Red Hat Integration 2022.Q1

Camel K のスタートガイド

初めての Camel K アプリケーションの開発および実行

Last Updated: 2023-03-25

Red Hat Integration 2022.Q1 Camel K のスタートガイド

初めての Camel K アプリケーションの開発および実行

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

Camel K のインストール、開発環境の設定、およびサンプルアプリケーションの実行方法を説明します。

目次

前書き

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージをご覧ください。

第1章 CAMEL K の紹介

本章では、Red Hat Integration - Camel K によって提供される概念、機能、およびクラウドネイティブアーキテクチャーについて紹介します。

- 「Camel K の概要」
- 「Camel K の機能」
- 「Kamelets」
- 「Camel K 開発ツール」
- 「Camel K ディストリビューション」

1.1. CAMEL K の概要

Red Hat Integration - Camel K は、OpenShift のクラウドでネイティブで実行される Apache Camel K からビルドされる軽量のインテグレーションフレームワークです。Camel K は、サーバーレスおよびマイクロサービスアーキテクチャー向けに特別に設計されています。Camel K を使用すると、Camel Domain Specific Language (DSL) で書かれたインテグレーションコードを直接 OpenShift で即座に実行することができます。Camel K は Apache Camel オープンソースコミュニティーのサブプロジェクトです (https://github.com/apache/camel-k)。

Camel K は Go プログラミング言語で実装され、Kubernetes Operator SDK を使用してクラウドにインテグレーションを自動的にデプロイします。たとえば、これには OpenShift でのサービスおよびルートの自動作成が含まれます。これにより、クラウドにインテグレーションをデプロイおよび再デプロイする際に、ターンアラウンドタイムが大幅に短縮されます (たとえば、数分から数秒未満に削減されます)。

Camel K ランタイムは、パフォーマンスを最適化します。Quarkus のクラウドネイティブ Java フレームワークは、起動時間を短縮し、メモリーおよび CPU フットプリントを削減するために、デフォルトで有効になっています。Camel K を開発者モードで実行する場合、インテグレーションが再デプロイされるまで待たずに、インテグレーション DSL のライブ更新を行うことができ、OpenShift のクラウドで結果を即時に表示できます。

Camel K を OpenShift Serverless および Knative Serving とともに使用すると、コンテナーは必要な場合のみ作成され、負荷の増減がゼロになるように自動スケーリングが使用されます。このため、サーバーのプロビジョニングとメンテナンスのオーバーヘッドがなくなり、コストが削減されるため、アプリケーションの開発に集中することができます。

Camel K を OpenShift Serverless および Knative Eventing とともに使用すると、システムのコンポーネントがサーバーレスアプリケーションのイベント駆動型アーキテクチャーで通信する方法を管理できます。これにより、パブリッシュ/サブスクライブモデルまたはイベントストリーミングモデルを使用したイベントプロデューサーとコンシューマー間の関係が切り離され、柔軟性と効率化を実現できます。

関連情報

- Apache Camel K の Web サイト
- OpenShift Serverless を使ってみる

1.2. CAMEL K の機能

Camel K には、以下の主要プラットフォームおよび機能が含まれています。

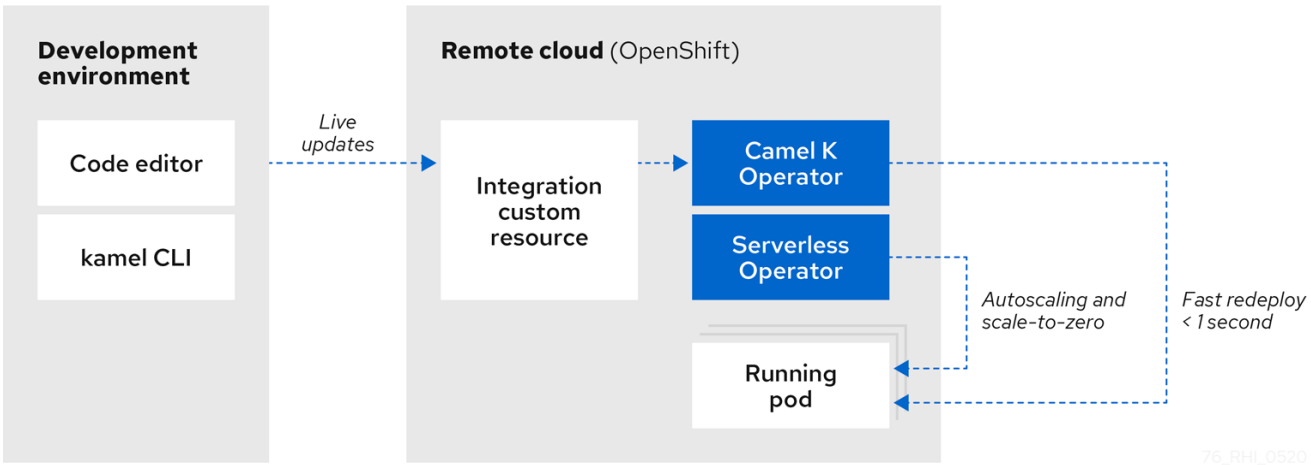
1.2.1. プラットフォームおよびコンポーネントのバージョン

- OpenShift Container 4.6 または Platform 4.9
- OpenShift Serverless 1.18.0
- Quarkus 2.2 Java ランタイム
- Apache Camel K 1.6
- Apache Camel 3.11.1
- Apache Camel Quarkus 2.2
- OpenJDK 11

1.2.2. Camel K の機能

- 自動スケーリングおよびゼロへのスケーリングを行うための Knative Serving
- イベント駆動型アーキテクチャーのための Knative Eventing
- デフォルトで Quarkus ランタイムを使用するパフォーマンスの最適化
- Java または YAML DSL で書かれた Camel インテグレーション
- Visual Studio Code での開発ツール
- OpenShift で Prometheus を使用したインテグレーションのモニタリング
- クイックスタートチュートリアル
- AWS、Jira、Salesforce などの外部システムへのコネクターの Kamelet Catalog

