Red Hat Fuse 7.13

OpenShift 上的 Fuse 指南

在 OpenShift 中安装和管理红帽 Fuse，在 OpenShift 中开发和部署 Fuse 应用程序

Last Updated: 2024-06-14

Red Hat Fuse 7.13 OpenShift 上的 Fuse 指南

在 OpenShift 中安装和管理红帽 Fuse，在 OpenShift 中开发和部署 Fuse 应用程序

法律通告

摘要

在 OpenShift 上使用 Fuse 指南

目录

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。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看我们的 CTO Chris Wright 信息。

Red Hat Fuse on OpenShift 可让您在 OpenShift Container Platform 上部署 Fuse 应用程序。

第 1 章 开始前

Release Notes

有关本发行版本的重要信息，请参阅发行注记。

版本兼容性和支持

如需了解版本兼容性和支持的详细信息，请参阅 Red Hat JBoss Fuse 支持的配置页面。

支持 Windows O/S

Windows O/S 完全支持 OpenShift 上 Fuse 的开发人员工具(**oc** client and Container Development Kit)。Linux 命令行语法中显示的示例也可以在 Windows O/S 上工作，只要它们被正确修改以遵循 Windows 命令行语法。

1.1. 比较：OPENSHIFT 上的 FUSE 独立和 FUSE

有几个主要功能区别：

- 使用 OpenShift 上的 Fuse 进行应用部署由应用程序以及打包在容器镜像内所需的所有运行时组件组成。应用程序不会与 Fuse Standalone 一起部署到运行时，应用程序镜像本身是一个通过 OpenShift 部署和管理的完整运行时环境。
- OpenShift 环境中的修补与 Fuse Standalone 不同，因为每个应用程序镜像都是完整的运行时环境。为应用补丁，在 OpenShift 中重新构建并重新部署应用镜像。OpenShift 核心管理功能允许滚动升级和并行部署来在升级过程中维护应用程序的可用性。
- Fuse 中提供的 Fabric 提供的置备和集群功能已被 Kubernetes 和 OpenShift 中的等效功能替代。不需要创建或配置单独的子容器，因为 OpenShift 会作为部署和扩展应用的一部分自动执行此操作。
- Fabric 端点在 OpenShift 环境中不使用。必须使用 Kubernetes 服务。
- 消息传递服务使用 A-MQ for OpenShift 镜像创建和管理，而不直接包含在 Karaf 容器中。*OpenShift 上的 Fuse 提供了 camel-amq 组件的增强版本，它允许通过 Kubernetes 与 OpenShift 中的消息传递服务无缝连接。*
- *强烈建议不要使用 Karaf shell 运行 Karaf 实例的实时更新，因为如果应用容器重启或扩展，则不会保留更新。这是不可变架构的基本原则，在 OpenShift 中实现可扩展性和灵活性至关重要。*
- *红帽支持直接链接到红帽 Fuse 组件的 Maven 依赖项。不支持由用户引入的第三方 Maven 依赖项。*
- *Apache Karaf 微容器中没有 SSH 代理，因此您无法使用 bin/client 控制台客户端连接到它。*
- *OpenShift 应用的 Fuse 中的协议兼容性和 Camel 组件：基于非 HTTP 的通信必须使用 TLS 和 SNI，才能从 OpenShift 外部路由到 Fuse 服务(Camel 消费者端点)。*

第 2 章 管理员入门

如果您是 OpenShift 管理员，您可以通过以下方法为 OpenShift 部署上的 Fuse 准备 OpenShift 集群：

1. 使用 **registry.redhat.io** 配置身份验证。
2. 在 OpenShift 镜像和模板上安装 Fuse。

2.1. 使用 REGISTRY.REDHAT.IO 身份验证。

在 OpenShift 上部署 Fuse 容器镜像之前，使用 **registry.redhat.io** 配置身份验证。

先决条件

- 集群管理员对 OpenShift Container Platform 集群的访问权限。
- 已安装 OpenShift **oc** 客户端工具。如需了解更多详细信息，请参阅 OpenShift CLI 文档。

流程

1. 以管理员身份登录您的 OpenShift 集群：

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

2. 打开您要在其中部署 Fuse 的项目：

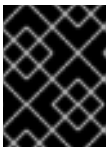
```
oc project myproject
```

3. 使用您的红帽客户门户网站帐户创建一个 **docker-registry** secret，将 **PULL_SECRET_NAME** 替换为 **psi-internal-registry** 创建：

```
oc create secret docker-registry psi-internal-registry \
  --docker-server=docker-registry.redhat.io \
  --docker-username=CUSTOMER_PORTAL_USERNAME \
  --docker-password=CUSTOMER_PORTAL_PASSWORD \
  --docker-email=EMAIL_ADDRESS
```

您应该看到以下输出：

```
secret/psi-internal-registry created
```



重要

您必须在要向 **registry.redhat.io** 进行身份验证的每个 OpenShift 项目命名空间中创建此 **docker-registry** secret。

4. 将机密链接到您的服务帐户，以使用机密拉取镜像。以下示例使用 **default** 服务帐户、**builder** 服务帐户和 **deployer** 服务帐户：

```
oc secrets link default psi-internal-registry
oc secrets link default psi-internal-registry --for=pull
```

```
oc secrets link builder psi-internal-registry
oc secrets link builder psi-internal-registry --for=pull
oc secrets link deployer psi-internal-registry
oc secrets link deployer psi-internal-registry --for=pull
```

