Red Hat Integration 2020-Q4

OpenShift での Camel K インテグレーションのデプロイ

テクノロジープレビューの Red Hat Integration - Camel K を初めて使用するユーザー向け

Last Updated: 2022-07-30

Red Hat Integration 2020-Q4 OpenShift での Camel K インテグレーションのデプロイ

テクノロジープレビューの Red Hat Integration - Camel K を初めて使用するユーザー向け

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

本ガイドは、Red Hat Integration - Camel K を紹介し、OpenShift のインストール方法を説明します。さらに、初心者向けに、Camel K インテグレーションをデプロイする方法、および OpenShift Serverless でチュートリアルを使用する方法について説明します。本ガイドでは、Camel K インテグレーションの設定方法および監視方法についても説明し、高度な機能に設定できる Camel K トレイト (trait) の参照詳細を提供します。

目次

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」
- 「テクノロジープレビューの Camel K の機能」
- 「Camel K クラウドネイティブアーキテクチャー」
- 「Camel K 開発ツール」
- 「Camel K ディストリビューション」

重要

Red Hat Integration - Camel K はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。Red Hat は実稼働環境でこれらを使用することを推奨していません。

これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、「テクノロジプレビュー機能のサポート範囲」を参照してください。

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. テクノロジープレビューの CAMEL K の機能

テクノロジーリビューの 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 Platform 4.6
- OpenShift Serverless 1.7
- Quarkus 1.7 Java runtime
- Camel 3.5
- Java 11

1.2.2. テクノロジープレビューの機能

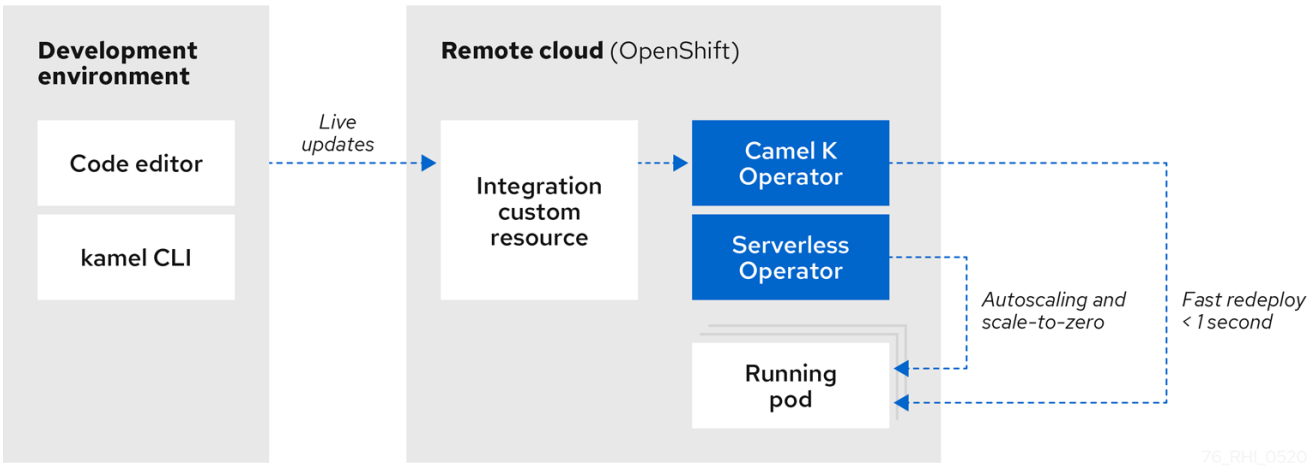
- 自動スケーリングおよびゼロへのスケーリングを行うための Knative Serving
- イベント駆動型アーキテクチャーのための Knative Eventing
- デフォルトで Quarkus ランタイムを使用するパフォーマンスの最適化
- Java、XML、または YAML DSL で書かれた Camel インテグレーション
- Visual Studio Code での開発ツール
- OpenShift で Prometheus を使用したインテグレーションのモニタリング
- 新しい Transformations や SaaS の例が含まれるクイックスタートチュートリアル
- AWS、Jira、Salesforce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

注記

テクノロジープレビューには、OpenShift での Camel K インテグレーションイメージのビルドのみが含まれます。Buildah または Kaniko イメージビルダーでの Camel K のインストールはテクノロジープレビューには含まれず、コミュニティーのサポートのみの対象となります。

1.3. CAMEL K クラウドネイティブアーキテクチャー

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。



Camel K は、Kubernetes カスタムリソースで Camel インテグレーションを自動的にラップし、クラウドにアップロードします。このアーキテクチャーには、以下の利点があります。

- より迅速な開発サイクルを実現する OpenShift でのクラウドネイティブインテグレーションおよび開発者のエクスペリエンス。
- Camel K の自動インストールおよび Camel K Operator を使用したインテグレーションのデプロイメント。
- 再デプロイを必要としない、Camel K 開発者モードを使用したコードのライブ更新。
- OpenShift Serverless Operator を使用した Knative での自動スケーリングおよびゼロへのスケーリング。
- Quarkus Java ランタイムを使用したパフォーマンスの最適化およびコストの削減:
  - ビルド時におけるコードの事前コンパイルおよび事前初期化。
  - 起動、デプロイ、および再デプロイ時間の短縮。
  - メモリーおよび CPU フットプリントの節約。
- Camel インテグレーションコードの依存関係の自動解決。
- コマンドラインおよびモデルラインでの Camel K トレイトを使用した高度な機能の設定。

その他のリソース

- Apache Camel architecture

1.3.1. 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。これには、ルートテンプレートとして高度なコネクターが含まれます。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。

その他のリソース

- Apache Camel Kamelets

1.4. CAMEL K 開発ツール

テクノロジープレビューの 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、XML など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の詳細は、「Camel K 開発環境の設定」 を参照してください。

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

1.5. 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-tech-preview/camel-k-rhel8-operator**	• **Operators → OperatorHub** の OpenShift Web コンソール • registry.redhat.io

ディストリビューション	説明	場所
Maven リポジトリー	Red Hat Integration - Camel K の Maven アーティファクト	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
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 ● Kamelet カタログ ● 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 CAMEL K のインストール

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 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。

- 「OpenShift OperatorHub からの Camel K のインストール」
- 「OperatorHub からの OpenShift Serverless のインストール」
- 「Camel K および OpenShift コマンドラインツールのインストール」

2.1. OPENSHIFT OPERATORHUB からの 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 コンソールから利用でき、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。OperatorHub の詳細は、OpenShift ドキュメント を参照してください。

前提条件

- クラスター管理者として OpenShift 4.6 クラスターにアクセスできる必要があります。

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。

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** を見つけます。
5. 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。
6. 以下のサブスクリプション設定を選択します。
   - **Update Channel** > **techpreview**
   - **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、Camel K Operator が使用できるようになるまでしばらく待ちます。

その他のリソース

- Operator の OpenShift クラスターへの追加

2.2. OPERATORHUB からの 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 コンソールから利用でき、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。OperatorHub の詳細は、OpenShift ドキュメント を参照してください。

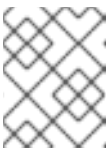
OpenShift Serverless Operator は、Knative Serving および Knative Eventing 機能の両方をサポートします。詳細は、「OpenShift Serverless の使用開始」を参照してください。

前提条件

- クラスター管理者として OpenShift 4.6 クラスターにアクセスできる必要があります。

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** > (**4.6** 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

その他のリソース

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

2.3. CAMEL K および OPENSHIFT コマンドラインツールのインストール

Camel K および OpenShift は、クラウドでインテグレーションをデプロイおよび管理するためのコマンドラインツールを提供します。ここでは、以下のコマンドラインインターフェース (CLI) ツールをインストールする方法を説明します。

- **kamel** - Camel K CLI
- **oc** - OpenShift Container Platform CLI
- **kn** - OpenShift Serverless CLI

これらのコマンドラインツールはすべて、Linux、Windows、および Mac で利用できます。

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。
  - 「OpenShift OperatorHub からの 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. OpenShift Container Platform Web コンソールで、開発者権限または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. ツールバーの  ヘルプアイコンをクリックし、**Command Line Tools** を選択します。
3. このツールがインストールされていない場合は、**oc** - OpenShift CLI アーカイブをダウンロードして展開します。詳細は、OpenShift CLI のドキュメント を参照してください。
4. このツールがインストールされていない場合は、**kn** - OpenShift Serverless CLI アーカイブをダウンロードし、展開します。詳細は、OpenShift Serverless CLI のドキュメント を参照してください。
5. **kamel** - Camel K CLI アーカイブをダウンロードして展開し、インストールします。
6. **kamel** バイナリーファイルをシステムパスに追加します。たとえば、Linux では、**/usr/bin** に **kamel** を配置することができます。
7. 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
```

8. 以下のコマンドを実行して、**kamel** クライアントツールのインストールを確認します。

```
$ kamel --help
```

関連情報

- OpenShift Container Platform CLI のドキュメント
- OpenShift Serverless CLI のドキュメント

第3章 CAMEL K の使用

本章では、開発環境を設定する方法と、Java、XML、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」
- 「XML 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

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。
  - 「OpenShift OperatorHub からの Camel K のインストール」
  - 「OperatorHub からの OpenShift Serverless のインストール」
- 「Camel K および OpenShift コマンドラインツールのインストール」

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。

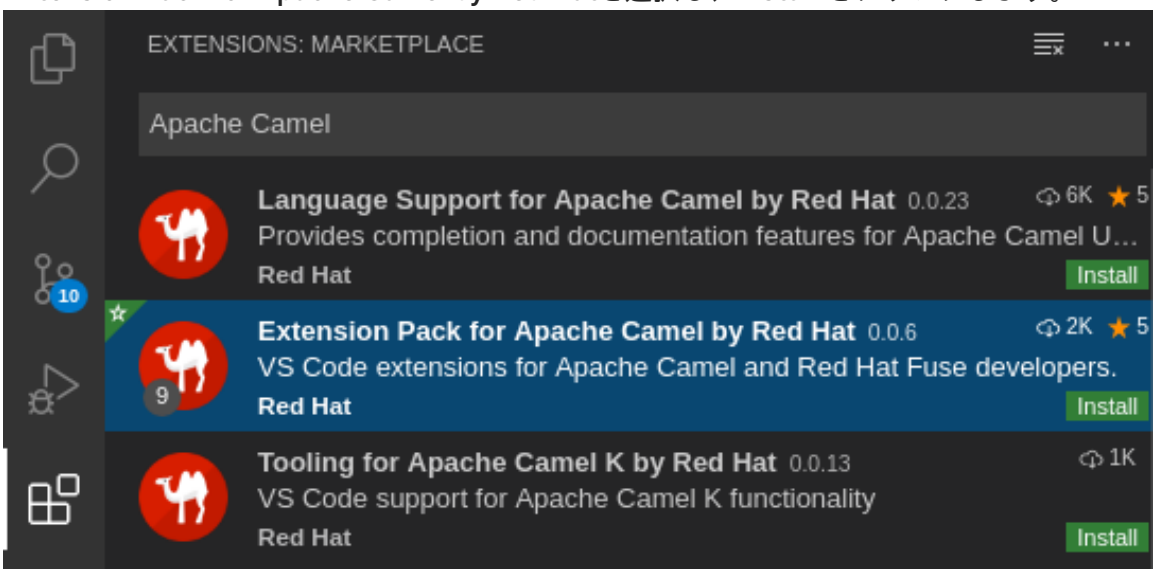
2. **code** コマンドを入力して VS Code エディターを起動します。詳細は、 VS Code コマンドラインのドキュメント を参照してください。

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

- 「Camel K 開発環境の設定」

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。以下に例を示します。

