Red Hat Integration 2023.q2

Getting Started with Camel K

Develop and run your first Camel K application

Last Updated: 2023-07-03

Red Hat Integration 2023.q2 Getting Started with Camel K

Develop and run your first Camel K application

Legal Notice

Abstract

How to install Camel K, set up your development environment, and run example applications.

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. INTRODUCTION TO CAMEL K

This chapter introduces the concepts, features, and cloud-native architecture provided by Red Hat Integration - Camel K:

- Section 1.1, “Camel K overview”
- Section 1.2, “Camel K features”
- Section 1.2.3, “Kamelets”
- Section 1.3, “Camel K development tooling”
- Section 1.4, “Camel K distributions”

1.1. CAMEL K OVERVIEW

Red Hat Integration - Camel K is a lightweight integration framework built from Apache Camel K that runs natively in the cloud on OpenShift. Camel K is specifically designed for serverless and microservice architectures. You can use Camel K to instantly run your integration code written in Camel Domain Specific Language (DSL) directly on OpenShift. Camel K is a subproject of the Apache Camel open source community: https://github.com/apache/camel-k.

Camel K is implemented in the Go programming language and uses the Kubernetes Operator SDK to automatically deploy integrations in the cloud. For example, this includes automatically creating services and routes on OpenShift. This provides much faster turnaround times when deploying and redeploying integrations in the cloud, such as a few seconds or less instead of minutes.

The Camel K runtime provides significant performance optimizations. The Quarkus cloud-native Java framework is enabled by default to provide faster start up times, and lower memory and CPU footprints. When running Camel K in developer mode, you can make live updates to your integration DSL and view results instantly in the cloud on OpenShift, without waiting for your integration to redeploy.

Using Camel K with OpenShift Serverless and Knative Serving, containers are created only as needed and are autoscaled under load up and down to zero. This reduces cost by removing the overhead of server provisioning and maintenance and enables you to focus on application development instead.

Using Camel K with OpenShift Serverless and Knative Eventing, you can manage how components in your system communicate in an event-driven architecture for serverless applications. This provides flexibility and creates efficiencies through decoupled relationships between event producers and consumers using a publish-subscribe or event-streaming model.

Additional resources

- Apache Camel K website
- Getting started with OpenShift Serverless

1.2. CAMEL K FEATURES

The Camel K includes the following main platforms and features:

1.2.1. Platform and component versions

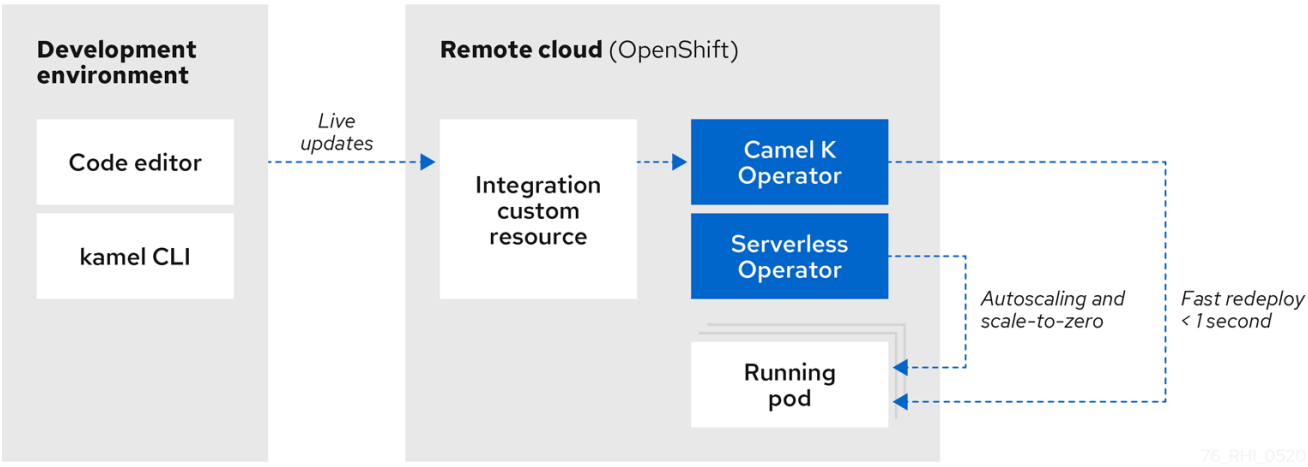
- OpenShift Container Platform 4.6, 4.9, 4.10, 4.11

- OpenShift Serverless 1.29.0
- Red Hat Build of Quarkus 2.7.6
- Red Hat Camel Extensions for Quarkus 2.7.1
- Apache Camel K 1.8.0
- Apache Camel 3.14.2
- OpenJDK 11

1.2.2. Camel K features

- Knative Serving for autoscaling and scale-to-zero
- Knative Eventing for event-driven architectures
- Performance optimizations using Quarkus runtime by default
- Camel integrations written in Java or YAML DSL
- Development tooling with Visual Studio Code
- Monitoring of integrations using Prometheus in OpenShift
- Quickstart tutorials
- Kamelet Catalog of connectors to external systems such as AWS, Jira, and Salesforce

The following diagram shows a simplified view of the Camel K cloud-native architecture:



Additional resources

- Apache Camel architecture

1.2.3. Kamelets

Kamelets hide the complexity of connecting to external systems behind a simple interface, which contains all the information needed to instantiate them, even for users who are not familiar with Camel.

Kamelets are implemented as custom resources that you can install on an OpenShift cluster and use in Camel K integrations. Kamelets are route templates that use Camel components designed to connect to external systems without requiring deep understanding of the component. Kamelets abstract the details of connecting to external systems. You can also combine Kamelets to create complex Camel integrations, just like using standard Camel components.

Additional resources

- Integrating Applications with Kamelets

1.3. CAMEL K DEVELOPMENT TOOLING

The Camel K provides development tooling extensions for Visual Studio (VS) Code, Red Hat CodeReady WorkSpaces, and Eclipse Che. The Camel-based tooling extensions include features such as automatic completion of Camel DSL code, Camel K modeline configuration, and Camel K traits.

