Red Hat Integration 2022.Q2

Kamelets でのアプリケーションの統合

アプリケーションの統合を簡素化するコネクターの設定

Last Updated: 2023-10-09

Red Hat Integration 2022.Q2 Kamelets でのアプリケーションの統合

アプリケーションの統合を簡素化するコネクターの設定

法律上の通知

概要

Kamelets は、アプリケーション統合の代替アプローチを提供します。Camel コンポーネントを直接使用する代わりに、Kamelets(独断的なルートテンプレート) を設定して接続を作成できます。

目次

はじめに

Kamelets は、外部システムに接続するデータパイプライン作成の複雑さを隠す、再利用可能なルートコンポーネントです。

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 KAMELETS の概要

Kamelets は、イベント駆動型のアーキテクチャーソリューションでビルディングブロックとして使用できる高レベルのコネクターです。これらは OpenShift クラスターにインストールし、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です。Kamelets は開発作業を加速します。これらは、データソース (イベントを出力する) およびデータシンク (イベントを消費する) の接続を単純化します。コードを記述するのではなく Kamelet パラメーターを設定するため、Kamelets を使用するのに Camel DSL を理解する必要はありません。

Kamelets を使用して、アプリケーションとサービスを相互に直接接続することができます。あるいは、以下の項目に接続することができます。

- Kafka トピック (Kamelets を使用した Kafka への接続 で説明)
- Knative 宛先 (チャネルまたはブローカー)(Kamelets を使用した Knative への接続 で説明)
- 特定の Camel URI (明示的な Camel URI への接続 で説明)

1.1. KAMELETS について

Kamelets は、Camel インテグレーションでコネクターとして動作するルートコンポーネント (カプセル化されたコード) です。Kamelets は、データの消費元 (ソース) およびデータの送信先 (シンク) を定義してデータパイプラインをアセンブルできるテンプレートとみなすことができます。Kamelets は、データのフィルタリング、マスク、および単純な計算ロジックを実行することもできます。

Kamelets には、以下の 3 つのタイプがあります。

- **ソース**: データを作成するルート。ソース Kamelet を使用してコンポーネントからデータを取得します。
- **シンク**: データを消費するルート。シンク Kamelet を使用して、データをコンポーネントに送信します。
- **アクション**: データに対してアクションを実行するルート。アクション Kamelet を使用して、ソース Kamelet からシンク Kamelet にデータを渡す際に、データを操作できます。

1.1.1. Kamelets を使用する理由

マイクロサービス および イベント駆動型アーキテクチャー ソリューションでは、Kamelets はイベントを生成するソースおよびイベントを消費するシンク用のビルディングブロックとして機能します。

Kamelets は、抽象化 (外部システムへの接続の複雑さを隠します) および再利用性 (コードを再利用し、異なるユースケースに適用する簡単な方法です) を提供します。

使用例を以下に示します。

- アプリケーションが Telegram からイベントを消費するようにするには、Kamelets を使用して Telegram ソースをイベントのチャネルにバインドできます。後に、それらのイベントに反応するように、アプリケーションをそのチャネルに接続できます。
- アプリケーションが Salesforce を直接 Slack に接続することを希望します。

Kamelets を使用すると、統合開発チームの効率が高くなります。Kamelets を再利用し、特定のニーズに合わせてインスタンスを設定できるチームメンバーと共有できます。基礎となる Camel K Operator はさまざまな作業を行います。これは Kamelet で定義されたインテグレーションをコンパイルし、ビルドし、パッケージ化し、デプロイします。

1.1.2. Kamelets を使用するユーザー

Kame を使用すると、Camel インテグレーションに必要なコーディングの量を減らすことができるため、Camel DSL に精通していない開発者に適しています。Kamelets は、Camel 以外の開発者の学習曲線を円滑化することができます。Camel を稼働させるのに、別のフレームワークまたは言語を学ぶ必要はありません。

Kamelets は、複雑な Camel 統合ロジックを再利用可能な Kamelet にカプセル化し、他のユーザーと共有したい経験のある Camel 開発者にも便利です。

1.1.3. Kamelets を使用するための前提条件

Kamelets を使用するには、以下の環境を設定する必要があります。

- 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、OpenShift および Camel K 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。
- Camel K のインストール で説明されているように、namespace またはクラスター全体に Camel K Operator がインストールされている。
- OpenShift コマンドライン (**cc**) インターフェイスツールがインストールされている。
- 必要に応じて、VS コードまたは別の開発ツールを Camel K プラグインと共にインストールしている。Camel ベースのツールエクステンションには、埋め込み Kamelet Catalog に基づく Camel URI の自動補完などの機能が含まれます。詳細は、**Camel K のスタートガイド** の Camel K 開発ツール セクションを参照してください。
  注記: Visual Studio (VS) Code Tooling エクステンションはコミュニティーのみです。

1.1.4. Kamelets の使用方法

Kamelet を使用する場合には通常、再利用可能なルートスニペットを定義する Kamelet 自体と、1 つ以上の Kamelets を参照してバインドする Kamelet Binding という 2 つのコンポーネントが必要です。Kamelet Binding は OpenShift リソース (**KameletBinding**) です。

Kamelet Binding リソース内では、以下を実行できます。

- シンクまたはソース Kamelet を、Kafka トピックまたは Knative 宛先 (チャネルまたはブローカー) のイベントチャネルに接続します。
- シンク Kamelet を直接 Camel Uniform Resource Identifier (URI) に接続します。URI およびシンク Kamelet の接続は最も一般的なユースケースであるものの、ソース Kamelet を Camel URI に接続することもできます。
- イベントのチャネルを中間層として使用せずに、シンクおよびソース Kamelet を直接相互に接続します。
- 同じ Kamelet Binding で同じ Kamelet を複数回参照します。
- ソース Kamelet からシンク Kamelet にデータを渡す際にデータを操作する、アクション Kamelet を追加します。
- イベントデータの送受信時に失敗した場合に Camel K が何を行うべきかを指定する、エラー処理ストラテジーを定義します。

実行時に、Camel K Operator は Kamelet Binding を使用して Camel K インテグレーションを生成し、実行します。

注記: Camel DSL の開発者は Camel K インテグレーションで Kamelets を直接使用できますが、Kamelets を実装する簡単な方法は、Kamelet Binding リソースを指定して高レベルのイベントフローを構築することです。

1.2. ソースおよびシンクの接続

2 つ以上のコンポーネント (外部アプリケーションまたはサービス) を接続する場合は Kamelets を使用します。各 Kamelet は基本的に設定プロパティーを持つルートテンプレートです。データの取得元となるコンポーネント (ソース) およびデータの送信先となるコンポーネント (シンク) を知っている必要があります。図 1.1 で説明されているように、ソースおよびシンクコンポーネントを接続するには、Kamelet Binding に Kamelets を追加します。

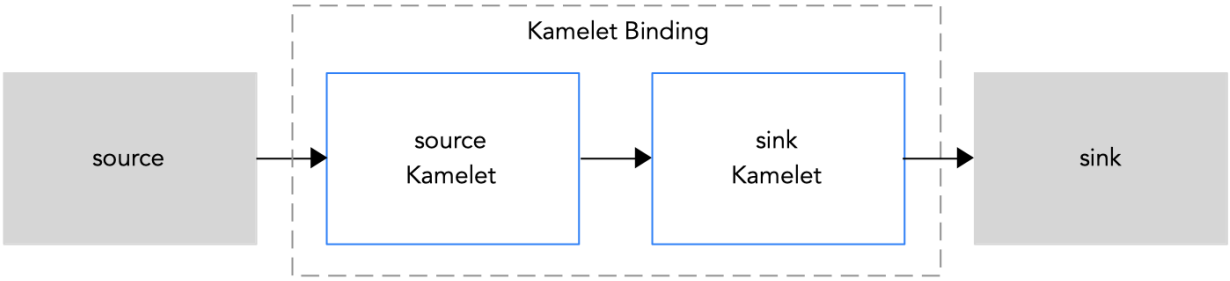


図 1.1: ソースからシンクへの Kamelet Binding

以下は、Kamelet Binding で Kamelets を使用する手順の概要です。

1. Camel K Operator をインストールします。これには、OpenShift プロジェクトのリソースとしての Kamelets のカタログが含まれます。
2. Kamelet Binding を作成します。Kamelet Binding 内で接続するサービスまたはアプリケーションを決定します。
3. Kamelet Catalog を表示して、使用するソースおよびシンクコンポーネントの Kamelets を検索します。
4. Kamelet Binding に追加する各 Kamelet について、設定が必要な設定プロパティーを決定します。
5. Kamelet Binding コードで、各 Kamelet への参照を追加し、必要なプロパティーを設定します。
6. Kamelet Binding を OpenShift プロジェクトのリソースとして適用します。

Camel K Operator は Kamelet Binding を使用してインテグレーションを生成し、実行します。

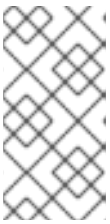
1.2.1. Camel K のインストール

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イスを提供します。

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。



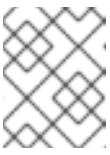
注記

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 をデプロイメント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

(オプション) Camel K リソース制限の指定

1.2.2. Kamelet カタログの表示

Camel K Operator をインストールする場合、Camel K インテグレーションで使用できる Kamelets のカタログが含まれます。

前提条件

Camel K のインストール で説明されているように、作業用 namespace またはクラスター全体に Camel K Operator がインストールされている。

手順

Camel K Operator でインストールされた Kamelets のリストを表示するには、以下を実行します。

1. ターミナルウィンドウで、OpenShift クラスターにログインします。

2. 利用可能な Kamelets のリストを表示する方法は、Camel K Operator のインストール方法 (特定の namespace またはクラスターモード) によって異なります。

   - Camel K Operator がクラスターモードでインストールされている場合は、以下のコマンドを使用して、利用可能な Kamelets を表示します。
     oc get kamelet -n openshift-operators

   - Camel K Operator が特定の namespace にインストールされている場合は、以下を行います。

     a. Camel K Operator がインストールされているプロジェクトを開きます。
     oc project <camelk-project>

     たとえば、Camel K Operator が **my-camel-k-project** プロジェクトにインストールされている場合は、以下のようになります。

oc project my-camel-k-project

b. 以下のコマンドを実行します。
oc get kamelets

注記

Red Hat がサポートする Kamelets のリストは、Red Hat Integration Release Notes を参照してください。

関連項目

カスタム Kamelet の Kamelet Catalog への追加

1.2.2.1. カスタム Kamelet の Kamelet Catalog への追加

要件に適した Kamelet がカタログに表示されない場合は、Camel DSL の開発者は Apache Camel Kamelets Developers Guide (コミュニティードキュメント) の説明どおりにカスタム Kamelet を作成できます。Kamelet は **YAML** 形式でコード化され、規則により **.kamelet.yaml** ファイル拡張子を持ちます。

前提条件

- Camel DSL 開発者がカスタムの Kamelet ファイルを提供している。
- Kamelet 名は、Camel K Operator がインストールされている OpenShift namespace で一意である必要があります。

手順

カスタム Kamelet を OpenShift namespace のリソースとして利用可能にするには、以下を実行します。

1. Kamelet **YAML** ファイル (**custom-sink.kamelet.yaml** など) をローカルフォルダーにダウンロードします。
2. OpenShift クラスターにログインします。
3. ターミナルウィンドウで、Camel K Operator がインストールされているプロジェクトを開きます (例: **my-camel-k-project**)。
oc project my-camel-k-project
4. **oc apply** コマンドを実行して、カスタム Kamelet をリソースとして namespace に追加します。
oc apply -f <custom-kamelet-filename>

   たとえば、以下のコマンドを使用して、現在のディレクトリーにある **custom-sink.kamelet.yaml** ファイルを追加します。

   oc apply -f custom-sink.kamelet.yaml
5. Kamelet がリソースとして利用できることを確認するには、以下のコマンドを使用して現在の namespace のすべての Kamelets のアルファベット順のリストを表示してから、カスタム Kamelet を検索します。
oc get kamelets

1.2.2.2. Kamelet の設定パラメーターの決定

Kamelet Binding で Kamelet への参照を追加する場合、Kamelet の名前を指定し、Kamelet のパラメーターを設定します。

前提条件

- 作業用 namespace またはクラスター全体に Camel K Operator がインストールされている。

手順

Kamelet の名前およびパラメーターを決定するには、以下を実行します。

1. ターミナルウィンドウで、OpenShift クラスターにログインします。
2. Kamelet の YAML ファイルを開きます。
   oc describe kamelets/<kamelet-name>

   たとえば、**ftp-source** Kamelet のコードを表示するには、Camel K Operator が現在の namespace にインストールされている場合は、以下のコマンドを使用します。

   oc describe kamelets/ftp-source

   Camel K Operator が cluster-mode でインストールされている場合は、以下のコマンドを使用します。

   oc describe -n openshift-operators kamelets/ftp-source
3. YAML ファイルで、**spec.definition** セクション (JSON-schema 形式で記述される) まで下方向にスクロールし、Kamelet のプロパティーのリストを表示します。セクションの最後の必須フィールドには、Kamelet を参照する際に設定する必要のあるプロパティーがリスト表示されます。
   たとえば、以下のコードは **ftp-source** Kamelet の **spec.definition** セクションからの抜粋です。このセクションは、Kamelet のすべての設定プロパティーの詳細を提供します。この Kamelet に必要なプロパティーは、**connectionHost**、**connectionPort**、**username**、**password**、および **directoryName** です。

```
spec:
  definition:
    title: "FTP Source"
    description: |-
      Receive data from an FTP Server.
    required:
      - connectionHost
      - connectionPort
      - username
      - password
      - directoryName
    type: object
    properties:
      connectionHost:
        title: Connection Host
        description: Hostname of the FTP server
        type: string
      connectionPort:
        title: Connection Port
```

```
    description: Port of the FTP server
    type: string
    default: 21
  username:
    title: Username
    description: The username to access the FTP server
    type: string
  password:
    title: Password
    description: The password to access the FTP server
    type: string
    format: password
    x-descriptors:
    - urn:alm:descriptor:com.tectonic.ui:password
  directoryName:
    title: Directory Name
    description: The starting directory
    type: string
  passiveMode:
    title: Passive Mode
    description: Sets passive mode connection
    type: boolean
    default: false
    x-descriptors:
    - 'urn:alm:descriptor:com.tectonic.ui:checkbox'
  recursive:
    title: Recursive
    description: If a directory, will look for files in all the sub-directories as well.
    type: boolean
    default: false
    x-descriptors:
    - 'urn:alm:descriptor:com.tectonic.ui:checkbox'
  idempotent:
    title: Idempotency
    description: Skip already processed files.
    type: boolean
    default: true
    x-descriptors:
    - 'urn:alm:descriptor:com.tectonic.ui:checkbox'
```

