Red Hat build of Eclipse Vert.x 4.2

Eclipse Vert.x 4.2 Migration Guide

For use with Eclipse Vert.x 4.2
Red Hat build of Eclipse Vert.x 4.2 Eclipse Vert.x 4.2 Migration Guide

For use with Eclipse Vert.x 4.2
Abstract

This guide provides information on how to upgrade your Eclipse Vert.x 3.x applications to Eclipse Vert.x 4.
# Table of Contents

**PREFACE** ........................................................................................................... 6

**PROVIDING FEEDBACK ON RED HAT DOCUMENTATION** ................................. 7

**MAKING OPEN SOURCE MORE INCLUSIVE** ....................................................... 8

**CHAPTER 1. CONFIGURING YOUR APPLICATION TO USE ECLIPSE VERT.X** .......... 9

**CHAPTER 2. ABOUT ECLIPSE VERT.X** .............................................................. 11

**CHAPTER 3. WHAT’S CHANGED IN ECLIPSE VERT.X 4** ..................................... 12
3.1. USE FUTURE METHODS FOR ASYNCHRONOUS OPERATIONS ................. 12
3.2. NO DEPENDENCY ON THE JACKSON DATABIND LIBRARY ..................... 13
3.3. HANDLING DEPRECATIONS AND REMOVALS ........................................... 13

**CHAPTER 4. CHANGES IN COMMON COMPONENTS** ......................................... 15
4.1. CHANGES IN MESSAGING ........................................................................... 15
4.1.1. Write and end methods in write streams are no longer fluent ............... 15
4.1.2. MessageProducer does not extend WriteStream .................................. 15
4.1.3. Removed the send methods from MessageProducer ......................... 15
4.2. CHANGES IN EVENTBUS ........................................................................... 15
4.2.1. Removed the request-response send methods in EventBus ................. 15
4.3. CHANGES IN FUTURE ............................................................................... 16
4.3.1. Support for multiple handlers for futures ........................................... 16
4.3.2. Removed the completer() method in future ....................................... 16
4.3.3. Removed the connection handler method in HTTP client request ....... 17
4.4. CHANGES IN VERTICLES ......................................................................... 17
4.4.1. Updates in the create verticle method ............................................... 17
4.4.2. Updates in the factory class and methods .......................................... 17
4.4.3. Removed the multithreaded worker verticles ...................................... 18
4.5. CHANGES IN THREADS ............................................................................ 18
4.5.1. Context affinity for non Eclipse Vert.x thread ..................................... 18
4.6. CHANGES IN HTTP ................................................................................... 19
4.6.1. Generic updates in Eclipse Vert.x HTTP methods ......................... 19
4.6.1.1. Updates in HTTP Methods for WebSocket .................................. 19
4.6.1.2. Setting the number of WebSocket connections ......................... 21
4.6.1.3. HttpMethod is available as a interface ...................................... 22
4.6.2. Changes in HTTP client ................................................................. 23
4.6.2.1. Migrating applications to Eclipse Vert.x web client .................... 23
4.6.2.2. Migrating applications to Eclipse Vert.x HTTP client ................. 24
4.6.2.2.1. Sending a simple request ....................................................... 24
4.6.2.2.2. Sending requests ................................................................. 26
4.6.2.2.3. Handling responses .............................................................. 26
4.6.2.3. Improvements in the Eclipse Vert.x HTTP client ......................... 28
4.6.2.3.1. HTTP client request and response methods take an asynchronous handler as input argument 28
4.6.2.3.2. Removed the connection handler method from HTTP client request 29
4.6.2.3.3. HTTP client tunneling using the net socket method .......... 29
4.6.2.3.4. New send() method in HttpClient class ............................... 30
4.6.2.3.5. HttpHeaders is an interface and contains MultiMap methods .... 30
4.6.2.3.6. CaseInsensitiveHeaders class is no longer public ................. 31
4.6.2.3.7. Checking the version of HTTP running on the server ............ 31
4.6.2.3.8. New methods in request options ......................................... 31
4.7. CHANGES IN CONNECTION METHODS ................................................. 32
4.7.1. Checking if authentication is required for client
4.7.2. Upgrade SSL method uses asynchronous handler

4.8. CHANGES IN LOGGING
4.8.1. Deprecated logging classes and methods
4.8.2. Removed Log4j I logger

4.9. CHANGES IN ECLIPSE VERT.X REACTIVE EXTENSIONS (RX)
4.9.1. Support for RxJava 3
4.9.2. Removed onComplete callback from write stream

4.10. CHANGES IN ECLIPSE VERT.X CONFIGURATION
4.10.1. New method to retrieve configuration

4.11. CHANGES IN JSON
4.11.1. Encapsulation of Jackson
4.11.2. Object mapping
4.11.3. Base64 encoder updated to Base64URL for JSON objects and arrays
4.11.4. Removed the JSON converter method from trust options

4.12. CHANGES IN ECLIPSE VERT.X WEB
4.12.1. Combined the functionality of user session handler in session handler
4.12.2. Removed the cookie interfaces
4.12.3. Favicon and error handlers use Vertx file system
4.12.4. Accessing the template engine
4.12.5. Removed the locale interface
4.12.6. Removed the acceptable locales method
4.12.7. Updated the method for mounting sub routers
4.12.8. Removed the create method with excluded strings for JWT authentication handling
4.12.9. Removed the create handler method that was used in OSGi environments
4.12.10. Removed the bridge options class
4.12.11. SocketJS socket event bus does not register a clustered event by default
4.12.13. OAuth2 authentication provider create methods require vertx as constructor argument

4.13. CHANGES IN ECLIPSE VERT.X WEB GRAPHQL
4.13.1. Updated methods to be supported on multiple language (polyglot) environments
4.13.2. Handling POST requests in Eclipse Vert.x Web GraphQL

4.14. CHANGES IN MICROMETER METRICS
4.14.1. TCP sent and received bytes are recorded as counters with equivalent HTTP request and response summaries
4.14.2. Renamed the metrics

4.15. CHANGES IN ECLIPSE VERT.X OPENAPI
4.15.1. New module uses router builder
4.15.2. New router builder methods
4.15.3. Handling security
4.15.4. Handling common failures
4.15.5. Accessing the OpenAPI contract model
4.15.6. Validating web requests without OpenAPI
4.15.7. Updates in the Eclipse Vert.x web API service

CHAPTER 5. CHANGES IN MICROSERVICES PATTERNS ........................................ 47
5.1. CHANGES IN ECLIPSE VERT.X CIRCUIT BREAKER ............................ 47
5.1.1. Removed execute command methods in circuit breaker

5.2. CHANGES IN ECLIPSE VERT.X SERVICE DISCOVERY ..................... 47
5.2.1. Removed create methods from service discovery that contain ServiceDiscovery argument
5.2.2. Service importer and exporter methods are no longer fluent
5.2.3. Kubernetes service importer is no longer registered automatically
CHAPTER 6. CHANGES IN ECLIPSE VERT.X AUTHENTICATION AND AUTHORIZATION

6.1. MIGRATING THE AUTHENTICATION APPLICATIONS
6.2. MIGRATING THE AUTHORIZATION APPLICATIONS
6.3. CHANGES IN KEY MANAGEMENT
   6.3.1. Secret options class is no longer available
   6.3.2. Updates in public secret keys management
   6.3.3. Changes in keystore management
6.4. DEPRECATED AND REMOVED AUTHENTICATION AND AUTHORIZATION METHODS
   6.4.1. List of removed authentication and authorization methods
   6.4.2. List of deprecated authentication and authorization methods
   6.4.3. List of deprecated authentication and authorization classes

CHAPTER 7. CHANGES IN PROTOCOLS

7.1. CHANGES IN ECLIPSE VERT.X GRPC
   7.1.1. New gRPC compiler plugin
   7.1.2. Migrating the generated code
   7.1.3. gRPC APIs support futures
7.2. CHANGES IN ECLIPSE VERT.X MQTT
   7.2.1. Some fluent methods in MQTT clients return future
   7.2.2. MqttWill messages return buffer
   7.2.3. Removed the deprecated MqttWill and authorization methods from MQTT
7.3. CHANGES IN ECLIPSE VERT.X SERVICE PROXY
   7.3.1. Using service proxy code generator

CHAPTER 8. CHANGES IN CLIENT COMPONENTS

8.1. CHANGES IN ECLIPSE VERT.X KAFKA CLIENT
   8.1.1. AdminUtils Class is no longer available
   8.1.2. Flush methods use asynchronous handler
8.2. CHANGES IN ECLIPSE VERT.X JDBC CLIENT
   8.2.1. Creating a pool
   8.2.2. Support for Typesafe Config
   8.2.3. Running SQL queries
      8.2.3.1. Running one shot queries
      8.2.3.2. Running queries on managed connections
   8.2.4. Support for stored procedures
8.3. CHANGES IN ECLIPSE VERT.X MAIL CLIENT
   8.3.1. MailAttachment is available as an interface
   8.3.2. Mail configuration interface extends the net client options
8.4. CHANGES IN ECLIPSE VERT.X AMQP CLIENT
   8.4.1. Removed methods in AMQP client that contain AmqpMessage argument
8.5. CHANGES IN ECLIPSE VERT.X MONGODB CLIENT
   8.5.1. Methods removed from MongoDB client
8.6. CHANGES IN EVENTBUS JAVASCRIPT CLIENT
   8.6.1. Versioning of JavaScript client
8.7. CHANGES IN ECLIPSE VERT.X REDIS CLIENT
   8.7.1. Migrating existing Redis client applications to new client
      8.7.1.1. Creating the client
      8.7.1.2. Migrating applications to RedisAPI
      8.7.1.3. Migrating applications directly to Redis client
      8.7.1.4. Migrating responses
   8.7.2. Updates in Eclipse Vert.x Redis client
      8.7.2.1. Removed deprecated term "slave" from Redis roles and node options

CHAPTER 9. CHANGES IN CLUSTERING
9.1. CLUSTERED FLAG REMOVED FROM OPTIONS CLASSES
9.2. CHANGES IN INFINISPAN CLUSTER MANAGER
  9.2.1. Updates in custom configurations
9.3. MIGRATING CLUSTERS
  9.3.1. Splitting the cluster
  9.3.2. Using Eclipse Vert.x EventBus Link

CHAPTER 10. MISCELLANEOUS CHANGES IN ECLIPSE VERT.X

10.1. REMOVED THE STARTER CLASS
10.2. ISOLATED DEPLOYMENT FOR JAVA 8
10.3. REMOVED HOOK METHODS FROM ECLIPSE VERT.X CONTEXT
10.4. REMOVED THE CLONE METHODS FROM OPTIONS
10.5. REMOVED EQUALS AND HASHCODE METHODS FROM OPTIONS
10.6. NEW METHOD TO CHECK FILE CACHING
10.7. SERVICE PROVIDER INTERFACE (SPI) METRICS
10.8. REMOVED THE POOLED BUFFER METHODS
10.9. METHODS TO CREATE CLIENTS THAT HAVE NO SHARED DATA SOURCES
10.10. CHANGES IN ECLIPSE VERT.X JUNIT5
  10.10.1. Support vertx-core module and updates in extensions
  10.10.2. Deprecated succeeding and failing methods in Eclipse Vert.x text context
This guide describes the updates in Eclipse Vert.x 4 release. Use the information to upgrade your Eclipse Vert.x 3.x applications to Eclipse Vert.x 4. It provides information about the new, deprecated and unsupported features in this release.

Depending on the modules used in your application, you can read the relevant section to know more about the changes in Eclipse Vert.x 4.
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. CONFIGURING YOUR APPLICATION TO USE ECLIPSE VERT.X

When you start configuring your applications to use Eclipse Vert.x, you must reference the Eclipse Vert.x BOM (Bill of Materials) artifact in the pom.xml file at the root directory of your application. The BOM is used to set the correct versions of the artifacts.

Prerequisites

- A Maven-based application

Procedure

1. Open the pom.xml file, add the io.vertx:vertx-dependencies artifact to the <dependencyManagement> section. Specify the type as pom and scope as import.

```xml
<project>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>io.vertx</groupId>
        <artifactId>vertx-dependencies</artifactId>
        <version>${vertx.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

2. Include the following properties to track the version of Eclipse Vert.x and the Eclipse Vert.x Maven Plugin you are using. Properties can be used to set values that change in every release. For example, versions of product or plugins.

```xml
<project>
  ...
  <properties>
    <vertx.version>${vertx.version}</vertx.version>
    <vertx-maven-plugin.version>${vertx-maven-plugin.version}</vertx-maven-plugin.version>
  </properties>
  ...
</project>
```

3. Specify vertx-maven-plugin as the plugin used to package your application:

```xml
<project>
  ...
  <build>
    <plugins>
      ...
      <plugin>
```
4. Include repositories and pluginRepositories to specify the repositories that contain the artifacts and plugins to build your application:

```xml
<project>
  ...
  <repositories>
    <repository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
    <repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>redhat-ga</id>
          <name>Red Hat GA Repository</name>
          <url>https://maven.repository.redhat.com/ga/</url>
        </pluginRepository>
        <repositories>
          ...
  </project>
```

Additional resources

- For more information about packaging your Eclipse Vert.x application, see the Vert.x Maven Plugin documentation.
CHAPTER 2. ABOUT ECLIPSE VERT.X

Eclipse Vert.x is a toolkit used for creating reactive, non-blocking, and asynchronous applications that run on Java Virtual Machine. (JVM). It contains several components that help you create reactive applications. It is designed to be cloud-native.

Since Eclipse Vert.x supports asynchronous applications it can be used to create applications with high volume of messages, large event processing, HTTP interactions, and so on.
CHAPTER 3. WHAT’S CHANGED IN ECLIPSE VERT.X 4

This section explains the fundamental differences between Eclipse Vert.x 4 and 3.x releases.

3.1. 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. You can use a combination of callback and future methods to migrate callback-based applications to Eclipse Vert.x 4. However, you can also continue using callbacks for asynchronous operations.

The following example shows how a callback was used for asynchronous operations in Eclipse Vert.x 3.x releases.

```java
WebClient client = WebClient.create(vertx);
HttpRequest request = client.get("/resource");

request.send(ar -> {
    if (ar.succeeded()) {
        HttpResponse response = ar.result();
    } else {
        Throwable error = ar.cause();
    }
});
```

The following example shows how to use callback and future methods together for asynchronous operations in Eclipse Vert.x 4.