The following VS Code development tooling extensions are available:

- VS Code Extension Pack for Apache Camel by Red Hat
    - Tooling for Apache Camel K extension
    - Language Support for Apache Camel extension
    - Debug Adapter for Apache Camel K
    - Additional extensions for OpenShift, Java and more

For details on how to set up these VS Code extensions for Camel K, see Setting up your Camel K development environment.

IMPORTANT

- The following plugin VS Code Language support for Camel - a part of the Camel extension pack provides support for content assist when editing Camel routes and **application.properties**.
- To install a supported Camel K tooling extension for VS code to create, run and operate Camel K integrations on OpenShift, see VS Code Tooling for Apache Camel K by Red Hat extension
- To install a supported Camel debug tool extension for VS code to debug Camel integrations written in Java, YAML or XML locally, see Debug Adapter for Apache Camel by Red Hat
- For details about configurations and components to use the developer tool with specific product versions, see Camel K Supported Configurations and Camel K Component Details

Note: The Camel K VS Code extensions are community features.

Eclipse Che also provides these features using the **vscode-camelk** plug-in.

For more information about scope of development support, see Development Support Scope of Coverage

Additional resources

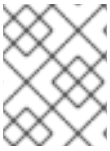
- VS Code tooling for Apache Camel K example
- Eclipse Che tooling for Apache Camel K

1.4. CAMEL K DISTRIBUTIONS

Table 1.1. Red Hat Integration - Camel K distributions

Distribution	Description	Location
Operator image	Container image for the Red Hat Integration - Camel K Operator: **integration/camel-k-rhel8-operator**	• OpenShift web console under **Operators → OperatorHub** • registry.redhat.io
Maven repository	Maven artifacts for Red Hat Integration - Camel K Red Hat provides Maven repositories that host the content we ship with our products. These repositories are available to download from the software downloads page. For Red Hat Integration - Camel K the following repositories are required: • rhi-common • rhi-camel-quarkus • rhi-camel-k Installation of Red Hat Integration - Camel K in a disconnected environment (offline mode) is not supported.	Software Downloads for Red Hat Integration
Source code	Source code for Red Hat Integration - Camel K	Software Downloads for Red Hat Integration

Distribution	Description	Location
Quickstarts	Quick start tutorials: • Basic Java integration • Event streaming integration • JDBC integration • JMS integration • Kafka integration • Knative integration • SaaS integration • Serverless API integration • Transformations integration	https://github.com/openshift-integration



NOTE

You must have a subscription for Red Hat Integration and be logged into the Red Hat Customer Portal to access the Red Hat Integration - Camel K distributions.

CHAPTER 2. PREPARING YOUR OPENSHIFT CLUSTER

This chapter explains how to install Red Hat Integration - Camel K and OpenShift Serverless on OpenShift, and how to install the required Camel K and OpenShift Serverless command-line client tools in your development environment.

2.1. INSTALLING CAMEL K

You can install the Red Hat Integration - Camel K Operator on your OpenShift cluster from the OperatorHub. The OperatorHub is available from the OpenShift Container Platform web console and provides an interface for cluster administrators to discover and install Operators.

After you install the Camel K Operator, you can install the Camel K CLI tool for command line access to all Camel K features.

Prerequisites

- You have access to an OpenShift 4.6 (or later) cluster with the correct access level, the ability to create projects and install operators, and the ability to install CLI tools on your local system.

NOTE

> You do not need to create a pull secret when installing Camel K from the OpenShift OperatorHub. The Camel K Operator automatically reuses the OpenShift cluster-level authentication to pull the Camel K image from `registry.redhat.io`.

- You installed the OpenShift CLI tool (**oc**) so that you can interact with the OpenShift cluster at the command line. For details on how to install the OpenShift CLI, see Installing the OpenShift CLI.

Procedure

1. In the OpenShift Container Platform web console, log in by using an account with cluster administrator privileges.
2. Create a new OpenShift project:
   a. In the left navigation menu, click **Home** > **Project** > **Create Project**.
   b. Enter a project name, for example, **my-camel-k-project**, and then click **Create**.
3. In the left navigation menu, click **Operators** > **OperatorHub**.
4. In the **Filter by keyword** text box, type **Camel K** and then click the **Red Hat Integration - Camel K Operator** card.
5. Read the information about the operator and then click **Install**. The Operator installation page opens.

6. Select the following subscription settings:
   - **Update Channel** > **latest**
   - Choose among the following 2 options:
     - **Installation Mode** > **A specific namespace on the cluster**> **my-camel-k-project**
     - **Installation Mode** > **All namespaces on the cluster (default)**> **Openshift operator**

NOTE

If you do not choose among the above two options, the system by default chooses a global namespace on the cluster then leading to openshift operator.

   - **Approval Strategy** > **Automatic**

NOTE

The **Installation mode** > **All namespaces on the cluster**and **Approval Strategy** > **Manual** settings are also available if required by your environment.

7. Click **Install**, and wait a few moments until the Camel K Operator is ready for use.
8. Download and install the Camel K CLI tool:
   a. From the **Help menu (?)** at the top of the OpenShift web console, select **Command line tools**.
   b. Scroll down to the **kamel - Red Hat Integration - Camel K - Command Line Interface** section.
   c. Click the link to download the binary for your local operating system (Linux, Mac, Windows).
   d. Unzip and install the CLI in your system path.
   e. To verify that you can access the Kamel K CLI, open a command window and then type the following:
      kamel --help

      This command shows information about Camel K CLI commands.

Next step

(optional) Specifying Camel K resource limits

2.1.1. Specifying Camel K resource limits

When you install Camel K, the OpenShift pod for Camel K does not have any limits set for CPU and memory (RAM) resources. If you want to define resource limits for Camel K, you must edit the Camel K subscription resource that was created during the installation process.