```
$ 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. XML での CAMEL K インテグレーションの開発

ここでは、従来の XML 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 XML 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

- 「Camel K 開発環境の設定」

手順

1. **kamel init** コマンドを入力して、シンプルなインテグレーションファイルを生成します。以下に例を示します。

```
$ kamel init hello-camel-k.xml
```

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、**hello-camel-k.xml** インテグレーションには、開始に役立つ Camel **タイマー** および **ログ** コンポーネントが自動的に含まれます。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- camel-k: language=xml -->

<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://camel.apache.org/schema/spring"
      xsi:schemaLocation="
         http://camel.apache.org/schema/spring
         http://camel.apache.org/schema/spring/camel-spring.xsd">

   <!-- Write your routes here, for example: -->
   <route id="xml">
      <from uri="timer:xml?period=1s"/>
      <setBody>
         <simple>Hello Camel K from ${routeId}</simple>
      </setBody>
      <to uri="log:info"/>
   </route>

</routes>
```

次のステップ

- 「Camel K インテグレーションの実行」

3.4. 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

- 「Camel K 開発環境の設定」

手順

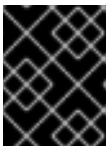
1. **kamel init** コマンドを入力して、シンプルなインテグレーションファイルを生成します。以下に例を示します。

```
$ 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。

```
# camel-k: language=yaml

# 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"
```



重要

YAML で作成されたインテグレーションには、パターン ***.camelk.yaml** または **# camel-k: language=yaml** の最初の行が含まれるファイル名が必要です。

関連情報

- Apache Camel インテグレーションを YAML で作成

3.5. 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、XML、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。以下に例を示します。

Java の例

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

XML の例

```
$ kamel run hello-camel-k.xml
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    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 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] 2020-04-11 14:26:43.449 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.PropertiesFunctionsConfigurer@5223e5ee executed in phase
Starting
[1] 2020-04-11 14:26:43.457 INFO  [main] RuntimeSupport - Looking up loader for language:
yaml
```

```
[1] 2020-04-11 14:26:43.655 INFO  [main] RuntimeSupport - Found loader
org.apache.camel.k.loader.yaml.YamlSourceLoader@1224144a for language yaml from
service definition
[1] 2020-04-11 14:26:43.658 INFO  [main] RoutesConfigurer - Loading routes from:
file:/etc/camel/sources/i-source-000/hello.camelk.yaml?language=yaml
[1] 2020-04-11 14:26:43.658 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesConfigurer@36c88a32 executed in phase
ConfigureRoutes
[1] 2020-04-11 14:26:43.661 INFO  [main] BaseMainSupport - Using properties from:
file:/etc/camel/conf/application.properties
[1] 2020-04-11 14:26:43.878 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.ContextConfigurer@65466a6a executed in phase
ConfigureContext
[1] 2020-04-11 14:26:43.879 INFO  [main] DefaultCamelContext - Apache Camel 3.0.1
(CamelContext: camel-k) is starting
[1] 2020-04-11 14:26:43.880 INFO  [main] DefaultManagementStrategy - JMX is disabled
[1] 2020-04-11 14:26:44.147 INFO  [main] DefaultCamelContext - StreamCaching is not in
use. If using streams then its recommended to enable stream caching. See more details at
http://camel.apache.org/stream-caching.html
[1] 2020-04-11 14:26:44.157 INFO  [main] DefaultCamelContext - Route: route1 started and
consuming from: timer://yaml?period=1s
[1] 2020-04-11 14:26:44.161 INFO  [main] DefaultCamelContext - Total 1 routes, of which 1
are started
[1] 2020-04-11 14:26:44.162 INFO  [main] DefaultCamelContext - Apache Camel 3.0.1
(CamelContext: camel-k) started in 0.283 seconds
[1] 2020-04-11 14:26:44.163 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesDumper@1c93084c executed in phase Started
[1] 2020-04-11 14:26:45.183 INFO  [Camel (camel-k) thread #1 - timer://yaml] info -
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

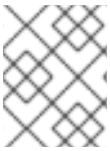
- **kamel run** コマンドの詳細については、**kamel run --help** を入力してください。
- デプロイメントのターンアラウンドタイムを短縮するには、「開発モードでの Camel K インテグレーションの実行」 を参照してください。
- インテグレーションを実行するための開発ツールの詳細は、Red Hat による Apache Camel K の VS Code ツール を参照してください。
- 「Camel K インテグレーションの管理」 も参照してください。

3.6. 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、XML、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
integration "hello-camel-k" created
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
Progress: integration "hello-camel-k" in phase Running
IntegrationPlatformAvailable for Integration hello-camel-k: camel-k
Integration hello-camel-k in phase Initialization
No IntegrationKitAvailable for Integration hello-camel-k: creating a new integration kit
Integration hello-camel-k in phase Building Kit
IntegrationKitAvailable for Integration hello-camel-k: kit-bq8t5cudeam3u3sj13tg
Integration hello-camel-k in phase Deploying
No CronJobAvailable for Integration hello-camel-k: different controller strategy used
(deployment)
DeploymentAvailable for Integration hello-camel-k: deployment name is hello-camel-k
No ServiceAvailable for Integration hello-camel-k: no http service required
No ExposureAvailable for Integration hello-camel-k: no target service found
Integration hello-camel-k in phase Running
[2] Monitoring pod hello-camel-k-866ccb5976-sjh8x[1] Monitoring pod hello-camel-k-
866ccb5976-l288p[2] 2020-04-11 14:44:53.691 INFO  [main] ApplicationRuntime - Add
listener: org.apache.camel.k.listener.ContextConfigurer@159f197
[2] 2020-04-11 14:44:53.694 INFO  [main] ApplicationRuntime - Add listener:
org.apache.camel.k.listener.RoutesConfigurer@1f0f1111
```

```
[2] 2020-04-11 14:44:53.695 INFO  [main] ApplicationRuntime - Add listener:
org.apache.camel.k.listener.RoutesDumper@6e0dec4a
[2] 2020-04-11 14:44:53.695 INFO  [main] ApplicationRuntime - Add listener:
org.apache.camel.k.listener.PropertiesFunctionsConfigurer@794cb805
[2] 2020-04-11 14:44:53.712 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.PropertiesFunctionsConfigurer@794cb805 executed in phase
Starting
[2] 2020-04-11 14:44:53.721 INFO  [main] RuntimeSupport - Looking up loader for language:
java
[2] 2020-04-11 14:44:53.723 INFO  [main] RuntimeSupport - Found loader
org.apache.camel.k.loader.java.JavaSourceLoader@3911c2a7 for language java from
service definition
[2] 2020-04-11 14:44:54.220 INFO  [main] RoutesConfigurer - Loading routes from:
file:/etc/camel/sources/i-source-000/HelloCamelK.java?language=java
[2] 2020-04-11 14:44:54.220 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesConfigurer@1f0f1111 executed in phase
ConfigureRoutes
[2] 2020-04-11 14:44:54.224 INFO  [main] BaseMainSupport - Using properties from:
file:/etc/camel/conf/application.properties
[2] 2020-04-11 14:44:54.385 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.ContextConfigurer@159f197 executed in phase
ConfigureContext
[2] 2020-04-11 14:44:54.386 INFO  [main] DefaultCamelContext - Apache Camel 3.0.1
(CamelContext: camel-k) is starting
[2] 2020-04-11 14:44:54.387 INFO  [main] DefaultManagementStrategy - JMX is disabled
[2] 2020-04-11 14:44:54.630 INFO  [main] DefaultCamelContext - StreamCaching is not in
use. If using streams then its recommended to enable stream caching. See more details at
http://camel.apache.org/stream-caching.html
[2] 2020-04-11 14:44:54.639 INFO  [main] DefaultCamelContext - Route: java started and
consuming from: timer://java?period=1s
[2] 2020-04-11 14:44:54.643 INFO  [main] DefaultCamelContext - Total 1 routes, of which 1
are started
[2] 2020-04-11 14:44:54.643 INFO  [main] DefaultCamelContext - Apache Camel 3.0.1
(CamelContext: camel-k) started in 0.258 seconds
[2] 2020-04-11 14:44:54.644 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesDumper@6e0dec4a executed in phase Started
[2] 2020-04-11 14:44:55.671 INFO  [Camel (camel-k) thread #1 - timer://java] info -
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。以下に例を示します。

```
...
integration "hello-camel-k" updated
...
[3] 2020-04-11 14:45:06.792 INFO  [main] DefaultCamelContext - Route: java started and
consuming from: timer://java?period=1s
[3] 2020-04-11 14:45:06.795 INFO  [main] DefaultCamelContext - Total 1 routes, of which 1
are started
[3] 2020-04-11 14:45:06.796 INFO  [main] DefaultCamelContext - Apache Camel 3.0.1
(CamelContext: camel-k) started in 0.323 seconds
[3] 2020-04-11 14:45:06.796 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesDumper@6e0dec4a executed in phase Started
```

```
[3] 2020-04-11 14:45:07.826 INFO  [Camel (camel-k) thread #1 - timer://java] info -
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Ciao 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」

3.7. 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、Prometheus のモニタリングおよび 3scale API Management のトレイトを設定し、外部 Maven ライブラリーの依存関係が含まれる 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。

```
// camel-k: language=java trait=prometheus.enabled=true trait=3scale.enabled=true
dependency=mvn:org.my/app:1.0
```

前提条件

- 「Camel K 開発環境の設定」
- Java、XML、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。

   Hello.java

```
// camel-k: language=java trait=prometheus.enabled=true trait=3scale.enabled=true
dependency=mvn:org.my/application:1.0 (1)

import org.apache.camel.builder.RouteBuilder;

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

      from("timer:java?period=1000")
```

```
        .bean(org.my.BusinessLogic) 2
        .log("${body}");

  }
}
```

1 modeline エントリーは、Prometheus でのモニタリングや 3scale での API 管理を有効にし、外部 Maven ライブラリーでの依存関係を指定します。

2 この Bean は、モードラインで設定された外部 Maven ライブラリーからのビジネスロジッククラスを使用します。

2. 以下のコマンドを入力してインテグレーションを実行します。

```
$ kamel run Hello.java
Modeline options have been loaded from source files
Full command: kamel run Hello.java --trait=prometheus.enabled=true --dependency
mvn:org.my/application:1.0
```

kamel run コマンドは、インテグレーションに指定されたモードオプションを出力します。

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 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」

