Table of Contents

ECLIPSE VERT.X 4 MIGRATION GUIDE

This guide describes the updates in Eclipse Vert.x 4 release. Use the information to upgrade your Vert.x 3.x applications to 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 Vert.x 4.

About Vert.x

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 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.

What's changed in Vert.x 4

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

Use future methods for asynchronous operations

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 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 Vert.x 3.x releases.

```
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 Vert.x 4.

```
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.

```
client.get("/resource1").send(ar1 -> {
  if (ar1.succeeded()) {
    HttpResponse response = ar.result();
    JsonObject json = response.body();
    client.put("/resource2").sendJsonObject(ar2 -> {
      if (ar2.succeeded()) {
        // Handle final result
      } else {
        Throwable failure2 = ar.cause();
      }
    });
  } else {
    Throwable failure1 = ar.cause();
  }
});
```

The following example shows how callbacks and futures can be used to compose two asynchronous operations in Vert.x 4. The error is handled only once in the end.

```
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();
  }
});
```

No dependency on the Jackson Databind library

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

In 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.

  ```
  <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.

Handling deprecations and removals

Some features and functions have been deprecated or removed in Vert.x 4. Before you migrate your applications to Vert.x 4, check for deprecations and removals.

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

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

```
// 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 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.

```
// 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
});
```

Changes in common components

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

Changes in messaging

This section explains the changes in the messaging methods.

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.

`MessageProducer` does not extend `WriteStream`

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

In the previous releases of 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.

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)`.

Changes in EventBus

The following section describes the changes in EventBus.

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 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 Vert.x 3.x releases.

Request

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

Reply

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

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

Request

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

Reply

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

Changes in future

This section explains the changes in future.

Support for multiple handlers for futures

From 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 Vert.x 3.x releases.

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

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

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

Removed the `completer()` method in future

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

In 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.

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 Vert.x 3.x releases.

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

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

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

Changes in verticles

This section explains the changes in the verticles.

Updates in the create verticle method

In earlier releases of Vert.x, `VerticleFactory.createVerticle()` method synchronously instantiated a verticle. From 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 Vert.x 3.x releases.

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

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

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

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 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 Vert.x 3.x application.

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

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

```
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.

Removed the multithreaded worker verticles

Multi-threaded worker verticle deployment option has been removed. This feature could only be used with Vert.x event-bus. Other 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.

Changes in threads

This section explains the changes in threads.

Context affinity for non Vert.x thread

The `Vertx.getOrCreateContext()` method creates a single context for each non Vert.x thread. The non 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 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 Vert.x 4, you must update the code:

JAVA

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

Changes in HTTP

This section explains the changes in HTTP methods.

Generic updates in Vert.x HTTP methods

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

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.

Removed methods	New methods
`getMaxWebsocketFrameSize()`	`getMaxWebSocketFrameSize()`
`setMaxWebsocketFrameSize()`	`setMaxWebSocketFrameSize()`
`getMaxWebsocketMessageSize()`	`getMaxWebSocketMessageSize()`
`setMaxWebsocketMessageSize()`	`setMaxWebSocketMessageSize()`
`getPerFrameWebsocketCompressionSupported()`	`getPerFrameWebSocketCompressionSupported()`
`setPerFrameWebsocketCompressionSupported()`	`setPerFrameWebSocketCompressionSupported()`
`getPerMessageWebsocketCompressionSupported()`	`getPerMessageWebSocketCompressionSupported()`
`setPerMessageWebsocketCompressionSupported()`	`setPerMessageWebSocketCompressionSupported()`
`getWebsocketAllowServerNoContext()`	`getWebSocketAllowServerNoContext()`
`setWebsocketAllowServerNoContext()`	`setWebSocketAllowServerNoContext()`
`getWebsocketCompressionLevel()`	`getWebSocketCompressionLevel()`
`setWebsocketCompressionLevel()`	`setWebSocketCompressionLevel()`
`getWebsocketPreferredClientNoContext()`	`getWebSocketPreferredClientNoContext()`
`setWebsocketPreferredClientNoContext()`	`setWebSocketPreferredClientNoContext()`
`getWebsocketSubProtocols()`	`getWebSocketSubProtocols()`
`setWebsocketSubProtocols()`	`setWebSocketSubProtocols()`

    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.

Removed Methods	Replacing Methods
`getTryUsePerMessageWebsocketCompression()`	`getTryUsePerMessageWebSocketCompression()`
`setTryUsePerMessageWebsocketCompression()`	`setTryUsePerMessageWebSocketCompression()`
`getTryWebsocketDeflateFrameCompression()`	`getTryWebSocketDeflateFrameCompression()`
`getWebsocketCompressionAllowClientNoContext()`	`getWebSocketCompressionAllowClientNoContext()`
`setWebsocketCompressionAllowClientNoContext()`	`setWebSocketCompressionAllowClientNoContext()`

Removed Methods	Replacing Methods
`getWebsocketCompressionLevel()`	`getWebSocketCompressionLevel()`
`setWebsocketCompressionLevel()`	`setWebSocketCompressionLevel()`
`getWebsocketCompressionRequestServerNoContext()`	`getWebSocketCompressionRequestServerNoContext()`
`setWebsocketCompressionRequestServerNoContext()`	`setWebSocketCompressionRequestServerNoContext()`

- The following handler methods in `HttpServer` class have been removed.

Deprecated Methods	New Methods
`websocketHandler()`	`webSocketHandler()`
`websocketStream()`	`webSocketStream()`

- `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 Vert.x 3.x.

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

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