```java
WebClient client = WebClient.create(vertx);
HttpRequest request = client.get("/resource");

Future<HttpResponse> response = request.send();

response.onComplete(ar -> {
    if (ar.succeeded()) {
        HttpResponse response = ar.result();
    } else {
        Throwable failure = ar.cause();
    }
});
```

Error handling is better with futures. In callbacks, you have to handle failures at every stage of the composition, whereas in futures you can handle the failure once in the end. In basic applications, you may not notice distinct difference between using callbacks and futures.

The following example shows how callbacks can be used to compose two asynchronous operations. You can see that the error is handled at every composition.

```java
client.get("/resource1").send(ar1 -> {
    if (ar1.succeeded()) {
        HttpResponse response = ar.result();
        JsonObject json = response.body();
        client.put("/resource2").sendJsonObject(ar2 -> {
            if (ar2.succeeded()) {
                //...
The following example shows how callbacks and futures can be used to compose two asynchronous operations in Eclipse Vert.x 4. The error is handled only once in the end.

```java
Future<HttpResponse> fut1 = client.get("/resource1").send();
Future<HttpResponse> fut2 = fut1.compose(response ->
  client.put("/resource2").sendJsonObject(response.body()));

fut2.onComplete(ar -> {
  if (ar.succeeded()) {
    // Handle final result
  } else {
    Throwable failure = ar.cause();
  }
});
```

3.2. NO DEPENDENCY ON THE JACKSON DATABIND LIBRARY

The JSON features in Eclipse Vert.x depend on Jackson library. Jackson Databind library enables object mapping of JSON.

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.

- If you are object mapping JSON, then you must explicitly add the dependency in your project descriptor in the `com.fasterxml.jackson.core:jackson-databind` jar.

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

In future, if you decide not to use object mapping of JSON, you can remove this dependency.

- If you are not object mapping JSON, the Jackson Databind library is not required. You can run your applications without this jar.

3.3. HANDLING DEPRECATIONS AND REMOVALS
Some 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.

- Some APIs were deprecated in an Eclipse Vert.x 3.x release and new equivalent APIs were provided in that release.
- The deprecated APIs have been removed in Eclipse Vert.x 4.

If your application uses a deprecated API, you should update your application to use the new API. This helps in migrating applications to the latest version of the product.

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.

The following example shows an EventBus method that was deprecated in an Eclipse Vert.x 3.x releases.

```java
// Send was deprecated in Vert.x 3.x release
vertx.eventBus().send("some-address", "hello world", ar -> {
    // Handle response here
});
```

The method `send(String,String,Handler<AsyncResult<Message>>)` has been replaced in Eclipse Vert.x 4 with the method `request(String,String,Handler<AsyncResult<Message>>)`.

The following example shows how to update your application to use the new method.

```java
// New method can be used in Vert.x 3.x and Vert.x 4.x releases
vertx.eventBus().request("some-address", "hello world", ar -> {
    // Handle response here
});
```
CHAPTER 4. CHANGES IN COMMON COMPONENTS

This section explains the changes in basic Eclipse Vert.x components.

4.1. CHANGES IN MESSAGING

This section explains the changes in the messaging methods.

4.1.1. Write and end methods in write streams are no longer fluent

The `WriteStream<T>.write()` and `WriteStream<T>.end()` methods are no longer fluent.

- Write and end callback methods return `void`.
- Other write and end methods return `Future<Void>`.

This is a breaking change. Update your applications if you have used the fluent aspect for write streams.

4.1.2. `MessageProducer` does not extend `WriteStream`

The `MessageProducer` interface does not extend the `WriteStream` interface.

In the previous releases of Eclipse Vert.x, the `MessageProducer` interface extended the `WriteStream` interface. The `MessageProducer` interface provided limited support for message back-pressure. Credit leaks would result in a reduction of credits in the message producer. If these leaks used all the credits, messages would not be sent.

However, `MessageConsumer` will continue to extend `ReadStream`. When `MessageConsumer` is paused and the pending message queue is full, the messages are dropped. This continues the integration with Rx generators to build message consuming pipelines.

4.1.3. Removed the send methods from `MessageProducer`

The send methods in the `MessageProducer` interface have been removed.

Use the methods `MessageProducer<T>.write(T)` instead of `MessageProducer<T>.send(T)` and `EventBus.request(String, Object, Handler)` instead of `MessageProducer.send(T, Handler)`.

4.2. CHANGES IN EVENTBUS

The following section describes the changes in EventBus.

4.2.1. Removed the request-response send methods in EventBus

The `EventBus.send(..., Handler<AsyncResult<Message<T>>>)` and `Message.reply(..., Handler<AsyncResult<Message<T>>>)` methods have been removed. These methods would have caused overloading issues in Eclipse Vert.x 4. The version of the method returning `Future<Message<T>>` would collide with the fire and forget version.

The request-response messaging pattern should use the new `request` and `replyAndRequest` methods.

- Use the method `EventBus.request(..., Handler<AsyncResult<Message<T>>>)` instead of `EventBus.send(..., Handler<AsyncResult<Message<T>>>)` to send a message.
Use the method `Message.replyAndRequest(..., Handler<AsyncResult<Message<T>>>)` instead of `Message.reply(..., Handler<AsyncResult<Message<T>>>)` to reply to the message.

The following example shows the request and reply to a message in Eclipse Vert.x 3.x releases.

**Request**

```java
eventBus.send("the-address", body, ar -> ...);
```

**Reply**

```java
eventBus.consumer("the-address", message -> {
    message.reply(body, ar -> ...);
});
```

The following example shows the request and reply to a message in Eclipse Vert.x 4.

**Request**

```java
eventBus.request("the-address", body, ar -> ...);
```

**Reply**

```java
eventBus.consumer("the-address", message -> {
    message.replyAndRequest(body, ar -> ...);
});
```

### 4.3. CHANGES IN FUTURE

This section explains the changes in future.

#### 4.3.1. Support for multiple handlers for futures

From Eclipse Vert.x 4 onward, multiple handlers are supported for a future. The `Future<T>.setHandler()` method used to set a single handler and has been removed. Use `Future<T>.onComplete()`, `Future<T>.onSuccess()`, and `Future<T>.onFailure()` methods instead to call handlers on completion, success, and failure results of an action.

The following example shows how to call a handler in Eclipse Vert.x 3.x releases.

```java
Future<String> fut = getSomeFuture();
fut.setHandler(ar -> ...);
```

The following example shows how to call the new `Future<T>.onComplete()` method in Eclipse Vert.x 4.

```java
Future<String> fut = getSomeFuture();
fut.onComplete(ar -> ...);
```

#### 4.3.2. Removed the completer() method in future
In earlier releases of Eclipse Vert.x, you would use the `Future.completer()` method to access `Handler<AsyncResult<T>>`, which was associated with the `Future`.

In Eclipse Vert.x 4, the `Future<T>.completer()` method has been removed. `Future<T>` directly extends `Handler<AsyncResult<T>>`. You can access all the handler methods using the `Future` object. The `Future` object is also a handler.

### 4.3.3. Removed the connection handler method in HTTP client request

The `HttpClientRequest.connectionHandler()` method has been removed. Use `HttpClient.connectionHandler()` method instead to call connection handlers for client requests in your application.

The following example shows how the `HttpClientRequest.connectionHandler()` method was used in Eclipse Vert.x 3.x releases.

```java
client.request().connectionHandler(conn -> {
    // Connection related code
}).end();
```

The following example shows how to use the new `HttpClient.connectionHandler()` method in Eclipse Vert.x 4.

```java
client.connectionHandler(conn -> {
    // Connection related code
});
```

### 4.4. CHANGES IN VERTICLES

This section explains the changes in the verticles.

#### 4.4.1. Updates in the create verticle method

In earlier releases of Eclipse Vert.x, `VerticleFactory.createVerticle()` method synchronously instantiated a verticle. From Eclipse Vert.x 4 onward, the method asynchronously instantiates the verticle and returns the callback `Callable<Verticle>` instead of the single verticle instance. This improvement enables the application to call this method once and invoke the returned callable multiple times for creating multiple instances.

The following code shows how verticles were instantiated in Eclipse Vert.x 3.x releases.

```java
Verticle createVerticle(String verticleName, ClassLoader classLoader) throws Exception;
```

The following code shows how verticles are instantiated in Eclipse Vert.x 4.

```java
void createVerticle(String verticleName, ClassLoader classLoader, Promise<Callable<Verticle>> promise);
```

#### 4.4.2. Updates in the factory class and methods

The `VerticleFactory` class has been simplified. The class does not require initial resolution of an identifier because the factory can instead use nested deployment to deploy the verticle.
If your existing applications use factories, in Eclipse Vert.x 4 you can update the code to use a callable when a promise completes or fails. The callable can be called several times.

The following example shows existing factories in an Eclipse Vert.x 3.x application.

```java
return new MyVerticle();
```

The following example shows how to update existing factories to use promise in Eclipse Vert.x 4.

```java
promise.complete(() -> new MyVerticle());
```

Use the `Vertx.executeBlocking()` method, if you want the factory to block code. When the factory receives the blocking code, it should resolve the promise and get the verticle instances from the promise.

### 4.4.3. Removed the multithreaded worker verticles

Multi-threaded worker verticle deployment option has been removed. This feature could only be used with Eclipse Vert.x event-bus. Other Eclipse Vert.x components such as HTTP did not support the feature.

Use the unordered `Vertx.executeBlocking()` method to achieve the same functionality as multi-threaded worker deployment.

### 4.5. CHANGES IN THREADS

This section explains the changes in threads.

#### 4.5.1. Context affinity for non Eclipse Vert.x thread

The `Vertx.getOrCreateContext()` method creates a single context for each non Eclipse Vert.x thread. The non Eclipse Vert.x threads are associated with a context the first time a context is created. In earlier releases, a new context was created each time the method was called from a non Eclipse Vert.x thread.

```java
new Thread(() -> {
    assertSame(vertx.getOrCreateContext(), vertx.getOrCreateContext());
}).start();
```

This change does not affect your applications, unless your application implicitly relies on a new context to be created with each invocation.

In the following example the n blocks run concurrently as each blocking code is called on a different context.

```java
for (int i = 0; i < n; i++) {
    vertx.executeBlocking(block, handler);
}
```

To get the same results in Eclipse Vert.x 4, you must update the code:

```java
for (int i = 0; i < n; i++) {
    vertx.executeBlocking(block, false, handler);
}
```
4.6. CHANGES IN HTTP

This section explains the changes in HTTP methods.

4.6.1. Generic updates in Eclipse Vert.x HTTP methods

The following section describes the miscellaneous updates in Eclipse Vert.x HTTP methods.

4.6.1.1. Updates in HTTP Methods for WebSocket

The changes in WebSocket are:

- The usage of the term WebSocket in method names was inconsistent. The method names had incorrect capitalization, for example, Websocket, instead of WebSocket. The methods that had inconsistent usage of WebSocket in the following classes have been removed. Use the new methods that have correct capitalization instead.

- The following methods in HttpServerOptions class have been removed.

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>New methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>getMaxWebsocketFrameSize()</td>
<td>getMaxWebSocketFrameSize()</td>
</tr>
<tr>
<td>setMaxWebsocketFrameSize()</td>
<td>setMaxWebSocketFrameSize()</td>
</tr>
<tr>
<td>getMaxWebSocketMessageSize()</td>
<td>getMaxWebSocketMessageSize()</td>
</tr>
<tr>
<td>setMaxWebSocketMessageSize()</td>
<td>setMaxWebSocketMessageSize()</td>
</tr>
<tr>
<td>getPerFrameWebsocketCompressionSupported()</td>
<td>getPerFrameWebSocketCompressionSupported()</td>
</tr>
<tr>
<td>setPerFrameWebsocketCompressionSupported()</td>
<td>setPerFrameWebSocketCompressionSupported()</td>
</tr>
<tr>
<td>getPerMessageWebsocketCompressionSupported()</td>
<td>getPerMessageWebSocketCompressionSupported()</td>
</tr>
<tr>
<td>setPerMessageWebsocketCompressionSupported()</td>
<td>setPerMessageWebSocketCompressionSupported()</td>
</tr>
<tr>
<td>getWebsocketAllowServerNoContext()</td>
<td>getWebSocketAllowServerNoContext()</td>
</tr>
<tr>
<td>setWebsocketAllowServerNoContext()</td>
<td>setWebSocketAllowServerNoContext()</td>
</tr>
<tr>
<td>getWebsocketCompressionLevel()</td>
<td>getWebSocketCompressionLevel()</td>
</tr>
<tr>
<td>setWebsocketCompressionLevel()</td>
<td>setWebSocketCompressionLevel()</td>
</tr>
<tr>
<td>getWebsocketPreferredClientNoContext()</td>
<td>getWebSocketPreferredClientNoContext()</td>
</tr>
</tbody>
</table>
The new methods for WebSocket subprotocols use `List<String>` data type instead of a comma separated string to store items.

- The following methods in `HttpClientOptions` class have been removed.

<table>
<thead>
<tr>
<th>Removed Methods</th>
<th>Replacing Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getTryUsePerMessageWebsocketCompression()</code></td>
<td><code>getTryUsePerMessageWebSocketCompression()</code></td>
</tr>
<tr>
<td><code>setTryUsePerMessageWebsocketCompression()</code></td>
<td><code>setTryUsePerMessageWebSocketCompression()</code></td>
</tr>
<tr>
<td><code>getTryWebSocketDeflateFrameCompression()</code></td>
<td><code>getTryWebSocketDeflateFrameCompression()</code></td>
</tr>
<tr>
<td><code>getWebsocketCompressionAllowClientNoContext()</code></td>
<td><code>getWebSocketCompressionAllowClientNoContext()</code></td>
</tr>
<tr>
<td><code>setWebsocketCompressionAllowClientNoContext()</code></td>
<td><code>setWebSocketCompressionAllowClientNoContext()</code></td>
</tr>
<tr>
<td><code>getWebsocketCompressionLevel()</code></td>
<td><code>getWebSocketCompressionLevel()</code></td>
</tr>
<tr>
<td><code>setWebsocketCompressionLevel()</code></td>
<td><code>setWebSocketCompressionLevel()</code></td>
</tr>
<tr>
<td><code>getWebsocketCompressionRequestServerNoContext()</code></td>
<td><code>getWebSocketCompressionRequestServerNoContext()</code></td>
</tr>
<tr>
<td><code>setWebsocketCompressionRequestServerNoContext()</code></td>
<td><code>setWebSocketCompressionRequestServerNoContext()</code></td>
</tr>
</tbody>
</table>

- The following handler methods in `HttpServer` class have been removed.
<table>
<thead>
<tr>
<th>Deprecated Methods</th>
<th>New Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>websocketHandler()</td>
<td>webSocketHandler()</td>
</tr>
<tr>
<td>websocketStream()</td>
<td>webSocketStream()</td>
</tr>
</tbody>
</table>

- **WebsocketRejectedException** is deprecated. The methods throw **UpgradeRejectedException** instead.