関連項目

Kamelet インスタンスのパラメーターの設定

1.2.3. Kamelet Binding でのソースおよびシンクコンポーネントの接続

Kamelet Binding 内で、ソースおよびシンクコンポーネントを接続します。

この手順の例では、図 1.2 で示されているように、以下の Kamelets を使用しています。

- ソース Kamelet の例 は **coffee-source** という名前です。この簡単な Kamelet は、Web サイトカタログからコーヒーのタイプについて無作為に生成されたデータを取得します。コーヒーのデータを取得する頻度 (秒単位) を決定する 1 つのパラメーター (**period**: **integer** 値) があります。デフォルト値 (1000 秒) があるため、このパラメーターは必須ではありません。

- シンク Kamelet のサンプルの名前は **log-sink** です。これはデータを取得し、それをログファイルに出力します。**log-sink** Kamelet は Kamelet Catalog で提供されます。

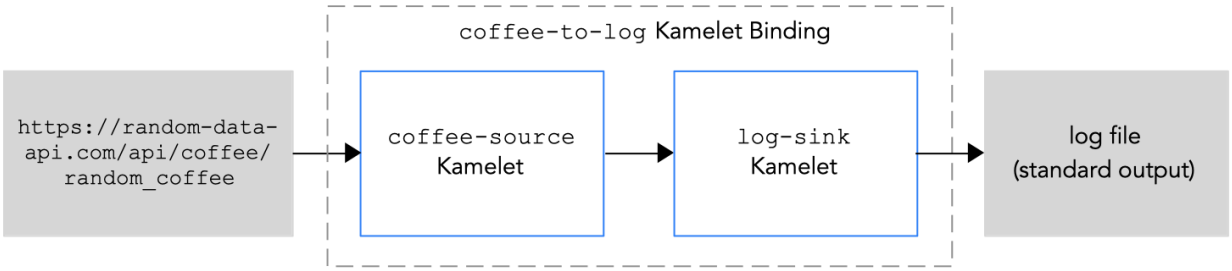


図 1.2: Kamelet Binding の例

前提条件

- Camel K インテグレーションの作成および編集方法を把握している。
- Camel K のインストール で説明されているように、**Red Hat Integration - Camel K**Operator が OpenShift namespace またはクラスターにインストールされ、Red Hat Integration Camel K CLI ツールをダウンロードしている。
- Camel K インテグレーションに追加する Kamelets と必要なインスタンスパラメーターを知っている必要があります。
- 使用する Kamelets は Kamelet Catalog で利用できます。
  この例では、**log-sink** Kamelet は Kamelet Catalog で提供されます。この例のソース Kamelet を使用する場合は、**coffee-source** code を **coffee-source.kamelet.yaml** という名前のローカルファイルにコピーして保存し、以下のコマンドを実行して Kamelet Catalog に追加します。

  oc apply -f coffee-source.kamelet.yaml

手順

1. OpenShift クラスターにログインします。
2. Camel K Operator がインストールされている作業プロジェクトを開きます。Camel K Operator を cluster-mode でインストールした場合、クラスター上の任意のプロジェクトで利用できます。
   たとえば、**my-camel-k-project** という名前の既存のプロジェクトを開くには、以下のコマンドを実行します。

   oc project my-camel-k-project
3. 以下のオプションのいずれかを使用して、新しい Kamelet Binding を作成します。
   - **kamel bind** コマンドを使用して Kamelet Binding を作成し、実行します (このオプションは、コマンドライン定義の助けとなる単純な Kamelet Binding に役に立ちます)
   - YAML ファイルを作成して Kamelet Binding を定義し、**oc apply** コマンドを使用して実行します (このオプションは Kamelet Binding 設定が複雑である場合に便利です)。
     kamel bind コマンドを使用した新しい Kamelet Binding の作成

以下の **kamel bind** 構文を使用して、ソースおよびシンク Kamelets および設定パラメーターを指定します。

```
kamel bind <kamelet-source> -p “<property>=<property-value>” <kamelet-sink> -p “<property>=<property-value>”
```

以下に例を示します。

```
kamel bind coffee-source -p “source.period=5000” log-sink -p "sink.showStreams=true"
```

Camel K Operator は **KameletBinding** リソースを生成し、対応する Camel K インテグレーションを実行します。

YAML ファイルを使用した新しい Kamelet Binding の作成

a. 任意のエディターで、以下の構造で YAML ファイルを作成します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name:
spec:
  source:
  sink:
```

b. Kamelet Binding の名前を追加します。
この例では、バインディングが **coffee-source** Kamelet を **log-sink** Kamelet に接続するため、名前は **coffee-to-log** になります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
spec:
  source:
  sink:
```

c. ソース Kamelet (例: **coffee-source**) を指定し、Kamelet のパラメーターを設定します。
注記: この例では、パラメーターは Kamelet Binding の YAML ファイル内に定義されます。または、Configuring Kamelet instance parameters で説明されているように、プロパティーファイル、ConfigMap、または Secret に Kamelet のパラメーターを設定できます。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
spec:
  source:
    ref
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
```

```
    properties:
      period: 5000
  sink:
```

d. シング Kamelet (例: **log-sink**) を指定し、Kamelet のパラメーターを設定します。**log-sink** Kamelet にオプションの **showStreams** パラメーターを使用してメッセージのボディーを表示します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
    properties:
      showStreams: true
```

e. YAML ファイルを保存します (例: **coffee-to-log.yaml**)。

f. KameletBinding をリソースとして OpenShift namespace に追加します (**KameletBinding**)。
oc apply -f <kamelet-binding>.yaml

以下に例を示します。

oc apply -f coffee-to-log.yaml

Camel K Operator は、**KameletBinding** リソースを使用して Camel K インテグレーションを生成し、実行します。

4. Kamelet Binding のステータスを表示するには、以下を実行します。
oc get kameletbindings

5. 対応するインテグレーションの状態を表示するには、**oc get integrations**

6. 出力を表示するには、以下を実行します。

- コマンドラインからのログを表示するには、ターミナルのウィンドウを開き、以下のコマンドを入力します。
kamel log <integration-name>

たとえば、インテグレーション名が **coffee-to-log** の場合は、以下のコマンドを使用します。

kamel log coffee-to-log

- OpenShift Web コンソールからのログを表示するには、以下を実行します。

  a. **Workloads** > **Pods** を選択します。

  b. Camel K インテグレーションの Pod の名前をクリックし、**Logs** をクリックします。
  以下の例のようなコーヒーイベントのリストが表示されるはずです。

  > INFO [log-sink-E80C5C904418150-0000000000000001] (Camel (camel-1) thread #0 - timer://tick) {"id":7259,"uid":"a4ecb7c2-05b8-4a49-b0d2-d1e8db5bc5e2","blend_name":"Postmodern Symphony","origin":"Huila, Colombia","variety":"Kona","notes":"delicate, chewy, black currant, red apple, star fruit","intensifier":"balanced"}

7. インテグレーションを停止するには、Kamelet Binding を削除します。
oc delete kameletbindings/<kameletbinding-name>

   以下に例を示します。

   oc delete kameletbindings/coffee-to-log

次のステップ

任意で以下を行います。

- Adding an operation to a Kamelet Binding で説明されているように、アクション Kamelets を中間ステップとして追加します。
- Adding an error handler policy to a Kamelet Binding で説明されているように、Kamelet Binding にエラー処理を追加します。

1.2.4. Kamelet インスタンスのパラメーターの設定

Kamelet を参照する場合は、Kamelet のインスタンスパラメーターを定義する以下のオプションを使用できます。

- Kamelet URI を指定する Kamelet Binding に直接以下の例では、Telegram BotFather が提供するボット承認トークンは **123456** です。
from("kamelet:telegram-source?authorizationToken=123456")
- 以下の形式を使用して、Kamelet プロパティーをグローバルに設定します (したがって、URI の値を指定する必要はありません)。
"camel.kamelet.<kamelet-name>.<property-name>=<value>”

  Camel K を使用したインテグレーションの開発および管理 の Camel K インテグレーションの設定 の章で説明されているように、以下の方法で Kamelet パラメーターを設定できます。

  - プロパティーとしての定義
  - プロパティーファイルでの定義
  - OpenShift ConfigMap またはシークレットでの定義

関連項目

kamelet の設定パラメーターの決定

1.2.5. イベントのチャネルへの接続

Kamelets の最も一般的なユースケースは、Kamelet Binding を使用して、Kafka トピックまたは Knative 宛先 (チャネルまたはブローカー) のイベントチャネルに接続することです。これを実行する利点は、データソースとシンクが独立しており、お互いに認識しないということです。このように切り離すことで、ビジネスシナリオのコンポーネントを個別に開発し、管理できます。ビジネスシナリオの一部として複数のデータシンクおよびソースがある場合、さまざまなコンポーネントを分離することがより重要です。たとえば、イベントシンクをシャットダウンする必要がある場合、イベントソースは影響を受けません。さらに、他のシンクが同じソースを使用する場合、それらは影響を受けません。

図 1.3 は、ソースおよびシンク Kamelets をイベントチャネルに接続するフローを示しています。

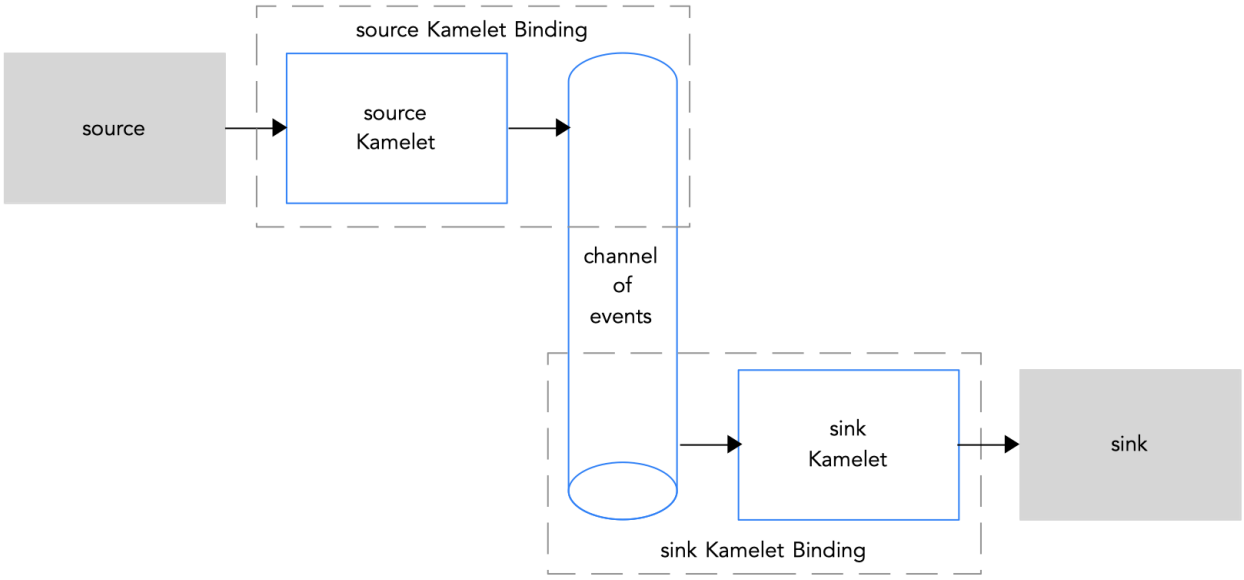


図 1.3: ソースおよびシンク Kamelets のイベントチャネルへの接続

Apache Kafka の stream-processing フレームワークを使用している場合、Kafka トピックへの接続方法に関する詳細は、Kamelets を使用した Kafka への接続 を参照してください。

Knative サーバーレスフレームワークを使用する場合、Knative 宛先 (チャネルまたはブローカー) への接続方法に関する詳細は、Kamelets を使用した Knative への接続 を参照してください。

1.2.6. 明示的な Camel URI への接続

Kamelet が明示的な Camel URI との間でイベントを送受信する Kamelet Binding を作成できます。通常、ソース Kamelet をイベントを受信できる URI にバインドします (つまり、URI を Kamelet Binding のシンクとして指定します)。イベントを受信する Camel URI の例は HTTP または HTTPS エンドポイントです。

また、Mamelet Binding のソースとして URI を指定することも可能ですが、一般的ではありません。イベントを送信する Camel URI の例は、タイマー、メール、または FTP エンドポイントです。

Kamelet を Camel URI に接続するには、Kamelet Binding でのソースおよびシンクコンポーネントの接続 の手順に従い、Kamelet ではなく **sink.uri** フィールドで明示的な Camel URI を指定します。

以下の例では、シンクの URI は架空の URI (https://mycompany.com/event-service) です。

```
apiVersion: camel.apache.org/v1alpha1
```

```
kind: KameletBinding
metadata:
  name: coffee-to-event-service
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    uri: https://mycompany.com/event-service
```

1.3. コネクション内のデータへの操作の適用

Kamelet とイベントチャネルの間に渡すデータで操作を実行する場合は、Kamelet Binding 内の中間ステップとしてアクション Kamelets を使用します。たとえば、アクション Kamelet を使用して、データをシリアライズまたはデシリアライズしたり、データをフィルターしたり、フィールドやメッセージヘッダーを挿入したりできます。

フィールドのフィルタリングや追加などの操作操作は、JSON データ (**Content-Type** ヘッダーが **application/json** に設定されている場合) でのみ動作します。イベントデータが JSON 以外の形式 (Avro または Protocol Buffers など) を使用する場合は、操作アクションおよびその後のシリアライズステップ (例: **protobuf-deserialize-action** または **avro-deserialize-action** Kamelet を参照する) の前にデシリアライズステップ (例: **protobuf-serialize-action** または **avro-serialize-action** Kamelet を参照する) を追加して、データの形式を変換する必要があります。接続のデータ形式を変換する方法は、Data conversion Kamelets を参照してください。