```
// 4.0
server.requestHandler(req -> {
  Future<WebSocket> fut = req.toWebSocket();
  fut.onSuccess(ws -> {
  });
});
```

Setting the number of WebSocket connections

In 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 Vert.x 3.x.

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

However, in 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 Vert.x 4.

```
// 4.0
options.setMaxWebSockets(30); // Maximum connection is set to 30 for each endpoint
```

`HttpMethod` is available as a interface

`HttpMethod` is available as a new interface.

In earlier releases of 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 Vert.x 4.

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

JAVA

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

The following example shows a switch block in Vert.x 4.

JAVA

```
switch (method.name()) {
  case "GET":
    ...
    break;
  case "PROPFIND";
    ...
    break;
}
```

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

```
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 Vert.x 4.

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

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

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

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

Changes in HTTP client

This section describes the changes in HTTP client.

The following types of Vert.x clients are available:

Vert.x web client

Use the 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.

Vert.x HTTP client

Use the 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 Vert.x 4.

NOTE | Vert.x web client is based on Vert.x HTTP client.

Migrating applications to Vert.x web client

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

The following example shows how to use HTTP client in 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 Vert.x 3.x and Vert.x 4 releases.

JAVA

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

Migrating applications to 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 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.
- `HttpClientRequest` instance is created when a request or response is possible. For example, an `HttpClientRequest` instance is created when the client connects to the server or a connection is reused from the pool.

Sending a simple request

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

JAVA

```
client.request(HttpMethod.GET, 80, "example.com", "/", ar -> {
  if (ar.succeeded()) {
    HttpClientRequest = ar.result();
    request.send(ar2 -> {
      if (ar2.succeeded()) {
        HttpClientResponse = ar2.result();
        int statusCode = response.statusCode();
        response.body(ar3 -> {
          if (ar3.succeeded()) {
            Buffer body = ar3.result();
            // Handle body entirely
          } else {
            // Handle server error, for example, connection closed
          }
        });
      } else {
        // Handle server error, for example, connection closed
      }
    });
  } else {
    // Connection error, for example, invalid server or invalid SSL certificate
  }
});
```

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 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("Unexpectd 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.

Sending requests

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

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

You could also send a body in the request.

```
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 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
  }
}));
```

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
  }
});
```

`end()` 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, `onSucces()`, `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 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 Vert.x 4.

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

    client.request(HttpMethod.GET, 8443, "localhost", "/")
      .onSuccess(request -> {
        request.response().onSuccess(resp -> {

        //Code to handle HTTP response
        });
      });
```

Improvements in the Vert.x HTTP client

This section describes the improvements in HTTP client.

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 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 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 Vert.x 3.x.

- To perform a GET operation:

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

- To POST with a buffer body:

JAVA
```
Future<HttpClientResponse> f2 = client.post(8080, "localhost", "/uri", HttpHeaders.set("foo", "bar"), Buffer.buffer("som
```

- To POST with a streaming body:

```
Future<HttpClientResponse> f3 = client.post(8080, "localhost", "/uri", HttpHeaders.set("foo", "bar"), asyncFile);
```

In 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 Vert.x 4.

```
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 -> {
      //
       });
     }
})
```

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 Vert.x 3.x releases.

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

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

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

HTTP client tunneling using the net socket method

HTTP tunnels can be created using the `HttpClientResponse.netSocket()` method. In 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 Vert.x 3.x releases.

```
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 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();
```

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 Vert.x 4.

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

HttpHeaders is an interface and contains `MultiMap` methods

In Vert.x 4, `HttpHeaders` is an interface. In earlier releases of 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 Vert.x 3.x releases.

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

The following examples show how to create `MultiMap` instances in Vert.x 4.

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

```
MultiMap headers = HttpHeaders.set("content-type", "application.data");
```

`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 Vert.x 3.x releases.

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

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

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

OR

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

Checking the version of HTTP running on the server

In earlier releases of 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 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.

