Red Hat build of Apache Camel Extensions for Quarkus 2.13

Camel Extensions for Quarkus Reference

Camel Extensions for Quarkus provided by Red Hat

Last Updated: 2023-06-22

Red Hat build of Apache Camel Extensions for Quarkus 2.13 Camel Extensions for Quarkus Reference

Camel Extensions for Quarkus provided by Red Hat

Abstract

Camel Extensions for Quarkus provides Quarkus extensions for many of the Camel components. This reference describes the settings for each of the extensions supported by Red Hat.

Table of Contents

PREFACE

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. EXTENSIONS OVERVIEW

1.1. SUPPORT LEVEL DEFINITIONS

New features, services, and components go through a number of support levels before inclusion in Camel Extensions for Quarkus as fully supported for production use. This is to ensure the right balance between providing the enterprise stability expected of our offerings with the need to allow our customers and partners to experiment with new Camel Extensions for Quarkus technologies while providing feedback to help guide future development activities.

Table 1.1. Camel Extensions for Quarkus support levels

Type	Description
Community Support	As part of Red Hat's commitment to upstream first, integration of new extensions into our Camel Extensions for Quarkus distribution begins in the upstream community. While these extensions have been tested and documented upstream, we have not reviewed the maturity of these extensions and they may not be formally supported by Red Hat in future product releases. **NOTE** Community extensions are listed on the extensions reference page of the Camel Quarkus community project.
Technology Preview	Technology Preview features provide early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. However, these features are not fully supported under Red Hat Subscription Level Agreements, may not be functionally complete, and are not intended for production use. As Red Hat considers making future iterations of Technology Preview features generally available, we will attempt to resolve any issues that customers experience when using these features.
Production Support	Production Support extensions are shipped in a formal Red Hat release and are fully supported. There are no documentation gaps and extensions have been tested on all supported configurations.

1.2. SUPPORTED EXTENSIONS

There are 60 extensions.

Table 1.2. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
AMQP	camel-quarkus-amqp	Production Support	Production Support	Messaging with AMQP protocol using Apache QPid Client.
Attachments	camel-quarkus-attachments	Production Support	Production Support	Support for attachments on Camel messages.
AWS 2 DynamoDB	camel-quarkus-aws2-ddb	Production Support	Production Support	Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream using AWS SDK version 2.x.
AWS 2 Kinesis	camel-quarkus-aws2-kinesis	Production Support	Production Support	Consume and produce records from AWS Kinesis Streams using AWS SDK version 2.x.
AWS 2 Lambda	camel-quarkus-aws2-lambda	Production Support	Production Support	Manage and invoke AWS Lambda functions using AWS SDK version 2.x.
AWS 2 S3 Storage Service	camel-quarkus-aws2-s3	Production Support	Production Support	Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.
AWS 2 Simple Notification System (SNS)	camel-quarkus-aws2-sns	Production Support	Production Support	Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.
AWS 2 Simple Queue Service (SQS)	camel-quarkus-aws2-sqs	Production Support	Production Support	Sending and receive messages to/from AWS SQS service using AWS SDK version 2.x.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Azure Storage Blob	camel-quarkus-azure-storage-blob	Technology Preview	Technology Preview	Store and retrieve blobs from Azure Storage Blob Service using SDK v12.
Azure Storage Queue	camel-quarkus-azure-storage-queue	Technology Preview	Technology Preview	The azure-storage-queue component is used for storing and retrieving the messages to/from Azure Storage Queue using Azure SDK v12.
Bean	camel-quarkus-bean	Production Support	Production Support	Invoke methods of Java beans
Bean Validator	camel-quarkus-bean-validator	Production Support	Production Support	Validate the message body using the Java Bean Validation API.
Browse	camel-quarkus-browse	Production Support	Production Support	Inspect the messages received on endpoints supporting BrowsableEndpoint.
Cassandra CQL	camel-quarkus-cassandraql	Production Support	Production Support	Integrate with Cassandra 2.0 using the CQL3 API (not the Thrift API). Based on Cassandra Java Driver provided by DataStax.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Core	camel-quarkus-core	Production Support	Production Support	Camel core functionality and basic Camel languages: Constant, ExchangeProperty, Header, Ref, Ref, Simple and Tokeinze
Cron	camel-quarkus-cron	Production Support	Production Support	A generic interface for triggering events at times specified through the Unix cron syntax.
CXF	camel-quarkus-cxf-soap	Technology Preview	Technology Preview	Expose SOAP WebServices using Apache CXF or connect to external WebServices using CXF WS client.
Dataformat	camel-quarkus-dataformat	Production Support	Production Support	Use a Camel Data Format as a regular Camel Component.
Direct	camel-quarkus-direct	Production Support	Production Support	Call another endpoint from the same Camel Context synchronously.
FHIR	camel-quarkus-fhir	Production Support	Production Support	Exchange information in the healthcare domain using the FHIR (Fast Healthcare Interoperability Resources) standard.
File	camel-quarkus-file	Production Support	Production Support	Read and write files.

Extension	Artifact	JVM Support Level	Native Support Level	Description
FTP	camel-quarkus-ftp	Production Support	Production Support	Upload and download files to/from FTP or SFTP servers.
Google BigQuery	camel-quarkus-google-bigquery	Production Support	Production Support	Google BigQuery data warehouse for analytics.
Google Pubsub	camel-quarkus-google-pubsub	Production Support	Production Support	Send and receive messages to/from Google Cloud Platform PubSub Service.
HTTP	camel-quarkus-http	Production Support	Production Support	Send requests to external HTTP servers using Apache HTTP Client 4.x.
Infinispan	camel-quarkus-infinispan	Production Support	Production Support	Read and write from/to Infinispan distributed key/value store and data grid.
JDBC	camel-quarkus-jdbc	Production Support	Production Support	Access databases through SQL and JDBC.
Jira	camel-quarkus-jira	Production Support	Production Support	Interact with JIRA issue tracker.
JMS	camel-quarkus-jms	Production Support	Production Support	Send and receive messages to/from a JMS Queue or Topic.
JPA	camel-quarkus-jpa	Production Support	Production Support	Store and retrieve Java objects from databases using Java Persistence API (JPA).

Extension	Artifact	JVM Support Level	Native Support Level	Description
JTA	camel-quarkus-jta	Production Support	Production Support	Enclose Camel routes in the transactions using Java Transaction API (JTA) and Narayana transaction manager
Kafka	camel-quarkus-kafka	Production Support	Production Support	Sent and receive messages to/from an Apache Kafka broker.
Kamelet	camel-quarkus-kamelet	Production Support	Production Support	The Kamelet Component provides support for interacting with the Camel Route Template engine.
Kubernetes	camel-quarkus-kubernetes	Technology Preview	Technology Preview	Perform operations against Kubernetes API
Log	camel-quarkus-log	Production Support	Production Support	Log messages to the underlying logging mechanism.
Mail	camel-quarkus-mail	Production Support	Production Support	Send and receive emails using imap, pop3 and smtp protocols.
MicroProfile Fault Tolerance	camel-quarkus-microprofile-fault-tolerance	Production Support	Production Support	Circuit Breaker EIP using MicroProfile Fault Tolerance.
MicroProfile Health	camel-quarkus-microprofile-health	Production Support	Production Support	Bridging Eclipse MicroProfile Health with Camel health checks.

Extension	Artifact	JVM Support Level	Native Support Level	Description
MicroProfile Metrics	camel-quarkus-microprofile-metrics	Production Support	Production Support	Expose metrics from Camel routes.
MLLP	camel-quarkus-mllp	Production Support	Production Support	Communicate with external systems using the MLLP protocol.
Mock	camel-quarkus-mock	Production Support	Production Support	Test routes and mediation rules using mocks.
MongoDB	camel-quarkus-mongodb	Technology Preview	Technology Preview	Perform operations on MongoDB documents and collections.
Netty	camel-quarkus-netty	Production Support	Production Support	Socket level networking using TCP or UDP with the Netty 4.x.
OpenAPI Java	camel-quarkus-openapi-java	Production Support	Production Support	Rest-dsl support for using OpenAPI doc
Paho	camel-quarkus-paho	Production Support	Production Support	Communicate with MQTT message brokers using Eclipse Paho MQTT Client.
Paho MQTT 5	camel-quarkus-paho-mqtt5	Production Support	Production Support	Communicate with MQTT message brokers using Eclipse Paho MQTT v5 Client.
Platform HTTP	camel-quarkus-platform-http	Production Support	Production Support	Expose HTTP endpoints using the HTTP server available in the current platform.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Quartz	camel-quarkus-quartz	Production Support	Production Support	Schedule sending of messages using the Quartz 2.x scheduler.
Rest	camel-quarkus-rest	Production Support	Production Support	Expose REST services and their OpenAPI Specification or call external REST services.
REST OpenApi	camel-quarkus-rest-openapi	Production Support	Production Support	Configure REST producers based on an OpenAPI specification document delegating to a component implementing the RestProducerFactory interface.
Salesforce	camel-quarkus-salesforce	Production Support	Production Support	Communicate with Salesforce using Java DTOs.
SEDA	camel-quarkus-seda	Production Support	Production Support	Asynchronously call another endpoint from any Camel Context in the same JVM.
Slack	camel-quarkus-slack	Technology Preview	Technology Preview	Send and receive messages to/from Slack.
SQL	camel-quarkus-sql	Production Support	Production Support	Perform SQL queries using Spring JDBC.
Telegram	camel-quarkus-telegram	Technology Preview	Technology Preview	Send and receive messages acting as a Telegram Bot API.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Timer	camel-quarkus-timer	Production Support	Production Support	Generate messages in specified intervals using java.util.Timer.
Validator	camel-quarkus-validator	Production Support	Production Support	Validate the payload using XML Schema and JAXP Validation.
Vert.x HTTP Client	camel-quarkus-vertx-http	Production Support	Production Support	Send requests to external HTTP servers using Vert.x
XQuery	camel-quarkus-saxon	Production Support	Production Support	Query and/or transform XML payloads using XQuery and Saxon.
Zip File	camel-quarkus-zipfile	Production Support	Production Support	Generate messages in specified intervals using java.util.Timer.

1.3. SUPPORTED DATA FORMATS

There are 10 data formats.

Table 1.3. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
Avro	camel-quarkus-avro	Production Support	Production Support	Serialize and deserialize messages using Apache Avro binary data format.
Avro Jackson	camel-quarkus-jackson-avro	Production Support	Production Support	Marshal POJOs to Avro and back using Jackson.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Bindy	camel-quarkus-bindy	Production Support	Production Support	Marshal and unmarshal between POJOs and Comma separated values (CSV) format using Camel Bindy Marshal and unmarshal between POJOs and fixed field length format using Camel Bindy Marshal and unmarshal between POJOs and key-value pair (KVP) format using Camel Bindy
HL7	camel-quarkus-hl7	Production Support	Production Support	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
Jackson	camel-quarkus-jackson	Production Support	Production Support	Marshal POJOs to JSON and back using Jackson
JacksonXML	camel-quarkus-jacksonxml	Production Support	Production Support	Unmarshal a XML payloads to POJOs and back using XMLMapper extension of Jackson.
JAXB	camel-quarkus-jaxb	Production Support	Production Support	Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.
JSON Gson	camel-quarkus-gson	Production Support	Production Support	Marshal POJOs to JSON and back using Gson.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Protobuf Jackson	camel-quarkus-jackson-protobuf	Production Support	Production Support	Marshal POJOs to Protobuf and back using Jackson.
SOAP dataformat	camel-quarkus-soap	Production Support	Production Support	Marshal Java objects to SOAP messages and back.

1.4. SUPPORTED LANGUAGES

There are 12 languages.

Table 1.4. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
Bean method	camel-quarkus-bean	Production Support	Production Support	Invoke methods of Java beans
Constant	camel-quarkus-core	Production Support	Production Support	A fixed value set only once during the route startup.
ExchangeProperty	camel-quarkus-core	Production Support	Production Support	Get the value of named Camel Exchange property.
File	camel-quarkus-core	Production Support	Production Support	For expressions and predicates using the file/simple language.
Header	camel-quarkus-core	Production Support	Production Support	Get the value of the named Camel Message header.
HL7 Terser	camel-quarkus-hl7	Production Support	Production Support	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.

Extension	Artifact	JVM Support Level	Native Support Level	Description
JSON Path	camel-quarkus-jsonpath	Production Support	Production Support	Evaluate a JsonPath expression against a JSON message body.
Ref	camel-quarkus-core	Production Support	Production Support	Look up an expression in the Camel Registry and evaluate it.
Simple	camel-quarkus-core	Production Support	Production Support	Evaluate Camel's built-in Simple language expression against the Camel Exchange.
Tokenize	camel-quarkus-core	Production Support	Production Support	Tokenize text payloads using the specified delimiter patterns.
XPath	camel-quarkus-xpath	Production Support	Production Support	Evaluate an XPath expression against an XML payload.
XQuery	camel-quarkus-saxon	Production Support	Production Support	Query and/or transform XML payloads using XQuery and Saxon.

1.5. MISCELLANEOUS EXTENSIONS

There are 3 miscellaneous extensions.

Table 1.5. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
OpenTelemetry	camel-quarkus-opentelemetry	Technology Preview	Technology Preview	Distributed tracing using OpenTelemetry

Extension	Artifact	JVM Support Level	Native Support Level	Description
YAML DSL	camel-quarkus-yaml-dsl	Production Support	Production Support	A YAML stack for parsing YAML route definitions
XML DSL	camel-quarkus-xml-io-dsl	Production Support	Production Support	Camel XML DSL with camel-xml-io.

CHAPTER 2. EXTENSIONS REFERENCE

This chapter provides reference information about Camel Extensions for Quarkus.

2.1. AMQP

Messaging with AMQP protocol using Apache QPid Client.

2.1.1. What's inside

- AMQP component, URI syntax: **amqp:destinationType:destinationName**

Please refer to the above link for usage and configuration details.