- The **HttpClient webSocket()** methods use **Handler<AsyncResult<WebSocket>>** instead of **Handler** or **Handler<Throwable>**.

- The number of overloaded methods to connect an HTTP client to a WebSocket has also been reduced by using the methods in **WebSocketConnectOptions** class.

- The **HttpServerRequest.upgrade()** method has been removed. This method was synchronous. Use the new method **HttpServerRequest.toWebSocket()** instead. This new method is asynchronous.

The following example shows the use of synchronous method in Eclipse Vert.x 3.x.

```java
// 3.x
server.requestHandler(req -> {
    WebSocket ws = req.upgrade();
});
```

The following example shows the use of asynchronous method in Eclipse Vert.x 4.

```java
// 4.x
server.requestHandler(req -> {
    Future<WebSocket> fut = req.toWebSocket();
    fut.onSuccess(ws -> {
    });
});
```

### 4.6.1.2. Setting the number of WebSocket connections

In Eclipse Vert.x 3.x, you could use the the HTTP client pool size to define the maximum number of WebSocket connections in an application. The value accessor methods **HttpClientOptions.maxPoolSize()** were used to get and set the WebSocket connections. The default number of connections was set to 4 for each endpoint.

The following example shows how WebSocket connections are set in Eclipse Vert.x 3.x.

```java
// 3.x
options.setMaxPoolSize(30); // Maximum connection is set to 30 for each endpoint
```

However, in Eclipse Vert.x 4, there is no pooling of WebSocket TCP connections, because the connections are closed after use. The applications use a different pool for HTTP requests. Use the value accessor methods **HttpClientOptions.maxWebSockets()** to get and set the WebSocket connections. The default number of connections is set to 50 for each endpoint.
The following example shows how to set WebSocket connections in Eclipse Vert.x 4.

```java
// 4.x
options.setMaxWebSockets(30); // Maximum connection is set to 30 for each endpoint
```

### 4.6.1.3. `HttpMethod` is available as a new interface

`HttpMethod` is available as a new interface.

In earlier releases of Eclipse Vert.x, `HttpMethod` was declared as an enumerated data type. As an enumeration, it limited the extensibility of HTTP. Further, it prevented serving other HTTP methods with this type directly. You had to use the `HttpMethod.OTHER` value along with the `rawMethod` attribute during server and client HTTP requests.

If you are using `HttpMethod` enumerated data type in a switch block, you can use the following code to migrate your applications to Eclipse Vert.x 4.

The following example shows a switch block in Eclipse Vert.x 3.x releases.

```java
switch (method) {
    case GET:
        ...
        break;
    case OTHER:
        String s = request.getRawMethod();
        if (s.equals("PROPFIND") {
            ...
        } else ...
}
```

You can also use the following code in Eclipse Vert.x 4.

```java
HttpMethod PROPFIND = HttpMethod.valueOf("PROPFIND");
if (method == HttpMethod.GET) {
    ...
} else if (method.equals(PROPFIND)) {
    ...
} else {
    ...
}
```
If you are using `HttpMethod.OTHER` value in your applications, use the following code to migrate the application to Eclipse Vert.x 4.

The following example shows you the code in Eclipse Vert.x 3.x releases.

```java
client.request(HttpMethod.OTHER, ...).setRawName("PROPFIND");
```

The following example shows you the code in Eclipse Vert.x 4.

```java
client.request(HttpMethod.valueOf("PROPFIND"), ...);
```

### 4.6.2. Changes in HTTP client

This section describes the changes in HTTP client.

The following types of Eclipse Vert.x clients are available:

**Eclipse Vert.x web client**

Use the Eclipse Vert.x web client when your applications are web oriented. For example, REST, encoding and decoding HTTP payloads, interpreting the HTTP status response code, and so on.

**Eclipse Vert.x HTTP client**

Use the Eclipse Vert.x HTTP client when your applications are used as HTTP proxy. For example, as an API gateway. The HTTP client has been updated and improved in Eclipse Vert.x 4.

**NOTE**

Eclipse Vert.x web client is based on Eclipse Vert.x HTTP client.

#### 4.6.2.1. Migrating applications to Eclipse Vert.x web client

The web client was available from Eclipse Vert.x 3.4.0 release. There is no change in the web client in Eclipse Vert.x 4.

The client provides simplified HTTP interactions and some additional features, such as HTTP session, JSON encoding and decoding, response predicates, which are not available in the Eclipse Vert.x HTTP Client.

The following example shows how to use HTTP client in Eclipse Vert.x 3.x releases.

```java
HttpClientRequest request = client.get(80, "example.com", "/", response -> {
    int statusCode = response.statusCode();
    response.exceptionHandler(err -> {
        // Handle connection error, for example, connection closed
    });
    response.bodyHandler(body -> {
        // Handle body entirely
    });
    request.exceptionHandler(err -> {
        // Handle connection error OR response error
    });
    request.end();

    });
```
The following example shows how to migrate an application to web client in Eclipse Vert.x 3.x and Eclipse Vert.x 4 releases.

```java
client.get(80, "example.com", "/some-uri")
    .send(ar -> {
        if (ar.succeeded()) {
            HttpResponse<Buffer> response = ar.result();
            // Handle response
        } else {
            // Handle error
        }
    });
```

4.6.2.2. Migrating applications to Eclipse Vert.x HTTP client

The HTTP client has fine grained control over HTTP interactions and focuses on the HTTP protocol.

The HTTP client has been updated and improved in Eclipse Vert.x 4:

- Simplified APIs with fewer interactions
- Robust error handling
- Support for connection reset for HTTP/1

The updates in HTTP client APIs are:

- The methods in `HttpClientRequest` such as, `get()`, `delete()`, `put()` have been removed. Use the method `HttpClientRequest> request(HttpMethod method, …)` instead.

  ```java
  HttpClientRequest request = client.get(80, "example.com", "/", response -> {
        int statusCode = response.statusCode();
        response.exceptionHandler(err -> {
            // Handle connection error, for example, connection closed
        });
        response.bodyHandler(body -> {
            // Handle body entirely
        });
    });
  request.exceptionHandler(err -> {
        // Handle connection error OR response error
    });
  request.end();
  ```

4.6.2.2.1. Sending a simple request

The following example shows how to send a GET request in Eclipse Vert.x 3.x releases.

```java
client.request(HttpMethod.GET, 80, "example.com", "/", ar -> {
```

The following example shows how to send a GET request in Eclipse Vert.x 4.

```java
client.request(HttpMethod.GET, 80, "example.com", "/", ar -> {
```
You can see that error handling is better in the new HTTP client.

The following example shows how to use future composition in a GET operation in Eclipse Vert.x 4.

```java
Future<Buffer> fut = client.request(HttpMethod.GET, 80, "example.com", "/")
    .compose(request -> request.send().compose(response -> {
        int statusCode = response.statusCode();
        if (statusCode == 200) {
            return response.body();
        } else {
            return Future.failedFuture("Unexpected status code");
        }
    }));
fut.onComplete(ar -> {
    if (ar.succeeded()) {
        Buffer body = ar.result();
        // Handle body entirely
    } else {
        // Handle error
    }
});
```

Future composition improves exception handling. The example checks if the status code is 200, otherwise it returns an error.
**WARNING**

When you use the HTTP client with futures, the `HttpClientResponse()` method starts emitting buffers as soon as it receives a response. To avoid this, ensure that the future composition occurs either on the event-loop (as shown in the example) or it should pause and resume the response.

### 4.6.2.2.2. Sending requests

In Eclipse Vert.x 3.x releases, you could use the `end()` method to send requests.

```java
request.end();
```

You could also send a body in the request.

```java
request.end(Buffer.buffer("hello world");
```

Since `HttpClientRequest` is a `Writestream<Buffer>`, you could also use a pipe to stream the request.

```java
writeStream.pipeTo(request, ar -> {
    if (ar.succeeded()) {
        // Sent the stream
    }
});
```

In Eclipse Vert.x 4, you can perform all the operations shown in the examples using the `get()` method. You can also use the new `send()` method to perform these operations. You can pass a buffer, a string, or a `ReadStream` as input to the `send()` method. The method returns an `HttpClientResponse` instance.

```java
// Send a request and process the response
request.onComplete(ar -> {
    if (ar.succeeded()) {
        HttpClientResponse response = ar.result();
        // Handle the response
    }
});
request.end();

// The new send method combines all the operations
request.send(ar -> {
    if (ar.succeeded()) {
        HttpClientResponse response = ar.result();
        // Handle the response
    }
});
```

### 4.6.2.2.3. Handling responses

The `HttpClientResponse` interface has been updated and improved with the following methods:
body() method

The `body()` method returns an asynchronous buffer. Use the `body()` method instead of `bodyHandler()`.

The following example shows how to use the `bodyHandler()` method to get the request body.

```java
response.bodyHandler(body -> {
    // Process the request body
});
response.exceptionHandler(err -> {
    // Could not get the request body
});
```

The following example shows how to use the `body()` method to get the request body.

```java
response.body(ar -> {
    if (ar.succeeded()) {
        // Process the request body
    } else {
        // Could not get the request body
    }
});
```

d() method

The `end()` method returns a future when a response is fully received successfully or failed. The method removes the response body. Use this method instead of `endHandler()` method.

The following example shows how to use the `endHandler()` method.

```java
response.endHandler(v -> {
    // Response ended
});
response.exceptionHandler(err -> {
    // Response failed, something went wrong
});
```

The following example shows how to use the `end()` method.

```java
response.end(ar -> {
    if (ar.succeeded()) {
        // Response ended
    } else {
        // Response failed, something went wrong
    }
});
```

You can also handle the response with methods such as, `onSuccess()`, `compose()`, `bodyHandler()` and so on. The following examples demonstrate handling responses using the `onSuccess()` method.

The following example shows how to use HTTP client with the `result()` method in Eclipse Vert.x 3.x releases.

```java
HttpClient client = vertx.createHttpClient(options);
```
client.request(HttpMethod.GET, 8443, "localhost", ",/")
    .onSuccess(request -> {
        request.onSuccess(resp -> {
            // Code to handle HTTP response
        });
    });

The following example shows how to use HTTP client with the **result()** method in Eclipse Vert.x 4.

```java
HttpClient client = vertx.createHttpClient(options);

client.request(HttpMethod.GET, 8443, "localhost", ",/")
    .onSuccess(request -> {
        request.onSuccess(resp -> {
            // Code to handle HTTP response
        });
    });
```

### 4.6.2.3. Improvements in the Eclipse Vert.x HTTP client

This section describes the improvements in HTTP client.

#### 4.6.2.3.1. HTTP client request and response methods take an asynchronous handler as input argument

The **HttpClient** and **HttpClientRequest** methods have been updated to use asynchronous handlers. The methods take **Handler<AsyncResult<HttpClientResponse>>** as input instead of **Handler<HttpClientResponse>**.

In earlier releases of Eclipse Vert.x, the **HttpClient** methods **getNow()**, **optionsNow()** and **headNow()** used to return **HttpClientRequest**, that you had to further send to perform a request. The **getNow()**, **optionsNow()** and **headNow()** methods have been removed. In Eclipse Vert.x 4, you can directly send a request with the required information using **Handler<AsyncResult<HttpClientResponse>>**.

The following examples show how to send a request in Eclipse Vert.x 3.x.

- **To perform a GET operation:**

  ```java
  Future<HttpClientResponse> f1 = client.get(8080, "localhost", ",/", HttpHeaders.set("foo", "bar"));
  ```

- **To POST with a buffer body:**

  ```java
  Future<HttpClientResponse> f2 = client.post(8080, "localhost", ",/", HttpHeaders.set("foo", "bar"), Buffer.buffer("some-data"));
  ```

- **To POST with a streaming body:**

  ```java
  Future<HttpClientResponse> f3 = client.post(8080, "localhost", ",/", HttpHeaders.set("foo", "bar"), asyncFile);
  ```
In Eclipse Vert.x 4, you can use the `requests` methods to create an `HttpClientRequest` instance. These methods can be used in basic interactions such as:

- Sending the request headers
- HTTP/2 specific operations such as setting a push handler, setting stream priority, pings, and so on.
- Creating a NetSocket tunnel
- Providing fine grained write control
- Resetting a stream
- Handling 100 continue headers manually

The following example shows you how to create an `HttpClientRequest` in Eclipse Vert.x 4.

```java
client.request(HttpMethod.GET, 8080, "example.com", "/resource", ar -> {
    if (ar.succeeded()) {
        HttpClientRequest request = ar.result();
        request.putHeader("content-type", "application/json")
        request.send(new JsonObject().put("hello", "world"))
            .onSuccess(response -> {
                //
            }).onFailure(err -> {
                //
            });
    }
});
```

4.6.2.3.2. Removed the connection handler method from HTTP client request

The `HttpClientRequest.connectionHandler()` method has been removed. Use `HttpClient.connectionHandler()` method instead to call connection handlers for client requests in your application.

The following example shows how the `HttpClientRequest.connectionHandler()` method was used in Eclipse Vert.x 3.x releases.

```java
client.request().connectionHandler(conn -> {
    // Connection related code
}).end();
```

The following example shows you how to use the new `HttpClient.connectionHandler()` method.

```java
client.connectionHandler(conn -> {
    // Connection related code
});
```

4.6.2.3.3. HTTP client tunneling using the net socket method

HTTP tunnels can be created using the `HttpClientResponse.netSocket()` method. In Eclipse Vert.x 4 this method has been updated.
To get a net socket for the connection of the request, send a socket handler in the request. The handler is called when the HTTP response header is received. The socket is ready for tunneling and can send and receive buffers.

The following example shows how to get net socket for a connection in Eclipse Vert.x 3.x releases.

```java
client.request(HttpMethod.CONNECT, uri, ar -> {
    if (ar.succeeded()) {
        HttpClientResponse response = ar.result();
        if (response.statusCode() == 200) {
            NetSocket so = response.netSocket();
        }
    }
}).end();
```

The following example shows how to get net socket for a connection in Eclipse Vert.x 4.

```java
client.request(HttpMethod.CONNECT, uri, ar -> {
}).netSocket(ar -> {
    if (ar.succeeded()) {
        // Got a response with a 200 status code
        NetSocket so = ar.result();
        // Go for tunneling
    }
}).end();
```

### 4.6.2.3.4. New send() method in HttpClient class

A new `send()` method is available in the `HttpClient` class.