服务帐户名称必须与 OpenShift 容器集使用的名称匹配。

注意

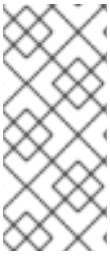
如果您不想使用红帽用户名和密码来创建 pull secret，您可以使用 registry 服务帐户创建身份验证令牌。

其他资源

有关向容器镜像进行身份验证的更多详情：

- 红帽容器镜像身份验证
- Red Hat registry 服务帐户

2.2. 在 OPENSHIFT 4.X 服务器上安装 FUSE 镜像流和模板



注意

在 Fuse 7.13 中，不支持在 IBM Power Systems、IBM Z 和 LinuxONE 上的 OpenShift 镜像流和模板上安装 Fuse。

IBM Power Systems、IBM Z 和 LinuxONE 仅支持在 OpenShift Operator 上使用 Fuse 安装的组件。

OpenShift Container Platform 4.x 使用 Samples Operator （在 OpenShift 命名空间中运行），安装和更新基于 Red Hat Enterprise Linux (RHEL)的 OpenShift Container Platform 镜像流和模板。在 OpenShift 镜像流和模板上安装 Fuse：

- **重新配置 Samples Operator**
- **将 Fuse 镜像流和模板添加到 Skipped Imagestreams 和 Skipped Templates 字段中。**
  - **跳过的 Imagestreams：位于 Samples Operator 清单中的 Imagestreams，但集群管理员希望 Operator 忽略或不管理。**
  - **跳过的模板：位于 Samples Operator 清单中的模板，但集群管理员希望 Operator 忽略或不予管理。**

先决条件

- **您可以访问 OpenShift 服务器。**
- **您已将身份验证配置为 registry.redhat.io。**

流程

1. **启动 OpenShift 4 服务器。**

2. *以管理员身份登录 OpenShift 服务器。*

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

3. *验证您是否使用您为其创建 docker-registry secret 的项目。*

```
oc project openshift
```

4. *查看 Samples operator 的当前配置。*

```
oc get configs.samples.operator.openshift.io -n openshift-cluster-samples-operator -o yaml
```

5. *配置 Samples operator 以忽略添加的 fuse 模板和镜像流。*

```
oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

6. *添加 Fuse imagestreams Skipped Imagestreams 部分，并将 Fuse 和 Spring Boot 2 模板添加到 Skipped Templates 部分。*

```
[...]
spec:
  architectures:
  - x86_64
  managementState: Managed
  skippedImagestreams:
  - fuse-console-rhel8
  - fuse-eap-openshift-jdk8-rhel7
  - fuse-eap-openshift-jdk11-rhel8
  - fuse-java-openshift-rhel8
  - fuse-java-openshift-jdk11-rhel8
  - fuse-karaf-openshift-rhel8
  - fuse-karaf-openshift-jdk11-rhel8
  - fuse-apicurito-generator-rhel8
  - fuse-apicurito-rhel8
  skippedTemplates:
  - s2i-fuse713-eap-camel-amq
  - s2i-fuse713-eap-camel-cdi
  - s2i-fuse713-eap-camel-cxf-jaxrs
  - s2i-fuse713-eap-camel-cxf-jaxws
  - s2i-fuse713-karaf-camel-amq
  - s2i-fuse713-karaf-camel-log
  - s2i-fuse713-karaf-camel-rest-sql
  - s2i-fuse713-karaf-cxf-rest
  - s2i-fuse713-spring-boot-2-camel-amq
  - s2i-fuse713-spring-boot-2-camel-config
  - s2i-fuse713-spring-boot-2-camel-drools
  - s2i-fuse713-spring-boot-2-camel-infinispan
  - s2i-fuse713-spring-boot-2-camel-rest-3scale
  - s2i-fuse713-spring-boot-2-camel-rest-sql
  - s2i-fuse713-spring-boot-2-camel
  - s2i-fuse713-spring-boot-2-camel-xa
  - s2i-fuse713-spring-boot-2-camel-xml
  - s2i-fuse713-spring-boot-2-cxf-jaxrs
```

```
  - s2i-fuse713-spring-boot-2-cxf-jaxws
  - s2i-fuse713-spring-boot-2-cxf-jaxrs-xml
  - s2i-fuse713-spring-boot-2-cxf-jaxws-xml
```

7. **在 OpenShift 镜像流上安装 Fuse。**

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001

oc create -n openshift -f ${BASEURL}/fis-image-streams.json
```

8. **在 OpenShift 快速启动模板上安装 Fuse：**

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json ;
 do oc create -n openshift -f \
 ${BASEURL}/quickstarts/${template}
 done
```

9. **安装 Spring Boot 2 Quickstart 模板：**

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
 spring-boot-2-cxf-jaxrs-template.json \
 spring-boot-2-cxf-jaxws-template.json \
 spring-boot-2-cxf-jaxrs-xml-template.json \
 spring-boot-2-cxf-jaxws-xml-template.json ;
 do oc create -n openshift -f \
 ${BASEURL}/quickstarts/${template}
 done
```

10. **（可选）查看 OpenShift 模板上安装的 Fuse：**

```
oc get template -n openshift
```

2.3. 在 OPENSHIFT 4.X 上安装 API DESIGNER

Red Hat Fuse on OpenShift 提供了 API Designer，它是一个基于 Web 的 API 设计工具，可用于设计 REST API。API Designer Operator 简化了 OpenShift Container Platform 4.x 上的 API Designer 的安装和升级。

