Red Hat build of Apache Camel K 1.10.5

Camel K 入门

开发并运行您的第一个 Camel K 应用程序

Last Updated: 2024-03-06

Red Hat build of Apache Camel K 1.10.5 Camel K 入门

开发并运行您的第一个 Camel K 应用程序

法律通告

摘要

如何安装红帽构建的 Apache Camel K，设置开发环境并运行示例应用程序。

目录

前言

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 CAMEL K 简介

本章介绍了 Red Hat Integration - Camel K 提供的概念、功能和云原生架构：

- 第 1.1 节 “Camel K 概述”
- 第 1.2 节 “Camel K 功能”
- 第 1.2.3 节 “kamelets”
- 第 1.3 节 “Camel K 开发工具”
- 第 1.4 节 “Camel K 分发”

1.1. CAMEL K 概述

Red Hat Integration - Camel K 是一个轻量级集成框架，从 Apache Camel K 构建，它在 OpenShift 上的云中原生运行。Camel K 专为无服务器和微服务架构而设计。您可以使用 Camel K 在 OpenShift 上直接运行使用 Camel 域特定语言(DSL)编写的集成代码。Camel K 是 Apache Camel 开源社区的子项目：https://github.com/apache/camel-k。

Camel K 使用 Go 编程语言实现，并使用 Kubernetes Operator SDK 在云中自动部署集成。例如，这包括在 OpenShift 上自动创建服务和路由。这在部署和重新部署云中集成时（如几秒或几分钟）提供了更快的时间。

Camel K 运行时提供了显著的性能优化。Quarkus 云原生 Java 框架默认为启用，以提供更快的启动时间，并减少内存和 CPU 占用量。在开发人员模式下运行 Camel K 时，您可以对集成 DSL 进行实时更新，并立即查看 OpenShift 上的云，而无需等待重新部署的集成。

使用带有 OpenShift Serverless 和 Knative Serving 的 Camel K 仅根据需要创建容器，并在负载下自动缩放为零。这通过消除服务器置备和维护的开销来降低成本，并可让您专注于应用程序开发。

将 Camel K 与 OpenShift Serverless 和 Knative Eventing 搭配使用，您可以管理系统中的组件如何在事件驱动的架构中进行通信。这通过使用发布订阅或事件流模型，通过事件生产者和消费者之间的分离关系提供灵活性和提高效率。

其他资源

- Apache Camel K 网站
- OpenShift Serverless 入门

1.2. CAMEL K 功能

Camel K 包括以下主要平台和特性：

1.2.1. 平台和组件版本

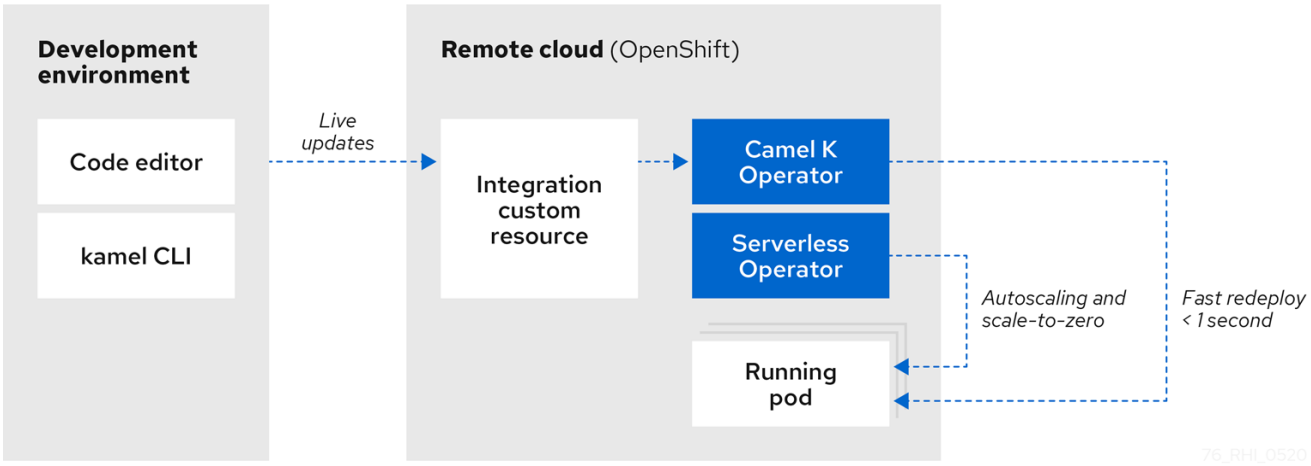
- OpenShift Container Platform 4.13, 4.14
- OpenShift Serverless 1.31.0
- Red Hat Build of Quarkus 2.13.8.Final-redhat-00006
- Red Hat Camel Extensions for Quarkus 2.13.3.redhat-00008

- Apache Camel K 1.10.5.redhat-00002
- Apache Camel 3.18.6.redhat-00007
- OpenJDK 11

1.2.2. Camel K 功能

- 用于自动扩展和缩减为零的 Knative Serving
- 用于事件驱动的架构的 Knative Eventing
- 默认情况下，使用 Quarkus 运行时的性能优化
- 使用 Java 或 YAML DSL 编写的 Camel 集成
- 使用 Visual Studio Code 开发工具
- 使用 OpenShift 中的 Prometheus 监控集成
- Quickstart 教程
- kamelet 连接器目录到外部系统，如 AWS、JIRA 和 Salesforce

下图显示了 Camel K 云原生架构的简化视图：



其他资源

- Apache Camel 架构

1.2.3. kamelets

kamelets 在简单接口后隐藏连接到外部系统的复杂性，其中包含实例化它们所需的所有信息，即使不熟悉 Camel 的用户。

kamelets 作为自定义资源实现，您可以在 OpenShift 集群上安装并在 Camel K 集成中使用。kamelets 是使用 Camel 组件用来连接到外部系统的路由模板，而无需深入了解组件。Kamelets 对连接到外部系统的详细信息进行了抽象。您还可以组合 Kamelets 来创建复杂的 Camel 集成，就像使用标准 Camel 组件一样。

其他资源

其他资源

- 将应用程序与 Kamelets 集成

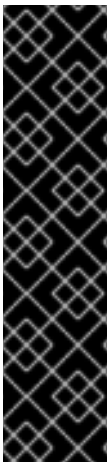
1.3. CAMEL K 开发工具

Camel K 为 Visual Studio (VS) Code、Red Hat CodeReady WorkSpaces 和 Eclipse Che 提供开发工具扩展。基于 Camel 的工具扩展包括自动完成 Camel DSL 代码、Camel K 模式行配置和 Camel K 特征等功能。

可用的 VS Code 开发工具扩展如下：

- 红帽用于 Apache Camel 的 VS Code 扩展包
  - Apache Camel K 扩展的工具
  - Apache Camel 扩展的语言支持
  - Apache Camel K 的调试适配器
  - OpenShift、Java 等等的额外扩展