アクション Kamelets には以下が含まれます。

- データフィルタリング Kamelets
- データ変換 Kamelets
- データ変更 Kamelets

1.3.1. Kamelet Binding への操作の追加

アクション Kamelet を実装するには、Kamelet Binding ファイルの **spec** セクションで、ソースセクションとシンクセクションの間に **steps** セクションを追加します。

前提条件

- Kamelet Binding でのソースおよびシンクコンポーネントの接続 で説明されているように、Kamelet Binding を作成している。
- Kamelet Binding に追加するアクション Kamelet とアクション Kamelet に必要なパラメーターを知っている必要があります。
  この手順の例では、**predicate-filter-action** Kamelet のパラメーターは **string** タイプの式で、コーヒーデータをフィルターして "deep" taste intensity を持つコーヒーだけをログに記録するための JSON パス式を提供します。**predicate-filter-action** Kamelet では、Kamelet Binding に Builder トレイト設定プロパティーを設定する必要があります。

この例には、イベントデータフォーマットが JSON であるため、この場合はオプションのデシリアライズおよびシリアライズアクションも含まれます。

手順

1. エディターで **KameletBinding** ファイルを開きます。
たとえば、以下は **coffee-to-log.yaml** ファイルの内容になります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
```

2. **source** セクションの上に **integration** セクションを追加し、(**predicate-filter-action** Kamelet で必要とされるように) 以下の Builder トレイト設定プロパティーを指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
spec:
  integration:
        traits:
          builder:
            configuration:
              properties:
                - "quarkus.arc.unremovable-
types=com.fasterxml.jackson.databind.ObjectMapper"
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
```

3. **source** セクションと **sink** セクションの間に **steps** セクションを追加し、アクション Kamelet を定義します。以下に例を示します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
  spec:
    integration:
          traits:
            builder:
              configuration:
                properties:
                  - "quarkus.arc.unremovable-
types=com.fasterxml.jackson.databind.ObjectMapper"
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  steps:
  - ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: json-deserialize-action
  - ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: predicate-filter-action
    properties:
      expression: "@.intensifier =~ /.*deep/"
  - ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: json-serialize-action
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
```

4. 変更を保存します。

5. 以下のように、**oc apply** コマンドを使用して **KameletBinding** リソースを更新します。
   oc apply -f coffee-to-log.yaml

   Camel K Operator は再生成し、更新された **KameletBinding** リソースに基づいて生成する CamelK インテグレーションを実行します。

6. Kamelet Binding のステータスを表示するには、以下を実行します。
   oc get kameletbindings

7. 対応するインテグレーションのステータスを表示するには、次のコマンドを実行します。
   oc get integrations

8. インテグレーションのログファイルの出力を表示するには、以下を実行します。
kamel logs <integration-name>

   たとえば、インテグレーション名が **coffee-to-log** の場合は、以下のコマンドを使用します。

   kamel logs coffee-to-log

9. インテグレーションを停止するには、Kamelet Binding を削除します。
oc delete kameletbindings/<kameletbinding-name>

   以下に例を示します。

   oc delete kameletbindings/coffee-to-log

1.3.2. アクション kamelets

- 「データフィルタリング Kamelets」
- 「データ変換 Kamelets」
- 「データ変更 Kamelets」

1.3.2.1. データフィルタリング Kamelets

たとえば、機密データの漏えいや不要なネットワーク課金の生成を防ぐためやに、ソースとシンクコンポーネント間で渡されるデータをフィルタリングすることができます。

以下の基準に基づいてデータをフィルターできます。

- **Kafka トピック名**: Topic Name Matches Filter Action Kamelet (**topic-name-matches-filter-action**) を設定して、指定の Java 正規表現に一致する名前を持つ Kafka トピックのイベントをフィルターします。詳細は、特定の Kafka トピックのイベントデータの絞り込み を参照してください。
- **ヘッダーキー**: Header Filter Action Kamelet (**has-header-filter-action**) を設定して、特定のメッセージヘッダーを持つイベントをフィルターします。
- **null 値**: Tombstone Filter Action Kamelet (**is-tombstone-filter-action**) を設定して、null ペイロードを持つイベントであるトゥームストーンイベントをフィルターします。
- **述語**: Predicate Filter Action Kamelet (**predicate-filter-action**) を設定して、指定の JSON パス式に基づいてイベントをフィルターします。**predicate-filter-action** Kamelet では、Kamelet Binding に以下の Builder トレイト 設定プロパティーを設定する必要があります。

```
spec:
  integration:
    traits:
      builder:
        configuration:
          properties:
           - "quarkus.arc.unremovable-types=com.fasterxml.
                jackson.databind.ObjectMapper"
```

注記

データフィルタリング Kamelets は、JSON データ (つまり、Content-Type ヘッダーが application/json に設定されている場合) で追加設定なしで機能します。イベントデータが JSON 以外の形式を使用する場合は、操作アクションおよびその後のシリアライズステップ (例: **protobuf-deserialize-action** または **avro-deserialize-action**) の前にデシリアライズステップ (例: **protobuf-serialize-action** または **avro-serialize-action**) を追加して、データの形式を変換する必要があります。接続のデータ形式を変換する方法は、Data conversion Kamelets を参照してください。

1.3.2.2. データ変換 Kamelets

以下のデータ変換 Kamelets を使用すると、ソースコンポーネントとシンクコンポーネント間で渡すデータの形式をシリアライズおよびデシリアライズできます。データ変換は、イベントデータのペイロード (キーまたはヘッダーではない) に適用されます。

- **Avro**: Apache Hadoop 用のデータのシリアライズおよびデータ交換サービスを提供するオープンソースプロジェクト。
  - Avro デシリアライザーアクション Kamelet (**avro-deserialize-action**)
  - Avro シリアライザーアクション Kamelet (**avro-deserialize-action**)
- **プロトコルバッファー**: 内部で使用する Google が開発した高パフォーマンスのコンパクトなバイナリーワイヤ形式で、内部ネットワークサービスと通信できます。
  - Protobuf Deserialize Action Kamelet (**protobuf-deserialize-action**)
  - Protobuf Serialize Action Kamelet (**protobuf-serialize-action**)
- **JSON** (JavaScript Object Notation): JavaScript プログラミング言語のサブセットをベースとしたデータ変換形式。JSON は、言語にまったく依存しないテキスト形式です。
  - JSON Deserialize Action Kamelet (**json-deserialize-action**)
  - JSON Serialize Action Kamelet (**json-serialize-action**)

注記

Avro および Protobuf のシリアライズ/デシリアライズ Kamelets で、スキーマ (JSON 形式を使用した単一行) を指定する必要があります。JSON のシリアライズ/デシリアライズ Kamelets には、その指定は必要ありません。

1.3.2.3. データ変更 Kamelets

以下のデータ変更 Kamelets を使用すると、ソースコンポーネントとシンクコンポーネント間で渡すデータに対して簡単な操作を実行できます。

- **フィールドの抽出**: **extract-field-action** Kamelet を使用して、データのボディーからフィールドをプルし、データの本文全体を抽出したフィールドに置き換えます。
- **フィールドの抽出**: **hoist-field-action** Kamelet を使用して、データの本文を 1 つのフィールドにラップします。
- **ヘッダーの挿入**: **insert-header-action** Kamelet を使用して、静的データまたはレコードメタデータのいずれかを使用してヘッダーフィールドを追加します。

- **フィールドの挿入**: **insert-field-action** Kamelet を使用して、静的データまたはレコードメタデータのいずれかを使用してフィールド値を追加します。

- mask **フィールドのマスク**: **mask-field-action** Kamelet を使用して、フィールド値をフィールドタイプの有効な null 値 (0、空の文字列など) または特定の置換値 (置換値は空でない文字列または数値である必要があります) に置き換えます。
  たとえば、リレーショナルデータベースからデータをキャプチャーして Kafka に送信し、このデータには保護されている (PCI / PII) 情報が含まれる場合、Kafka クラスターがまだ認定されていないため、保護されている情報をマスクする必要があります。

- **フィールドの置き換え**: **replace-field-action** Kamelet を使用して、フィールドをフィルターまたは名前に変更します。名前を変更するフィールド、無効にするフィールド (exclude)、有効にするフィールド (include) を指定できます。

- **値/キー**:(Kafka の場合) **value-to-key-action** Kamelet を使用して、レコードキーをペイロードのフィールドのサブセットから作成した新しいキーに置き換えます。イベントキーを、データが Kafka に書き込まれる前に、イベント情報に基づく値に設定できます。たとえば、データベーステーブルからレコードを読み取る場合、顧客 ID に基づいて Kafka のレコードをパーティションできます。

1.4. コネクション内でのエラーの処理

イベントデータの送受信時に実行中のインテグレーションで障害が発生した場合に Camel K Operator が実行することを指定するには、任意で以下のエラー処理ポリシーのいずれかを Kamelet Binding に追加します。

- エラーハンドラーなし: インテグレーションで発生する障害を無視します。

- ログエラーハンドラー: ログメッセージを標準出力に送信します。

- デッドレターチャネルエラーハンドラー: 障害のあるイベントを、障害発生イベントで特定のロジックを実行できる別のコンポーネント (サードパーティー URI、キュー、別の Kamelet など) にリダイレクトします。また、デッドレターエンドポイントに送信する前に、複数の回数メッセージエクスチェンジの再配信の試行にも対応しています。

- Bean エラーハンドラー: エラーの処理にカスタム Bean を使用するように指定します。

- Ref エラーハンドラー: エラーの処理に Bean を使用するように指定します。Bean は実行時に Camel レジストリーで利用可能である必要があります。

1.4.1. エラー処理ポリシーの Kamelet Binding への追加

ソースとシンク接続間のイベントデータの送受信時のエラーを処理するには、エラーハンドラーポリシーを Kamelet Binding に追加します。

前提条件

- 使用するエラーハンドラーポリシーのタイプを知っている必要があります。

- 既存の **KameletBinding** YAML ファイルがある。

手順

Kamelet Binding にエラー処理を実装するには、以下を実行します。

1. エディターで **KameletBinding** YAML ファイルを開きます。

2. **sink** 定義の後に、エラーハンドラーセクションを **spec** セクションに追加します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: example-kamelet-binding
spec:
  source:
    ...
  sink:
    ...
  errorHandler: ...
```

たとえば、**coffee-to-log** Kamelet Binding で、ログハンドラーを追加して、エラーがログファイルに送信される最大回数を指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffee-to-log
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
  errorHandler:
    log:
      parameters:
        maximumRedeliveries: 3
```

3. ファイルを保存します。

1.4.2. エラーハンドラー

- 「エラーハンドラーなし」
- 「ログエラーハンドラー」
- 「デッドレターチャネルエラーハンドラー」
- 「Bean エラーハンドラー」
- 「Ref エラーハンドラー」

1.4.2.1. エラーハンドラーなし

インテグレーションで発生した失敗を無視する場合は Kamelet Binding に **errorHandler** セクションが含まれないようにするか、以下の例に示すように **none** に設定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: my-kamelet-binding
spec:
  source:
...
  sink:
...
  errorHandler:
    none:
```

1.4.2.2. ログエラーハンドラー

障害を処理するデフォルトの動作では、ログメッセージを標準出力に送信することです。オプションで、以下の例のように、ログエラーハンドラーを使用して再配信や遅延ポリシーなどの他の動作を指定できます。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: my-kamelet-binding
spec:
  source:
...
  sink:
...
  errorHandler:
    log:
      parameters:
        maximumRedeliveries: 3
        redeliveryDelay: 2000
```

1.4.2.3. デッドレターチャネルエラーハンドラー

デッドレターチャネルを使用すると、以下の例のように、失敗したイベントを、失敗したイベントの処理方法を定義できる他のコンポーネント (サードパーティーの URI、キュー、別の Kamelet など) にリダイレクトすることができます。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: my-kamelet-binding
spec:
  source:
  ...
  sink:
  ...
  errorHandler:
    dead-letter-channel:
      endpoint:
```

```
      ref: 1
        kind: Kamelet
        apiVersion: camel.apache.org/v1alpha1
        name: error-handler
      properties: 2
        message: "ERROR!"
        ...
    parameters: 3
      maximumRedeliveries: 1
```

1. **endpoint** には、**ref** または **uri** を使用できます。Camel K Operator は、**kind**、**apiVersion**、および **name** の値に応じて **ref** を解釈します。任意の Kamelet、Kafka Topic チャネル、または Knative 宛先を使用できます。
2. エンドポイントに属する**プロパティー**(この例では **error-handler** という名前の Kamelet へ)。
3. dead-letter-channel エラーハンドラータイプに属する **Parameters**。

1.4.2.4. Bean エラーハンドラー

Bean エラーハンドラーでは、エラーを処理するカスタム Bean を指定して、Error Handler の機能を拡張できます。**type** には、ErrorHandlerBuilder の完全修飾名 **ErrorHandlerBuilder** を指定します。**properties** では、**type** に仕様した **ErrorHandlerBuilder** が必要とするプロパティーを設定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: my-kamelet-binding
spec:
  source:
...
  sink:
...
  errorHandler:
    bean:
      type: "org.apache.camel.builder.DeadLetterChannelBuilder"
      properties:
        deadLetterUri: log:error
```

1.4.2.5. Ref エラーハンドラー

Ref エラーハンドラーを使用すると、ランタイム時に Camel レジストリーで利用可能になることが予想される Bean を使用できます。以下の例では、**my-custom-builder** は実行時に検索する Bean の名前です。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: my-kamelet-binding
spec:
  source:
    ...
  sink:
```

```
...
errorHandler:
  ref: my-custom-builder
```

関連項目:

- Camel K error handling
- Features support by various Error Handlers

第2章 KAMELETS を使用した KAFKA への接続

Apache Kafka は、耐障害性のあるリアルタイムデータフィードを作成する、オープンソースの分散型 publish/subscribe メッセージングシステムです。Kafka は多数のコンシューマー (外部コネクション) 用にデータを素早く保存およびレプリケートします。

Kafka は、ストリーミングイベントを処理するソリューションの構築に役立ちます。分散されたイベント駆動型のアーキテクチャーでは、イベントをキャプチャーし、通信し、処理するのに役立つバックボーンが必要です。Kafka は、データソースとイベントをアプリケーションに接続する通信バックボーンとして機能します。