The following code shows how to send a request in Eclipse Vert.x 4.

```java
Future<HttpClientResponse> f1 = client.send(HttpMethod.GET, 8080, "localhost", "/uri",
HttpHeaders.set("foo", "bar"));
```

### 4.6.2.3.5. HttpHeaders is an interface and contains MultiMap methods

In Eclipse Vert.x 4, `HttpHeaders` is an interface. In earlier releases of Eclipse Vert.x, `HttpHeaders` was a class.

The following new `MultiMap` methods have been added in the `HttpHeaders` interface. Use these methods to create `MultiMap` instances.

- `MultiMap.headers()`
- `MultiMap.set(CharSequence name, CharSequence value)`
- `MultiMap.set(String name, String value)`

The following example shows how `MultiMap` instances were created in Eclipse Vert.x 3.x releases.

```java
MultiMap headers = MultiMap.caseInsensitiveMultiMap();
```

The following examples show how to create `MultiMap` instances in Eclipse Vert.x 4.
4.6.2.3.6. **CaseInsensitiveHeaders class is no longer public**

The **CaseInsensitiveHeaders** class is no longer public. Use the **MultiMap.caseInsensitiveMultiMap()** method to create a multi-map implementation with case insensitive keys.

The following example shows how **CaseInsensitiveHeaders** method was used in Eclipse Vert.x 3.x releases.

```java
CaseInsensitiveHeaders headers = new CaseInsensitiveHeaders();
```

The following examples show how **MultiMap** method is used in Eclipse Vert.x 4.

```java
MultiMap multiMap = MultiMap#caseInsensitiveMultiMap();
```

**OR**

```java
MultiMap headers = HttpHeaders.headers();
```

4.6.2.3.7. **Checking the version of HTTP running on the server**

In earlier releases of Eclipse Vert.x, the version of HTTP running on a server was checked only if the application explicitly called the **HttpServerRequest.version()** method. If the HTTP version was HTTP/1.x, the method would return the 501 HTTP status, and close the connection.

From Eclipse Vert.x 4 onward, before a request is sent to the server, the HTTP version on the server is automatically checked by calling the **HttpServerRequest.version()** method. The method returns the HTTP version instead of throwing an exception when an invalid HTTP version is found.

4.6.2.3.8. **New methods in request options**

In Eclipse Vert.x 4, the following new methods are available in the **RequestOptions** class:

- Header
- FollowRedirects
- Timeout

The following example shows how to use the new methods.

```java
client.request(HttpMethod.GET, 8080, "example.com", "/resource", ar -> {
    if (ar.succeeded()) {
        HttpClientRequest request = ar.result();
        request.putHeader("content-type", "application/json")
        .send(new JsonObject().put("hello", "world"))
        .onSuccess(response -> {
            //
        }).onFailure(err -> {
            //
        };
```
4.7. CHANGES IN CONNECTION METHODS

This section explains the changes in connection methods.

4.7.1. Checking if authentication is required for client

The `NetServerOptions.isClientAuthRequired()` method has been removed. Use the `getClientAuth()` == `ClientAuth.REQUIRED` enumerated type to check if client authentication is required.

The following example shows how to use a switch statement to check if authentication of the client is required.

```java
switch (options.getClientAuth()) {
    case REQUIRED:
        // ... behavior same as in releases prior to {VertX} {v4}
        break;
    default:
        // fallback statement...
}
```

The following example shows how to use the check if authentication of the client is required in Eclipse Vert.x 4.

```java
if (options.getClientAuth() == ClientAuth.REQUIRED) {
    // behavior in releases prior to {VertX} {v4}
}
```

4.7.2. Upgrade SSL method uses asynchronous handler

The `NetSocket.upgradeToSsl()` method has been updated to use `Handler<AsyncResult>` instead of `Handler`. The handler is used to check if the channel has been successfully upgraded to SSL or TLS.

4.8. CHANGES IN LOGGING

This section explains the changes in logging.

4.8.1. Deprecated logging classes and methods

The logging classes `Logger` and `LoggerFactory` along with their methods have been deprecated. These logging classes and methods will be removed in a future release.

4.8.2. Removed Log4j 1 logger

The Log4j 1 logger is no longer available. However, if you want to use Log4j 1 logger, it is available with SLF4J.

4.9. CHANGES IN ECLIPSE VERT.X REACTIVE EXTENSIONS (RX)
This section describes the changes in Reactive Extensions (Rx) in Eclipse Vert.x. Eclipse Vert.x uses the RxJava library.

### 4.9.1. 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.

To upgrade to RxJava 3, you must make the following changes:

- In the `pom.xml` file, under `<dependency>` change the RxJava 1 and 2 bindings from `vertx-rx-java` or `vertx-rx-java2` to `vertx-rx-java3`.
- In your application, update the imports from `io.vertx.reactivex.*` to `io.vertx.rxjava3.*`.
- In your application, update the imports for RxJava 3 types also. For more information, see What’s new section in RxJava 3 documentation.

#### 4.9.2. Removed onComplete callback from write stream

The `WriteStreamSubscriber.onComplete()` callback has been removed. This callback was invoked if `WriteStream` had pending streams of data to be written.

In Eclipse Vert.x 4, use the callbacks `WriteStreamSubscriber.onWriteStreamEnd()` and `WriteStreamSubscriber.onWriteStreamError()` instead. These callbacks are called after `WriteStream.end()` is complete.

```
WriteStreamSubscriber<Buffer> subscriber = writeStream.toSubscriber();

subscriber.onComplete(() -> {
  // Called after writeStream.end() is invoked, even if operation has not completed
});

subscriber.onWriteStreamEnd(() -> {
  // Called after writeStream.end() is invoked and completes successfully
});

subscriber.onWriteStreamError(() -> {
  // Called after writeStream.end() is invoked and fails
});
```

### 4.10. CHANGES IN ECLIPSE VERT.X CONFIGURATION

The following section describes the changes in Eclipse Vert.x configuration.
4.10.1. New method to retrieve configuration

The method ConfigRetriever.getConfigAsFuture() has been removed. Use the method retriever.getConfig() instead.

The following example shows how configuration was retrieved in Eclipse Vert.x 3.x releases.

```java
Future<JsonObject> fut = ConfigRetriever.getConfigAsFuture(retriever);
```

The following example shows how to retrieve configuration in Eclipse Vert.x 4.

```java
fut = retriever.getConfig();
```

4.11. CHANGES IN JSON

This section describes changes in JSON.

4.11.1. Encapsulation of Jackson

All the methods in the JSON class that implement Jackson types have been removed. Use the following methods instead:

<table>
<thead>
<tr>
<th>Removed Fields/Methods</th>
<th>New methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Json.mapper() field</td>
<td>DatabindCodec.mapper()</td>
</tr>
<tr>
<td>Json.prettyMapper() field</td>
<td>DatabindCodec.prettyMapper()</td>
</tr>
<tr>
<td>Json.decodeValue(Buffer, TypeReference&lt;T&gt;)</td>
<td>JacksonCodec.decodeValue(Buffer, TypeReference)</td>
</tr>
<tr>
<td>Json.decodeValue(String, TypeReference&lt;T&gt;)</td>
<td>JacksonCodec.decodeValue(String, TypeReference)</td>
</tr>
</tbody>
</table>

For example, use the following code:

- When using Jackson `TypeReference`:
  - In Eclipse Vert.x 3.x releases:
    ```java
    List<Foo> foo1 = Json.decodeValue(json, new TypeReference<List<Foo>>() {});
    ```
  - In Eclipse Vert.x 4 release:
    ```java
    List<Foo> foo2 = io.vertx.core.json.jackson.JacksonCodec.decodeValue(json, new TypeReference<List<Foo>>() {});
    ```

- Referencing an `ObjectMapper`:
  - In Eclipse Vert.x 3.x releases:
In Eclipse Vert.x 4 release:

```java
ObjectMapper mapper = Json.mapper;
```

- In Eclipse Vert.x 3.x releases:

  ```java
  mapper = io.vertx.core.json.jackson.DatabindCodec.mapper();
  ```

- Setting an `ObjectMapper`:
  - In Eclipse Vert.x 3.x releases:
    ```java
    Json.mapper = someMapper;
    ```
  - From Eclipse Vert.x 4 onward, you cannot write a mapper instance. You should use your own static mapper or configure the `Databind.mapper()` instance.

### 4.11.2. Object mapping

In earlier releases, the Jackson core and Jackson databind dependencies were required at runtime.

From Eclipse Vert.x 4 onward, only the Jackson core dependency is required.

You will require the Jackson databind dependency only if you are object mapping JSON. In this case, you must explicitly add the dependency in your project descriptor in the `com.fasterxml.jackson.core:jackson-databind` jar.

The following methods are supported for the mentioned types.

- **Methods**
  - `JsonObject.mapFrom(Object)`
  - `JsonObject.mapTo(Class)`
  - `Json.decodeValue(Buffer, Class)`
  - `Json.decodeValue(String, Class)`
  - `Json.encode(Object)`
  - `Json.encodePrettily(Object)`
  - `Json.encodeToBuffer(Object)`

- **Type**
  - `JsonObject` and `JsonArray`
  - `Map` and `List`
  - `Number`
  - `Boolean`
  - `Enum`
  - `byte[]` and `Buffer`
The following methods are supported only with Jackson bind:

- `JsonObject.mapTo(Object)`
- `JsonObject.mapFrom(Object)`

### 4.11.3. Base64 encoder updated to Base64URL for JSON objects and arrays

The Eclipse Vert.x JSON types implement RFC-7493. In earlier releases of Eclipse Vert.x, the implementation incorrectly used Base64 encoder instead of Base64URL. This has been fixed in Eclipse Vert.x 4, and Base64URL encoder is used in the JSON types.

If you want to continue using the Base64 encoder in Eclipse Vert.x 4, you can use the configuration flag `legacy`. The following example shows how to set the configuration flag in Eclipse Vert.x 4.

```java
java -Dvertx.json.base64=legacy ...
```

During your migration from Eclipse Vert.x 3.x to Eclipse Vert.x 4 if you have partially migrated your applications, then you will have applications on both version 3 and 4. In such cases where you have two versions of Eclipse Vert.x you can use the following utility to convert the Base64 string to Base64URL.

```java
public String toBase64(String base64Url) {
    return base64Url
        .replace('+', '-
        .replace('/', '_
    }

public String toBase64Url(String base64) {
    return base64
        .replace('-', '+
        .replace('_', '/
    }
```

You must use the utility methods in the following scenarios:

- Handling integration while migrating from Eclipse Vert.x 3.x releases to Eclipse Vert.x 4.
- Handling interoperability with other systems that use Base64 strings.

Use the following example code to convert a Base64URL to Base64 encoder.

```java
String base64url = someJsonObject.getString("base64encodedElement");
String base64 = toBase64(base64url);
```

The helper functions `toBase64` and `toBase64Url` enable only JSON migrations. If you use object mapping to automatically map JSON objects to a Java POJO in your applications, then you must create a custom object mapper to convert the Base64 string to Base64URL.

The following example shows you how to create a object mapper with custom Base64 decoder.

```java
// simple deserializer from Base64 to byte[]
class ByteArrayDeserializer extends JsonDeserializer<byte[]> { 
    ByteArrayDeserializer() { 
```
4.11.4. Removed the JSON converter method from trust options

The `TrustOptions.toJSON` method has been removed.

4.12. CHANGES IN ECLIPSE VERT.X WEB

The following section describes the changes in Eclipse Vert.x web.

4.12.1. Combined the functionality of user session handler in session handler

In earlier releases of Eclipse Vert.x, you had to specify both the `UserSessionHandler` and `SessionHandler` handlers when working in a session.

To simplify the process, in Eclipse Vert.x 4, the `UserSessionHandler` class has been removed and its functionality has been added in the `SessionHandler` class. In Eclipse Vert.x 4, to work with sessions you must specify only one handler.

4.12.2. Removed the cookie interfaces

The following cookie interfaces have been removed:

- `io.vertx.ext.web.Cookie`
- `io.vertx.ext.web.handler.CookieHandler`


4.12.3. Favicon and error handlers use Vertx file system

The create methods in `FaviconHandler` and `ErrorHandler` have been updated. You must pass a `Vertx` instance object in the create methods. These methods access file system. Passing the `Vertx` object ensures consistent access to files using the 'Vertx' file system.
The following example shows how create methods were used in Eclipse Vert.x 3.x releases.

```java
FaviconHandler.create();
ErrorHandler.create();
```

The following example shows how create methods should be used in Eclipse Vert.x 4.

```java
FaviconHandler.create(vertx);
ErrorHandler.create(vertx);
```

### 4.12.4. Accessing the template engine

Use the method `TemplateEngine.unwrap()` to access the template engine. You can then apply customizations and configurations to the template.

The following methods that are used to get and set the engine configurations have been deprecated. Use the `TemplateEngine.unwrap()` method instead.

- `HandlebarsTemplateEngine.getHandlebars()`
- `HandlebarsTemplateEngine.getResolvers()`
- `HandlebarsTemplateEngine.setResolvers()`
- `JadeTemplateEngine.getJadeConfiguration()`
- `ThymeleafTemplateEngine.getThymeleafTemplateEngine()`
- `ThymeleafTemplateEngine.setMode()`

### 4.12.5. Removed the locale interface

The `io.vertx.ext.web.Locale` interface has been removed. Use the `io.vertx.ext.web.LanguageHeader` interface instead.

### 4.12.6. Removed the acceptable locales method

The `RoutingContext.acceptableLocales()` method has been removed. Use the `RoutingContext.acceptableLanguages()` method instead.

### 4.12.7. Updated the method for mounting sub routers

In earlier releases of Eclipse Vert.x, the `Router.mountSubRouter()` method incorrectly returned a `Router`. This has been fixed, and the method now returns a `Route`.

### 4.12.8. Removed the create method with excluded strings for JWT authentication handling

The `JWTAuthHandler.create(JWTAuth authProvider, String skip)` method has been removed. Use the `JWTAuthHandler.create(JWTAuth authProvider)` method instead.

The following example shows how JWT authentication handler was created in Eclipse Vert.x 3.x releases.
The following example shows how JWT authentication handler was created in Eclipse Vert.x 4.

```java
router
    .route().handler(JWTAuthHandler(jwtAuth, "/excluded/path")
```

4.12.9. Removed the create handler method that was used in OSGi environments

In Eclipse Vert.x 4, OSGi environment is no longer supported. The `StaticHandler.create(String, ClassLoader)` method has been removed because the method was used in the OSGi environment.

If you have used this method in your applications, then in Eclipse Vert.x 4 you can either add the resources to the application classpath or serve resources from the file system.