有关如何为 Camel K 设置这些 VS Code 扩展的详情，请参阅设置 Camel K 开发环境。



重要

- 以下插件 VS Code Language support for Camel （Camel extension pack 的一部分）提供了在编辑 Camel 路由和 **application.properties** 时提供内容协助。
- 要为在 OpenShift 上创建、运行和操作 Camel K 集成的 VS 代码安装受支持的 Camel K 工具扩展，请参阅 红帽扩展为 Apache Camel K 的 VS Code 工具工具
- 要为 VS 代码安装受支持的 Camel 调试工具扩展，以调试用 Java、YAML 或 XML 本地编写的 Camel 集成，请参阅 红帽 Apache Camel 的 Debug Adapter。
- 有关与特定产品版本搭配使用开发人员工具的配置和组件的详情，请参阅 Camel K 支持的配置 和 Camel K 组件详情

注意： Camel K VS Code 扩展是社区功能。

Eclipse Che 还提供使用 **vscode-camelk** 插件的这些功能。

有关 开发支持范围的更多信息，请参阅开发支持覆盖范围

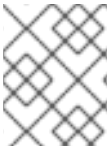
其他资源

- 用于 Apache Camel K 的 VS Code 工具
- 适用于 Apache Camel K 的 Eclipse Che 工具

1.4. CAMEL K 分发

表 1.1. Red Hat Integration - Camel K 发行版

分发	描述	位置
Operator 镜像	Red Hat Integration 的容器镜像 - Camel K Operator: **integration/camel-k-rhel8-operator**	• OpenShift Web 控制台在 **Operators → OperatorHub** 下 • registry.redhat.io
Maven 存储库	Red Hat Integration 的 Maven 工件 - Camel K 红帽提供了 Maven 存储库，托管我们提供的产品的内容。这些软件仓库可从软件下载页面下载。 对于 Red Hat Integration - Camel K，需要以下软件仓库： • rhi-common • rhi-camel-quarkus • rhi-camel-k 不支持在断开连接的环境中安装 Red Hat Integration - Camel K（离线模式）。	红帽构建的 Apache Camel 软件下载
源代码	Red Hat Integration 的源代码 - Camel K	红帽构建的 Apache Camel 软件下载
Quickstarts	快速入门指南： • 基本 Java 集成 • 事件流集成 • JDBC 集成 • JMS 集成 • Kafka 集成 • Knative 集成 • SaaS 集成 • Serverless API 集成 • 转换集成	https://github.com/openshift-integration



注意

您必须有 Apache Camel K 构建的订阅，并登录到红帽客户门户网站以访问 Red Hat Integration - Camel K 发行版。

第 2 章 准备 OPENSHIFT 集群

本章介绍了如何在 OpenShift 上安装 Red Hat Integration - Camel K 和 OpenShift Serverless，以及如何在开发环境中安装所需的 Camel K 和 OpenShift Serverless 命令行工具。

- 第 2.1 节 “安装 Camel K”
- 第 2.2 节 “安装 OpenShift Serverless”

2.1. 安装 CAMEL K

您可以从 OperatorHub 在 OpenShift 集群上安装 Red Hat Integration - Camel K Operator。OperatorHub 由 OpenShift Container Platform Web 控制台获得，为集群管理员提供了一个界面，以发现和安装 Operator。

安装 Camel K Operator 后，您可以安装 Camel K CLI 工具以命令行访问所有 Camel K 功能。

先决条件

- 您可以访问具有正确访问级别的 OpenShift 4.6 （或更新版本）集群、创建项目和安装操作器的功能，以及在本地系统上安装 CLI 工具。

注意

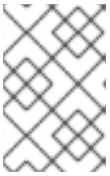
> You do not need to create a pull secret when installing Camel K from the OpenShift OperatorHub. The Camel K Operator automatically reuses the OpenShift cluster-level authentication to pull the Camel K image from `registry.redhat.io`.

- 已安装 OpenShift CLI 工具(**oc**)，以便可以在命令行中与 OpenShift 集群交互。有关如何安装 OpenShift CLI 的详情，请参阅安装 OpenShift CLI。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 创建新的 OpenShift 项目：
   a. 在左侧导航菜单中点 **Home** > **Project** > **Create Project**。
   b. 输入项目名称，如 **my-camel-k-project**，然后单击 **Create**。
3. 在左侧导航菜单中点 **Operators** > **OperatorHub**。
4. 在 **Filter by keyword** 文本框中，键入 **Camel K**，然后点 **Red Hat Integration - Camel K Operator** 卡。
5. 阅读有关操作器的信息，然后单击 **Install**。Operator 安装页面将打开。
6. 选择以下订阅设置：
   - **Update Channel** > **latest**
   - 选择以下 2 个选项：

- Installation Mode > A specific namespace on the cluster> my-camel-k-project
- Installation Mode > All namespaces on the cluster (default)> Openshift operator



注意

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

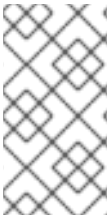
- Approval Strategy > Automatic

注意

如果您的环境需要，可以使用 **Installation mode** > **All namespaces on the cluster** 和 **Approval Strategy** > **Manual** settings。

7. 点 **Install**，等待片刻，直到 Camel K Operator 准备就绪。
8. 下载并安装 Camel K CLI 工具：
   a. 在 OpenShift Web 控制台顶部的 **帮助菜单(?)** 中，选择 **Command line tools**。
   b. 向下滚动到 **kamel - Red Hat Integration - Camel K - Command Line Interface**部分。
   c. 单击链接以下载本地操作系统的二进制文件(Linux、Mac、Windows)。
   d. 在您的系统路径中解压并安装 CLI。
   e. 要验证您可以访问 Kamel K CLI，请打开一个命令窗口，然后输入以下内容：
   kamel --help

   此命令显示有关 Camel K CLI 命令的信息。



注意

如果使用 OLM 从 OperatorHub 卸载 Camel K operator，则 CRD 不会被删除。要改回到以前的 Camel K Operator，您必须使用以下命令手动删除 CRD。

oc get crd -l app=camel-k -o name | xargs oc delete

后续步骤

（可选） 指定 Camel K 资源限值

2.1.1. 一致的集成平台设置

您可以创建命名空间本地 Integration Platform 资源，以覆盖 Operator 中使用的设置。

这些命名空间本地平台设置必须默认从 Operator 使用的 Integration Platform 派生。也就是说，只有明确指定的设置会覆盖 Operator 中使用的平台默认值。

因此，您必须使用一致的平台设置层次结构，其中全局 Operator 平台设置始终代表用户指定平台设置的基础。

1. 对于全局 Camel K operator，如果 IntegrationPlatform 指定非默认 spec.build.buildStrategy，则此值也会传播到命名空间 Camel-K operator 之后安装。