Prerequisite

- You have cluster administrator access to an OpenShift project in which the Camel K Operator is installed as described in Installing Camel K.
- You know the resource limits that you want to apply to the Camel K subscription. For more information about resource limits, see the following documentation:
  - Setting deployment resources in the OpenShift documentation.
  - Managing Resources for Containers in the Kubernetes documentation.

Procedure

1. Log in to the OpenShift Web console.
2. Select **Operators** > **Installed Operators** > **Operator Details** > **Subscription**.
3. Select **Actions** > **Edit Subscription**.
   The file for the subscription opens in the YAML editor.
4. Under the **spec** section, add a **config.resources** section and provide values for memory and cpu as shown in the following example:

```
spec:
  channel: default
  config:
    resources:
      limits:
        memory: 512Mi
        cpu: 500m
      requests:
        cpu: 200m
        memory: 128Mi
```

5. Save your changes.

OpenShift updates the subscription and applies the resource limits that you specified.

2.2. INSTALLING OPENSHIFT SERVERLESS

You can install the OpenShift Serverless Operator on your OpenShift cluster from the OperatorHub. The OperatorHub is available from the OpenShift Container Platform web console and provides an interface for cluster administrators to discover and install Operators.

The OpenShift Serverless Operator supports both Knative Serving and Knative Eventing features. For more details, see installing OpenShift Serverless Operator.

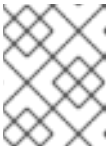
Prerequisites

- You have cluster administrator access to an OpenShift project in which the Camel K Operator is installed.
- You installed the OpenShift CLI tool (**oc**) so that you can interact with the OpenShift cluster at the command line. For details on how to install the OpenShift CLI, see Installing the OpenShift CLI.

Procedure

Procedure

1. In the OpenShift Container Platform web console, log in by using an account with cluster administrator privileges.
2. In the left navigation menu, click **Operators** > **OperatorHub**.
3. In the **Filter by keyword** text box, enter **Serverless** to find the **OpenShift Serverless Operator**.
4. Read the information about the Operator and then click **Install** to display the Operator subscription page.
5. Select the default subscription settings:
   - **Update Channel** > Select the channel that matches your OpenShift version, for example, **4.12**
   - **Installation Mode** > **All namespaces on the cluster**
   - **Approval Strategy** > **Automatic**

   NOTE

   The **Approval Strategy** > **Manual** setting is also available if required by your environment.
6. Click **Install**, and wait a few moments until the Operator is ready for use.
7. Install the required Knative components using the steps in the OpenShift documentation:
   - Installing Knative Serving
   - Installing Knative Eventing
8. (Optional) Download and install the OpenShift Serverless CLI tool:
   a. From the Help menu (?) at the top of the OpenShift web console, select **Command line tools**.
   b. Scroll down to the **kn - OpenShift Serverless - Command Line Interface**section.
   c. Click the link to download the binary for your local operating system (Linux, Mac, Windows)
   d. Unzip and install the CLI in your system path.
   e. To verify that you can access the **kn** CLI, open a command window and then type the following:
   kn --help

   This command shows information about OpenShift Serverless CLI commands.

   For more details, see the OpenShift Serverless CLI documentation.

Additional resources

- Installing OpenShift Serverless in the OpenShift documentation

2.3. CONFIGURING MAVEN REPOSITORY FOR CAMEL K

For Camel K operator, you can provide the Maven settings in a **ConfigMap** or a secret.

Procedure

1. To create a **ConfigMap** from a file, run the following command.

```
oc create configmap maven-settings --from-file=settings.xml
```

Created **ConfigMap** can be then referenced in the **IntegrationPlatform** resource, from the **spec.build.maven.settings** field.

Example

```
apiVersion: camel.apache.org/v1
kind: IntegrationPlatform
metadata:
  name: camel-k
spec:
  build:
    maven:
      settings:
        configMapKeyRef:
            key: settings.xml
            name: maven-settings
```

Or you can edit the **IntegrationPlatform** resource directly to reference the ConfigMap that contains the Maven settings using following command:

```
oc edit ip camel-k
```

Configuring CA certificates for remote Maven repositories

You can provide the CA certificates, used by the Maven commands to connect to the remote Maven repositories, in a Secret.

Procedure

1. Create a Secret from file using following command:

```
oc create secret generic maven-ca-certs --from-file=ca.crt
```

2. Reference the created Secret in the **IntegrationPlatform** resource, from the **spec.build.maven.caSecret** field as shown below.

```
apiVersion: camel.apache.org/v1
kind: IntegrationPlatform
metadata:
  name: camel-k
spec:
  build:
     maven:
```

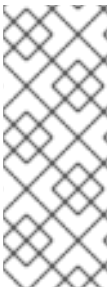
```
caSecret:
  key: tls.crt
  name: tls-secret
```

CHAPTER 3. DEVELOPING AND RUNNING CAMEL K INTEGRATIONS

This chapter explains how to set up your development environment and how to develop and deploy simple Camel K integrations written in Java and YAML. It also shows how to use the **kamel** command line to manage Camel K integrations at runtime. For example, this includes running, describing, logging, and deleting integrations.

3.1. SETTING UP YOUR CAMEL K DEVELOPMENT ENVIRONMENT

You must set up your environment with the recommended development tooling before you can automatically deploy the Camel K quick start tutorials. This section explains how to install the recommended Visual Studio (VS) Code IDE and the extensions that it provides for Camel K.



NOTE

- The Camel K VS Code extensions are community features.
- VS Code is recommended for ease of use and the best developer experience of Camel K. This includes automatic completion of Camel DSL code and Camel K traits. However, you can manually enter your code and tutorial commands using your chosen IDE instead of VS Code.