New methods in request options

In 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")
    request.send(new JsonObject().put("hello", "world"))
      .onSuccess(response -> {
      //
      }).onFailure(err -> {
      //
       });
     }
})
```

Changes in connection methods

This section explains the changes in connection methods.

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.

```
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 Vert.x 4.

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

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.

Changes in logging

This section explains the changes in logging.

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.

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`.

Changes in Vert.x Reactive Extensions (Rx)

This section describes the changes in Reactive Extensions (Rx) in Vert.x. Vert.x uses the RxJava library.

Support for RxJava 3

From Vert.x 4.1.0, RxJava 3 is supported.

- A new rxified API is available in the `io.vertx.rxjava3` package.
- Integration with 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.

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

JAVA

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

The following example shows how to create the adapter from a `WriteStream` in Vert.x 3.x releases.

JAVA

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

The following examples show how to create the adapter from a WriteStream using the new callback methods in Vert.x 4 release:

JAVA

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

JAVA

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

Changes in Vert.x configuration

The following section describes the changes in Vert.x configuration.

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 Vert.x 3.x releases.

JAVA

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

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

JAVA

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

Changes in JSON

This section describes changes in JSON.

Encapsulation of Jackson

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

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

For example, use the following code:

- When using Jackson `TypeReference` :
  - In Vert.x 3.x releases:

```
List<Foo> foo1 = Json.decodeValue(json, new TypeReference<List<Foo>>() {});
```
JAVA

  - In Vert.x 4 release:

```
List<Foo> foo2 = io.vertx.core.json.jackson.JacksonCodec.decodeValue(json, new TypeReference<List<Foo>>() {});
```
JAVA

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

```
ObjectMapper mapper = Json.mapper;
```
JAVA

  - In Vert.x 4 release:

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

- Setting an `ObjectMapper` :
  - In Vert.x 3.x releases:

```
Json.mapper = someMapper;
```
JAVA

  - From Vert.x 4 onward, you cannot write a mapper instance. You should use your own static mapper or configure the `Databind.mapper()` instance.

Object mapping

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

From 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`
  - `Instant`

The following methods are supported only with Jackson bind:

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

Base64 encoder updated to Base64URL for JSON objects and arrays

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

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

JAVA

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

During your migration from Vert.x 3.x and 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 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 Vert.x 3.x releases to 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() {
  }

  public byte[] deserialize(JsonParser p, DeserializationContext ctxt) {
    String text = p.getText();
    return Base64.getDecoder()
      .decode(text);
  }
}

// ...

ObjectMapper mapper = new ObjectMapper();

// create a custom module to address the Base64 decoding
SimpleModule module = new SimpleModule();
module.addDeserializer(byte[].class, new ByteArrayDeserializer());
mapper.registerModule(module);

// JSON to POJO with custom deserializer
mapper.readValue(json, MyClass.class);
```

Removed the JSON converter method from trust options

The `TrustOptions.toJSON` method has been removed.

Changes in Vert.x web

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

Combined the functionality of user session handler in session handler

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

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

Removed the cookie interfaces

The following cookie interfaces have been removed:

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

Use the `io.vertx.core.http.Cookie` interface instead.

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 Vert.x 3.x releases.

JAVA

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

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

JAVA

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

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()`

Removed the locale interface

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

Removed the acceptable locales method

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

Updated the method for mounting sub routers

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

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 Vert.x 3.x releases.

JAVA

```
router
   // protect everything but "/excluded/path"
   .route().handler(JWTAuthHandler(jwtAuth, "/excluded/path")
```

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

JAVA

```
router
   .route("/excluded/path").handler(/* public access to "/excluded/path" */)
   // protect everything
   .route().handler(JWTAuthHandler(jwtAuth)
```

Removed the create handler method that was used in OSGi environments

In 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 Vert.x 4 you can either add the resources to the application classpath or serve resources from the file system.

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.

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
Router router = Router.router(vertx);
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
```

New method for adding authentication provider

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.

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

From 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.

Changes in Vert.x Web GraphQL

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

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.

Handling POST requests in Vert.x Web GraphQL

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

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

Changes in Micrometer metrics

The following section describes the changes in Micrometer metrics.

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 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`

Renamed the metrics

The following metrics have been renamed.