2. buildStrategy 的默认值为 routine。

```
$ oc get ip camel-k -o yaml -n openshift-operators
apiVersion: camel.apache.org/v1
kind: IntegrationPlatform
metadata:
  labels:
    app: camel-k
  name: camel-k
  namespace: openshift-operators
spec:
  build:
    buildStrategy: pod
```

3. 全局 operator IntegrationPlatform 的参数 buildStrategy 可通过以下方式之一编辑：

   - 从仪表板中
     - Administrator 视图：Operators → Installed operators → in namespace openshift-operators （即全局安装的 operator），选择 Red Hat Integration - Camel K → Integration Platform → YAML
     - 现在添加或编辑（如果已存在）spec.build.buildStrategy: pod
     - 点 Save
   - 使用以下命令：之后安装任何命名空间 Camel K operator 都会继承来自全局 IntegrationPlatform 的设置。

```
oc patch ip/camel-k -p '{"spec":{"build":{"buildStrategy": "pod"}}}' --type merge -n openshift-operators
```

2.1.2. 指定 Camel K 资源限值

安装 Camel K 时，Camel K 的 OpenShift pod 没有为 CPU 和内存(RAM)资源设置任何限制。如果要为 Camel K 定义资源限值，您必须编辑在安装过程中创建的 Camel K 订阅资源。

前提条件

- 您有集群管理员访问安装 Camel K Operator 的 OpenShift 项目，如 安装 Camel K 所述。
- 您知道要应用到 Camel K 订阅的资源限值。有关资源限值的更多信息，请参阅以下文档：
  - 在 OpenShift 文档中 设置部署资源。
  - Kubernetes 文档中的管理容器资源。

流程

1. 登录 OpenShift Web 控制台。
2. 选择 **Operators** > **Installed Operators** > **Operator Details** > **Subscription**。

3. 选择 **Actions** > **Edit Subscription**。
   订阅的文件在 YAML 编辑器中打开。

4. 在 **spec** 部分下，添加一个 **config.resources** 部分，并为 memory 和 cpu 提供值，如下例所示：

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

5. 保存您的更改。

OpenShift 更新订阅并应用您指定的资源限值。

2.2. 安装 OPENSHIFT SERVERLESS

您可以从 OperatorHub 在 OpenShift 集群上安装 OpenShift Serverless Operator。OperatorHub 由 OpenShift Container Platform Web 控制台获得，为集群管理员提供了一个界面，以发现和安装 Operator。

OpenShift Serverless Operator 支持 Knative Serving 和 Knative Eventing 功能。如需了解更多详细信息，请参阅安装 OpenShift Serverless Operator。

先决条件

- 具有集群管理员访问安装了 Camel K Operator 的 OpenShift 项目。
- 已安装 OpenShift CLI 工具(**oc**)，以便可以在命令行中与 OpenShift 集群交互。有关如何安装 OpenShift CLI 的详情，请参阅安装 OpenShift CLI。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 在左侧导航菜单中点 **Operators** > **OperatorHub**。
3. 在 **Filter by keyword** 文本框中，输入 **Serverless** 以查找 **OpenShift Serverless Operator**。
4. 阅读 Operator 的信息，然后点 **Install** 显示 Operator 订阅页面。
5. 选择默认订阅设置：
   - **Update Channel** > 选择与 OpenShift 版本匹配的频道，如 **4.14**
   - **Installation Mode** > **All namespaces on the cluster**
   - **Approval Strategy** > **Automatic**

注意

如果您的环境需要，也可以使用 **Approval Strategy** > **Manual** 设置。

6. 点 **Install**，等待片刻，直到 Operator 准备就绪为止。
7. 使用 OpenShift 文档中的步骤安装所需的 Knative 组件：
   - 安装 Knative Serving
   - 安装 Knative Eventing
8. （可选）下载并安装 OpenShift Serverless CLI 工具：
   a. 在 OpenShift Web 控制台顶部的帮助菜单(?)中，选择 **Command line tools**。
   b. 向下滚动到 **kn - OpenShift Serverless - Command Line Interface**部分。
   c. 单击链接以下载本地操作系统的二进制文件(Linux、Mac、Windows)
   d. 在您的系统路径中解压并安装 CLI。
   e. 要验证是否可以访问 **kn** CLI，请打开一个命令窗口，然后输入以下内容：
   kn --help

   此命令显示有关 OpenShift Serverless CLI 命令的信息。

   如需了解更多详细信息，请参阅 OpenShift Serverless CLI 文档。

其他资源

- 在 OpenShift 文档中安装 OpenShift Serverless

2.3. 为 CAMEL K 配置 MAVEN 存储库

对于 Camel K operator，您可以在 **ConfigMap** 或 secret 中提供 Maven 设置。

流程

1. 要从文件创建 **ConfigMap**，请运行以下命令：

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

然后，创建的 **ConfigMap** 可以在 **IntegrationPlatform** 资源中从 **spec.build.maven.settings** 字段引用。

示例

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

```
    settings:
      configMapKeyRef:
          key: settings.xml
          name: maven-settings
```

或者，您可以使用以下命令直接编辑 **IntegrationPlatform** 资源来引用包含 Maven 设置的 ConfigMap：

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

为远程 Maven 存储库配置 CA 证书

您可以在 Secret 中提供 CA 证书，供 Maven 命令用于连接远程 Maven 存储库。

流程

1. 使用以下命令从文件创建 Secret：

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

2. 从 **spec.build.maven.caSecret** 字段中引用 **IntegrationPlatform** 资源中创建的 Secret，如下所示。

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

第 3 章 开发并运行 CAMEL K 集成

本章介绍了如何设置开发环境以及如何开发和部署使用 Java 和 YAML 编写的简单 Camel K 集成。它还演示了如何使用 **kamel** 命令行来管理 Camel K 集成在运行时。例如，这包括运行、描述、日志记录和删除集成。

- 第 3.1 节 “设置 Camel K 开发环境”
- 第 3.2 节 “在 Java 中开发 Camel K 集成”
- 第 3.3 节 “在 YAML 中开发 Camel K 集成”
- 第 3.4 节 “运行 Camel K 集成”
- 第 3.5 节 “在开发模式下运行 Camel K 集成”
- 第 3.6 节 “使用 modeline 运行 Camel K 集成”
- 第 3.7 节 “Build”
- 第 3.8 节 “在环境间提升”

3.1. 设置 CAMEL K 开发环境

在自动部署 Camel K 快速启动指南前，您必须使用推荐的开发工具设置您的环境。本节介绍如何安装推荐的 Visual Studio (VS)代码 IDE 及其为 Camel K 提供的扩展。

注意

- Camel K VS Code 扩展是社区功能。
- 推荐使用 VS Code 以及 Camel K 的最佳开发人员体验。这包括 Camel DSL 代码和 Camel K trait 的自动完成功能。但是，您可以使用您选择的 IDE 而不是 VS Code 手动输入代码和教程命令。