4.12.10. Removed the bridge options class

The `sockjs.BridgeOptions` class has been removed. Use the new `sockjs.SockJSBridgeOptions` class instead. The `sockjs.SockJSBridgeOptions` class contains all the options that are required to configure the event bus bridge.

There is no change in the behavior of the new class, except that the name of the data object class has changed.

In previous releases, when you used `sockjs.BridgeOptions` class to add new bridges, there were a lot of duplicate configurations. The new class contains all the possible common configurations, and removes duplicate configurations.

4.12.11. SockJS socket event bus does not register a clustered event by default

SockJSSocket no longer registers a clustered event bus consumer by default. If you want to write to the socket using the event bus, you must enable the `writeHandler` in `SockJSHandlerOptions`. When you enable the `writeHandler`, the event bus consumer is set to local by default.

```java
SockJSHandlerOptions options = new SockJSHandlerOptions()
    .setRegisterWriteHandler(true); // enable the event bus consumer registration
SockJSHandler sockJSHandler = SockJSHandler.create(vertx, options);
router.mountSubRouter("/myapp", sockJSHandler.socketHandler(sockJSSocket -> {
    // Retrieve the writeHandlerID and store it (For example, in a local map)
    String writeHandlerID = sockJSSocket.writeHandlerID();
    }));
```

You can configure the event bus consumer to a cluster.

```java
SockJSHandlerOptions options = new SockJSHandlerOptions()
    .setRegisterWriteHandler(true) // enable the event bus consumer registration
    .setLocalWriteHandler(false) // register a clustered event bus consumer
```

The `SessionHandler.setAuthProvider(AuthProvider)` method has been deprecated. Use the `SessionHandler.addAuthProvider()` method instead. The new method allows an application to work with multiple authentication providers and link the session objects to these authentication providers.

4.12.13. OAuth2 authentication provider create methods require `vertx` as constructor argument

From Eclipse Vert.x 4, `OAuth2Auth.create(Vertx vertx)` method requires `vertx` as a constructor argument. The `vertx` argument uses a secure non-blocking random number generator to generate nonce which ensures better security for applications.

4.13. CHANGES IN ECLIPSE VERT.X WEB GRAPHQL

The following section describes the changes in Eclipse Vert.x Web GraphQL.

**IMPORTANT**

Eclipse Vert.x Web GraphQL is provided as 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.

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

4.13.1. Updated methods to be supported on multiple language (polyglot) environments

The following methods have been updated and are now supported on polyglot environments: *UploadScalar* is now a factory, use the method `UploadScalar.create()` instead.

- `VertxBatchLoader` is now a factory, use the method `io.vertx.ext.web.handler.graphql.dataloader.VertxBatchLoader.create()` instead.
- `VertxDataFetcher` is now a factory, use the method `io.vertx.ext.web.handler.graphql.schema.VertxDataFetcher.create()` instead.
- `VertxPropertyDataFetcher` is now a factory, use the method `io.vertx.ext.web.handler.graphql.schema.VertxPropertyDataFetcher.create()` instead.

4.13.2. Handling POST requests in Eclipse Vert.x Web GraphQL

In prior releases, the Eclipse Vert.x Web GraphQL handler could process its own POST requests. It did not need Eclipse Vert.x Web `BodyHandler` to process the requests. However, this implementation was susceptible to DDoS attacks.

From Eclipse Vert.x 4 onward, to process POST requests `BodyHandler` is required. You must install `BodyHandler` before installing Eclipse Vert.x Web GraphQL handler.

4.14. CHANGES IN MICROMETER METRICS
The following section describes the changes in Micrometer metrics.

### 4.14.1. TCP sent and received bytes are recorded as counters with equivalent HTTP request and response summaries

In prior releases, the following metrics were recorded as distribution summaries for sockets. From Eclipse Vert.x 4 onward, these metrics are logged as counter, which report the amount of data exchanged.

- **Net client**
  - `vertx_net_client_bytes_read`
  - `vertx_net_client_bytes_written`

- **Net server**
  - `vertx_net_server_bytes_read`
  - `vertx_net_server_bytes_written`

For these counters, equivalent distribution summaries have been introduced for HTTP. These summaries are used to collect information about the request and response sizes.

- **HTTP client**
  - `vertx_http_client_request_bytes`
  - `vertx_http_client_response_bytes`

- **HTTP server**
  - `vertx_http_server_request_bytes`
  - `vertx_http_server_response_bytes`

### 4.14.2. Renamed the metrics

The following metrics have been renamed.

<table>
<thead>
<tr>
<th>Old metrics name</th>
<th>New metrics name</th>
<th>Updated in components</th>
</tr>
</thead>
<tbody>
<tr>
<td>*_connections</td>
<td>*_active_connections</td>
<td>Net client and server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP client and server</td>
</tr>
<tr>
<td>*_bytesReceived</td>
<td>*_bytes_read</td>
<td>Datagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Net client and server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP client and server</td>
</tr>
<tr>
<td>Old metrics name</td>
<td>New metrics name</td>
<td>Updated in components</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>*bytesSent</td>
<td>*bytes_written</td>
<td>Datagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Net client and server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP client and server</td>
</tr>
<tr>
<td>*requests</td>
<td>*active_requests</td>
<td>HTTP client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP server</td>
</tr>
<tr>
<td>*requestCount_total</td>
<td>*requests_total</td>
<td>HTTP client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP server</td>
</tr>
<tr>
<td>*responseTime_seconds</td>
<td>*response_time_seconds</td>
<td>HTTP client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP server</td>
</tr>
<tr>
<td>*responseCount_total</td>
<td>*responses_total</td>
<td>HTTP client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP server</td>
</tr>
<tr>
<td>*wsConnections</td>
<td>*active_ws_connections</td>
<td>HTTP client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP server</td>
</tr>
<tr>
<td>vertx_http_client_queue_delay_seconds</td>
<td>vertx_http_client_queue_time_seconds</td>
<td></td>
</tr>
<tr>
<td>vertx_http_client_queue_size</td>
<td>vertx_http_client_queue_pending</td>
<td></td>
</tr>
<tr>
<td>vertx_http_server_requestResetCount_total</td>
<td>vertx_http_server_request_resets_total</td>
<td></td>
</tr>
<tr>
<td>vertx_eventbus_bytesWritten</td>
<td>vertx_eventbus_bytes_written</td>
<td></td>
</tr>
<tr>
<td>vertx_eventbus_bytesRead</td>
<td>vertx_eventbus_bytes_read</td>
<td></td>
</tr>
<tr>
<td>vertx_eventbus_replyFailures</td>
<td>vertx_eventbus_reply_failures</td>
<td></td>
</tr>
<tr>
<td>vertx_pool_queue_delay_seconds</td>
<td>vertx_pool_queue_time_seconds</td>
<td></td>
</tr>
<tr>
<td>vertx_pool_queue_size</td>
<td>vertx_pool_queue_pending</td>
<td></td>
</tr>
</tbody>
</table>
4.15. CHANGES IN ECLIPSE VERT.X OPENAPI

In Eclipse Vert.x 4, a new module vertx-web-openapi is available. Use this module alone with vertx-web to develop contract-driven applications.

The new module works well with Eclipse Vert.x Web Router. The new module requires the following Eclipse Vert.x dependencies:

- vertx-json-schema
- vertx-web-validation

The new module is available in the package io.vertx.ext.web.openapi.

In Eclipse Vert.x 4, the older OpenAPI module vertx-web-api-contract is supported to facilitate the migration to the new module. It is recommended that you move to the new module vertx-web-openapi to take advantage of the new functionality.

4.15.1. New module uses router builder

The vertx-web-openapi module uses RouterBuilder to build the Eclipse Vert.x Web router. This router builder is similar to the router builder OpenAPI3RouterFactory in vertx-web-api-contract module.

To start working with the vertx-web-openapi module, instantiate the RouterBuilder.

```java
RouterBuilder.create(vertx, "petstore.yaml").onComplete(ar -> {
    if (ar.succeeded()) {
        // Spec loaded with success
        RouterBuilder routerBuilder = ar.result();
    } else {
        // Something went wrong during router builder initialization
        Throwable exception = ar.cause();
    }
});
```

You can also instantiate the RouterBuilder using futures.

```java
RouterBuilder.create(vertx, "petstore.yaml")
    .onSuccess(routerBuilder -> {
        // Spec loaded with success
    })
    .onFailure(exception -> {
        // Something went wrong during router builder initialization
    });
```
NOTE

The vertx-web-openapi module uses the Eclipse Vert.x file system APIs to load the files. Therefore, you do not have to specify / for the classpath resources. For example, you can specify petstore.yaml in your application. The RouterBuilder can identify the contract from your classpath resources.

4.15.2. New router builder methods

In most cases, you can search and replace usages of old OpenAPI3RouterFactory methods with the new RouterBuilder methods. The following table lists a few examples of old and new methods.

<table>
<thead>
<tr>
<th>Old OpenAPI3RouterFactory methods</th>
<th>New RouterBuilder methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>routerFactory.addHandlerByOperationId(&quot;get Pets&quot;, handler)</td>
<td>routerBuilder.operation(&quot;getPets&quot;).handler(handler)</td>
</tr>
<tr>
<td>routerFactory.addFailureHandlerByOperationId(&quot;getPets&quot;, handler)</td>
<td>routerBuilder.operation(&quot;getPets&quot;).failureHandler(handler)</td>
</tr>
<tr>
<td>routerFactory.mountOperationToEventBus(&quot;getPets&quot;, &quot;getpets.myapplication&quot;)</td>
<td>routerBuilder.operation(&quot;getPets&quot;).routeToEventBus(&quot;getpets.myapplication&quot;)</td>
</tr>
<tr>
<td>routerFactory.addGlobalHandler(handler)</td>
<td>routerBuilder.rootHandler(handler)</td>
</tr>
<tr>
<td>routerFactory.addBodyHandler(handler)</td>
<td>routerBuilder.bodyHandler(handler)</td>
</tr>
<tr>
<td>routerFactory.getRouter()</td>
<td>routerBuilder.createRouter()</td>
</tr>
</tbody>
</table>

Use the following syntax to access the parsed request parameters:

```java
RequestParameters parameters = routingContext.get(io.vertx.ext.web.validation.ValidationHandler.REQUEST_CONTEXT_KEY);
int aParam = parameters.queryParameter("aParam").getInteger();
```

4.15.3. Handling security

In Eclipse Vert.x 4, the methods RouterFactory.addSecurityHandler() and OpenAPI3RouterFactory.addSecuritySchemaScopeValidator() are no longer available.

Use the RouterBuilder.securityHandler() method instead. This method accepts io.vertx.ext.web.handler.AuthenticationHandler as an handler. The method automatically recognizes OAuth2Handler and sets up the security schema.

The new security handlers also implement the operations defined in the OpenAPI specification.

4.15.4. Handling common failures

In vertx-web-openapi module, the following failure handlers are not available. You must set up failure handlers using the Router.errorHandler(int, Handler) method.
Old methods in `vertx-web-api-contract` module | New methods in `vertx-web-openapi` module
---|---
routerFactory.setValidationFailureHandler(handler) | router.errorHandler(400, handler)
routerBuilder.setNotImplementedFailureHandler(handler) | router.errorHandler(501, handler)

### 4.15.5. Accessing the OpenAPI contract model

In Eclipse Vert.x 4, the OpenAPI contract is not mapped to plain old Java object (POJO). So, the additional swagger-parser dependency is no longer required. You can use the getters and resolvers to retrieve specific components of the contract.

The following example shows how to retrieve a specific component using a single operation.

```java
JsonObject model = routerBuilder.operation("getPets").getOperationModel();
```

The following example shows how to retrieve the full contract.

```java
JsonObject contract = routerBuilder.getOpenAPI().getOpenAPI();
```

The following example shows you how to resolve parts of the contract.

```java
JsonObject petModel = routerBuilder.getOpenAPI().getCached(JsonPointer.from("/components/schemas/Pet"));
```

### 4.15.6. Validating web requests without OpenAPI

In the `vertx-web-api-contract` module, you could validate HTTP requests using `HTTPRequestValidationHandler`. You did not have to use OpenAPI for validations.

In Eclipse Vert.x 4, to validate HTTP requests use `vertx-web-validation` module. You can import this module and validate requests without using OpenAPI. Use `ValidationHandler` to validate requests.

### 4.15.7. Updates in the Eclipse Vert.x web API service

The `vertx-web-api-service` module has been updated and can be used with the `vertx-web-validation` module. If you are working with `vertx-web-openapi` module, there is no change in the web service functionality.

However, if you do not use OpenAPI, then to use the web service module with `vertx-web-validation` module you must use the `RouteToEBServiceHandler` class.

```java
router.get("/api/transactions")
.handler(
    ValidationHandlerBuilder.create(schemaParser)
    .queryParameter(optionalParam("from", stringSchema()))
    .queryParameter(optionalParam("to", stringSchema()))
    .build()
```
The `vertx-web-api-service` module does not support `vertx-web-api-contract`. So, when you upgrade to Eclipse Vert.x 4, you must migrate your Eclipse Vert.x OpenAPI applications to `vertx-web-openapi` module.
CHAPTER 5. CHANGES IN MICROSERVICES PATTERNS

This section explains the changes in microservices patterns.

5.1. CHANGES IN ECLIPSE VERT.X CIRCUIT BREAKER

The following section describes the changes in Eclipse Vert.x circuit breaker.

5.1.1. Removed execute command methods in circuit breaker

The following methods have been removed from the CircuitBreaker class because they cannot be used with futures.

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>CircuitBreaker.executeCommand()</td>
<td>CircuitBreaker.execute()</td>
</tr>
<tr>
<td>CircuitBreaker.executeCommandWithFallback()</td>
<td>CircuitBreaker.executeWithFallback()</td>
</tr>
</tbody>
</table>

5.2. CHANGES IN ECLIPSE VERT.X SERVICE DISCOVERY

The following section describes the changes in Eclipse Vert.x service discovery.

5.2.1. Removed create methods from service discovery that contain ServiceDiscovery argument

The following create methods in service discovery that have Handler<AmqpMessage> as an argument have been removed. These methods cannot be used with futures.

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceDiscovery.create(…, Handler&lt;ServiceDiscovery&gt; completionHandler)</td>
<td>ServiceDiscovery.create(Vertx)</td>
</tr>
<tr>
<td>ServiceDiscovery.create(…, Handler&lt;ServiceDiscovery&gt; completionHandler)</td>
<td>ServiceDiscovery.create(Vertx, ServiceDiscoveryOptions)</td>
</tr>
</tbody>
</table>

5.2.2. Service importer and exporter methods are no longer fluent

The ServiceDiscovery.registerServiceImporter() and ServiceDiscovery.registerServiceExporter() methods are no longer fluent. The methods return Future<Void>.