以下は、Camel K クラウドネイティブアーキテクチャーを簡単に表した図になります。



関連情報

- Apache Camel architecture

1.2.3. Kamelets

Kamelets は、Camel に精通していないユーザーであっても、インスタンス化するために必要なすべての情報が含まれるシンプルなインターフェースを使用することで、複雑な外部システムへの接続を単純化します。

Kamelets は、OpenShift クラスターにインストールでき、Camel K インテグレーションで使用できるカスタムリソースとして実装されます。Kamelets は、コンポーネントの詳細な理解なしに外部システムに接続するために設計された Camel コンポーネントを使用するルートテンプレートです。Kamelets は、外部システムへ接続する詳細を抽象化します。また、Kamelets を組み合わせることで、標準の Camel コンポーネントを使用するのと同じように、複雑な Camel インテグレーションを作成することもできます。

関連情報

- アプリケーションの Kamelets との統合

1.3. CAMEL K 開発ツール

Camel K は、Visual Studio (VS) Code、Visual Studio (VS) Code、Red Hat CodeReady WorkSpaces、および Eclipse Che の開発ツールエクステンションを提供します。Camel ベースのツールエクステンションには、Camel DSL コードの自動補完、Camel K モードライン設定、Camel K トレイトなどの機能が含まれます。Didact チュートリアルツールエクステンションは、Camel K クイックスタートのチュートリアルコマンドを自動実行します。

以下の VS Code 開発ツールエクステンションを使用できます。

- Red Hat による Apache Camel の VS Code エクステンションパック
  - Apache Camel K エクステンションのツール
  - Apache Camel エクステンションの言語サポート
  - OpenShift、Java 等の追加エクステンション
  - VS Code エクステンションの Didact チュートリアルツール

Camel K にこれらの VS Code エクステンションを設定する方法は、「Setting up your Camel K development environment」を参照してください。

注記: Camel K VS Code エクステンションはコミュニティー機能です。

Red Hat CodeReady Workspaces および Eclipse Che も、**vscode-camelk** プラグインを使用してこれらの機能を提供します。

関連情報

- Red Hat エクステンションによる Apache Camel K の VS Code ツール
- Apache Camel K サンプルの VS Code ツール
- Apache Camel K の Eclipse Che ツール
- Red Hat CodeReady WorkSpaces クラウドツール

1.4. CAMEL K ディストリビューション

表1.1 Red Hat Integration - Camel K ディストリビューション

ディストリビューション	説明	場所
Operator イメージ	Red Hat Integration - Camel K Operator のコンテナーイメージ: **integration/camel-k-rhel8-operator**	• **Operators → OperatorHub** の OpenShift Web コンソール • registry.redhat.io
Maven リポジトリー	Red Hat Integration - Camel K の Maven アーティファクト Red Hat は、Red Hat 製品に同梱されたコンテンツをホストする Maven リポジトリーを提供します。これらのリポジトリーは、ソフトウェアダウンロードページからダウンロードできます。 Red Hat Integration - Camel K の場合、以下のリポジトリーが必要です。 • rhi-common • rhi-camel-quarkus • rhi-camel-k 本リリースでは、Red Hat Integration - Camel K をオフラインモードでインストールすることはサポートされません。	Red Hat Integration のソフトウェアダウンロード
ソースコード	Red Hat Integration - Camel K のソースコード	Red Hat Integration のソフトウェアダウンロード

ディストリビューション	説明	場所
クイックスタート	クイックスタートチュートリアル: ● 基本的な Java インテグレーション ● イベントストリーミングのインテグレーション ● JDBC インテグレーション ● JMS インテグレーション ● Kafka インテグレーション ● Knative のインテグレーション ● SaaS のインテグレーション ● Serverless API のインテグレーション ● 変換のインテグレーション	https://github.com/openshift-integration

注記

Red Hat Integration - Camel K ディストリビューションにアクセスするには、Red Hat Integration のサブスクリプションが必要で、Red Hat カスタマーポータルにログインする必要があります。

第2章 OPENSHIFT クラスターの準備

本章では、Red Hat Integration - Camel K および OpenShift Serverless を OpenShift にインストールする方法と、開発環境に必要な Camel K および OpenShift Serverless コマンドラインクライアントツールをインストールする方法について説明します。

- 「Camel K のインストール」
- 「OpenShift Serverless のインストール」

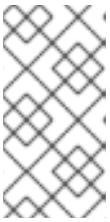
2.1. CAMEL K のインストール

OperatorHub から OpenShift クラスターで Red Hat Integration - Camel K Operator をインストールできます。OperatorHub は OpenShift Container Platform Web コンソールから使用でき、クラスター管理者が Operator を検出およびインストールするためのインターフェースを提供します。

Camel K Operator のインストール後に、コマンドラインですべての Camel K 機能にアクセスする Camel K CLI ツールをインストールできます。

前提条件

- 適切なアクセスレベルで OpenShift 4.6 (またはそれ以降の) クラスターにアクセスできること。この場合、プロジェクトの作成および Operator のインストールができること。また、CLI ツールをローカルシステムにインストールできること。



注記

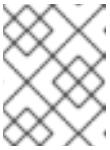
OpenShift OperatorHub から Camel K をインストールする場合は、プルシークレットを作成する必要はありません。Camel K Operator は OpenShift クラスターレベルの認証を自動的に再利用して、**registry.redhat.io** から Camel K イメージをプルします。

- コマンドラインで OpenShift クラスターと対話できるように OpenShift CLI ツール (**oc**) をインストールしていること。OpenShift CLI のインストール方法の詳細は、「Installing the OpenShift CLI」を参照してください。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. 新しい OpenShift プロジェクトを作成します。
   a. 左側のナビゲーションメニューで、**Home** > **Project** > **Create Project** とクリックします。
   b. プロジェクト名 (例: **my-camel-k-project**) を入力し、**Create** をクリックします。
3. 左側のナビゲーションメニューで、**Operators** > **OperatorHub** とクリックします。
4. **Filter by keyword** テキストボックスに **Camel K** を入力し、**Red Hat Integration - Camel K Operator** カードをクリックします。
5. Operator に関する情報を確認し、続いて **Install** をクリックします。Operator インストールページが開きます。