Kamelets を使用して、Kafka と外部リソース間の通信を設定できます。Kamelets を使用すると、コードを作成せずに、Kafka stream-processing フレームワークでデータをあるエンドポイントから別のエンドポイントに移動する方法を設定できます。Kamelets は、パラメーター値を指定して設定するルートテンプレートです。

たとえば、Kafka はデータをバイナリー形式で保存します。Kamelets を使用して、外部接続との間で送受信するデータをシリアライズおよびデシリアライズできます。Kamelets を使用すると、スキーマを検証し、データに追加、フィルタリング、マスクなどの変更を加えることができます。Kamelet はエラーを対処および処理できます。

2.1. KAMELETS を使用した KAFKA への接続の概要

Apache Kafka のストリーム処理フレームワークを使用する場合は、Kamelets を使用してサービスおよびアプリケーションを Kafka トピックに接続できます。Kamelet Catalog は、特に Kafka トピックへの接続のために以下の Kamelets を提供します。

- **kafka-sink**: データプロデューサーから Kafka トピックにイベントを移動します。Kamelet Binding では、**kafka-sink** Kamelet をシンクとして指定します。
- **kafka-source**: Kafka トピックからデータコンシューマーにイベントを移動します。Kamelet Binding では、**kafka-source** Kamelet をソースとして指定します。

図 2.1 は、ソースおよびシンク Kamelets を Kafka トピックに接続するフローを示しています。

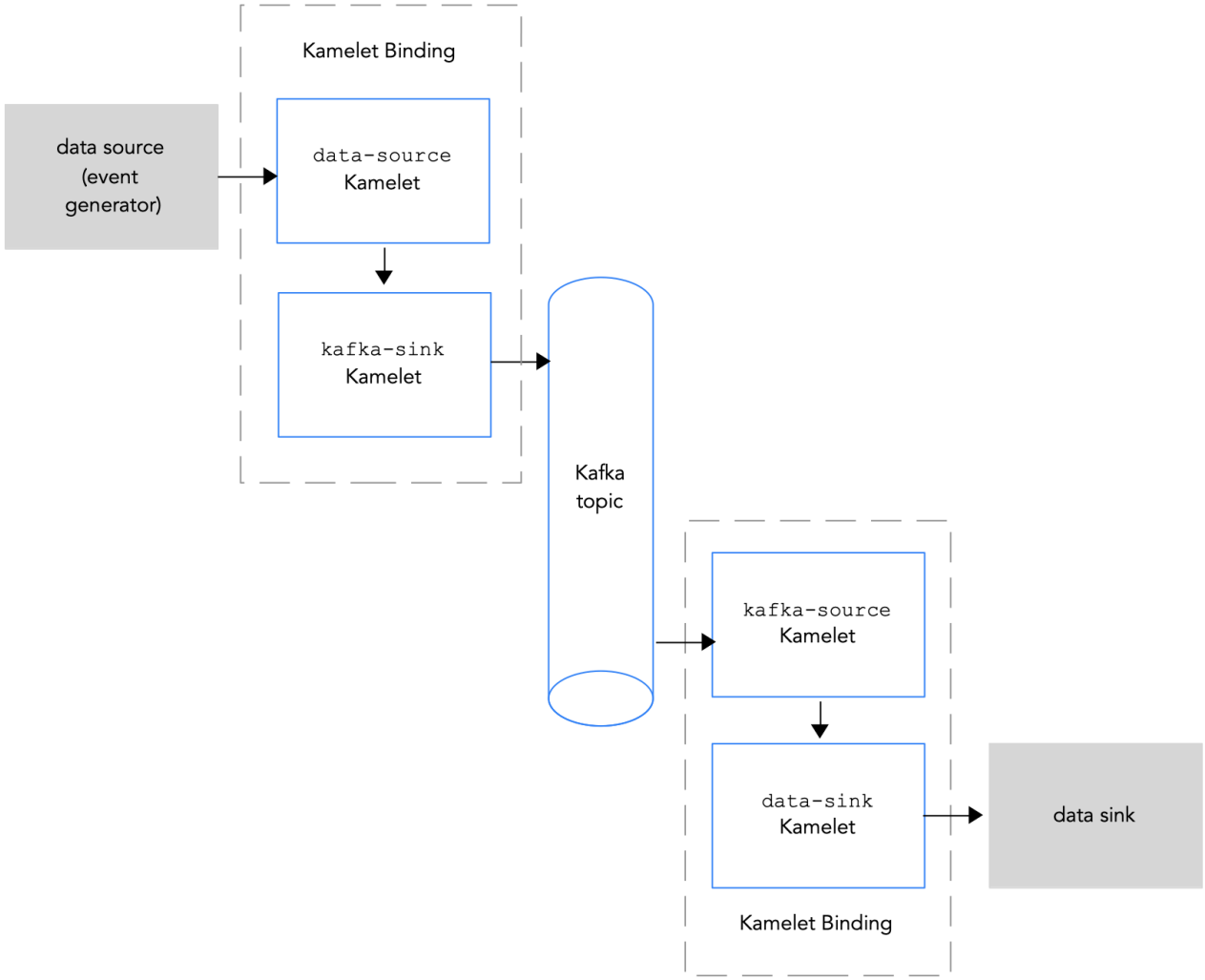


図 2.1: Kamelets と Kafka トピックのデータフロー

以下は、Kamelets および Kamelet Bindings を使用してアプリケーションとサービスを Kafka トピックに接続する基本的な手順の概要です。

1. Kafka を設定します。
    a. 必要な OpenShift Operator をインストールします。
        - OpenShift Streams for Apache Kafka の場合は、Camel K Operator、Camel K CLI、および Red Hat OpenShift Application Services (RHOAS) CLI をインストールします。
        - AMQ Streams の場合は、Camel K および AMQ Streams Operator ならびに Camel K CLI をインストールします。
    b. Kafka インスタンスを作成します。Kafka インスタンスはメッセージブローカーとして動作します。ブローカーにはトピックが含まれ、ストレージとメッセージの渡しをオーケストレーションします。
    c. Kafka トピックを作成します。トピックは、データの保存先を提供します。
    d. Kafka 認証クレデンシャルを取得します。
2. Kafka トピックに接続するサービスまたはアプリケーションを決定します。

3. Kamelet Catalog を表示して、インテグレーションに追加するソースおよびシンクコンポーネントの Kamelets を検索します。また、使用する各 Kamelet に必要な設定パラメーターを決定します。

4. Kamelet Binding を作成します。

   - データソース (データを生成するコンポーネント) を Kafka トピックに接続する Kamelet Binding を作成します (**kafka-sink** Kamelet を使用します)。

   - (**kafka-source** Kamelet を使用して)kafka トピックをデータシンク (データを使用するコンポーネント) に接続する Kamelet Binding を作成します。

5. 必要に応じて、Kamelet Binding 内で 1 つ以上のアクション Kamelets を中間ステップとして追加して、Kafka トピックとデータソースまたはシンク間で渡すデータを操作します。

6. 必要に応じて、Kamelet Binding 内でエラーを処理する方法を定義します。

7. Kamelet Binding をリソースとしてプロジェクトに適用します。
   Camel K Operator は、Kamelet Binding ごとに個別の Camel K インテグレーションを生成します。

2.2. KAFKA の設定

Kafka を設定するには、必要な OpenShift Operator のインストール、Kafka インスタンスの作成、Kafka トピックの作成が必要になります。

以下の Red Hat 製品のいずれかを使用して Kafka を設定します。

- **Red Hat Advanced Message Queuing (AMQ) ストリーム**: 自己管理の Apache Kafka オファリング。AMQ Streams はオープンソースの Strimzi をベースとしており、Red Hat Integration の一部として組み込まれています。AMQ Streams は、パブリッシュ/サブスクライブメッセージングブローカーが含まれる Apache Kafka をベースとした分散型でスケーラブルなストリーミングプラットフォームです。Kafka Connect は、Kafka ベースのシステムを外部システムと統合するフレームワークを提供します。Kafka Connect を使用すると、外部システムと Kafka ブローカーとの間で双方向にデータをストリーミングするように ソースおよびシンクコネクターを設定できます。

- **Red Hat OpenShift Streams for Apache Kafka**- Apache Kafka の実行プロセスを簡素化するマネージドクラウドサービスです。これにより、新しいクラウドネイティブアプリケーションを構築、デプロイ、およびスケーリングする際、または既存システムを現代化する際に、効率的な開発者エクスペリエンスが提供されます。

2.2.1. AMQ Streams を使用した Kafka の設定

AMQ Streams は、OpenShift クラスターで Apache Kafka を実行するプロセスを簡素化します。

2.2.1.1. AMQ Streams の OpenShift クラスターの準備

Camel K または Kamelets および Red Hat AMQ Streams を使用するには、以下の Operator およびツールをインストールする必要があります。

- **Red Hat Integration - AMQ Streams**Operator: OpenShift Cluster と AMQ Streams for Apache Kafka インスタンスの間の通信を管理します。

- **Red Hat Integration - Camel K**Operator: Camel K (OpenShift のクラウドでネイティブに実行される軽量なインテグレーションフレームワーク) をインストールし、管理します。

- **Camel K CLI** ツール: すべての Camel K 機能にアクセスできます。

前提条件

- Apache Kafka の概念を理解している。
- 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、OpenShift および Camel K CLI をローカルシステムにインストールできること。
- 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。

手順

AMQ Streams を使用して Kafka を設定するには、以下を行います。

1. OpenShift クラスターの Web コンソールにログインします。
2. インテグレーションを作成する予定のプロジェクト (例: **my-camel-k-kafka**) を作成または開きます。
3. Camel K のインストール の説明に従って、Camel K Operator および Camel K CLI をインストールします。
4. AMQ Streams Operator をインストールします。
   a. 任意のプロジェクトから **Operators** > **OperatorHub** を選択します。
   b. **Filter by Keyword** フィールドに **AMQ Streams** を入力します。
   c. **Red Hat Integration - AMQ Streams**カードをクリックしてから **Install** をクリックします。
   Install Operator ページが開きます。
   d. デフォルトを受け入れ、**Install** をクリックします。
5. **Operators** > **Installed Operators** を選択し、Camel K および AMQ Streams Operator がインストールされていることを確認します。

次のステップ

AMQ Streams を使用した Kafka トピックの設定

2.2.1.2. AMQ Streams を使用した Kafka トピックの設定

Kafka トピックは、Kafka インスタンスのデータの保存先を提供します。データを送信する前に、Kafka トピックを設定する必要があります。

前提条件

- OpenShift クラスターにアクセスできる。
- OpenShift クラスターの準備 の説明どおりに、**Red Hat Integration - Camel K**Operator および **Red Hat Integration - AMQ Streams**Operator がインストールされている。
- OpenShift CLI (**oc**) および Camel K CLI (**kamel**) をインストールしている。

手順

AMQ Streams を使用して Kafka トピックを設定するには、以下を行います。

1. OpenShift クラスターの Web コンソールにログインします。
2. **Projects** を選択してから、**Red Hat Integration - AMQ Streams**Operator をインストールしたプロジェクトをクリックします。たとえば、**my-camel-k-kafka** プロジェクトをクリックします。
3. **Operators** > **Installed Operators** の順に選択し、**Red Hat Integration - AMQ Streams**をクリックします。
4. Kafka クラスターを作成します。
   a. **Kafka** で、**Create instance** をクリックします。
   b. **kafka-test** などクラスターの名前を入力します。
   c. その他のデフォルトを受け入れ、**Create** をクリックします。
   Kafka インスタンスを作成するプロセスの完了に数分かかる場合があります。

   ステータスが ready になったら、次のステップに進みます。
5. Kafka トピックを作成します。
   a. **Operators** > **Installed Operators** の順に選択し、**Red Hat Integration - AMQ Streams**をクリックします。
   b. **Kafka Topic** で **Create Kafka Topic** をクリックします。
   c. トピックの名前を入力します (例: **test-topic**)。
   d. その他のデフォルトを受け入れ、**Create** をクリックします。

2.2.2. OpenShift Streams を使用した Kafka の設定

Red Hat OpenShift Streams for Apache Kafka は、Apache Kafka の実行プロセスを簡素化する管理クラウドサービスです。

OpenShift Streams for Apache Kafka を使用するには、Red Hat アカウントにログインする必要があります。

関連項目

- Product documentation for Red Hat OpenShift Streams for Apache Kafka

2.2.2.1. OpenShift Streams の OpenShift クラスターの準備

Red Hat OpenShift Streams for Apache Kafka 管理クラウドサービスを使用するには、以下の Operator およびツールをインストールする必要があります。

- **OpenShift Application Services (RHOAS) CLI**: ターミナルからアプリケーションサービスを管理できます。
- **Red Hat Integration - Camel K**Operator は、Camel K (OpenShift のクラウドでネイティブに実行される軽量なインテグレーションフレームワーク) をインストールし、管理します。

- **Camel K CLI ツール**: すべての Camel K 機能にアクセスできます。

前提条件

- Apache Kafka の概念を理解している。
- 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、OpenShift および Apache Camel K CLI をローカルシステムにインストールできること。
- 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。

手順

1. クラスター管理者アカウントで OpenShift Web コンソールにログインします。
2. Camel K または Kamelets アプリケーションの OpenShift プロジェクトを作成します。
   a. **Home** > **Projects** を選択します。
   b. **Create Project** をクリックします。
   c. プロジェクトの名前 (例: **my-camel-k-kafka**) を入力し、続いて **Create** をクリックします。
3. Getting started with the rhoas CLI の説明に従って、RHOAS CLI をダウンロードし、インストールします。
4. Camel K のインストール の説明に従って、Camel K Operator および Camel K CLI をインストールします。
5. **Red Hat Integration - Camel K**Operator がインストールされていることを確認するには、**Operators** > **Installed Operators** の順にクリックします。

次のステップ

RHOAS を使用した Kafka トピックの設定

2.2.2.2. RHOAS を使用した Kafka トピックの設定

Kafka は **トピック** に関するメッセージを整理します。各トピックには名前があります。アプリケーションは、トピックにメッセージを送信し、トピックからメッセージを取得します。Kafka トピックは、Kafka インスタンスのデータの保存先を提供します。データを送信する前に、Kafka トピックを設定する必要があります。