2.1.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-amqp</artifactId>
</dependency>
```

2.1.3. Usage

2.1.3.1. Message mapping with **org.w3c.dom.Node**

The Camel AMQP component supports message mapping between **javax.jms.Message** and **org.apache.camel.Message**. When wanting to convert a Camel message body type of **org.w3c.dom.Node**, you must ensure that the **camel-quarkus-jaxp** extension is present on the classpath.

2.1.3.2. Native mode support for javax.jms.ObjectMessage

When sending JMS message payloads as **javax.jms.ObjectMessage**, you must annotate the relevant classes to be registered for serialization with **@RegisterForReflection(serialization = true)**. Note that this extension automatically sets **quarkus.camel.native.reflection.serialization-enabled = true** for you. Refer to the native mode user guide for more information.

2.1.4. Camel Quarkus limitations

2.1.4.1. Connection Pooling

JMS connection pooling isn't supported yet since there is still an open issue with quarkus-qpid-jms.

2.1.5. transferException option in native mode

To use the **transferException** option in native mode, you must enable support for object serialization. Refer to the native mode user guide for more information.

You will also need to enable serialization for the exception classes that you intend to serialize. For example.

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

2.1.6. Additional Camel Quarkus configuration

The extension leverages the Quarkus Qpid JMS extension. A ConnectionFactory bean is automatically created and wired into the AMQP component for you. The connection factory can be configured via the Quarkus Qpid JMS configuration options.

2.2. ATTACHMENTS

Support for attachments on Camel messages

2.2.1. What's inside

- Attachments

Please refer to the above link for usage and configuration details.

2.2.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-attachments</artifactId>
</dependency>
```

2.3. AVRO

Serialize and deserialize messages using Apache Avro binary data format.

2.3.1. What's inside

- Avro data format

Please refer to the above link for usage and configuration details.

2.3.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-avro</artifactId>
```

```
</dependency>
```

2.3.3. Additional Camel Quarkus configuration

Beyond standard usages known from vanilla Camel, Camel Quarkus adds the possibility to parse the Avro schema at build time both in JVM and Native mode.

The approach to generate Avro classes from Avro schema files is the one coined by the **quarkus-avro** extension. It requires the following:

1. Store ***.avsc** files in a folder named **src/main/avro** or **src/test/avro**
2. In addition to the usual **build** goal of **quarkus-maven-plugin**, add the **generate-code** goal:

```
<plugin>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>generate-code-and-build</id>
            <goals>
                <goal>generate-code</goal>
                <goal>build</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Please see a working configuration in Camel Quarkus Avro integration test and Quarkus Avro integration test.

2.4. AWS 2 DYNAMODB

Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream using AWS SDK version 2.x.

2.4.1. What's inside

- AWS DynamoDB component, URI syntax: **aws2-ddb:tableName**
- AWS DynamoDB Streams component, URI syntax: **aws2-ddbstream:tableName**

Please refer to the above links for usage and configuration details.

2.4.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-ddb</artifactId>
</dependency>
```

2.4.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.4.4. Additional Camel Quarkus configuration

2.4.4.1. Optional integration with Quarkus Amazon DynamoDB

If desired, it is possible to use the Quarkus Amazon DynamoDB extension in conjunction with Camel Quarkus AWS 2 DynamoDB. Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

   ```
   quarkus.dynamodb.sync-client.type=apache
   ```

2. The **DynamoDbClient** has to be made "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that e.g. by adding a dummy bean injecting **DynamoDbClient**:

   ```
   import javax.enterprise.context.ApplicationScoped;
   import io.quarkus.arc.Unremovable;
   import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

   @ApplicationScoped
   @Unremovable
   class UnremovableDynamoDbClient {
       @Inject
       DynamoDbClient dynamoDbClient;
   }
   ```

2.5. AWS 2 KINESIS

Consume and produce records from AWS Kinesis Streams using AWS SDK version 2.x.

2.5.1. What's inside

- AWS Kinesis component, URI syntax: **aws2-kinesis:streamName**
- AWS Kinesis Firehose component, URI syntax: **aws2-kinesis-firehose:streamName**

Please refer to the above links for usage and configuration details.

2.5.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-kinesis</artifactId>
```

```
</dependency>
```

2.5.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.6. AWS 2 LAMBDA

Manage and invoke AWS Lambda functions using AWS SDK version 2.x.

2.6.1. What's inside

- AWS Lambda component, URI syntax: **aws2-lambda:function**

Please refer to the above link for usage and configuration details.

2.6.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-lambda</artifactId>
</dependency>
```

2.6.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.6.4. Additional Camel Quarkus configuration

2.6.4.1. Not possible to leverage quarkus-amazon-lambda by Camel aws2-lambda extension

Quarkus-amazon-lambda extension allows you to use Quarkus to build your AWS Lambdas, whereas Camel component manages (deploy, undeploy, …) existing functions. Therefore, it is not possible to use **quarkus-amazon-lambda** as a client for Camel **aws2-lambda** extension.

2.7. AWS 2 S3 STORAGE SERVICE

Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.

2.7.1. What's inside

- AWS S3 Storage Service component, URI syntax: **aws2-s3://bucketNameOrArn**

Please refer to the above link for usage and configuration details.

2.7.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-s3</artifactId>
</dependency>
```

2.7.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.7.4. Additional Camel Quarkus configuration

2.7.4.1. Optional integration with Quarkus Amazon S3

If desired, it is possible to use the Quarkus Amazon S3 extension in conjunction with Camel Quarkus AWS 2 S3 Storage Service. Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

    ```
    quarkus.s3.sync-client.type=apache
    ```

2. The **S3Client** has to be made "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that e.g. by adding a dummy bean injecting **S3Client**:

    ```
    import javax.enterprise.context.ApplicationScoped;
    import io.quarkus.arc.Unremovable;
    import software.amazon.awssdk.services.s3.S3Client;

    @ApplicationScoped
    @Unremovable
    class UnremovableS3Client {
        @Inject
        S3Client s3Client;
    }
    ```

2.8. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS)

Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.

2.8.1. What's inside

- AWS Simple Notification System (SNS) component , URI syntax: **aws2-sns:topicNameOrArn**

Please refer to the above link for usage and configuration details.

2.8.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-sns</artifactId>
</dependency>
```

2.8.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.8.4. Additional Camel Quarkus configuration

2.8.4.1. Optional integration with Quarkus Amazon SNS

If desired, it is possible to use the Quarkus Amazon SNS extension in conjunction with Camel Quarkus AWS 2 Simple Notification System (SNS). Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

   ```
   quarkus.sns.sync-client.type=apache
   ```

2. The **SnsClient** has to be made "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that e.g. by adding a dummy bean injecting **SnsClient**:

   ```
   import javax.enterprise.context.ApplicationScoped;
   import io.quarkus.arc.Unremovable;
   import software.amazon.awssdk.services.sns.SnsClient;

   @ApplicationScoped
   @Unremovable
   class UnremovableSnsClient {
       @Inject
       SnsClient snsClient;
   }
   ```

2.9. AWS 2 SIMPLE QUEUE SERVICE (SQS)

Send and receive messages to/from AWS SQS service using AWS SDK version 2.x.

2.9.1. What's inside

- AWS Simple Queue Service (SQS) component, URI syntax: **aws2-sqs:queueNameOrArn**

Please refer to the above link for usage and configuration details.

2.9.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-sqs</artifactId>
</dependency>
```

2.9.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.9.4. Additional Camel Quarkus configuration

2.9.4.1. Optional integration with Quarkus Amazon SQS

If desired, it is possible to use the Quarkus Amazon SQS extension in conjunction with Camel Quarkus AWS 2 Simple Queue Service (SQS). Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

   ```
   quarkus.sqs.sync-client.type=apache
   ```

2. The **SqsClient** has to be made "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that e.g. by adding a dummy bean injecting **SqsClient**:

   ```
   import javax.enterprise.context.ApplicationScoped;
   import io.quarkus.arc.Unremovable;
   import software.amazon.awssdk.services.sqs.SqsClient;

   @ApplicationScoped
   @Unremovable
   class UnremovableSqsClient {
       @Inject
       SqsClient sqsClient;
   }
   ```

2.10. AZURE STORAGE BLOB SERVICE

Store and retrieve blobs from Azure Storage Blob Service using SDK v12.

2.10.1. What's inside

- Azure Storage Blob Service component, URI syntax: **azure-storage-blob:accountName/containerName**

Please refer to the above link for usage and configuration details.

2.10.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-azure-storage-blob</artifactId>
</dependency>
```

2.10.3. Usage

2.10.3.1. Micrometer metrics support

If you wish to enable the collection of Micrometer metrics for the Reactor Netty transports, then you should declare a dependency on **quarkus-micrometer** to ensure that they are available via the Quarkus metrics HTTP endpoint.

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

2.10.4. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.11. AZURE STORAGE QUEUE SERVICE

The azure-storage-queue component is used for storing and retrieving the messages to/from Azure Storage Queue using Azure SDK v12.

2.11.1. What's inside

- Azure Storage Queue Service component, URI syntax: **azure-storage-queue:accountName/queueName**

Please refer to the above link for usage and configuration details.

2.11.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-azure-storage-queue</artifactId>
</dependency>
```

2.11.3. Usage

2.11.3.1. Micrometer metrics support

If you wish to enable the collection of Micrometer metrics for the Reactor Netty transports, then you should declare a dependency on **quarkus-micrometer** to ensure that they are available via the Quarkus metrics HTTP endpoint.

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

2.11.4. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.12. BEAN VALIDATOR

Validate the message body using the Java Bean Validation API.

2.12.1. What's inside

- Bean Validator component, URI syntax: **bean-validator:label**

Please refer to the above link for usage and configuration details.

2.12.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-bean-validator</artifactId>
</dependency>
```

2.12.3. Usage

2.12.3.1. Configuring the ValidatorFactory

Implementation of this extension leverages the Quarkus Hibernate Validator extension.

Therefore it is not possible to configure the **ValidatorFactory** by Camel's properties (**constraintValidatorFactory**, **messageInterpolator**, **traversableResolver**, **validationProviderResolver** and **validatorFactory**).

You can configure the **ValidatorFactory** by the creation of beans which will be injected into the default **ValidatorFactory** (created by Quarkus). See the Quarkus CDI documentation for more information.

2.12.3.2. Custom validation groups in native mode

When using custom validation groups in native mode, all the interfaces need to be registered for reflection (see the documentation).

Example:

```
@RegisterForReflection
public interface OptionalChecks {
}
```

2.12.4. Camel Quarkus limitations

It is not possible to describe your constraints as XML (by providing the file META-INF/validation.xml), only Java annotations are supported. This is caused by the limitation of the Quarkus Hibernate Validator extension (see the issue).

2.13. BEAN

Invoke methods of Java beans

2.13.1. What's inside

- Bean component, URI syntax: **bean:beanName**
- Bean Method language
- Class component, URI syntax: **class:beanName**

Please refer to the above links for usage and configuration details.

2.13.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-bean</artifactId>
</dependency>
```

2.13.3. Usage

Except for invoking methods of beans available in Camel registry, Bean component and Bean method language can also invoke Quarkus CDI beans.

2.14. BINDY

Marshal and unmarshal between POJOs on one side and Comma separated values (CSV), fixed field length or key-value pair (KVP) formats on the other side using Camel Bindy

2.14.1. What's inside

- Bindy CSV data format
- Bindy Fixed Length data format
- Bindy Key Value Pair data format

Please refer to the above links for usage and configuration details.

2.14.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-bindy</artifactId>
</dependency>
```

2.14.3. Camel Quarkus limitations

When using camel-quarkus-bindy in native mode, only the build machine's locale is supported.

For instance, on build machines with french locale, the code below:

```
BindyDataFormat dataFormat = new BindyDataFormat();
dataFormat.setLocale("ar");
```

formats numbers the arabic way in JVM mode as expected. However, it formats numbers the french way in native mode.

Without further tuning, the build machine's default locale would be used. Another locale could be specified with the quarkus.native.user-language and quarkus.native.user-country configuration properties.

2.15. BROWSE

Inspect the messages received on endpoints supporting BrowsableEndpoint.

2.15.1. What's inside

- Browse component, URI syntax: **browse:name**

Please refer to the above link for usage and configuration details.

2.15.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-browse</artifactId>
</dependency>
```

2.16. CASSANDRA CQL

Integrate with Cassandra 2.0 using the CQL3 API (not the Thrift API). Based on Cassandra Java Driver provided by DataStax.

2.16.1. What's inside

- Cassandra CQL component, URI syntax: **cql:beanRef:hosts:port/keyspace**

Please refer to the above link for usage and configuration details.

2.16.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-cassandraql</artifactId>
</dependency>
```

2.16.3. Additional Camel Quarkus configuration

2.16.3.1. Cassandra aggregation repository in native mode

In order to use Cassandra aggregation repositories like **CassandraAggregationRepository** in native mode, you must enable native serialization support.

In addition, if your exchange bodies are custom types, then they must be registered for serialization by annotating their class declaration with **@RegisterForReflection(serialization = true)**.

2.17. CONTROL BUS

Manage and monitor Camel routes.

2.17.1. What's inside

- Control Bus component, URI syntax: **controlbus:command:language**

Please refer to the above link for usage and configuration details.

2.17.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-controlbus</artifactId>
</dependency>
```

2.17.3. Usage

2.17.3.1. Statistics

When using the **stats** command endpoint, the **camel-quarkus-management** extension must be added as a project dependency to enable JMX. Maven users will have to add the following to their **pom.xml**:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-management</artifactId>
</dependency>
```

2.17.3.2. Languages

The following languages are supported for use in the Control Bus extension in Camel Extensions for Quarkus:

- Bean language
- ExchangeProperty language
- Header language
- Simple language

2.17.3.2.1. Bean

The Bean language can be used to invoke a method on a bean to control the state of routes. The **org.apache.camel.quarkus:camel-quarkus-bean** extension must be added to the classpath. Maven users must add the following dependency to the POM:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-bean</artifactId>
</dependency>
```

In native mode, the bean class must be annotated with **@RegisterForReflection**.

2.17.3.2.2. Simple

The Simple language can be used to control the state of routes. The following example uses a **ProducerTemplate** to stop a route with the id **foo**:

```
template.sendBody(
    "controlbus:language:simple",
    "${camelContext.getRouteController().stopRoute('foo')}"
);
```

To use the OGNL notation, the **org.apache.camel.quarkus:camel-quarkus-bean** extension must be added as a dependency.

In native mode, the classes used in the OGNL notation must be registered for reflection. In the above code snippet, the **org.apache.camel.spi.RouteController** class returned from **camelContext.getRouteController()** must be registered. As this is a third-party class, it cannot be annotated with **@RegisterForReflection** directly - instead you can annotate a different class and specifying the target classes to register. For example, the class defining the Camel routes could be annotated with **@RegisterForReflection(targets = { org.apache.camel.spi.RouteController.class })**.