Prerequisites

- You must have access to an OpenShift cluster on which the Camel K Operator and OpenShift Serverless Operator are installed:
  - Installing Camel K
  - Installing OpenShift Serverless from the OperatorHub

Procedure

1. Install VS Code on your development platform. For example, on Red Hat Enterprise Linux:
   a. Install the required key and repository:

```
$ sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
$ sudo sh -c 'echo -e "[code]\nname=Visual Studio
Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\ngpgcheck=
\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
/etc/yum.repos.d/vscode.repo'
```

b. Update the cache and install the VS Code package:

```
$ yum check-update
$ sudo yum install code
```

For details on installing on other platforms, see the VS Code installation documentation.

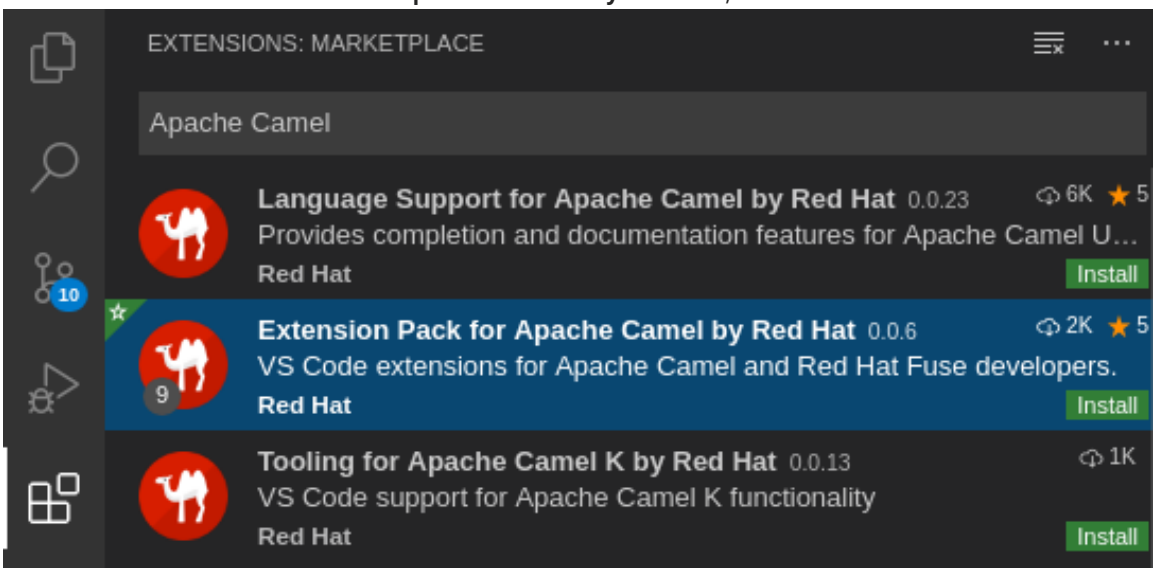
2. Enter the **code** command to launch the VS Code editor. For more details, see the VS Code command line documentation.

3. Install the VS Code Camel Extension Pack, which includes the extensions required for Camel K. For example, in VS Code:

   a. In the left navigation bar, click **Extensions**.

   b. In the search box, enter **Apache Camel**.

   c. Select the **Extension Pack for Apache Camel by Red Hat**, and click **Install**.



For more details, see the instructions for the Extension Pack for Apache Camel by Red Hat .

Additional resources

- VS Code Getting Started documentation
- VS Code Tooling for Apache Camel K by Red Hat extension
- VS Code Language Support for Apache Camel by Red Hat extension
- Apache Camel K and VS Code tooling example

3.2. DEVELOPING CAMEL K INTEGRATIONS IN JAVA

This section shows how to develop a simple Camel K integration in Java DSL. Writing an integration in Java to be deployed using Camel K is the same as defining your routing rules in Camel. However, you do not need to build and package the integration as a JAR when using Camel K.

You can use any Camel component directly in your integration routes. Camel K automatically handles the dependency management and imports all the required libraries from the Camel catalog using code inspection.

Prerequisites

- Setting up your Camel K development environment

Procedure

1. Enter the **kamel init** command to generate a simple Java integration file. For example:

```
$ kamel init HelloCamelK.java
```

2. Open the generated integration file in your IDE and edit as appropriate. For example, the **HelloCamelK.java** integration automatically includes the Camel **timer** and **log** components to help you get started:

```
// camel-k: language=java

import org.apache.camel.builder.RouteBuilder;

public class HelloCamelK extends RouteBuilder {
  @Override
  public void configure() throws Exception {

    // Write your routes here, for example:
    from("timer:java?period=1s")
      .routeId("java")
      .setBody()
        .simple("Hello Camel K from ${routeId}")
      .to("log:info");

  }
}
```

Next steps

- Running Camel K integrations

3.3. DEVELOPING CAMEL K INTEGRATIONS IN YAML

This section explains how to develop a simple Camel K integration in YAML DSL. Writing an integration in YAML to be deployed using Camel K is the same as defining your routing rules in Camel.

You can use any Camel component directly in your integration routes. Camel K automatically handles the dependency management and imports all the required libraries from the Camel catalog using code inspection.

Prerequisites

- Setting up your Camel K development environment

Procedure

1. Enter the **kamel init** command to generate a simple YAML integration file. For example:

```
$ kamel init hello.camelk.yaml
```

2. Open the generated integration file in your IDE and edit as appropriate. For example, the **hello.camelk.yaml** integration automatically includes the Camel **timer** and **log** components to help you get started:

```
# Write your routes here, for example:
- from:
    uri: "timer:yaml"
    parameters:
      period: "1s"
    steps:
      - set-body:
          constant: "Hello Camel K from yaml"
      - to: "log:info"
```

3.4. RUNNING CAMEL K INTEGRATIONS

You can run Camel K integrations in the cloud on your OpenShift cluster from the command line using the **kamel run** command.

Prerequisites

- Setting up your Camel K development environment .
- You must already have a Camel integration written in Java or YAML DSL.