Old metrics name	New metrics name	Updated in components
`*_connections`	`*_active_connections`	Net client and server HTTP client and server
`*_bytesReceived`	`*_bytes_read`	Datagram Net client and server HTTP client and server
`*_bytesSent`	`*_bytes_written`	Datagram Net client and server HTTP client and server
`*_requests`	`*_active_requests`	HTTP client HTTP server
`*_requestCount_total`	`*_requests_total`	HTTP client HTTP server
`*_responseTime_seconds`	`*_response_time_seconds`	HTTP client HTTP server
`*_responseCount_total`	`*_responses_total`	HTTP client HTTP server
`*_wsConnections`	`*_active_ws_connections`	HTTP client HTTP server
`vertx_http_client_queue_delay_seconds`	`vertx_http_client_queue_time_seconds`	
`vertx_http_client_queue_size`	`vertx_http_client_queue_pending`	
`vertx_http_server_requestResetCount_total`	`vertx_http_server_request_resets_total`	
`vertx_eventbus_bytesWritten`	`vertx_eventbus_bytes_written`	
`vertx_eventbus_bytesRead`	`vertx_eventbus_bytes_read`	
`vertx_eventbus_replyFailures`	`vertx_eventbus_reply_failures`	
`vertx_pool_queue_delay_seconds`	`vertx_pool_queue_time_seconds`	

Old metrics name	New metrics name	Updated in components
`vertx_pool_queue_size`	`vertx_pool_queue_pending`	
`vertx_pool_inUse`	`vertx_pool_in_use`	

Changes in Vert.x OpenAPI

In 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 Vert.x Web `Router`. The new module requires the following Vert.x dependencies:

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

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

In 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.

New module uses router builder

The `vertx-web-openapi` module uses `RouterBuilder` to build the Vert.x Web router. This router builder is similar to the router builer `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 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.

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.

Old `OpenAPI3RouterFactory` methods	New `RouterBuilder` methods
`routerFactory.addHandlerByOperationId("getPets", handler)`	`routerBuilder.operation("getPets").handler(handler)`
`routerFactory.addFailureHandlerByOperationId("getPets", handler)`	`routerBuilder.operation("getPets").failureHandler(handler`

Old `OpenAPI3RouterFactory` methods	New `RouterBuilder` methods
`routerFactory.mountOperationToEventBus("getPets", "getpets.myapplication")`	`routerBuilder.operation("getPets").routeToEventBus("getpe`
`routerFactory.addGlobalHandler(handler)`	`routerBuilder.rootHandler(handler)`
`routerFactory.addBodyHandler(handler)`	`routerBuilder.bodyHandler(handler)`
`routerFactory.getRouter()`	`routerBuilder.createRouter()`

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();
```

Handling security

In 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 (https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.md#operationObject).

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)`

Accessing the OpenAPI contract model

In 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"));
```

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 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.

Updates in the 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()
  ).handler(
    RouteToEBServiceHandler.build(eventBus, "transactions.myapplication", "getTransactionsList")
  );
```

The `vertx-web-api-service` module does not support `vertx-web-api-contract` . So, when you upgrade to Vert.x 4, you must migrate your Vert.x OpenAPI applications to `vertx-web-openapi` module.

Changes in microservices patterns

This section explains the changes in microservices patterns.

Changes in Vert.x circuit breaker

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

Removed execute command methods in circuit breaker

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

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

Changes in Vert.x service discovery

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

Removed create methods from service discovery that contain `ServiceDiscovery` argument

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

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

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>`.

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 Vert.x 4, this bridge is no longer registered automatically. Even if you have added the bridge in the classpath of your 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);
```

Changes in Vert.x authentication and authorization

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

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

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

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

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 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 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 Vert.x 4.

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()) {
     // Never needs type casting
     User user = res.result();
   }
});
```

Migrating the authorization applications

Authorization is a new feature in 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 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 Vert.x 3.x releases.

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");
    }
  }
});
```

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 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.

Changes in key management

In 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.

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 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 Vert.x 4.

JAVA

```
new PubSecKeyOptions()
    .setAlgorithm("HS256")
    .setBuffer("password")
```

Updates in public secret keys management

In 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 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 Vert.x 4, you must specify both the public and private key.

The following example shows how to configure key pair in Vert.x 4.

JAVA

```
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);
```

You can now handle X509 certificates using `PubSecKeyOptions` .

JAVA

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

Changes in keystore management

In 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 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 Vert.x 3.x, all keystore entries would share the same password, that is, the keystore password. In 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 Vert.x 3.x.

JAVA

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

In 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 Vert.x 4.