作为 OpenShift 管理员，您要将 API Designer Operator 安装到 OpenShift 项目（命名空间）。安装 Operator 后，Operator 会在所选命名空间中运行。但是，要使 API Designer 作为服务（作为 OpenShift 管理员）提供，或者开发人员必须创建一个 API Designer 实例。API Designer 服务提供用于访问 API Designer Web 控制台的 URL。

先决条件

- 具有 OpenShift 集群的管理员访问权限。
- 您已将身份验证配置为 **registry.redhat.io**。

流程

1. 启动 OpenShift 4.x 服务器。
2. 在 Web 浏览器中，导航到浏览器中的 OpenShift 控制台。使用您的凭证登录到控制台。
3. 点 Operators，然后点 OperatorHub。
4. 在搜索字段中，键入 API Designer。
5. 点 Red Hat Integration - API Designer卡。Red Hat Integration - API DesignerOperator 安装页面将打开。
6. 点 Install。此时会打开 Install Operator 页面。
   a. 对于 更新频道，请选择 fuse-console-7.13.x。
   b. 对于 Installation 模式，请从集群中的命名空间列表中选择命名空间（项目）。
   c. 对于 Approval Strategy，选择 Automatic 或 Manual 来配置 OpenShift 如何处理 API Designer Operator 的更新。
      - 如果选择 Automatic 更新，当 API Designer Operator 的新版本可用时，OpenShift Operator Lifecycle Manager (OLM)会自动升级 API Designer 的运行实例，而无需人为干预。
      - 如果选择手动 更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才能将 API Designer Operator 更新至新版本。
7. 点 Install 使 API Designer Operator 可供指定命名空间（项目）使用。
8. 要验证 API Designer 是否已安装到项目中，点 Operators，然后点 Installed Operators 在列表中看到 Red Hat Integration - API Designer。

后续步骤

安装 API Designer Operator 后，API Designer 必须通过创建 API Designer 实例来作为服务添加到 OpenShift 项目中。此任务可以通过两种方式完成：

- OpenShift 管理员可以按照 第 2.3.1 节 “将 API Designer 作为一个服务添加到 OpenShift 4.x 项目中” 中的步骤操作。
- OpenShift 开发人员可按照设计 API 中所述的步骤操作。
  API Designer 服务提供用于访问 API Designer Web 控制台的 URL。

2.3.1. 将 API Designer 作为一个服务添加到 OpenShift 4.x 项目中

在 OpenShift 4.x 项目中安装了 API Designer 操作器后，您可以（或一个 OpenShift 开发人员）将它添加为 OpenShift 项目的服务。API Designer 服务提供开发人员用来访问 API Designer Web 控制台的 URL。

注意

如需了解 OpenShift 开发人员要作为服务添加到 OpenShift 4.x 项目的步骤，请参阅 设计 API。

先决条件

- 具有 OpenShift 集群的管理员访问权限。
- API Designer 操作器安装到当前的 OpenShift 项目中。

流程

1. 在 OpenShift Web 控制台中，点 Operators，然后点 Installed Operators。
2. 在 Name 列中，点 Red Hat Integration - API Designer。
3. 在 Provided APIs 下，点 Create instance。
   此时会打开 API Designer 实例的最小起始模板的默认表单。接受默认值，或者（可选）编辑它们。
4. 单击 Create 以创建一个新的 apicurito-service。OpenShift 为新的 API Designer 服务启动 pod、服务和其他组件。
5. 验证 API Designer 服务是否可用：
   a. 点 Operators，然后点 Installed Operators。
   b. 在 Provided APIs 列中，点 Apicurito CRD。
      在 Operator Details 页面中，会列出 apicurito-service。
6. 打开 API Designer：
   a. 选择 Networking > Routes。
   b. 确保选择了正确的项目。
   c. 在 apicurito-service-ui 行的 Location 列中，单击 URL。
      API Designer Web 控制台在新的浏览器标签页中打开。

2.3.2. 在 OpenShift 4.x 上升级 API Designer

Red Hat OpenShift 4.x 处理对 Operator 的更新，包括 Red Hat Fuse operator。如需更多信息，请参阅 Operator OpenShift 文档。

反过来，Operator 更新可触发应用程序升级。应用程序升级如何根据应用程序的配置方式的不同而有所不同。

对于 API Designer 应用程序，升级 API Designer 操作器时，OpenShift 会自动升级集群上的任何 API 设计程序应用程序。

注意

从 API Designer 7.8 升级到 API Designer 7.9 时，正常的 Operator 升级过程无法正常工作。要将 API Designer 从 Fuse 7.8 升级到 Fuse 7.9，您必须删除 7.8 API Designer 操作器，然后安装 7.9 API Designer operator。

2.3.3. API Designer 的 metering 标签

您可以使用 OpenShift Metering Operator 分析已安装的 API Designer 操作器、UI 组件和代码生成器，以确定您是否符合红帽订阅。如需有关 Metering 的更多信息，请参阅 OpenShift 文档。

下表列出了 API Designer 的 metering 标签。

表 2.1. API Designer Metering 标签

标签	可能的值
com.company	**Red_Hat**
rht.prod_name	**Red_Hat_Integration**
rht.prod_ver	**7.13**
rht.comp	**fuse**
rht.comp_ver	**7.13**
rht.subcomp	**fuse-apicurito** **apicurito-service-ui** **apicurito-service-generator**
rht.subcomp_t	**infrastructure**