Alternatively, add the following line to your **src/main/resources/application.properties**:

```
quarkus.camel.native.reflection.include-patterns = org.apache.camel.spi.RouteController
```

2.17.4. Camel Quarkus limitations

2.17.4.1. Statistics

The **stats** action is not available in native mode as JMX is not supported on GraalVM. Therefore, attempting to build a native image with the **camel-quarkus-management** extension on the classpath will result in a build failure.

This feature is not supported in Camel Extensions for Quarkus.

2.18. CORE

Camel core functionality and basic Camel languages/ Constant, ExchangeProperty, Header, Ref, Simple and Tokenize

2.18.1. What's inside

- Constant language
- ExchangeProperty language
- File language
- Header language
- Ref language
- Simple language
- Tokenize language

Please refer to the above links for usage and configuration details.

2.18.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
```

```
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-core</artifactId>
</dependency>
```

2.18.3. Additional Camel Quarkus configuration

2.18.3.1. Simple language

2.18.3.1.1. Using the OGNL notation

When using the OGNL notation from the simple language, the **camel-quarkus-bean** extension should be used.

For instance, the simple expression below is accessing the **getAddress()** method on the message body of type **Client**.

```
---
simple("${body.address}")
---
```

In such a situation, one should take an additional dependency on the camel-quarkus-bean extension as described here. Note that in native mode, some classes may need to be registered for reflection. In the example above, the **Client** class needs to be registered for reflection.

2.18.3.1.2. Using dynamic type resolution in native mode

When dynamically resolving a type from simple expressions like:

- **simple("${mandatoryBodyAs(TYPE)}")**
- **simple("${type:package.Enum.CONSTANT}")**
- **from("…").split(bodyAs(TYPE.class))**
- **simple("${body} is TYPE")**

It may be needed to register some classes for reflection manually.

For instance, the simple expression below is dynamically resolving the type **java.nio.ByteBuffer** at runtime:

```
---
simple("${body} is 'java.nio.ByteBuffer'")
---
```

As such, the class **java.nio.ByteBuffer** needs to be registered for reflection.

2.18.3.1.3. Using the simple language with classpath resources in native mode

If your route is supposed to load a Simple script from classpath, like in the following example

```
from("direct:start").transform().simple("resource:classpath:mysimple.txt");
```

then you need to use Quarkus **quarkus.native.resources.includes** property to include the resource in the native executable as demonstrated below:

```
quarkus.native.resources.includes = mysimple.txt
```

2.18.3.1.4. Configuring a custom bean via properties in native mode

When specifying a custom bean via properties in native mode with configuration like **#class:*** or **#type:***, it may be needed to register some classes for reflection manually.

For instance, the custom bean definition below involves the use of reflection for bean instantiation and setter invocation:

```
---
camel.beans.customBeanWithSetterInjection =
#class:org.example.PropertiesCustomBeanWithSetterInjection
camel.beans.customBeanWithSetterInjection.counter = 123
---
```

As such, the class **PropertiesCustomBeanWithSetterInjection** needs to be registered for reflection, note that field access could be omitted in this case.

Configuration property	Type	Default
🔒 quarkus.camel.bootstrap.enabled When set to true, the **CamelRuntime** will be started automatically.	**boolean**	**true**
🔒 quarkus.camel.service.discovery.exclude-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will not be discoverable via the **org.apache.camel.spi.FactoryFinder** mechanism. The excludes have higher precedence than includes. The excludes defined here can also be used to veto the discoverability of services included by Camel Quarkus extensions. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	

Configuration property	Type	Default
🔒 quarkus.camel.service.discovery.include-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will be discoverable via the **org.apache.camel.spi.FactoryFinder** mechanism unless the given file is excluded via **exclude-patterns**. Note that Camel Quarkus extensions may include some services by default. The services selected here added to those services and the exclusions defined in **exclude-patterns** are applied to the union set. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	string	
🔒 quarkus.camel.service.registry.exclude-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will not be added to Camel registry during application's static initialization. The excludes have higher precedence than includes. The excludes defined here can also be used to veto the registration of services included by Camel Quarkus extensions. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	string	
🔒 quarkus.camel.service.registry.include-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will be added to Camel registry during application's static initialization unless the given file is excluded via **exclude-patterns**. Note that Camel Quarkus extensions may include some services by default. The services selected here added to those services and the exclusions defined in **exclude-patterns** are applied to the union set. Example values:**META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	string	
🔒 quarkus.camel.runtime-catalog.components If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel components available in the application; otherwise component JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	boolean	true

Configuration property	Type	Default
🔒 **quarkus.camel.runtime-catalog.languages** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel languages available in the application; otherwise language JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	boolean	true
🔒 **quarkus.camel.runtime-catalog.dataformats** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel data formats available in the application; otherwise data format JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	boolean	true
🔒 **quarkus.camel.runtime-catalog-models** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel EIP models available in the application; otherwise EIP model JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	boolean	true
🔒 **quarkus.camel.routes-discovery.enabled** Enable automatic discovery of routes during static initialization.	boolean	true
🔒 **quarkus.camel.routes-discovery.exclude-patterns** Used for exclusive filtering scanning of RouteBuilder classes. The exclusive filtering takes precedence over inclusive filtering. The pattern is using Ant-path style pattern. Multiple patterns can be specified separated by comma. For example to exclude all classes starting with Bar use: **/Bar* To exclude all routes form a specific package use: com/mycompany/bar/* To exclude all routes form a specific package and its sub-packages use double wildcards: com/mycompany/bar/** And to exclude all routes from two specific packages use: com/mycompany/bar/*,com/mycompany/stuff/*	string	

Configuration property	Type	Default
🔒 quarkus.camel.routes-discovery.include-patterns Used for inclusive filtering scanning of RouteBuilder classes. The exclusive filtering takes precedence over inclusive filtering. The pattern is using Ant-path style pattern. Multiple patterns can be specified separated by comma. For example to include all classes starting with Foo use: **/Foo* To include all routes form a specific package use: com/mycompany/foo/* To include all routes form a specific package and its sub-packages use double wildcards: com/mycompany/foo/** And to include all routes from two specific packages use: com/mycompany/foo/*,com/mycompany/stuff/*	string	
🔒 quarkus.camel.native.resources.exclude-patterns Replaced by **quarkus.native.resources.excludes** in Camel Quarkus 2.0.0. Using this property throws an exception at build time.	string	
🔒 quarkus.camel.native.resources.include-patterns Replaced by **quarkus.native.resources.includes** in Camel Quarkus 2.0.0. Using this property throws an exception at build time.	string	
🔒 quarkus.camel.native.reflection.exclude-patterns A comma separated list of Ant-path style patterns to match class names that should be excluded from registering for reflection. Use the class name format as returned by the **java.lang.Class.getName()** method: package segments delimited by period **.** and inner classes by dollar sign **$**. This option narrows down the set selected by**include-patterns**. By default, no classes are excluded. This option cannot be used to unregister classes which have been registered internally by Quarkus extensions.	string	

Configuration property	Type	Default
🔒 quarkus.camel.native.reflection.include-patterns A comma separated list of Ant-path style patterns to match class names that should be registered for reflection. Use the class name format as returned by the **java.lang.Class.getName()** method: package segments delimited by period **.** and inner classes by dollar sign **$**. By default, no classes are included. The set selected by this option can be narrowed down by **exclude-patterns**. Note that Quarkus extensions typically register the required classes for reflection by themselves. This option is useful in situations when the built in functionality is not sufficient. Note that this option enables the full reflective access for constructors, fields and methods. If you need a finer grained control, consider using **io.quarkus.runtime.annotations.RegisterForReflection** annotation in your Java code. For this option to work properly, at least one of the following conditions must be satisfied: - There are no wildcards (***** or **/**) in the patterns - The artifacts containing the selected classes contain a Jandex index (**META-INF/jandex.idx**) - The artifacts containing the selected classes are registered for indexing using the **quarkus.index-dependency.*** family of options in**application.properties** - e.g. quarkus.index-dependency.my-dep.group-id = org.my-group quarkus.index-dependency.my-dep.artifact-id = my-artifact where **my-dep** is a label of your choice to tell Quarkus that **org.my-group** and with **my-artifact** belong together.	**string**	
🔒 quarkus.camel.native.reflection.serialization-enabled If **true**, basic classes are registered for serialization; otherwise basic classes won't be registered automatically for serialization in native mode. The list of classes automatically registered for serialization can be found in CamelSerializationProcessor.BASE_SERIALIZATION_CLASSES. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **true** except for making the behavior consistent with native mode.	**boolean**	**false**

Configuration property	Type	Default
🔒 quarkus.camel.csimple.on-build-time-analysis-failure What to do if it is not possible to extract CSimple expressions from a route definition at build time.	org.apache.camel.quarkus.core.CamelConfig.FailureRem	warn

Configuration property	Type	Default
🔒 quarkus.camel.event-bridge.enabled Whether to enable the bridging of Camel events to CDI events. This allows CDI observers to be configured for Camel events. E.g. those belonging to the **org.apache.camel.quarkus.core.events**, **org.apache.camel.quarkus.main.events** & **org.apache.camel.impl.event** packages. Note that this configuration item only has any effect when observers configured for Camel events are present in the application.	**boolean**	**true**
🔒 quarkus.camel.source-location-enabled Build time configuration options for enable/disable camel source location	**boolean**	**false**
🔒 quarkus.camel.main.shutdown.timeout A timeout (with millisecond precision) to wait for **CamelMain#stop()** to finish	**java.time.Duration**	**PT3S**

Configuration property	Type	Default
🔒 **quarkus.camel.main.arguments.on-unknown** The action to take when **CamelMain** encounters an unknown argument. fail - Prints the **CamelMain** usage statement and throws a **RuntimeException** ignore - Suppresses any warnings and the application startup proceeds as normal warn - Prints the **CamelMain** usage statement but allows the application startup to proceed as normal	**org.apache.camel.quarkus.core.CamelConfig.FailureRem**	**warn**

Configuration property	Type	Default

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.19. CRON

A generic interface for triggering events at times specified through the Unix cron syntax.

2.19.1. What's inside

- Cron component, URI syntax: **cron:name**

Please refer to the above link for usage and configuration details.

2.19.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-cron</artifactId>
</dependency>
```

2.19.3. Additional Camel Quarkus configuration

The cron component is a generic interface component, as such Camel Quarkus users will need to use the cron extension together with another extension offering an implementation.

2.20. CXF

Expose SOAP WebServices using Apache CXF or connect to external WebServices using CXF WS client.

2.20.1. What's inside

- CXF component

There are two URI formats for this endpoint:

cxf:bean:cxfEndpoint

cxfEndpoint represents a bean ID that references a bean in the Spring bean registry.

cxf://someAddress

someAddress specifies the CXF endpoint's address.

See the CXF component for usage and configuration details.

2.20.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-cxf-soap</artifactId>
</dependency>
```

2.20.3. Usage

2.20.3.1. General

camel-quarkus-cxf-soap uses extensions from the CXF Extensions for Quarkus project - **quarkus-cxf**. This means that **quarkus-cxf** provides the set of supported use cases and WS specifications.

IMPORTANT

To learn about supported use cases and WS specifications, see the Quarkus CXF Reference.

2.20.3.2. Dependency management

Camel Extensions for Quarkus manages the CXF and **quarkus-cxf** versions. You do not need to select compatible versions for those projects.

2.20.3.3. Client

With **camel-quarkus-cxf-soap** (no additional dependencies required), you can use CXF clients as producers in Camel routes:

```
import org.apache.camel.builder.RouteBuilder;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.context.SessionScoped;
import javax.enterprise.inject.Produces;
import javax.inject.Named;

@ApplicationScoped
public class CxfSoapClientRoutes extends RouteBuilder {

    @Override
    public void configure() {

        /* You can either configure the client inline */
        from("direct:cxfUriParamsClient")
                .to("cxf://http://localhost:8082/calculator-ws?
wsdlURL=wsdl/CalculatorService.wsdl&dataFormat=POJO&serviceClass=org.foo.CalculatorService")
;

        /* Or you can use a named bean produced below by beanClient() method */
        from("direct:cxfBeanClient")
                .to("cxf:bean:beanClient?dataFormat=POJO");
    }

    @Produces
    @SessionScoped
    @Named
    CxfEndpoint beanClient() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(CalculatorService.class);
        result.setAddress("http://localhost:8082/calculator-ws");
        result.setWsdlURL("wsdl/CalculatorService.wsdl"); // a resource in the class path
        return result;
    }
}
```

The **CalculatorService** may look like the following:

```
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService(targetNamespace = CalculatorService.TARGET_NS)
public interface CalculatorService {

    public static final String TARGET_NS = "http://acme.org/wscalculator/Calculator";

    @WebMethod
    public int add(int intA, int intB);

    @WebMethod
    public int subtract(int intA, int intB);

    @WebMethod
    public int divide(int intA, int intB);

    @WebMethod
    public int multiply(int intA, int intB);
}
```

NOTE

JAX-WS annotations are required. The Simple CXF Frontend is not supported. Complex parameter types require JAXB annotations to work properly in native mode.

TIP

You can test this client application against the quay.io/l2x6/calculator-ws:1.2 container that implements this service endpoint interface:

```
$ docker run -p 8082:8080 quay.io/l2x6/calculator-ws:1.2
```

NOTE

quarkus-cxf supports injecting SOAP clients using **@io.quarkiverse.cxf.annotation.CXFClient** annotation. Refer to the SOAP Clients chapter of the **quarkus-cxf** user guide for more details.

2.20.3.4. Server

With **camel-quarkus-cxf-soap**, you can expose SOAP endpoints as consumers in Camel routes. No additional dependencies are required for this use case.