5.2.3. Kubernetes service importer is no longer registered automatically
The `vertx-service-discovery-bridge-kubernetes` adds the `KubernetesServiceImporter` discovery bridge. The bridge imports services from Kubernetes or Openshift into the Eclipse Vert.x service discovery.

From Eclipse Vert.x 4, this 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.

The following example shows you how to manually register the bridge.

```java
JsonObject defaultConf = new JsonObject();
serviceDiscovery.registerServiceImporter(new KubernetesServiceImporter(), defaultConf);
```
CHAPTER 6. CHANGES IN ECLIPSE VERT.X AUTHENTICATION AND AUTHORIZATION

The following sections describe the changes in Eclipse Vert.x authentication and authorization.

The Eclipse Vert.x authentication module has major updates in Eclipse Vert.x 4. The `io.vertx.ext.auth.AuthProvider` interface has been split into two new interfaces:

- `io.vertx.ext.auth.authentication.AuthenticationProvider`

**IMPORTANT**

Authentication feature is provided as 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.

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

- `io.vertx.ext.auth.authorization.AuthorizationProvider`

This update enables any provider to independently perform either authentication and authorization.

### 6.1. MIGRATING THE AUTHENTICATION APPLICATIONS

The authentication mechanism has changed at the result level. In earlier releases, the result was a `User` object, which was provider specific. In Eclipse Vert.x 4, the result is a common implementation of `io.vertx.ext.auth.User`.

The following example shows how a user was authenticated in Eclipse Vert.x 3.x releases.

```java
JsonObject authInfo = new JsonObject()
    .put("username", "john")
    .put("password", "super$ecret");

// omitting the error handling for brevity
provider.authenticate(authInfo, res -> {
    if (res.succeeded()) {
        // may require type casting for example on OAuth2
        User user = res.result();
    }
});
```

The following example shows how to authenticate a user in Eclipse Vert.x 4.

```java
JsonObject authInfo = new JsonObject()
    .put("username", "john")
    .put("password", "super$ecret");

// omitting the error handling for brevity
```
6.2. MIGRATING THE AUTHORIZATION APPLICATIONS

Authorization is a new feature in Eclipse Vert.x 4. In earlier releases, you could only check if a user was authorized to perform the tasks on the User object. This meant that the provider was responsible for both authentication and authorization of the user.

In Eclipse Vert.x 4, the User object instances are not associated with a particular authentication provider. So you can authenticate and authorize a user using different providers. For example, you can authenticate a user using OAuth2 and perform authorization checks against MongoDB or SQL database.

The following example shows how an application checks if a user can use Printer #1234 in Eclipse Vert.x 3.x releases.

```java
provider.authenticate(authInfo, res -> {
    if (res.succeeded()) {
        // Never needs type casting
        User user = res.result();
    }
});
```

This authorization worked for JDBC and MongoDB. However it did not work for providers such as OAuth2, because the provider did not perform authorization checks. From Eclipse Vert.x 4, it is possible to perform such authorization checks by using different providers.

```java
// omitting the error handling for brevity
user.isAuthorized("printers:printer1234", res -> {
    if (res.succeeded()) {
        boolean hasAuthority = res.result();
        if (hasAuthority) {
            System.out.println("User can use the printer");
        } else {
            System.out.println("User cannot use the printer");
        }
    }
});
```

You can check authorizations on roles, permissions, logic operations, wildcards and any other implementation you add.

6.3. CHANGES IN KEY MANAGEMENT

In Eclipse Vert.x 4, there are major updates in handling keys. The most important change is that when a provider.authenticate(authInfo, res -> {
    if (res.succeeded()) {
        // Never needs type casting
        User user = res.result();
    }
});

// omitting the error handling for brevity
user.isAuthorized("printers:printer1234", res -> {
    if (res.succeeded()) {
        boolean hasAuthority = res.result();
        if (hasAuthority) {
            System.out.println("User can use the printer");
        } else {
            System.out.println("User cannot use the printer");
        }
    }
});
```

This authorization worked for JDBC and MongoDB. However it did not work for providers such as OAuth2, because the provider did not perform authorization checks. From Eclipse Vert.x 4, it is possible to perform such authorization checks by using different providers.

```java
// omitting the error handling for brevity
provider.getAuthorizations(user, res -> {
    if (res.succeeded()) {
        if (PermissionBasedAuthorization.create("printer1234").match(user)) {
            System.out.println("User can use the printer");
        } else {
            System.out.println("User cannot use the printer");
        }
    }
});
```

You can check authorizations on roles, permissions, logic operations, wildcards and any other implementation you add.
In Eclipse Vert.x 4, there are major updates in handling keys. The most important change is that when a key loads, there is no distinction between public buffer and private buffer.

The following classes have been updated:

- `io.vertx.ext.auth.KeyStoreOptions` used to work with jce keystores
- `io.vertx.ext.auth.SecretOptions` used to handle symmetric secrets
- `io.vertx.ext.auth.PubSecKeyOptions` used to handle public secret keys

The following section describes the changes in key management.

### 6.3.1. Secret options class is no longer available

The `SecretOptions` class is no longer available. Use the new `PubSecKeyOptions` class instead to work with a cryptographic key.

The following example shows how methods of `SecretOptions` class were used in Eclipse Vert.x 3.x releases.

```java
new SecretOptions()
    .setType("HS256")
    .setSecret("password")
```

The following example shows how methods of `PubSecKeyOptions` class should be used in Eclipse Vert.x 4.

```java
new PubSecKeyOptions()
    .setAlgorithm("HS256")
    .setSecretKey("password")
```

### 6.3.2. Updates in public secret keys management

In Eclipse Vert.x 3.x, the configuration object in public secret key management assumed that:

- Keys are configured as key-pairs.
- Key data is a PKCS8 encoded string without standard delimiters.

The following example shows how to configure key pair in Eclipse Vert.x 3.x.

```java
new PubSecKeyOptions()
    .setPublicKey(
        // remove the PEM boundaries
        pubPemString
            .replaceAll("-----BEGIN PUBLIC KEY----")
            .replaceAll("-----END PUBLIC KEY----")
    )
    .setSecretKey(
        // remove the PEM boundaries
        secPemString
            .replaceAll("-----BEGIN PUBLIC KEY----")
            .replaceAll("-----END PUBLIC KEY----")
    );
```

In Eclipse Vert.x 4, you must specify both the public and private key.
The following example shows how to configure key pair in Eclipse Vert.x 4.

```
PubSecKeyOptions pubKey =
    new PubSecKeyOptions()
    // the buffer is the exact contents of the PEM file and had boundaries included in it
    .setBuffer(pubPemString);

PubSecKeyOptions secKey =
    new PubSecKeyOptions()
    // the buffer is the exact contents of the PEM file and had boundaries included in it
    .setBuffer(secPemString);

PubSecKeyOptions x509Certificate =
    new PubSecKeyOptions()
    // the buffer is the exact contents of the PEM file and had boundaries included in it
    .setBuffer(x509PemString);
```

You can now handle X509 certificates using `PubSecKeyOptions`.

```
new KeyStoreOptions()
    .setPath("path/to/keystore.jks")
    .setPassword("keystore-password");
```

### 6.3.3. Changes in keystore management

In Eclipse Vert.x 3.x, `KeyStoreOptions` assumes that the keystore format is `jceks`, and the stored password is the same as the password of the key. As `jceks` is a proprietary format, it is recommended to use a standard format, such as JDK, instead.

When you use `KeyStoreOptions` in Eclipse Vert.x 4, you can specify a store type. For example, store types such as PKCS11, PKCS12, and so on can be set. The default store type is `jceks`.

In Eclipse Vert.x 3.x, all keystore entries would share the same password, that is, the keystore password. In Eclipse Vert.x 4, each keystore entry can have a dedicated password. If you do not want to set password for each keystore entry, you can configure the keystore password as the default password for all entries.

The following example shows how to load a `jceks` keystore in Eclipse Vert.x 3.x.

```
new KeyStoreOptions()
    .setPath("path/to/keystore.jks")
    .setPassword("key-specific-password");
```

In Eclipse Vert.x 4, the default format is assumed to be the default format configured by JDK. The format is `PKCS12` in Java 9 and above.

The following example shows how to load a `jceks` keystore in Eclipse Vert.x 4.

```
new KeyStoreOptions()
    .setPath("path/to/keystore.jks")
    // Modern JDKs use `jceks` keystore. But this type is not the default
    // If the type is not set to `jceks` then probably `pkcs12` will be used
    .setType("jceks")
    .setPassword("key-specific-password")
    // optionally if your keys have different passwords
    // and if a key specific id is not provided it defaults to
    // the keystore password
    .putPasswordProtection("key-id", "key-specific-password");
```
6.4. DEPRECATED AND REMOVED AUTHENTICATION AND AUTHORIZATION METHODS

The following sections list methods deprecated and removed for authentication and authorization.

6.4.1. List of removed authentication and authorization methods

The following methods have been removed:

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth2Auth.createKeycloak()</td>
<td>KeycloakAuth.create(vertx, JsonObject) ()</td>
</tr>
<tr>
<td>OAuth2Auth.create(Vertx, OAuth2FlowType, OAuth2ClientOptions)()</td>
<td>OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))</td>
</tr>
<tr>
<td>OAuth2Auth.create(Vertx, OAuth2FlowType)</td>
<td>OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))</td>
</tr>
<tr>
<td>User.isAuthorised()</td>
<td>User.isAuthorized()</td>
</tr>
<tr>
<td>User.setAuthProvider()</td>
<td>No replacing method</td>
</tr>
<tr>
<td>AccessToken.refreshToken()</td>
<td>AccessToken.opaqueRefreshToken()</td>
</tr>
<tr>
<td>io.vertx.ext.auth.jwt.JWTOptions data object</td>
<td>io.vertx.ext.jwt.JWTOptions data object</td>
</tr>
<tr>
<td>Oauth2ClientOptions.isUseAuthorizationHeader()</td>
<td>No replacing method</td>
</tr>
<tr>
<td>Oauth2ClientOptions.scopeSeparator()</td>
<td>No replacing method</td>
</tr>
</tbody>
</table>

6.4.2. List of deprecated authentication and authorization methods

The following methods have been deprecated:

<table>
<thead>
<tr>
<th>Deprecated methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth2Auth.decodeToken()</td>
<td>AuthProvider.authenticate()</td>
</tr>
<tr>
<td>OAuth2Auth.introspectToken()</td>
<td>AuthProvider.authenticate()</td>
</tr>
<tr>
<td>OAuth2Auth.getFlowType()</td>
<td>No replacing method</td>
</tr>
<tr>
<td>OAuth2Auth.loadJWK()</td>
<td>OAuth2Auth.jwkSet()</td>
</tr>
</tbody>
</table>
### 6.4.3. List of deprecated authentication and authorization classes

The following classes have been deprecated:

<table>
<thead>
<tr>
<th>Deprecated class</th>
<th>Replacing class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AbstractUser</code></td>
<td>Create user objects using the <code>User.create(JsonObject)</code> method.</td>
</tr>
<tr>
<td><code>AuthOptions</code></td>
<td>No replacing class</td>
</tr>
<tr>
<td><code>JDBCAuthOptions</code></td>
<td><code>JDBCAuthenticationOptions</code> for authentication and <code>JDBCAuthorizationOptions</code> for authorization</td>
</tr>
<tr>
<td><code>JDBCHashStrategy</code></td>
<td>No replacing class</td>
</tr>
<tr>
<td><code>OAuth2RBAC</code></td>
<td><code>AuthorizationProvider</code></td>
</tr>
<tr>
<td><code>Oauth2Response</code></td>
<td>Recommended to use <code>WebClient</code> class</td>
</tr>
<tr>
<td><code>KeycloakHelper</code></td>
<td>No replacing class</td>
</tr>
</tbody>
</table>
CHAPTER 7. CHANGES IN PROTOCOLS

This section explains the changes in networking protocols.

7.1. CHANGES IN ECLIPSE VERT.X GRPC

The following section describes the changes in Eclipse Vert.x gRPC.

7.1.1. New gRPC compiler plugin

In Eclipse Vert.x 4, the module protoc-gen-grpc-java is no longer available. This module was a fork of the official gRPC compiler. In earlier releases of Eclipse Vert.x, you had to work with this fork. This fork is maintained by the Eclipse project. Working with the fork was complex.

In previous releases, to work with gRPC, the following details were added to pom.xml file.

```xml
<!-- Vert.x 3.x -->
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <configuration>
    <protocArtifact>com.google.protobuf:protoc:3.2.0:exe:${os.detected.classifier}</protocArtifact>
    <pluginId>grpc-java</pluginId>
    <!-- NOTE: the gav coordinates point to the 3.x only compiler fork -->
    <pluginArtifact>io.vertx:protoc-gen-grpc-java:${vertx.grpc.version}:exe:${os.detected.classifier}</pluginArtifact>
  </configuration>
</plugin>

... ...
```

In Eclipse Vert.x 4, a new gRPC compiler plugin is available. This plugin uses the official gRPC compiler instead of the fork. To work with the new gRPC plugin, add the following details to pom.xml file.

```xml
<!-- Vert.x 4.x -->
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <configuration>
    <protocArtifact>com.google.protobuf:protoc:3.2.0:exe:${os.detected.classifier}</protocArtifact>
    <pluginId>grpc-java</pluginId>
    <!-- NOTE: the gav coordinates point to the official compiler -->
    <pluginArtifact>io.grpc:protoc-gen-grpc-java:${vertx.grpc.version}:exe:${os.detected.classifier}</pluginArtifact>
    <protocPlugins>
      <!-- NEW: a plugin is added to generate the Vert.x specific code -->
      <protocPlugin>
        <id>vertx-grpc-protoc-plugin</id>
        <groupId>io.vertx</groupId>
        <artifactId>vertx-grpc-protoc-plugin</artifactId>
        <version>${vertx.version}</version>
        <mainClass>io.vertx.grpc.protoc.plugin.VertxGrpcGenerator</mainClass>
      </protocPlugin>
    </protocPlugins>
</plugin>
```
7.1.2. Migrating the generated code

In Eclipse Vert.x 4, the new compiler is used. When the new gRPC plugin is used, the generated code is not written in the same source file. This is because the compiler does not allow custom code generation on its base class. The plugins must generate a new class with a different name to save the code.

In earlier releases of Eclipse Vert.x, the older gRPC plugin would write the generated code in the same source file.

For example, if you have the following descriptor:

```xml
<configuration>
...
</plugin>
```

In Eclipse Vert.x 3.x, the code would be generated in the `GreeterGrpc` class.