前提条件

- 適切なアクセスレベルで OpenShift 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、OpenShift および Camel K CLI をローカルシステムにインストールできること。
- OpenShift クラスターの準備 の手順に従って、OpenShift CLI (**oc**)、Camel K CLI (**kamel**)、および RHOAS CLI (**rhoas**) ツールをインストールしている。
- OpenShift クラスターの準備 の説明どおりに、**Red Hat Integration - Camel K**Operator がインストールされている。

- Red Hat Cloud サイト にログインしている。

手順

Red Hat OpenShift Streams for Apache Kafka を使用して Kafka トピックを設定するには、以下を行います。

1. コマンドラインから OpenShift クラスターにログインします。
2. プロジェクトを開きます。以下に例を示します。
   oc project my-camel-k-kafka
3. Camel K Operator がプロジェクトにインストールされていることを確認します。
   oc get csv

   結果には、Red Hat Camel K Operator が表示され、それが **Succeeded** フェーズにあることを示します。
4. Kafka インスタンスを準備し、RHOAS に接続します。
   a. 以下のコマンドを使用して RHOAS CLI にログインします。
      rhoas login
   b. **kafka-test** などの kafka インスタンスを作成します。
      rhoas kafka create kafka-test

      Kafka インスタンスを作成するプロセスの完了に数分かかる場合があります。
5. Kafka インスタンスのステータスを確認するには、以下を実行します。
   rhoas status

   Web コンソールでステータスを表示することもできます。

   https://cloud.redhat.com/application-services/streams/kafkas/

   ステータスが **ready** になったら、次のステップに進みます。
6. 新しい Kafka トピックを作成します。
   rhoas kafka topic create --name test-topic
7. Kafka インスタンス (クラスター) を Openshift Application Services インスタンスに接続します。
   rhoas cluster connect
8. クレデンシャルトークンを取得するスクリプトの手順に従います。
   以下のような出力が表示されるはずです。

```
Token Secret "rh-cloud-services-accesstoken-cli" created successfully
Service Account Secret "rh-cloud-services-service-account" created successfully
KafkaConnection resource "kafka-test" has been created
KafkaConnection successfully installed on your cluster.
```

次のステップ

- Kafka クレデンシャルの取得

2.2.2.3. Kafka クレデンシャルの取得

アプリケーションまたはサービスを Kafka インスタンスに接続するには、まず以下の Kafka クレデンシャルを取得する必要があります。

- ブートストラップ URL を取得します。
- クレデンシャル (ユーザー名とパスワード) を使用してサービスアカウントを作成します。

OpenShift Streams では、認証プロトコルは SASL_SSL です。

前提条件

- Kafka インスタンスを作成し、ステータスが ready である。
- Kafka トピックを作成している。

手順

1. Kafka ブローカーの URL (ブートストラップ URL) を取得します。
rhoas status

   コマンドは、以下のような出力を返します。

```
Kafka
----------------------------------------------------------------
ID:                 1ptdfZRHmLKwqW6A3YKM2MawgDh
Name:                 my-kafka
Status:               ready
Bootstrap URL:        my-kafka--ptdfzrhmlkwqw-a-ykm-mawgdh.kafka.devshift.org:443
```

2. ユーザー名とパスワードを取得するには、以下の構文を使用してサービスアカウントを作成します。
rhoas service-account create --name "<account-name>" --file-format json

注記

サービスアカウントの作成時に、ファイル形式および場所を選択してクレデンシャルを保存できます。詳細は、**rhoas service-account create --help** コマンドを参照してください。

   以下に例を示します。

   rhoas service-account create --name "my-service-acct" --file-format json

   サービスアカウントが作成され、JSON ファイルに保存されます。

3. サービスアカウントの認証情報を確認するには、**credentials.json** ファイルを表示します。
cat credentials.json

   コマンドは、以下のような出力を返します。

```
{"clientID":"srvc-acct-eb575691-b94a-41f1-ab97-50ade0cd1094", "password":"facf3df1-3c8d-4253-aa87-8c95ca5e1225"}
```

4. Kakfa トピックとの間でメッセージを送受信する権限を付与します。以下のコマンドを使用します。ここで、**clientID** は (ステップ 3 からの)**credentials.json** ファイルで指定される値に置き換えます。

```
rhoas kafka acl grant-access --producer --consumer --service-account $CLIENT_ID --topic test-topic --group all
```

以下に例を示します。

```
rhoas kafka acl grant-access --producer --consumer --service-account srvc-acct-eb575691-b94a-41f1-ab97-50ade0cd1094 --topic test-topic --group all
```

2.3. KAMELET BINDING でのデータソースの KAFKA トピックへの接続

データソースを Kafka トピックに接続するには、**図 2.2**で説明されているように Kamelet Binding を作成します。

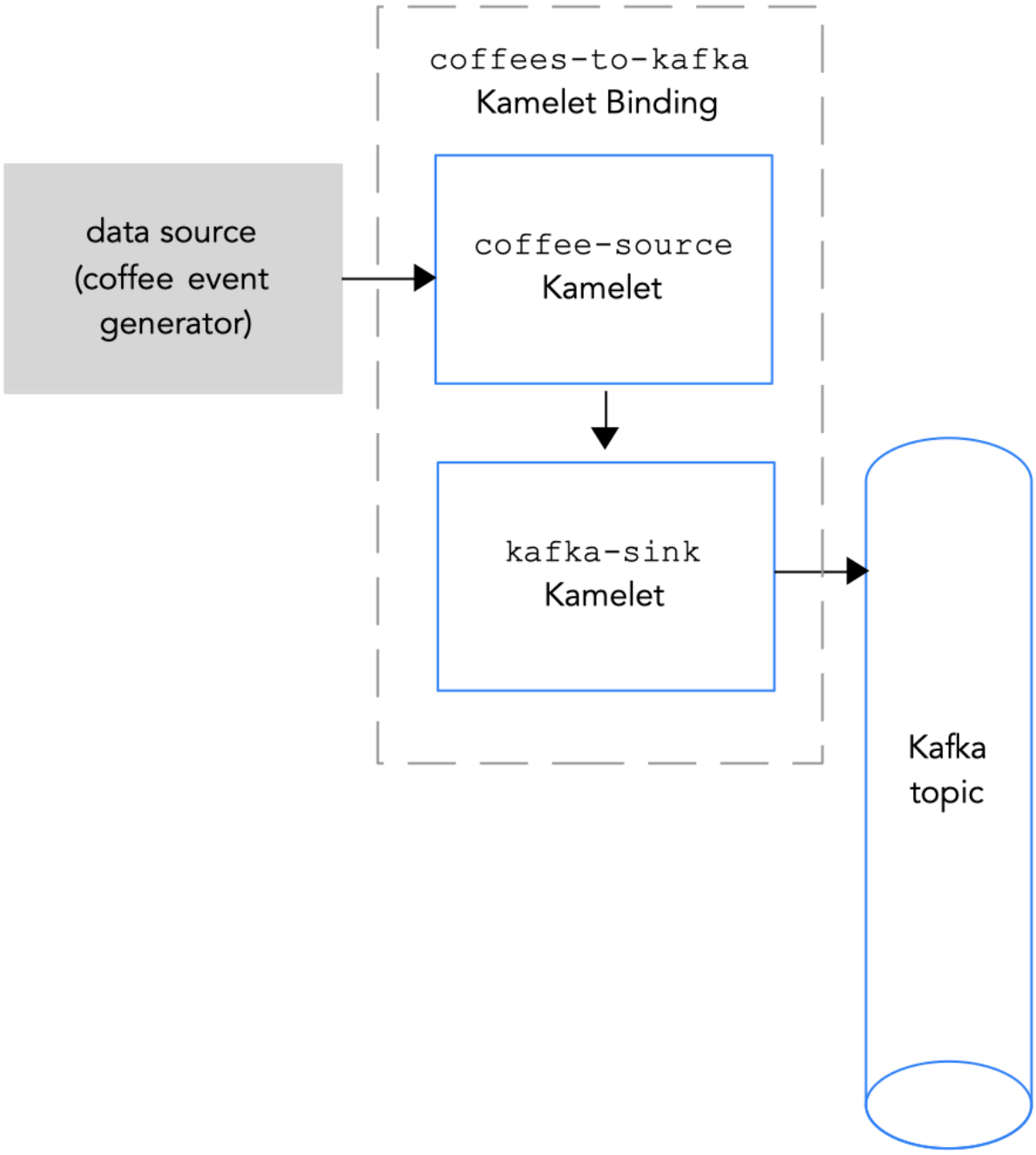


図 2.2 データソースの Kafka トピックへの接続

前提条件

- イベントの送信先となる Kafka トピックの名前を知っておく必要があります。
  この手順の例では、イベントを受信するために **test-topic** を使用します。
- Kafka インスタンスの以下のパラメーターの値を知っている必要があります。
  - **bootstrapServers**: Kafka Broker URL のコンマ区切りリスト
  - **password**: Kafka に対して認証を行うためのパスワード。OpenShift Streams では、これは **credentials.json** ファイル **password** です。AMQ Streams の認証されていない kafka インスタンスでは、任意の空でない文字列を指定できます。

- **user**: Kafka に対して認証するユーザー名。OpenShift Streams では、これは **credentials.json** ファイル **clientID** です。AMQ Streams の認証されていない kafka インスタンスでは、任意の空でない文字列を指定できます。
  OpenShift Streams の使用時にこれらの値を取得する方法は、Kafka クレデンシャルの取得 を参照してください。

- **securityProtocol**: Kafka ブローカーと通信するためのセキュリティープロトコルを知っている必要があります。OpenShift Streams の Kafka クラスターでは、**SASL_SSL** (デフォルト) です。AMQ Streams 上の Kafka クラスターでは、**PLAINTEXT** になります。

- Camel K インテグレーションに追加する Kamelets と必要なインスタンスパラメーターを知っている必要があります。
  この手順の Kamelets の例は次のとおりです。

  - **coffee-source** Kamelet: 各イベントを送信する頻度を指定するオプションのパラメーター **period** があります。ソース Kamelet の例 のコードを、**coffee-source.kamelet.yaml** ファイルという名前のファイルにコピーしてから、以下のコマンドを実行して、これをリソースとして namespace に追加できます。
    oc apply -f coffee-source.kamelet.yaml

  - Kamelet Catalog で提供される **kafka-sink** Kamelet。Kafka トピックがこのバインディングでデータ (データコンシューマー) を受信しているため、**kafka-sink** Kamelet を使用します。

手順

データソースを Kafka トピックに接続するには、Kamelet Binding を作成します。

1. 任意のエディターで、以下の基本構造で YAML ファイルを作成します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name:
spec:
  source:
  sink:
```

2. Kamelet Binding の名前を追加します。この例では、バインディングが **coffee-source** Kamelet を **kafka-sink** Kamelet に接続するため、名前は **coffees-to-kafka** になります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffees-to-kafka
spec:
  source:
  sink:
```

3. Kamelet Binding のソースの場合は、データソース Kamelet を指定し (たとえば、**coffee-source** Kamelet はコーヒーに関するデータが含まれるイベントを生成します)、Kamelet のパラメーターを設定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
```

```
  name: coffees-to-kafka
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
```

4. Kamelet Binding のシンクの場合は、**kafka-sink** Kamelet およびその必要なプロパティーを指定します。
   たとえば、Kafka クラスターが OpenShift Streams 上にある場合:

   - **user** プロパティーには、**clientID** 指定します (例: **srvc-acct-eb575691-b94a-41f1-ab97-50ade0cd1094**)
   - **password** プロパティーには、 **password** 指定します (例: **facf3df1-3c8d-4253-aa87-8c95ca5e1225**)
   - **securityProtocol** プロパティーを設定する必要はありません。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffees-to-kafka
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: kafka-sink
    properties:
      bootstrapServers: "my-kafka--ptdfzrhmlkwqw-a-ykm-mawgdh.kafka.devshift.org:443"
      password: "facf3df1-3c8d-4253-aa87-8c95ca5e1225"
      topic: "test-topic"
      user: "srvc-acct-eb575691-b94a-41f1-ab97-50ade0cd1094"
```

   別の例として、Kafka クラスターが AMQ Streams 上にある場合、**securityProtocol** プロパティーを **“PLAINTEXT”** に設定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffees-to-kafka
spec:
  source:
```

