven project. However, if you are using custom scripts to parse the BOM, you must update them to ensure that the parsing continues to work correctly after you upgrade your application to Red Hat build of Quarkus 1.11.

4.8. CHANGE IN REST ENDPOINT PATH RESOLUTION

WARNING

This change might break the REST endpoints in your application when you upgrade your application from Red Hat build of Quarkus 1.7 to Red Hat build of Quarkus 1.11. Ensure that you update your endpoint paths after migrating your application.

In Red Hat build of Quarkus 1.11, paths of application and non-application REST endpoints are resolved relative to the common absolute root path. The default common root path for REST endpoints is set to:

- / for REST endpoints directly exposed by the main REST controller class of your application. You can change the default path for application endpoints by changing the value of the **quarkus.http.root-path** property in the **application.properties** file of your project.
- **q** for REST endpoints for services provided by tools integrated with your application (for purposes such as application health monitoring or metrics collection). You can change the default path for application endpoints by changing the value of the **quarkus.http.non-application-root-path** property in the **application.properties** file of your project.

Note, that relative root paths are nested under the root path defined by the **quarkus.http.root-path** property. This means that, for example, if the root path defined in the **quarkus.http.root-path** property is set to /, and the root path for non-application endpoints defined by the **quarkus.http.non-applicationroot-path** property is set to **q**, the absolute endpoint path for the non-application endpoint is /**q**/**<non-application-endpoint-name>**.

However, you can also configure the paths of individual non-application endpoints explicitly to be located at /**q**/**<non-application-endpoint-name>**.

Because endpoint paths are interpreted as relative to the the root paths set by **quarkus.http.root-path** and **quarkus.http.non-application-root-path** you must exclude the leading slash (/) character from the custom paths and sub-paths that you configure for endpoints in your application.

For example, when you expose a metrics endpoint for Prometheus in a REST Controller your application, you must set the endpoint path in the **@Path** annotation to **metrics** to ensure that your endpoint is exposed at /**q**/**metrics**. Setting the same path value to /**metrics** exposes your metrics endpoint at /**metrics**.

Example of configuring non-application endpoints under a separate namespace

For example, you can set the following properties in the **application.properties** file of your project to expose a **hello** endpoint application at the /**api** root path, and the **metrics** endpoint at the /**api**/**q** path:

application.properties

```
quarkus.http.root-path=/api
quarkus.http.non-application-root-path=q
```

In this configuration, both the /**api**/**hello** and the /**api**/**q**/**metrics** are public. This means that any user with the permission to access the /**api**/**hello** can also send a request to the /**api**/**q**/**metrics** endpoint and receive a valid response.

When you want to make the **health** endpoint non-public, you can set the root path for non-application endpoints in the **application.properties** to the /**q** namespace:

application.properties

```
quarkus.http.root-path=/api
quarkus.http.non-application-root-path=/q
```

In this configuration, the /**api**/**hello** endpoint is public, but the /**q**/**metrics** is exposed in a separate namespace for which you can configure different access permissions.

In Red Hat build of Quarkus 1.11, requests sent to the original non-application endpoint paths are automatically redirected to the new paths in the /**q** namespace.

You can set the following attribute in the **application.properties** file of your project to disable the automatic redirection of non-application endpoint paths:

```
quarkus.http.redirect-to-non-application-root-path=false
```

You can switch your applications to using the absolute endpoint root path for all endpoints by setting the value of the **quarkus.http.non-application-root-path** to a variable that resolves to the value of the absolute application endpoint root:

```
quarkus.http.non-application-root-path=${quarkus.http.root-path}
```

4.9. ADDITIONAL CONFIGURATION PROPERTIES ARE REQUIRED WHEN PROCESSING CONFIGMAP OBJECTS FOR DEPLOYING REST APPLICATION TO RED HAT OPENSHIFT CONTAINER PLATFORM

WARNING

This change might break the configuration that you use to deploy your applications to OpenShift when you upgrade your application from Red Hat build of Quarkus 1.7 to Red Hat build of Quarkus 1.11. You must update the **applications.properties** file of your application to ensure that configuration parameters provided in your ConfigMap are recognized by your application.