4.1. 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

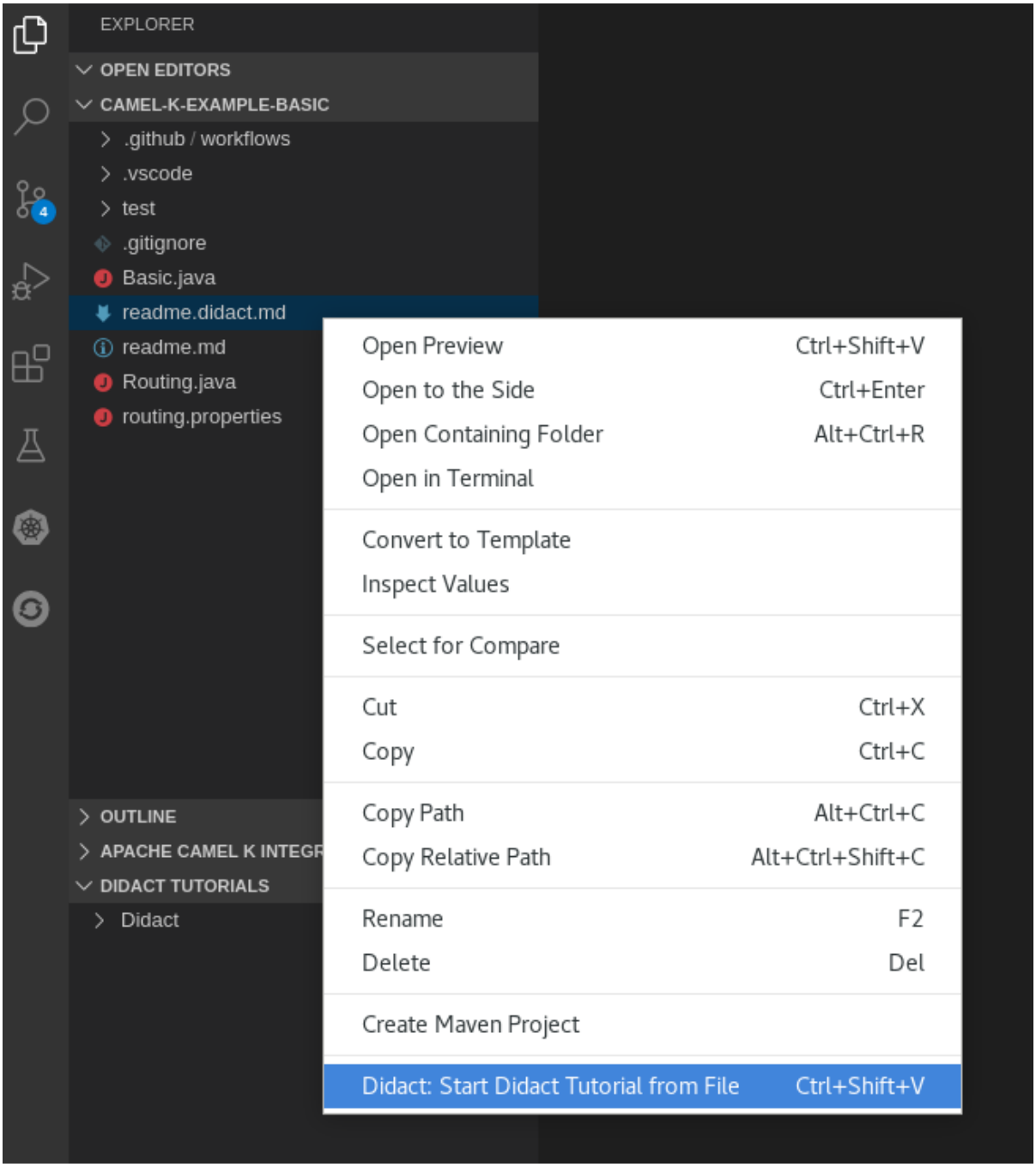
- GitHub https://github.com/openshift-integration/camel-k-example-basic のチュートリアル readme を確認します。
- Camel K をインストールするためにクラスター管理者として OpenShift クラスターへアクセスできる必要があります (「OpenShift OperatorHub からの Camel K のインストール」 を参照)。
- **kamel** コマンドがインストールされている必要があります。「Camel K および OpenShift コマンドラインツールのインストール」を参照してください。
- 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。「Camel K 開発環境の設定」 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。以下に例を示します。



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-basic からコマンドを手動で入力できます。

その他のリソース

- 「Java での Camel K インテグレーションの開発」

4.2. 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 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。
  - 「OpenShift OperatorHub からの Camel K のインストール」
  - 「OperatorHub からの OpenShift Serverless のインストール」
- **kamel** コマンドがインストールされている必要があります。「Camel K および OpenShift コマンドラインツールのインストール」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。「Camel K 開発環境の設定」 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。

その他のリソース

- Knative Eventing の仕組み
- Knative Serving の仕組み

4.3. 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

- GitHub https://github.com/openshift-integration/camel-k-example-transformations 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 (「OpenShift OperatorHub からの Camel K のインストール」 を参照)。
- **kamel** コマンドがインストールされている必要があります。「Camel K および OpenShift コマンドラインツールのインストール」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。「Camel K 開発環境の設定」 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/

4.4. 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 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。
  - 「OpenShift OperatorHub からの Camel K のインストール」
  - 「OperatorHub からの OpenShift Serverless のインストール」
- **kamel** コマンドがインストールされている必要があります。「Camel K および OpenShift コマンドラインツールのインストール」を参照してください。
- 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。「Camel K 開発環境の設定」 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

4.5. 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 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。
  - 「OpenShift OperatorHub からの Camel K のインストール」
  - 「OperatorHub からの OpenShift Serverless のインストール」
- **kamel** コマンドがインストールされている必要があります。「Camel K および OpenShift コマンドラインツールのインストール」を参照してください。
- 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。「operator を使用した 3scale のデプロイ 」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。「Camel K 開発環境の設定」 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

4.6. 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 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 (「OpenShift OperatorHub からの Camel K のインストール」 を参照)。
- **kamel** コマンドがインストールされている必要があります。「Camel K および OpenShift コマンドラインツールのインストール」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。「Camel K 開発環境の設定」 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/

第5章 CAMEL K インテグレーションの管理

Camel K コマンドラインまたは開発ツールを使用して、Red Hat Integration - Camel K インテグレーションを管理できます。本章では、コマンドラインで Camel K インテグレーションを管理する方法を説明し、VS Code 開発ツールの使用方法を説明する追加のリソースへのリンクを提供します。

- 「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のロギングレベルの管理」

5.1. CAMEL K インテグレーションの管理

コマンドラインで OpenShift クラスターの Camel K インテグレーションを管理するためのさまざまなオプションがあります。ここでは、以下のコマンドを使用する簡単な例を紹介します。

- **kamel get**
- **kamel describe**
- **kamel ログ**
- **kamel delete**

前提条件

- 「Camel K 開発環境の設定」
- Java、XML、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、Camel K Operator が OpenShift クラスターで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
```

2. **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。以下に例を示します。

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

3. **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 kit-bqatqib5t4kse5vukt40
```

4. **kamel describe** コマンドを入力し、インテグレーションに関する詳細情報を出力します。

```
$ kamel describe integration hello
```

```
kamel describe integration hello
Name:               hello
Namespace:          camel-k-test
Creation Timestamp:  Tue, 14 Apr 2020 16:57:04 +0100
Phase:              Running
Runtime Version:    1.1.0
Kit:                kit-bqatqib5t4kse5vukt40
Image:              image-registry.openshift-image-registry.svc:5000/camel-k-test/camel-k-kit-
bqatqib5t4kse5vukt40@sha256:3788d571e6534ab27620b6826e6a4f10c23fc871d2f8f60673
b7c20e617d6463
Version:            1.0.0-RC2
Dependencies:
  camel:log
  camel:timer
  mvn:org.apache.camel.k/camel-k-loader-yaml
  mvn:org.apache.camel.k/camel-k-runtime-main
Sources:
  Name              Language  Compression  Ref  Ref Key
  hello.camelk.yaml  yaml      false
Conditions:
  Type                       Status  Reason                      Message
  IntegrationPlatformAvailable  True    IntegrationPlatformAvailable  camel-k
  IntegrationKitAvailable       True    IntegrationKitAvailable       kit-bqatqib5t4kse5vukt40
  CronJobAvailable              False   CronJobNotAvailableReason     different controller
strategy used (deployment)
  DeploymentAvailable           True    DeploymentAvailable           deployment name is hello
  ServiceAvailable              False   ServiceNotAvailable           no http service required
  ExposureAvailable             False   RouteNotAvailable             no target service found
```

5. **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] 2020-04-14 16:03:41.205 INFO  [Camel (camel-k) thread #1 - timer://yaml] info -
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from yaml]
[1] 2020-04-14 16:03:42.205 INFO  [Camel (camel-k) thread #1 - timer://yaml] info -
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from yaml]
[1] 2020-04-14 16:03:43.204 INFO  [Camel (camel-k) thread #1 - timer://yaml] info -
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from yaml]
...
```

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

7. **kamel delete** を入力して、OpenShift にデプロイされたインテグレーションを削除します。

```
$ kamel delete hello
Integration hello deleted
```

関連情報

- ロギングの詳細は、「Camel K インテグレーションのロギングレベルの管理」 を参照してください。
- デプロイメントのターンアラウンドタイムを短縮するには、「開発モードでの Camel K インテグレーションの実行」 を参照してください。

- インテグレーションを管理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。

5.2. CAMEL K インテグレーションのロギングレベルの管理

Camel K は Apache Log4j 2 をインテグレーションのロギングフレームワークとして使用します。**logging.level** プレフィックスをインテグレーションプロパティーとして指定すると、実行時にコマンドラインでさまざまなロガーのロギングレベルを設定できます。以下に例を示します。

```
--property logging.level.org.apache.camel=DEBUG
```

前提条件

- 「Camel K 開発環境の設定」

手順

1. **kamel run** コマンドを入力し、**--property** オプションを使用してログレベルを指定します。以下に例を示します。