例子

- *API Designer 操作器 示例：*

```
apicurito-operator
com.company: Red_Hat
rht.prod_name: Red_Hat_Integration
rht.prod_ver: 7.13
rht.comp: Fuse
rht.comp_ver: 7.13
rht.subcomp: fuse-apicurito
rht.subcomp_t: infrastructure
```

- *API Designer UI 组件示例：*

```
com.company: Red_Hat
rht.prod_name: Red_Hat_Integration
```

```
rht.prod_ver: 7.13
rht.comp: Fuse
rht.comp_ver: 7.13
rht.subcomp: apicurito-service-ui
rht.subcomp_t: infrastructure
```

- API Designer Generator 组件示例：

```
com.company: Red_Hat
rht.prod_name: Red_Hat_Integration
rht.prod_ver: 7.13
rht.comp: Fuse
rht.comp_ver: 7.13
rht.subcomp: apicurito-service-generator
rht.subcomp_t: infrastructure
```

2.3.4. 在受限环境中安装 API Designer 的注意事项

在受限环境中安装的 OpenShift 集群默认无法访问红帽提供的 OperatorHub 源，因为这些远程源需要足够的互联网连接。在这种情况下，要安装 API designer Operator，您必须完成以下先决条件：

- 为 Operator Lifecycle Manager (OLM)禁用默认远程 OperatorHub 源。
- 使用有完全互联网访问的工作站来创建 OperatorHub 内容的本地镜像。
- 将 OLM 配置为，从本地源而不是默认的远程源安装和管理 Operator。

如需更多信息，请参阅 OpenShift 文档中的 *在受限网络中使用 Operator Lifecycle Manager* 部分。创建 OperatorHub 的本地镜像后，您可以执行以下步骤。

- 按照在 *OpenShift 4.x 上安装 API Designer* 所述，使用镜像的 OperatorHub 安装 API Designer。
- 根据将 API Designer 作为服务添加到 *OpenShift 4.x 项目中所述，添加 API Designer*作为服务。

2.4. 在 OPENSHIFT 4.X 中设置 FUSE 控制台

在 OpenShift 4.x 上，设置 Fuse 控制台涉及安装和部署它。您有安装和部署 Fuse 控制台的这些选项：

- *第 2.4.1 节 “使用 OperatorHub 在 OpenShift 4.x 上安装和部署 Fuse 控制台”*
  您可以使用 Fuse Console Operator 安装和部署 Fuse 控制台，以便它能够访问特定命名空间中的 Fuse 应用程序。Operator 会为您处理 Fuse 控制台的安全。
- *第 2.4.2 节 “使用命令行在 OpenShift 4.x 上安装和部署 Fuse 控制台”*
  您可以使用命令行和其中一个 Fuse 控制台模板来安装和部署 Fuse 控制台，以便它可以访问 OpenShift 集群或特定命名空间中的多个命名空间中的 Fuse 应用程序。在部署前，您必须通过生成客户端证书来保护 Fuse 控制台。

另外，您可以为 Fuse 控制台自定义基于角色的访问控制(RBAC)，如 *第 2.4.3 节 “OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制”* 所述。

2.4.1. 使用 OperatorHub 在 OpenShift 4.x 上安装和部署 Fuse 控制台

要在 OpenShift 4.x 上安装 Fuse 控制台，您可以使用 OpenShift OperatorHub 提供的 Fuse Console Operator。要部署 Fuse 控制台，请创建一个已安装的 Operator 实例。

先决条件

- *您已使用 registry.redhat.io 配置身份验证，如为容器镜像 使用 registry.redhat.io 验证中所述。*
- *如果要为 Fuse 控制台自定义基于角色的访问控制(RBAC)，则必须在安装 Fuse Console Operator 的同一 OpenShift 命名空间中有一个 RBAC 配置映射文件。如果要使用默认的 RBAC 行为，如 OpenShift 4.x 上 Fuse Console 的基于角色的访问控制中所述，您不需要提供配置映射文件。*

流程

安装和部署 Fuse 控制台：

1. *以具有集群管理员访问权限的用户身份登录 Web 浏览器中的 OpenShift 控制台。*
2. *点 Operators，然后点 OperatorHub。*
3. *在搜索字段窗口中，键入 Fuse Console 来过滤操作器列表。*
4. *点 Fuse Console Operator。*
5. *在 Fuse Console Operator 安装窗口中，点 Install。*

   Create Operator Subscription 表单将打开。

   - *对于 更新频道，请选择 7.13.x。*
   - *对于 Installation Mode，接受默认值（集群中的特定命名空间）。*

     请注意，在安装 Operator 后，当部署 Fuse 控制台时，您可以选择监控集群中的所有命名空间中的应用程序，或者仅在安装了 Fuse Console operator 的命名空间中监控应用程序。
   - *对于 Installed Namespace，选择要在其中安装 Fuse Console Operator 的命名空间。*
   - *对于 Update Approval，您可以选择 Automatic 或 Manual 来配置 OpenShift 如何处理 Fuse Console Operator 的更新。*

- 如果选择 Automatic 更新，当有新版本的 Fuse Console Operator 可用时，OpenShift Operator Lifecycle Manager (OLM)将自动升级 Fuse 控制台的运行实例，而无需人为干预。

- 如果选择手动 更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Fuse Console Operator 更新至新版本。

6. 点 Install。

   OpenShift 在当前命名空间中安装 Fuse Console Operator。

7. 要验证安装，点 Operators，然后点 Installed Operators。您可以在操作器列表中看到 Fuse 控制台。

8. 使用 OpenShift Web 控制台部署 Fuse 控制台：

   a. 在 Installed Operators 列表中，单击 Name 列下的 Fuse Console。

   b. 在 Provided APIs 下的 Operator Details 页面中，点 Create Instance。

      接受配置默认值或选择性地编辑它们。

      对于 Replicas，如果要提高 Fuse 控制台性能（例如在高可用性环境中），您可以增加分配给 Fuse 控制台的 pod 数量。

      对于 Rbac （基于角色的访问控制），只有在 config Map 字段中指定一个值，如果要自定义默认的 RBAC 行为，并且是否安装了 Fuse Console Operator 的命名空间中已存在 ConfigMap 文件。如需有关 RBAC 的更多信息，请参阅 OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制。

      对于 Nginx，请参阅 Fuse Console Operator 安装的性能调优。

c.

