Red Hat build of Eclipse Vert.x 4.3

Release Notes for Eclipse Vert.x 4.3

For use with Eclipse Vert.x 4.3.7

Last Updated: 2023-02-16

Red Hat build of Eclipse Vert.x 4.3 Release Notes for Eclipse Vert.x 4.3

For use with Eclipse Vert.x 4.3.7

Abstract

This Release Note contains important information related to Eclipse Vert.x 4.3.7

Table of Contents

PREFACE

Date of release: 2023-02-13

RED HAT BUILD OF ECLIPSE VERT.X 4.3.7 - PLANNED END OF LIFE

The Red Hat build of Eclipse Vert.x 4.3.7 is the last supported release that Red Hat plans to provide. The full support ends on May 31, 2023. See the product life cycle page for details. Red Hat will continue to deliver security and bug fixes for Red Hat build of Eclipse Vert.x with 4.3.x releases until the product end of life.

You can migrate Eclipse Vert.x applications to the Red Hat build of Quarkus.

Red Hat build of Quarkus

Quarkus is a Kubernetes-native Java framework tailored for JVM and native compilation, created using the best Java libraries and standards. It provides an effective solution for running Java applications in environments such as serverless, microservices, containers, Kubernetes, FaaS, or the cloud.
The reactive capabilities of Quarkus use Eclipse Vert.x internally, and you can reuse Eclipse Vert.x applications in Quarkus. Therefore, migration of Eclipse Vert.x applications to Quarkus is the recommended option.

See the Quarkus product page and documentation for more information.

We will create resources to help you with the migration process.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments.

This section explains how to submit feedback.

Prerequisites

- You are logged in to the Red Hat Customer Portal.
- In the Red Hat Customer Portal, view the document in **Multi-page HTML** format.

Procedure

To provide your feedback, perform the following steps:

1. Click the **Feedback** button in the top-right corner of the document to see existing feedback.

   NOTE

   The feedback feature is enabled only in the **Multi-page HTML** format.

2. Highlight the section of the document where you want to provide feedback.
3. Click the **Add Feedback** pop-up that appears near the highlighted text.
   A text box appears in the feedback section on the right side of the page.
4. Enter your feedback in the text box and click **Submit**.
   A documentation issue is created.
5. To view the issue, click the issue tracker link in the feedback view.

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. REQUIRED INFRASTRUCTURE COMPONENT VERSIONS

When you work with Red Hat build of Eclipse Vert.x, you can use the following components. However, Red Hat does not provide support for components listed below except Red Hat OpenShift cluster and Red Hat OpenJDK.

Required components

The following components are required to build and develop applications using Eclipse Vert.x.

Component name	Version
Maven	3.6.0 or later
JDK[a]	8, 11 or 17

[a] A full JDK installation is required, because JRE does not provide tools for compiling Java applications from source.

Optional components

Red Hat recommends using the following components depending on your development and production environments.

Component name	Version
git	2.0 or later
OpenShift Maven Plugin	1.1.1
oc command line tool	3.11 or later[a]
Access to a Red Hat OpenShift cluster[b]	3.11 or later

[a] The version of the **oc** CLI tool should correspond to the version of OCP that you are using.

[b] OpenShiftCluster is supported by Red Hat

CHAPTER 2. SUPPORTED ECLIPSE VERT.X RUNTIME COMPONENT CONFIGURATIONS AND INTEGRATIONS

The following resource defines the supported configurations and integrations of Red Hat products with Eclipse Vert.x:

- For a list of technologies that are supported for integration with Eclipse Vert.x in production environments see the Supported Eclipse Vert.x configurations and integrations.
- For a list of Eclipse Vert.x runtime artifacts and their versions see the component details page.

CHAPTER 3. FEATURES

3.1. NEW AND CHANGED FEATURES

This section describes the new functionalities introduced in this release. It also contains information about changes in the existing functionalities.

3.1.1. New or changed features introduced in the 4.3 release

Eclipse Vert.x 4.3 provides the following new or changed features.

3.1.1.1. Micrometer adds the metric type to JMX object names

From Eclipse Vert.x 4.3.4 onward, because of the upgrade to Micrometer 1.9.3, object names now include the metric type when using Eclipse Vert.x Micrometer Metrics with Java Management Extensions (JMX).

This enhancement is only relevant for users of the **micrometer-registry-jmx** module.

3.1.1.2. Eclipse Vert.x with GraphQL Java 19 uses platform locale by default

From Eclipse Vert.x 4.3.3 onward, Eclipse Vert.x supports version 19 of GraphQL Java, which is the Java server implementation of the GraphQL query language. When using GraphQL Java 19, if you do not set a locale in the JVM, the GraphQL engine now uses the JVM default locale, which is the locale of the platform where the JVM is installed. Alternatively, you can configure the JVM default **Locale** to use a different value or you can use the Eclipse Vert.x Web GraphQL handler to set a custom locale.

NOTE

Eclipse Vert.x 4.3.3 or later also supports version 18 of GraphQL Java.

3.1.1.3. Users who use **jackson-databind** features must include this dependency in their projects

From Eclipse Vert.x 4.3.2 onward, if you use the Jackson Databind library with the **vertx-web-openapi**, **vertx-auth-webauthn**, or **vertx-config-yaml** module, you must add the following dependency to the project descriptor:

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
</dependency>
```

Because the Jackson Databind library has caused some security vulnerabilities and other modules typically only use Jackson Databind to perform some internal action, the use of Eclipse Vert.x parsers supersedes any need to use the **vertx-web-openapi**, **vertx-auth-webauthn**, or **vertx-config-yaml** module with Jackson Databind. However, if you want to continue using any of these modules with Jackson Databind, you must explicitly include this dependency in your project, as shown in the preceding example.

3.1.1.4. Changes in body handler setup with Eclipse Vert.x OpenAPI

From Eclipse Vert.x 4.3.1 onward, Eclipse Vert.x OpenAPI requires use of the **routerBuilder.rootHandler()** method, to ensure that the body handler is set up in the correct order *after* any PLATFORM or SECURITY_POLICY handlers.

For example:

```
BodyHandler bodyHandler = BodyHandler.create("my-uploads");
routerBuilder.rootHandler(bodyHandler);
```

In earlier releases of Eclipse Vert.x, Eclipse Vert.x OpenAPI supported the **routerBuild.bodyHandler()** method for adding the body handler. However, the **bodyHandler()** method had the following disadvantages:

- Eclipse Vert.x did not perform any validation to ensure that the setup was in the correct order.
- Eclipse Vert.x OpenAPI stored the body handler as a special handler to ensure that it would always be the first handler on the route, but this was not always guaranteed.

The **bodyHandler()** method is deprecated in Eclipse Vert.x 4.3.1. The preceding **rootHandler** call now supersedes the following **bodyHandler** call that was available in previous versions:

```
BodyHandler bodyHandler = BodyHandler.create("my-uploads");
routerBuilder.bodyHandler(bodyHandler);
```

3.1.1.5. Eclipse Vert.x reactive Oracle client enhancements for BLOB and RAW data values

From Eclipse Vert.x 4.3.1 onward, the Eclipse Vert.x reactive Oracle client includes the following enhancements for **BLOB** and **RAW** data:

- When reading **BLOB** or **RAW** data, the client now returns an **io.vertx.core.buffer.Buffer** value.
  For example:

  ```
  client.preparedQuery("SELECT data FROM images WHERE id = ?")
     .execute(Tuple.of(id))
     .onComplete(ar -> {
        if (ar.succeeded()) {
           Row row = ar.result().iterator().next();