```
$ kamel run --property logging.level.org.apache.camel=DEBUG HelloCamelK.java --dev
...
[1] 2020-04-13 17:02:17.970 DEBUG [main] PropertiesComponentFactoryResolver - Detected and using PropertiesComponent: org.apache.camel.component.properties.PropertiesComponent@3e92efc3
[1] 2020-04-13 17:02:17.974 INFO  [main] ApplicationRuntime - Listener org.apache.camel.k.listener.PropertiesFunctionsConfigurer@4b5a5ed1 executed in phase Starting
[1] 2020-04-13 17:02:17.984 INFO  [main] RuntimeSupport - Looking up loader for language: java
[1] 2020-04-13 17:02:17.987 INFO  [main] RuntimeSupport - Found loader org.apache.camel.k.loader.java.JavaSourceLoader@4facf68f for language java from service definition
[1] 2020-04-13 17:02:18.553 INFO  [main] RoutesConfigurer - Loading routes from: file:/etc/camel/sources/i-source-000/HelloCamelK.java?language=java
[1] 2020-04-13 17:02:18.553 INFO  [main] ApplicationRuntime - Listener org.apache.camel.k.listener.RoutesConfigurer@49c386c8 executed in phase ConfigureRoutes
[1] 2020-04-13 17:02:18.555 DEBUG [main] PropertiesComponent - Parsed location: /etc/camel/conf/application.properties
[1] 2020-04-13 17:02:18.557 INFO  [main] BaseMainSupport - Using properties from: file:/etc/camel/conf/application.properties
[1] 2020-04-13 17:02:18.563 DEBUG [main] BaseMainSupport - Properties from Camel properties component:
[1] 2020-04-13 17:02:18.598 DEBUG [main] RoutesConfigurer - RoutesCollectorEnabled: org.apache.camel.k.main.ApplicationRuntime$NoRoutesCollector@2f953efd
[1] 2020-04-13 17:02:18.598 DEBUG [main] RoutesConfigurer - Adding routes into CamelContext from RoutesBuilder: Routes: []
[1] 2020-04-13 17:02:18.598 DEBUG [main] DefaultCamelContext - Adding routes from builder: Routes: []
...
```

2. **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

- ロギングフレームワークの詳細は Apache Log4j 2 のドキュメント を参照してください。
- ログを表示する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。

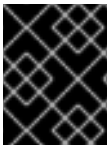
第6章 CAMEL K インテグレーションのモニタリング

Red Hat Integration - Camel K モニタリングは、Prometheus モニタリングシステム https://prometheus.io/ をベースとしています。本章では、実行時に Red Hat Integration - Camel K インテグレーションを監視するためにオプションを使用する方法について説明します。OpenShift Monitoring の一部としてすでにデプロイされている Prometheus Operator を使用して、独自のアプリケーションを監視することができます。

- 「OpenShift でのユーザーワークロードモニタリングの有効化」
- 「Camel K インテグレーションメトリクスの設定」
- 「カスタム Camel K インテグレーションメトリクスの追加」

6.1. OPENSHIFT でのユーザーワークロードモニタリングの有効化

OpenShift 4.3 以降には、OpenShift Monitoring の一部としてすでにデプロイされている組み込みの Prometheus Operator が含まれています。ここでは、OpenShift Monitoring で独自のアプリケーションサービスのモニタリングを有効にする方法について説明します。このオプションを使用すると、個別の Prometheus インスタンスをインストールおよび管理するための追加のオーバーヘッドが発生しません。



重要

個別の Prometheus Operator を使用した Camel K インテグレーションのモニタリングは、テクノロジープレビューには含まれていません。

前提条件

- Camel K Operator がインストールされている OpenShift クラスターにクラスター管理者としてアクセスできる必要があります。「OpenShift OperatorHub からの Camel K のインストール」を参照してください。

手順

1. 以下のコマンドを実行して、**cluster-monitoring-config** ConfigMap オブジェクトが **openshift-monitoring project** に存在するかどうかを確認します。

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config
```

2. **cluster-monitoring-config** ConfigMap がない場合は作成します。

```
$ oc -n openshift-monitoring create configmap cluster-monitoring-config
```

3. **cluster-monitoring-config** ConfigMap を編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

4. **data:config.yaml:** で、**enableUserWorkload** を **true** に設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
```

関連情報

- OpenShift ドキュメントの「独自のサービスのモニタリング」

6.2. CAMEL K インテグレーションメトリクスの設定

実行時に Camel K Prometheus トレイトを使用すると、Camel K インテグレーションのモニタリングを自動的に設定できます。これにより、依存関係およびインテグレーション Pod の設定が自動化され、Prometheus によって検出および表示されるメトリクスエンドポイントが公開されます。Camel Quarkus MicroProfile Metrics エクステンション は、OpenMetrics 形式でデフォルトの Camel K メトリクスを自動的に収集および公開します。

前提条件

- OpenShift で、独自のサービスのモニタリングが有効になっている必要があります。「OpenShift でのユーザーワークロードモニタリングの有効化」 を参照してください。

手順

1. 以下のコマンドを入力して、Prometheus トレイトを有効にして Camel K インテグレーションを実行します。

```
$ kamel run myIntegration.java -t prometheus.enabled=true
```

   または、以下のようにインテグレーションプラットフォームを更新すると、Prometheus トレイトを 1 度グローバルに有効にすることができます。

```
$ oc patch ip camel-k --type=merge -p '{"spec":{"traits":{"prometheus":{"configuration":{"enabled":"true"}}}}}'
```

2. Prometheus で Camel K インテグレーションメトリクスのモニタリングを確認します。たとえば、埋め込み Prometheus の場合は、OpenShift 管理者または開発者 Web コンソールで **Monitoring** > **Metrics** と選択します。

3. 表示する Camel K メトリクスを入力します。たとえば、**Administrator** コンソールの **Insert Metric at Cursor** で **application_camel_context_uptime_seconds** を入力し、**Run Queries** をクリックします。

4. **Add Query** をクリックして追加のメトリクスを表示します。

その他のリソース

- 「Prometheus トレイト」
- Camel Quarkus MicroProfile Metrics

6.3. カスタム CAMEL K インテグレーションメトリクスの追加

Java コードで Camel MicroProfile Metrics コンポーネントおよびアノテーションを使用すると、カスタムメトリクスを Camel K インテグレーションに追加できます。その後、これらのカスタムメトリクスは Prometheus によって自動的に検出され、表示されます。

ここでは、Camel MicroProfile Metrics アノテーションを Camel K インテグレーションおよびサービス実装コードに追加する例を紹介します。

前提条件

- OpenShift で、独自のサービスのモニタリングが有効になっている必要があります。「OpenShift でのユーザーワークロードモニタリングの有効化」 を参照してください。

手順

1. Camel MicroProfile Metrics コンポーネントアノテーションを使用して、カスタムメトリクスを Camel インテグレーションコードに登録します。以下の例は、**Metrics.java** インテグレーションを示しています。

```
// camel-k: language=java trait=prometheus.enabled=true dependency=mvn:org.my/app:1.0
1

import org.apache.camel.Exchange;
import org.apache.camel.LoggingLevel;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.microprofile.metrics.MicroProfileMetricsConstants;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class Metrics extends RouteBuilder {

  @Override
  public void configure() {
      onException()
          .handled(true)
          .maximumRedeliveries(2)
          .logStackTrace(false)
          .logExhausted(false)
          .log(LoggingLevel.ERROR, "Failed processing ${body}")
          // Register the 'redelivery' meter
          .to("microprofile-metrics:meter:redelivery?mark=2")
          // Register the 'error' meter
          .to("microprofile-metrics:meter:error"); 2

      from("timer:stream?period=1000")
          .routeId("unreliable-service")
          .setBody(header(Exchange.TIMER_COUNTER).prepend("event #"))
          .log("Processing ${body}...")
          // Register the 'generated' meter
          .to("microprofile-metrics:meter:generated") 3
          // Register the 'attempt' meter via @Metered in Service.java
          .bean("service") 4
          .filter(header(Exchange.REDELIVERED))
              .log(LoggingLevel.WARN, "Processed ${body} after
${header.CamelRedeliveryCounter} retries")
```

```
            .setHeader(MicroProfileMetricsConstants.HEADER_METER_MARK,
header(Exchange.REDELIVERY_COUNTER))
            // Register the 'redelivery' meter
            .to("microprofile-metrics:meter:redelivery") 5
        .end()
        .log("Successfully processed ${body}")
        // Register the 'success' meter
        .to("microprofile-metrics:meter:success"); 6
    }
}
```

1. Camel K モードラインを使用して、Prometheus トレイトと Maven 依存関係を自動的に設定します。
2. **error**: 処理されていないイベントの数に対応するエラー数のメトリック。
3. **generated**: 処理されるイベント数のメトリック。
4. **attempt**: 受信イベントを処理するためにサービス Bean に対して行われる呼び出し数のメトリック。
5. **redelivery**: イベントを処理するために行われた再試行回数のメトリック。
6. **success**: 正常に処理されたイベント数のメトリクス。

2. 必要に応じて Camel MicroProfile Metrics アノテーションを実装ファイルに追加します。以下の例は、ランダムに失敗を生成する Camel K インテグレーションによって呼び出される **service** Bean を示しています。

```
package com.redhat.integration;

import java.util.Random;

import org.apache.camel.Exchange;
import org.apache.camel.RuntimeExchangeException;

import org.eclipse.microprofile.metrics.Meter;
import org.eclipse.microprofile.metrics.annotation.Metered;
import org.eclipse.microprofile.metrics.annotation.Metric;

import javax.inject.Named;
import javax.enterprise.context.ApplicationScoped;

@Named("service")
@ApplicationScoped
@io.quarkus.arc.Unremovable

public class Service {

  //Register the attempt meter
  @Metered(absolute = true)
  public void attempt(Exchange exchange) { 1
    Random rand = new Random();
      if (rand.nextDouble() < 0.5) {
        throw new RuntimeExchangeException("Random failure", exchange); 2
```

```
    }
  }
}
```

1. **@Metered** MicroProfile Metrics アノテーションは meter を宣言し、名前はメトリクスメソッド名 (この場合は **attempt**) に基づいて自動的に生成されます。
2. この例は、メトリクスのエラーを生成するために無作為に失敗します。

3. 「Camel K インテグレーションメトリクスの設定」 の手順に従い、インテグレーションを実行し、Prometheus でカスタム Camel K メトリクスを確認します。
この例では、**Metrics.java** ですでに Camel K モードラインが使用され、Prometheus と **Service.java** に必要な Maven 依存関係が自動的に設定されます。

関連情報

- Camel MicroProfile Metrics component
- Camel Quarkus MicroProfile Metrics Extension

第7章 CAMEL K インテグレーションの設定

本章では、以下のプロパティーを使用して Red Hat Integration - Camel K インテグレーションを設定するために使用できるオプションについて説明します。

- 「プロパティーを使用した Camel K インテグレーションの設定」
- 「プロパティーファイルを使用した Camel K インテグレーションの設定」
- 「OpenShift ConfigMap を使用した Camel K プロパティーの設定」
- 「OpenShift シークレットを使用した Camel K プロパティーの設定」
- 「Camel インテグレーションコンポーネントの設定」
- 「Camel K インテグレーション依存関係の設定」

7.1. プロパティーを使用した CAMEL K インテグレーションの設定

実行時にコマンドラインで Camel K インテグレーションのプロパティーを設定できます。プロパティープレースホルダー（例：**{{my.message}}）を使用してインテグレーションでプロパティーを定義する場合、--property my.message =Hello** のようにコマンドラインでプロパティー値を指定できます。1 つのコマンドで複数のプロパティーを指定できます。

前提条件

- 「Camel K 開発環境の設定」

手順

1. プロパティーを使用する Camel インテグレーションを開発します。以下の簡単なルートには **{{my.message** }} プロパティープレースホルダーが含まれています。

```
...
  from("timer:java?period=1s")
    .routeId("java")
    .setBody()
      .simple("{{my.message}} from ${routeId}")
    .to("log:info");
...
```

2. **--property** オプションを使用して **kamel run** コマンドを入力し、実行時にプロパティー値を設定します。以下に例を示します。

```
$ kamel run --property my.message="Hola Mundo" HelloCamelK.java --dev
...
[1] 2020-04-13 15:39:59.213 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesDumper@6e0dec4a executed in phase Started
[1] 2020-04-13 15:40:00.237 INFO  [Camel (camel-k) thread #1 - timer://java] info -
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hola Mundo from java]
...
```

関連情報

- 「プロパティーファイルを使用した Camel K インテグレーションの設定」
- 「OpenShift ConfigMap を使用した Camel K プロパティーの設定」
- 「OpenShift シークレットを使用した Camel K プロパティーの設定」

7.2. プロパティーファイルを使用した CAMEL K インテグレーションの設定

実行時にコマンドラインでプロパティーファイルを指定すると、Camel K インテグレーションに複数のプロパティーを設定できます。プロパティープレースホルダー（例： **{{my.items** }}）を使用してインテグレーションでプロパティーを定義する場合、**--property-file my-integration.properties** のようにプロパティーファイルを使用してコマンドラインでプロパティー値を指定できます。

前提条件

- 「Camel K 開発環境の設定」

手順

1. インテグレーションプロパティーファイルを定義します。以下は、**Routing.properties ファイルからの簡単な例になります。**