Procedure

1. Log into your OpenShift cluster using the **oc** client tool, for example:

```
$ oc login --token=my-token --server=https://my-cluster.example.com:6443
```

2. Ensure that the Camel K Operator is running, for example:

```
$ oc get pod
NAME                                READY   STATUS    RESTARTS   AGE
camel-k-operator-86b8d94b4-pk7d6    1/1     Running   0          6m28s
```

3. Enter the **kamel run** command to run your integration in the cloud on OpenShift. For example:

Java example

```
$ kamel run HelloCamelK.java
integration "hello-camel-k" created
```

YAML example

```
$ kamel run hello.camelk.yaml
integration "hello" created
```

4. Enter the **kamel get** command to check the status of the integration:

```
$ kamel get
NAME      PHASE          KIT
hello     Building Kit   myproject/kit-bq666mjej725sk8sn12g
```

When the integration runs for the first time, Camel K builds the integration kit for the container image, which downloads all the required Camel modules and adds them to the image classpath.

5. Enter **kamel get** again to verify that the integration is running:

```
$ kamel get
NAME      PHASE   KIT
hello     Running myproject/kit-bq666mjej725sk8sn12g
```

6. Enter the **kamel log** command to print the log to **stdout**:

```
$ kamel log hello
[1] 2021-08-11 17:58:40,573 INFO  [org.apa.cam.k.Runtime] (main) Apache Camel K
Runtime 1.7.1.fuse-800025-redhat-00001
[1] 2021-08-11 17:58:40,653 INFO  [org.apa.cam.qua.cor.CamelBootstrapRecorder] (main)
bootstrap runtime: org.apache.camel.quarkus.main.CamelMainRuntime
[1] 2021-08-11 17:58:40,844 INFO  [org.apa.cam.k.lis.SourcesConfigurer] (main) Loading
routes from: SourceDefinition{name='camel-k-embedded-flow', language='yaml',
location='file:/etc/camel/sources/camel-k-embedded-flow.yaml', }
[1] 2021-08-11 17:58:41,216 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Routes startup summary (total:1 started:1)
[1] 2021-08-11 17:58:41,217 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Started route1 (timer://yaml)
[1] 2021-08-11 17:58:41,217 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Apache Camel 3.10.0.fuse-800010-redhat-00001 (camel-1) started in 136ms (build:0ms
init:100ms start:36ms)
[1] 2021-08-11 17:58:41,268 INFO  [io.quarkus] (main) camel-k-integration 1.6.6 on JVM
(powered by Quarkus 1.11.7.Final-redhat-00009) started in 2.064s.
[1] 2021-08-11 17:58:41,269 INFO  [io.quarkus] (main) Profile prod activated.
[1] 2021-08-11 17:58:41,269 INFO  [io.quarkus] (main) Installed features: [camel-bean,
camel-core, camel-k-core, camel-k-runtime, camel-log, camel-support-common, camel-timer,
camel-yaml-dsl, cdi]
[1] 2021-08-11 17:58:42,423 INFO  [info] (Camel (camel-1) thread #0 - timer://yaml)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from yaml]
...
```

7. Press **Ctrl-C** to terminate logging in the terminal.

Additional resources

- For more details on the **kamel run** command, enter **kamel run --help**
- For faster deployment turnaround times, see Running Camel K integrations in development mode
- For details of development tools to run integrations, see VS Code Tooling for Apache Camel K by Red Hat
- See also Managing Camel K integrations

Running An Integration Without CLI

You can run an integration without a CLI (Command Line Interface) and create an Integration Custom Resource with the configuration to run your application.

For example, execute the following sample route.

```
kamel run Sample.java -o yaml
```

It returns the expected Integration Custom Resource.

```
apiVersion: camel.apache.org/v1
kind: Integration
metadata:
  creationTimestamp: null
  name: my-integration
  namespace: default
spec:
  sources:
  - content: "
    import org.apache.camel.builder.RouteBuilder;
    public class Sample extends RouteBuilder {
      @Override
      public void configure()
      throws Exception {
        from(\"timer:tick\")
        .log(\"Hello Integration!\");
       }
      }"
    name: Sample.java
status: {}
```

Save this custom resource in a yaml file, **my-integration.yaml**. Now, run the integration that contains the Integration Custom Resource using the **oc** command line, the UI, or the API to call the OpenShift cluster. In the following example, **oc** CLI is used from the command line.

```
oc apply -f my-integration.yaml
...
integration.camel.apache.org/my-integration created
```

The operator runs the Integration.

NOTE

- Kubernetes supports Structural Schemas for CustomResourceDefinitions.
- For more details about Camel K traits see, Camel K trait configuration reference .

Schema changes on Custom Resources

The strongly-typed Trait API imposes changes on the following CustomResourceDefinitions: **integrations**, **integrationkits', and `integrationplatforms.**

Trait properties under **spec.traits.<trait-id>.configuration** are now defined directly under **spec.traits.<trait-id>.**

```
traits:
  container:
    configuration:
      enabled: true
      name: my-integration
```

↓ ↓ ↓

```
traits:
  container:
    enabled: true
    name: my-integration
```

Backward compatibility is possible in this implementation. To achieve backward compatibility, the **Configuration** field with **RawMessage** type is provided for each trait type, so that the existing integrations and resources are read from the new Camel K version.

When the old integrations and resources are read, the legacy configuration in each trait (if any) is migrated to the new Trait API fields. If the values are predefined on the new API fields, they precede the legacy ones.

```
type Trait struct {
    // Can be used to enable or disable a trait. All traits share this common property.
    Enabled *bool `property:"enabled" json:"enabled,omitempty"`

    // Legacy trait configuration parameters.
    // Deprecated: for backward compatibility.
    Configuration *Configuration `json:"configuration,omitempty"`
}

// Deprecated: for backward compatibility.
type Configuration struct {
    RawMessage `json:",inline"`
}
```

3.5. RUNNING CAMEL K INTEGRATIONS IN DEVELOPMENT MODE

You can run Camel K integrations in development mode on your OpenShift cluster from the command line. Using development mode, you can iterate quickly on integrations in development and get fast feedback on your code.