点 Create。

Fuse Console Operator Details 页面将打开并显示部署状态。

9.

打开 Fuse 控制台：

a.

对于 命名空间 部署：在 OpenShift Web 控制台中，打开安装 Fuse Console operator 的项目，然后选择 Overview。在 Project Overview 页面中，向下滚动到 Launcher 部分，再单击 Fuse Console 链接。

对于 集群部署，在 OpenShift Web 控制台的标题栏中，点网格图标()。在弹出菜单中，单击 红帽应用程序 下的 Fuse 控制台 URL 链接。

b.

登录到 Fuse 控制台。

浏览器中打开了 Authorize Access 页面，其中列出了所需权限。

c.

点击 Allow selected permissions。

Fuse 控制台在浏览器中打开，并显示您有权访问的 Fuse 应用程序 pod。

10.

对于您要查看的应用程序，点 Connect。

此时将打开一个新浏览器窗口，显示 Fuse 控制台中的应用程序。

2.4.2. 使用命令行在 OpenShift 4.x 上安装和部署 Fuse 控制台

在 OpenShift 4.x 中，您可以选择其中一个部署选项来从命令行安装和部署 Fuse 控制台：

- 集群 - Fuse 控制台可以发现并连接到在 OpenShift 集群上的多个命名空间（项目）间部署的 Fuse 应用程序。要部署此模板，您必须具有 OpenShift 集群的管理员角色。
- 具有基于角色的访问控制的集群 - 具有可配置的基于角色的访问控制(RBAC)的集群模板。如

需更多信息，请参阅 OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制。

- 命名空间 - Fuse 控制台可以访问特定的 OpenShift 项目（命名空间）。要部署此模板，您必须具有 OpenShift 项目的管理员角色。

- 带有基于角色的访问控制的命名空间 - 带有可配置 RBAC 的命名空间模板。如需更多信息，请参阅 OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制。

要查看 Fuse 控制台模板的参数列表，请运行以下命令：

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-namespace-os4.json
```

先决条件

- 在安装和部署 Fuse 控制台前，您必须生成一个使用服务签名证书颁发机构签名的客户端证书，如生成证书以保护 OpenShift 4.x 上的 Fuse 控制台安全 中所述。

- 有 OpenShift 集群的集群管理员角色。

- 您已使用 registry.redhat.io 配置身份验证，如 为容器镜像 使用 registry.redhat.io 验证 中所述。

- 已安装 Fuse Console 镜像流（以及其他 Fuse 镜像流），如 在 OpenShift 4.x 服务器上安装 Fuse 镜像流和模板 中所述。

流程

1. 使用以下命令验证是否安装了 Fuse Console 镜像流，以检索所有模板的列表：

```
oc get template -n openshift
```

2. 另外，如果要使用新发行标签更新已安装的镜像流，请使用以下命令将 Fuse Console 镜像导入到 openshift 命名空间：

```
oc import-image fuse7/fuse-console-rhel8:1.10 --from=registry.redhat.io/fuse7/fuse-console-rhel8:1.10 --confirm -n openshift
```

3. 运行以下命令来获取 **Fuse Console APP_NAME** 值：

```
oc process --parameters -f TEMPLATE-FILENAME
```

其中 **TEMPLATE-FILENAME** 是以下模板之一：

- 集群模板：

  https://github.com/jboss-fuse/application-templates/blob/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-cluster-os4.json

- 使用可配置的 **RBAC** 的集群模板：

  https://github.com/jboss-fuse/application-templates/blob/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-cluster-rbac.yml

- 命名空间模板：

  https://github.com/jboss-fuse/application-templates/blob/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-namespace-os4.json

- 带有可配置的 **RBAC** 的命名空间模板：

  https://github.com/jboss-fuse/application-templates/blob/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-namespace-rbac.yml

例如，对于使用可配置的 **RBAC** 的集群模板，请运行以下命令：

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-cluster-rbac.yml
```

4. 从您在 *OpenShift 4.x 上保护 Fuse 控制台* 生成的证书，使用以下命令创建 **secret** 并将其挂

载到 Fuse 控制台中（其中 APP_NAME 是 Fuse Console 应用程序的名称）。

```
oc create secret tls APP_NAME-tls-proxying --cert server.crt --key server.key
```

5. *运行以下命令，基于 Fuse Console 模板的本地副本创建新应用程序（其中 myproject 是 OpenShift 项目的名称，mytemp 是包含 Fuse Console 模板的本地目录的路径，myhost 是用于访问 Fuse 控制台的主机名：*

- *对于集群模板：*

```
oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-cluster-os4.json  -p ROUTE_HOSTNAME=myhost
```

- *对于使用 RBAC 模板的集群：*

```
oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-cluster-rbac.yml -p ROUTE_HOSTNAME=myhost
```