```
# List of items for random generation
items=*radiator *engine *door window

# Marker to identify priority items
priority-marker=*
```

2. プロパティーファイルに定義されたプロパティーを使用する Camel インテグレーションを開発します。**Routing.java** 統合からの以下の例は **{{items}} および {{ priority-marker** }} プロパティープレースホルダーを使用します。

```
...
from("timer:java?period=6000")
    .id("generator")
    .bean(this, "generateRandomItem({{items}})")
    .choice()
      .when().simple("${body.startsWith('{{priority-marker}}')}")
        .transform().body(String.class, item -> item.substring(priorityMarker.length()))
        .to("direct:priorityQueue")
      .otherwise()
        .to("direct:standardQueue");
...
```

3. **--property-file** オプションを指定して **kamel run** コマンドを入力します。以下に例を示します。

```
$ kamel run Routing.java --property-file routing.properties --dev
...
[1] 2020-04-13 15:20:30.424 INFO  [main] ApplicationRuntime - Listener
org.apache.camel.k.listener.RoutesDumper@6e0dec4a executed in phase Started
[1] 2020-04-13 15:20:31.461 INFO  [Camel (camel-k) thread #1 - timer://java] priority -
!!Priority item: engine
[1] 2020-04-13 15:20:37.426 INFO  [Camel (camel-k) thread #1 - timer://java] standard -
```

```
Standard item: window
[1] 2020-04-13 15:20:43.429 INFO  [Camel (camel-k) thread #1 - timer://java] priority -
!!Priority item: door
...
```

関連情報

- 「基本的な Camel K Java インテグレーションのデプロイ」
- 「プロパティーを使用した Camel K インテグレーションの設定」

7.3. OPENSHIFT CONFIGMAP を使用した CAMEL K プロパティーの設定

OpenShift ConfigMap を使用すると、Camel K インテグレーションに複数のプロパティーを設定できます。プロパティープレースホルダー（例： **{{my.message** }}）を使用してインテグレーションでプロパティーを定義する場合、ConfigMap を使用して実行時にプロパティー値を指定できます。ConfigMap の **application.properties** セクションにログレベルなどの追加のプロパティーを指定することもできます。

前提条件

- 「Camel K 開発環境の設定」

手順

1. プロパティーを使用する Camel インテグレーションを開発します。以下の簡単なルートには **{{my.message** }} プロパティープレースホルダーが含まれます。

```
...
  from("timer:java?period=1s")
    .routeId("java")
    .setBody()
      .simple("{{my.message}} from ${routeId}")
    .to("log:info");
...
```

2. 設定プロパティーが含まれる ConfigMap を定義します。以下に例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap
data:
  application.properties: |
    my.message=Bonjour le monde
    logging.level.org.apache.camel=DEBUG
```

この例では、**my.message** プロパティーの値を設定し、**application.properties** に **org.apache.camel** パッケージのログレベルを設定します。

3. インテグレーションと同じ OpenShift namespace に ConfigMap を作成します。

```
$ oc apply -f my-configmap.yaml
configmap/my-configmap created
```

4. **--configmap** オプションを指定してインテグレーションを実行し、ConfigMap に設定プロパティーを指定します。

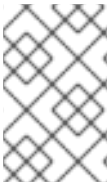
```
$ kamel run --configmap=my-configmap HelloCamelK.java --dev
...
[1] 2020-04-14 14:18:20.654 DEBUG [Camel (camel-k) thread #1 - timer://java] DefaultReactiveExecutor - Queuing reactive work: CamelInternalProcessor - UnitOfWork - afterProcess - DefaultErrorHandler[sendTo(log://info)] - ID-hello-camel-k-5df4bcd7dc-zq4vw-1586873876659-0-25
[1] 2020-04-14 14:18:20.654 DEBUG [Camel (camel-k) thread #1 - timer://java] SendProcessor - >>>> log://info Exchange[ID-hello-camel-k-5df4bcd7dc-zq4vw-1586873876659-0-25]
[1] 2020-04-14 14:18:20.655 INFO  [Camel (camel-k) thread #1 - timer://java] info - Exchange[ExchangePattern: InOnly, BodyType: String, Body: Bonjour le monde from java]
...
```

関連情報

- 「OpenShift シークレットを使用した Camel K プロパティーの設定」

7.4. OPENSHIFT シークレットを使用した CAMEL K プロパティーの設定

OpenShift シークレットを使用すると、Camel K インテグレーションに複数のプロパティーを設定できます。プロパティープレースホルダー（例： **{{my.message** }}）を使用してインテグレーションでプロパティーを定義する場合、シークレットを使用して実行時にプロパティー値を指定できます。Secret の **application.properties** セクションにログレベルなどの追加のプロパティーを指定することもできます。



注記

シークレットを使用したインテグレーションプロパティーの設定は、ConfigMap を使用した設定と似ています。主な違いは、Secret の **application.properties** の内容を base64 でエンコーディングする必要がある場合がある点です。

前提条件

- 「Camel K 開発環境の設定」

手順

1. プロパティーを使用する Camel インテグレーションを開発します。以下の簡単なルートには **{{my.message** }} プロパティープレースホルダーが含まれます。

```
...
  from("timer:java?period=1s")
    .routeId("java")
    .setBody()
      .simple("{{my.message}} from ${routeId}")
    .to("log:info");
...
```

2. 設定プロパティーが含まれるシークレットを定義します。以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  application.properties: |

bXkubWVzc2FnZT1IZWxsbyBXb3JsZAogICAgbG9nZ2luZy5sZXZlbC5vcmcuYXBhY2hlLmNhbWVs
    PURFQlVHCg==
```

この例では、**my.message** プロパティーの値を **Hello World** に設定し、**org.apache.camel** パッケージのログレベルを **DEBUG** に設定します。これらの設定は、base64 でエンコードされた形式で **application.properties** に指定されます。

3. インテグレーションと同じ OpenShift namespace にシークレットを作成します。

```
$ oc apply -f my-secret.yaml
secret/my-secret created
```

4. **--secret** オプションを指定してインテグレーションを実行し、シークレットの設定プロパティーを指定します。

```
$ kamel run --secret=my-secret HelloCamelK.java --dev
[1] 2020-04-14 14:30:29.788 DEBUG [Camel (camel-k) thread #1 - timer://java]
DefaultReactiveExecutor - Queuing reactive work: CamelInternalProcessor - UnitOfWork -
afterProcess - DefaultErrorHandler[sendTo(log://info)] - ID-hello-camel-k-68f85d99b9-srd92-
1586874486770-0-144
[1] 2020-04-14 14:30:29.789 DEBUG [Camel (camel-k) thread #1 - timer://java]
SendProcessor - >>>> log://info Exchange[ID-hello-camel-k-68f85d99b9-srd92-
1586874486770-0-144]
[1] 2020-04-14 14:30:29.789 INFO  [Camel (camel-k) thread #1 - timer://java] info -
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello World from java]
```

関連情報

- 「OpenShift ConfigMap を使用した Camel K プロパティーの設定」

7.5. CAMEL インテグレーションコンポーネントの設定

Camel コンポーネントは、インテグレーションコードにプログラミングを使用して設定でき、実行時にコマンドラインで設定プロパティーを使用して設定することもできます。以下の構文を使用して Camel コンポーネントを設定できます。

```
camel.component.${scheme}.${property}=${value}
```

たとえば、段階的なイベント駆動型アーキテクチャーの Camel **seda** コンポーネントのキューサイズを変更するには、コマンドラインで以下のプロパティーを設定します。

```
camel.component.seda.queueSize=10
```

前提条件

- 「Camel K 開発環境の設定」

手順

- **kamel run** コマンドを入力し、**--property** オプションを使用して Camel コンポーネント設定を指定します。以下に例を示します。

```
$ kamel run --property camel.component.seda.queueSize=10 examples/Integration.java
```

関連情報

- 「プロパティーを使用した Camel K インテグレーションの設定」
- Apache Camel SEDA component

7.6. CAMEL K インテグレーション依存関係の設定

Camel K は、インテグレーションコードの実行に必要なさまざまな依存関係を自動的に解決します。ただし、**kamel run --dependency** オプションを使用すると、実行時にコマンドラインに依存関係を明示的に追加できます。

以下のインテグレーションの例では Camel K の依存関係の自動解決が使用されます。

```
...
  from("imap://admin@myserver.com")
    .to("seda:output")
...
```

このインテグレーションには **imap:** プレフィックスで始まるエンドポイントがあるため、Camel K は自動的に **camel-mail** コンポーネントを必要な依存関係のリストに追加できます。**seda:** エンドポイントは、すべてのインテグレーションに自動的に追加される **camel-core** に属しているため、Camel K はこのコンポーネントにその他の依存関係を追加しません。

Camel K 依存関係の自動解決は、実行時にユーザーに対して透過的です。これは、開発ループを終了せずに必要なすべてのコンポーネントを素早く追加できるため、開発モードで非常に便利です。

kamel run --dependency または **-d** オプションを使用して、依存関係を明示的に追加できます。これを使用して、Camel カタログに含まれていない依存関係を指定する必要がある場合があります。コマンドラインで複数の依存関係を指定できます。

前提条件

- 「Camel K 開発環境の設定」

手順

- **kamel run** コマンドを入力し、**-d** オプションを使用して依存関係を指定します。以下に例を示します。