JAVA

```
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("keystore-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");
```

Deprecated and removed authentication and authorization methods

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

List of removed authentication and authorization methods

The following methods have been removed:

Removed methods	Replacing methods
`OAuth2Auth.createKeycloak()`	`KeycloakAuth.create(vertx, JsonObject) ()`
`OAuth2Auth.create(Vertx, OAuth2FlowType, OAuth2ClientOptions)()`	`OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))`

Removed methods	Replacing methods
`OAuth2Auth.create(Vertx, OAuth2FlowType)`	`OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))`
`User.isAuthorised()`	`User.isAuthorized()`
`User.setAuthProvider()`	No replacing method
`AccessToken.refreshToken()`	`AccessToken.opaqueRefreshToken()`
`io.vertx.ext.auth.jwt.JWTOptions` data object	`io.vertx.ext.jwt.JWTOptions` data object
`Oauth2ClientOptions.isUseAuthorizationHeader()`	No replacing method
`Oauth2ClientOptions.scopeSeparator()`	No replacing method

List of deprecated authentication and authorization methods

The following methods have been deprecated:

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

List of deprecated authentication and authorization classes

The following classes have been deprecated:

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

Changes in protocols

This section explains the changes in networking protocols.

Changes in Vert.x gRPC

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

New gRPC compiler plugin

In 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 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 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 4.0 -->
<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>
  </configuration>
  ...
</plugin>
```

Migrating the generated code

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

For example, if you have the following descriptor:

```proto
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
```

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

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

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

JAVA
```
// 4.x
VertxGreeterGrpc.GreeterVertxImplBase service =
  new VertxGreeterGrpc.GreeterVertxImplBase() {
      ...
  }
```

gRPC APIs support futures

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

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

JAVA
```
// 3.x
GreeterGrpc.GreeterVertxImplBase service =
  new GreeterGrpc.GreeterVertxImplBase() {
        @Override
        public void sayHello(HelloRequest request, Promise<HelloReply> future) {
          future.complete(
              HelloReply.newBuilder().setMessage(request.getName()).build());
        }
  }
```

The following example shows the use of futures in 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());
      }
  }
```

Changes in Vert.x MQTT

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

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 Vert.x 4.

The following example shows the use of `publish()` method in 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 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);
```

`MqttWill` messages return buffer

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

Removed the deprecated `MqttWill` and authorization methods from MQTT

The following MQTT methods have been removed:

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

Changes in Vert.x Service Proxy

The following section describes the changes in service proxy.

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:

JAVA

```
<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.

Changes in client components

This section explains the changes in Vert.x clients.

Changes in Vert.x Kafka client

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

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.

Flush methods use asynchronous handler

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

Changes in Vert.x JDBC client

From 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 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 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 Vert.x JDBC client.

Creating a pool

In 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 Vert.x 3.x.

JAVA

```
// 3.x
SQLClient client = JDBCClient.create(vertx, jsonConfig);
```

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

JAVA

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

NOTE

Though the Vert.x 3.x APIs are supported in 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 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 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();
  });
```

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
      new JDBCConnectOptions()
        // H2 connection string
        .setJdbcUrl("jdbc:h2:~/test")
        // username
        .setUser("sa")
        // password
        .setPassword(""),
      // configure the pool
      new PoolOptions()
        .setMaxSize(16)
    );
```

NOTE

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.

Running SQL queries

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

Running one shot queries

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

JAVA

```
// 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 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"));
    }
  });
```

Running queries on managed connections

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

JAVA

```
pool
  .getConnection()
  .onFailure(e -> {
    // Failed to get a connection
  })
  .onSuccess(conn -> {
    conn
      .query("SELECT * FROM user")
      .execute()
      .onFailure(e -> {
        // Handle the failure
        // Important, do not forget to return the connection
        conn.close();
      })
      .onSuccess(rows -> {
        for (Row row : rows) {
          System.out.println(row.getString("FIRST_NAME"));
        }
        // Important, do not forget to return the connection
        conn.close();
      });
  });
```

Support for stored procedures

Stored procedures are supported in the JDBC client.

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

JAVA

```
// 3.x
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 Vert.x 4.

JAVA

```
// 4.0
client
  .preparedQuery("{call new_customer(?, ?)}")
  .execute(Tuple.of("Paulo", "Lopes"))
  .onSuccess(rows -> {
    ...
  });
```

In Vert.x 3.x, the support for combining the `IN` and `OUT` arguments was very limited due to the available types. In 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 Vert.x 3.x.

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 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 `OUT`, you can specify its return type. In the example, the `OUT` argument is specified as type `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.

Changes in Vert.x mail client

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

`MailAttachment` is available as an interface

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

Mail configuration interface extends the net client options

`MailConfig` interface extends the `NetClientOptions` interface. Due to this extension, mail configuration also supports the proxy setting of the `NetClient`.

Changes in Vert.x AMQP client

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

Removed methods in AMQP client that contain `AmqpMessage` argument

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.

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

`AmqpConnection createReceiver(.., Handler<AmqpMessage> messageHandler, Handler<AsyncResult<AmqpReceiver>> completionHandler)`	`AmqpConnection createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)`

Changes in Vert.x MongoDB client

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

Methods removed from MongoDB client

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

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

Changes in EventBus JavaScript client

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

- Maven Central (https://repo1.maven.org/maven2/io/vertx/vertx-web/3.9.4/vertx-web-3.9.4-client.js)
- NPM (https://www.npmjs.com/package/vertx3-eventbus-client)
- Bower.io (https://bower.io/search/?q=vertx3-eventbus-client)
- CDNJS (https://cdnjs.com/libraries/vertx)
- webjars (https://www.webjars.org)

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

- CDN (https://unpkg.io/@vertx/eventbus-bridge-client.js@1.0.0/vertx-eventbus.js)
- npm packages (https://www.npmjs.com/package/@vertx/eventbus-bridge-client.js)

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

  - JSON scripts