```
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: kafka-sink
    properties:
      bootstrapServers: "broker.url:9092"
      password: "testpassword"
      topic: "test-topic"
      user: "testuser"
      securityProtocol: "PLAINTEXT"
```

5. YAML ファイルを保存します (例:**coffees-to-kafka.yaml**)。
6. OpenShift プロジェクトにログインします。
7. Kamelet Binding をリソースとして OpenShift namespace に追加します。
   oc apply -f <kamelet binding filename>

   以下に例を示します。

   oc apply -f coffees-to-kafka.yaml

   Camel K Operator は、**KameletBinding** リソースを使用して Camel K インテグレーションを生成し、実行します。ビルドに数分かかる場合があります。
8. **KameletBinding** リソースのステータスを表示するには、次のコマンドを実行します。
   oc get kameletbindings
9. インテグレーションの状態を表示するには、以下を実行します。
   oc get integrations
10. インテグレーションのログを表示するには、以下を実行します。
    kamel logs <integration> -n <project>

    以下に例を示します。

    kamel logs coffees-to-kafka -n my-camel-k-kafka

関連項目

- Kafka 接続内でのデータへの操作の適用
- コネクション内でのエラーの処理
- Kamelet Binding での Kafka トピックのデータシンクへの接続

2.4. KAMELET BINDING での KAFKA トピックのデータシンクへの接続

Kafka トピックをデータシンクに接続するには、**図 2.3** にあるように Kamelet Binding を作成します。

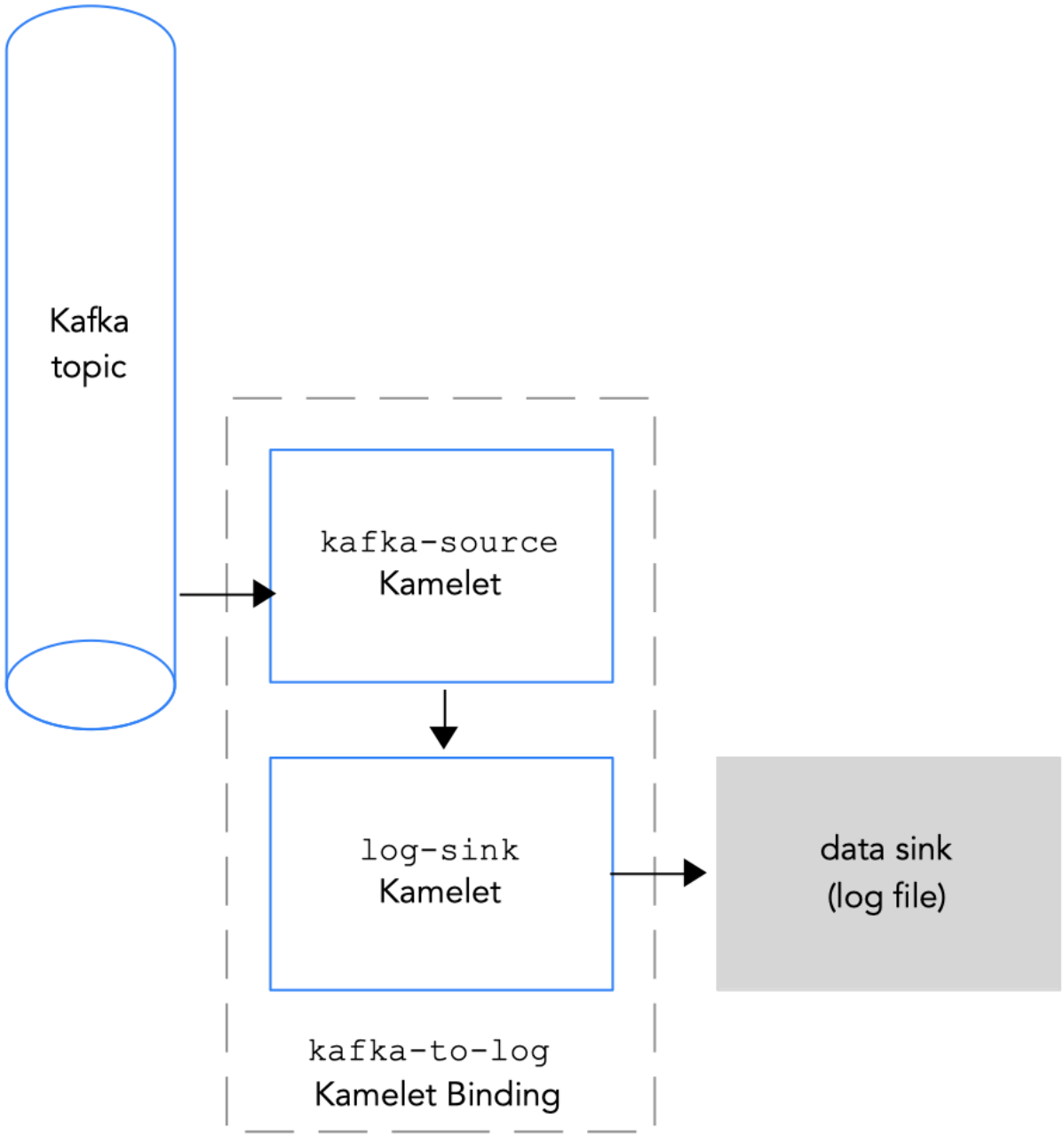


図 2.3 Kafka トピックのデータシンクへの接続

前提条件

- イベントの送信元となる Kafka トピックの名前を知っておく必要があります。この手順の例では、イベントを送信するために **test-topic** を使用します。これは、Kamelet Binding でのデータソースの Kafka トピックへの接続 でコーヒーソースからイベントを受信するのに使用したトピックと同じです。
- Kafka インスタンスの以下のパラメーターの値を知っている必要があります。
  - **bootstrapServers**: Kafka Broker URL のコンマ区切りリスト
  - **password**: Kafka に対して認証を行うためのパスワード。
  - **user**: Kafka に対して認証するユーザー名。
    OpenShift Streams の使用時にこれらの値を取得する方法は、Kafka クレデンシャルの取得 を参照してください。

- Kafka ブローカーとの通信のセキュリティープロトコルを把握している。OpenShift Streams の Kafka クラスターでは、**SASL_SSL** (デフォルト) です。AMQ Streams 上の Kafka クラスターでは、**PLAINTEXT** になります。

- Camel K インテグレーションに追加する Kamelets と必要なインスタンスパラメーターを知っている必要があります。この手順の Kamelets の例は、Katmelet Catalog に記載されています。

  - **kafka-source** Kamelet: このバインディングでは Kafka トピックがデータを送信するため (データプロデューサー)、**kafka-source** Kamelet を使用します。必須パラメーターの値の例は次のとおりです。

    - bootstrapServers - **"broker.url:9092"**
    - password - **"testpassword"**
    - user - **"testuser"**
    - topic - **"test-topic"**
    - **securityProtocol**: OpenShift Streams の Kafka クラスターの場合には、 **SASL_SSL** がデフォルト値であるため、このパラメーターを設定する必要はありません。AMQ ストリームの Kafka クラスターの場合は、このパラメーターの値は **"PLAINTEXT"** です。

  - **log-sink** Kamelet: **log-sink** を使用して、**kafka-source** Kamelet から受信するデータをログに記録します。必要に応じて、**showStreams** パラメーターを指定して、データのメッセージボディーを表示します。**log-sink** Kamelet は、デバッグに役立ちます。

手順

Kafka トピックをデータシンクに接続するには、Kamelet Binding を作成します。

1. 任意のエディターで、以下の基本構造で YAML ファイルを作成します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name:
spec:
  source:
  sink:
```

2. Kamelet Binding の名前を追加します。この例では、バインディングが **kafka-source** Kamelet を **log-sink** Kamelet に接続するため、名前は **kafka-to-log** になります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: kafka-to-log
spec:
  source:
  sink:
```

3. Kamelet Binding のソースの場合は、**kafka-source** Kamelet を指定し、そのパラメーターを設定します。
   たとえば、Kafka クラスターが OpenShift Streams にある場合 (**securityProtocol** パラメーターを設定する必要はありません)。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: kafka-to-log
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: kafka-source
    properties:
      bootstrapServers: "broker.url:9092"
      password: "testpassword"
      topic: "test-topic"
      user: "testuser"
  sink:
```

たとえば、Kafka クラスターが AMQ Streams にある場合は、**securityProtocol** パラメーターを **"PLAINTEXT"** に設定する必要があります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: kafka-to-log
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: kafka-source
    properties:
      bootstrapServers: "broker.url:9092"
      password: "testpassword"
      topic: "test-topic"
      user: "testuser"
      securityProtocol: "PLAINTEXT"
  sink:
```

4. Kamelet Binding のシンクでは、データコンシューマー Kamelet (例: **log-sink** Kamelet) を指定し、Kamelet のパラメーターを設定します。以下に例を示します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: kafka-to-log
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: kafka-source
    properties:
      bootstrapServers: "broker.url:9092"
      password: "testpassword"
      topic: "test-topic"
```

```
    user: "testuser"
    securityProtocol: "PLAINTEXT" // only for AMQ streams
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
    properties:
      showStreams: true
```

5. YAML ファイルを保存します (例: **kafka-to-log.yaml**)。
6. OpenShift プロジェクトにログインします。
7. Kamelet Binding をリソースとして OpenShift namespace に追加します。
   oc apply -f <kamelet binding filename>

   以下に例を示します。

   oc apply -f kafka-to-log.yaml

   Camel K Operator は、**KameletBinding** リソースを使用して Camel K インテグレーションを生成し、実行します。ビルドに数分かかる場合があります。
8. **KameletBinding** リソースのステータスを表示するには、次のコマンドを実行します。
   oc get kameletbindings
9. インテグレーションの状態を表示するには、以下を実行します。
   oc get integrations
10. インテグレーションのログを表示するには、以下を実行します。
    kamel logs <integration> -n <project>

    以下に例を示します。

    kamel logs kafka-to-log -n my-camel-k-kafka

    この出力では、以下の例のようにコーヒーイベントが表示されるはずです。

    ```
    INFO  [log-sink-E80C5C904418150-0000000000000001] (Camel (camel-1) thread #0 - timer://tick) {"id":7259,"uid":"a4ecb7c2-05b8-4a49-b0d2-d1e8db5bc5e2","blend_name":"Postmodern Symphony","origin":"Huila, Colombia","variety":"Kona","notes":"delicate, chewy, black currant, red apple, star fruit","intensifier":"balanced"}
    ```
11. 実行中のインテグレーションを停止するには、関連付けられた Kamelet Binding リソースを削除します。
    oc delete kameletbindings/<kameletbinding-name>

    以下に例を示します。

    oc delete kameletbindings/kafka-to-log

関連項目

- Kafka 接続内でのデータへの操作の適用

- エラー処理ポリシーの Kamelet Binding への追加

2.5. KAFKA 接続内でのデータへの操作の適用

Kamelet と Kafka トピック間で渡されるデータで操作を実行する場合は、Kamelet Binding 内の中間ステップとして、アクション Kamelets を使用します。

- コネクション内のデータへの操作の適用
- 異なる宛先トピックへのイベントデータのルーティング
- 特定の Kafka トピックのイベントデータの絞り込み

2.5.1. 異なる宛先トピックへのイベントデータのルーティング

Kafka インスタンスへのコネクションを設定する場合、イベントが異なる Kafka トピックにルーティングされるように、オプションでイベントデータからのトピック情報を変換できます。以下の変換アクション Kamelets のいずれかを使用します。

- **Regex Router**: 正規表現と代替文字列を使用してメッセージのトピックを変更します。たとえば、トピック接頭辞を削除する場合は、接頭辞を追加するか、トピック名の一部を削除します。Regex Router Action Kamelet (**regex-router-action**) を設定します。
- **TimeStamp**: 元のトピックとメッセージのタイムスタンプに基づいてメッセージのトピックを変更します。たとえば、タイムスタンプに基づいて異なるテーブルまたはインデックスに書き込む必要があるシンクを使用する場合などです。たとえば、Kafka から Elasticsearch にイベントを書きみ、各イベントはイベント自体の情報に基づいて異なるインデックスに移動する必要がある場合。Timestamp Router Action Kamelet(**timestamp-router-action**) を設定します。
- **Message TimeStamp**: 元のトピック値とメッセージ値フィールドからのタイムスタンプフィールドに基づいてメッセージのトピックを変更します。Message Timestamp Router Action Kamelet(**message-timestamp-router-action**) を設定します。
- **述語**: Predicate Filter Action Kamelet (**predicate-filter-action**) を設定して、指定の JSON パス式に基づいてイベントをフィルターします。

前提条件

- Kamelet Binding でのデータソースの Kafka トピックへの接続 で説明されているように、シンクが **kafka-sink** Kamelet である Kamelet Binding を作成している。
- Kamelet Binding に追加する変換のタイプを知っている必要があります。

手順

宛先トピックを変更するには、Kamelet Binding 内の中間ステップとして変更アクション Kamelets の 1 つを使用します。

アクション Kamelet を Kamelet Binding に追加する方法は、Kamelet Binding への操作の追加 を参照してください。

2.5.2. 特定の Kafka トピックのイベントデータの絞り込み

多くの異なる Kafka トピックにレコードを生成するソース Kamelet を使用し、レコードを 1 つの Kafka トピックに絞り込む場合は、Kamelet Binding の中間ステップとして **topic-name-matches-filter-action** Kamelet を追加します。

前提条件

- YAML ファイルに Kamelet Binding を作成している。
- イベンデータを絞り込む Kafka トピックの名前を知っておく必要があります。

手順

1. Kamelet Binding を編集して、ソースとシンク Kamelets の間の中間ステップとして **topic-name-matches-filter-action** Kamelet を追加します。
通常、ソース Kamelet として **kafka-source** Kamelet を使用し、トピックを必要な **topic** パラメーターの値として指定します。

   以下の Kamelet Binding の例では、**kafka-source** Kamelet は **test-topic、test-topic-2、および test-topic-3** Kafka トピックを指定し、 **topic-name-matches-filter-action** Kamelet は、 **topic-test** トピックからイベントデータをフィルターするように指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: kafka-to-log-by-topic
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: kafka-source
    properties:
      bootstrapServers: "broker.url:9092"
      password: "testpassword"
      topic: "test-topic, test-topic-2, test-topic-3"
      user: "testuser"
      securityProtocol: "PLAINTEXT" // only for AMQ streams
steps:
  - ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: topic-name-matches-filter-action
    properties:
      regex:  "test-topic"
sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
    properties:
        showStreams: true
```

   kafka-source Kamelet 以外のソース Kamelet からのトピックをフィルタリングする場合は、Kafka トピック情報を指定する必要があります。以下の例のように、**insert-header-action** Kamelet を使用して、Kamelet Binding の **topic-name-matches-filter-action** ステップの前に Kafka トピックフィールドを中間ステップとして追加できます。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
```

```
metadata:
  name: coffee-to-log-by-topic
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
steps:
  - ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: insert-header-action
    properties:
      name:  "KAFKA.topic"
      value:  "test-topic"
  - ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: topic-name-matches-filter-action
    properties:
      regex:  "test-topic"
sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: log-sink
    properties:
      showStreams: true
```

2. Kamelet Binding YAML ファイルを保存します。

第3章 KAMELETS を使用した KNATIVE への接続

Kamelets を Knative 宛先 (チャネルまたはブローカー) に接続することができます。Red Hat OpenShift Serverless はオープンソースの Knative プロジェクト をベースとし、エンタープライズレベルのサーバーレスプラットフォームを有効にすることで、ハイブリッドおよびマルチクラウド環境における移植性と一貫性をもたらします。OpenShift Serverless には、Knative Eventing および Knative Serving コンポーネントのサポートが含まれます。

Red Hat OpenShift Serverless、Knative Eventing、および Knative Serving では、サーバーレスアプリケーションと共に イベント駆動型のアーキテクチャー を使用し、パブリッシュサブスクライブまたはイベントストリーミングモデルを使用してイベントプロデューサーとコンシューマー間の関係を切り離すことができます。Knative Eventing は、標準の HTTP POST リクエストを使用してイベントプロデューサーとコンシューマー間でイベントを送受信します。これらのイベントは CloudEvents 仕様 に準拠しており、すべてのプログラミング言語でのイベントの作成、解析、および送受信を可能にします。