```
$ kamel run -d mvn:com.google.guava:guava:26.0-jre -d camel-mina2 Integration.java
```

注記

-trait dependencies.enabled=false のように、依存関係トレイトを無効にすると、依存関係の自動解決を無効することができます。ただし、これはほとんどの場合で推奨されません。

その他のリソース

- 「開発モードでの Camel K インテグレーションの実行」
- 「Camel K トレイトおよびプロファイルの設定」
- Apache Camel Mail component
- Apache Camel SEDA component

第8章 CAMEL K トレイト設定の参考情報

本章では、**トレイト**を使用して実行時にコマンドラインで設定できる高度な機能とコア機能に関する参考情報を紹介します。Camel K は、特定の機能および技術を設定する **機能トレイト** (feature trait) を提供します。Camel K は、内部の Camel K コア機能を設定する **プラットフォームトレイト** を提供します。

重要

テクノロジープレビューの Red Hat Integration - Camel K には、**OpenShift** および **Knative** プロファイルが含まれます。**Kubernetes** プロファイルのサポートはコミュニティーのみに限定されます。

このテクノロジープレビューには、インテグレーションの Java、XML、および YAML DSL が含まれています。Groovy、JavaScript、Kourtlin などのその他の言語のサポートはコミュニティーのみに限定されます。

本章には、以下が含まれます。

- 「Camel K トレイトおよびプロファイルの設定」

Camel K 機能トレイト

- 「3scale トレイト」
- 「アフィニティートレイト」
- 「cron トレイト」
- 「GC トレイト」
- 「Istio トレイト」
- 「Jolokia トレイト」
- 「Knative トレイト」
- 「Knative Service トレイト」
- 「Master トレイト」
- 「Prometheus トレイト」
- 「Quarkus トレイト」
- 「Route トレイト」
- 「Service トレイト」

Camel K コアプラットフォームトレイト

- 「Builder トレイト」
- 「Camel トレイト」
- 「Container トレイト」

- 「Dependencies トレイト」
- 「Deployer トレイト」
- 「Deployment トレイト」
- 「Environment トレイト」
- 「JVM トレイト」
- 「Openapi トレイト」
- 「Owner トレイト」
- 「プラットフォームトレイト」

8.1. CAMEL K トレイトおよびプロファイルの設定

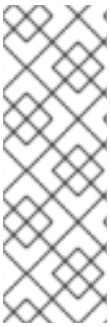
ここでは、実行時に 高度な Camel K 機能を設定するために使用される **トレイト** および **プロファイル** の重要な Camel K の概念について説明します。

Camel K トレイト

Camel K トレイトは高度な機能およびコア機能で、Camel K インテグレーションをカスタマイズするためにコマンドラインで設定できます。たとえば、これには、3scale API Management、Quarkus、Knative、Prometheus などのテクノロジーとの対話を設定する **機能トレイト** (feature trait) が含まれます。Camel K は、Camel サポート、コンテナー、依存関係の解決、JVM サポートなどの重要なコアプラットフォーム機能を設定する、内部の**プラットフォームトレイト** も提供します。

Camel K プロファイル

Camel K プロファイルは、Camel K インテグレーションが実行されるターゲットクラウドプラットフォームを定義します。テクノロジープレビューの Camel K は **OpenShift** および **Knative** プロファイルをサポートします。



注記

OpenShift でインテグレーションを実行するときに、OpenShift Serverless がクラスターにインストールされている場合、Camel K は **Knative** プロファイルを使用します。OpenShift Serverless がインストールされていない場合、Camel K は **OpenShift** プロファイルを使用します。

kamel run --profile オプションを使用して、実行時にプロファイルを指定することもできます。

Camel K は、インテグレーションが実行されるターゲットプロファイルを考慮し、便利なデフォルトをすべてのトレイトに提供します。ただし、上級ユーザーはカスタム動作の Camel K トレイトを設定できます。一部のトレイトは、**OpenShift** や **Knative** などの特定のプロファイルにのみ適用されます。詳細は、各トレイトの説明にある利用可能なプロファイルを参照してください。

Camel K トレイトの設定

各 Camel トレイトには、コマンドラインでトレイトを設定するために使用する一意の ID があります。たとえば、以下のコマンドは、インテグレーションの OpenShift Service の作成を無効にします。

```
$ kamel run --trait service.enabled=false my-integration.yaml
```

-t オプションを使用してトレイトを指定することもできます。

Camel K トレイトプロパティー

enabled プロパティーを使用して、各トレイトを有効または無効にすることができます。すべてのトレイトには、ユーザーが明示的にアクティブ化しない場合に有効にする必要があるかどうかを判断する独自の内部ロジックがあります。

警告

プラットフォームトレイトを無効にすると、プラットフォームの機能性が低下する可能性があります。

一部のトレイトには、環境に応じてトレイトの自動設定を有効または無効にするために使用する **auto** プロパティーがあります。たとえば、3scale、Cron、Knative などのトレイトが含まれます。この自動設定では、**enabled** プロパティーが明示的に設定されていない場合にトレイトを有効または無効にすることができ、トレイトの設定を変更することができます。

ほとんどのトレイトには、コマンドラインで設定できる追加のプロパティーがあります。詳細は、これ以降のセクションで各トレイトの説明を参照してください。

8.2. CAMEL K 機能トレイト

8.2.1. 3scale トレイト

3scale トレイトを使用すると、3scale が生成されたサービスを検出し、API 管理で使用できるようにするアノテーションを自動作成できます。

3scale トレイトはデフォルトで無効になっています。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.1.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait 3scale.[key]=[value] --trait 3scale.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
3scale.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
3scale.auto	bool	トレイトの自動設定を有効にします。

プロパティー	タイプ	説明
3scale.scheme	string	サービスとの通信に使用するスキーム（デフォルトは **http**）。
3scale.path	string	API がパブリッシュされるパス（デフォルト /）
3scale.port	int	サービスが公開されるポート（デフォルトは **80**）
3scale.description-path	string	Open-API 仕様が公開されるパス（デフォルトは **/openapi.json**）

8.2.2. アフィニティートレイト

ノード上のラベルを基にして、Pod 間のアフィニティーと非アフィニティーで、またはすでにノード上で実行中の Pod のラベルを基にして、インテグレーション Pod がスケジュールできるノードの制約を可能にします。

デフォルトでは無効になっています。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.2.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait affinity.[key]=[value] --trait affinity.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
affinity.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
affinity.pod-affinity	bool	常にインテグレーションの複数のレプリカを同じノードに配置します (デフォルトは **false**)。
affinity.pod-anti-affinity	bool	常にインテグレーションの複数のレプリカを同じノードに配置しません (デフォルトは **false**)。
affinity.node-affinity-labels	[]string	ノードのラベルに基づいて、インテグレーション Pod がスケジュール可能なノードのセットを定義します。
affinity.pod-affinity-labels	[]string	インテグレーション Pod と共に同じ場所に配置する Pod のセット (指定の namespace に対してラベルセレクターが一致するもの) を定義します。

プロパティー	タイプ	説明
affinity.pod-anti-affinity-labels	[]string	インテグレーション Pod と共に同じ場所に配置しない Pod のセット (指定の namespace に対してラベルセレクターが一致するもの) を定義します。

8.2.2.2. 例

- 組み込みノードラベル **kubernetes.io/hostname** を使用して特定のノードで インテグレーション Pod をスケジュールするには、以下を実行します。

```
$ kamel run -t affinity.node-affinity-labels="kubernetes.io/hostname in(node-66-50.hosted.k8s.tld)" ...
```

- （ **Exists** Operator を使用して）ノードごとに単一のインテグレーション Pod をスケジュールするには、以下を実行します。

```
$ kamel run -t affinity.pod-anti-affinity-labels="camel.apache.org/integration" ...
```

- インテグレーション Pod を他のインテグレーション Pod と同じ場所に配置するには、以下を実行します。

```
$ kamel run -t affinity.pod-affinity-labels="camel.apache.org/integration in(it1, it2)" ...
```

***-labels** オプションは、ラベル セレクター からの要件に従います。これらには複数の値を指定でき、要件リストは AND (指定のすべての条件を適用) として解釈されます。たとえば、ノードごとに単一のインテグレーション Pod をスケジュールし、かつ Camel K operator Pod と同じ場所に配置しない場合は、以下を実行します。