```
import org.apache.camel.builder.RouteBuilder;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Produces;
import javax.inject.Named;

@ApplicationScoped
public class CxfSoapRoutes extends RouteBuilder {

    @Override
    public void configure() {
        /* A CXF Service configured through a CDI bean */
        from("cxf:bean:helloBeanEndpoint")
                .setBody().simple("Hello ${body} from CXF service");

        /* A CXF Service configured through Camel URI parameters */
        from("cxf:///hello-inline?wsdlURL=wsdl/HelloService.wsdl&serviceClass=org.foo.HelloService")
                        .setBody().simple("Hello ${body} from CXF service");
    }

    @Produces
    @ApplicationScoped
    @Named
    CxfEndpoint helloBeanEndpoint() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(HelloService.class);
        result.setAddress("/hello-bean");
        result.setWsdlURL("wsdl/HelloService.wsdl");
        return result;
    }
}
```

The path for these two services depends on the value of the **quarkus.cxf.path** configuration property which can for example be set in **application.properties**:

application.properties

```
quarkus.cxf.path = /soap-services
```

With this configuration in place, you can access the two services at **http://localhost:8080/soap-services/hello-bean** and **http://localhost:8080/soap-services/hello-inline**, respectively.

You can access the WSDL by adding **?wsdl** to the above URLs.

IMPORTANT

Do not use **quarkus.cxf.path = /** in your application unless you are 100% sure no other extension will want to expose HTTP endpoints.

As of CEQ 2.13.3 the default value of **quarkus.cxf.path** is /. The default value will prevent other extensions from exposing HTTP endpoints.

This affects RESTEasy, Vert.x, SmallRye Health and others. If you use any of them, you should set **quarkus.cxf.path** to some specific path, such as **/services**, which is the default starting with **Camel Extensions for Quarkus 3.0.0** / **quarkus-cxf 2.0.0**.

NOTE

quarkus-cxf supports alternative ways of exposing SOAP endpoints. Refer to the SOAP Services chapter of the **quarkus-cxf** user guide for more details.

2.20.3.5. Logging of requests and responses

You can enable verbose logging of SOAP messages for clients and servers with **org.apache.cxf.ext.logging.LoggingFeature**:

```
import org.apache.camel.builder.RouteBuilder;
import org.apache.cxf.ext.logging.LoggingFeature;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.context.SessionScoped;
import javax.enterprise.inject.Produces;
import javax.inject.Named;

@ApplicationScoped
public class MyBeans {

    @Produces
    @ApplicationScoped
    @Named("prettyLoggingFeature")
    public LoggingFeature prettyLoggingFeature() {
        final LoggingFeature result = new LoggingFeature();
        result.setPrettyLogging(true);
        return result;
    }

    @Inject
    @Named("prettyLoggingFeature")
    LoggingFeature prettyLoggingFeature;

    @Produces
    @SessionScoped
    @Named
    CxfEndpoint cxfBeanClient() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(CalculatorService.class);
        result.setAddress("https://acme.org/calculator");
        result.setWsdlURL("wsdl/CalculatorService.wsdl");
        result.getFeatures().add(prettyLoggingFeature);
        return result;
    }

    @Produces
    @ApplicationScoped
    @Named
    CxfEndpoint helloBeanEndpoint() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(HelloService.class);
        result.setAddress("/hello-bean");
        result.setWsdlURL("wsdl/HelloService.wsdl");
        result.getFeatures().add(prettyLoggingFeature);
        return result;
    }
}
```

NOTE

io.quarkiverse.cxf:quarkus-cxf-rt-features-logging provides support for **org.apache.cxf.ext.logging.LoggingFeature** as a **camel-quarkus-cxf-soap** dependency.

You do not need to add it explicitly to your application.

2.20.3.6. WS Specifications

The extent of supported WS specifications is given by the Quarkus CXF project.

IMPORTANT

To learn about supported use cases and WS specifications, see the Quarkus CXF Reference.

If your application requires some other WS specification, you must add a Quarkus CXF dependency covering it.

In Camel Extensions for Quarkus we support all extensions listed with the support level **Stable**.

TIP

You can use integration tests as executable examples of applications that implement various WS specifications:

- Camel Extensions for Quarkus integration tests
- Quarkus CXF integration tests

2.20.3.7. Tooling

quarkus-cxf wraps the following two CXF tools:

- **wsdl2Java** - for generating service classes from WSDL
- **java2ws** - for generating WSDL from Java classes

IMPORTANT

For **wsdl2Java** to work properly, your application must directly depend on **io.quarkiverse.cxf:quarkus-cxf**.

TIP

While **wsdlvalidator** is not supported, you can use **wsdl2Java** with the following configuration in **application.properties** to validate your WSDLs:

application.properties

```
quarkus.cxf.codegen.wsdl2java.additional-params = -validate
```

2.21. DATA FORMAT

Use a Camel Data Format as a regular Camel Component.

For more details of the supported data formats in Camel Extensions for Quarkus, see Supported Data Formats.

2.21.1. What's inside

- Data Format component, URI syntax: **dataformat:name:operation**

Refer to the above link for usage and configuration details.

2.21.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-dataformat</artifactId>
</dependency>
```

2.22. DATASET

Provide data for load and soak testing of your Camel application.

2.22.1. What's inside

- Dataset component, URI syntax: **dataset:name**
- DataSet Test component, URI syntax: **dataset-test:name**

Please refer to the above links for usage and configuration details.

2.22.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-dataset</artifactId>
</dependency>
```

2.23. DIRECT

Call another endpoint from the same Camel Context synchronously.

2.23.1. What’s inside

- Direct component, URI syntax: **direct:name**

Please refer to the above link for usage and configuration details.

2.23.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-direct</artifactId>
</dependency>
```

2.24. FHIR

Exchange information in the healthcare domain using the FHIR (Fast Healthcare Interoperability Resources) standard. Marshall and unmarshall FHIR objects to/from JSON. Marshall and unmarshall FHIR objects to/from XML.

2.24.1. What’s inside

- FHIR component, URI syntax: **fhir:apiName/methodName**
- FHIR JSon data format
- FHIR XML data format

Please refer to the above links for usage and configuration details.

2.24.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-fhir</artifactId>
</dependency>
```

2.24.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.24.4. Additional Camel Quarkus configuration

By default, only FHIR versions **R4** & **DSTU3** are enabled in native mode, since they are the default values on the FHIR component and DataFormat.

Configuration property	Type	Default
🔒 **quarkus.camel.fhir.enable-dstu2** Enable FHIR DSTU2 Specs in native mode.	**boolean**	**false**
🔒 **quarkus.camel.fhir.enable-dstu2_hl7org** Enable FHIR DSTU2_HL7ORG Specs in native mode.	**boolean**	**false**
🔒 **quarkus.camel.fhir.enable-dstu2_1** Enable FHIR DSTU2_1 Specs in native mode.	**boolean**	**false**
🔒 **quarkus.camel.fhir.enable-dstu3** Enable FHIR DSTU3 Specs in native mode.	**boolean**	**false**

Configuration property	Type	Default
🔒 **quarkus.camel.fhir.enable-r4** Enable FHIR R4 Specs in native mode.	boolean	true
🔒 **quarkus.camel.fhir.enable-r5** Enable FHIR R5 Specs in native mode.	boolean	false

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.25. FILE

Read and write files.

2.25.1. What's inside

- File component, URI syntax: **file:directoryName**

Please refer to the above link for usage and configuration details.

2.25.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-file</artifactId>
</dependency>
```

2.25.3. Additional Camel Quarkus configuration

2.25.3.1. Having only a single consumer in a cluster consuming from a given endpoint

When the same route is deployed on multiple JVMs, it could be interesting to use this extension in conjunction with the Master one. In such a setup, a single consumer will be active at a time across the whole camel master namespace.

For instance, having the route below deployed on multiple JVMs:

```
from("master:ns:timer:test?period=100").log("Timer invoked on a single JVM at a time");
```

It’s possible to enable the file cluster service with a property like below:

```
quarkus.camel.cluster.file.enabled = true
quarkus.camel.cluster.file-root = target/cluster-folder-where-lock-file-will-be-held
```

As a result, a single consumer will be active across the **ns** camel master namespace. It means that, at a given time, only a single timer will generate exchanges across all JVMs. In other words, messages will be logged every 100ms on a single JVM at a time.

The file cluster service could further be tuned by tweaking **quarkus.camel.cluster.file.*** properties.

Configuration property	Type	Default
🔒 **quarkus.camel.cluster.file.enabled** Whether a File Lock Cluster Service should be automatically configured according to 'quarkus.camel.cluster.file.*' configurations.	**boolean**	**false**
🔒 **quarkus.camel.cluster.file-id** The cluster service ID (defaults to null).	**string**	
🔒 **quarkus.camel.cluster.file-root** The root path (defaults to null).	**string**	

Configuration property	Type	Default
🔒 **quarkus.camel.cluster.file-order** The service lookup order/priority (defaults to 2147482647).	**java.lang.Integer**	
🔒 **quarkus.camel.cluster.file.acquire-lock-delay** The time to wait before starting to try to acquire lock (defaults to 1000ms).	**string**	
🔒 **quarkus.camel.cluster.file.acquire-lock-interval** The time to wait between attempts to try to acquire lock (defaults to 10000ms).	**string**	
🔒 **quarkus.camel.cluster.file.attributes** The custom attributes associated to the service (defaults to empty map).	**Map<String,String>**	

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.26. FTP

Upload and download files to/from SFTP, FTP or SFTP servers

2.26.1. What's inside

- FTP component, URI syntax: **ftp:host:port/directoryName**
- FTPS component, URI syntax: **ftps:host:port/directoryName**
- SFTP component, URI syntax: **sftp:host:port/directoryName**

Please refer to the above links for usage and configuration details.

2.26.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-ftp</artifactId>
</dependency>
```

2.27. GSON

Marshal POJOs to JSON and back using Gson

2.27.1. What's inside

- JSON Gson data format

Please refer to the above link for usage and configuration details.

2.27.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-gson</artifactId>
</dependency>
```

2.27.3. Additional Camel Quarkus configuration

2.27.3.1. Marshaling/Unmarshaling objects in native mode

When marshaling/unmarshaling objects in native mode, all the serialized classes need to be registered for reflection. As such, when using **GsonDataFormat.setUnmarshalType(…)**,

GsonDataFormat.setUnmarshalTypeName(…) and even **GsonDataFormat.setUnmarshalGenericType(…)**, the unmarshal type as well as sub field types should be registered for reflection. See a working example in this integration test.

2.28. GOOGLE BIGQUERY

Access Google Cloud BigQuery service using SQL queries or Google Client Services API

2.28.1. What's inside

- Google BigQuery component, URI syntax: **google-bigquery:projectId:datasetId:tableId**
- Google BigQuery Standard SQL component, URI syntax: **google-bigquery-sql:projectId:queryString**

Please refer to the above links for usage and configuration details.

2.28.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-google-bigquery</artifactId>
</dependency>
```

2.28.3. Usage

If you want to read SQL scripts from the classpath with **google-bigquery-sql** in native mode, then you will need to ensure that they are added to the native image via the **quarkus.native.resources.includes** configuration property. Please check Quarkus documentation for more details.

2.29. GOOGLE PUBSUB

Send and receive messages to/from Google Cloud Platform PubSub Service.

2.29.1. What's inside

- Google Pubsub component, URI syntax: **google-pubsub:projectId:destinationName**

Please refer to the above link for usage and configuration details.

2.29.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-google-pubsub</artifactId>
```

```
</dependency>
```

2.29.3. Camel Quarkus limitations

By default, the Camel PubSub component uses JDK object serialization via **ObjectOutputStream** whenever the message body is anything other than **String** or **byte[]**.

Since such serialization is not yet supported by GraalVM, this extension provides a custom Jackson based serializer to serialize complex message payloads as JSON.

If your payload contains binary data, then you will need to handle that by creating a custom Jackson Serializer / Deserializer. Refer to the Quarkus Jackson guide for information on how to do this.

2.30. HL7

Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.

2.30.1. What's inside

- HL7 data format
- HL7 Terser language

Please refer to the above links for usage and configuration details.

2.30.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-hl7</artifactId>
</dependency>
```

2.30.3. Camel Quarkus limitations

For MLLP with TCP, Netty is the only supported means of running an Hl7 MLLP listener. Mina is not supported since it has no GraalVM native support at present.

Optional support for **HL7MLLPNettyEncoderFactory** & **HL7MLLPNettyDecoderFactory** codecs can be obtained by adding a dependency in your project **pom.xml** to **camel-quarkus-netty**.

2.31. HTTP

Send requests to external HTTP servers using Apache HTTP Client 4.x.

2.31.1. What's inside

- HTTP component, URI syntax: **http://httpUri**
- HTTPS (Secure) component, URI syntax: **https://httpUri**

Please refer to the above links for usage and configuration details.

2.31.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-http</artifactId>
</dependency>
```

2.31.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.31.4. Additional Camel Quarkus configuration

- Check the Character encodings section of the Native mode guide if you expect your application to send or receive requests using non-default encodings.

2.32. INFINISPAN

Read and write from/to Infinispan distributed key/value store and data grid.

2.32.1. What's inside

- Infinispan component, URI syntax: **infinispan:cacheName**

Please refer to the above link for usage and configuration details.

2.32.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-infinispan</artifactId>
</dependency>
```

2.32.3. Additional Camel Quarkus configuration

2.32.3.1. Infinispan Client Configuration

You can either configure the Infinispan client via the relevant Camel Infinispan component & endpoint options, or you may use the Quarkus Infinispan extension configuration properties .

2.32.3.2. Camel Infinispan **InfinispanRemoteAggregationRepository** in native mode

If you chose to use the **InfinispanRemoteAggregationRepository** in native mode, then you must enable native serialization support.

2.33. JACKSON

Marshal POJOs to JSON and back using Jackson

2.33.1. What's inside

- JSON Jackson data format

Please refer to the above link for usage and configuration details.

2.33.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson</artifactId>
</dependency>
```

2.33.3. Usage

2.33.3.1. Configuring the Jackson **ObjectMapper**

There are a few ways of configuring the **ObjectMapper** that the **JacksonDataFormat** uses. These are outlined below.

2.33.3.1.1. **ObjectMapper** created internally by **JacksonDataFormat**

By default, **JacksonDataFormat** will create its own **ObjectMapper** and use the various configuration options on the **DataFormat** to configure additional Jackson modules, pretty printing and other features.

2.33.3.1.2. Custom **ObjectMapper** for **JacksonDataFormat**

You can pass a custom **ObjectMapper** instance to **JacksonDataFormat** as follows.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

public class Routes extends RouteBuilder {
    public void configure() {
        ObjectMapper mapper = new ObjectMapper();
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        dataFormat.setObjectMapper(mapper);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

2.33.3.1.3. Using the Quarkus Jackson **ObjectMapper** with **JacksonDataFormat**

The Quarkus Jackson extension exposes an **ObjectMapper** CDI bean which can be discovered by the **JacksonDataFormat**.

```
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

public class Routes extends RouteBuilder {
    public void configure() {
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        // Make JacksonDataFormat discover the Quarkus Jackson `ObjectMapper` from the Camel
registry
        dataFormat.setAutoDiscoverObjectMapper(true);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

If you are using the JSON binding mode in the Camel REST DSL and want to use the Quarkus Jackson **ObjectMapper**, it can be achieved as follows.