先决条件

- 您必须有权访问安装了 Camel K Operator 和 OpenShift Serverless Operator 的 OpenShift 集群：
  - 安装 Camel K
  - 从 OperatorHub 安装 OpenShift Serverless

流程

1. 在您的开发平台上安装 VS Code。例如，在 Red Hat Enterprise Linux 中：
   a. 安装所需的密钥和存储库：

```
$ 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=1
\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
/etc/yum.repos.d/vscode.repo'
```

b. 更新缓存并安装 VS Code 软件包：

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

有关在其他平台上安装的详情，请查看 VS Code 安装文档。

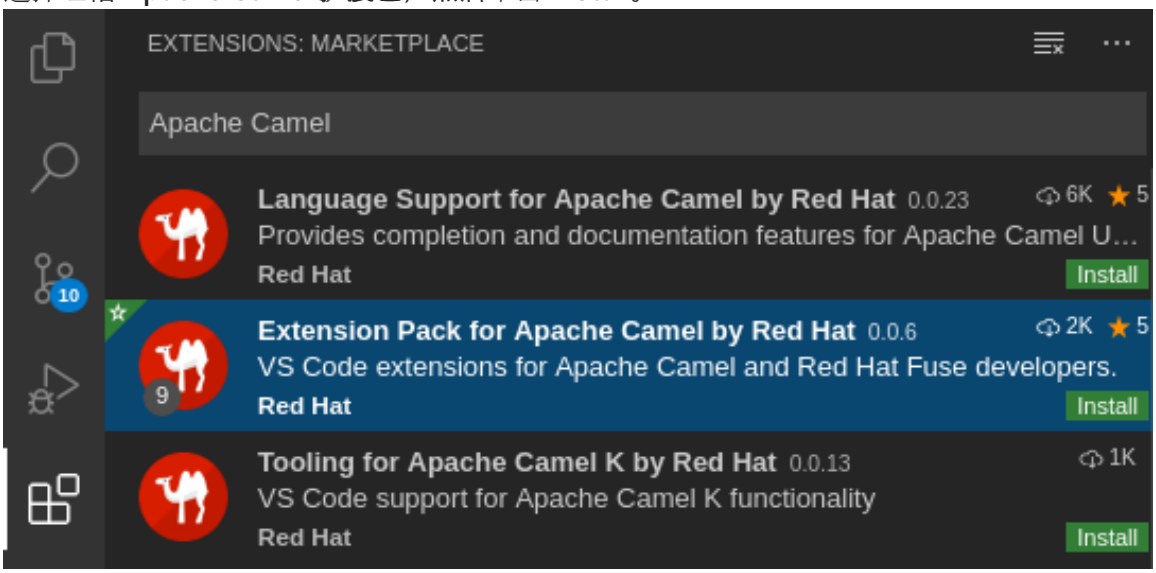
2. 输入 **code** 命令启动 VS Code 编辑器。如需了解更多详细信息，请参阅 VS Code 命令行文档。

3. 安装 VS Code Camel 扩展包，其中包括 Camel K 所需的扩展。例如，在 VS Code 中：

a. 在左侧导航栏中，单击 **Extensions**。

b. 在搜索框中，输入 **Apache Camel**。

c. 选择红帽 **Apache Camel 扩展包**，然后单击 **Install**。



如需了解更多详细信息，请参阅红帽 Apache Camel 扩展包的说明。

其他资源

- VS Code 入门文档
- 红帽扩展用于 Apache Camel K 的 VS Code 工具
- 红帽扩展对 Apache Camel 的 VS Code 语言支持
- Apache Camel K 和 VS Code 工具示例
- 要将 Camel 应用程序从 Camel 3.x 升级到 3.y，请参阅 Camel 3.x 升级指南。

3.2. 在 JAVA 中开发 CAMEL K 集成

本节介绍如何在 Java DSL 中开发简单的 Camel K 集成。编写 Java 中的集成以使用 Camel K 部署与在 Camel 中定义路由规则的方式相同。但是，在使用 Camel K 时，您不需要构建并打包集成为 JAR。

您可以直接在集成路由中使用任何 Camel 组件。Camel K 会自动处理依赖项管理，并使用代码检查从 Camel 目录中导入所有所需的库。

先决条件

- 设置 Camel K 开发环境

流程

1. 输入 **kamel init** 命令来生成简单的 Java 集成文件。例如：

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

2. 打开 IDE 中生成的集成文件，并根据需要编辑。例如，**HelloCamelK.java** 集成自动包含 Camel **计时器** 和**日志** 组件，以帮助您开始：

```
// 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");

  }
}
```

后续步骤

- 运行 Camel K 集成

3.3. 在 YAML 中开发 CAMEL K 集成

本节介绍如何在 YAML DSL 中开发简单的 Camel K 集成。编写 YAML 中的集成以使用 Camel K 部署与在 Camel 中定义路由规则的方式相同。

您可以直接在集成路由中使用任何 Camel 组件。Camel K 会自动处理依赖项管理，并使用代码检查从 Camel 目录中导入所有所需的库。

先决条件

- 设置 Camel K 开发环境

流程

1. 输入 **kamel init** 命令来生成简单的 YAML 集成文件。例如：

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

2. 打开 IDE 中生成的集成文件，并根据需要编辑。例如，**hello.camelk.yaml** 集成自动包含 Camel **计时器** 和**日志** 组件，以帮助您开始：

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

3.4. 运行 CAMEL K 集成

您可以使用 **kamel run** 命令从命令行运行 OpenShift 集群上的 Camel K 集成。

先决条件

- 设置 Camel K 开发环境 .
- 您必须已有一个使用 Java 或 YAML DSL 编写的 Camel 集成。

流程

1. 使用 **oc** 客户端工具登录到 OpenShift 集群，例如：

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

2. 确保 Camel K Operator 正在运行，例如：

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

3. 输入 **kamel run** 命令，以在 OpenShift 上的云中运行集成。例如：

   Java 示例

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

   YAML 示例

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

4. 输入 **kamel get** 命令来检查集成的状态：

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

   当集成首次运行时，Camel K 为容器镜像构建集成工具包，它会下载所有必要的 Camel 模块，并将它们添加到镜像类路径中。

5. 再次输入 **kamel get** 来验证集成是否正在运行：

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

6. 输入 **kamel log** 命令将日志输出到 **stdout** ：

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

7. 按 **Ctrl-C** 终止终端中的日志记录。

其他资源

- 有关 **kamel run** 命令的详情，请输入 **kamel run --help**
- 有关更快的部署周转时间，请参阅 在开发模式下运行 Camel K 集成
- 有关运行集成的开发工具的详情，请参阅 红帽 Apache Camel K 的 VS Code 工具
- 另请参阅 管理 Camel K 集成