```
$ kamel run -t affinity.pod-anti-affinity-labels="camel.apache.org/integration" -t affinity.pod-anti-affinity-labels="camel.apache.org/component=operator" ...
```

詳細は、Kubernetes 公式ドキュメントの「Assigning Pods to Nodes」を参照してください。

8.2.3. cron トレイト

Cron トレイトを使用すると、定期的なタイマー/cron ベースのインテグレーションの動作をカスタマイズできます。

通常、インテグレーションには Pod が常に稼働している必要がありますが、バッチジョブなどの一部の定期的なタスクでは、特定の時間や数分の定期的な遅延でアクティベートする必要があります。このようなタスクで cron トレイトを使用すると、インテグレーションの実行が必要がないときにリソースを節約するため、標準デプロイメントではなく Kubernetes CronJob としてインテグレーションを実現できます。

以下のコンポーネントから開始するインテグレーションは、cron トレイト（**タイマー**、**cron**、**quartz**）によって評価されます。

Kubernetes CronJob を使用するルールは次のとおりです。**タイマー**: ピリオドを cron 式として記述できる場合。例:**timer:tick?period=60000**. - **cron**,**quartz**: cron 式に秒数が含まれていない場合（または "seconds" の部分は 0）。例:**cron:tab?schedule=0/2$+*+*+*+?** or **quartz:trigger?cron=0+0/2+*+*+*+?**.

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.3.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait cron.[key]=[value] --trait cron.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
cron.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
cron.schedule	string	インテグレーション全体の CronJob スケジュール。複数のルートが宣言されている場合、このメカニズムが正常に機能するためには同じスケジュールが必要になります。
cron.components	string	スケジュールが Kubernetes によって外部でトリガーされたときに機能するためにカスタマイズする必要がある Camel コンポーネントのコンマ区切りリスト。指定されたコンポーネントごとに、特定のカスタマイザーがアクティベートされます。たとえば、**タイマー** コンポーネントの場合、**cron-timer** カスタマイズャーがアクティベートされます（**org.apache.camel.k:camel-k-runtime-cron** ライブラリーにあります）。 現在、サポートされているコンポーネントは **cron**、**タイマー**、および **quartz** です。
cron.fallback	bool	インテグレーションを Kubernetes CronJob としてマテリアル化するのではなく、**cron** エンドポイント(**quartz**)のデフォルトの Camel 実装を使用します。
cron.concurrency-policy	string	ジョブの同時実行を処理する方法を指定します。有効な値は Allow、Forbit (デフォルト)、および Replace です。Allow: CronJob の同時実行を許可します。Forbid: 同時実行を禁止し、前の実行が終了していない場合は次の実行をスキップします。Replace: 同時実行をキャンセルし、新しいジョブに置き換えます。
cron.auto	bool	すべてのルートが定期的なコンシューマーから開始される場合（**cron**、**タイマー**、および **quartz** のみ）またはパッシブコンシューマー（例： **direct** はパッシブコンシューマー）から開始されると、インテグレーションを CronJob として自動的にデプロイします。 定期的なコンシューマーすべてに期間が同じで、cron スケジュールとして表現することができます（例： **1m** は **0/1 * * * ***, **35m** または **50s** として表現することはできません）。

8.2.4. GC トレイト

GC トレイトは、インテグレーションの更新時に不要になったすべてのリソースをガベージコレクションで処理します。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.4.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait gc.[key]=[value] --trait gc.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
gc.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
gc.discovery-cache	./pkg/trait.discoveryCacheType	使用する検出クライアントキャッシュ。**無効** 化、**ディスク**、または **メモリー**（デフォルト **メモリー**）

8.2.5. Istio トレイト

Istio トレイトは、サイドカー注入やアウトバウンド IP 範囲など、Istio サービスに関連するプロパティーの設定を可能にします。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.5.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait istio.[key]=[value] --trait istio.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
istio.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
istio.allow	string	Istio プロキシー（デフォルトで **は10.0.0.0/8,172.16.0.0/12,192.168.0.0/16）でインターセプトできない CIDR サブネットの（コンマ区切り）リストを設定します。**

プロパティー	タイプ	説明
istio.inject	bool	ラベル **sidecar.istio.io/inject** の値を強制します。デフォルトで、ラベルはデプロイメントで **true** に設定され、Knative Service では設定されません。

8.2.6. Jolokia トレイト

Jolokia トレイトは、Jolokia Java エージェントをアクティベートおよび設定します。

https://jolokia.org/reference/html/agents.html を参照してください。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.6.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait jolokia.[key]=[value] --trait jolokia.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
jolokia.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
jolokia.ca-cert	string	PEM でエンコードされた CA 証明書ファイルパス。**プロトコル** が **https** で、**use-ssl-client-authentication** が **true** の場合に適用されます（OpenShift のデフォルトは **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt**）。
jolokia.client-principal	[]string	Jolokia エンドポイントへのアクセスを許可するためにクライアント証明書で指定する必要のあるプリンシパル。**プロトコル** が **https** で、**use-ssl-client-authentication** が **true** の場合に適用されます（デフォルトの **clientPrincipal=cn=system:master-proxy,cn=hawtio-online.hawtio.svc** および **cn=fuse-console.fuse.svc**）。
jolokia.discovery-enabled	bool	マルチキャスト要求をリッスン（デフォルトは **false**）
jolokia.extended-client-check	bool	クライアント証明書の拡張キー使用セクションにクライアントフラグが含まれること。**プロトコル** が **https** で、**use-ssl-client-authentication** が **true** の場合に適用されます（OpenShift のデフォルト）。

プロパティー	タイプ	説明
jolokia.host	string	Jolokia エージェントがバインドする必要のあるホストアドレス。「*"」 または「**0**.0.0.0」を指定すると、サーバーはすべてのネットワークインターフェースにバインドします（デフォルトは「*****」）。
jolokia.password	string	認証に使用されるパスワード。**ユーザー** オプションが設定されている場合に適用されます。
jolokia.port	int	Jolokia エンドポイントポート（デフォルトは **8778**）。
jolokia.protocol	string	使用するプロトコル（**http** または **https** （OpenShift のデフォルト **https** ）
jolokia.user	string	認証に使用されるユーザー
jolokia.use-ssl-client-authentication	bool	認証にクライアント証明書を使用するかどうか（OpenShift のデフォルトは **true** ）。
jolokia.options	[]string	追加の Jolokia オプションのリストは、「 JVM agent configuration options」に定義されています。

8.2.7. Knative トレイト

Knative トレイトは Knative リソースのアドレスを自動検出し、実行中のインテグレーションに注入します。

完全な Knative 設定は、JSON 形式の CAMEL_KNATIVE_CONFIGURATION に注入されます。その後、Camel Knative コンポーネントは完全な設定を使用してルートを設定します。

このトレイトは、Knative プロファイルがアクティブであるとデフォルトで有効になります。

このトレイトは、**Knative** プロファイルで利用できます。

8.2.7.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait knative.[key]=[value] --trait knative.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
knative.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。

プロパティー	タイプ	説明
knative.configuration	string	Knative の完全な設定を JSON 形式で注入するために使用されます。
knative.channel-sources	[]string	インテグレーションルートのソースとして使用されるチャネルの一覧。簡単なチャネル名または完全な Camel URI を含めることができます。
knative.channel-sinks	[]string	インテグレーションルートの宛先として使用されるチャネルの一覧。簡単なチャネル名または完全な Camel URI を含めることができます。
knative.endpoint-sources	[]string	インテグレーションルートのソースとして使用されるチャネルの一覧。
knative.endpoint-sinks	[]string	インテグレーションルートの宛先として使用されるエンドポイントの一覧。簡単なエンドポイント名または完全な Camel URI を含めることができます。
knative.event-sources	[]string	インテグレーションがサブスクライブされるイベントタイプのリスト。簡単なイベントタイプまたは完全な Camel URI を含めることができます (デフォルト以外の特定のブローカーを使用するため)。
knative.event-sinks	[]string	インテグレーションが生成するイベントタイプのリスト。簡単なイベントタイプまたは完全な Camel URI を含めることができます (特定のブローカーを使用するため)。
knative.filter-source-channels	bool	ヘッダー「ce-knativehistory」を基にしてイベントのフィルターを有効にします。これは今後の Knative バージョンで削除される可能性のある実験的なヘッダーであるため、インテグレーションが 1 つ以上のチャネルからリッスンする場合にのみフィルターが有効になります。
knative.camel-source-compat	bool	Knative CamelSource の 0.15 より前の互換性修正を有効にします
knative.sink-binding	bool	Knative SinkBinding リソース経由のインテグレーションからシンクへのバインドを許可します。これは、インテグレーションが単一のシンクをターゲットにする場合に使用できます。デフォルトでは無効になっています。
knative.auto	bool	すべてのトレイトプロパティーの自動検出を有効にします。

8.2.8. Knative Service トレイト

Knative Service トレイトは、標準の Kubernetes デプロイメントではなく、Knative サービスとしてインテグレーションを実行する場合に、オプションの設定を可能にします。

Knative Services としてインテグレーションを実行すると、自動スケーリング (およびゼロへのスケーリング) 機能が追加されますが、この機能はルートが HTTP エンドポイントコンシューマーを使用する場合にのみ有用です。

このトレイトは、**Knative** プロファイルで利用できます。

8.2.8.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait knative-service.[key]=[value] --trait knative-service.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
knative-service.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
knative-service.autoscaling-class	string	Knative 自動スケーリングクラスプロパティーを設定します（**hpa.autoscaling.knative.dev** または **kpa.autoscaling.knative.dev** の自動スケーリングを設定します）。 詳細は、Knative ドキュメントを参照してください。
knative-service.autoscaling-metric	string	Knative 自動スケーリングメトリクスプロパティーを設定します (例: **concurrency** または **cpu** ベースの自動スケーリングを設定します)。 詳細は、Knative ドキュメントを参照してください。
knative-service.autoscaling-target	int	各 Pod に許可される同時実行レベルまたは CPU の割合 (自動スケーリングメトリクスによる) を設定します。 詳細は、Knative ドキュメントを参照してください。
knative-service.min-scale	int	インテグレーションに対して稼働している必要がある Pod の最小数。デフォルトは **ゼロ** であるため、設定された期間に使用されなければインテグレーションはゼロにスケールダウンされます。 詳細は、Knative ドキュメントを参照してください。
knative-service.max-scale	int	インテグレーションで並行して実行できる Pod 数の上限。Knative には、インストールによって異なる独自の上限値があります。 詳細は、Knative ドキュメントを参照してください。

プロパティー	タイプ	説明
knative-service.auto	bool	以下のすべての条件が保持されると、インテグレーションを Knative サービスとして自動的にデプロイします。 ● インテグレーションは Knative プロファイルを使用する。 ● すべてのルートは、HTTP ベースのコンシューマーまたはパッシブコンシューマーから開始される (例: **direct** はパッシブコンシューマー)。

8.2.9. Master トレイト

Master トレイトでは、Kubernetes リソースを自動的に利用して、リーダー選出を行い、特定のインスタンスでのみ **マスター** ルートを開始するためにインテグレーションを設定できます。

これは、ルートでマスターエンドポイントを使用する際に自動的にアクティベートされます **（例： from("master:lockname:telegram:bots")...**

注記

このトレイトでは、configmap を読み書きし、Pod を読み取りために、特別なパーミッションがインテグレーションサービスアカウントに追加されます。インテグレーションの実行時に、デフォルト以外のサービスアカウントを使用することが推奨されます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.9.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait master.[key]=[value] --trait master.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
master.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
master.auto	bool	トレイトの自動設定を有効にします。
master.include-delegate-dependencies	bool	このフラグがアクティブである場合、operator はソースコードを分析し、委譲エンドポイントに必要な依存関係を追加します。たとえば、**master:lockname:timer** を使用する場合、**camel:timer** は依存関係のセットに自動的に追加されます。これはデフォルトで有効になっています。
master.configmap	string	ロックを保存するために使用される configmap の名前。デフォルトは <integration-name>-lock です。

プロパティー	タイプ	説明
master.label-key	string	ロックと競合するすべての Pod を特定するために使用されるラベル。デフォルトは camel.apache.org/integration です。
master.label-value	string	ロックと競合するすべての Pod を特定するために使用されるラベル値。デフォルトはインテグレーション名です。

8.2.10. Prometheus トレイト

Prometheus トレイトは、Prometheus と互換性のあるエンドポイントを設定します。また、このトレイトは **Service** および **ServiceMonitor** リソースでインテグレーションを公開するため、Prometheus Operator の使用時にエンドポイントを自動的にスクレープできます。

公開されるメトリクスは、設定されたランタイムによって異なります。デフォルトの Quarkus ランタイムでは、メトリクスは MicroProfile Metrics を使用して公開されます。Java メインランタイムでは、メトリクスは Prometheus JMX エクスポーターを使用して公開されます。

警告

ServiceMonitor リソースを作成するには、Prometheus Operator のカスタムリソース定義をインストールする必要があります。Prometheus トレイトが Prometheus Operator なしで機能するように、**service-monitor** を **false** に設定できます。

Prometheus トレイトはデフォルトで無効になっています。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.10.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait prometheus.[key]=[value] --trait prometheus.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
prometheus.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
prometheus.port	int	Prometheus エンドポイントポート（デフォルトは **9779** または Quarkus を使用した **8080**）。

プロパティー	タイプ	説明
prometheus.service-monitor	bool	**ServiceMonitor** リソースが作成されるかどうか（デフォルトは **true**）。
prometheus.service-monitor-labels	[]string	**ServiceMonitor** リソースラベル。**service-monitor** が **true** の場合に適用されます。
prometheus.configmap	string	Prometheus JMX エクスポーター設定が含まれるカスタム ConfigMap を使用する場合（ **コンテンツ** ConfigMap キー下）。このプロパティーが空のまま (デフォルト) であると、Camel K はインテグレーションに標準の Prometheus 設定を生成します。これは、Quarkus を使用する場合は該当しません。

8.2.11. Quarkus トレイト

Quarkus トレイトにより、Quarkus ランタイムがアクティベートされます。

これはデフォルトで有効になっています。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

8.2.11.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait quarkus.[key]=[value] --trait quarkus.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
quarkus.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
quarkus.native	bool	Quarkus ランタイムタイプ (将来的な使用のために予約)

8.2.11.2. サポートされる Camel コンポーネント

Quarkus が有効な状態を実行する場合、Camel K は Camel Quarkus エクステンションとして利用できる Camel コンポーネントをそのまま追加設定なしでのみサポートします。

エクステンションの一覧は、Camel Quarkus ドキュメント を参照してください。

8.2.12. Routeトレイト

Route トレイトを使用すると、インテグレーションの OpenShift ルートの作成を設定できます。

このトレイトは、**OpenShift** のプロファイルで利用できます。

8.2.12.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait route.[key]=[value] --trait route.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
route.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
route.host	string	ルートによって公開されるホストを設定します。
route.tls-termination	string	**edge**、**passthrough**、または **reencrypt** などの TLS 終端タイプ。 詳細は、OpenShift ドキュメントを参照してください。
route.tls-certificate	string	TLS 証明書の内容。 詳細は、OpenShift ドキュメントを参照してください。
route.tls-key	string	TLS 証明書キーの内容。 詳細は、OpenShift ドキュメントを参照してください。
route.tls-ca-certificate	string	TLS 認証局の証明書の内容。 詳細は、OpenShift ドキュメントを参照してください。
route.tls-destination-ca-certificate	string	宛先 CA 証明書は、最終宛先の CA 証明書の内容を提供します。reencrypt の停止を使用する場合、ルーターがセキュアな接続のヘルスチェックに使用するためにこのファイルを提供する必要があります。このフィールドが指定されていない場合、ルーターは独自の宛先 CA を提供し、短いサービス名 (service.namespace.svc) を使用してホスト名の検証を実行する可能性があります。これにより、インフラストラクチャーが生成した証明書を自動的に検証できます。 詳細は、OpenShift ドキュメントを参照してください。
route.tls-insecure-edge-termination-policy	string	セキュアでないトラフィック (**Allow**、**Disable**、または**Redirect** トラフィックなど) に対応する方法を設定します。 詳細は、OpenShift ドキュメントを参照してください。

8.2.13. Service トレイト

サービストレイトは、Service リソースとのインテグレーションを公開し、同じ namespace の他のアプリケーション (またはインテグレーション) からアクセスできるようにします。

インテグレーションが HTTP エンドポイントを公開できる Camel コンポーネントに依存する場合は、デフォルトで有効になっています。

このトレイトは **Kubernetes および OpenShift** プロファイルで利用できます。

8.2.13.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait service.[key]=[value] --trait service.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
service.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
service.auto	bool	サービスを作成する必要がある場合にコードから自動検出されます。
service.node-port	bool	Service が NodePort として公開できるようにします。

8.3. CAMEL K プラットフォームトレイト

8.3.1. Builder トレイト

Builder トレイトは、IntegrationKits を構築および設定するために最適なストラテジーを決定するために内部で使用されます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Builder トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.1.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait builder.[key]=[value] --trait builder.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
builder.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
builder.verbose	bool	サポートするビルドコンポーネント (OpenShift ビルド Pod など) で詳細なロギングを有効にします。Kaniko および Buildah はサポートされません。

8.3.2. Container トレイト

Container トレイトを使用すると、インテグレーションが実行されるコンテナーのプロパティーを設定できます。

また、コンテナーに関連付けられたサービスの設定も提供します。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Container トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.2.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait container.[key]=[value] --trait container.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
container.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
container.auto	bool	
container.request-cpu	string	必要な CPU の最小量。
container.request-memory	string	必要なメモリーの最小容量。
container.limit-cpu	string	必要な CPU の最大量。

プロパティー	タイプ	説明
container.limit-memory	string	必要なメモリーの最大容量。
container.expose	bool	kubernetes サービス経由の公開を有効または無効にするために使用できます。
container.port	int	コンテナーによって公開される別のポートを設定します (デフォルトは **8080**)。
container.port-name	string	コンテナーによって公開されるポートに異なるポート名を設定します (デフォルトは **http**)。
container.service-port	int	コンテナーポートを公開するサービスポートを設定します (デフォルト **80**)。
container.service-port-name	string	コンテナーポートを公開するサービスポート名を設定します (デフォルト **http**)。
container.name	string	メインのコンテナー名。デフォルトでは、名前付き **integration** になります。
container.probes-enabled	bool	コンテナーのプローブで ProbesEnabled を有効/無効にします (デフォルトは **false**)。
container.probe-path	string	プローブでアクセスするパス（デフォルトは **/health**）。このプロパティーは quarkus ランタイムではサポートされず、このプロパティーを設定すると、インテグレーションの起動に失敗するため注意してください。
container.liveness-initial-delay	int32	コンテナーが起動してから liveness プローブが開始されるまでの秒数。
container.liveness-timeout	int32	プローブがタイムアウトするまでの秒数。liveness プローブに適用されます。
container.liveness-period	int32	プローブを実行する頻度。liveness プローブに適用されます。
container.liveness-success-threshold	int32	プローブの失敗後、正常とみなされるための最小の連続成功回数。liveness プローブに適用されます。
container.liveness-failure-threshold	int32	プローブの正常実行後、失敗とみなされるための最小連続失敗回数。liveness プローブに適用されます。
container.readiness-initial-delay	int32	コンテナーが起動してから readiness プローブが開始されるまでの秒数。

プロパティー	タイプ	説明
container.readiness-timeout	int32	プローブがタイムアウトするまでの秒数。readiness プローブに適用されます。
container.readiness-period	int32	プローブを実行する頻度。readiness プローブに適用されます。
container.readiness-success-threshold	int32	プローブの失敗後、正常とみなされるための最小の連続成功回数。readiness プローブに適用されます。
container.readiness-failure-threshold	int32	プローブの正常実行後、失敗とみなされるための最小連続失敗回数。readiness プローブに適用されます。

8.3.3. Camel トレイト

Camel トレイトを使用すると Apache Camel K ランタイムおよび関連ライブラリーのバージョンを設定できますが、無効にすることはできません。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Camel トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.3.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait camel.[key]=[value] --trait camel.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
camel.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
camel.runtime-version	string	インテグレーションに使用する camel-k-runtime バージョン。Integration Platform に設定されたデフォルトのバージョンを上書きします。

8.3.4. Dependencies トレイト

Dependencies トレイトは、ユーザーが実行するインテグレーションに基づいてランタイムの依存関係を自動的に追加するために内部で使用されます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Dependencies トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.4.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait dependencies.[key]=[value] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
dependencies.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。

8.3.5. Deployer トレイト

Deployer トレイトを使用すると、インテグレーションをデプロイする高レベルのリソースの種類を明示的に選択できます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Deployer トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.5.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait deployer.[key]=[value] --trait deployer.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
deployer.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
deployer.kind	string	インテグレーションを実行するリソースを作成する際に、**deployment**、**cron-job**、または **knative-service** との間で必要なデプロイメントの種類を明示的に選択できます。

8.3.6. Deployment トレイト

Deployment トレイトは、インテグレーションがクラスターで実行されるようにする Kubernetes デプロイメントを生成します。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Deployment トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.6.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait deployment.[key]=[value] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
deployment.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。

8.3.7. Environment トレイト

Environment トレイトは、**NAMESPACE**、**POD_NAME** などのインテグレーションコンテナーに標準の環境変数を注入するために内部で使用されます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Environment トレイトは**プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.7.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait environment.[key]=[value] --trait environment.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
environment.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
environment.container-meta	bool	

8.3.8. JVM トレイト

JVM トレイトは、インテグレーションを実行する JVM の設定に使用されます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

JVM トレイトは **プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.8.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait jvm.[key]=[value] --trait jvm.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
jvm.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
jvm.debug	bool	リモートデバッグをアクティベートし、たとえばポート転送を使用して、デバッガーを JVM に接続できるようにします。
jvm.debug-suspend	bool	メインクラスがロードされる直前にターゲット JVM を一時停止します。
jvm.print-command	bool	コンテナーログに JVM の開始に使用されるコマンドを出力します（デフォルトは **true**）。
jvm.debug-address	string	新たに起動された JVM をリッスンするトランスポートアドレス（デフォルトは ***:5005**）
jvm.options	[]string	JVM オプションの一覧

8.3.9. Openapi トレイト

OpenAPI DSL トレイトは、OpenAPI 仕様からインテグレーションを作成できるように内部で使用されます。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

openapi トレイトは **プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.9.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait openapi.[key]=[value] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
openapi.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。

8.3.10. Owner トレイト

Owner トレイトは、作成されたすべてのリソースが作成されたインテグレーションに属すようにし、インテグレーションのアノテーションおよびラベルをこれらの所有されたリソースに転送します。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

Owner トレイトは **プラットフォームトレイト** です。よって、これを無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.10.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait owner.[key]=[value] --trait owner.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
owner.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
owner.target-annotations	[]string	転送するアノテーションのセット
owner.target-labels	[]string	転送するラベルのセット

8.3.11. プラットフォームトレイト

プラットフォームトレイトは、インテグレーションプラットフォームをインテグレーションに割り当てるために使用されるベーストレイトです。

プラットフォームが見つからない場合、トレイトはデフォルトのプラットフォームを作成できます。この機能は、プラットフォームにカスタム設定が必要ない場合に便利です (例: 組み込みのコンテナーイメージレジストリーがあるため OpenShift ではデフォルトの設定が動作します)。

このトレイトは **Kubernetes、Knative、および OpenShift** プロファイルで利用できます。

警告

プラットフォームトレイト を無効にすると、プラットフォームの機能性が低下する可能性があります。

8.3.11.1. 設定

CLI でインテグレーションを実行する際にトレイトプロパティーを指定できます。