- *对于命名空间模板：*

```
oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-namespace-os4.json
```

- *对于使用 RBAC 模板的命名空间：*

```
oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fuse-console-namespace-rbac.yml
```

6. *要配置 Fuse 控制台使其能够打开 OpenShift Web 控制台，请运行以下命令来设置 OPENSHIFT_WEB_CONSOLE_URL 环境变量：*

```
oc set env dc/${APP_NAME} OPENSHIFT_WEB_CONSOLE_URL=`oc get -n openshift-config-managed cm console-public -o jsonpath={.data.consoleURL}`
```

7. *运行以下命令，获取 Fuse 控制台部署的状态和 URL：*

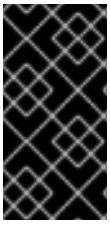
```
oc status
```

8. 要从浏览器访问 Fuse 控制台，请使用在第 7 步中返回的 URL （例如 https://fuse-console.192.168.64.12.nip.io）。

2.4.2.1. 生成证书来保护 OpenShift 4.x 上的 Fuse 控制台

在 OpenShift 4.x 上，若要在 Fuse 控制台代理和 Jolokia 代理之间保持连接，必须在部署 Fuse 控制台前生成客户端证书。服务签名证书颁发机构私钥必须用于为客户端证书签名。

只有在您要使用命令行安装和部署 Fuse 控制台时，才必须 按照以下步骤操作。如果使用 Fuse Console Operator，它会为您处理此任务。



重要

您必须为每个 OpenShift 集群生成并签署单独的客户端证书。对于多个集群，不要使用相同的证书。

先决条件

- 有 集群管理员 对 OpenShift 集群的访问权限。
- 如果您要为多个 OpenShift 集群生成证书，并且之前在当前目录中为不同的集群生成证书，请执行以下操作之一以确保为当前集群生成不同的证书：
  - 从当前目录中删除现有证书文件（如 ca.crt、ca.key 和 ca.srl）。
  - 更改到其他工作目录。例如，如果您的当前工作目录命名为 cluster1，请创建一个新的 cluster2 目录，并将工作目录改为其中：

```
mkdir ../cluster2
```

```
cd ../cluster2
```

流程

1. 以具有集群管理员访问权限的用户身份登录 OpenShift：

```
oc login -u <user_with_cluster_admin_role>
```

2. 执行以下命令来检索服务签名证书颁发机构密钥：

- 检索证书：

```
oc get secrets/signing-key -n openshift-service-ca -o "jsonpath={.data['tls\.crt']}" | base64 --decode > ca.crt
```

- 检索私钥：

```
oc get secrets/signing-key -n openshift-service-ca -o "jsonpath={.data['tls\.key']}" | base64 --decode > ca.key
```

3. 生成客户端证书，如 Kubernetes 证书管理 中所述，使用 easyrsa、openssl 或 cfssl。

以下是使用 openssl 的示例命令：

a. 生成私钥：

```
openssl genrsa -out server.key 2048
```

b. 编写 CSR 配置文件。

```
cat <<EOT >> csr.conf
 [ req ]
 default_bits = 2048
 prompt = no
 default_md = sha256
 distinguished_name = dn

 [ dn ]
 CN = fuse-console.fuse.svc

 [ v3_ext ]
 authorityKeyIdentifier=keyid,issuer:always
```

```
  keyUsage=keyEncipherment,dataEncipherment,digitalSignature
  extendedKeyUsage=serverAuth,clientAuth
EOT
```

在这里，CN 参数中的值指的是应用程序使用的命名空间。

c. 生成 CSR：

```
openssl req -new -key server.key -out server.csr -config csr.conf
```

d. 发布签名证书：

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt
-days 10000 -extensions v3_ext -extfile csr.conf
```

后续步骤

您需要此证书来为 Fuse 控制台创建 secret，如 使用命令行在 OpenShift 4.x 上安装和部署 Fuse 控制台 中所述。

2.4.3. OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制

Fuse 控制台提供基于角色的访问控制(RBAC)，它根据 OpenShift 提供的用户授权推断访问。在 Fuse 控制台中，RBAC 决定用户在 pod 上执行 MBean 操作。

如需有关 OpenShift 授权的信息，请参阅 OpenShift 文档中的使用 RBAC 定义和应用权限 部分。

当使用 Operator 在 OpenShift 上安装 Fuse 控制台时，默认启用基于角色的访问控制。

如果要使用模板安装 Fuse 控制台，为它实施基于角色的访问权限，则必须使用可通过 RBAC 配置的模板之一(fuse-console-cluster-rbac.yml 或 fuse-console-namespace-rbac.yml)来安装 Fuse 控制台，如 使用命令行在 OpenShift 4.x 上安装和部署 Fuse 控制台。

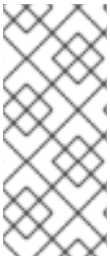
Fuse 控制台 RBAC 利用 OpenShift 中容器集资源的 操作动词 访问，以确定用户在 Fuse 控制台中对 pod 的 MBean 操作的访问。默认情况下，Fuse 控制台有两个用户角色：

- *admin*

  如果用户可以在 OpenShift 中更新 pod，则用户会限制 Fuse 控制台的 admin 角色。用户可以在 Fuse 控制台中为 pod 执行写入 MBean 操作。

- *viewer*

  如果用户可以在 OpenShift 中获取 pod，则用户会限制 Fuse 控制台的 viewer 角色。用户可以在 Fuse 控制台中为 pod 执行 只读 MBean 操作。



注意

如果您使用非RBAC 模板安装 Fuse 控制台，则只有授予 pod 资源 更新 动词的 OpenShift 用户有权执行 Fuse Console MBeans 操作。授予 pod 资源的 get 动词的 用户可以查看 pod，但不能执行任何 Fuse 控制台操作。

其他资源

- *确定 OpenShift 4.x 上 Fuse 控制台的访问角色*
- *在 OpenShift 4.x 上自定义对 Fuse 控制台进行基于角色的访问权限*
- *在 OpenShift 4.x 上为 Fuse 控制台禁用基于角色的访问控制*

2.4.3.1. 确定 OpenShift 4.x 上 Fuse 控制台的访问角色

Fuse 控制台基于角色的访问控制是从用户的 pod 的 OpenShift 权限中推断出来的。要确定向特定用户授予的 Fuse 控制台访问角色，请获取授予 Pod 用户的 OpenShift 权限。

先决条件

- *您知道用户名。*
- *您知道 pod 的名称。*

流程

- 要确定用户是否有用于 pod 的 Fuse Console admin 角色，请运行以下命令来查看用户是否可以更新 OpenShift 上的 pod：