Kamelets を使用して CloudEvents を Knative に送信し、Knative からイベントコンシューマーに送信できます。Kamelets はメッセージを CloudEvents に変換し、それらを使用して CloudEvents 内のデータの事前処理および後処理を適用できます。

3.1. KAMELETS を使用した KNATIVE への接続の概要

Knative stream-processing フレームワークを使用する場合は、Kamelets を使用してサービスおよびアプリケーションを Knative 宛先 (チャネルまたはブローカー) に接続することができます。

図 3.1 は、ソースとシンク Kamelets を Knative 宛先に接続するフローを示します。

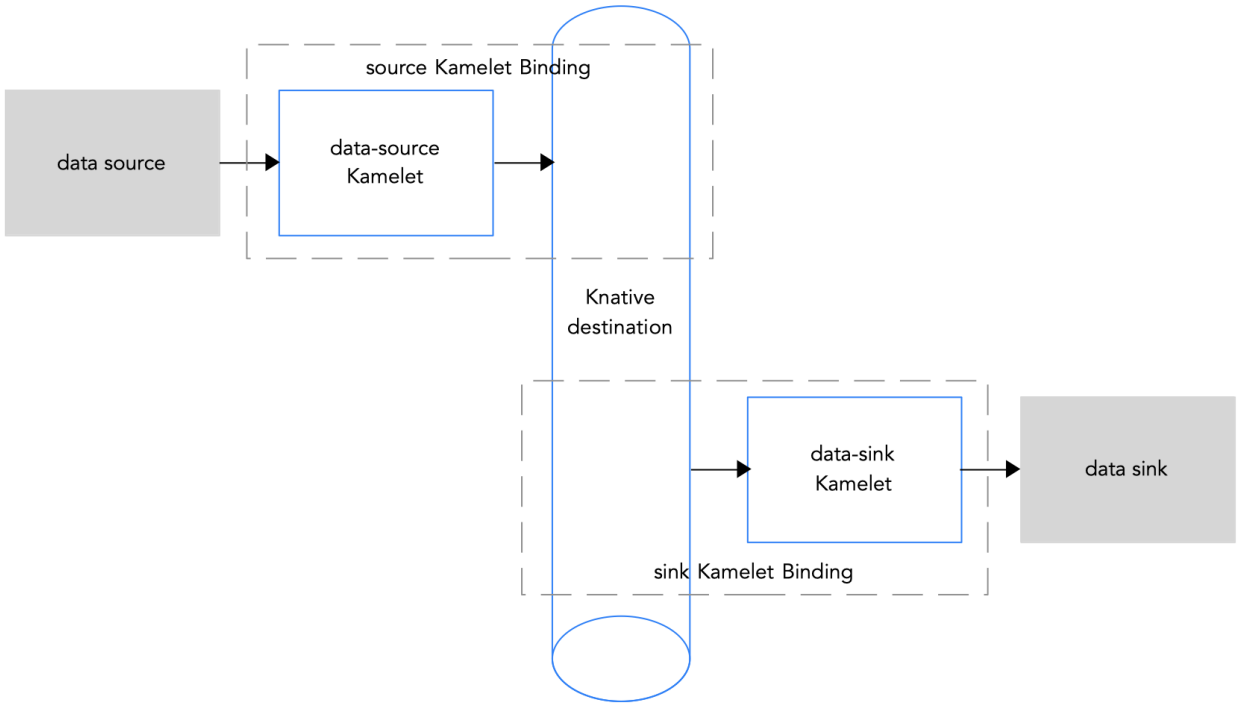


図 3.1: Kamelets と Knative チャネルによるデータフロー

以下は、Kamelets および Kamelet Bindings を使用してアプリケーションやサービスを Knative 宛先に接続するための基本的な手順の概要です。

1. Knative を設定します。

a. Camel K および OpenShift Serverless Operator をインストールして、OpenShift クラスターを準備します。

b. 必要な Knative Serving および Eventing コンポーネントをインストールします。

c. Knative チャネルまたはブローカーを作成します。

2. Knative チャネルまたはブローカーに接続するサービスまたはアプリケーションを決定します。

3. Kamelet Catalog を表示して、インテグレーションに追加するソースおよびシンクコンポーネントの Kamelets を検索します。また、使用する各 Kamelet に必要な設定パラメーターを決定します。

4. Kamelet Binding を作成します。

- ソース Kamelet を Knative チャネル (またはブローカー) に接続する Kamelet Binding を作成します。

- Knative チャネル (またはブローカー) をシンク Kamelet に接続する Kamelet Binding を作成します。

5. オプションとして、Kamelet Binding 内の中間ステップとして 1 つまたは複数のアクション Kamelets を追加して、Knative チャネル (またはブローカー) およびデータソース/シンク間で渡されるデータを操作します。

6. 必要に応じて、Kamelet Binding 内でエラーを処理する方法を定義します。

7. Kamelet Binding をリソースとしてプロジェクトに適用します。

Camel K Operator は、Kamelet Binding ごとに個別の Camel インテグレーションを生成します。

Kamelet Binding を Knative チャネルまたはブローカーをイベントのソースとして使用するように設定する場合、Camel K Operator は対応するインテグレーションを Knative Serving サービスとして実現し、Knative が提供する自動スケーリング機能を活用します。

3.2. KNATIVE の設定

Knative を設定するには、必要な OpenShift Operator をインストールし、Knative チャネルを作成します。

3.2.1. OpenShift クラスターの準備

Kamelets および OpenShift Serverless を使用するには、以下の Operator、コンポーネント、および CLI ツールをインストールします。

- **Red Hat Integration - Camel K**Operator および CLI ツール: Operator は Camel K (OpenShift のクラウドでネイティブに実行される軽量なインテグレーションフレームワーク) をインストールし、管理します。**kamel** CLI ツールを使用すると、すべての Camel K 機能にアクセスできます。
  Camel K のインストール のインストール手順を参照してください。

- **OpenShift Serverless** Operator: サーバーレスを実行するためのコンテナー、マイクロサービス、および機能を有効にする API のコレクションです。Serverless アプリケーションはオンデマンドでスケールアップおよびスケールダウン (ゼロまで) でき、数多くのイベントソースによりトリガーされます。OpenShift Serverless Operator のインストール時に、これは **knative-**

serving namespace (Knative Serving コンポーネントのインストール用) および **knative-eventing** namespace (Knative Eventing コンポーネントのインストールに必要) を自動的に作成します。

- **Knative Eventing** コンポーネント
- **Knative Serving** コンポーネント
- **Knative CLI** ツール (**kn**): コマンドラインまたは Shell スクリプトから Knative リソースを作成できます。

3.2.1.1. 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イスを提供します。

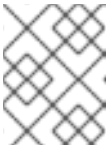
OpenShift Serverless Operator は、Knative Serving および Knative Eventing 機能の両方をサポートします。詳細は、installing OpenShift Serverless Operator を参照してください。

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** > OpenShift バージョンと一致するチャンネルを選択 (**4.10** など)
   - **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 をデプロイメント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 documentation を参照してください。

関連情報

- OpenShift ドキュメントの Installing OpenShift Serverless

3.2.2. Knative チャネルの作成

Knative チャネルは、イベントを転送するカスタムリソースです。イベントがイベントソースまたは生成側からチャネルに送信された後に、これらのイベントはサブスクリプションを使用して複数の Knative サービスまたは他のシンクに送信できます。

この例では、開発の目的で OpenShift Serverless で使用する **InMemoryChannel** チャネルを使用します。**InMemoryChannel** タイプのチャネルには、以下の制限事項があることに注意してください。

- イベントの永続性は利用できません。Pod がダウンすると、その Pod のイベントが失われます。
- **InMemoryChannel** チャネルはイベントの順序を実装しないため、チャネルで同時に受信される 2 つのイベントはいずれの順序でもサブスクライバーに配信できます。
- サブスクライバーがイベントを拒否する場合、再配信はデフォルトで試行されません。Subscription オブジェクトの delivery 仕様を変更することで、再配信の試行を設定できます。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および Knative Serving コンポーネントが OpenShift Container Platform クラスターにインストールされている。
- *OpenShift Serverless CLI (**kn**) がインストールされている。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. OpenShift クラスターにログインします。
2. インテグレーションアプリケーションを作成するプロジェクトを開きます。以下に例を示します。
   oc project camel-k-knative
3. Knative (**kn**) CLI コマンドを使用してチャネルを作成します。
   kn channel create <channel_name> --type <channel_type>

   たとえば、**mychannel** という名前のチャネルを作成するには、以下を実行します。

   kn channel create mychannel --type messaging.knative.dev:v1:InMemoryChannel
4. チャネルが存在することを確認するには、以下のコマンドを入力してすべての既存チャネルをリスト表示します。
   kn channel list

   チャネルがリストに表示されます。

次のステップ

- Kamelet Binding でのデータソースの Knative 宛先への接続
- Kamelet Binding での Knative 宛先のデータシンクへの接続

3.2.3. Knative ブローカーの作成

Knative ブローカーは、CloudEvents のプールを収集するためのイベントメッシュを定義するカスタムリソースです。OpenShift Serverless は、**kn** CLI を使用して作成できるデフォルト Knative ブローカーを提供します。

たとえば、アプリケーションが複数のイベントタイプを処理し、各イベントタイプのチャネルを作成したくない場合は、Kamelet Binding のブローカーを使用できます。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および Knative Serving コンポーネントが OpenShift Container Platform クラスターにインストールされている。
- *OpenShift Serverless CLI (**kn**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. OpenShift クラスターにログインします。

2. インテグレーションアプリケーションを作成するプロジェクトを開きます。以下に例を示します。
   oc project camel-k-knative

3. この Knative (**kn**) CLI コマンドを使用してブローカーを作成します。
   kn broker create default

4. ブローカーが存在することを確認するには、以下のコマンドを入力してすべての既存ブローカーをリスト表示します。
   kn ブローカーリスト

デフォルトブローカーがリストに表示されるはずです。

次のステップ

- Kamelet Binding でのデータソースの Knative 宛先への接続
- Kamelet Binding での Knative 宛先のデータシンクへの接続

3.3. KAMELET BINDING でのデータソースの KNATIVE 宛先への接続

データソースを Knative 宛先 (チャネルまたはブローカー) に接続するには、**図 3.2** に示されるように Kamelet Binding を作成します。

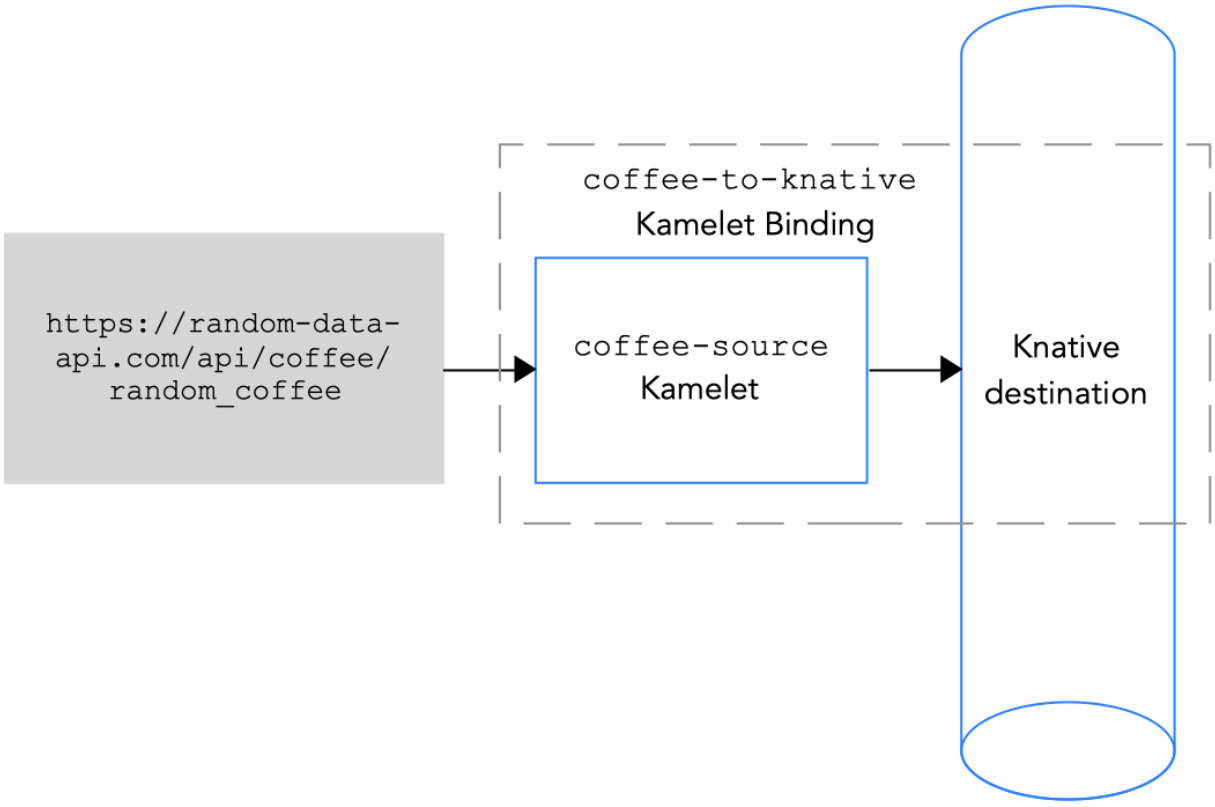


図 3.2 データソースの Knative 宛先への接続

Knative 宛先は Knative チャネルまたは Knative ブローカーになります。

データをチャネルに送信する場合、チャネルには 1 つのイベントタイプのみがあります。Kamelet Binding でチャネルのプロパティー値を指定する必要はありません。

データをブローカーに送信する場合、ブローカーが複数のイベントタイプを処理できるため、Kamelet Binding でブローカーを参照する場合は、タイププロパティーの値を指定する必要があります。

前提条件

- イベントの送信先となる Knative チャネルまたはブローカーの名前およびタイプを知っている必要があります。
  この手順の例では、**mychannel** という名前の **InMemoryChannel** チャネルまたは **default** という名前のブローカーを使用します。ブローカーのサンプルでは、coffee イベントの **type** プロパティーが is **coffee** になります。