在 CLI 中运行集成

您可以在没有 CLI （命令行接口）的情况下运行集成，并使用配置创建一个 集成自定义资源 以运行应用程序。

例如，执行以下示例路由。

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

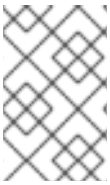
它返回预期的集成自定义资源。

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

将此自定义资源保存到 yaml 文件 **my-integration.yaml** 中。现在，使用 **oc** 命令行、UI 或 API 调用 OpenShift 集群，运行包含集成自定义资源的集成。在以下示例中，从命令行使用 **oc** CLI。

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

Operator 运行 Integration。



注意

- Kubernetes 支持 CustomResourceDefinitions 的 Structural Schemas。
- 有关 Camel K traits 的详情，请参阅 Camel K trait 配置参考 。

自定义资源的模式更改

强烈输入的 Trait API 对以下 CustomResourceDefinitions 进行了更改： **集成、integrationkits' 和 'integrationplatforms。**

spec.traits.<traits.<traits.<traits.< traits **.<traits.<traits.<traits> 下的特征属性现在直接定义。**

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

↓↓↓

```
traits:
  container:
    enabled: true
```

```
    name: my-integration
```

在这个实现中可以向后兼容。为了实现向后兼容，会为每个特征类型提供带有 **RawMessage** 类型的 **Configuration** 字段，以便现有集成和资源从新的红帽构建的 Apache Camel K 版本中读取。

当读取旧的集成和资源时，每个特征中的旧配置（若有）将迁移到新的 Trait API 字段。如果在新的 API 字段中预定义了值，它们会先于旧的字段。

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

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

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

3.5. 在开发模式下运行 CAMEL K 集成

您可以在命令行中以开发模式运行 Camel K 集成。通过使用开发模式，您可以快速迭代开发中的集成，并快速获得对代码的反馈。

当您使用 **--dev** 选项指定 **kamel run** 命令时，这会立即在云中部署集成，并在终端中显示集成日志。然后，您可以更改代码，并查看立即应用到 OpenShift 上的远程集成 Pod 的更改。终端会自动显示云中远程集成的所有重新部署。

注意

Camel K 在开发模式中生成的工件与您在生产环境中运行的工件相同。开发模式的目的是加快开发速度。

先决条件

- 设置 Camel K 开发环境 .
- 您必须已有一个使用 Java 或 YAML DSL 编写的 Camel 集成。

流程

1. 使用 **oc** 客户端工具登录到 OpenShift 集群，例如：

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

2. 确保 Camel K Operator 正在运行，例如：

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

3. 输入带有 **--dev** 的 **kamel run** 命令，以在云中的 OpenShift 上以开发模式运行您的集成。下面显示了一个简单的 Java 示例：

```
$ kamel run HelloCamelK.java --dev
Condition "IntegrationPlatformAvailable" is "True" for Integration hello-camel-k: test/camel-k
Integration hello-camel-k in phase "Initialization"
Integration hello-camel-k in phase "Building Kit"
Condition "IntegrationKitAvailable" is "True" for Integration hello-camel-k: kit-
c49sqn4apkb4qgn55ak0
Integration hello-camel-k in phase "Deploying"
Progress: integration "hello-camel-k" in phase Initialization
Progress: integration "hello-camel-k" in phase Building Kit
Progress: integration "hello-camel-k" in phase Deploying
Integration hello-camel-k in phase "Running"
Condition "DeploymentAvailable" is "True" for Integration hello-camel-k: deployment name is
hello-camel-k
Progress: integration "hello-camel-k" in phase Running
Condition "CronJobAvailable" is "False" for Integration hello-camel-k: different controller
strategy used (deployment)
Condition "KnativeServiceAvailable" is "False" for Integration hello-camel-k: different
controller strategy used (deployment)
Condition "Ready" is "False" for Integration hello-camel-k
Condition "Ready" is "True" for Integration hello-camel-k
[1] Monitoring pod hello-camel-k-7f85df47b8-js7cb
...
...
[1] 2021-08-11 18:34:44,069 INFO  [org.apa.cam.k.Runtime] (main) Apache Camel K
Runtime 1.7.1.fuse-800025-redhat-00001
[1] 2021-08-11 18:34:44,167 INFO  [org.apa.cam.qua.cor.CamelBootstrapRecorder] (main)
bootstrap runtime: org.apache.camel.quarkus.main.CamelMainRuntime
[1] 2021-08-11 18:34:44,362 INFO  [org.apa.cam.k.lis.SourcesConfigurer] (main) Loading
routes from: SourceDefinition{name='HelloCamelK', language='java',
location='file:/etc/camel/sources/HelloCamelK.java', }
[1] 2021-08-11 18:34:46,180 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Routes startup summary (total:1 started:1)
[1] 2021-08-11 18:34:46,180 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Started java (timer://java)
[1] 2021-08-11 18:34:46,180 INFO  [org.apa.cam.imp.eng.AbstractCamelContext] (main)
Apache Camel 3.10.0.fuse-800010-redhat-00001 (camel-1) started in 243ms (build:0ms
init:213ms start:30ms)
[1] 2021-08-11 18:34:46,190 INFO  [io.quarkus] (main) camel-k-integration 1.6.6 on JVM
(powered by Quarkus 1.11.7.Final-redhat-00009) started in 3.457s.
[1] 2021-08-11 18:34:46,190 INFO  [io.quarkus] (main) Profile prod activated.
[1] 2021-08-11 18:34:46,191 INFO  [io.quarkus] (main) Installed features: [camel-bean,
camel-core, camel-java-joor-dsl, camel-k-core, camel-k-runtime, camel-log, camel-support-
common, camel-timer, cdi]
[1] 2021-08-11 18:34:47,200 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
[1] 2021-08-11 18:34:48,180 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
[1] 2021-08-11 18:34:49,180 INFO  [info] (Camel (camel-1) thread #0 - timer://java)
Exchange[ExchangePattern: InOnly, BodyType: String, Body: Hello Camel K from java]
...
```

4. 编辑集成 DSL 文件的内容，保存您的更改，并查看终端中立即显示的更改。例如：

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

5. 按 **Ctrl-C** 终止终端中的日志记录。

其他资源

- 有关 **kamel run** 命令的详情，请输入 **kamel run --help**
- 有关运行集成的开发工具的详情，请参阅 红帽 Apache Camel K 的 VS Code 工具
- 管理 Camel K 集成
- 配置 Camel K 集成依赖项

3.6. 使用 MODELINE 运行 CAMEL K 集成

您可以使用 Camel K modeline 在 Camel K 集成源文件中指定多个配置选项，这些文件在运行时执行。这样可以节省重新输入多个命令行选项的时间，并有助于防止输入错误。