```
import org.apache.camel.builder.RouteBuilder;

@ApplicationScoped
public class Routes extends RouteBuilder {
    public void configure() {
        restConfiguration().dataFormatProperty("autoDiscoverObjectMapper", "true");
        // REST definition follows...
    }
}
```

You can perform customizations on the Quarkus **ObjectMapper** with a **ObjectMapperCustomizer**.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import io.quarkus.jackson.ObjectMapperCustomizer;

@Singleton
public class RegisterCustomModuleCustomizer implements ObjectMapperCustomizer {
    public void customize(ObjectMapper mapper) {
        mapper.registerModule(new CustomModule());
    }
}
```

It's also possible to **@Inject** the Quarkus **ObjectMapper** and pass it to the **JacksonDataFormat**.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;
```

```
@ApplicationScoped
public class Routes extends RouteBuilder {
    @Inject
    ObjectMapper mapper;

    public void configure() {
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        dataFormat.setObjectMapper(mapper);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

2.34. AVRO JACKSON

Marshal POJOs to Avro and back using Jackson.

2.34.1. What's inside

- Avro Jackson data format

Please refer to the above link for usage and configuration details.

2.34.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson-avro</artifactId>
</dependency>
```

2.35. PROTOBUF JACKSON

Marshal POJOs to Protobuf and back using Jackson.

2.35.1. What's inside

- Protobuf Jackson data format

Please refer to the above link for usage and configuration details.

2.35.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson-protobuf</artifactId>
```

```
</dependency>
```

2.36. JACKSONXML

Unmarshal an XML payloads to POJOs and back using XMLMapper extension of Jackson.

2.36.1. What's inside

- Jackson XML data format

Please refer to the above link for usage and configuration details.

2.36.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jacksonxml</artifactId>
</dependency>
```

2.37. JAXB

Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.

2.37.1. What's inside

- JAXB data format

Please refer to the above link for usage and configuration details.

2.37.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jaxb</artifactId>
</dependency>
```

2.37.3. Usage

2.37.3.1. Native mode **ObjectFactory** instantiation of non-JAXB annotated classes

When performing JAXB marshal operations with a custom **ObjectFactory** to instantiate POJO classes that do not have JAXB annotations, you must register those POJO classes for reflection in order for them to be instantiated in native mode. E.g via the **@RegisterForReflection** annotation or configuration property **quarkus.camel.native.reflection.include-patterns**.

Refer to the Native mode user guide for more information.

2.38. JDBC

Access databases through SQL and JDBC.

2.38.1. What's inside

- JDBC component, URI syntax: **jdbc:dataSourceName**

Please refer to the above link for usage and configuration details.

2.38.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jdbc</artifactId>
</dependency>
```

2.38.3. Additional Camel Quarkus configuration

2.38.3.1. Configuring a DataSource

This extension leverages Quarkus Agroal for **DataSource** support. Setting up a **DataSource** can be achieved via configuration properties. It is recommended that you explicitly name the datasource so that it can be referenced in the JDBC endpoint URI. E.g like **to("jdbc:camel")**.

```
quarkus.datasource.camel.db-kind=postgresql
quarkus.datasource.camel.username=your-username
quarkus.datasource.camel.password=your-password
quarkus.datasource.camel.jdbc.url=jdbc:postgresql://localhost:5432/your-database
quarkus.datasource.camel.jdbc.max-size=16
```

If you choose to not name the datasource, you can resolve the default **DataSource** by defining your endpoint like **to("jdbc:default")**.

2.38.3.1.1. Zero configuration with Quarkus Dev Services

In dev and test mode you can take advantage of Configuration Free Databases. All you need to do is reference the default database in your routes. E.g **to("jdbc:default")**.

2.39. JIRA

Interact with JIRA issue tracker.

2.39.1. What's inside

- Jira component, URI syntax: **jira:type**

Please refer to the above link for usage and configuration details.

2.39.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jira</artifactId>
</dependency>
```

2.39.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.40. JMS

Sent and receive messages to/from a JMS Queue or Topic.

2.40.1. What's inside

- JMS component, URI syntax: **jms:destinationType:destinationName**

Please refer to the above link for usage and configuration details.

2.40.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jms</artifactId>
</dependency>
```

2.40.3. Usage

2.40.3.1. Message mapping with **org.w3c.dom.Node**

The Camel JMS component supports message mapping between **javax.jms.Message** and **org.apache.camel.Message**. When wanting to convert a Camel message body type of **org.w3c.dom.Node**, you must ensure that the **camel-quarkus-jaxp** extension is present on the classpath.

2.40.3.2. Native mode support for javax.jms.ObjectMessage

When sending JMS message payloads as **javax.jms.ObjectMessage**, you must annotate the relevant classes to be registered for serialization with **@RegisterForReflection(serialization = true)**. Note that this extension automatically sets **quarkus.camel.native.reflection.serialization-enabled = true** for you. Refer to the native mode user guide for more information.

2.40.3.3. Support for Connection pooling and X/Open XA distributed transactions

NOTE

Connection pooling is a Technical Preview feature in this release of Camel Extensions for Quarkus.

To use connection pooling in the **camel-quarkus-jms** components, you must add **io.quarkiverse.artemis:quarkus-artemis** and **io.quarkiverse.messaginghub:quarkus-pooled-jms** to your pom.xml and set the following configuration:

```
quarkus.pooled-jms.max-connections = 8
```

You can use the **quarkus-pooled-jms** extension to get pooling and XA support for JMS connections. Refer to the quarkus-pooled-jms extension documentation for more information. Currently, it only works with **quarkus-artemis-jms** extension. Just add these two dependencies to your **pom.xml**:

```
<dependency>
    <groupId>io.quarkiverse.messaginghub</groupId>
    <artifactId>quarkus-pooled-jms</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkiverse.artemis</groupId>
    <artifactId>quarkus-artemis-jms</artifactId>
</dependency>
```

Note that pooling is enabled by default.

To enable XA, you need to add the following configuration to your **application.properties**:

```
quarkus.pooled-jms.xa.enabled=true
```

NOTE

clientID and **durableSubscriptionName** are not supported in pooling connections. If **setClientID** is called on a **reused** connection from the pool, an **IllegalStateException** will be thrown. You will get some error messages such like **Cause: setClientID can only be called directly after the connection is created**

2.40.4. transferException option in native mode

To use the **transferException** option in native mode, you must enable support for object serialization. Refer to the native mode user guide for more information.

You will also need to enable serialization for the exception classes that you intend to serialize. For example.

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

2.41. JPA

Store and retrieve Java objects from databases using Java Persistence API (JPA).

2.41.1. What's inside

- JPA component, URI syntax: **jpa:entityType**

Please refer to the above link for usage and configuration details.

2.41.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jpa</artifactId>
</dependency>
```

2.41.3. Additional Camel Quarkus configuration

The extension leverages Quarkus Hibernate ORM to provide the JPA implementation via Hibernate.

Refer to the Quarkus Hibernate ORM documentation to see how to configure Hibernate and your datasource,

When a single persistence unit is used, the Camel Quarkus JPA extension will automatically configure the JPA component with a **EntityManagerFactory** and **TransactionManager**.

2.42. JSON PATH

Evaluate a JSONPath expression against a JSON message body

2.42.1. What's inside

- JSONPath language

Please refer to the above link for usage and configuration details.

2.42.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jsonpath</artifactId>
</dependency>
```

2.43. JTA

Enclose Camel routes in transactions using Java Transaction API (JTA) and Narayana transaction manager

2.43.1. What's inside

- JTA

Please refer to the above link for usage and configuration details.

2.43.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jta</artifactId>
</dependency>
```

2.43.3. Usage

This extension should be added when you need to use the **transacted()** EIP in the router. It leverages the transaction capabilities provided by the narayana-jta extension in Quarkus.

Refer to the Quarkus Transaction guide for the more details about transaction support. For a simple usage:

```
from("direct:transaction")
    .transacted()
    .to("sql:INSERT INTO A TABLE ...?dataSource=ds1")
    .to("sql:INSERT INTO A TABLE ...?dataSource=ds2")
    .log("all data are in the ds1 and ds2")
```

Support is provided for various transaction policies.

Policy	Description
PROPAGATION_MANDATORY	Support a current transaction; throw an exception if no current transaction exists.
PROPAGATION_NEVER	Do not support a current transaction; throw an exception if a current transaction exists.
PROPAGATION_NOT_SUPPORTED	Do not support a current transaction; rather always execute non-transactionally.

Policy	Description
PROPAGATION_REQUIRED	Support a current transaction; create a new one if none exists.
PROPAGATION_REQUIRES_NEW	Create a new transaction, suspending the current transaction if one exists.
PROPAGATION_SUPPORTS	Support a current transaction; execute non-transactionally if none exists.

2.44. JSLT

Query or transform JSON payloads using an JSLT.

2.44.1. What's inside

- JSLT component, URI syntax: **jslt:resourceUri**

Please refer to the above link for usage and configuration details.

2.44.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jslt</artifactId>
</dependency>
```

2.44.3. allowContextMapAll option in native mode

The **allowContextMapAll** option is not supported in native mode as it requires reflective access to security sensitive camel core classes such as **CamelContext** & **Exchange**. This is considered a security risk and thus access to the feature is not provided by default.

2.44.4. Additional Camel Quarkus configuration

2.44.4.1. Loading JSLT templates from classpath in native mode

This component typically loads the templates from classpath. To make it work also in native mode, you need to explicitly embed the templates files in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the route below would load the JSLT schema from a classpath resource named **transformation.json**:

```
from("direct:start").to("jslt:transformation.json");
```

To include this (an possibly other templates stored in **.json** files) in the native image, you would have to add something like the following to your **application.properties** file:

```
quarkus.native.resources.includes = *.json
```

2.44.4.2. Using JSLT functions in native mode

When using JSLT functions from camel-quarkus in native mode, the classes hosting the functions would need to be registered for reflection. When registering the target function is not possible, one may end up writing a stub as below.

```
@RegisterForReflection
public class MathFunctionStub {
    public static double pow(double a, double b) {
        return java.lang.Math.pow(a, b);
    }
}
```

The target function **Math.pow(…)** is now accessible through the **MathFunctionStub** class that could be registered in the component as below:

```
@Named
JsltComponent jsltWithFunction() throws ClassNotFoundException {
    JsltComponent component = new JsltComponent();
    component.setFunctions(singleton(wrapStaticMethod("power",
"org.apache.cq.example.MathFunctionStub", "pow")));
    return component;
}
```

2.45. KAFKA

Sent and receive messages to/from an Apache Kafka broker.

2.45.1. What's inside

- Kafka component, URI syntax: **kafka:topic**

Please refer to the above link for usage and configuration details.

2.45.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-kafka</artifactId>
</dependency>
```

2.45.3. Usage

2.45.3.1. Quarkus Kafka Dev Services

Camel Quarkus Kafka can take advantage of Quarkus Kafka Dev services to simplify development and testing with a local containerized Kafka broker.

Kafka Dev Services is enabled by default in dev & test mode. The Camel Kafka component is automatically configured so that the **brokers** component option is set to point at the local containerized Kafka broker. Meaning that there's no need to configure this option yourself.

This functionality can be disabled with the configuration property **quarkus.kafka.devservices.enabled=false**.

2.45.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.kafka.kubernetes-service-binding.merge-configuration If **true** then any Kafka configuration properties discovered by the Quarkus Kubernetes Service Binding extension (if configured) will be merged with those set via Camel Kafka component or endpoint options. If **false** then any Kafka configuration properties discovered by the Quarkus Kubernetes Service Binding extension are ignored, and all of the Kafka component configuration is driven by Camel.	**boolean**	**true**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.46. KAMELET

Materialize route templates

2.46.1. What's inside

- Kamelet component, URI syntax: **kamelet:templateId/routeId**

Please refer to the above link for usage and configuration details.

2.46.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-kamelet</artifactId>
</dependency>
```

▪

2.46.3. Usage

2.46.3.1. Pre-load Kamelets at build-time

This extension allows to pre-load a set of Kamelets at build time using the **quarkus.camel.kamelet.identifiers** property.

2.46.3.2. Using the Kamelet Catalog

A set of pre-made Kamelets can be found on the /camel-kamelets/latest[Kamelet Catalog]. To use the Kamelet from the catalog you need to copy their yaml definition (that you can find in the camel-kamelet repo) on your project in the classpath. Alternatively you can add the **camel-kamelets-catalog** artifact to your **pom.xml**:

```
<dependency>
    <groupId>org.apache.camel.kamelets</groupId>
    <artifactId>camel-kamelets-catalog</artifactId>
</dependency>
```

This artifact add all the kamelets available in the catalog to your Camel Quarkus application for build time processing. If you include it with the scope **provided** the artifact should not be part of the runtime classpath, but at build time, all the kamelets listed via **quarkus.camel.kamelet.identifiers** property should be preloaded.

2.46.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.kamelet.identifiers** List of kamelets identifiers to pre-load at build time. Each individual identifier is used to set the related **org.apache.camel.model.RouteTemplateDefinition** id.	**string**	

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.47. KUBERNETES

Perform operations against Kubernetes API

2.47.1. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

▪

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-kubernetes</artifactId>
</dependency>
```

2.47.2. Additional Camel Quarkus configuration

IMPORTANT

In this release of Camel Extensions for Quarkus, the **camel-quarkus-kubernetes** extension is only supported when used with the **camel-quarkus-master** extension as a cluster service. Additionally, in order for the **camel-quarkus-kubernetes** extension to be supported, you must explicitly add a dependency on the **quarkus-openshift-client** extension in your application.

2.47.2.1. Automatic registration of a Kubernetes Client instance

The extension automatically registers a Kubernetes Client bean named **kubernetesClient**. You can reference the bean in your routes like this:

```
from("direct:pods")
    .to("kubernetes-pods:///?kubernetesClient=#kubernetesClient&operation=listPods")
```

By default the client is configured from the local kubeconfig file. You can customize the client configuration via properties within **application.properties**:

```
quarkus.kubernetes-client.master-url=https://my.k8s.host
quarkus.kubernetes-client.namespace=my-namespace
```

The full set of configuration options are documented in the Quarkus Kubernetes Client guide.

2.47.2.2. Having only a single consumer in a cluster consuming from a given endpoint

When the same route is deployed on multiple pods, it could be interesting to use this extension in conjunction with the Master one. In such a setup, a single consumer will be active at a time across the whole camel master namespace.

For instance, having the route below deployed on multiple pods:

```
from("master:ns:timer:test?period=100").log("Timer invoked on a single pod at a time");
```

It’s possible to enable the kubernetes cluster service with a property like below:

```
quarkus.camel.cluster.kubernetes.enabled = true
```

As a result, a single consumer will be active across the **ns** camel master namespace. It means that, at a given time, only a single timer will generate exchanges across the whole cluster. In other words, messages will be logged every 100ms on a single pod at a time.