```
{
  "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>
```

Versioning of JavaScript client

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

However, from 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 Vert.x release, unless there is a version change.

Changes in Vert.x Redis client

In Vert.x 4, use the `Redis` class to work with Redis client. The class `RedisClient` is no longer available.

NOTE

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

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.

Creating the client

The following example shows how a Redis client was created in 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 Vert.x 4.

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

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

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

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 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 Vert.x 4. The example uses a list for setting the key-value pairs instead of hard coding options. See Redis SET command (https://redis.io/commands/set) 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());
        }
      });
    }
});
```

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 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 Vert.x 4.

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

Migrating responses

In 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 closet 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
```

Updates in Vert.x Redis client

This section describes changes in Redis client.

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 you usage of `SLAVE` role in Vert.x 3.x releases.

```
// Before (3.x)
Redis.createClient(
      rule.vertx(),
      new RedisOptions()
        .setType(RedisClientType.SENTINEL)
        .addConnectionString("redis://localhost:5000")
        .setMasterName("sentinel7000")
        .setRole(RedisRole.SLAVE));
```

The following example shows you usage of `REPLICA` role in Vert.x 4.

```
// After (4.0)
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 Vert.x 3.x releases.

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

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

```
// After (4.0)
options.setUseReplicas(RedisReplicas);
```

Changes in Vert.x Cassandra client

The Vert.x Cassandra client enables applications to interact with an Apache Cassandra service.

The following section describes the changes in Vert.x Cassandra client.

Changes in driver versions

In Vert.x 4, the Cassandra client uses Datastax Java driver with version 4.x. This driver version is not backward compatible. You will have to migrate your Vert.x 3.x applications to Vert.x 4. As a part of the migration process, you should import new Datastax driver classes and change the way you create statements.

The following example shows Datastax Java driver classes and create statements in Vert.x 3.x releases.

```
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;
import com.datastax.driver.core.SimpleStatement;
import io.vertx.cassandra.CassandraClient;

SimpleStatement statement =
  new SimpleStatement("SELECT release_version FROM system.local");
cassandraClient.execute(statement, done -> {
    ResultSet results = done.result();
    Row row = resultSet.one();
    System.out.println(row.getString("release_version"));
});
```

The following example shows Datastax Java driver classes and create statements in Vert.x 4.

```
import com.datastax.oss.driver.api.core.cql.ResultSet;
import com.datastax.oss.driver.api.core.cql.Row;
import com.datastax.oss.driver.api.core.cql.SimpleStatement;
import io.vertx.cassandra.CassandraClient;

SimpleStatement statement =
  SimpleStatement.newInstance("SELECT release_version FROM system.local");
cassandraClient.execute(statement, done -> {
    ResultSet results = done.result();
    Row row = resultSet.one();
    System.out.println(row.getString("release_version"));
});
```

Objects mappers are not supported

Object mappers are no longer supported because of changes in Datastax Java driver 4.x.

The following mapping construction is not supported in Vert.x 4 release:

```
@Table(keyspace = "test", name = "names")
public class MappedClass {
  @PartitionKey
  private String name;
   ...
}
```

Updates in Cassandra client options

The `CassandraClientOptions.setPort()`, `CassandraClientOptions.getContactPoints()`, and `CassandraClientOptions.setContactPoints()` methods have been removed. Use the `CassandraClientOptions.addContactPoint()` method instead.

The following example shows the use of older methods in Vert.x 3.x releases.

```
import io.vertx.cassandra.CassandraClient;
import io.vertx.cassandra.CassandraClientOptions;
CassandraClientOptions options = new CassandraClientOptions()
  .setContactPoints(List.of("node1.address", "node2.address", "node3.address"))
  .setPort(9142)
  .setKeyspace("my_keyspace");
CassandraClient client = CassandraClient.create(vertx, options);
```

The following example shows how to use `CassandraClientOptions.addContactPoint()`method in Vert.x 4.

```
import io.vertx.cassandra.CassandraClient;
import io.vertx.cassandra.CassandraClientOptions;
CassandraClientOptions options = new CassandraClientOptions()
  .addContactPoint("node1.address", 9142)
  .addContactPoint("node2.address", 9142)
  .addContactPoint("node3.address", 9142)
  .setKeyspace("my_keyspace");
CassandraClient client = CassandraClient.create(vertx, options);
```

RabbitMQ Client

RabbitMQOptions changes

In Vert.x 4 The RabbitMQ options inherits from NetClientOptions to leverage common connection options including SSL. Two options have been removed in favor for their base class equivalent, hence they have been renamed.