           // Use io.vertx.core.buffer.Buffer when reading
           Buffer data = row.getBuffer("data");
        }
     });
  ```

  NOTE

  This change was introduced for consistency as part of fixing an issue with **RAW** values as query parameters. In earlier releases of Eclipse Vert.x, **BLOB** or **RAW** data was returned as a byte array.

- When writing or filtering **BLOB** data, the data is now represented by a new **io.vertx.oracleclient.data.Blob** type.
  For example:

```
client.preparedQuery("INSERT INTO images (name, data) VALUES (?, ?)")
    // Use io.vertx.oracleclient.data.Blob when inserting
    .execute(Tuple.of("beautiful-sunset.jpg", Blob.copy(imageBuffer)))
    .onComplete(ar -> {
        // Do something
    });
```

3.1.1.6. Retrieval of automatically generated keys disabled by default

From Eclipse Vert.x 4.3.1 onward, in the Eclipse Vert.x Oracle reactive client, the retrieval of automatically generated keys is disabled by default. The Eclipse Vert.x Oracle reactive client does not typically need to retrieve automatically generated keys, because most applications do not rely on the **ROWID**.

This enhancement also facilitates queries such as **INSERT…SELECT**, which cannot run successfully when the retrieval of automatically generated keys is enabled.

3.1.1.7. Use of **io.vertx.core.shareddata.ClusterSerializable** interface

From Eclipse Vert.x 4.3.0 onward, Eclipse Vert.x supports the **io.vertx.core.shareddata.ClusterSerializable** interface for reading and writing objects to and from a buffer, when these objects are either read from an AsyncMap or decoded from an EventBus message body.

Earlier releases of Eclipse Vert.x supported the **io.vertx.core.shareddata.impl.ClusterSerializable** interface. However, because this interface was provided in an implementation package, it was considered potentially less reliable. The **io.vertx.core.shareddata.impl.ClusterSerializable** interface is now deprecated in Eclipse Vert.x 4.3.0 and made public.

3.1.1.8. Renaming of **requestsTagsProvider** option for Micrometer request metrics

From Eclipse Vert.x 4.3.0 onward, in the **MicrometerMetricsOptions** class, the **requestsTagsProvider** option for server request metrics is renamed **serverRequestTagsProvider**. This enhancement is required because a similar **clientRequestTagsProvider** option is also now available for client request metrics.

In earlier releases of Eclipse Vert.x, the **requestsTagsProvider** option used a getter and a setter, which were named **getRequestsTagsProvider** and **setRequestsTagsProvider**, respectively. In Eclipse Vert.x 4.3.0 and later versions, the getter and setter for the **serverRequestTagsProvider** option are renamed **getServerRequestTagsProvider** and **setServerRequestTagsProvider**.

3.1.1.9. OAuth2 OBO calls expect explicit **OAuth2Credentials** rather than **TokenCredentials**

From Eclipse Vert.x 4.3.0 onward, when OAuth2 authorization is configured in on-behalf-of (OBO) mode, OAuth2 requires that an **OAuth2Credentials** object is explicitly specified to authorize requests.

For example:

```
oauth2.authenticate(
  new Oauth2Credentials().setAssertion("head.body.signature").addScope("a").addScope("b"))
```

In earlier releases of Eclipse Vert.x, OAuth2 authorization in OBO mode allowed the use of **TokenCredentials**. However, because the flow is optional, to allow reuse of the same **OAuth2Credentials** object, the preceding **Oauth2Credentials** call now supersedes the following type of **TokenCredentials** call that was available in previous versions:

```
oauth2.authenticate(
  new TokenCredentials("head.body.signature").addScope("a").addScope("b"));
```

3.1.1.10. RoutingContext.fileUploads() method returns a list

From Eclipse Vert.x 4.3.0 onward, the **RoutingContext.fileUploads()** method returns a **List<FileUpload>** value. Storing file uploads in a list helps to preserve the order of the uploads.

For example:

```
List<FileUpload> uploads = ctx.fileUploads();
```

In earlier releases of Eclipse Vert.x, the **RoutingContext.fileUploads()** method returned a **Set<FileUpload>** value. However, storing file uploads in a set was not consistent with the World Wide Web Consortium (W3C) specification for form content types, because it did not preserve the correct order and users could not rely on the upload name to be a unique key. The preceding example now supersedes the following method declaration that was available in previous versions:

```
Set<FileUpload> uploads = ctx.fileUploads();
```

3.1.1.11. Single method to implement a sub router

From Eclipse Vert.x 4.3.0 onward, the **Route.subRouter(Router)** method is the only supported way to implement a sub router.

Earlier releases of Eclipse Vert.x supported two different methods for implementing sub routers:

- **Route.subRouter(Router)**
- **Router.mountSubRouter(String, Router)**

However, the behavior between these two methods was inconsistent, because the **Router.mountSubRouter** method allowed any path whereas the **Route.subRouter** method explicitly requires a wildcard asterisk (*) to represent the sub routing path. The **Router.mountSubRouter** method also delegated to the **Route.subRouter** method by appending the missing wildcard.

In Eclipse Vert.x 4.3.0 and later versions, the **Router.mountSubRouter** method is deprecated. Router objects must now also use the **Route.subRouter** method to implement sub routers. For example:

```
router.route("/eventbus/*").subRouter(otherRouter);
```

The preceding **router.route().subRouter()** call now supersedes the following type of **router.mountSubRouter()** call that was available in previous versions:

```
router.mountSubRouter("/eventbus", otherRouter);
```

NOTE

In previous releases, router objects could also use the **router.route().subRouter()** call as an alternative to using **router.mountSubRouter()**.

3.1.1.12. Caching of parsed request body across multiple handler invocations

From Eclipse Vert.x 4.3.0 onward, after a body handler parses the body of a web request, the body handler provides the body buffer to the routing context for the request. Caching the body buffer in the routing context means that multiple different handlers that want a decoded view of the request body can get the cached result without having to parse the body again. This enhancement also supports situations where the body content is of type **application/json**.

The **RoutingContext** class provides a new **body()** method that is used to get the request body as a specified type.

For example:

```
RoutingContext.body().asString()
RoutingContext.body().asString(String encoding)
RoutingContext.body().asJsonObject()
RoutingContext.body().asJsonArray()
RoutingContext.body().asJsonObject(int maxLength)
RoutingContext.body().asJsonArray(int maxLength)
RoutingContext.body().buffer()
```

The new **body()** getter also provides the following additional functionality:

```
// the length of the buffer (-1) for null buffers
RoutingContext.body().length()

// Converting to POJO
RoutingContext.body().asPOJO(Class<T> clazz)
RoutingContext.body().asPOJO(Class<T> clazz, int maxLength)
```

This enhancement provides the following advantages:

- The **body()** getter is never null, which helps to avoid any need to perform null checks.
- The request body needs to be parsed only once unless the base buffer changes. Any changes to the base buffer trigger another parse and the cached values are overridden at that point.

In earlier releases of Eclipse Vert.x, the **RoutingContext** class provided the following methods that are now deprecated in favor of using the **body()** method:

```
RoutingContext.getBodyAsString()
RoutingContext.getBodyAsString(String encoding)
RoutingContext.getBodyAsJson()
RoutingContext.getBodyAsJsonArray()
RoutingContext.getBodyAsJson(int maxLength)
RoutingContext.getBodyAsJsonArray(int maxLength)
RoutingContext.getBody()
```

3.1.1.13. Changes in EventBus notification defaults

From Eclipse Vert.x 4.3.0 onward, to avoid unnecessary traffic, notifications about changes in the Eclipse Vert.x circuit breaker state are disabled by default. To enable these notifications, call the **setNotificationAddress** method of the **CircuitBreakerOptions** object with a parameter that is not null.

For example:

```
CircuitBreakerOptions options = new CircuitBreakerOptions()
   .setNotificationAddress(CircuitBreakerOptions.DEFAULT_NOTIFICATION_ADDRESS);
```

When you enable notifications as shown in the preceding example, the default behavior is to send notifications to local consumers only. To send notifications on a cluster-wide basis, call the **setNotificationLocalOnly** method with a parameter of **false**.

For example:

```
CircuitBreakerOptions options = new CircuitBreakerOptions()
   .setNotificationAddress(CircuitBreakerOptions.DEFAULT_NOTIFICATION_ADDRESS)
   .setNotificationLocalOnly(false);
```

3.1.1.14. Changes in MySQL client batch execution

From Eclipse Vert.x 4.3.0 onward, the Eclipse Vert.x reactive SQL client supports pipelined queries and runs batch queries in pipelining mode by default. Pipelining means that requests are sent on the same connection without waiting for responses to previous requests.

In earlier releases of Eclipse Vert.x, because MySQL does not have native protocol support for batching, the SQL client ran batch queries by running prepared queries in a sequence, which the user could operate directly through API calls.

3.1.1.15. MongoDB enhancements for hints and hint strings

From Eclipse Vert.x 4.3.0 onward, Eclipse Vert.x includes the following MongoDB enhancements for hints and hint strings:

- **FindOptions** objects now support hints of type **JSONObject**. This supersedes the behavior in previous releases where **FindOptions** objects supported hints of type **String**.
- **BulkOperations** and **UpdateOptions** objects also now support hints of type **JSONObject**. The **BulkOperations** and **UpdateOptions** classes each provide **getHint()** and **setHint()** methods for this purpose.
- **BulkOperations**, **UpdateOptions**, and **FindOptions** objects also now support hint strings of type **String**. The **BulkOperations**, **UpdateOptions**, and **FindOptions** classes each provide **getHintString()** and **setHintString()** methods for this purpose.

3.1.1.16. Changes in building a schema

From Eclipse Vert.x 4.3.0 onward, when building a schema, use the JSON representation that the Eclipse Vert.x JSON schema provides. The JSON representation allows use of any validator.

For example:

```
JsonSchema schema = JsonSchema.of(dsl.toJson());
```

In earlier releases of Eclipse Vert.x, the **SchemaBuilder** class provided a **build()** method, which required use of a specific implementation of a validator. The **build()** method is deprecated in Eclipse Vert.x 4.3.0. The preceding JsonSchema example now supersedes the following type of **build()** method call that was available in previous versions:

```
Schema schema = dsl.build(parser);
```

3.1.2. New features introduced in earlier 4.x releases

The following new features were introduced in earlier 4.x releases.

3.1.2.1. Java 17 support

From Eclipse Vert.x 4.2.7 onward, Eclipse Vert.x is certified for use with Red Hat OpenJDK 17.

3.1.2.2. HTTP header validation in **RequestOptions**

From Eclipse Vert.x 4.2.4 onward, the **RequestOptions** method validates HTTP headers, and the request fails if a header name is invalid.

In earlier releases of Eclipse Vert.x, the **HTTPClientRequest** validated HTTP headers, because the **RequestOptions** method used a Multimap implementation that did not validate header names.

3.1.2.3. Use **simple** as the default locale for collation

From Eclipse Vert.x 4.2.4 onward, the **simple** locale is used as the default locale for MongoDB collation.

Eclipse Vert.x 4.2.3 introduced support for the collation options to support language-specific rules for comparing strings. In Eclipse Vert.x 4.2.3, the platform default was used as the default locale. However, because the platform default is not a constant value, it could lead to failures on systems that use a locale that is not supported by MongoDB. For example, **Locale.FR** would work successfully, but **Locale.FR_FR** would not be supported

3.1.2.4. **StaticHandler** file system configuration changes

From Eclipse Vert.x 4.2.4 onward, the **StaticHandler** configuration properties for the webroot directory and file system access are defined in the **StaticHandler** factory constructor call.

For example, the following constructor call defines a webroot directory, **static/resources**, and relative file system access:

```
StaticHandler.create(FileSystemAccess.RELATIVE, "static/resources");
```

For example, the following constructor call defines a webroot directory, **/home/paulo/Public**, and root file system access:

```
StaticHandler.create(FileSystemAccess.ROOT, "/home/paulo/Public");
```

In earlier releases of Eclipse Vert.x, the **allowRootFileSystemAccess** and **webroot** properties were defined by using setters. However, these property values were not final, which could lead to invalid static configuration. In Eclipse Vert.x 4.2.4, the preceding constructor call now supersedes the following setter declarations:

```
StaticHandler.create()
  .setAllowRootFileSystemAccess(true)
  .setWebRoot("/home/paulo/Public");
```

NOTE

The **StaticHandler.create()** method still uses default values of **RELATIVE** and **webroot** as in earlier releases.

3.1.2.5. Random server port sharing within a verticle

From Eclipse Vert.x 4.2.0 onward, two distinct HTTP servers that are bound with a negative port number, such as **-1**, share the same random port within the instances of a specific verticle deployment. This means that multiple HTTP servers bound with port **-1** will share the same random port. Similarly, multiple HTTP servers bound with port **-2** will share the same random port, and so on. This port sharing behavior that is based on negative port numbers is independent of the verticle, because it allows different HTTP servers to have a different random port.

In earlier releases of Eclipse Vert.x, random server port sharing was based on two HTTP servers bound with port **0**. However, this prevented the same verticle from binding two HTTP servers with different random ports within the instances of the same verticle.

3.1.2.6. HTTP Server cookie changes

Eclipse Vert.x 4.2.0 includes a new method, **Set<Cookie> cookies()**, that enables getting all cookies.

In earlier releases of Vert.x, the **HttpServerRequest** and **HttpServerResponse** interfaces used the following method that is now deprecated in Eclipse Vert.x 4.2.0:

```
Map<String, Cookie> cookieMap()
```

The RFC 6265 - HTTP State Management Mechanism specification states that each cookie is uniquely identified based on the tuple **<name, domain, path>**. However, the **Map<String, Cookie> cookieMap()** method used in earlier releases of Eclipse Vert.x wrongly assumed that cookies could be identified based on their name only. This meant that when multiple cookies shared the same name, the map held the last cookie to be parsed, and any previously parsed value was silently overwritten.

3.1.2.7. Context management with GraphQLContext object

Eclipse Vert.x 4.2.0 supports version 17 of GraphQL Java, which is the Java server implementation of the GraphQL query language. With GraphQL Java 17, the **GraphQLContext** object is now the standard for sharing contextual data between components of a GraphQL Java application.

Eclipse Vert.x 4.2.0 introduces the following new mechanism to configure GraphQL execution:

```
GraphQLHandler handler = GraphQLHandler.create(graphQL).beforeExecute(builderWithContext ->
{
  DataLoader<String, Link> linkDataLoader = DataLoaderFactory.newDataLoader(linksBatchLoader);
  DataLoaderRegistry dataLoaderRegistry = new DataLoaderRegistry().register("link",
linkDataLoader);
  builderWithContext.builder().dataLoaderRegistry(dataLoaderRegistry);
});
```

In earlier releases of Eclipse Vert.x, the following hooks were used in Vert.x Web GraphQL handlers to configure a data loader. The following hooks are now deprecated in Eclipse Vert.x 4.2.0.