The kubernetes cluster service could further be tuned by tweaking **quarkus.camel.cluster.kubernetes.*** properties.

Configuration property	Type	Default
quarkus.camel.cluster.kubernetes.enabled Whether a Kubernetes Cluster Service should be automatically configured according to 'quarkus.camel.cluster.kubernetes.*' configurations.	boolean	false
quarkus.camel.cluster.kubernetes-id The cluster service ID (defaults to null).	string	
quarkus.camel.cluster.kubernetes.master-url The URL of the Kubernetes master (read from Kubernetes client properties by default).	string	
quarkus.camel.cluster.kubernetes.connection-timeout-millis The connection timeout in milliseconds to use when making requests to the Kubernetes API server.	java.lang.Integer	
quarkus.camel.cluster.kubernetes.namespace The name of the Kubernetes namespace containing the pods and the configmap (autodetected by default).	string	
quarkus.camel.cluster.kubernetes.pod-name The name of the current pod (autodetected from container host name by default).	string	

Configuration property	Type	Default
🔒 **quarkus.camel.cluster.kubernetes.jitter-factor** The jitter factor to apply in order to prevent all pods to call Kubernetes APIs in the same instant (defaults to 1.2).	java.lang.Double	
🔒 **quarkus.camel.cluster.kubernetes.lease-duration-millis** The default duration of the lease for the current leader (defaults to 15000).	java.lang.Long	
🔒 **quarkus.camel.cluster.kubernetes.renew-deadline-millis** The deadline after which the leader must stop its services because it may have lost the leadership (defaults to 10000).	java.lang.Long	

Configuration property	Type	Default
🔒 **quarkus.camel.cluster.kubernetes.retry-period-millis** The time between two subsequent attempts to check and acquire the leadership. It is randomized using the jitter factor (defaults to 2000).	java.lang.Long	
🔒 **quarkus.camel.cluster.kubernetes-order** Service lookup order/priority (defaults to 2147482647).	java.lang.Integer	
🔒 **quarkus.camel.cluster.kubernetes.resource-name** The name of the lease resource used to do optimistic locking (defaults to 'leaders'). The resource name is used as prefix when the underlying Kubernetes resource can manage a single lock.	string	
🔒 **quarkus.camel.cluster.kubernetes.lease-resource-type** The lease resource type used in Kubernetes, either 'config-map' or 'lease' (defaults to 'lease').	org.apache.ca	

Configuration property	Type	Default
	me.component.kubernetes.cluster.LeaseResourceType	
🔒 quarkus.camel.cluster.kubernetes.rebalancing Whether the camel master namespace leaders should be distributed evenly across all the camel contexts in the cluster.	boolean	true

Configuration property	Type	Default
🔒 **quarkus.camel.cluster.kubernetes-labels** The labels key/value used to identify the pods composing the cluster, defaults to empty map.	**Map<String,String>**	

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.48. LOG

Log messages to the underlying logging mechanism.

2.48.1. What's inside

- Log component, URI syntax: **log:loggerName**

Please refer to the above link for usage and configuration details.

2.48.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-log</artifactId>
</dependency>
```

2.49. MAIL

Send and receive emails using imap, pop3 and smtp protocols. Marshal Camel messages with attachments into MIME-Multipart messages and back.

2.49.1. What's inside

- IMAP component, URI syntax: **imap:host:port**
- IMAPS (Secure) component, URI syntax: **imaps:host:port**
- MIME Multipart data format
- POP3 component, URI syntax: **pop3:host:port**
- POP3S component, URI syntax: **pop3s:host:port**
- SMTP component, URI syntax: **smtp:host:port**
- SMTPS component, URI syntax: **smtps:host:port**

Please refer to the above links for usage and configuration details.

2.49.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mail</artifactId>
</dependency>
```

2.50. MASTER

Have only a single consumer in a cluster consuming from a given endpoint; with automatic failover if the JVM dies.

2.50.1. What's inside

- Master component, URI syntax: **master:namespace:delegateUri**

Please refer to the above link for usage and configuration details.

2.50.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-master</artifactId>
</dependency>
```

2.50.3. Additional Camel Quarkus configuration

This extension can be used in conjunction with extensions below:

- Camel Quarkus File
- Camel Quarkus Kubernetes

2.51. MICROPROFILE FAULT TOLERANCE

Circuit Breaker EIP using Microprofile Fault Tolerance

2.51.1. What's inside

- Microprofile Fault Tolerance

Please refer to the above link for usage and configuration details.

2.51.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-microprofile-fault-tolerance</artifactId>
</dependency>
```

2.52. MICROPROFILE HEALTH

Expose Camel health checks via MicroProfile Health

2.52.1. What's inside

- Microprofile Health

Please refer to the above link for usage and configuration details.

2.52.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-microprofile-health</artifactId>
</dependency>
```

2.52.3. Usage

By default, classes extending **AbstractHealthCheck** are registered as both liveness and readiness checks. You can override the **isReadiness** method to control this behaviour.

Any checks provided by your application are automatically discovered and bound to the Camel registry. They will be available via the Quarkus health endpoints **/q/health/live** and **/q/health/ready**.

You can also provide custom **HealthCheckRepository** implementations and these are also automatically discovered and bound to the Camel registry for you.

Refer to the Quarkus health guide for further information.

2.52.3.1. Provided health checks

Some checks are automatically registered for your application.

2.52.3.1.1. Camel Context Health

Inspects the Camel Context status and causes the health check status to be **DOWN** if the status is anything other than 'Started'.

2.52.3.1.2. Camel Route Health

Inspects the status of each route and causes the health check status to be **DOWN** if any route status is not 'Started'.

2.52.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.health.enabled** Set whether to enable Camel health checks	**boolean**	**true**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.53. MICROPROFILE METRICS

Expose metrics from Camel routes.

2.53.1. What’s inside

- MicroProfile Metrics component, URI syntax: **microprofile-metrics:metricType:metricName**

Please refer to the above link for usage and configuration details.

2.53.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-microprofile-metrics</artifactId>
</dependency>
```

2.53.3. Usage

The microprofile-metrics component automatically exposes a set of Camel application metrics. Some of these include:

2.53.3.1. Camel Context metrics

Metric Name	Type
camel.context.status The status of the Camel Context represented by the **ServiceStatus** enum ordinal	Gauge
camel.context.uptime The Camel Context uptime in milliseconds	Gauge
camel.context.exchanges.completed.total The total number of completed exchanges	Counter
camel.context.exchanges.failed.total The total number of failed exchanges	Counter
camel.context.exchanges.inflight.total The total number of inflight exchanges	Gauge
camel.context.exchanges.total The total number of all exchanges	Counter
camel.context.externalRedeliveries.total The total number of all external redeliveries	Counter
camel.context.failuresHandled.total The total number of all failures handled	Counter

2.53.3.2. Camel Route metrics

Metric Name	Type
camel.route.count The number of routes	Gauge
camel.route.running.count The number of running routes	Gauge
camel.route.exchanges.completed.total The total number of completed exchanges for the route	Counter
camel.route.exchanges.failed.total The total number of failed exchanges for the route	Counter
camel.route.exchanges.inflight.total The total number of inflight exchanges for the route	Gauge
camel.route.exchanges.total The total number of all exchanges for the route	Counter
camel.route.externalRedeliveries.total The total number of all external redeliveries for the route	Counter
camel.route.failuresHandled.total The total number of all failures handled for the route	Counter

All metrics are tagged with the name of the Camel Context and the id of the route where applicable.

You can also produce your own customized metrics in your Camel routes. For more information, refer to the microprofile-metrics component documentation.

Metrics are exposed to Quarkus as application metrics and they can be browsed at http://localhost:8080/q/metrics/application.

2.53.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.metrics.enable-route-policy** Set whether to enable the MicroProfileMetricsRoutePolicyFactory for capturing metrics on route processing times.	**boolean**	**true**
🔒 **quarkus.camel.metrics.enable-message-history** Set whether to enable the MicroProfileMetricsMessageHistoryFactory for capturing metrics on individual route node processing times. Depending on the number of configured route nodes, there is the potential to create a large volume of metrics. Therefore, this option is disabled by default.	**boolean**	**false**
🔒 **quarkus.camel.metrics.enable-exchange-event-notifier** Set whether to enable the MicroProfileMetricsExchangeEventNotifier for capturing metrics on exchange processing times.	**boolean**	**true**
🔒 **quarkus.camel.metrics.enable-route-event-notifier** Set whether to enable the MicroProfileMetricsRouteEventNotifier for capturing metrics on the total number of routes and total number of routes running.	**boolean**	**true**
🔒 **quarkus.camel.metrics.enable-camel-context-event-notifier** Set whether to enable the MicroProfileMetricsCamelContextEventNotifier for capturing metrics about the CamelContext, such as status and uptime.	**boolean**	**true**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.54. MLLP

Communicate with external systems using the MLLP protocol.

2.54.1. What's inside

- MLLP component, URI syntax: **mllp:hostname:port**

Please refer to the above link for usage and configuration details.

2.54.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mllp</artifactId>
</dependency>
```

2.54.3. Additional Camel Quarkus configuration

- Check the Character encodings section of the Native mode guide if you wish to use the **defaultCharset** component option.

2.55. MOCK

Test routes and mediation rules using mocks.

2.55.1. What's inside

- Mock component, URI syntax: **mock:name**

Please refer to the above link for usage and configuration details.

2.55.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mock</artifactId>
</dependency>
```

2.55.3. Usage

To use camel-mock capabilities in tests it is required to get access to MockEndpoint instances.

CDI injection could be used for accessing instances (see Quarkus documentation). You can inject camelContext into test using **@Inject** annotation. Camel context can be then used for obtaining mock endpoints. See the following example:

```
import javax.inject.Inject;
import org.apache.camel.CamelContext;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.component.mock.MockEndpoint;
import org.junit.jupiter.api.Test;

import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class MockJvmTest {

    @Inject
    CamelContext camelContext;

    @Inject
    ProducerTemplate producerTemplate;

    @Test
    public void test() throws InterruptedException {

        producerTemplate.sendBody("direct:start", "Hello World");

        MockEndpoint mockEndpoint = camelContext.getEndpoint("mock:result", MockEndpoint.class);
        mockEndpoint.expectedBodiesReceived("Hello World");

        mockEndpoint.assertIsSatisfied();
    }
}
```

Route used for the example test:

```
import javax.enterprise.context.ApplicationScoped;

import org.apache.camel.builder.RouteBuilder;

@ApplicationScoped
public class MockRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start").to("mock:result");
    }
}
```

2.55.4. Camel Quarkus limitations

Injection of CDI beans (described in Usage) does not work in native mode.

In the native mode the test and the application under test are running in two different processes and it is not possible to share a mock bean between them (see Quarkus documentation).

2.56. MONGODB

Perform operations on MongoDB documents and collections.

2.56.1. What's inside

- MongoDB component, URI syntax: **mongodb:connectionBean**

Please refer to the above link for usage and configuration details.

2.56.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mongodb</artifactId>
</dependency>
```

2.56.3. Additional Camel Quarkus configuration

The extension leverages the Quarkus MongoDB Client extension. The Mongo client can be configured via the Quarkus MongoDB Client configuration options.

The Camel Quarkus MongoDB extension automatically registers a MongoDB client bean named **camelMongoClient**. This can be referenced in the mongodb endpoint URI **connectionBean** path parameter. For example:

```
from("direct:start")
.to("mongodb:camelMongoClient?database=myDb&collection=myCollection&operation=findAll")
```

If your application needs to work with multiple MongoDB servers, you can create a "named" client and reference in your route by injecting a client and the related configuration as explained in the Quarkus MongoDB extension client injection. For example:

```
//application.properties
quarkus.mongodb.mongoClient1.connection-string = mongodb://root:example@localhost:27017/
```

```
//Routes.java

    @ApplicationScoped
    public class Routes extends RouteBuilder {
        @Inject
        @MongoClientName("mongoClient1")
        MongoClient mongoClient1;

        @Override
        public void configure() throws Exception {
            from("direct:defaultServer")
                .to("mongodb:camelMongoClient?
database=myDb&collection=myCollection&operation=findAll")

            from("direct:otherServer")
                .to("mongodb:mongoClient1?
database=myOtherDb&collection=myOtherCollection&operation=findAll");
        }
    }
```

Note that when using named clients, the "default" **camelMongoClient** bean will still be produced. Refer to the Quarkus documentation on Multiple MongoDB Clients for more information.

2.57. NETTY

Socket level networking using TCP or UDP with Netty 4.x.

2.57.1. What's inside

- Netty component, URI syntax: **netty:protocol://host:port**

Please refer to the above link for usage and configuration details.

2.57.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-netty</artifactId>
</dependency>
```

2.58. OPENAPI JAVA

Expose OpenAPI resources defined in Camel REST DSL

2.58.1. What's inside

- Openapi Java

Please refer to the above link for usage and configuration details.

2.58.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-openapi-java</artifactId>
</dependency>
```

2.58.3. Usage

You can use this extension to expose REST DSL services to Quarkus OpenAPI. With **quarkus-smallrye-openapi**, you can access them by **/q/openapi?format=json**.

Refer to the Quarkus OpenAPI guide for further information.

This is an experimental feature. You can enable it by

```
quarkus.camel.openapi.expose.enabled=true
```

> **WARNING**
>
> It's the user's responsibility to use **@RegisterForReflection** to register all model classes for reflection.
>
> It doesn't support the rest services used in **org.apache.camel.builder.LambdaRouteBuilder** right now. Also, it can not use CDI injection in the RouteBuilder **configure()** since we get the rest definitions at build time while CDI is unavailable.

2.58.4. Camel Quarkus limitations

The **apiContextIdListing** configuration option is not supported. Since multiple **CamelContext`s are not supported and Quarkus applications run standalone, there is no scenario where attempting to resolve OpenApi specifications for a specific `CamelContext** would be useful. It also introduces some additional overhead of requiring JMX (which is not supported in native mode) & additional Camel Quarkus extensions for processing XML.

2.59. OPENTELEMETRY

Distributed tracing using OpenTelemetry

2.59.1. What's inside

- OpenTelemetry

Please refer to the above link for usage and configuration details.

2.59.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-opentelemetry</artifactId>
</dependency>
```

2.59.3. Usage

The extension automatically creates a Camel **OpenTelemetryTracer** and binds it to the Camel registry.