6. 以下のサブスクリプション設定を選択します。
   - **Update Channel** > **latest**
   - **Installation Mode** > **A specific namespace on the cluster**> **my-camel-k-project**
   - **Approval Strategy** > **Automatic**

   注記

   ご使用の環境で必要な場合は **Installation mode** > **All namespaces on the cluster** および **Approval Strategy** > **Manual** 設定も使用できます。

7. **Install** をクリックし、その後 Camel K Operator が使用できるようになるまでしばらく待ちます。
8. Camel K CLI ツールをダウンロードし、インストールします。
   a. OpenShift Web コンソールの上部にある Help メニュー (?) から、**Command line tools** を選択します。
   b. **kamel - Red Hat Integration - Camel K - Command Line Interface**セクションまでスクロールダウンします。
   c. ローカルのオペレーティングシステム (Linux、Mac、Windows) のバイナリーをダウンロードするためのリンクをクリックします。
   d. CLI を展開してシステムパスにインストールします。
   e. Kamel K CLI にアクセスできることを確認するには、コマンドウィンドウを開き、以下のコマンドを入力します。
      kamel --help

      このコマンドは、Camel K CLI コマンドに関する情報を表示します。

次のステップ

（任意設定）: Specifying Camel K resource limits

2.1.1. Specifying Camel K resource limits

Camel K をインストールする場合、Camel K 用の OpenShift Pod には、CPU およびメモリー (RAM) リソースの制限がありません。Camel K のリソース制限を定義する場合は、インストールプロセスで作成された Camel K サブスクリプションリソースを編集する必要があります。

前提条件

- 「 Installing Camel K 」で説明されているように、Camel K Operator がインストールされている OpenShift プロジェクトにクラスター管理者としてアクセスできる。
- Camel K サブスクリプションに適用するリソース制限を把握している。リソース制限の詳細は、以下のドキュメントを参照してください。
  - OpenShift ドキュメントの「 Setting deployment resources 」
  - Kubernetes ドキュメントの「Managing Resources for Containers」

手順

1. OpenShift Web コンソールにログインします。
2. **Operators** > **Installed Operators** > **Operator Details** > **Subscription** の順に選択します。
3. **Actions** > **Edit Subscription** を選択します。
   YAML エディターでサブスクリプションのファイルが開きます。
4. **spec** セクションで、以下の例のように **config.resources** セクションを追加し、メモリーと CPU の値を指定します。

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

5. 変更を保存します。

OpenShift はサブスクリプションを更新し、指定したリソース制限を適用します。

2.2. OPENSHIFT SERVERLESS のインストール

OperatorHub から OpenShift クラスターに OpenShift Serverless Operator をインストールできます。OperatorHub は OpenShift Container Platform Web コンソールから使用でき、クラスター管理者が Operator を検出およびインストールするためのインターフェースを提供します。

OpenShift Serverless Operator は、Knative Serving および Knative Eventing 機能の両方をサポートします。詳細は、OpenShift Serverless Operator のインストール について参照してください。

前提条件

- Camel K Operator がインストールされている OpenShift プロジェクトにクラスター管理者としてアクセスできる。
- コマンドラインで OpenShift クラスターと対話できるように OpenShift CLI ツール (**oc**) をインストールしていること。OpenShift CLI のインストール方法の詳細は、「Installing the OpenShift CLI」を参照してください。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. 左側のナビゲーションメニューで、**Operators** > **OperatorHub** とクリックします。
3. **Filter by keyword** テキストボックスで **Serverless** を入力し、**OpenShift Serverless Operator** を見つけます。

4. Operator に関する情報を読み、**Install** をクリックして Operator サブスクリプションページを表示します。

5. デフォルトのサブスクリプション設定を選択します。

   - **Update Channel** > (**4.9** など、OpenShift バージョンと一致するチャンネルを選択)
   - **Installation Mode** > **All namespaces on the cluster**
   - **Approval Strategy** > **Automatic**

   注記

   ご使用の環境で必要な場合は **Approval Strategy** > **Manual** 設定も使用できます。

6. **Install** をクリックし、Operator が使用できるようになるまでしばらく待ちます。

7. OpenShift ドキュメントの手順に従って、必要な Knative コンポーネントをインストールします。

   - Knative Serving のインストール
   - Knative Eventing のインストール

8. (任意) OpenShift Serverless CLI ツールをダウンロードし、インストールします。

   a. OpenShift Web コンソールの上部にある Help メニュー (?) から、**Command line tools** を選択します。

   b. **kn - OpenShift Serverless - Command Line Interface**セクションまでスクロールダウンします。

   c. ローカルのオペレーティングシステム (Linux、Mac、Windows) のバイナリーをダウンロードするためのリンクをクリックします。

   d. CLI を展開してシステムパスにインストールします。

   e. **kn** CLI にアクセスできることを確認するには、コマンドウィンドウを開き、以下のコマンドを入力します。
   kn --help

   このコマンドは、OpenShift Serverless CLI コマンドに関する情報を表示します。

   詳細は、OpenShift Serverless CLI のドキュメント を参照してください。

関連情報

- OpenShift ドキュメントの「OpenShift Serverless のインストール 」

2.3. CAMEL K の MAVEN リポジトリーの設定

Camel K オペレーターの場合、**ConfigMap**またはシークレットで Maven 設定を提供できます。

手順

1. ファイルから **ConfigMap**を作成するには、次のコマンドを実行します。

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

作成された **ConfigMap** は、**spec.build.maven.settings** フィールドから **IntegrationPlatform** リソースで参照できます。

例

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

または、次のコマンドを使用して、**IntegrationPlatform** リソースを直接編集し、Maven 設定を含む ConfigMap を参照することができます。

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

リモート Maven リポジトリーの CA 証明書の設定

Maven コマンドがリモート Maven リポジトリーに接続するために使用する CA 証明書をシークレットで提供できます。

手順

1. 次のコマンドを使用して、ファイルからシークレットを作成します。

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

2. 以下に示すように、**spec.build.maven.caSecret** フィールドから、**IntegrationPlatform** リソースで作成したシークレットを参照します。

```
apiVersion: camel.apache.org/v1
kind: IntegrationPlatform
metadata:
  name: camel-k
spec:
  build:
    maven:
      caSecret:
        key: tls.crt
        name: tls-secret
```