以下示例显示了 Java 集成文件中的 modeline 条目，它启用了 3scale 并限制集成容器内存。

先决条件

- 设置 Camel K 开发环境
- 您必须已有一个使用 Java 或 YAML DSL 编写的 Camel 集成。

流程

1. 在您的集成文件中添加 Camel K modeline 条目。例如：

   ThreeScaleRest.java

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

public class ThreeScaleRest extends RouteBuilder {

  @Override
  public void configure() throws Exception {
     rest().get("/")
       .to("direct:x");

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

   启用容器和 3scale 特征，通过 3scale 公开路由并限制容器内存。

2. 运行集成，例如：

```
kamel run ThreeScaleRest.java
```

   kamel run 命令输出集成中指定的任何模式选项，例如：

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

其他资源

- Camel K 模式行选项
- 有关运行模式集成的开发工具的详情，请参阅 Apache Camel K Modeline 的 IDE 支持 。

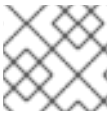
3.7. BUILD

Build 资源描述了组合容器镜像的过程，该容器镜像符合 Integration 或 IntegrationKit 的要求。

构建的结果是一个 IntegrationKit，必须为多个集成 重复使用。

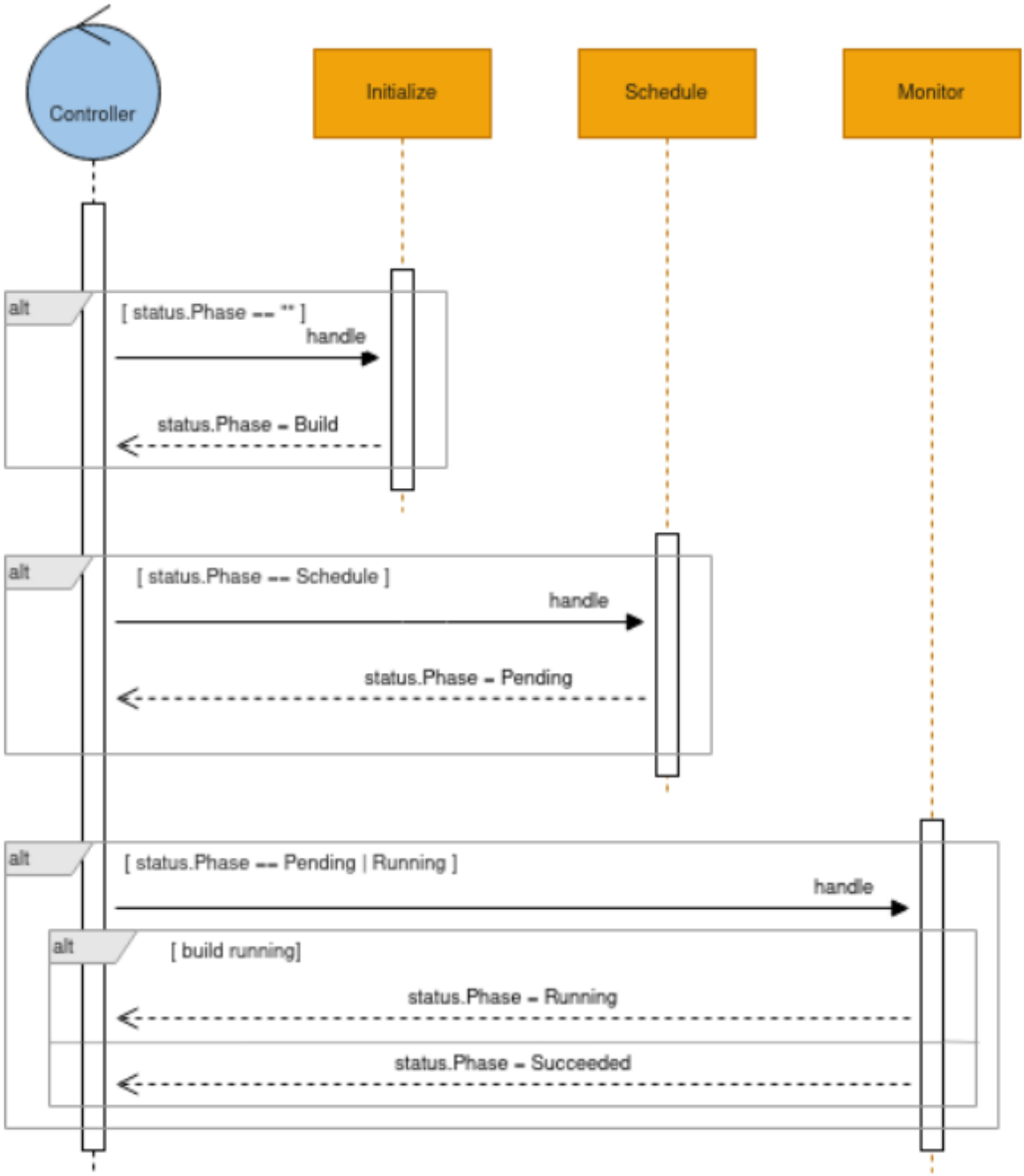
```
type Build struct {
 Spec   BuildSpec    1
 Status BuildStatus   2
}
type BuildSpec struct {
 Tasks []Task         3
}
```

1. 所需的状态
2. 当前时间对象的状态
3. 构建任务



注意

完整的 go 定义 可在此处找到。



3.7.1. 构建策略

您可以从不同的构建策略中选择。构建策略定义如何执行构建，并且遵循可用的策略。

- buildStrategy: pod （每个构建都在单独的 pod 中运行，Operator 会监控 pod 状态）

- buildStrategy: routine （每个构建都作为 operator pod 中的 go 例程运行）

注意

routine 是默认策略。

以下描述允许您决定何时使用哪个策略。

例程 ：提供稍快的构建，因为没有启动额外的 pod，加载的构建依赖项（如 Maven 依赖项）会在构建之间缓存。适用于正在执行的普通构建数量，并且只有几个并行运行构建。

Pod ：防止 Operator 的内存压力，因为构建不会消耗 Operator Go 运行时的 CPU 和内存。适用于正在执行的许多构建以及许多并行构建。

3.7.2. 构建队列

必须对多个集成重复使用 IntegrationKits 及其基础镜像，以完成有效的资源管理，并为 Camel K 集成优化构建和启动时间。

为重复使用镜像，操作器将按顺序排队构建。这样，Operator 便可将高效的镜像分层用于集成。

注意

默认情况下，构建根据其布局（如 native、fast-jar）和构建命名空间按顺序排列。

但是，构建可能无法按顺序运行，但根据某些条件相互并行运行。

- 例如，原生构建将始终与其他构建并行运行。
- 另外，当构建需要使用自定义 IntegrationPlatform 运行时，它可能会与使用默认 operator IntegrationPlatform 运行的其他构建并行运行。
- 通常，当无法重复使用构建的镜像层时，构建被强制与其他构建并行运行。