When you specify the **kamel run** command with the **--dev** option, this deploys the integration in the cloud immediately and shows the integration logs in the terminal. You can then change the code and see the changes automatically applied instantly to the remote integration Pod on OpenShift. The terminal automatically displays all redeployments of the remote integration in the cloud.

NOTE

The artifacts generated by Camel K in development mode are identical to those that you run in production. The purpose of development mode is faster development.

Prerequisites

- Setting up your Camel K development environment .
- You must already have a Camel integration written in Java or YAML DSL.

Procedure

1. Log into your OpenShift cluster using the **oc** client tool, for example:

```
$ oc login --token=my-token --server=https://my-cluster.example.com:6443
```

2. Ensure that the Camel K Operator is running, for example:

```
$ oc get pod
NAME                              READY   STATUS    RESTARTS   AGE
camel-k-operator-86b8d94b4-pk7d6   1/1     Running   0          6m28s
```

3. Enter the **kamel run** command with **--dev** to run your integration in development mode on OpenShift in the cloud. The following shows a simple Java example:

```
$ kamel run HelloCamelK.java --dev
Condition "IntegrationPlatformAvailable" is "True" for Integration hello-camel-k: test/camel-k
Integration hello-camel-k in phase "Initialization"
Integration hello-camel-k in phase "Building Kit"
Condition "IntegrationKitAvailable" is "True" for Integration hello-camel-k: kit-
c49sqn4apkb4qgn55ak0
Integration hello-camel-k in phase "Deploying"
Progress: integration "hello-camel-k" in phase Initialization
Progress: integration "hello-camel-k" in phase Building Kit
Progress: integration "hello-camel-k" in phase Deploying
Integration hello-camel-k in phase "Running"
Condition "DeploymentAvailable" is "True" for Integration hello-camel-k: deployment name is
hello-camel-k
Progress: integration "hello-camel-k" in phase Running
Condition "CronJobAvailable" is "False" for Integration hello-camel-k: different controller
strategy used (deployment)
Condition "KnativeServiceAvailable" is "False" for Integration hello-camel-k: different
controller strategy used (deployment)
Condition "Ready" is "False" for Integration hello-camel-k
Condition "Ready" is "True" for Integration hello-camel-k
[1] Monitoring pod hello-camel-k-7f85df47b8-js7cb
...
...
[1] 2021-08-11 18:34:44,069 INFO  [org.apa.cam.k.Runtime] (main) Apache Camel K
Runtime 1.7.1.fuse-800025-redhat-00001
[1] 2021-08-11 18:34:44,167 INFO  [org.apa.cam.qua.cor.CamelBootstrapRecorder] (main)
bootstrap runtime: org.apache.camel.quarkus.main.CamelMainRuntime
[1] 2021-08-11 18:34:44,362 INFO  [org.apa.cam.k.lis.SourcesConfigurer] (main) Loading
routes from: SourceDefinition{name='HelloCamelK', language='java',
location='file:/etc/camel/sources/HelloCamelK.java', }
[1] 2021-08-11 18:34:46,180 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Routes startup summary (total:1 started:1)
[1] 2021-08-11 18:34:46,180 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Started java (timer://java)
```

```
[1] 2021-08-11 18:34:46,180 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Apache Camel 3.10.0.fuse-800010-redhat-00001 (camel-1) started in 243ms (build:0ms
init:213ms start:30ms)
[1] 2021-08-11 18:34:46,190 INFO  [io.quarkus] (main) camel-k-integration 1.6.6 on JVM
(powered by Quarkus 1.11.7.Final-redhat-00009) started in 3.457s.
[1] 2021-08-11 18:34:46,190 INFO  [io.quarkus] (main) Profile prod activated.
[1] 2021-08-11 18:34:46,191 INFO  [io.quarkus] (main) Installed features: [camel-bean,
camel-core, camel-java-joor-dsl, camel-k-core, camel-k-runtime, camel-log, camel-support-
common, camel-timer, cdi]
[1] 2021-08-11 18:34:47,200 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
[1] 2021-08-11 18:34:48,180 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
[1] 2021-08-11 18:34:49,180 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
...
```

4. Edit the content of your integration DSL file, save your changes, and see the changes displayed instantly in the terminal. For example:

```
...
integration "hello-camel-k" updated
...
[2] 2021-08-11 18:40:54,173 INFO  [org.apa.cam.k.Runtime] (main) Apache Camel K
Runtime 1.7.1.fuse-800025-redhat-00001
[2] 2021-08-11 18:40:54,209 INFO  [org.apa.cam.qua.cor.CamelBootstrapRecorder] (main)
bootstrap runtime: org.apache.camel.quarkus.main.CamelMainRuntime
[2] 2021-08-11 18:40:54,301 INFO  [org.apa.cam.k.lis.SourcesConfigurer] (main) Loading
routes from: SourceDefinition{name='HelloCamelK', language='java',
location='file:/etc/camel/sources/HelloCamelK.java', }
[2] 2021-08-11 18:40:55,796 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Routes startup summary (total:1 started:1)
[2] 2021-08-11 18:40:55,796 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Started java (timer://java)
[2] 2021-08-11 18:40:55,797 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Apache Camel 3.10.0.fuse-800010-redhat-00001 (camel-1) started in 174ms (build:0ms
init:147ms start:27ms)
[2] 2021-08-11 18:40:55,803 INFO  [io.quarkus] (main) camel-k-integration 1.6.6 on JVM
(powered by Quarkus 1.11.7.Final-redhat-00009) started in 3.025s.
[2] 2021-08-11 18:40:55,808 INFO  [io.quarkus] (main) Profile prod activated.
[2] 2021-08-11 18:40:55,809 INFO  [io.quarkus] (main) Installed features: [camel-bean,
camel-core, camel-java-joor-dsl, camel-k-core, camel-k-runtime, camel-log, camel-support-
common, camel-timer, cdi]
[2] 2021-08-11 18:40:56,810 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
[2] 2021-08-11 18:40:57,793 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
...
```

5. Press **Ctrl-C** to terminate logging in the terminal.

Additional resources

- For more details on the **kamel run** command, enter **kamel run --help**

- For details of development tools to run integrations, see VS Code Tooling for Apache Camel K by Red Hat
- Managing Camel K integrations
- Configuring Camel K integration dependencies

3.6. RUNNING CAMEL K INTEGRATIONS USING MODELINE

You can use the Camel K modeline to specify multiple configuration options in a Camel K integration source file, which are executed at runtime. This creates efficiencies by saving you the time of re-entering multiple command line options and helps to prevent input errors.