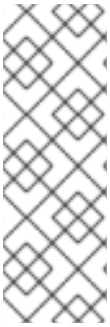
第3章 CAMEL K インテグレーションの開発および実行

本章では、開発環境を設定する方法と、Java および YAML で書かれた簡単な Camel K インテグレーションを開発およびデプロイする方法を説明します。また、**kamel** コマンドラインを使用して起動時に Camel K インテグレーションを管理する方法も説明します。たとえば、これには、インテグレーションの実行、記述、ログ、および削除が含まれます。

- 「Camel K 開発環境の設定」
- 「Java での Camel K インテグレーションの開発」
- 「YAML での Camel K インテグレーションの開発」
- 「Camel K インテグレーションの実行」
- 「開発モードでの Camel K インテグレーションの実行」
- 「モードラインを使用した Camel K インテグレーションの実行」

3.1. CAMEL K 開発環境の設定

Camel K クイックスタートチュートリアルを自動的にデプロイする前に、推奨される開発ツールで環境を設定する必要があります。ここでは、推奨の Visual Studio (VS) コード IDE と Camel K に提供されるエクステンションをインストールする方法について説明します。



注記

- Camel K VS Code エクステンションはコミュニティー機能です。
- VS Code は使いやすく、Camel K の開発者エクスペリエンスが優れているため推奨されます。これには、Camel DSL コードと Camel K トレイトの自動補完や、チュートリアルのコマンドの自動実行が含まれます。ただし、VS Code の代わりに選択した IDE を使用して、コードおよびチュートリアルコマンドを手動で入力することができます。

前提条件

- Camel K Operator および OpenShift Serverless Operator がインストールされている OpenShift クラスターにアクセスできる必要があります。
  - Camel K のインストール
  - OperatorHub からの OpenShift Serverless のインストール

手順

1. 開発プラットフォームに VS Code をインストールします。たとえば、Red Hat Enterprise Linux の場合は次のようにインストールします。
   a. 必要なキーおよびリポジトリーをインストールします。

```
$ 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. キャッシュを更新し、VS Code パッケージをインストールします。

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

他のプラットフォームにインストールする場合の詳細は、VS Code のインストールに関するドキュメント を参照してください。

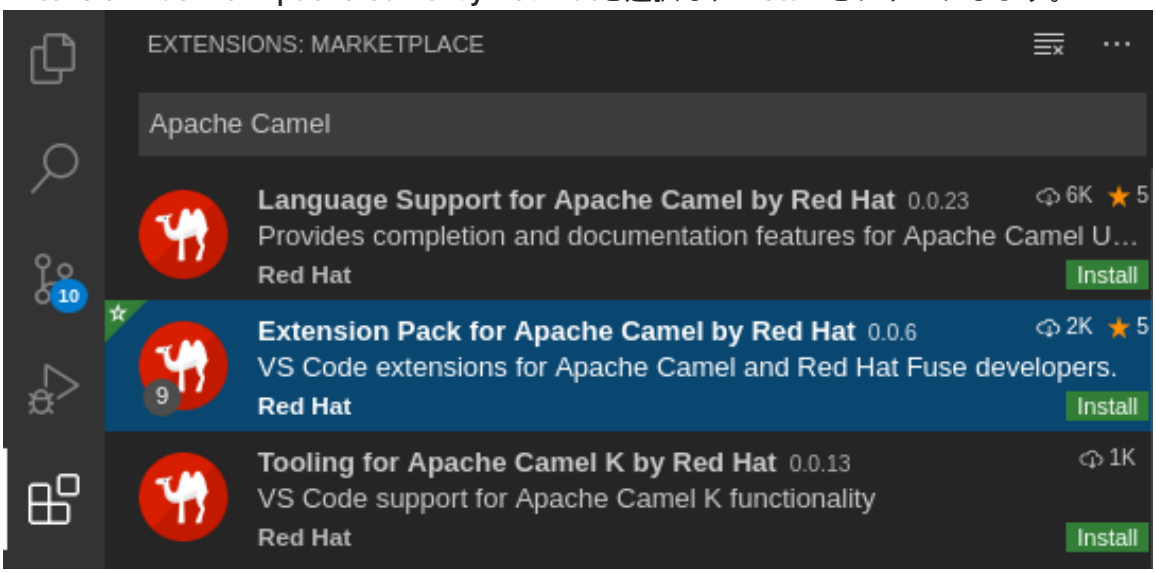
2. **code** コマンドを入力して VS Code エディターを起動します。詳細は、 VS Code コマンドラインのドキュメント を参照してください。

3. Camel K に必要なエクステンションが含まれる VS Code Camel Extension Pack をインストールします。たとえば、VS Code で以下を行います。

a. 左側のナビゲーションバーで **Extensions** をクリックします。

b. 検索ボックスに **Apache Camel** と入力します。

c. **Extension Pack for Apache Camel by Red Hat**を選択し、**Install** をクリックします。



詳細は、「RExtension Pack for Apache Camel by Red Hat 」手順を参照してください。

4. VS Code Didact エクステンションをインストールします。これを使用すると、チュートリアルのリンクをクリックしてクイックスタートのチュートリアルコマンドを自動的に実行できます。たとえば、VS Code で以下を行います。

a. 左側のナビゲーションバーで **Extensions** をクリックします。

b. 検索ボックスに **Didact** と入力します。

c. エクステンションを選択し、**Install** をクリックします。
詳細は、Didact エクステンション の説明を参照してください。

関連情報

- VS Code の Getting Started
- Red Hat エクステンションによる Apache Camel K の VS Code ツール

- Red Hat エクステンションによる Apache Camel の VS Code 言語サポート
- Apache Camel K および VS Code ツールの例

3.2. JAVA での CAMEL K インテグレーションの開発

ここでは、Java DSL で簡単な Camel K インテグレーションを開発する方法を説明します。Camel K を使用して、Java でデプロイするインテグレーションを作成することは、Camel でルーティングルールを定義することと同じです。しかし、Camel K を使用する場合は、インテグレーションを JAR としてビルドおよびパッケージ化する必要はありません。

インテグレーションルートで Camel コンポーネントを直接使用できます。Camel K は依存関係管理を自動的に処理し、コード検査を使用して Camel カタログから必要なライブラリーをすべてインポートします。

前提条件

- Setting up your Camel K development environment

手順

1. **kamel init** コマンドを入力して、簡単な Java インテグレーションファイルを生成します。以下は例になります。

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

2. IDE で生成されたインテグレーションファイルを開き、必要に応じて編集します。たとえば、すぐに使えるように、**HelloCamelK.java** インテグレーションには Camel **timer** および **log** コンポーネントが自動的に含まれます。