- Camel インテグレーションに追加する Kamelet と必要なインスタンスパラメーターを把握している。
  この手順の Kamelet の例では、**coffee-source** Kamelet です。各イベントを送信する頻度を指定するオプションのパラメーター **period** があります。ソース Kamelet の例 のコードを、**coffee-source.kamelet.yaml** ファイルという名前のファイルにコピーしてから、以下のコマンドを実行して、これをリソースとして namespace に追加できます。

  oc apply -f coffee-source.kamelet.yaml

手順

データソースを Knative 宛先に接続するには、Kamelet Binding を作成します。

1. 任意のエディターで、以下の基本構造で YAML ファイルを作成します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name:
spec:
  source:
  sink:
```

2. Kamelet Binding の名前を追加します。この例では、バインディングが **coffee-source** Kamelet を Knative 宛先に接続するため、名前は **coffees-to-knative** になります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffees-to-knative
spec:
  source:
  sink:
```

3. Kamelet Binding のソースの場合は、データソース Kamelet を指定し (たとえば、**coffee-source** Kamelet はコーヒーに関するデータが含まれるイベントを生成します)、Kamelet のパラメーターを設定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
```

```
  name: coffees-to-knative
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
```

4. Kamelet Binding のシンクでは、Knative チャネルまたはブローカーおよび必要なパラメーターを指定します。
以下の例では、Knative チャネルをシンクとして指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffees-to-knative
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
    ref:
      apiVersion: messaging.knative.dev/v1
      kind: InMemoryChannel
      name: mychannel
```

この例では、Knative ブローカーをシンクとして指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: coffees-to-knative
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: coffee-source
    properties:
      period: 5000
  sink:
      ref:
        kind: Broker
        apiVersion: eventing.knative.dev/v1
        name: default
      properties:
        type: coffee
```

5. YAML ファイルを保存します (例: **coffees-to-knative.yaml**)。
6. OpenShift プロジェクトにログインします。
7. Kamelet Binding をリソースとして OpenShift namespace に追加します。
   oc apply -f <kamelet binding filename>

   以下に例を示します。

   oc apply -f coffees-to-knative.yaml

   Camel K Operator は、**KameletBinding** リソースを使用して Camel K インテグレーションを生成し、実行します。ビルドに数分かかる場合があります。
8. **KameletBinding** のステータスを表示するには、以下を実行します。
   oc get kameletbindings
9. インテグレーションの状態を表示するには、以下を実行します。
   oc get integrations
10. インテグレーションのログを表示するには、以下を実行します。
    kamel logs <integration> -n <project>

    以下に例を示します。

    kamel logs coffees-to-knative -n my-camel-knative

次のステップ

- Kamelet Binding での Knative 宛先のデータシンクへの接続

関連項目

- コネクション内のデータへの操作の適用
- コネクション内でのエラーの処理

3.4. KAMELET BINDING での KNATIVE 宛先のデータシンクへの接続

Knative 宛先をデータシンクに接続するには、**図 3.3** にあるように Kamelet Binding を作成します。

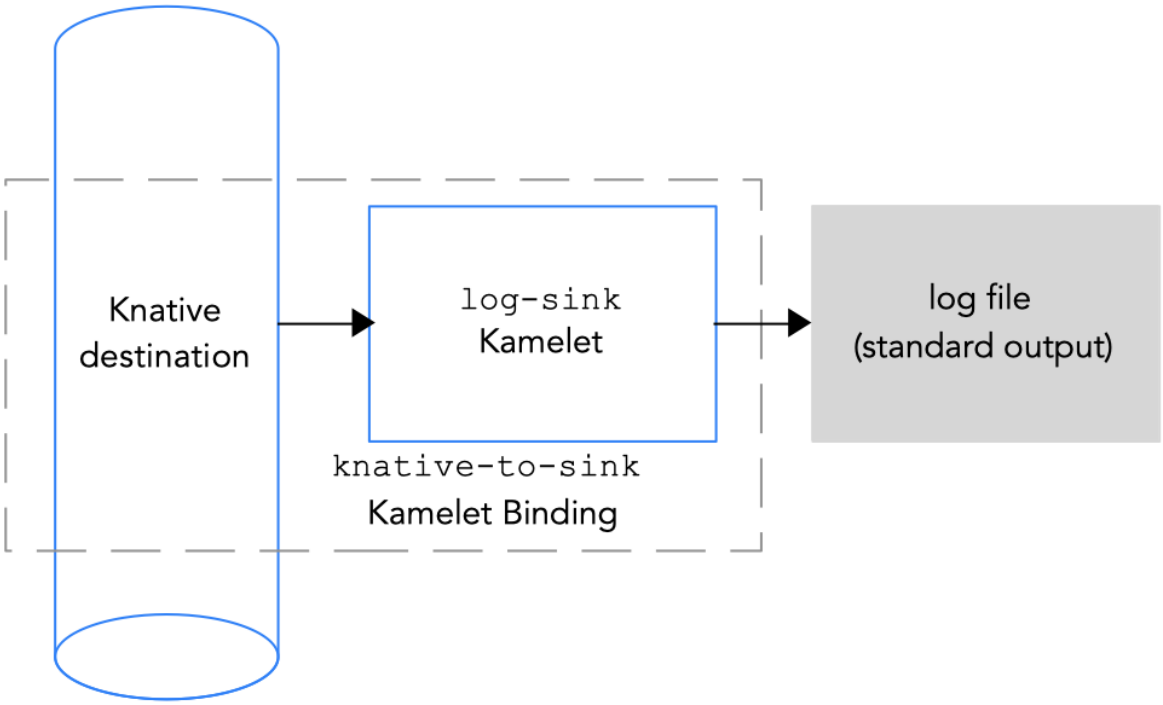


図 3.3 Knative 宛先のデータシンクへの接続

Knative 宛先は Knative チャネルまたは Knative ブローカーになります。

データをチャネルから送信する場合、チャネルには 1 つのイベントタイプのみがあります。Kamelet Binding でチャネルのプロパティー値を指定する必要はありません。

データをブローカーから送信する場合、ブローカーが複数のイベントタイプを処理できるため、Kamelet Binding でブローカーを参照する場合は、タイププロパティーの値を指定する必要があります。

前提条件

- イベントの受信元となる Knative チャネルのタイプまたはブローカーの名前を知っている必要があります。ブローカーでは、受信するイベントのタイプも知っている必要があります。
  この手順の例では、mychannel という名前の InMemoryChannel チャネルまたは mybroker という名前のブローカーおよびコーヒーイベント (type プロパティー) を使用します。これらは、Kamelet Binding でのデータソースの Knative チャネルへの接続 でコーヒーソースからイベントを受信するのに使用した宛先の例と同じです。

- Camel インテグレーションに追加する Kamelet と必要なインスタンスパラメーターを把握している。
  この手順の Kamelet の例は、Kamelet Catalog で提供される **log-sink** Kamelet であり、これはテストおよびデバッグに役立ちます。データのメッセージボディーを表示するために指定された **showStreams** パラメーター。

手順

Knative チャネルをデータシンクに接続するには、Kamelet Binding を作成します。

1. 任意のエディターで、以下の基本構造で YAML ファイルを作成します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name:
spec:
  source:
  sink:
```

2. Kamelet Binding の名前を追加します。この例では、バインディングが Knative 宛先を **log-sink** Kamelet に接続するため、名前は **knative-to-log** になります。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: knative-to-log
spec:
  source:
  sink:
```

3. Kamelet Binding のソースでは、Knative チャネルまたはブローカーおよび必要なパラメーターを指定します。
以下の例では、Knative チャネルをソースとして指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: knative-to-log
spec:
  source:
    ref:
      apiVersion: messaging.knative.dev/v1
      kind: InMemoryChannel
      name: mychannel
  sink:
```

この例では、Knative ブローカーをソースとして指定します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: knative-to-log
spec:
  source:
    ref:
      kind: Broker
      apiVersion: eventing.knative.dev/v1
      name: default
    properties:
      type: coffee
sink:
```

4. Kamelet Binding のシンクでは、データコンシューマー Kamelet (例: **log-sink** Kamelet) を指定し、Kamelet のパラメーターを設定します。以下に例を示します。

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: knative-to-log
spec:
  source:
    ref:
      apiVersion: messaging.knative.dev/v1
      kind: InMemoryChannel
      name: mychannel
  sink:
    ref:
      apiVersion: camel.apache.org/v1alpha1
      kind: Kamelet
      name: log-sink
    properties:
      showStreams: true
```

5. YAML ファイルを保存します (例: **knative-to-log.yaml**)。

6. OpenShift プロジェクトにログインします。

7. Kamelet Binding をリソースとして OpenShift namespace に追加します (**oc apply -f <kamelet binding filename>**)。
以下に例を示します。

   oc apply -f knative-to-log.yaml

   Camel K Operator は、**KameletBinding** リソースを使用して Camel K インテグレーションを生成し、実行します。ビルドに数分かかる場合があります。

8. **KameletBinding** のステータスを表示するには、以下を実行します。
oc get kameletbindings

9. インテグレーションの状態を表示するには、以下を実行します。
oc get integrations

10. インテグレーションのログを表示するには、以下を実行します。
kamel logs <integration> -n <project>

    以下に例を示します。

    kamel logs knative-to-log -n my-camel-knative

    この出力では、以下の例のようにコーヒーイベントが表示されるはずです。

```
[1] INFO [sink] (vert.x-worker-thread-1) {"id":254,"uid":"8e180ef7-8924-4fc7-ab81-
d6058618cc42","blend_name":"Good-morning Star","origin":"Santander,
Colombia","variety":"Kaffa","notes":"delicate, creamy, lemongrass, granola,
soil","intensifier":"sharp"}
[1] INFO [sink] (vert.x-worker-thread-2) {"id":8169,"uid":"3733c3a5-4ad9-43a3-9acc-
d4cd43de6f3d","blend_name":"Caf? Java","origin":"Nayarit, Mexico","variety":"Red
Bourbon","notes":"unbalanced, full, granola, bittersweet chocolate,
nougat","intensifier":"delicate"}
```

11. 実行中のインテグレーションを停止するには、関連付けられた Kamelet Binding リソースを削除します。
oc delete kameletbindings/<kameletbinding-name>

以下に例を示します。

oc delete kameletbindings/knative-to-log

関連項目

- コネクション内のデータへの操作の適用
- コネクション内でのエラーの処理

第4章 KAMELETS 参照

4.1. KAMELET 構造

通常、Kamelet は YAML ドメイン固有の言語でコーディングされます。ファイル名の接頭辞は、Kamelet の名前です。たとえば、**FTP sink** という名前の Kamelet のファイル名は **ftp-sink.kamelet.yaml** です。

OpenShift では、Kamelet は、(ファイル名ではなく) Kamelet の名前を表すリソースであることに注意してください。

概略では、Kamelet リソースは以下を説明します。

- Kamelet の ID、および Kamelet のタイプ (**source**、**sink**、**action**) 等のその他の情報が含まれるメタデータセクション。
- Kamelet の設定に使用できるパラメーターセットが含まれる定義 (JSON-schema 仕様)。
- Kamelet によって想定される入出力に関する情報が含まれるオプションの **types** セクション。
- Kamelet の実装を定義する YAML DSL の Camel テンプレート。

以下の図は、Kamelet とその部分の例を示しています。

Kamelet 構造の例

```
telegram-text-source.kamelet.yaml
apiVersion: camel.apache.org/v1alpha1
kind: Kamelet
metadata:
  name: telegram-source (1)
  annotations: (2)
    camel.apache.org/catalog.version: "master-SNAPSHOT"
    camel.apache.org/kamelet.icon: "data:image/..."
    camel.apache.org/provider: "Red Hat"
    camel.apache.org/kamelet.group: "Telegram"
  labels: (3)
    camel.apache.org/kamelet.type: "source"
spec:
  definition: (4)
    title: "Telegram Source"
    description: |-
      Receive all messages that people send to your telegram bot.
      To create a bot, contact the @botfather account using the
      Telegram app.
      The source attaches the following headers to the messages:
      - chat-id / ce-chatid: the ID of the chat where the
      message comes from
    required:
      - authorizationToken
    type: object
    properties:
      authorizationToken:
        title: Token
        description: The token to access your bot on Telegram, that you
```

```
            can obtain from the Telegram "Bot Father".
          type: string
          format: password
        x-descriptors:
        - urn:alm:descriptor:com.tectonic.ui:password
  types: 5
    out:
      mediaType: application/json
  dependencies:
  - "camel:jackson"
  - "camel:kamelet"
  - "camel:telegram"
  template: 6
    from:
      uri: telegram:bots
      parameters:
        authorizationToken: "{{authorizationToken}}"
      steps:
      - set-header:
        name: chat-id
        simple: "${header[CamelTelegramChatId]}"
      - set-header:
        name: ce-chatid
        simple: "${header[CamelTelegramChatId]}"
      - marshal:
        json: {}
      - to: "kamelet:sink"
```

1. Kamelet ID: Kamelet を参照する場合は Camel K インテグレーションでこの ID を使用します。
2. アイコンなどのアノテーションは、Kamelet の表示機能を提供します。
3. ラベルを使用すると、ユーザーは Kamelets にクエリーできます (例: ソース、シンク、または アクションにより)。
4. JSON-schema 仕様形式の Kamelet およびパラメーターの説明。
5. 出力のメディアタイプ (スキーマを含む)。
6. Kamelet の動作を定義するルートテンプレート。

4.2. ソース KAMELET の例

以下は、**coffee-source** Kamelet の例の内容です。

```
apiVersion: camel.apache.org/v1alpha1
kind: Kamelet
metadata:
  name: coffee-source
  labels:
    camel.apache.org/kamelet.type: "source"
spec:
  definition:
    title: "Coffee Source"
    description: "Retrieve a random coffee from a catalog of coffees"
```

```
  properties:
    period:
      title: Period
      description: The interval between two events in seconds
      type: integer
      default: 1000
types:
  out:
    mediaType: application/json
template:
  from:
    uri: timer:tick
    parameters:
      period: "{{period}}"
    steps:
    - to: "https://random-data-api.com/api/coffee/random_coffee"
    - to: "kamelet:sink"
```