```
GraphQLHandler handler = GraphQLHandler.create(graphQL).dataLoaderRegistry(rc -> {
  DataLoader<String, Link> linkDataLoader = DataLoader.newDataLoader(linksBatchLoader);
  return new DataLoaderRegistry().register("link", linkDataLoader);
});
```

3.1.2.8. OpenJDK11 OpenShift images support multiple architectures

OpenJ9 images for IBM Z and IBM Power Systems have been deprecated. The following OpenJDK11 image has been updated to support multiple architectures:

- **ubi8/openjdk-11**

You can use the OpenJDK11 image with the following architectures:

- x86 (x86_64)
- s390x (IBM Z)
- ppc64le (IBM Power Systems)

3.1.2.9. Support Eclipse Vert.x Runtime on FIPS enabled Red Hat Enterprise Linux (RHEL) system

Red Hat build of Eclipse Vert.x runs on a FIPS enabled RHEL system and uses FIPS certified libraries provided by RHEL.

3.1.2.10. HTTP client redirect handler propagates headers

From Eclipse Vert.x 4.1.0 onward, if there are headers in an HTTP redirect, then the HTTP client redirect handler propagates the headers to the next request. This change enables the redirect handler to have more control over the entire redirected request.

In earlier releases of Eclipse Vert.x, where there were redirected requests with headers, the HTTP client would handle the headers after the redirect.

The following example shows you how redirects are handled in Eclipse Vert.x 4.1.0:

```
RequestOptions options = new RequestOptions();
options.setMethod(HttpMethod.GET);
options.setHost(uri.getHost());
options.setPort(port);
options.setSsl(ssl);
options.setURI(requestURI);

// From 4.1.0 propagate headers
options.setHeaders(resp.request().headers());
options.removeHeader(CONTENT_LENGTH);
```

3.1.2.11. Upgrade to Infinispan 12

In Eclipse Vert.x 4.1.0, the Infinispan cluster manager has been updated and is based on Infinispan 12.

Infinispan 11 had a bug, which did not allow storing of byte arrays in a multimap cache. As a wordaround, the Eclipse Vert.x cluster manager had to use an internal Infinispan class, **WrappedBytes**, to store eventbus subscription data. This issue has been fixed in Infinispan 12.

3.1.2.12. JSON configuration takes precedence over connection string options in MongoDB Client

In Eclipse Vert.x 4.1.0, the JSON configuration options are applied even if a **connection_string** option is available.

The following configuration options are now applied:

```
{
   mongo:{
      db_name: "mydb"
      connection_string: "mongodb://localhost:27017"
      maxPoolSize: 10
      minPoolSize: 3
   }
}
```

In earlier releases of Eclipse Vert.x, the JSON configuration options were ignored when connection string was available. For example, consider the previous example. In earlier releases of Eclipse Vert.x, **db_name**, **maxPoolSize**, and **minPoolSize** options would have been ignored.

3.1.2.13. Removed the deprecated JWT options methods

From Eclipse Vert.x 4.0 onward, the JWT and OAuth2 handlers are used to handle scopes.

From Eclipse Vert.x 4.1.0 onward, **JWTOptions.setScopes(List<String>)**, **JWTOptions.addScope(String)** and **JWTOptions.withScopeDelimiter(String)** methods have been removed. These methods did not comply with the specification.

The following example shows you how to handle scopes in Eclipse Vert.x 4.1.0.

```
// before 4.1.0
JWTAuthOptions authConfig = new JWTAuthOptions()
  .setJWTOptions(new JWTOptions()
    .addScope("a")
    .addScope("b")
    .withScopeDelimiter(" ")));

JWTAuth authProvider = JWTAuth.create(vertx, authConfig);

router.route("/protected/*").handler(JWTAuthHandler.create(authProvider));

// in 4.1.0
JWTAuth authProvider = JWTAuth.create(vertx, new JWTAuthOptions());

router.route("/protected/*").handler(
  JWTAuthHandler.create(authProvider)
    .addScope("a")
    .addScope("b")
    .withScopeDelimiter(" "));
```

3.1.2.14. Deprecated the custom formatter method that accepts a function

From Eclipse Vert.x 4.1.0, **LoggerHandler.customFormatter(Function)** method has been deprecated. The function takes as input an **HttpServerRequest** and returns a formatted log string. Because the output is a string, it is not possible to access the context.

Use the new method **LoggerHandler customFormatter(LoggerFormatter formatter)** instead. The method takes as input a custom formatter that gives access to the context.

3.1.2.15. New exception to handle HTTP failures

From Eclipse Vert.x 4.1.0, a new exception class **io.vertx.ext.web.handler.HttpException** is available that can be used to handle HTTP failures. You can use the exception to specify custom status codes other than 500. For example, new **HttpException(401, "Forbidden")** indicates that the requests that are forbidden should return status code 401.

3.1.2.16. Support for RxJava 3

From Eclipse Vert.x 4.1.0, RxJava 3 is supported.

- A new rxified API is available in the **io.vertx.rxjava3** package.
- Integration with Eclipse Vert.x JUnit5 is provided by the **vertx-junit5-rx-java3** binding.

3.1.2.17. Context server interceptor binds all types of data and is more secure

From Eclipse Vert.x 4.0.3, the **ContextServerInterceptor.bind()** method binds all types of data to the context. The method is more secure now as it does not expose the storage details.

In releases prior to Eclipse Vert.x 4.0.3, the method used to bind only 'String' data type to context. It also exposed the storage details.

To use the updated **ContextServerInterceptor.bind()** method, you must update your application.

The following example shows the code in releases prior to Eclipse Vert.x 4.0.3.

```
// Example code from previous releases

class X extends ContextServerInterceptor {
  @Override
  public void bind(Metadata metadata, ConcurrentMap<String, String> context) {
```

The following example shows the replacing code fpr Eclipse Vert.x 4.0.3 release.

```
// Replacing code for Eclipse Vert.x 4.0.3 release

class X extends ContextServerInterceptor {
  @Override
  public void bind(Metadata metadata) {
```

3.1.2.18. Matching of ending slash (/) in route paths that end with wildcard character is no longer required

In releases prior to Eclipse Vert.x 4.0.3, if routes were defined with a path ending in slash and a wildcard /*, the routes would be called only if the matching request also included the ending slash /. This rule caused problems when the wildcard was empty.

From Eclipse Vert.x 4.0.3 onward, this rule is no longer applied. You can create routes whose paths end in a slash (/). However, it is not mandatory to specify the slash in the request URLs.

Also, you can create and use request URLs to call routes that end with wildcards in their path instead of slash (/). For example, routes with wildcard can be defined as **/foo/***. Here the route has to match an open wildcard at the end of the path. The request URL can be **/foo**.

The table shows the behavior in Eclipse Vert.x 4.0.3 and previous releases when you send a request URL **/foo/***. You can see that the ending slash is optional in Eclipse Vert.x 4.0.3 and request matches the route.

Route	Eclipse Vert.x 4.0.3	Releases prior to Eclipse Vert.x 4.0.3
/foo	Match	No Match
/foofighters	No Match	No Match
/foo/	Match	Match
/foo/bar	Match	Match

3.1.2.19. Removed the **autoRegistrationOfImporters** attribute from service discovery options

The **autoRegistrationOfImporters** attribute has been removed from service discovery options.

3.1.2.20. Authenticate method in authentication provider class updated to support **token** as input credentials

In releases prior to Eclipse Vert.x 4.0.3, the AuthenticationProvider.authenticate() method would incorrectly take **jwt: someValue** as input credentials.

From Eclipse Vert.x 4.0.3, the AuthenticationProvider.authenticate() method has been updated and takes **token: someValue** as input credentials. This change ensures that both JSON and typed APIs are consistent and can be used interchangeably.

The following code shows the implementation for the authenticate method in releases prior to Eclipse Vert.x 4.0.3.