因此，为了避免许多并行运行构建，Operator 使用最多的运行构建设置数量来限制运行构建量。

您可以在 IntegrationPlatform 设置中设置这个限制。

此限制的默认值基于构建策略。

- buildStrategy: pod (MaxRunningBuilds=10)
- buildStrategy: routine (MaxRunningBuilds=3)

3.8. 在环境间提升

当集成在集群中运行后，您可以将该集成移到更高的环境中。也就是说，您可以 **在开发环境中** 测试集成，在获取结果后，您可以将其移到 **生产环境中**。

Camel K 通过使用 **kamel promote** 命令实现此目标。通过这个命令，您可以将集成从一个命名空间移到另一个命名空间中。

先决条件

- 设置 Camel K 开发环境
- 您必须已有一个使用 Java 或 YAML DSL 编写的 Camel 集成。
- 确保源 operator 和目标 Operator 都使用相同的容器 registry，默认 registry （如果通过 OperatorHub 安装 Camel K operator）为 **registry.redhat.io**
- 另外，请确保目标命名空间提供集成所需的 Configmap、Secret 或 Kamelets。

注意

要使用同一容器 registry，您可以在安装阶段使用 **--registry** 选项，或更改 IntegrationPlatform 来相应地反映。

代码示例

1. 以下是使用 Configmap 在 HTTP 端点上公开一些消息的简单集成。您可以开始在名为 **development** 的命名空间中创建此类集成和测试。

```
kubectl create configmap my-cm --from-literal=greeting="hello, I am development!" -n
development
```

PromoteServer.java

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

public class PromoteServer extends RouteBuilder {
  @Override
  public void configure() throws Exception {
    from("platform-http:/hello?
httpMethodRestrict=GET").setBody(simple("resource:classpath:greeting"));
  }
}
```

2. 现在运行它。

```
kamel run --dev -n development PromoteServer.java --config configmap:my-cm [-t
service.node-port=true]
```

3. 您必须根据 Kubernetes 平台和您要提供的风险级别调整服务特征。之后，您可以对其进行测试。

```
curl http://192.168.49.2:32116/hello
hello, I am development!
```

4. 测试集成后，您可以将其移至生产环境。您必须有目的地环境(Openshift 命名空间)可以与一个 operator （共享同一操作器容器 registry）以及任何配置（如此处使用的 configmap）就绪。为此，请在目标命名空间中创建一个。

```
kubectl create configmap my-cm --from-literal=greeting="hello, I am production!" -n
production
```

注意

为安全起见，需要检查确保目标中存在预期的资源，如 Configmap、Secret 和 Kamelets。如果缺少其中任何这些资源，则集成不会移动。

5. 现在，您可以促进集成。

```
kamel promote promote-server -n development --to production
kamel logs promote-server -n production
```

6. 测试提升的集成。

```
curl http://192.168.49.2:30764/hello
hello, I am production!
```

由于集成被重新使用同一容器镜像，新的应用将立即执行。另外，集成的不可变性也保证，使用的容器与开发中测试的容器完全相同（更改只是配置）。

注意

在测试中运行的集成不会以任何方式更改，并在停止前保持运行。

第 4 章 升级 CAMEL K

您可以自动升级安装的 Camel K operator，但它不会自动升级 Camel K 集成。您必须手动触发 Camel K 集成的升级。本章介绍了如何升级 Camel K operator 和 Camel K 集成。

4.1. 升级 CAMEL K OPERATOR

安装的 Camel K operator 的订阅指定一个更新频道，如 **1.10.x** 频道，用于跟踪和接收 Operator 的更新。要升级 Operator 以开始跟踪并从更新频道接受更新，您可以更改订阅中的更新频道。如需有关为 已安装的 Operator 更改更新频道的更多信息，请参阅升级已安装的 Operator。

注意

- 安装的 Operator 无法变为比当前频道旧的频道。

如果订阅中的批准策略被设置为 Automatic，则升级过程会在所选频道中提供新的 Operator 版本时立即启动。如果批准策略被设置为 Manual，则必须手动批准待处理的升级。

先决条件

- Camel K operator 使用 Operator Lifecycle Manager (OLM)安装。

流程

1. 在 OpenShift Container Platform web 控制台的 **Administrator** 视角中，进入 **Operators → Installed Operators**。
2. 点 **Camel K Operator**。
3. 点 **Subscription** 标签页。
4. 点 **Channel** 中的更新频道的名称。
5. 点击您要更改的更新频道。例如，**latest**。点击 **Save**。这将开始升级到最新的 Camel K 版本。

对于带有自动批准策略 的订阅，升级会自动开始。返回到 **Operators → Installed Operators** 页面，以监控升级的进度。完成后，状态将变为 Succeeded 和 Up to date。

对于采用手动批准策略的订阅，您可以从 Subscription 选项卡中手动批准升级。

4.2. 升级 CAMEL K 集成

当您触发 Camel K operator 的升级时，Operator 会准备要升级的集成，但不会为每个升级触发升级，以避免服务中断。在升级 Operator 时，集成自定义资源不会自动升级到较新的版本，例如，Operator 可能会处于 **1.10.3** 版本，而集成则报告自定义资源的 **status.version** 字段（上一版本 **1.8.2**）。

先决条件

Camel K Operator 使用 Operator Lifecycle Manager (OLM)安装和升级。

流程

- 打开一个终端，再运行以下命令来升级 Camel K 交集：

```
kamel rebuild myintegration
```

这将清除集成资源的状态，Operator 将使用升级版本的工件（如 **1.10.3** 版本）开始部署集成。

4.3. 降级 CAMEL K

您可以通过安装 Operator 的早期版本来降级到旧版本的 Camel K operator。这需要使用 OC CLI 手动触发。有关使用 CLI 安装特定版本 Operator 的更多信息，请参阅安装 Operator 的特定版本。

重要

您必须删除现有 Camel K 操作器，然后安装 Operator 的 specifc 版本，因为 OLM 不支持降级。

安装旧版本的操作器后，请使用 **kamel rebuild** 命令将集成降级到 Operator 版本。例如，