In order to send the captured traces to a tracing system, you need to configure some properties within **application.properties** like those below.

```
# Identifier for the origin of spans created by the application
quarkus.application.name=my-camel-application

# For OTLP
quarkus.opentelemetry.tracer.exporter.otlp.endpoint=http://localhost:4317

# For Jaeger
quarkus.opentelemetry.tracer.exporter.jaeger.endpoint=http://localhost:14250
```

Note that you must add a dependency to the OpenTelemetry exporter that you want to work with.

At present, Quarkus has support for Jaeger and the OpenTelemetry Protocol Specification (OTLP).

For Jaeger:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-opentelemetry-exporter-jaeger</artifactId>
</dependency>
```

For OTLP:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-opentelemetry-exporter-otlp</artifactId>
</dependency>
```

Refer to the Quarkus OpenTelemetry guide for a full list of configuration options.

Route endpoints can be excluded from tracing by configuring a property named **quarkus.camel.opentelemetry.exclude-patterns** in **application.properties**. For example:

```
# Exclude all direct & netty-http endpoints from tracing
quarkus.camel.opentelemetry.exclude-patterns=direct:*,netty-http:*
```

2.59.3.1. Tracing CDI bean method execution

When instrumenting the execution of CDI bean methods from Camel routes, you should annotate such methods with **io.opentelemetry.extension.annotations.WithSpan**. Methods annotated with **@WithSpan** will create a new Span and establish any required relationships with the current Trace context.

For example, to instrument a CDI bean from a Camel route, first ensure the appropriate methods are annotated with **@WithTrace**.

```
@ApplicationScoped
@Named("myBean")
```

```
public class MyBean {
    @WithSpan
    public String greet() {
        return "Hello World!";
    }
}
```

Next, use the bean in your Camel route.

IMPORTANT

To ensure that the sequence of recorded spans is correct, you must use the full **to("bean:")** endpoint URI and not the shortened **.bean()** EIP DSL method.

```
public class MyRoutes extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("direct:executeBean")
                .to("bean:myBean?method=greet");
    }
}
```

There is more information about CDI instrumentation in the Quarkus OpenTelemetry guide.

2.59.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.opentelemetry.encoding** Sets whether header names need to be encoded. Can be useful in situations where OpenTelemetry propagators potentially set header name values in formats that are not compatible with the target system. E.g for JMS where the specification mandates header names are valid Java identifiers.	**boolean**	**false**
🔒 **quarkus.camel.opentelemetry.exclude-patterns** Sets whether to disable tracing for endpoint URIs that match the given patterns. The pattern can take the following forms: 1. An exact match on the endpoint URI. E.g platform-http:/some/path 2. A wildcard match. E.g platform-http:* 3. A regular expression matching the endpoint URI. E.g platform-http:/prefix/.*	**string**	

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.60. PAHO

Communicate with MQTT message brokers using Eclipse Paho MQTT Client.

2.60.1. What's inside

- Paho component, URI syntax: **paho:topic**

Please refer to the above link for usage and configuration details.

2.60.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-paho</artifactId>
</dependency>
```

2.61. PAHO MQTT5

Communicate with MQTT message brokers using Eclipse Paho MQTT v5 Client.

2.61.1. What's inside

- Paho MQTT 5 component, URI syntax: **paho-mqtt5:topic**

Please refer to the above link for usage and configuration details.

2.61.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-paho-mqtt5</artifactId>
</dependency>
```

2.62. PLATFORM HTTP

This extension allows for creating HTTP endpoints for consuming HTTP requests.

It is built on top of the Eclipse Vert.x HTTP server provided by the **quarkus-vertx-http** extension.

2.62.1. What's inside

- Platform HTTP component, URI syntax: **platform-http:path**

Please refer to the above link for usage and configuration details.

2.62.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-platform-http</artifactId>
</dependency>
```

2.62.3. Usage

2.62.3.1. Basic Usage

Serve all HTTP methods on the **/hello** endpoint:

```
from("platform-http:/hello").setBody(simple("Hello ${header.name}"));
```

Serve only GET requests on the **/hello** endpoint:

```
from("platform-http:/hello?httpMethodRestrict=GET").setBody(simple("Hello ${header.name}"));
```

2.62.3.2. Using **platform-http** via Camel REST DSL

To be able to use Camel REST DSL with the **platform-http** component, add **camel-quarkus-rest** to your **pom.xml**:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-rest</artifactId>
</dependency>
```

Then you can use the Camel REST DSL:

```
rest()
    .get("/my-get-endpoint")
        .to("direct:handleGetRequest");

    .post("/my-post-endpoint")
        .to("direct:handlePostRequest");
```

2.62.3.3. Handling **multipart/form-data** file uploads

You can restrict the uploads to certain file extensions by white listing them:

```
from("platform-http:/upload/multipart?fileNameExtWhitelist=html,txt&httpMethodRestrict=POST")
    .to("log:multipart")
    .process(e -> {
            final AttachmentMessage am = e.getMessage(AttachmentMessage.class);
            if (am.hasAttachments()) {
                am.getAttachments().forEach((fileName, dataHandler) -> {
                    try (InputStream in = dataHandler.getInputStream()) {
                        // do something with the input stream
                    } catch (IOException ioe) {
                        throw new RuntimeException(ioe);
                    }
                });
            }
        });
```

2.62.3.4. Securing platform-http endpoints

Quarkus provides a variety of security and authentication mechanisms which can be used to secure **platform-http** endpoints. Refer to the Quarkus Security documentation for further details.

Within a route, it is possible to obtain the authenticated user and its associated **SecurityIdentity** and **Principal**:

```
from("platform-http:/secure")
    .process(e -> {
        Message message = e.getMessage();
        QuarkusHttpUser user =
message.getHeader(VertxPlatformHttpConstants.AUTHENTICATED_USER,
QuarkusHttpUser.class);
        SecurityIdentity securityIdentity = user.getSecurityIdentity();
        Principal principal = securityIdentity.getPrincipal();
        // Do something useful with SecurityIdentity / Principal. E.g check user roles etc.
    });
```

Also check the **quarkus.http.body.*** configuration options in Quarkus documentation, esp. **quarkus.http.body.handle-file-uploads**, **quarkus.http.body.uploads-directory** and **quarkus.http.body.delete-uploaded-files-on-end**.

2.62.3.5. Implementing a reverse proxy

Platform HTTP component can act as a reverse proxy, in that case **Exchange.HTTP_URI**, **Exchange.HTTP_HOST** headers are populated from the absolute URL received on the request line of the HTTP request.

Here's an example of a HTTP proxy that simply redirects the Exchange to the origin server.

```
from("platform-http:proxy")
    .toD("http://"
        + "${headers." + Exchange.HTTP_HOST + "}");
```

2.62.4. Additional Camel Quarkus configuration

2.62.4.1. Platform HTTP server configuration

Configuration of the platform HTTP server is managed by Quarkus. Refer to the Quarkus HTTP configuration guide for the full list of configuration options.

To configure SSL for the Platform HTTP server, follow the secure connections with SSL guide. Note that configuring the server for SSL with **SSLContextParameters** is not currently supported.

2.62.4.2. Character encodings

Check the Character encodings section of the Native mode guide if you expect your application to send or receive requests using non-default encodings.

2.63. QUARTZ

Schedule sending of messages using the Quartz 2.x scheduler.

2.63.1. What's inside

- Quartz component, URI syntax: **quartz:groupName/triggerName**

Please refer to the above link for usage and configuration details.

2.63.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```xml
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-quartz</artifactId>
</dependency>
```

2.63.3. Usage

2.63.3.1. Clustering

Support for Quartz clustering is provided by the Quarkus Quartz extension. The following steps outline how to configure Quarkus Quartz for use with Camel.

1. Enable Quartz clustered mode and configure a **DataSource** as a persistence Quartz job store. An example configuration is as follows.

```
# Quartz configuration
quarkus.quartz.clustered=true
quarkus.quartz.store-type=jdbc-cmt
quarkus.quartz.start-mode=forced

# Datasource configuration
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=quarkus_test
quarkus.datasource.password=quarkus_test
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost/quarkus_test

# Optional automatic creation of Quartz tables
quarkus.flyway.connect-retries=10
quarkus.flyway.table=flyway_quarkus_history
quarkus.flyway.migrate-at-start=true
quarkus.flyway.baseline-on-migrate=true
quarkus.flyway.baseline-version=1.0
quarkus.flyway.baseline-description=Quartz
```

2. Add the correct JDBC driver extension to your application that corresponds to the value of **quarkus.datasource.db-kind**. In the above example **postgresql** is used, therefore the following JDBC dependency would be required. Adjust as necessary for your needs. Agroal is also required for **DataSource** support.

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-jdbc-postgresql</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-agroal</artifactId>
</dependency>
```

3. Quarkus Flyway can automatically create the necessary Quartz database tables for you. Add **quarkus-flyway** to your application (optional).

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-flyway</artifactId>
</dependency>
```

Also add a Quartz database creation script for your chosen database kind. The Quartz project provides ready made scripts that can be copied from here. Add the SQL script to **src/main/resources/db/migration/V1.0.0__QuarkusQuartz.sql**. Quarkus Flyway will detect it on startup and will proceed to create the Quartz database tables.

4. Configure the Camel Quartz component to use the Quarkus Quartz scheduler.

```
@Produces
@Singleton
@Named("quartz")
public QuartzComponent quartzComponent(Scheduler scheduler) {
    QuartzComponent component = new QuartzComponent();
    component.setScheduler(scheduler);
    return component;
}
```

Further customization of the Quartz scheduler can be done via various configuration properties. Refer to to the Quarkus Quartz Configuration guide for more information.

2.64. REF

Route messages to an endpoint looked up dynamically by name in the Camel Registry.

2.64.1. What's inside

- Ref component, URI syntax: **ref:name**

Please refer to the above link for usage and configuration details.

2.64.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```xml
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-ref</artifactId>
</dependency>
```

2.64.3. Usage

CDI producer methods can be harnessed to bind endpoints to the Camel registry, so that they can be resolved using the **ref** URI scheme in Camel routes.

For example, to produce endpoint beans:

```java
@ApplicationScoped
public class MyEndpointProducers {
    @Inject
    CamelContext context;

    @Singleton
    @Produces
    @Named("endpoint1")
    public Endpoint directStart() {
        return context.getEndpoint("direct:start");
    }

    @Singleton
    @Produces
    @Named("endpoint2")
    public Endpoint logEnd() {
        return context.getEndpoint("log:end");
    }
}
```

Use **ref:** to refer to the names of the CDI beans that were bound to the Camel registry:

```java
public class MyRefRoutes extends RouteBuilder {
    @Override
    public void configure() {
        // direct:start -> log:end
        from("ref:endpoint1")
            .to("ref:endpoint2");
    }
}
```

2.65. REST

Expose REST services and their OpenAPI Specification or call external REST services.

2.65.1. What's inside

- REST component, URI syntax: **rest:method:path:uriTemplate**
- REST API component, URI syntax: **rest-api:path**

Please refer to the above links for usage and configuration details.

2.65.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-rest</artifactId>
</dependency>
```

2.65.3. Additional Camel Quarkus configuration

This extension depends on the Platform HTTP extension and configures it as the component that provides the REST transport.

2.65.3.1. Path parameters containing special characters with platform-http

When using the **platform-http** REST transport, some characters are not allowed within path parameter names. This includes the '-' and '$' characters.

In order to make the below example REST **/dashed/param** route work correctly, a system property is required **io.vertx.web.route.param.extended-pattern=true**.

```
import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

    @Override
    public void configure() {
        rest("/api")
            // Dash '-' is not allowed by default
            .get("/dashed/param/{my-param}")
            .to("direct:greet")

            // The non-dashed path parameter works by default
            .get("/undashed/param/{myParam}")
            .to("direct:greet");

            from("direct:greet")
                .setBody(constant("Hello World"));
    }
}
```

There is some more background to this in the Vert.x Web documentation.

2.65.3.2. Configuring alternate REST transport providers

To use another REST transport provider, such as **netty-http** or **servlet**, you need to add the respective extension as a dependency to your project and set the provider in your **RouteBuilder**. E.g. for **servlet**, you'd have to add the **org.apache.camel.quarkus:camel-quarkus-servlet** dependency and the set the provider as follows:

```
import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

    @Override
    public void configure() {
        restConfiguration()
                .component("servlet");
        ...
    }
}
```

2.66. REST OPENAPI

Configure REST producers based on an OpenAPI specification document delegating to a component implementing the RestProducerFactory interface.

2.66.1. What's inside

- REST OpenApi component, URI syntax: **rest-openapi:specificationUri#operationId**

Please refer to the above link for usage and configuration details.

2.66.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-rest-openapi</artifactId>
</dependency>
```

2.66.3. Usage

2.66.3.1. Required Dependencies

A **RestProducerFactory** implementation must be available when using the rest-openapi extension. The currently known extensions are:

- camel-quarkus-http
- camel-quarkus-netty-http

Maven users will need to add one of these dependencies to their **pom.xml**, for example:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-http</artifactId>
</dependency>
```

Depending on which mechanism is used to load the OpenApi specification, additional dependencies may be required. When using the **file** resource locator, the **org.apache.camel.quarkus:camel-quarkus-file** extension must be added as a project dependency. When using **ref** or **bean** to load the specification, not only must the **org.apache.camel.quarkus:camel-quarkus-bean** dependency be added, but the bean itself must be annotated with **@RegisterForReflection**.

When using the **classpath** resource locator with native code, the path to the OpenAPI specification must be specified in the **quarkus.native.resources.includes** property of the **application.properties** file. For example:

```
quarkus.native.resources.includes=openapi.json
```

2.67. SALESFORCE

Communicate with Salesforce using Java DTOs.

2.67.1. What's inside

- Salesforce component, URI syntax: **salesforce:operationName:topicName**

Please refer to the above link for usage and configuration details.

2.67.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-salesforce</artifactId>
</dependency>
```

2.67.3. Usage

2.67.3.1. Generating Salesforce DTOs with the **salesforce-maven-plugin**

Test content.