```
new JsonObject().put("jwt", "token...");
```

The following code shows the implementation for the authenticate method in Eclipse Vert.x 4.0.3 release.

```
new JsonObject().put("token", "token...");
```

3.1.2.21. Get method for PEM keys returns **Buffer** instead of a **String**

The **PubSecKeyOptions.getBuffer()** method returns the PEM or secret key buffer. In releases prior to Eclipse Vert.x 4.0.2, the key buffer was stored and returned as a **String**. However, it is recommended to save secrets as a **Buffer**. From Eclipse Vert.x 4.0.2 onward, the method stores and returns the key buffer as a **Buffer**. This change improves the security and handling of secrets.

The **PubSecKeyOptions.setBuffer()** method continues to accept a **String** argument. In the set method, an overload for Buffer has been added to safely handle non ASCII secret materials. This change does not require any change to the existing code.

3.1.2.22. Kubernetes service importer is no longer registered automatically

From Eclipse Vert.x 4, the **KubernetesServiceImporter** discovery bridge is no longer registered automatically. Even if you have added the bridge in the classpath of your Maven project, it will not be automatically registered.

You must manually register the bridge after creating the **ServiceDiscovery** instance.

3.1.2.23. Use future methods for asynchronous operations

Eclipse Vert.x 4 uses futures for asynchronous operations. Every callback method has a corresponding future method.

Futures can be used to compose asynchronous operations. When you use futures, the error handling is better. Therefore, it is recommended to use a combination of callback and futures in your applications.

3.1.2.24. No dependency on the Jackson Databind library

In Eclipse Vert.x 4, Jackson Databind is an optional Maven dependency. If you want to use this dependency, you must explicitly add it in the classpath. For example, if you are object mapping JSON, then you must explicitly add the dependency.

3.1.2.25. Handling deprecations and removals

In Eclipse Vert.x 4, new enhanced features have been provided. The old features and functions have been deprecated or removed in Eclipse Vert.x 4. Before you migrate your applications to Eclipse Vert.x 4, check for deprecations and removals.

The Java compiler generates warnings when deprecated APIs are used. You can use the compiler to check for deprecated methods while migrating applications to Eclipse Vert.x 4.

3.1.2.26. Support for distributed tracing

Eclipse Vert.x 4 supports distributed tracing. You can use tracing to monitor microservices and identify performance issues.

Eclipse Vert.x 4 integrates with OpenTracing system.

The following Eclipse Vert.x components can log traces:

- HTTP server and HTTP client
- Eclipse Vert.x SQL client
- Eclipse Vert.x Kafka client

IMPORTANT

Tracing is available as Technology Preview. 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.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

3.1.2.27. New publishing location for EventBus JavaScript Client

In Eclipse Vert.x 4, the EventBus JavaScript client, **vertx-web-client.js** is not published as a Red Hat artifact in the Maven repository.

The client is published in the npm repository. You can access the client from the following location: @vertx/eventbus-bridge-client.js

3.1.2.28. Deploy Eclipse Vert.x applications using OpenShift Maven plugin

Use the OpenShift Maven plugin to deploy your Eclipse Vert.x applications on OpenShift. The Fabric8 Maven plugin is no longer supported. For more information, see the section migrating from Fabric8 Maven Plugin to Eclipse JKube.

3.1.2.29. Eclipse Vert.x metering labels for OpenShift

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

NOTE

- Do not add metering labels to any pods that an operator or a template deploys and manages.
- You can apply labels to pods using the Metering Operator on OpenShift Container Platform version 4.8 and earlier. From version 4.9 onward, the Metering Operator is no longer available without a direct replacement.

Eclipse Vert.x should use the following metering labels:

- **com.company: Red_Hat**
- **rht.prod_name: Red_Hat_Runtimes**
- **rht.prod_ver: 2023-Q1**
- **rht.comp: Vert.x**
- **rht.comp_ver: 4.3.7**
- **rht.subcomp: <leave_blank>**
- **rht.subcomp_t: application**

Additional resources

Additional resources

- Configuring and using Metering in OpenShift Container Platform

3.1.2.30. Support for OpenJDK 8 and OpenJDK 11 RHEL 8 Universal Base Images (UBI8)

Eclipse Vert.x introduces support for building and deploying Eclipse Vert.x applications to OpenShift with OCI-compliant Universal Base Images for Red Hat OpenJDK 8 and Red Hat OpenJDK 11 on RHEL 8.

The RHEL 8 OpenJDK Universal Base Images replace the RHEL 8 OpenJDK builder images. The RHEL 8 OpenJDK base images are no longer supported for use with Eclipse Vert.x.

3.2. DEPRECATED FEATURES

This section lists the functionalities deprecated or removed in this release.

3.2.1. Features deprecated in the 4.3 release

The following functionalities are deprecated in the 4.3 release.

- Eclipse Vert.x Core

Removed elements	Replacing elements
io.vertx.core.shareddata.impl.ClusterSerializable	**io.vertx.core.shareddata.ClusterSerializable**

- Eclipse Vert.x Micrometer Metrics

Deprecated methods	Replacing methods
io.vertx.micrometer.MicrometerMetricsOptions.getRequestsTagsProvider()	**io.vertx.micrometer.MicrometerMetricsOptions.getServerRequestsTagsProvider()**
io.vertx.micrometer.MicrometerMetricsOptions.setRequestsTagsProvider()	**io.vertx.micrometer.MicrometerMetricsOptions.setServerRequestsTagsProvider()**
io.vertx.micrometer.VertxInfluxDbOptions.getNumThreads()	No replacing method
io.vertx.micrometer.VertxInfluxDbOptions.setNumThreads()	No replacing method

- Eclipse Vert.x Web

Deprecated methods	Replacing methods
Router.mountSubRouter(String, Router)	**Router.route(String).subRouter(Router)**
RoutingContext.getBodyAsString()	**RoutingContext.body().asString()**

Deprecated methods	Replacing methods
RoutingContext.getBodyAsString(String encoding)	**RoutingContext.body().asString(String encoding)**
RoutingContext.getBodyAsJson()	**RoutingContext.body().asJsonObject()**
RoutingContext.getBodyAsJsonArray()	**RoutingContext.body().asJsonArray()**
RoutingContext.getBodyAsJson(int maxLength)	**RoutingContext.body().asJsonObject(int maxLength)**
RoutingContext.getBodyAsJsonArray(int maxLength)	**RoutingContext.body().asJsonArray(int maxLength)**
RoutingContext.getBody()	**RoutingContext.body().buffer()**
RouterBuilder.bodyHandler()	**RouterBuilder.rootHandler()**

- **SchemaBuilder**

Removed methods	Replacing methods
build()	Use the JSON representation that the Eclipse Vert.x Json Schema provides. For example: **JsonSchema schema = JsonSchema.of(dsl.toJson());**

3.2.2. Features deprecated in earlier 4.x releases

The following functionalities were deprecated or removed in earlier 4.x releases.

- **HttpServerOptions**

Removed methods	Replacing methods
getMaxWebsocketFrameSize()	**getMaxWebSocketFrameSize()**
setMaxWebsocketFrameSize()	**setMaxWebSocketFrameSize()**
getMaxWebsocketMessageSize()	**getMaxWebSocketMessageSize()**
setMaxWebsocketMessageSize()	**setMaxWebSocketMessageSize()**
getPerFrameWebsocketCompressionSupported()	**getPerFrameWebSocketCompressionSupported()**

Removed methods	Replacing methods
setPerFrameWebsocketCompressionSupported()	setPerFrameWebSocketCompressionSupported()
getPerMessageWebsocketCompressionSupported()	getPerMessageWebSocketCompressionSupported()
setPerMessageWebsocketCompressionSupported()	setPerMessageWebSocketCompressionSupported()
getWebsocketAllowServerNoContext()	getWebSocketAllowServerNoContext()
setWebsocketAllowServerNoContext()	setWebSocketAllowServerNoContext()
getWebsocketCompressionLevel()	getWebSocketCompressionLevel()
setWebsocketCompressionLevel()	setWebSocketCompressionLevel()
getWebsocketPreferredClientNoContext()	getWebSocketPreferredClientNoContext()
setWebsocketPreferredClientNoContext()	setWebSocketPreferredClientNoContext()
getWebsocketSubProtocols()	getWebSocketSubProtocols()
setWebsocketSubProtocols()	setWebSocketSubProtocols()

- Eclipse Vert.x Web

Removed elements	Replacing elements
io.vertx.ext.web.Cookie	io.vertx.core.http.Cookie