```
kamel rebuild myintegration
```

第 5 章 CAMEL K 快速开始开发人员教程

Red Hat Integration - Camel K 根据 https://github.com/openshift-integration 提供的集成用例提供快速启动开发人员教程。本章详细介绍了如何设置和部署以下教程：

- 第 5.1 节 “部署基本 Camel K Java 集成”
- 第 5.2 节 “部署 Camel K Serverless 与 Knative 集成”
- 第 5.3 节 “部署 Camel K 转换集成”
- 第 5.4 节 “部署 Camel K Serverless 事件流集成”
- 第 5.5 节 “部署基于 Camel K Serverless API 的集成”
- 第 5.6 节 “部署 Camel K SaaS 集成”
- 第 5.7 节 “部署 Camel K JDBC 集成”
- 第 5.8 节 “部署 Camel K JMS 集成”
- 第 5.9 节 “部署 Camel K Kafka 集成”

5.1. 部署基本 CAMEL K JAVA 集成

本教程介绍了如何在 OpenShift 上的云中运行简单的 Java 集成，对集成应用配置和路由，以及作为 Kubernetes CronJob 运行集成。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须已安装了 Camel K operator 和 **kamel** CLI。请参阅安装 Camel K。
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库。

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-basic**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
   或者，如果您没有安装 VS Code，您可以手动输入 部署基本 Camel K Java 集成 的命令 。

其他资源

- 在 Java 中开发 Camel K 集成

5.2. 部署 CAMEL K SERVERLESS 与 KNATIVE 集成

本教程演示了如何在事件驱动的架构中部署 Camel K 与 OpenShift Serverless 集成。本教程使用 Knative Eventing 代理在 Bitcoin 交易演示中使用事件发布订阅模式进行通信。

本教程还介绍了如何使用 Camel K 集成来与多个外部系统连接到 Knative 事件源。Camel K 集成也使用 Knative Serving 根据需要自动缩放和缩减为零。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K 和 OpenShift Serverless：
  - 安装 Camel K
  - 从 OperatorHub 安装 OpenShift Serverless
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库：

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-knative**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
   如果您没有安装 VS Code，您可以手动输入 部署 Camel K Knative 集成 的命令 。

其他资源

- 关于 Knative Eventing
- 关于 Knative Serving

5.3. 部署 CAMEL K 转换集成

本教程介绍了如何在 OpenShift 上运行 Camel K Java 集成，该集成将 XML 等数据转换为 JSON，并将它存储在 PostgreSQL 等数据库中。

教程示例使用 CSV 文件查询 XML API，并使用收集的数据来构建有效的 GeoJSON 文件，该文件存储在 PostgreSQL 数据库中。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K。请参阅 安装 Camel K。

- 您必须按照教程读主题中的说明安装 Crunchy Postgres for Kubernetes，这在 OpenShift 集群中是必需的。
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库：

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-transformations**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
   如果您没有安装 VS Code，您可以手动输入 部署 Camel K 转换集成的命令 。

其他资源

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

5.4. 部署 CAMEL K SERVERLESS 事件流集成

本教程演示了在 Knative Eventing 中使用 Camel K 和 OpenShift Serverless 进行事件驱动的架构。

本教程介绍了如何在带有 AMQ Broker 集群的 AMQ Streams 集群中安装 Camel K 和 Serverless，以及如何部署事件流项目来运行全局 hazard 警报演示应用程序。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K 和 OpenShift Serverless：
  - 安装 Camel K
  - 从 OperatorHub 安装 OpenShift Serverless
- 您必须按照教程阅读中的说明在 OpenShift 集群上安装额外所需的 Operator：
  - AMQ Streams Operator
  - AMQ Broker Operator
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库。

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-event-streaming**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
或者，如果您没有安装 VS Code，您可以手动输入 部署 Camel K 事件流集成的命令 。

其他资源

- Red Hat AMQ 文档
- OpenShift Serverless 文档

5.5. 部署基于 CAMEL K SERVERLESS API 的集成

本教程演示了在 OpenShift 上使用 Camel K 和 OpenShift Serverless 与 Knative Serving 进行基于 API 的集成，以及使用 3scale API Management 在 OpenShift 上管理 API。

本教程介绍了如何配置基于 Amazon S3 的存储、设计 OpenAPI 定义并运行调用演示 API 端点的集成。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K 和 OpenShift Serverless：
  - 安装 Camel K
  - 从 OperatorHub 安装 OpenShift Serverless
- 您还可以在 OpenShift 系统上安装可选的 Red Hat Integration - 3scale Operator 来管理 API。请参阅使用 Operator 部署 3scale。
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库。

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-api**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
或者，如果您没有安装 VS Code，您可以手动输入 部署 Camel K API 集成 的命令 。

其他资源

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

5.6. 部署 CAMEL K SAAS 集成

本教程介绍了如何在 OpenShift 上运行 Camel K Java 集成，该集成将两个广泛使用的软件作为服务(SaaS)提供商连接。

本教程示例演示了如何使用基于 REST 的 Camel 组件集成 Salesforce 和 ServiceNow SaaS 供应商。在这个简单示例中，每个新的 Salesforce Case 都复制到相应的 ServiceNow Incident 中，其中包含 Salesforce Case Number。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K。请参阅 安装 Camel K。
- 您必须有 Salesforce 登录凭证和 ServiceNow 登录凭证。
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库：

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-saas**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
   如果您没有安装 VS Code，您可以手动输入 部署 Camel K SaaS 集成的命令 。

其他资源

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

5.7. 部署 CAMEL K JDBC 集成

本教程演示了如何通过 JDBC 驱动程序开始使用 Camel K 和 SQL 数据库。本教程介绍了如何将一个集成生成数据设置为 Postgres 数据库（您可以使用您选择的任何相关数据库），以及如何从同一数据库读取数据。

先决条件

- 请参阅 GitHub 中的教程阅读主题。

- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K。
  - 安装 Camel K
- 您必须按照教程读主题中的说明安装 Crunchy Postgres for Kubernetes，这在 OpenShift 集群中是必需的。
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库。

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-jdbc**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
   或者，如果您没有安装 VS Code，您可以手动输入 部署 Camel K JDBC 集成 的命令 。

5.8. 部署 CAMEL K JMS 集成

本教程介绍了如何使用 JMS 连接到消息代理，以便使用和生成消息。有两个示例：

- JMS Sink：本教程演示了如何向 JMS 代理生成消息。
- JMS Source：本教程演示了如何使用 JMS 代理的消息。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K。
  - 安装 Camel K
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库：

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-jms**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。

如果您没有安装 VS Code，您可以手动输入 部署 Camel K JMS 集成 的命令 。

其他资源

- JMS Sink
- JMS Source

5.9. 部署 CAMEL K KAFKA 集成

本教程演示了如何在 Apache Kafka 中使用 Camel K。本教程介绍了如何通过 Red Hat OpenShift Streams for Apache Kafka 设置 Kafka 主题，并将其与 Camel K 结合使用。

先决条件

- 请参阅 GitHub 中的教程阅读主题。
- 您必须具有集群管理员访问 OpenShift 集群才能安装 Camel K。
  - 安装 Camel K
- Visual Studio (VS) Code 是可选的，但推荐提供最佳的开发人员体验。请参阅 设置 Camel K 开发环境。

流程

1. 克隆教程 Git 存储库：

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

2. 在 VS Code 中，选择 **File → Open Folder → camel-k-example-kafka**。
3. 在 VS Code 导航树中，单击 **readme.md** 文件。这会在 VS Code 中打开一个新标签页，以显示教程说明。
4. 按照教程说明进行操作。
   如果您没有安装 VS Code，您可以手动输入 部署 Camel K Kafka 集成 的命令 。