```
oc auth can-i update pods/<pod> --as <user>
```

如果响应是 yes，则用户具有 pod 的 Fuse 控制台 管理员角色。用户可以在 Fuse 控制台中为 pod 执行写入 MBean 操作。

- 要确定用户是否有用于 pod 的 Fuse Console viewer 角色，请运行以下命令来查看用户是否可以在 OpenShift 上获取 pod：

```
oc auth can-i get pods/<pod> --as <user>
```

如果响应是 yes，则用户具有 pod 的 Fuse Console viewer 角色。用户可以在 Fuse 控制台中为 pod 执行 只读 MBean 操作。根据上下文，Fuse 控制台可防止具有 viewer 角色的用户执行写入 MBean 操作，方法是禁用一个选项，或者在用户尝试写入 MBean 操作时显示"operation not allowed for this user"消息。

如果没有，则用户不绑定到任何 Fuse 控制台角色，用户无法在 Fuse 控制台中查看 pod。

其他资源

- OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制
- 在 OpenShift 4.x 上自定义对 Fuse 控制台进行基于角色的访问权限
- 在 OpenShift 4.x 上为 Fuse 控制台禁用基于角色的访问控制

2.4.3.2. 在 OpenShift 4.x 上自定义对 Fuse 控制台进行基于角色的访问权限

如果您使用 OperatorHub 安装 Fuse 控制台，则默认启用 基于角色的访问控制(RBAC)，如 OpenShift 4.x 上 Fuse Console 的基于角色的访问控制所述。如果要自定义 Fuse Console RBAC 行为，在部署 Fuse 控制台前，您必须提供 ConfigMap 文件（定义自定义 RBAC 行为）。您必须将自定义 ConfigMap 文件放在安装 Fuse Console Operator 的同一命名空间中。

如果您使用命令行模板安装 Fuse 控制台，则 deployment-cluster-rbac.yml 和 deployment-namespace-rbac.yml 模板会创建一个包含配置文件(ACL.yml)的 ConfigMap。配置文件定义 MBean 操作允许的角色。

前提条件

- *您可以使用 OperatorHub 或其中一个 Fuse 控制台 RBAC 模板(deployment-cluster-rbac.yml 或 deployment-namespace-rbac.yml)安装 Fuse 控制台。*

流程

自定义 Fuse 控制台 RBAC 角色：

1. *如果使用命令行安装 Fuse 控制台，安装模板会包含默认的 ConfigMap 文件，以便您可以跳过下一步。*

   如果您在部署 Fuse 控制台前，使用 OperatorHub 安装 Fuse 控制台，则创建一个 RBAC ConfigMap：

   a. *确保当前的 OpenShift 项目是您要安装 Fuse 控制台的项目。例如，如果您要在 fusetest 项目中安装 Fuse 控制台，请运行以下命令：*

   ```
   oc project fusetest
   ```

   b. *要从模板创建 Fuse Console RBAC ConfigMap 文件，请运行以下命令：*

   ```
   oc process -f {sb2-templates-base-url}/fuse-console-operator-rbac.yml -p
   APP_NAME=fuse-console | oc create -f -
   ```

2. *运行以下命令，在编辑器中打开 ConfigMap：*

   ```
   oc edit cm $APP_NAME-rbac
   ```

   例如：

   ```
   oc edit cm fuse-console-rbac
   ```

3. 编辑该文件。

4. 保存文件以使改变生效。OpenShift 会自动重启 Fuse 控制台 Pod。

其他资源

- OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制
- 确定 OpenShift 4.x 上 Fuse 控制台的访问角色
- 在 OpenShift 4.x 上为 Fuse 控制台禁用基于角色的访问控制

2.4.3.3. 在 OpenShift 4.x 上为 Fuse 控制台禁用基于角色的访问控制

如果您使用命令行安装 Fuse 控制台，并且指定了 Fuse 控制台 RBAC 模板之一，Fuse Console 的 HAWTIO_ONLINE_RBAC_ACL 环境变量会将基于角色的访问控制(RBAC) ConfigMap 配置文件路径传递给 OpenShift 服务器。如果没有指定 HAWTIO_ONLINE_RBAC_ACL 环境变量，则禁用 RBAC 支持，并且只有被授予 pod 资源(OpenShift 中) 更新 动词的用户才会在 Fuse 控制台中调用 pod 的 MBeans 操作。

请注意，当您使用 OperatorHub 安装 Fuse 控制台时，默认启用基于角色的访问控制，并且不会应用 HAWTIO_ONLINE_RBAC_ACL 环境变量。

前提条件

您可以使用命令行安装 Fuse 控制台，并指定其中一个 Fuse 控制台 RBAC 模板(deployment-cluster-rbac.yml 或 deployment-namespace-rbac.yml)。

流程

为 Fuse 控制台禁用基于角色的访问控制：

1. 在 OpenShift 中，编辑 Fuse 控制台的 Deployment Config 资源。

2. 删除整个 HAWTIO_ONLINE_RBAC_ACL 环境变量定义。

（请注意，仅清除其值并不够）。

3. 保存文件以使改变生效。OpenShift 会自动重启 Fuse 控制台 Pod。

其他资源

- OpenShift 4.x 上 Fuse 控制台的基于角色的访问控制
- 确定 OpenShift 4.x 上 Fuse 控制台的访问角色
- 在 OpenShift 4.x 上自定义对 Fuse 控制台进行基于角色的访问权限

2.4.4. 在 OpenShift 4.x 上升级 Fuse 控制台

Red Hat OpenShift 4.x 处理对 Operator 的更新，包括 Red Hat Fuse operator。如需更多信息，请参阅 Operator OpenShift 文档。

另外，Operator 更新可以根据应用程序的配置方式触发应用程序升级。

对于 Fuse 控制台应用程序，您还可以通过编辑应用程序自定义资源定义的 **.spec.version** 字段来触发对应用程序的升级。

前提条件

- 有 OpenShift 集群管理员权限。

流程

升级 Fuse 控制台应用程序：

1. 在终端窗口中，使用以下命令更改应用程序自定义资源定义的 **.spec.version** 字段：