io.vertx.ext.web.handler.CookieHandler	io.vertx.core.http.Cookie
io.vertx.ext.web.Locale	io.vertx.ext.web.LanguageHeader
RoutingContext.acceptableLocales()	RoutingContext.acceptableLanguages()
StaticHandler.create(String, ClassLoader)	---
SessionHandler.setAuthProvider(AuthProvider)	SessionHandler.addAuthProvider()

Removed elements	Replacing elements
HandlebarsTemplateEngine.getHandlebars()HandlebarsTemplateEngine.getResolvers()HandlebarsTemplateEngine.setResolvers()JadeTemplateEngine.getJadeConfiguration()ThymeleafTemplateEngine.getThymeleafTemplateEngine()ThymeleafTemplateEngine.setMode()	**TemplateEngine.unwrap()**

- Messaging

Removed methods	Replacing methods
MessageProducer<T>.send(T)	**MessageProducer<T>.write(T)**
MessageProducer.send(T,Handler)	**EventBus.request(String,Object,Handler)**

- EventBus

Removed methods	Replacing methods
EventBus.send(…, Handler<AsyncResult<Message<T>>>)Message.reply(…, Handler<AsyncResult<Message<T>>>)	**replyAndRequest**

- Handlers

Removed methods	Replacing methods
Future<T>.setHandler()	**Future<T>.onComplete()Future<T>.onSuccess()Future<T>.onFailure()**
HttpClientRequest.connectionHandler()	**HttpClient.connectionHandler()**

- JSON

Removed Fields/Methods	New methods
Json.mapper() field	**DatabindCodec.mapper()**
Json.prettyMapper() field	**DatabindCodec.prettyMapper()**
Json.decodeValue(Buffer, TypeReference<T>)	**JacksonCodec.decodeValue(Buffer, TypeReference)**

Removed Fields/Methods	New methods
Json.decodeValue(String, TypeReference<T>)	**JacksonCodec.decodeValue(String, TypeReference)**

- JUnit5

Deprecated methods	New methods
VertxTestContext.succeeding()	**VertxTestContext.succeedingThenComplete()**
VertxTestContext.failing()	**VertxTestContext.failingThenComplete()**

- Reactive Extensions (Rx)

Deprecated methods	New methods
WriteStreamSubscriber.onComplete()	**WriteStreamSubscriber.onWriteStreamEnd()WriteStreamSubscriber.onWriteStreamError()**

- Circuit breaker

Removed methods	Replacing methods
CircuitBreaker.executeCommand()	**CircuitBreaker.execute()**
CircuitBreaker.executeCommandWithFallback()	**CircuitBreaker.executeWithFallback()**

- MQTT

Removed methods	Replacing methods
MqttWill.willMessage()	**MqttWill.getWillMessage()**
MqttWill.willTopic()	**MqttWill.getWillTopic()**
MqttWill.willQos()	**MqttWill.getWillQos()**
MqttAuth.username()	**MqttAuth.getUsername()**
MqttAuth.password()	**MqttAuth.getPassword()**
MqttClientOptions.setKeepAliveTimeSeconds()	**MqttClientOptions.setKeepAliveInterval()**

- AMQP client

Removed methods	Replacing methods
AmqpClient.createReceiver(String address, Handler<AmqpMessage> messageHandler, …)	**AmqpClient createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)**
AmqpConnection createReceiver(…, Handler<AsyncResult<AmqpReceiver>> completionHandler)	**AmqpConnection createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)**
AmqpConnection createReceiver(.., Handler<AmqpMessage> messageHandler, Handler<AsyncResult<AmqpReceiver>> completionHandler)	**AmqpConnection createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)**

- Authentication and authorization

Removed elements	Replacing elements
OAuth2Options.isUseBasicAuthorizationHeader()	No replacing method
OAuth2Options.setUseBasicAuthorizationHeader()	No replacing method
OAuth2Options.getClientSecretParameterName()	No replacing method
OAuth2Options.setClientSecretParameterName()	No replacing method
OAuth2Auth.createKeycloak()	**KeycloakAuth.create(vertx, JsonObject) ()**
OAuth2Auth.create(Vertx, OAuth2FlowType, OAuth2ClientOptions) ()	**OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))**
OAuth2Auth.create(Vertx, OAuth2FlowType)	**OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))**
User.isAuthorised()	**User.isAuthorized()**
AccessToken.refreshToken()	**AccessToken.opaqueRefreshToken()**

Removed elements	Replacing elements
io.vertx.ext.auth.jwt.JWTOptions data object	**io.vertx.ext.jwt.JWTOptions** data object
SecretOptions class	**PubSecKeyOptions** class

Deprecated methods	Replacing methods
OAuth2Auth.decodeToken()	**AuthProvider.authenticate()**
OAuth2Auth.introspectToken()	**AuthProvider.authenticate()**
OAuth2Auth.getFlowType()	No replacing method
OAuth2Auth.loadJWK()	**OAuth2Auth.jwkSet()**
Oauth2ClientOptions.isUseAuthorizationHeader()	No replacing method

Deprecated class	Replacing class
AbstractUser	Create user objects using the ` User.create(JsonObject)` method.
AuthOptions	No replacing class
JDBCAuthOptions	**JDBCAuthenticationOptions** for authentication and **JDBCAuthorizationOptions** for authorization
JDBCHashStrategy	No replacing class
OAuth2RBAC	**AuthorizationProvider**
Oauth2Response	Recommended to use **WebClient** class
KeycloakHelper	No replacing class

- Service discovery

Removed methods	Replacing methods
ServiceDiscovery.create(…, Handler<ServiceDiscovery> completionHandler)	**ServiceDiscovery.create(Vertx)**
ServiceDiscovery.create(…, Handler<ServiceDiscovery> completionHandler)	**ServiceDiscovery.create(Vertx, ServiceDiscoveryOptions)**

- Eclipse Vert.x configuration

Removed methods	Replacing methods
ConfigRetriever.getConfigAsFuture()	**retriever.getConfig()**

- MongoDB client

Removed methods	Replacing methods
MongoClient.update()	**MongoClient.updateCollection()**
MongoClient.updateWithOptions()	**MongoClient.updateCollectionWithOptions()**
MongoClient.replace()	**MongoClient.replaceDocuments()**
MongoClient.replaceWithOptions()	**MongoClient.replaceDocumentsWithOptions()**
MongoClient.remove()	**MongoClient.removeDocuments()**
MongoClient.removeWithOptions()	**MongoClient.removeDocumentsWithOptions()**
MongoClient.removeOne()	**MongoClient.removeDocument()**
MongoClient.removeOneWithOptions	**MongoClient.removeDocumentsWithOptions()**

- Clients with no shared data sources

Deprecated Methods	New Methods
MongoClient.createNonShared()	**MongoClient.create()**

Deprecated Methods	New Methods
JDBCClient.createNonShared()	**wJDBCClient.create()**
CassandraClient.createNonShared()	**CassandraClient.create()**
MailClient.createNonShared()	**MailClient.create()**

- Hook methods

Removed Methods	New Methods
Context.addCloseHook()	No replacing method
Context.removeCloseHook()	No replacing method

- Clone methods

Removed Methods	New Methods
KeyCertOptions.clone()	**KeyCertOptions.copy()**
TrustOptions.clone()	**TrustOptions.copy()**
SSLEngineOptions.clone()	**SSLEngineOptions.copy()**

- **VertxOptions**

Removed Methods	New Methods
VertxOptions.equals()	No replacing method
VertxOptions.hashCode()	No replacing method
VertxOptions.fileResolverCachingEnabled()	**FileSystemOptions.isFileCachingEnabled()**

- Pooled buffer

Removed Methods	New Methods
TCPSSLOptions.isUsePooledBuffers()	No replacing method
TCPSSLOptions.setUsePooledBuffers()	No replacing method

CHAPTER 4. RELEASE COMPONENTS

4.1. SUPPORTED ARTIFACTS INTRODUCED IN THIS RELEASE

No artifacts have been moved from Technology Preview to fully supported in this release.

4.2. TECHNOLOGY PREVIEW ARTIFACTS INTRODUCED IN THIS RELEASE

This section describes the Technology Preview artifacts introduced in this release.

4.2.1. Technology Preview artifacts introduced in the 4.3 release

The following artifacts are provided as Technology Preview in the 4.3 release.

- **vertx-grpc-client**
  The Eclipse Vert.x gRPC Client is a new Google Remote Procedure Call (gRPC) client that relies on the Eclipse Vert.x HTTP client. The Eclipse Vert.x gRPC Client provides two alternative ways to interact with a server:
  - A gRPC request-and-response-oriented API that does not require a generated stub
  - A generated stub with a gRPC channel

  NOTE

  The Eclipse Vert.x gRPC Client supersedes the integrated Netty-based gRPC client.

- **vertx-grpc-server**
  The Eclipse Vert.x gRPC Server is a new Google Remote Procedure Call (gRPC) server that relies on the Eclipse Vert.x HTTP server. The Eclipse Vert.x gRPC Server provides two alternative ways to interact with a client:
  - A gRPC request-and-response-oriented API that does not require a generated stub
  - A generated stub with a service bridge

  NOTE

  The Eclipse Vert.x gRPC Server supersedes the integrated Netty-based gRPC server.

- **vertx-grpc-common**