```
// 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");

  }
}
```

次のステップ

- Camel K インテグレーションの実行

3.3. YAML での CAMEL K インテグレーションの開発

ここでは、YAML DSL で簡単な Camel K インテグレーションを開発する方法を説明します。Camel K を使用して、デプロイするインテグレーションを YAML で作成することは、Camel でルーティングルールを定義することと同じです。

インテグレーションルートで Camel コンポーネントを直接使用できます。Camel K は依存関係管理を自動的に処理し、コード検査を使用して Camel カタログから必要なライブラリーをすべてインポートします。

前提条件

- Setting up your Camel K development environment

手順

1. **kamel init** コマンドを入力して、簡単な YAML インテグレーションファイルを生成します。以下は例になります。

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

2. IDE で生成されたインテグレーションファイルを開き、必要に応じて編集します。たとえば、すぐに使えるように、**hello.camelk.yaml** インテグレーションには Camel **timer** および **log** コンポーネントが自動的に含まれます。

```
# 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. CAMEL K インテグレーションの実行

kamel run コマンドを使用すると、コマンドラインから OpenShift クラスターのクラウドで Camel K インテグレーションを実行できます。

前提条件

- Camel K 開発環境の設定
- Java または YAML DSL で記述された Camel インテグレーションが作成済みである必要があります。

手順

1. 以下の例のように、**oc** クライアントツールを使用して OpenShift クラスターにログインします。

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

2. 以下の例のように、Camel K Operator が稼働していることを確認します。

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

3. **kamel run** コマンドを入力し、OpenShift のクラウドでインテグレーションを実行します。以下は例になります。

Java の例

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

YAML の例

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

4. **kamel get** コマンドを入力し、インテグレーションの状態を確認します。

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

インテグレーションが初めて実行されると、Camel K はコンテナーイメージのインテグレーションキットをビルドします。インテグレーションキットは、必要なすべての Camel モジュールをダウンロードし、イメージクラスパスに追加します。

5. **kamel get** を再度入力して、インテグレーションが稼働していることを確認します。

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

6. **kamel log** コマンドを入力して、ログを **stdout** に出力します。

```
$ 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 1.6.0 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. **Ctrl-C** キーを押して、ターミナルでログインを終了します。

関連情報

- **kamel run** コマンドの詳細については、**kamel run --help** を入力してください。
- デプロイメントのターンアラウンド時間を短縮するには、「Running Camel K integrations in development mode」を参照してください。
- インテグレーションを実行するための開発ツールの詳細は、Red Hat による Apache Camel K の VS Code ツール を参照してください。
- 「Managing Camel K integrations」も参照してください。

3.5. 開発モードでの CAMEL K インテグレーションの実行

開発モードの Camel K インテグレーションは、コマンドラインから OpenShift クラスターで実行できます。開発モードを使用すると、開発段階にてインテグレーションで繰り返しを迅速に行うことができ、コードに関するフィードバックを即座に受け取ることができます。

--dev オプションを指定して **kamel run** コマンドを実行すると、インテグレーションがクラウドで即座にデプロイされ、ターミナルにインテグレーションログが表示されます。次に、コードを変更でき、変更が自動的かつ即座に OpenShift のリモートインテグレーション Pod に適用されることを確認できます。ターミナルには、クラウドのリモートインテグレーションの再デプロイメントすべてが自動的に表示されます。

注記

開発モードで Camel K によって生成されたアーティファクトは、実稼働環境で実行するアーティファクトと同じです。開発モードの目的は、より迅速な開発を行うことです。

前提条件

- Camel K 開発環境の設定
- Java または YAML DSL で記述された Camel インテグレーションが作成済みである必要があります。

手順

1. 以下の例のように、**oc** クライアントツールを使用して OpenShift クラスターにログインします。

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

2. 以下の例のように、Camel K Operator が稼働していることを確認します。

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

3. **kamel run** コマンドと **--dev** を入力し、クラウドの OpenShift にて開発モードでインテグレーションを実行します。以下は、簡単な Java の例になります。

```
$ 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 1.6.0 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. インテグレーション DSL ファイルの内容を編集し、変更を保存します。ターミナルで変更が即座に表示されることを確認します。以下は例になります。

```
...
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 1.6.0 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. **Ctrl-C** キーを押して、ターミナルでログインを終了します。

関連情報

- **kamel run** コマンドの詳細については、**kamel run --help** を入力してください。
- インテグレーションを実行するための開発ツールの詳細は、Red Hat による Apache Camel K の VS Code ツール を参照してください。
- Camel K インテグレーションの管理
- Camel K インテグレーション依存関係の設定

3.6. モードラインを使用した CAMEL K インテグレーションの実行

Camel K モードラインを使用すると、起動時に実行される Camel K インテグレーションソースファイルに複数の設定オプションを指定できます。これにより、複数のコマンドラインオプションを再入力する時間を節約できるため効率を良くすることができ、入力エラーを防ぐことができます。

以下の例は、3scale を有効にし、インテグレーションコンテナーのメモリーを制限する Java インテグレーションファイルからのモードラインエントリーを示しています。

前提条件

- Setting up your Camel K development environment
- Java または YAML DSL で記述された Camel インテグレーションが作成済みである必要があります。

手順

1. Camel K モードラインエントリーをインテグレーションファイルに追加します。以下に例を示します。

   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) コンテナーと 3scale トレイトの両方を有効にし、3scale からルートを公開し、コンテナーメモリーを制限します。

2. インテグレーションを実行します。以下に例を示します。