```java
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}  // 3.x
}
```

In Eclipse Vert.x 4, the code is generated in the `VertxGreeterGrpc` class.

```java
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}  // 4.x
}
```

7.1.3. gRPC APIs support futures

In Eclipse Vert.x 4, the gRPC APIs support futures. The gRPC plugin generates promisified APIs. These APIs use the standard Eclipse Vert.x input and output arguments, which makes it easier to create standard Eclipse Vert.x applications.

The following example shows the use of promise in Eclipse Vert.x 3.x:

```java
// 3.x
GreeterGrpc.GreeterVertxImplBase service =
  new GreeterGrpc.GreeterVertxImplBase() {
    ...
  }
```

```java
// 4.x
VertxGreeterGrpc.GreeterVertxImplBase service =
  new VertxGreeterGrpc.GreeterVertxImplBase() {
    ...
  }
```
The following example shows the use of futures in Eclipse Vert.x 4.

```java
// 4.x
VertxGreeterGrpc.GreeterVertxImplBase service =
new VertxGreeterGrpc.GreeterVertxImplBase() {
    @Override
    public Future<HelloReply> sayHello(HelloRequest request) {
        return Future.succeededFuture(
            HelloReply.newBuilder()
                .setMessage(request.getName())
                .build());
    }
};
```

### 7.2. CHANGES IN ECLIPSE VERT.X MQTT

The following section describes the changes in Eclipse Vert.x MQTT.

#### 7.2.1. Some fluent methods in MQTT clients return future

Some fluent methods in `MqttClient` class return `Future` instead of being fluent. For example, methods such as, `MqttClient.connect()`, `MqttClient.disconnect()`, `MqttClient.publish()` return future in Eclipse Vert.x 4.

The following example shows the use of `publish()` method in Eclipse Vert.x 3.x releases.

```java
client
    .publish("hello", Buffer.buffer("hello"), MqttQoS.EXACTLY_ONCE, false, false)
    .publish("hello", Buffer.buffer("hello"), MqttQoS.AT_LEAST_ONCE, false, false);
```

The following example shows the use of `publish()` method in Eclipse Vert.x 4 release.

```java
client.publish("hello", Buffer.buffer("hello"), MqttQoS.EXACTLY_ONCE, false, false);
client.publish("hello", Buffer.buffer("hello"), MqttQoS.AT_LEAST_ONCE, false, false);
```

#### 7.2.2. `MqttWill` messages return buffer

The `MqttWill` data object wraps a string message as an Eclipse Vert.x buffer instead of a byte array.

#### 7.2.3. Removed the deprecated `MqttWill` and authorization methods from MQTT

The following MQTT methods have been removed:

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MqttWill.willMessage()</code></td>
<td><code>MqttWill.getWillMessage()</code></td>
</tr>
<tr>
<td><code>MqttWill.willTopic()</code></td>
<td><code>MqttWill.getWillTopic()</code></td>
</tr>
<tr>
<td><code>MqttWill.willQos()</code></td>
<td><code>MqttWill.getWillQos()</code></td>
</tr>
</tbody>
</table>
7.3. CHANGES IN ECLIPSE VERT.X SERVICE PROXY

The following section describes the changes in service proxy.

7.3.1. Using service proxy code generator

The ServiceProxyProcessor class has been removed.

To use the service proxy code generator, you must import `vertx-codegen` with processor classifier in your classpath:

```xml
<dependencies>
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-codegen</artifactId>
    <classifier>processor</classifier>
  </dependency>
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-service-proxy</artifactId>
  </dependency>
</dependencies>
```

Service proxy reuses `io.vertx.codegen.CodeGenProcessor` from `vertx-codegen` to start the code generation of service proxy and handler.
CHAPTER 8. CHANGES IN CLIENT COMPONENTS

This section explains the changes in Eclipse Vert.x clients.

8.1. CHANGES IN ECLIPSE VERT.X KAFKA CLIENT

The following section describes the changes in Eclipse Vert.x Kafka client.

8.1.1. AdminUtils Class is no longer available

The AdminUtils class is no longer available. Use the new KafkaAdminClient class instead to perform administrative operations on a Kafka cluster.

8.1.2. Flush methods use asynchronous handler

The flush methods in KafkaProducer class use Handler<AsyncResult<Void>> instead of Handler<Void>.

8.2. CHANGES IN ECLIPSE VERT.X JDBC CLIENT

From Eclipse Vert.x 4, the JDBC client supports SQL client. The SQL common module has also been merged in JDBC client, that is, io.vertx:vertx-sql-common merged in io.vertx:vertx-jdbc-client module. You will have to remove the io.vertx:vertx-sql-common dependency file because io.vertx:vertx-jdbc-client will include it. With the merging of SQL common client, all the database APIs have been consolidated into the JDBC client.

In Eclipse Vert.x 4, the SQL client has been updated to include the following clients:

- Reactive PostgreSQL client. In earlier releases, it included a reactive PostgreSQL client.
- Reactive MySQL client
- Reactive DB2 client
- Continues to include reactive PostgreSQL client. This client was available in Eclipse Vert.x 3.x releases as well.
- Existing JDBC client now includes both the JDBC client API and the SQL client API

The reactive implementations use the database network protocols. This makes them resource-efficient.

JDBC calls to database are blocking calls. The JDBC client uses worker threads to make these calls non-blocking.

The following section describes the changes in Eclipse Vert.x JDBC client.

8.2.1. Creating a pool

In Eclipse Vert.x 4, you can create a pool using the JDBC client APIs. In earlier releases, you could create only clients. You could not create pools.

The following example shows how to create a client in Eclipse Vert.x 3.x.
The following example shows how to create a pool in Eclipse Vert.x 4.

```java
// 4.x
JDBCPool pool = JDBCPool.pool(vertx, jsonConfig);
```

NOTE

Though the Eclipse Vert.x 3.x APIs are supported in Eclipse Vert.x 4, it is recommended that you use the new JDBC client APIs in your applications.

A pool enables you to perform simple queries. You do not need to manage connections for simple queries. However, for complex queries or multiple queries, you must manage your connections.

The following example shows how to manage connections for queries in Eclipse Vert.x 3.x.

```java
// 3.x
client.getConnection(res -> {
    if (res.succeeded()) {
        SQLConnection connection = res.result();
        // Important, do not forget to return the connection
        connection.close();
    } else {
        // Failed to get connection
    }
});
```

The following example shows how to manage connections for queries in Eclipse Vert.x 4.

```java
// 4.x
pool
    .getConnection()
    .onFailure(e -> {
        // Failed to get a connection
    })
    .onSuccess(conn -> {
        // Important, do not forget to return the connection
        conn.close();
    });
```

8.2.2. Support for Typsesafe Config

You can use `jsonConfig` for configurations. However, using the `jsonConfig` may sometimes result in errors. To avoid these errors, the JDBC client introduces Typesafe Config.

The following example shows the basic structure of a Typesafe Config.

```java
// 4.x ONLY!!!
JDBCPool pool = JDBCPool.pool(vertx,
    // configure the connection
```
To use Typesafe Config, you must include the `agroal` connection pool in your project. The pool does not expose many configuration options and makes the configuration easy to use.

### 8.2.3. Running SQL queries

This section shows you how to run queries in the JDBC client.

#### 8.2.3.1. Running one shot queries

The following example shows how to run queries without managing the connection in Eclipse Vert.x 3.x.

```java
new JDBCConnectOptions()
    // H2 connection string
    .setJdbcUrl("jdbc:h2:~/test")
    // username
    .setUser("sa")
    // password
    .setPassword(""),
    // configure the pool
    new PoolOptions()
    .setMaxSize(16)
); // 3.x
client.query("SELECT * FROM user WHERE emp_id > ?", new JsonArray().add(1000), res -> {
    if (res.succeeded()) {
        ResultSet rs = res2.result();
        // You can use these results in your application
    }
});
```

The following example shows how to run queries without managing the connection in Eclipse Vert.x 4.

```java
// 4.x
pool
    .preparedQuery("SELECT * FROM user WHERE emp_id > ?")
    // the emp id to look up
    .execute(Tuple.of(1000))
    .onSuccess(rows -> {
        for (Row row : rows) {
            System.out.println(row.getString("FIRST_NAME"));
        }
    });
```

#### 8.2.3.2. Running queries on managed connections

The following example shows how to run queries on managed connections in Eclipse Vert.x 4.

```java
pool
    .getConnection()
```
8.2.4. Support for stored procedures

Stored procedures are supported in the JDBC client.

The following example shows how to pass **IN** arguments in Eclipse Vert.x 3.x.

```java
connection.callWithParams("call new_customer(?, ?)",
    new JsonArray().add("John").add("Doe").null(),
    res -> {
        if (res.succeeded()) {
            // Success!
        } else {
            // Failed!
        }
    });
```

The following example shows how to pass **IN** arguments in Eclipse Vert.x 4.

```java
client.preparedQuery("call new_customer(?, ?)"").execute(Tuple.of("Paulo", "Lopes"))
.onSuccess(rows -> {
    ...
});
```

In Eclipse Vert.x 3.x, the support for combining the **IN** and **OUT** arguments was very limited due to the available types. In Eclipse Vert.x 4, the pool is type safe and can handle the combination of **IN** and **OUT** arguments. You can also use **INOUT** parameters in your applications.

The following example shows handling of arguments in Eclipse Vert.x 3.x.
The following example shows handling of arguments in Eclipse Vert.x 4.

```java
// 3.x
connection.callWithParams(
    "\{ call customer_lastname(?, ?) \}",
    new JsonArray().add("John"),
    new JsonArray().addNull().add("VARCHAR"),
    res -> {
        if (res.succeeded()) {
            ResultSet result = res.result();
        } else {
            // Failed!
        }
    });

The following example shows handling of arguments in Eclipse Vert.x 4.

```java
// 4.x
client
    .preparedQuery("\{call customer_lastname(?, ?)\}"")
    .execute(Tuple.of("John", SqlOutParam.OUT(JDBCType.VARCHAR)))
    .onSuccess(rows -> {
        ...
    });
```

In the JDBC client, the data types have been updated.

- For an argument of type \texttt{OUT}, you can specify its return type. In the example, the \texttt{OUT} argument is specified as type \texttt{VARCHAR} which is a JDBC constant.
- The types are not bound by JSON limitations. You can now use database specific types instead of text constants for the type name.

### 8.3. Changes in Eclipse Vert.x Mail Client

The following section describes the changes in Eclipse Vert.x mail client.

#### 8.3.1. MailAttachment is available as an interface

From Eclipse Vert.x 4 onwards, \texttt{MailAttachment} is available as an interface. It enables you to use the mail attachment functionality in a stream. In earlier releases of Eclipse Vert.x, \texttt{MailAttachment} was available as a class and attachment for mails was represented as a data object.

#### 8.3.2. Mail configuration interface extends the net client options

\texttt{MailConfig} interface extends the \texttt{NetClientOptions} interface. Due to this extension, mail configuration also supports the proxy setting of the \texttt{NetClient}.

### 8.4. Changes in Eclipse Vert.x AMQP Client

The following section describes the changes in Eclipse Vert.x AMQP client.

#### 8.4.1. Removed methods in AMQP client that contain \texttt{AmqpMessage} argument

---

63
The AMQP client methods that had `Handler<AmqpMessage>` as an argument have been removed. In earlier releases, you could set this handler on `ReadStream<AmqpMessage>`. However, if you migrate your applications to use futures, such methods cannot be used.

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AmqpClient.createReceiver(String address, Handler&lt;AmqpMessage&gt; messageHandler, ...)</code></td>
<td><code>AmqpClient createReceiver(String address, Handler&lt;AsyncResult&lt;AmqpReceiver&gt;&gt; completionHandler)</code></td>
</tr>
<tr>
<td><code>AmqpConnection createReceiver(..., Handler&lt;AsyncResult&lt;AmqpReceiver&gt;&gt; completionHandler)</code></td>
<td><code>AmqpConnection createReceiver(String address, Handler&lt;AsyncResult&lt;AmqpReceiver&gt;&gt; completionHandler)</code></td>
</tr>
<tr>
<td><code>AmqpConnection createReceiver(String address, Handler&lt;AsyncResult&lt;AmqpReceiver&gt;&gt; completionHandler)</code></td>
<td><code>AmqpConnection createReceiver(String address, Handler&lt;AsyncResult&lt;AmqpReceiver&gt;&gt; completionHandler)</code></td>
</tr>
</tbody>
</table>

8.5. CHANGES IN ECLIPSE VERT.X MONGODB CLIENT

The following section describes the changes in Eclipse Vert.x MongoDB client.

8.5.1. Methods removed from MongoDB client

The following methods have been removed from `MongoClient` class.

<table>
<thead>
<tr>
<th>Removed methods</th>
<th>Replacing methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MongoClient.update()</code></td>
<td><code>MongoClient.updateCollection()</code></td>
</tr>
<tr>
<td><code>MongoClient.updateWithOptions()</code></td>
<td><code>MongoClient.updateCollectionWithOptions()</code></td>
</tr>
<tr>
<td><code>MongoClient.replace()</code></td>
<td><code>MongoClient.replaceDocuments()</code></td>
</tr>
<tr>
<td><code>MongoClient.replaceWithOptions()</code></td>
<td><code>MongoClient.replaceDocumentsWithOptions()</code></td>
</tr>
<tr>
<td><code>MongoClient.remove()</code></td>
<td><code>MongoClient.removeDocuments()</code></td>
</tr>
<tr>
<td><code>MongoClient.removeWithOptions()</code></td>
<td><code>MongoClient.removeDocumentsWithOptions()</code></td>
</tr>
<tr>
<td><code>MongoClient.removeOne()</code></td>
<td><code>MongoClient.removeDocument()</code></td>
</tr>
<tr>
<td><code>MongoClient.removeOneWithOptions()</code></td>
<td><code>MongoClient.removeDocumentsWithOptions()</code></td>
</tr>
</tbody>
</table>
8.6. CHANGES IN EVENTBUS JAVASCRIPT CLIENT

In Eclipse Vert.x 4, the EventBus JavaScript client module is available in a new location. You will have to update your build systems to use the module from the new location.

In Eclipse Vert.x 3.x, the event bus JavaScript client was available in various locations, for example:

- Maven Central
- NPM
- Bower.io
- CDNJS
- webjars

In Eclipse Vert.x 4, the JavaScript client is available only in npm. The EventBus JavaScript client module can be accessed from the following locations:

- CDN
- npm packages

Use the following code in your build scripts to access the module.