  The Eclipse Vert.x gRPC Common artifact provides common functionality that the Eclipse Vert.x gRPC Client and the Eclipse Vert.x gRPC Server both use.

- **vertx-grpc-aggregator**
  The Eclipse Vert.x gRPC Aggregator consists of a Project Object Model (POM) file. The Eclipse Vert.x gRPC Aggregator does not provide any additional functionality.

4.2.2. Technology Preview artifacts introduced in earlier 4.x releases

The following artifacts that were available as Technology Preview from previous 4.x releases continue to be Technology Preview in this release.

- **vertx-auth-otp**
  The Eclipse Vert.x OTP Auth provider is an implementation of the **AuthenticationProvider** interface that uses one-time passwords to perform authentication. The Eclipse Vert.x OTP Auth provider supports the Google Authenticator. You can use any convenient library to create the quick response (QR) with a key. You can also transfer the key in base32 format.

- **vertx-oracle-client**
  The Eclipse Vert.x reactive Oracle client is a client for the Oracle server. It is an API that helps in database scalability and has low overhead. Because the API is reactive and non-blocking, you can handle multiple database connections with a single thread.

  NOTE

  The Eclipse Vert.x reactive Oracle client requires that you use the Oracle JDBC driver. Red Hat does not provide support for the Oracle JDBC driver.

  The Eclipse Vert.x reactive Oracle client requires that you use JDK 11 or JDK 17.

- **vertx-http-proxy**
  The Eclipse Vert.x HTTP proxy is a reverse proxy. Using this module you can easily create proxies. The proxy server can also dynamically resolve the DNS queries from origin server.

- **vertx-web-proxy**
  The Eclipse Vert.x web proxy enables you to mount an Eclipse Vert.x HTTP proxy in an Eclipse Vert.x web router.

- **vertx-opentelemetry**
  Open Telemetry tracing is supported. You can use Open Telemetry for HTTP and event bus tracing.

4.3. ARTIFACTS REMOVED IN THIS RELEASE

No artifacts are removed in this release.

4.4. ARTIFACTS DEPRECATED IN THIS RELEASE

No artifacts are marked as deprecated in this release.

CHAPTER 5. FIXED ISSUES

This Eclipse Vert.x release incorporates all bugfixes from community release of version 4.3.7. Issues resolved in the community release are listed in the Eclipse Vert.x 4.3.7 wiki page.

5.1. FIXED ISSUES IN THE 4.3 RELEASE

This section describes issues that are fixed in Eclipse Vert.x 4.3.

5.1.1. Threads blocked with **io.vertx.coreVertxException** when using gRPC for client and server communication

In Eclipse Vert.x 4.3.4 or earlier, Google Remote Procedure Call (gRPC) communication between a client and a server blocked threads and resulted in an **io.vertx.core.VertxException** error. This issue occurred if the number of available event-loop threads was insufficient, which caused the Eclipse Vert.x gRPC Server (**vertx-grpc-server**) or the Eclipse Vert.x gRPC Client (**vertx-grpc-client**) to self-deadlock.

This issue is resolved in the Eclipse Vert.x 4.3.5 release. The internal context for the SSL initialization now uses a worker context rather than an event-loop context.

5.1.2. JDBCClient error when searching table data by **ROWID**

In Eclipse Vert.x 4.2, the **vertx-jdbc-client** with the Oracle JDBC driver threw a **java.sql.SQLException** when attempting to retrieve table data by using the **ROWID**. This issue occurred because, in Eclipse Vert.x 4.2, the **ROWID** was available as an array of bytes. In earlier releases of Eclipse Vert.x, the **ROWID** was available as a string type.

This issue is resolved in the Eclipse Vert.x 4.3.1 release. Eclipse Vert.x 4.2 performed custom type casting based on plain Java types. Even though this behavior typically produces correct results, it might incorrectly identify special database types. Because custom type casting is not a built-in feature of modern JDBC drivers, the JDBC client now relies on the driver to perform vendor-specific casts that are more suitable than the old Eclipse Vert.x heuristics.

5.1.3. JDBCPool error when parsing **ROWID**

In Eclipse Vert.x 4.2, the JDBC pool threw a **java.lang.UnsupportedOperationException** when attempting to parse the **ROWID**. This issue occurred because, in Eclipse Vert.x 4.2, the **ROWID** could not be parsed directly as an array of bytes. In earlier releases of {VertX), the **ROWID** could be parsed as a string type.

This issue is resolved in the Eclipse Vert.x 4.3.1 release based on the same solution described in the preceding section.

5.1.4. Unexpected results in stored procedure calls when using a PostgreSQL JDBC driver 9.0 or later

In Eclipse Vert.x 4.2, the **vertx-jdbc-client** with a PostgreSQL JDBC driver 9.0 or later produced unexpected results in stored procedure calls. This issue occurred because, in Eclipse Vert.x 4.2, the **vertx-jdbc-client** did not support the explicit SQL type information that modern PostgreSQL database drivers and servers require when executing callable statements.

This issue is resolved in the Eclipse Vert.x 4.3.1 release. Callable **ResultSet** metadata is now extracted from all sources involved in the query, from the first database response, as well as the individual result sets that form part of the response. The complete information allows the JDBC client to correctly identify the type of data in a column and to perform the correct casts.

5.2. FIXED ISSUES IN EARLIER 4.X RELEASES

This section describes issues that were fixed in earlier Eclipse Vert.x 4.x releases.

5.2.1. Google Guava classes included in GraphQL builds

In the Eclipse Vert.x 4.0.0 and 4.0.2 releases, the **vertx-web-graphql** dependency was not usable. This was because an incomplete build of GraphQL Java with version 16.1.0.redhat-00001 was used. In the incomplete GraphQL build, the Guava classes were missing.

This issue is resolved in the Eclipse Vert.x 4.0.3 release. The release includes the GraphQL Java 16.1.0.redhat-00002 version, which is a complete build with Guava classes. These Guava classes are shaded into the jar.

5.2.2. vertx-opentracing available in Eclipse Vert.x builds

The **vertx-opentracing** dependency was introduced as a Technical Preview feature in Eclipse Vert.x 4.0.0. However, the dependency was not available in Eclipse Vert.x 4.0.0 and 4.0.2 releases.

This issue is resolved in the Eclipse Vert.x 4.0.3 release. The release includes the **vertx-opentracing** dependency.

CHAPTER 6. KNOWN ISSUES

6.1. KNOWN ISSUES IN THE 4.3 RELEASE

No issues are known to affect this release.

6.2. KNOWN ISSUES IN EARLIER 4.X RELEASES

This section describes known issues from earlier Eclipse Vert.x 4.x releases.

6.2.1. Tokens issued by RH-SSO result in OAuth2 validation failures after migrating to Eclipse Vert.x 4.2

Description

If you are using Red Hat Single Sign-On (RH-SSO) as an identity provider, tokens that were valid in earlier releases of Eclipse Vert.x will fail validation checks when you migrate to Eclipse Vert.x 4.2.

Cause

In Eclipse Vert.x 4.2, OAuth2 authentication provides stricter security validation than earlier releases of Eclipse Vert.x.

Workaround

By default, RH-SSO issues tokens with the **account** audience rather than the client ID. If you are using RH-SSO as an identity provider, after you migrate to Eclipse Vert.x 4.2, you must explicitly add the **account** audience and the client ID to the JWTOptions configuration to ensure that tokens are successfully validated.

6.2.2. Compilation error when using **vertx-oracle-client** with JDK 8

Description

In Eclipse Vert.x 4.2, the **vertx-oracle-client** throws a **bad class file** compilation error when you use OpenJDK 8.

Cause

In Eclipse Vert.x 4.2, the **vertx-oracle-client** is designed to work with OpenJDK 11 or later.

Workaround

Use OpenJDK 11 or OpenJDK 17.

6.2.3. **KubernetesServiceImporter()** cannot be directly registered in Eclipse Vert.x Reactive Extensions (Rx)

Description

You cannot directly register **KubernetesServiceImporter()** with the Reactive Extensions (Rx) for Eclipse Vert.x.

Cause

Service importers do not have a generated RxJava 2 implementation.

Workaround

You must create an instance of **KubernetesServiceImporter** and encapsulate it with **{@link io.vertx.reactivex.servicediscovery.spi.ServiceImporter}** as shown in the following example:

```
{@link
examples.RxServiceDiscoveryExamples#register(io.vertx.reactivex.servicediscovery.ServiceDiscovery)}
```

The following example shows how to register **KubernetesServiceImporter()** in Eclipse Vert.x Reactive Extensions (Rx).

```
ServiceDiscovery discovery = ServiceDiscovery.create(vertx);
discovery.getDelegate().registerServiceImporter(new KubernetesServiceImporter(), new
JsonObject());
```

6.2.4. Red Hat AMQ Streams images are not available for IBM Z and IBM Power Systems

The Red Hat AMQ Streams Operator and Kafka images are not available for IBM Z and IBM Power Systems. Since the images are not available, the **vertx-kafka-client** module is not certified to work with AMQ Streams on IBM Z and IBM Power Systems.

6.2.5. Connection between a RHEL 8-based database application and a RHEL 7-based MySQL 5.7 database fails due to TLS protocol version mismatch

Description

Attempting to open a TLS-secured connection using OpenSSL between an application container built on a RHEL 8-based OpenJDK builder image and a database container built on a RHEL 7-based MySQL 5.7 container image results in a connection failure due to a **javax.net.ssl.SSLHandshakeException** at runtime: For more detail, view the issue in JIRA .

```
...
Caused by: javax.net.ssl.SSLHandshakeException: No appropriate protocol (protocol is disabled or
cipher suites are inappropriate)
...
```

Cause

The issue occurs due to a difference in the latest supported TLS protocol version between RHEL 7 and RHEL 8. The TLS implementation on RHEL 7 supports TLS protocol versions 1.0 (deprecated), 1.1, and 1.2. The TLS implementation on RHEL 8 also supports TLS protocol version 1.3, which is also the default TLS version used in RHEL 8-based builder images. This discrepancy may cause a TLS protocol version mismatch between application components while negotiating a TLS handshake, which in turn causes the connection between the application and database containers to fail.

Workaround

To prevent the issue described above, manually specify a TLS protocol version that is supported on both operating system versions in your database connection string. For example:

```
jdbc:mysql://testdb-mysql:3306/testdb?enabledTLSProtocols=TLSv1.2
```

6.2.6. False Connection reset by peer error messages when calling application endpoint

Making an HTTP request on an endpoint of an Eclipse Vert.x application using either the **curl** tool or a Java HTTP client, produces the following error in the logs after each request:

```
io.vertx.core.net.impl.ConnectionBase
SEVERE: java.io.IOException: Connection reset by peer
```

This behavior is caused by the interaction of the Netty application framework and the HAProxy load-balancer used by OpenShift. The error occurs due to existing HTTP connections being re-used by HAProxy without closing. Even though the error message is logged, no error condition occurs. HTTP requests are handled correctly and the application responds as expected.

CHAPTER 7. ADVISORIES RELATED TO THIS RELEASE

The following advisories have been issued to document enhancements, bugfixes, and CVE fixes included in this release.

- RHSA-2023:0577