To generate Salesforce DTOs for your project, use the **salesforce-maven-plugin**. The example code snippet below creates a single DTO for the **Account** object.

```
<plugin>
    <groupId>org.apache.camel.maven</groupId>
    <artifactId>camel-salesforce-maven-plugin</artifactId>
    <version>3.18.6</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <clientId>${env.SALESFORCE_CLIENTID}</clientId>
                <clientSecret>${env.SALESFORCE_CLIENTSECRET}</clientSecret>
                <userName>${env.SALESFORCE_USERNAME}</userName>
                <password>${env.SALESFORCE_PASSWORD}</password>
                <loginUrl>https://login.salesforce.com</loginUrl>

<packageName>org.apache.camel.quarkus.component.salesforce.generated</packageName>
                <outputDirectory>src/main/java</outputDirectory>
                <includes>
                    <include>Account</include>
                </includes>
            </configuration>
        </execution>
    </executions>
</plugin>
```

2.67.4. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.68. XQUERY

Query and/or transform XML payloads using XQuery and Saxon.

2.68.1. What's inside

- XQuery component, URI syntax: **xquery:resourceUri**
- XQuery language

Please refer to the above links for usage and configuration details.

2.68.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-saxon</artifactId>
</dependency>
```

2.68.3. Additional Camel Quarkus configuration

This component is able to load XQuery definitions from classpath. To make it work also in native mode, you need to explicitly embed the queries in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the two routes below load an XQuery script from two classpath resources named **myxquery.txt** and **another-xquery.txt** respectively:

```
from("direct:start").transform().xquery("resource:classpath:myxquery.txt", String.class);
from("direct:start").to("xquery:another-xquery.txt");
```

To include these (an possibly other queries stored in **.txt** files) in the native image, you would have to add something like the following to your **application.properties** file:

```
quarkus.native.resources.includes = *.txt
```

2.69. SCHEDULER

Generate messages in specified intervals using java.util.concurrent.ScheduledExecutorService.

2.69.1. What's inside

- Scheduler component, URI syntax: **scheduler:name**

Please refer to the above link for usage and configuration details.

2.69.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-scheduler</artifactId>
</dependency>
```

2.70. SEDA

Asynchronously call another endpoint from any Camel Context in the same JVM.

2.70.1. What's inside

- SEDA component, URI syntax: **seda:name**

Please refer to the above link for usage and configuration details.

2.70.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-seda</artifactId>
</dependency>
```

2.71. SLACK

Send and receive messages to/from Slack.

2.71.1. What's inside

- Slack component, URI syntax: **slack:channel**

Please refer to the above link for usage and configuration details.

2.71.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-slack</artifactId>
</dependency>
```

2.71.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.72. SOAP DATAFORMAT

Marshal Java objects to SOAP messages and back.

2.72.1. What's inside

- SOAP data format

Please refer to the above link for usage and configuration details.

2.72.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-soap</artifactId>
</dependency>
```

2.73. SNMP

Receive traps and poll SNMP (Simple Network Management Protocol) capable devices.

2.73.1. What's inside

- SNMP component

URI syntax: **snmp:host:port**

Refer to the SNMP component for usage and configuration details.

2.73.2. Maven coordinates

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-snmp</artifactId>
</dependency>
```

2.73.3. Camel Quarkus limitations

This extension uses **org.snmp4j:snmp4j** as an SNMP protocol implementation. This is different from Camel 3.18.x using **org.apache.servicemix.bundles.snmp4j:org.apache.servicemix.bundles**.

The motivation for this change is twofold:

1. **org.snmp4j:snmp4j** brings more stability and fixes many vulnerabilities.
2. Camel switches to **org.snmp4j.snmp4j** in version 4 anyway, so Camel Quarkus users can enjoy the same benefits a bit earlier.

This change has no impact on configuration of the SNMP component.

2.73.3.1. SNMP v3 Limitation

SNMP version 3 is supported only for the operation **poll**. (This limitation is caused by an issue in Camel 3.18.6. For more information, see CAMEL-19298).

2.74. SQL

Perform SQL queries.

2.74.1. What's inside

- SQL component, URI syntax: **sql:query**
- SQL Stored Procedure component, URI syntax: **sql-stored:template**

Please refer to the above links for usage and configuration details.

2.74.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-sql</artifactId>
</dependency>
```

2.74.3. Additional Camel Quarkus configuration

2.74.3.1. Configuring a DataSource

This extension leverages Quarkus Agroal for **DataSource** support. Setting up a **DataSource** can be achieved via configuration properties.

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=your-username
quarkus.datasource.password=your-password
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/your-database
quarkus.datasource.jdbc.max-size=16
```

The Camel SQL component will automatically resolve the **DataSource** bean from the registry. When configuring multiple datasources, you can specify which one is to be used on an SQL endpoint via the URI options **datasource** or **dataSourceRef**. Refer to the SQL component documentation for more details.

2.74.3.1.1. Zero configuration with Quarkus Dev Services

In dev and test mode you can take advantage of Configuration Free Databases. The Camel SQL component will be automatically configured to use a **DataSource** that points to a local containerized instance of the database matching the JDBC driver type that you have selected.

2.74.3.2. SQL scripts

When configuring **sql** or **sql-stored** endpoints to reference script files from the classpath, set the following configuration property to ensure that they are available in native mode.

```
quarkus.native.resources.includes = queries.sql, sql/*.sql
```

2.74.3.3. SQL aggregation repository in native mode

In order to use SQL aggregation repositories like **JdbcAggregationRepository** in native mode, you must enable native serialization support.

In addition, if your exchange bodies are custom types, they must be registered for serialization by annotating their class declaration with **@RegisterForReflection(serialization = true)**.

2.75. TELEGRAM

Send and receive messages acting as a Telegram Bot Telegram Bot API.

2.75.1. What's inside

- Telegram component, URI syntax: **telegram:type**

Please refer to the above link for usage and configuration details.

2.75.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-telegram</artifactId>
</dependency>
```

2.75.3. Usage

2.75.4. Webhook Mode

The Telegram extension supports usage in the webhook mode.

In order to enable webhook mode, users need first to add a REST implementation to their application. Maven users, for example, can add **camel-quarkus-rest** extension to their **pom.xml** file:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-rest</artifactId>
</dependency>
```

2.75.5. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.76. TIMER

Generate messages in specified intervals using java.util.Timer.

2.76.1. What's inside

- Timer component, URI syntax: **timer:timerName**

Please refer to the above link for usage and configuration details.

2.76.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-timer</artifactId>
</dependency>
```

2.77. VERT.X HTTP CLIENT

Camel HTTP client support with Vert.x

2.77.1. What’s inside

- Vert.x HTTP Client component, URI syntax: **vertx-http:httpUri**

Please refer to the above link for usage and configuration details.

2.77.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-vertx-http</artifactId>
</dependency>
```

2.77.3. transferException option in native mode

To use the **transferException** option in native mode, you must enable support for object serialization. Refer to the native mode user guide for more information.

You will also need to enable serialization for the exception classes that you intend to serialize. For example.

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

2.77.4. Additional Camel Quarkus configuration

2.77.5. allowJavaSerializedObject option in native mode

When using the **allowJavaSerializedObject** option in native mode, the support of serialization might need to be enabled. Please, refer to the native mode user guide for more information.

2.77.5.1. Character encodings

Check the Character encodings section of the Native mode guide if the application is expected to send and receive requests using non-default encodings.

2.78. VALIDATOR

Validate the payload using XML Schema and JAXP Validation.

2.78.1. What's inside

- Validator component, URI syntax: **validator:resourceUri**

Please refer to the above link for usage and configuration details.

2.78.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-validator</artifactId>
</dependency>
```

2.79. VELOCITY

Transform messages using a Velocity template.

2.79.1. What's inside

- Velocity component, URI syntax: **velocity:resourceUri**

Please refer to the above link for usage and configuration details.

2.79.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-velocity</artifactId>
</dependency>
```

2.79.3. Usage

2.79.3.1. Custom body as domain object in the native mode

When using a custom object as message body and referencing its properties in the template in the native mode, all the classes need to be registered for reflection (see the documentation).

Example:

```
@RegisterForReflection
public interface CustomBody {
}
```

2.79.4. allowContextMapAll option in native mode

The **allowContextMapAll** option is not supported in native mode as it requires reflective access to security sensitive camel core classes such as **CamelContext** & **Exchange**. This is considered a security risk and thus access to the feature is not provided by default.

2.79.5. Additional Camel Quarkus configuration

This component typically loads Velocity templates from classpath. To make it work also in native mode, you need to explicitly embed the templates in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the route below would load the Velocity template from a classpath resource named **template/simple.vm**:

```
from("direct:start").to("velocity://template/simple.vm");
```

To include this (an possibly other templates stored in **.vm** files in the **template** directory) in the native image, you would have to add something like the following to your **application.properties** file:

```
quarkus.native.resources.includes = template/*.vm
```

2.80. XML IO DSL

An XML stack for parsing XML route definitions

2.80.1. What’s inside

- XML DSL

Please refer to the above link for usage and configuration details.

2.80.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-xml-io-dsl</artifactId>
</dependency>
```

2.80.3. Additional Camel Quarkus configuration

2.80.3.1. XML file encodings

By default, some XML file encodings may not work out of the box in native mode. Please, check the Character encodings section to learn how to fix.

2.81. XPATH

Evaluates an XPath expression against an XML payload

2.81.1. What’s inside

- XPath language

Please refer to the above link for usage and configuration details.

2.81.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-xpath</artifactId>
</dependency>
```

2.81.3. Additional Camel Quarkus configuration

This component is able to load xpath expressions from classpath resources. To make it work also in native mode, you need to explicitly embed the expression files in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the route below would load an XPath expression from a classpath resource named **myxpath.txt**:

```
from("direct:start").transform().xpath("resource:classpath:myxpath.txt");
```

To include this (an possibly other expressions stored in **.txt** files) in the native image, you would have to add something like the following to your **application.properties** file:

```
quarkus.native.resources.includes = *.txt
```

2.82. XSLT

Transforms XML payload using an XSLT template.

2.82.1. What’s inside

- XSLT component, URI syntax: **xslt:resourceUri**

Please refer to the above link for usage and configuration details.

2.82.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-xslt</artifactId>
</dependency>
```

▪

2.82.3. Additional Camel Quarkus configuration

To optimize XSLT processing, the extension needs to know the locations of the XSLT templates at build time. The XSLT source URIs have to be passed via the **quarkus.camel.xslt.sources** property. Multiple URIs can be separated by comma.

```
quarkus.camel.xslt.sources = transform.xsl, classpath:path/to/my/file.xsl
```

Scheme-less URIs are interpreted as **classpath:** URIs.

Only **classpath:** URIs are supported on Quarkus native mode. **file:**, **http:** and other kinds of URIs can be used on JVM mode only.

<xsl:include> and **<xsl:messaging>** XSLT elements are also supported in JVM mode only right now.

If **aggregate** DSL is used, **XsltSaxonAggregationStrategy** has to be used such as

```
from("file:src/test/resources?noop=true&sortBy=file:name&antInclude=*.xml")
   .routeId("aggregate").noAutoStartup()
   .aggregate(new XsltSaxonAggregationStrategy("xslt/aggregate.xsl"))
   .constant(true)
   .completionFromBatchConsumer()
   .log("after aggregate body: ${body}")
   .to("mock:transformed");
```

Also, it’s only supported on JVM mode.

2.82.3.1. Configuration

TransformerFactory features can be configured using following property:

```
quarkus.camel.xslt.features."http\://javax.xml.XMLConstants/feature/secure-processing"=false
```

2.82.3.2. Extension functions support

Xalan’s extension functions do work properly only when:

1. Secure-processing is disabled
2. Functions are defined in a separate jar
3. Functions are augmented during native build phase. For example, they can be registered for reflection:

```
@RegisterForReflection(targets = { my.Functions.class })
public class FunctionsConfiguration {
}
```

NOTE

The content of the XSLT source URIs is parsed and compiled into Java classes at build time. These Java classes are the only source of XSLT information at runtime. The XSLT source files may not be included in the application archive at all.

Configuration property	Type	Default
🔒 **quarkus.camel.xslt.sources** A comma separated list of templates to compile.	**string**	
🔒 **quarkus.camel.xslt.package-name** The package name for the generated classes.	**string**	**org.apache.camel.quarkus.component.xslt.generated**
🔒 **quarkus.camel.xslt.features** TransformerFactory features.	**Map<String,Boolean>**	

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.83. YAML DSL

A YAML stack for parsing YAML route definitions

2.83.1. What's inside

- YAML DSL

Please refer to the above link for usage and configuration details.

2.83.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-yaml-dsl</artifactId>
</dependency>
```

2.83.3. Usage

2.83.3.1. Native mode

The following constructs when defined within Camel YAML DSL markup, require you to register classes for reflection. Refer to the Native mode guide for details.

2.83.3.1.1. Bean definitions

The YAML DSL provides the capability to define beans as follows.

```
- beans:
    - name: "greetingBean"
      type: "org.acme.GreetingBean"
      properties:
        greeting: "Hello World!"
- route:
    id: "my-yaml-route"
    from:
      uri: "timer:from-yaml?period=1000"
      steps:
        - to: "bean:greetingBean"
```

In this example, the **GreetingBean** class needs to be registered for reflection. This applies to any types that you refer to under the **beans** key in your YAML routes.

```
@RegisterForReflection
public class GreetingBean {
}
```

2.83.3.1.2. Exception handling

Camel provides various methods of handling exceptions. Some of these require that any exception classes referenced in their DSL definitions are registered for reflection.

on-exception

```
- on-exception:
    handled:
      constant: "true"
    exception:
      - "org.acme.MyHandledException"
    steps:
      - transform:
          constant: "Sorry something went wrong"
```

```
@RegisterForReflection
public class MyHandledException {
}
```

throw-exception

```
- route:
    id: "my-yaml-route"
    from:
      uri: "direct:start"
      steps:
        - choice:
            when:
              - simple: "${body} == 'bad value'"
                steps:
                  - throw-exception:
                      exception-type: "org.acme.ForcedException"
                      message: "Forced exception"
            otherwise:
              steps:
                - to: "log:end"
```

```
@RegisterForReflection
public class ForcedException {
}
```

do-catch

```
- route:
    id: "my-yaml-route2"
    from:
      uri: "direct:tryCatch"
      steps:
        - do-try:
            steps:
              - to: "direct:readFile"
            do-catch:
              - exception:
                  - "java.io.FileNotFoundException"
                steps:
                  - transform:
                      constant: "do-catch caught an exception"
```

```
@RegisterForReflection(targets = FileNotFoundException.class)
public class MyClass {
}
```

2.84. ZIP FILE

Compression and decompress streams using java.util.zip.ZipStream.

2.84.1. What's inside

- Zip File data format

Please refer to the above link for usage and configuration details.

2.84.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-zipfile</artifactId>
</dependency>
```