In Red Hat build of Quarkus 1.11, when you configure a Quarkus application based on Resteasy to Red Hat OpenShift Container Platform, and provide the configuration parameters for the application in a ConfigMap that is specified in the **quarkus.openshift**, you must also specify the **quarkus.openshift.app-secret** and **quarkus-openshift.app-configmap** properties in your **application.properties** file to ensure that your application recognizes and processes the ConfigMap.

- Add the following properties in the **application.properties** file of your application to ensure that your application recognizes the ConfigMap:

  application.properties

  ```
  quarkus.openshift.app-secret=<secret-name>
  quarkus-openshift.app-configmap=<configmap-name>
  ```

  You must replace the **<secret-name>** with the name of the secret that you want to use, and you must replace **<configmap-name>** with the name of the ConfigMap that you want to use.

CHAPTER 5. RED HAT BUILD OF QUARKUS SUPPORTED PLATFORMS, CONFIGURATIONS, EXTENSIONS, AND DEPENDENCIES

- For a list of supported configurations and tested integrations see the Red Hat build of Quarkus Supported Configurations page (login required).
- For a list of supported Maven artifacts see the Red Hat build of Quarkus Component Details page (login required).

For a list of extensions and dependencies available as Technology Preview in Red Hat build of Quarkus 1.11, see the Component details overview.

5.1. SUPPORTED EXTENSIONS, DEPENDENCIES AND PLUGINS

The following supported extensions are added in Red Hat build of Quarkus 1.11:

- Quarkus Micrometer
- Quarkus OpenID Connect Client
- Quarkus OpenID Connect Client Filter
- Quarkus Resteasy Multipart

For a list of Red Hat build of Quarkus extensions, dependencies, and plugins that Red Hat supports for use in production environments see the Red Hat build of Quarkus Component Details page (login required).

5.2. DEVELOPMENT SUPPORT

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

Features

- Live development mode
- Remote development mode
- Dev UI

Plug-ins

- **protobuf-maven-plugin**

CHAPTER 6. DEPRECATED COMPONENTS AND FEATURES

The components and features listed in this section are deprecated with Red Hat build of Quarkus 1.11. 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 components and features deprecated in Red Hat build of Quarkus 1.11 Red Hat build of Quarkus Component Details page (login required).

CHAPTER 7. TECHNOLOGY PREVIEW

This section lists features and extensions that are available as Technology Preview in Red Hat build of Quarkus 1.11.

IMPORTANT

These features are for Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for 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 on Red Hat Technology Preview features, see Technology Preview Features Scope.

7.1. TECHNOLOGY PREVIEW FEATURES

7.1.1. Packaging Quarkus application as a fast-jar

The fast-jar packaging format is an alternative to the default JAR packaging format that provides faster startup times for your applications. You can enable fast-jar packaging by setting the following property in the **application.properties** file of your project:

application.properties

```
quarkus.package.type=fast-jar
```

Alternatively, you can append the **-Dquarkus.package.type=fast-jar** property to the command that you use to package your application:

```
mvn clean package -Dquarkus.package.type=fast-jar
```

7.2. TECHNOLOGY PREVIEW EXTENSIONS AND DEPENDENCIES

For a list of extensions and dependencies available as Technology Preview in Red Hat build of Quarkus 1.11, see the Red Hat build of Quarkus Component Details page (login required).

CHAPTER 8. KNOWN ISSUES

This section lists known issues with Red Hat build of Quarkus 1.11.

- Issue #11633 Missing zero-configuration solution for OpenShift Serverless. This issue affects only deployment of Quarkus native Serverless applications.
- QUARKUS-695 Quarkus Keycloak Authorization is not compatible with the latest version of Red Hat Single Sign-On 7.4.
- QUARKUS-697 Default method fallback does not work when using when using MicroProfile HTTP Client in native node
- QUARKUS-719 Quarkus Reactive PG Client displays a **Fail to read any response from the server, the underlying connection might get lost unexpectedly** error message when trying to use a database connection connection from pgPool even when the connection is available.

Revised on 2021-04-26 13:49:04 UTC