JAVA

```java
// Vert.x RabbitMQOptions  3.x:

RabbitMQOptions options = new RabbitMQOptions()
  .setConnectionRetries(42)
  .setConnectionRetryDelay(500);


// Vert.x RabbitMQOptions 4.x:
RabbitMQOptions options = new RabbitMQOptions()
  .setReconnectAttempts(42)
  .setReconnectInterval(500);
```

Changes in clustering

This section explains the changes in clustering.

Clustered flag removed from options classes

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

Changes in Hazelcast cluster manager

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

Updates in custom configurations

The Hazelcast cluster manager is based on Hazelcast 4.

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

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

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

- Add the map `__vertx.nodeInfo` .

```
<map name="__vertx.nodeInfo">
  <backup-count>1</backup-count>
</map>
```

Changes in Infinispan cluster manager

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

Updates in custom configurations

The Infinispan cluster manager is based on Infinispan 12.

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

The 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 Vert.x 3.x application:

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

```
<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 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.

Changes in Apache Ignite cluster manager

The Vert.x Ignite cluster manager has been rewritten to support the new Clustering SPI in Vert.x 4. This should not have any notable difference for the developer.

The second change was about the configuration. The reason for doing so was to reduce the transient dependencies introduced by the `ignite-spring` extension which was used to parse XML configurations. This change removes more than 10MB of assets from the dependencies.

If the cluster manager is used without any configuration changes there is no migration required but if there was a custom `ignite.xml` used, it can be either switched to the new `ignite.json` format or the `ignite-spring` dependency can be added back to you classpath.

Migrate to custom ignite.json

To overide the `default-ignite.json` you can simply add `ignite.json` to your classpath. The json format supports customizing ignite options, cache options, discovery options and ssl options. For further customization you need to fall back to the old XML format.

In the example below the default config is extended to activate TLS for cluster communication.

JSON

```
{
  "cacheConfiguration": [{
    "name": "__vertx.*",
    "cacheMode": "REPLICATED",
    "readFromBackup": false,
    "atomicityMode": "ATOMIC",
    "writeSynchronizationMode": "FULL_SYNC"
  }, {
    "name": "*",
    "cacheMode": "PARTITIONED",
    "backups": 1,
    "readFromBackup": false,
    "atomicityMode": "ATOMIC",
    "writeSynchronizationMode": "FULL_SYNC"
  }],
  "sslContextFactory": {
    "protocol": "TLSv1.2",
    "keyStoreFilePath": "server.jks",
    "keyStorePassword": "changeme",
    "trustStoreFilePath": "server.jks",
    "trustStorePassword": "changeme",
    "trustAll": false
  },
  "includeEventTypes": ["EVT_CACHE_OBJECT_PUT", "EVT_CACHE_OBJECT_REMOVED"],
  "metricsLogFrequency": 0
}
```

Migrate back to ignite.xml with ignite-spring

Add the `ignite-spring` dependency and provide the `ignite.xml` configuration to your classpath.

- Maven (in your `pom.xml`):

XML

```
<dependency>
  <groupId>org.apache.ignite</groupId>
  <artifactId>ignite-spring</artifactId>
  <version>${ignite.version}</version>
</dependency>
```

- Gradle (in your `build.gradle` file):

GROOVY