**JSON scripts**

```json
{
  "devDependencies": {
    "@vertx/eventbus-bridge-client.js": "1.0.0-1"
  }
}
```

**XML scripts**

```xml
<dependency>
  <groupId>org.webjars.npm</groupId>
  <artifactId>vertx__eventbus-bridge-client.js</artifactId>
  <version>1.0.0-1</version>
</dependency>
```

8.6.1. Versioning of JavaScript client

Before Eclipse Vert.x 4, every Eclipse Vert.x release included a new release of the JavaScript client.

However, from Eclipse Vert.x 4 onward, a new version of JavaScript client will be available in npm only if there changes in the client. You do not need to update your client application for every Eclipse Vert.x release, unless there is a version change.

8.7. CHANGES IN ECLIPSE VERT.X REDIS CLIENT

In Eclipse Vert.x 4, use the Redis class to work with Redis client. The class RedisClient is no longer available.
To help you migrate your applications from `RedisClient` to `Redis` class, a helper class `RedisAPI` is available. `RedisAPI` enables you to replicate the functionality similar to `RedisClient` class.

The new class contains all the enhancements in protocols and Redis server features. Use the new class to:

- Work with all Redis commands
- Connect to single servers
- Connect to high availability servers where Redis Sentinel is enabled
- Connect to cluster configurations of Redis
- Execute requests in Redis extensions
- Communicate with both RESP2 and RESP3 server protocol servers

8.7.1. Migrating existing Redis client applications to new client

You can migrate your existing applications to new `Redis` client directly or use the helper class `RedisAPI` to migrate your applications in two steps.

Before migrating the applications you must create the client.

8.7.1.1. Creating the client

The following example shows how a Redis client was created in Eclipse Vert.x 3.x releases.

```java
// Create the redis client (3.x)
RedisClient client = RedisClient
    .create(vertx, new RedisOptions().setHost(host));
```

The following example shows how to create a Redis client in Eclipse Vert.x 4.

```java
// Create the redis client (4.x)
Redis client = Redis
    .createClient(vertx, "redis://server.address:port");
```

In Eclipse Vert.x 4, the client uses the following standard connection string syntax:

```
redis[s]:/[user]:password@]server[:port]/[database]
```

8.7.1.2. Migrating applications to RedisAPI

Using the 'RedisAPI' you can now decide how to manage the connection:

- You can let the client manage the connection for you using a pool.

Or
You can control the connection by requesting a new connection. You must ensure to close or return the connection when done.

You must create the client and then update the applications to handle requests.

The following example shows how to handle requests after creating the client in Eclipse Vert.x 3.x releases.

```java
// Using 3.x
// omitting the error handling for brevity
client.set("key", "value", s -> {
    if (s.succeeded()) {
        System.out.println("key stored");
        client.get("key", g -> {
            if (s.succeeded()) {
                System.out.println("Retrieved value: " + s.result());
            }
        });
    }
});
```

The following example shows how to handle requests after creating the client in Eclipse Vert.x 4. The example uses a list for setting the key-value pairs instead of hard coding options. See Redis SET command for more information on arguments available for the command.

```java
// Using 4.x
// omitting the error handling for brevity

// 1. Wrap the client into a RedisAPI
api = RedisAPI.api(client);

// 2. Use the typed API
api.set(
    Arrays.asList("key", "value"), s -> {
        if (s.succeeded()) {
            System.out.println("key stored");
            client.get("key", g -> {
                if (s.succeeded()) {
                    System.out.println("Retrieved value: " + s.result());
                }
            });
        }
    });
```

8.7.1.3. Migrating applications directly to Redis client

When you migrate to the new Redis client directly:

- You can use all the new Redis commands.
- You can use extensions.
- You may reduce a few conversions from helper class to new client, which might improve the performance of your application.
You must create the client and then update the applications to handle requests.

The following example shows how to set and get requests after creating the client in Eclipse Vert.x 3.x releases.

```
// Using 3.x
// omitting the error handling for brevity
client.set("key", "value", s -> {
    if (s.succeeded()) {
        System.out.println("key stored");
        client.get("key", g -> {
            if (s.succeeded()) {
                System.out.println("Retrieved value: " + s.result());
            }
        });
    }
});
```

The following example shows how to handle requests after creating the client in Eclipse Vert.x 4.

```
// Using 4.x
// omitting the error handling for brevity
import static io.vertx.redis.client.Request.cmd;
import static io.vertx.redis.client.Command.*;
client.send(cmd(SET).arg("key").arg("value"), s -> {
    if (s.succeeded()) {
        System.out.println("key stored");
        client.send(cmd(GET).arg("key"), g -> {
            if (s.succeeded()) {
                System.out.println("Retrieved value: " + s.result());
            }
        });
    }
});
```

In Eclipse Vert.x 4, all the interactions use the `send(Request)` method.

### 8.7.1.4. Migrating responses

In Eclipse Vert.x 3.x, the client used to hardcode all known commands till Redis 5, and the responses were also typed according to the command.

In the new client, the commands are not hardcoded. The responses are of the type `Response`. The new wire protocol has more range of types.

In older client, a response would be of following types:

- `null`
- `Long`
- `String`
- **JsonArray**
- **JsonObject** (For `INFO` and `HMGET` array responses)

In the new client, the response is of following types:

- `null`
- **Response**

The **Response** object has type converters. For example, converters such as:

- `toString()`
- `toInteger()`
- `toBoolean()`
- `toBuffer()`

If the received data is not of the requested type, then the type converters convert it to the closest possible data type. When the conversion to a particular type is not possible, the **UnsupportedOperationException** is thrown. For example, conversion from `String` to `List` or `Map` is not possible.

You can also handle collections, because the **Response** object implements the **Iterable** interface.

The following example shows how to perform a MGET request.

```java
// Using 4.x
// omitting the error handling for brevity
import static io.vertx.redis.client.Request.cmd;
import static io.vertx.redis.client.Command.*;

client.send(cmd(MGET).arg("key1").arg("key2").arg("key3"), mget -> {
  mget.result().forEach(value -> {
    // Use the single value
```

### 8.7.2. Updates in Eclipse Vert.x Redis client

This section describes changes in Redis client.

#### 8.7.2.1. Removed deprecated term "slave" from Redis roles and node options

The deprecated term "slave" has been replaced with "replica" in Redis roles and node options.

**Roles**

The following example shows the usage of **SLAVE** role in Eclipse Vert.x 3.x releases.

```java
// Before (3.x)
Redis.createClient(
  rule.vertx(),
  new RedisOptions()
```
The following example shows the usage of **REPLICA** role in Eclipse Vert.x 4.

```java
// After (4.x)
Redis.createClient(
    rule.vertx(),
    new RedisOptions()
        .setType(RedisClientType.SENTINEL)
        .addConnectionString("redis://localhost:5000")
        .setMasterName("sentinel7000")
        .setRole(RedisRole.REPLICA));
```

### Node options

The following example shows you usage of node type **RedisSlaves** in Eclipse Vert.x 3.x releases.

```java
// Before (3.9)
options.setUseSlaves(RedisSlaves);
```

The following example shows you usage of node type **RedisReplicas** in Eclipse Vert.x 4.

```java
// After (4.x)
options.setUseReplicas(RedisReplicas);
```
CHAPTER 9. CHANGES IN CLUSTERING

This section explains the changes in clustering.

9.1. CLUSTERED FLAG REMOVED FROM OPTIONS CLASSES

The methods and boolean value that were used to specify, get, and set clustering in Eclipse Vert.x applications have been removed from VertxOptions and EventBusOptions classes.

9.2. CHANGES IN INFINISPAN CLUSTER MANAGER

The following section describes the changes in the Infinispan cluster manager.

9.2.1. Updates in custom configurations

The Infinispan cluster manager is based on Infinispan 12.

In Eclipse Vert.x 4, the clustering SPI has been redesigned. The subscription data model has changed. As a result of this, Eclipse Vert.x 3.x nodes and Eclipse Vert.x 4 nodes cannot be added together in the same Infinispan cluster.

The Eclipse Vert.x applications are not impacted by this change as the EventBus and SharedData APIs remain the same.

If you had a custom Infinispan configuration file in your Eclipse Vert.x 3.x application:

- Change the __vertx.subs cache type to replicated instead of distributed.
- Add the replicated cache __vertx.nodeInfo.

```xml
<cache-container default-cache="distributed-cache">
  <distributed-cache name="distributed-cache"/>
  <replicated-cache name="__vertx.subs"/>
  <replicated-cache name="__vertx.haInfo"/>
  <replicated-cache name="__vertx.nodeInfo"/>
  <distributed-cache-configuration name="__vertx.distributed.cache.configuration"/>
</cache-container>
```

If you run an Eclipse Vert.x cluster on Openshift, the infinispan-cloud JAR is no longer needed. The JAR has been removed from the dependencies section of the build file. The configuration files that were bundled in this JAR are now included in the infinispan-core JAR.

9.3. MIGRATING CLUSTERS

It is important to decide the migration strategy for your codebase. This is because you cannot add Eclipse Vert.x 3.x nodes and Eclipse Vert.x 4 nodes together in a single cluster for the following reasons:

- Cluster manager upgrades - Major version upgrades in cluster managers prevent backward compatibility.
- Subscription data changes - Eclipse Vert.x has changed the format of the EventBus subscription data stored in cluster managers.
Transport protocol changes - Eclipse Vert.x has changed some fields in the message transport protocol in the cluster.

If you have an Eclipse Vert.x cluster for a single application or for some closely related microservices, you can migrate the entire codebase to the new cluster at one time.

However, if you cannot migrate the codebase at one time, use the recommendations in this section to migrate an Eclipse Vert.x 3.x codebase to Eclipse Vert.x 4.

9.3.1. Splitting the cluster

If you have a cluster where different teams have deployed verticles for their applications, you can consider splitting the Eclipse Vert.x 3.x cluster into smaller ones. Note that after splitting the cluster, the separated components will not be able to communicate using the clustering features. You can split the cluster using the following components:

- EventBus request and reply - HTTP or RESTful web services, gRPC
- EventBus send and publish - Messaging systems, Postgres LISTEN and NOTIFY, Redis Pub and Sub
- Shared Data - Redis, Infinispan

After you split the cluster, each team can move to Eclipse Vert.x 4 when they are ready or if required.

9.3.2. Using Eclipse Vert.x EventBus Link

If you cannot split your cluster, then use Vert.x EventBus Link to migrate your codebase incrementally.

Vert.x EventBus Link is a tool that connects an Eclipse Vert.x 3.x clustered EventBus to an Eclipse Vert.x 4 clustered EventBus.

The migration of shared data API, that is, maps, counters and locks is not supported.

The tool creates an EventBusLink object that implements the EventBus interface. An instance of EventBusLink is created on at least one node of each cluster. The instance is created by providing a set of addresses and its behavior depends on the message paradigm:

- fire and forget and request and reply - The message is sent to the remote cluster.
- publish - The message is sent to both this cluster and the remote cluster.

The Eclipse Vert.x EventBus Link creates a WebSocket server to receive messages and uses a WebSocket client to send them.

See the sections get started and using for more details.
CHAPTER 10. MISCELLANEOUS CHANGES IN ECLIPSE VERT.X

The following section describes miscellaneous changes in Eclipse Vert.x 4.

10.1. REMOVED THE STARTER CLASS

The Starter class has been removed. Use the Launcher class instead to start your Eclipse Vert.x applications without the main() method.

10.2. ISOLATED DEPLOYMENT FOR JAVA 8

Eclipse Vert.x 4 supports Java 11. This Java version does not support isolated class loading. In Eclipse Vert.x 4, isolated class loading will be supported for Java 8.

10.3. REMOVED HOOK METHODS FROM ECLIPSE VERT.X CONTEXT

The methods Context.addCloseHook() and Context.removeCloseHook() methods have been removed from the Context class. These methods have been moved to the internal interface InternalContext.

10.4. REMOVED THE CLONE METHODS FROM OPTIONS

The methods KeyCertOptions.clone(), TrustOptions.clone(), and SSLEngineOptions.clone() have been removed. Use the methods KeyCertOptions.copy(), TrustOptions.copy(), and SSLEngineOptions.copy() instead.

10.5. REMOVED EQUALS AND HASHCODE METHODS FROM OPTIONS

The VertxOptions.equals() and VertxOptions.hashCode() methods have been removed.

10.6. NEW METHOD TO CHECK FILE CACHING

The VertxOptions.fileResolverCachingEnabled() method has been removed. Use FileSystemOptions.isFileCachingEnabled() method instead to check if file caching has been enabled to resolve classpaths.

10.7. SERVICE PROVIDER INTERFACE (SPI) METRICS

The Metrics.isEnabled() method has been removed. The Service Provider Interface (SPI) metrics will return a null object to indicate that metrics are not enabled.

10.8. REMOVED THE POOLED BUFFER METHODS

The pooled buffer methods TCPSSLOptions.isUsePooledBuffers() and TCPSSLOptions.setUsePooledBuffers() have been removed.

10.9. METHODS TO CREATE CLIENTS THAT HAVE NO SHARED DATA SOURCES
Use the following new methods to create clients that do not have shared data sources with other clients. These methods maintain their own data sources.

<table>
<thead>
<tr>
<th>Deprecated Methods</th>
<th>New Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoClient.createNonShared()</td>
<td>MongoClient.create()</td>
</tr>
<tr>
<td>JDBCClient.createNonShared()</td>
<td>wJDBCClient.create()</td>
</tr>
<tr>
<td>CassandraClient.createNonShared()</td>
<td>CassandraClient.create()</td>
</tr>
<tr>
<td>MailClient.createNonShared()</td>
<td>MailClient.create()</td>
</tr>
</tbody>
</table>

10.10. CHANGES IN ECLIPSE VERT.X JUNIT5

The following section describes the changes in Eclipse Vert.x JUnit5.

10.10.1. Support vertx-core module and updates in extensions

The vertx-core module has been updated to use a service provider interface for parameter injection. This change resulted in following updates in JUnit5:

- You must call the Vertx parameter before any parameter that requires it for creation. For example, when injecting a WebClient.

- vertx-junit5 module supports only the vertx-core module.

- reactivemessaging-junit5-extensions module hosts extensions that contain extra parameter types, such as, WebClient.

- RxJava 1 and 2 bindings are now available as vertx-junit5-rx-java and vertx-junit5-rx-java2 modules in the vertx-junit5-extensions repository. From Eclipse Vert.x 4.1.0, the RxJava 3 binding vertx-junit5-rx-java3 is available.

10.10.2. Deprecated succeeding and failing methods in Eclipse Vert.x text context

The VertxTestContext.succeeding() and VertxTestContext.failing() methods have been deprecated. Use VertxTestContext.succeedingThenComplete() and VertxTestContext.failingThenComplete() methods instead.