The following example shows a modeline entry from a Java integration file that enables 3scale and limits the integration container memory.

Prerequisites

- Setting up your Camel K development environment
- You must already have a Camel integration written in Java or YAML DSL.

Procedure

1. Add a Camel K modeline entry to your integration file. For example:

   ThreeScaleRest.java

```
// camel-k: trait=3scale.enabled=true trait=container.limit-memory=256Mi (1)
import org.apache.camel.builder.RouteBuilder;

public class ThreeScaleRest extends RouteBuilder {

  @Override
  public void configure() throws Exception {
     rest().get("/")
       .route()
       .setBody().constant("Hello");
  }
}
```

   (1) Enables both the container and 3scale traits, to expose the route through 3scale and to limit the container memory.

2. Run the integration, for example:

```
kamel run ThreeScaleRest.java
```

   The **kamel run** command outputs any modeline options specified in the integration, for example:

```
Modeline options have been loaded from source files
Full command: kamel run ThreeScaleRest.java --trait=3scale.enabled=true --
trait=container.limit-memory=256Mi
```

Additional resources

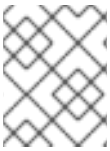
- Camel K modeline options
- For details of development tools to run modeline integrations, see Introducing IDE support for Apache Camel K Modeline.

CHAPTER 4. UPGRADING CAMEL K

You can upgrade installed Camel K operator automatically, but it does not automatically upgrade the Camel K integrations. You must manually trigger the upgrade for the Camel K integrations. This chapter explains how to upgrade both Camel K operator and Camel K integrations.

4.1. UPGRADING CAMEL K OPERATOR

The subscription of an installed Camel K operator specifies an update channel, for example, **1.6.0** channel, which is used to track and receive updates for the operator. To upgrade the operator to start tracking and receiving updates from a newer channel, you can change the update channel in the subscription. See Upgrading installed operators for more information about changing the update channel for installed operator.



NOTE

- Installed Operators cannot change to a channel that is older than the current channel.

If the approval strategy in the subscription is set to Automatic, the upgrade process initiates as soon as a new Operator version is available in the selected channel. If the approval strategy is set to Manual, you must manually approve pending upgrades.

Prerequisites

- Camel K operator is installed using Operator Lifecycle Manager (OLM).

Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Click the **Camel K Operator**.
3. Click the **Subscription** tab.
4. Click the name of the update channel under **Channel**.
5. Click the newer update channel that you want to change to. For example, **latest**. Click **Save**. This will start the upgrade to the latest Camel K version.

For subscriptions with an Automatic approval strategy, the upgrade begins automatically. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the upgrade. When complete, the status changes to Succeeded and Up to date.

For subscriptions with a Manual approval strategy, you can manually approve the upgrade from the Subscription tab.

4.2. UPGRADING CAMEL K INTEGRATIONS

When you trigger the upgrade for Camel K operator, the operator prepares the integrations to be upgraded, but does not trigger an upgrade for each one, to avoid service interruptions. When upgrading the operator, integration custom resources are not automatically upgraded to the newer version, so for

example, the operator may be at version **1.6.0**, while integrations report the **status.version** field of the custom resource the previous version **1.4.1**.

Prerequisites

Camel K operator is installed and upgraded using Operator Lifecycle Manager (OLM).

Procedure

- Open the terminal and run the following command to upgrade the Camel K intergations.

```
kamel rebuild myintegration
```

This will clear the status of the integration resource and the operator will start the deployment of the integration using the artifacts from upgraded version, for example, version **1.6.0**.

4.3. DOWNGRADING CAMEL K

You can downgrade to older version of Camel K operator by installing a previous version of the operator. This needs to be triggered manually using OC CLI. For more infromation about installing specific version of the operator using CLI see Installing a specific version of an Operator .

IMPORTANT

You must remove the existing Camel K operator and then install the specifc version of the operator as downgrading is not supported in OLM.

Once you install the older version of operator, use the **kamel rebuild** command to downgrade the integrations to the operator version. For example,

```
kamel rebuild myintegration
```

CHAPTER 5. CAMEL K QUICK START DEVELOPER TUTORIALS

Red Hat Integration - Camel K provides quick start developer tutorials based on integration use cases available from https://github.com/openshift-integration. This chapter provides details on how to set up and deploy the following tutorials:

- Section 5.1, "Deploying a basic Camel K Java integration"
- Section 5.2, "Deploying a Camel K Serverless integration with Knative"
- Section 5.3, "Deploying a Camel K transformations integration"
- Section 5.4, "Deploying a Camel K Serverless event streaming integration"
- Section 5.5, "Deploying a Camel K Serverless API-based integration"
- Section 5.6, "Deploying a Camel K SaaS integration"
- Section 5.7, "Deploying a Camel K JDBC integration"
- Section 5.8, "Deploying a Camel K JMS integration"
- Section 5.9, "Deploying a Camel K Kafka integration"

5.1. DEPLOYING A BASIC CAMEL K JAVA INTEGRATION

This tutorial demonstrates how to run a simple Java integration in the cloud on OpenShift, apply configuration and routing to an integration, and run an integration as a Kubernetes CronJob.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-basic/tree/1.6.x.
- You must have installed the Camel K operator and the **kamel** CLI. See Installing Camel K .
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See link:Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-basic.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-basic**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying basic Camel K Java integration .