```
kamel run --trait platform.[key]=[value] --trait platform.[key2]=[value2] Integration.java
```

以下の設定オプションが利用できます。

プロパティー	タイプ	説明
platform.enabled	bool	トレイトを有効または無効にするのに使用できます。すべてのトレイトがこの共通プロパティーを共有します。
platform.create-default	bool	プラットフォームがない場合にデフォルトのプラットフォーム (空のプラットフォーム) を作成します。
platform.auto	bool	デフォルトのプラットフォームを作成できる場合に環境から自動検出します (OpenShift のみで作成)。

第9章 CAMEL K コマンドリファレンス

本章では、Camel K コマンドラインインターフェース (CLI) の参考情報と、**kamel** コマンドの使用例を紹介します。本章では、ランタイムで実行される Camel K インテグレーションソースファイルで指定できる Camel K モードラインオプションの参考情報も提供します。

本章には、以下が含まれます。

- 「Camel K コマンドライン」
- 「Camel K モードラインオプション」

9.1. CAMEL K コマンドライン

Camel K CLI は、OpenShift で Camel K インテグレーションを実行するためのメインエントリーポイントとして **kamel** コマンドを提供します。ここでは、最も一般的に使用される **kamel** コマンドの詳細を説明します。

表9.1 kamel コマンド

名前	説明	例
help	利用可能なコマンドの完全リストを取得します。詳細は、各コマンドのパラメーターとして **--help** を入力します。	• **kamel help** • **kamel run --help**
init	Java、XML、または YAML に実装された空の Camel K ファイルを初期化します。	**kamel init MyIntegration.java**
run	OpenShift でインテグレーションを実行します。	**kamel run MyIntegration.java**
debug	ローカルデバッガーを使用してリモートインテグレーションをデバッグします。	**kamel debug my-integration**
get	OpenShift にデプロイされたインテグレーションを取得します。	**kamel get**
describe	Camel K リソースの詳細情報を取得します。これには、**integration**、**kit**、または **platform** が含まれます。	**kamel describe integration my-integration**
log	実行中のインテグレーションのログを出力します。	**kamel log my-integration**
delete	OpenShift にデプロイされたインテグレーションを削除します。	**kamel delete my-integration**

関連情報

- 「Camel K および OpenShift コマンドラインツールのインストール」

9.2. CAMEL K モードラインオプション

Camel K モードラインを使用すると、たとえば **kamel run MyIntegration.java** を使用して、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設定オプションを入力できます。詳細は「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」 を参照してください。

ここでは、最も一般的に使用されるモードラインオプションについての参考情報を提供します。

表9.2 Camel K モードラインオプション

オプション	説明
dependency	インテグレーションに含める外部ライブラリーを追加します。たとえば、Maven の場合は **dependency=mvn:org.my/app:1.0 を使用します**。または GitHub の場合は **dependency=github:my-account:camel-k-example-project:master** を使用します。
env	インテグレーションコンテナーに環境変数を設定します。例: **env=MY_ENV_VAR=my-value**
ラベル	インテグレーションのラベルを追加します。例: **label=my.company=hello**
name	インテグレーション名を追加します。例: **name=my-integration**
open-api	OpenAPI v2 仕様を追加します。たとえば、**open-api=path/to/my-hello-api.json** です。
profile	デプロイメントに使用する Camel K トレイトプロファイルを設定します。例: **openshift**
プロパティー	インテグレーションプロパティーを追加します。例：**property=my.message="Hola Mundo"**
property-file	プロパティーファイルをインテグレーションにバインドします。例：**property-file=my-integration.properties**
resource	外部リソースを追加します。例： **resource=path/to/my-hello.txt**
trait	トレイトで Camel K 機能またはコア機能を設定します。例：**trait=cron.enabled=true**