```
kamel run ThreeScaleRest.java
```

   kamel run コマンドは、インテグレーションで指定されたモードオプションを出力します。以下に例を示します。

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

関連情報

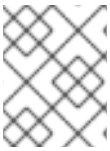
- Camel K モードラインオプション
- モードラインのインテグレーションを実行するための開発ツールの詳細は、「Introducing IDE support for Apache Camel K Modeline」を参照してください。

第4章 CAMEL K のアップグレード

インストールされた Camel K Operator を自動的にアップグレードできますが、Camel K インテグレーションは自動的にアップグレードされません。Camel K インテグレーションのアップグレードを手動でトリガーする必要があります。本章では、Camel K Operator と Camel K インテグレーションの両方をアップグレードする方法を説明します。

4.1. CAMEL K OPERATOR のアップグレード

インストールされた Camel K Operator のサブスクリプションは、Operator の更新を追跡し、受信するために使用される更新チャネル (例: **1.6.0** チャネル) を指定します。Operator をアップグレードして新規チャネルからの更新の追跡および受信を開始するために、サブスクリプションで更新チャネルを変更できます。インストールされた Operator の更新チャネルの変更に関する詳細は、「インストールされた Operator のアップグレード」を参照してください。



注記

- インストールされた Operator は、現在のチャネルよりも古いチャネルに切り換えることはできません。

サブスクリプションの承認ストラテジーが Automatic に設定されている場合、アップグレードプロセスは、選択したチャネルで新規 Operator バージョンが利用可能になるとすぐに開始します。承認ストラテジーが Manual に設定されている場合、保留中のアップグレードを手動で承認する必要があります。

前提条件

- Camel K Operator が Operator Lifecycle Manager (OLM) を使用してインストールされている。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. **Camel K Operator** をクリックします。
3. **Subscription** タブをクリックします。
4. **Channel** の下にある更新チャネルの名前をクリックします。
5. 変更する新しい更新チャネルをクリックします たとえば、**latest** です。**保存** をクリックします。これにより、最新の Camel K バージョンへのアップグレードが開始されます。

Automatic 承認ストラテジーのあるサブスクリプションの場合、アップグレードは自動的に開始します。**Operators → Installed Operators**ページに戻り、アップグレードの進捗をモニターします。完了時に、ステータスは Succeeded および Up to date に変更されます。

Manual 承認ストラテジーのあるサブスクリプションの場合、Subscription タブからアップグレードを手動で承認できます。

4.2. CAMEL K インテグレーションのアップグレード

Camel K Operator のアップグレードをトリガーすると、Operator はインテグレーションのアップグレードを準備しますが、サービスの中断を避けるためにアップグレードはトリガーされません。

Operator をアップグレードする際に、インテグレーションカスタムリソースは自動的に新しいバージョンにアップグレードされないため、たとえば Operator は **1.6.0** バージョンであるのに対して、インテグレーションは、カスタムリソースの **status.version** フィールドに以前のバージョンの **1.4.1** を報告する可能性があります。

前提条件

Camel K Operator が Operator Lifecycle Manager (OLM) を使用してインストールおよびアップグレードされている。

手順

- ターミナルを開き、以下のコマンドを実行し、Camel K インテグレーションをアップグレードします。

```
kamel rebuild myintegration
```

これにより、インテグレーションリソースのステータスがクリアされ、Operator はアップグレードされたバージョン (例: バージョン **1.6.0**) からのアーティファクトを使用してインテグレーションのデプロイメントを開始します。

4.3. CAMEL K のダウングレード

以前のバージョンのオペレーターをインストールすることにより、古いバージョンの Camel K オペレーターにダウングレードできます。これは、OCCLI を使用して手動でトリガーする必要があります。CLI を使用した特定バージョンの Operator のインストールについての詳細は、「Operator の特定のバージョンのインストール」を参照して ください。

重要

OLM ではダウングレードがサポートされていないため、既存の Camel K オペレーターを削除してから、特定のバージョンのオペレーターをインストールする必要があります。

古いバージョンの operator をインストールしたら、**kamel rebuild** コマンドを使用して統合を operator バージョンにダウングレードします。以下に例を示します。