Additional resources

- Developing Camel K integrations in Java

5.2. DEPLOYING A CAMEL K SERVERLESS INTEGRATION WITH KNATIVE

This tutorial demonstrates how to deploy Camel K integrations with OpenShift Serverless in an event-driven architecture. This tutorial uses a Knative Eventing broker to communicate using an event publish-subscribe pattern in a Bitcoin trading demonstration.

This tutorial also shows how to use Camel K integrations to connect to a Knative event mesh with multiple external systems. The Camel K integrations also use Knative Serving to automatically scale up and down to zero as needed.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-knative/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K and OpenShift Serverless:
  - Installing Camel K
  - Installing OpenShift Serverless from the OperatorHub
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-knative.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-knative**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K Knative integration .

Additional resources

- About Knative Eventing
- About Knative Serving

5.3. DEPLOYING A CAMEL K TRANSFORMATIONS INTEGRATION

This tutorial demonstrates how to run a Camel K Java integration on OpenShift that transforms data such as XML to JSON, and stores it in a database such as PostgreSQL.

The tutorial example uses a CSV file to query an XML API and uses the data collected to build a valid GeoJSON file, which is stored in a PostgreSQL database.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-transformations/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K. See Installing Camel K .
- You must follow the instructions in the tutorial readme to install the PostgreSQL Operator by Dev4Ddevs.com, which is required on your OpenShift cluster
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-transformations.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-transformations**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K transformations integration .

Additional resources

- https://operatorhub.io/operator/postgresql-operator-dev4devs-com
- https://geojson.org/

5.4. DEPLOYING A CAMEL K SERVERLESS EVENT STREAMING INTEGRATION

This tutorial demonstrates using Camel K and OpenShift Serverless with Knative Eventing for an event-driven architecture.

The tutorial shows how to install Camel K and Serverless with Knative in an AMQ Streams cluster with an AMQ Broker cluster, and how to deploy an event streaming project to run a global hazard alert demonstration application.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-event-streaming/tree/1.6.x.

- You must have cluster administrator access to an OpenShift cluster to install Camel K and OpenShift Serverless:
  - Installing Camel K
  - Installing OpenShift Serverless from the OperatorHub
- You must follow the instructions in the tutorial readme to install the additional required Operators on your OpenShift cluster:
  - AMQ Streams Operator
  - AMQ Broker Operator
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-event-streaming.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-event-streaming**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K event stream integration .

Additional resources

- Red Hat AMQ documentation
- OpenShift Serverless documentation

5.5. DEPLOYING A CAMEL K SERVERLESS API-BASED INTEGRATION

This tutorial demonstrates using Camel K and OpenShift Serverless with Knative Serving for an API-based integration, and managing an API with 3scale API Management on OpenShift.

The tutorial shows how to configure Amazon S3-based storage, design an OpenAPI definition, and run an integration that calls the demonstration API endpoints.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-api/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K and OpenShift Serverless:
  - Installing Camel K

    - Installing OpenShift Serverless from the OperatorHub

- You can also install the optional Red Hat Integration - 3scale Operator on your OpenShift system to manage the API. See Deploying 3scale using the Operator.
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-api.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-api**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K API integration .

Additional resources

- Red Hat 3scale API Management documentation
- OpenShift Serverless documentation

5.6. DEPLOYING A CAMEL K SAAS INTEGRATION

This tutorial demonstrates how to run a Camel K Java integration on OpenShift that connects two widely-used Software as a Service (SaaS) providers.

The tutorial example shows how to integrate the Salesforce and ServiceNow SaaS providers using REST-based Camel components. In this simple example, each new Salesforce Case is copied to a corresponding ServiceNow Incident that includes the Salesforce Case Number.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-saas/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K. See Installing Camel K .
- You must have Salesforce login credentials and ServiceNow login credentials.
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-saas.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-saas**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions .
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K SaaS integration .

Additional resources

- https://www.salesforce.com/
- https://www.servicenow.com/

5.7. DEPLOYING A CAMEL K JDBC INTEGRATION

This tutorial demonstrates how to get started with Camel K and an SQL database via JDBC drivers. This tutorial shows how to set up an integration producing data into a Postgres database (you can use any relational database of your choice) and also how to read data from the same database.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-jdbc/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K.
  - Installing Camel K
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-jdbc.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-jdbc**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K JDBC integration .

Additional resources

- Create your own Postgres sample database .

5.8. DEPLOYING A CAMEL K JMS INTEGRATION

This tutorial demonstrates how to use JMS to connect to a message broker in order to consume and produce messages. There are two examples:

- JMS Sink: this tutorial demonstrates how to produce a message to a JMS broker.
- JMS Source: this tutorial demonstrates how to consume a message from a JMS broker.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-jms/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K.
  - Installing Camel K
- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-jms.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-jms**.
3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.
4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K JMS integration .

Additional resources

- JMS Sink
- JMS Source

5.9. DEPLOYING A CAMEL K KAFKA INTEGRATION

This tutorial demonstrates how to use Camel K with Apache Kafka. This tutorial demonstrates how to set up a Kafka Topic via Red Hat OpenShift Streams for Apache Kafka and to use it in conjunction with Camel K.

Prerequisites

- See the tutorial readme in GitHub: https://github.com/openshift-integration/camel-k-example-kafka/tree/1.6.x.
- You must have cluster administrator access to an OpenShift cluster to install Camel K.

- Installing Camel K

- Visual Studio (VS) Code is optional but recommended for the best developer experience. See Setting up your Camel K development environment .

Procedure

1. Clone the tutorial Git repository:

```
$ git clone git@github.com:openshift-integration/camel-k-example-kafka.git
```

2. In VS Code, select **File → Open Folder → camel-k-example-kafka**.

3. In the VS Code navigation tree, click the **readme.md** file. This opens a new tab in VS Code to display the tutorial instructions.

4. Follow the tutorial instructions.
   Alternatively, if you do not have VS Code installed, you can manually enter the commands from deploying Camel K Kafka integration .