```
compile 'org.apache.ignite:ignite-spring:${ignite.version}'
```

Example from the original `default-ignite.xml` for this ClusterManager:

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/util
                           http://www.springframework.org/schema/util/spring-util.xsd">

  <bean class="org.apache.ignite.configuration.IgniteConfiguration">

    <property name="discoverySpi">
      <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
        <property name="ipFinder">
          <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast.TcpDiscoveryMulticastIpFinder"/>
        </property>
      </bean>
    </property>

    <property name="cacheConfiguration">
      <list>
        <bean class="org.apache.ignite.configuration.CacheConfiguration">
          <property name="name" value="__vertx.*"/>
          <property name="cacheMode" value="REPLICATED"/>
          <property name="readFromBackup" value="false"/>
          <property name="atomicityMode" value="ATOMIC"/>
          <property name="writeSynchronizationMode" value="FULL_SYNC"/>
        </bean>
        <bean class="org.apache.ignite.configuration.CacheConfiguration">
          <property name="name" value="*"/>
          <property name="cacheMode" value="PARTITIONED"/>
          <property name="backups" value="1"/>
          <property name="readFromBackup" value="false"/>
          <property name="atomicityMode" value="ATOMIC"/>
          <property name="affinity">
            <bean class="org.apache.ignite.cache.affinity.rendezvous.RendezvousAffinityFunction">
              <property name="partitions" value="128"/>
            </bean>
          </property>
          <property name="writeSynchronizationMode" value="FULL_SYNC"/>
        </bean>
      </list>
    </property>

    <property name="includeEventTypes">
      <list>
        <util:constant static-field="org.apache.ignite.events.EventType.EVT_CACHE_OBJECT_PUT"/>
        <util:constant static-field="org.apache.ignite.events.EventType.EVT_CACHE_OBJECT_REMOVED"/>
      </list>
    </property>

    <property name="gridLogger">
      <bean class="io.vertx.spi.cluster.ignite.impl.VertxLogger"/>
    </property>

    <property name="metricsLogFrequency" value="0"/>
  </bean>
</beans>
```

Migrating clusters

It is important to decide the migration strategy for your codebase. This is because you cannot add Vert.x 3.x nodes and 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 - Vert.x has changed the format of the EventBus subscription data stored in cluster managers.
- Transport protocol changes - Vert.x has changed some fields in the message transport protocol in the cluster.

If you have an 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 Vert.x 3.x codebase to Vert.x 4.

Splitting the cluster

If you have a cluster where different teams have deployed verticles for their applications, you can consider splitting the 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 Vert.x 4 when they are ready or if required.

Using Vert.x EventBus Link

If you cannot split your cluster, then use Vert.x EventBus Link (https://github.com/vert-x3/vertx-eventbus-link) to migrate your codebase incrementally.

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

> **WARNING** 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 Vert.x EventBus Link creates a WebSocket server to receive messages and uses a WebSocket client to send them.

See the sections get started (https://github.com/vert-x3/vertx-eventbus-link#getting-started) and using (https://github.com/vert-x3/vertx-eventbus-link#using) for more details.

Miscellaneous changes in Vert.x

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

Removed the `Starter` class

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

Isolated deployment for Java 8

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

Removed hook methods from 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`.

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.

Removed equals and hashcode methods from options

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

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.

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.

Removed the pooled buffer methods

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

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.

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

Changes in Vert.x JUnit5

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

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.
- `reactiverse-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.

Deprecated succeeding and failing methods in Vert.x text context

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

Kotlin changes

Generated coroutine extension methods

Vert.x 3 generates extension suspending extension methods, e.g `get(Handler<AsyncResult<Result>>` generates `getAwait()` extension method.

```
KOTLIN
// getSomethingAwait() is a generated extension method
// and is deprecated
var result = getSomethingAwait();
```

Vert.x 4 provides a future based model and the Vert.x `Future` has an `await()` extension methods that does the same. The code above can simply be rewritten as

```
KOTLIN
var result = getSomething().await();
```

Script compiler

Vert.x 3 provides a Kotlin Script compiler based on Kotlin Compiler API.

The API often breaks and creates friction for upgrading Kotlin versions.

This is removed in Vert.x 4.0.

Such scripts should now be handled by Kotlin tools, e.g the Kotlin compiler has options to compile these scripts.

Testing Mongo DB Code

The vertx-embedded-mongo-db (https://github.com/vert-x3/vertx-embedded-mongo-db) module is no longer supported in Vert.x 4. An alternative can be vertx-wiremongo (https://github.com/NoEnv/vertx-wiremongo) although it takes a different approach. While vertx-embedded-mongo-db spun up a full blown mongo server, vertx-wiremongo is an alternative in-memory implementation of Vert.x' `MongoClient` interface that allows specific mocking.

Let's assume we want to test our persistence layer's `findApples` method to make sure it creates the correct mongo query. Here's what such a test may look like with vertx-wiremongo:

JAVA

```java
@Test
public void testFindApples(TestContext ctx) {

  wiremongo.find()
    .inCollection("warehouse")
    .withQuery(q -> q.getString("type").equals("apple"))
    .returns(List.of(new JsonObject()
      .put("type", "apple")
      .put("mass", 161)
      .put("expiration", new JsonObject()
        .put("$date", Instant.now().plus(7, DAYS)))));

  db.findApples()
    .onSuccess(l -> {
      ctx.assertEquals(1, l.size());
      ctx.assertEquals(161, l.get(0).getInteger("mass"));
    })
    .onComplete(ctx.asyncAssertSuccess());
}
```

A complete documentation can be found here (https://github.com/NoEnv/vertx-wiremongo).

Last updated 2024-09-22 04:01:10 UTC