```
kamel rebuild myintegration
```

第5章 CAMEL K クイックスタートの開発者チュートリアル

Red Hat Integration - Camel K は、https://github.com/openshift-integration のインテグレーションユースケースを基にした、クイックスタートの開発者チュートリアルを提供します。本章では、以下のチュートリアルを設定およびデプロイする方法について説明します。

- 「基本的な Camel K Java インテグレーションのデプロイ」
- 「Knative での Camel K Serverless インテグレーションのデプロイ」
- 「Camel K 変換インテグレーションのデプロイ」
- 「Camel K Serverless イベントストリーミングインテグレーションのデプロイ」
- 「Camel K Serverless API ベースのインテグレーションのデプロイ」
- 「Camel K SaaS インテグレーションのデプロイ」
- 「Camel K JDBC 統合のデプロイ」
- 「Camel K JMS 統合のデプロイメント」
- 「Camel K Kafka 統合のデプロイ」

5.1. 基本的な CAMEL K JAVA インテグレーションのデプロイ

このチュートリアルでは、OpenShift のクラウドで簡単な Java インテグレーションを実行する方法、設定およびルーティングをインテグレーションに適用する方法、およびインテグレーションを Kubernetes CronJob として実行する方法を実証します。

前提条件

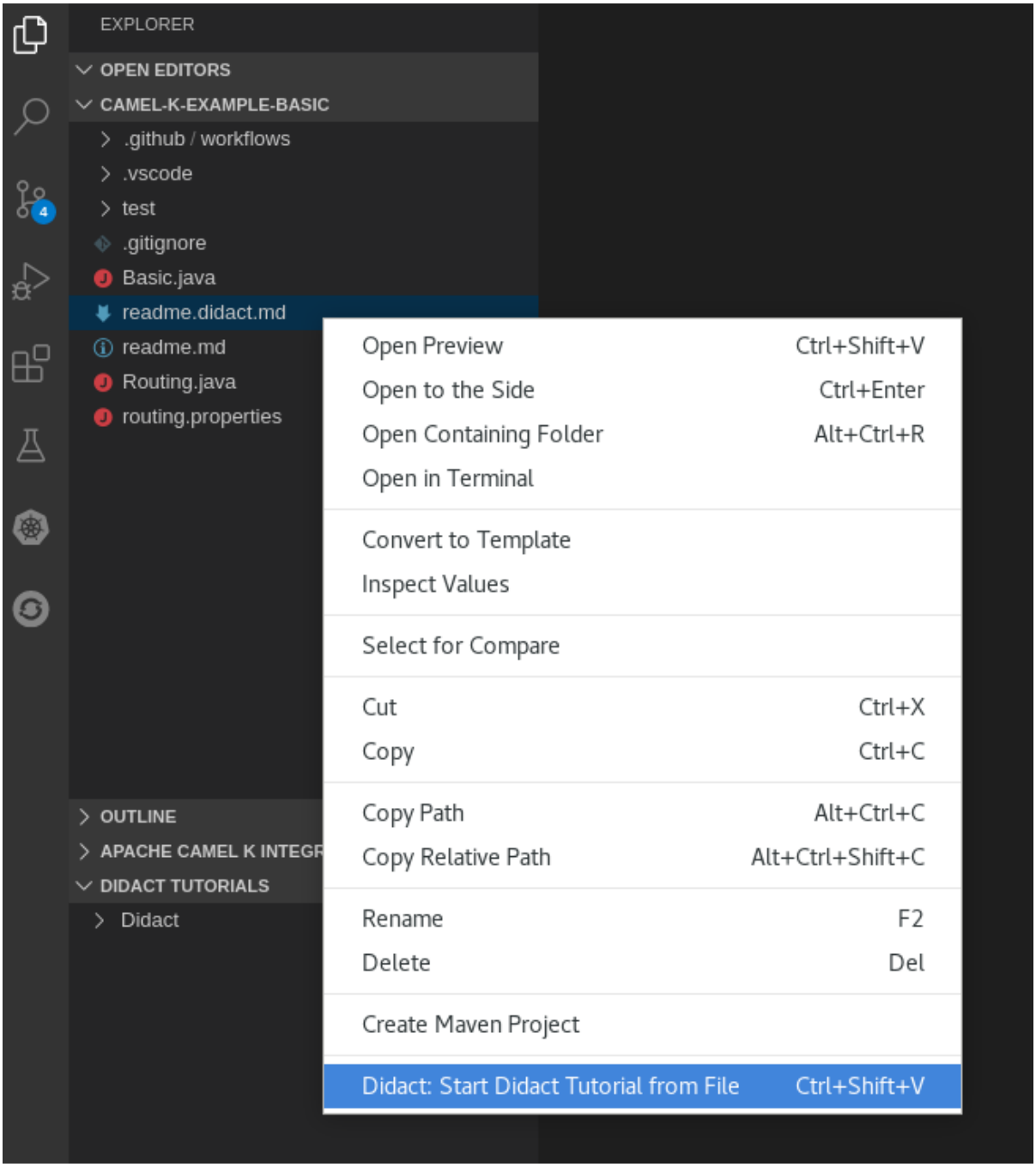
- GitHub https://github.com/openshift-integration/camel-k-example-basic/tree/1.6.x のチュートリアル readme を確認します。
- Camel K Operator および **kamel** CLI がインストールされている必要があります。「 Installing Camel K 」を参照してください。
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-basic** と選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。以下は例になります。



これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。

4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-basic からコマンドを手動で入力できます。

関連情報

- Java での Camel K インテグレーションの開発

5.2. KNATIVE での CAMEL K SERVERLESS インテグレーションのデプロイ

このチュートリアルでは、イベント駆動型のアーキテクチャーで OpenShift Serverless と Camel K イン

テグレーションをデプロイする方法を説明します。このチュートリアルでは、Bitcoin 取引のデモンストレーションでイベントパブリッシュ/サブスクライブパターンを使用して、通信に Knative Eventing ブローカーを使用します。

このチュートリアルでは、Camel K インテグレーションを使用して、複数の外部システムを持つ Knative イベントメッシュに接続する方法についても説明します。Camel K インテグレーションは、必要に応じて自動的にゼロにスケーリングするために Knative Serving も使用します。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-knative/tree/1.6.x のチュートリアル readme を確認します。
- Camel K および OpenShift Serverless をインストールするために、クラスター管理者として OpenShift クラスターにアクセスできる必要があります。
  - Camel K のインストール
  - OperatorHub からの OpenShift Serverless のインストール
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-knative** を選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。
4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
   VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-native からコマンドを手動で入力できます。

関連情報

- Knative Eventing について
- Knative Serving について

5.3. CAMEL K 変換インテグレーションのデプロイ

このチュートリアルでは、XML などのデータを JSON に変換し、PostgreSQL などのデータベースに保存する Camel K Java インテグレーションを OpenShift で実行する方法を実証します。

チュートリアルの例では、CSV ファイルを使用して XML API をクエリーし、収集したデータを使用して、PostgreSQL データベースに保存される有効な GeoJSON ファイルをビルドします。

前提条件

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-transformations/tree/1.6.x のチュートリアル readme を確認します。
- Camel K をインストールするためにクラスター管理者として OpenShift クラスターにアクセスできる必要があります。「 Installing Camel K 」を参照してください。
- チュートリアルの readme の手順に従い、OpenShift クラスターで必要な Dev4Ddevs.com による PostgreSQL Operator をインストールする必要があります。
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-transformations** を選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。
4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
   VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-transformations からコマンドを手動で入力できます。

関連情報

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

5.4. CAMEL K SERVERLESS イベントストリーミングインテグレーションのデプロイ

このチュートリアルでは、イベント駆動型のアーキテクチャーに Camel K と Knative Eventing のある OpenShift Serverless を使用する方法を実証します。

このチュートリアルでは、Camel K と Knative のある Serverless を AMQ Broker クラスターのある AMQ Streams クラスターにインストールする方法と、イベントストリーミングプロジェクトをデプロイして、グローバルハザードアラートデモンストレーションアプリケーションを実行する方法も実証します。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-event-streaming/tree/1.6.x のチュートリアル readme を確認します。

- Camel K および OpenShift Serverless をインストールするために、クラスター管理者として OpenShift クラスターにアクセスできる必要があります。
  - Camel K のインストール
  - OperatorHub からの OpenShift Serverless のインストール
- チュートリアル readme の手順に従って、OpenShift クラスターで必要な追加の Operator をインストールする必要があります。
  - AMQ Streams Operator
  - AMQ Broker Operator
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-event-streaming** と選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。
4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
   VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-event-streaming からコマンドを手動で入力できます。

関連情報

- Red Hat AMQ のドキュメント
- OpenShift Serverless のドキュメント

5.5. CAMEL K SERVERLESS API ベースのインテグレーションのデプロイ

このチュートリアルでは、API ベースのインテグレーションに Camel K と Knative Serving のある OpenShift Serverless を使用する方法と、OpenShift で 3scale API Management のある API を管理する方法を実証します。

このチュートリアルは、Amazon S3 ベースのストレージの設定方法、OpenAPI 定義の設計方法、およびデモンストレーション API エイドポイントを呼び出すインテグレーションの実行方法を示しています。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-api/tree/1.6.x のチュートリアル readme を確認します。

- Camel K および OpenShift Serverless をインストールするために、クラスター管理者として OpenShift クラスターにアクセスできる必要があります。
  - Camel K のインストール
  - OperatorHub からの OpenShift Serverless のインストール
- 任意の Red Hat Integration - 3scale Operator を OpenShift システムにインストールして API を管理することも可能。「Deploying 3scale using the Operator」を参照してください。
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-api** と選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。
4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
   VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-api からコマンドを手動で入力できます。

関連情報

- Red Hat 3scale API Management ドキュメント
- OpenShift Serverless のドキュメント

5.6. CAMEL K SAAS インテグレーションのデプロイ

このチュートリアルでは、広く使用されている 2 つのソフトウェアを SaaS (Software as a Service) プロバイダーとして接続する Camel K Java インテグレーションを OpenShift で実行する方法を実証します。

チュートリアルの例は、REST ベースの Camel コンポーネントを使用して Salesforce および ServiceNow の SaaS プロバイダーを統合する方法を示しています。簡単なこの例では、新しい Salesforce Case はそれぞれ Salesforce Case Number が含まれる該当の ServiceNow Incident にコピーされます。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-saas/tree/1.6.x のチュートリアル readme を確認します。
- Camel K をインストールするためにクラスター管理者として OpenShift クラスターにアクセスできる必要があります。「 Installing Camel K 」を参照してください。

- Salesforce のログインクレデンシャルと ServiceNow のログインクレデンシャルが必要です。
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-saas** と選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。
4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
   VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-saas からコマンドを手動で入力できます。

関連情報

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

5.7. CAMEL K JDBC 統合のデプロイ

このチュートリアルでは、JDBC ドライバーを介して Camel K と SQL データベースの使用を開始する方法を示します。このチュートリアルでは、Postgres データベースにデータを生成する統合をセットアップする方法 (任意のリレーショナルデータベースを使用できる) と、同じデータベースからデータを読み取る方法を示します。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-jdbc/tree/1.6.x のチュートリアル readme を確認します。
- Camel K をインストールするためにクラスター管理者として OpenShift クラスターへアクセスできる必要があります。
  - Camel K のインストール
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-jdbc** と選択します。

3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。

4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-jdbc からコマンドを手動で入力できます。

関連情報

- 独自の Postgres サンプルデータベースの作成

5.8. CAMEL K JMS 統合のデプロイメント

このチュートリアルでは、JMS を使用してメッセージブローカーに接続し、メッセージを消費および生成する方法を示します。2 つの例があります。

- JMS シンク: このチュートリアルでは、JMS ブローカーへのメッセージを生成する方法を示します。
- JMS ソース: このチュートリアルは、JMS ブローカーからのメッセージを使用する方法を示しています。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-jms/tree/1.6.x のチュートリアル readme を確認します。
- Camel K をインストールするためにクラスター管理者として OpenShift クラスターへアクセスできる必要があります。
  - Camel K のインストール
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-jms** と選択します。

3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。

4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。

VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-jms からコマンドを手動で入力できます。

関連情報

- JMS シンク
- JMS ソース

5.9. CAMEL K KAFKA 統合のデプロイ

このチュートリアルでは、Apache Kafka で Camel K を使用する方法を示します。このチュートリアルでは、Red Hat OpenShift Streams for Apache Kafka を介して Kafka トピックを設定し、Camel K と組み合わせて使用する方法を示します。

前提条件

- GitHub https://github.com/openshift-integration/camel-k-example-kafka/tree/1.6.x のチュートリアル readme を確認します。
- Camel K をインストールするためにクラスター管理者として OpenShift クラスターへアクセスできる必要があります。
  - Camel K のインストール
- Visual Studio (VS) コードは任意ですが、最良の開発者エクスペリエンスを提供するために推奨されます。「 Setting up your Camel K development environment 」を参照してください。

手順

1. チュートリアル Git リポジトリーのクローンを作成します。

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

2. VS Code で **File → Open Folder → camel-k-example-kafka** と選択します。
3. VS Code ナビゲーションツリーで、**readme.didact.md** ファイルを右クリックし、**Didact: Start Didact Tutorial from File** を選択します。これにより、VS Code に新しい Didact タブが開き、チュートリアルの手順が表示されます。
4. チュートリアルの手順に従い、提供されるリンクをクリックしてコマンドを自動的に実行します。
   VS Code が Didact エクステンションとともにインストールされていない場合には、https://github.com/openshift-integration/camel-k-example-kafka からコマンドを手動で入力できます。