```
oc patch -n <project-name> <custom-resource-name> --type='merge' -p '{"spec":{"version":"1.7.1"}}'
```

例如：

```
oc patch -n myproject hawtio/example-fuseconsole --type='merge' -p '{"spec":
{"version":"1.7.1"}}'
```

2. 检查应用程序的状态是否已更新：

```
oc get -n myproject hawtio/example-fuseconsole
```

响应显示有关应用程序的信息，包括版本号：

```
NAME                AGE   URL                                IMAGE
example-fuseconsole   1m    https://fuseconsole.192.168.64.38.nip.io
docker.io/fuseconsole/online:1.7.1
```

当您更改 .spec.version 字段的值时，OpenShift 会自动重新部署应用。

3. 检查版本更改触发的重新部署状态：

```
oc rollout status deployment.v1.apps/example-fuseconsole
```

成功部署会显示这个响应：

```
deployment "example-fuseconsole" successfully rolled out
```

2.4.5. 在 OpenShift 4.x 服务器上升级 Fuse 镜像流和模板

OpenShift Container Platform 4.x 使用 Samples Operator，它在 OpenShift 命名空间中运行，升级和更新基于 Red Hat Enterprise Linux (RHEL)的 OpenShift Container Platform 镜像流和模板。

升级 OpenShift 镜像流和模板上的 Fuse：

- 重新配置 Samples Operator

- 将 Fuse 镜像流和模板添加到 Skipped Imagestreams 和 Skipped Templates 字段中。
  - 跳过的 Imagestreams：位于 Samples Operator 清单中的 Imagestreams，但集群管理员希望 Operator 忽略或不管理。
  - 跳过的模板：位于 Samples Operator 清单中的模板，但集群管理员希望 Operator 忽略或不予管理。

先决条件

- 您可以访问 OpenShift 服务器。
- 您已将身份验证配置为 registry.redhat.io。

流程

1. 启动 OpenShift 4 服务器。
2. 以管理员身份登录 OpenShift 服务器。

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

3. 验证您是否使用您为其创建 docker-registry secret 的项目。

```
oc project openshift
```

4. 查看 Samples operator 的当前配置。

```
oc get configs.samples.operator.openshift.io -n openshift-cluster-samples-operator -o yaml
```

5. 配置 Samples operator 以忽略添加的 fuse 模板和镜像流。

```
oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

6. ***添加 Fuse imagestreams Skipped Imagestreams 部分，并将 Fuse 和 Spring Boot 2 模板添加到 Skipped Templates 部分。***

```
[...]
spec:
  architectures:
  - x86_64
  managementState: Managed
  skippedImagestreams:
  - fuse-console-rhel8
  - fuse-eap-openshift-jdk8-rhel7
  - fuse-eap-openshift-jdk11-rhel8
  - fuse-java-openshift-rhel8
  - fuse-java-openshift-jdk11-rhel8
  - fuse-karaf-openshift-rhel8
  - fuse-karaf-openshift-jdk11-rhel8
  - fuse-apicurito-generator-rhel8
  - fuse-apicurito-rhel8
  skippedTemplates:
  - s2i-fuse713-eap-camel-amq
  - s2i-fuse713-eap-camel-cdi
  - s2i-fuse713-eap-camel-cxf-jaxrs
  - s2i-fuse713-eap-camel-cxf-jaxws
  - s2i-fuse713-karaf-camel-amq
  - s2i-fuse713-karaf-camel-log
  - s2i-fuse713-karaf-camel-rest-sql
  - s2i-fuse713-karaf-cxf-rest
  - s2i-fuse713-spring-boot-2-camel-amq
  - s2i-fuse713-spring-boot-2-camel-config
  - s2i-fuse713-spring-boot-2-camel-drools
  - s2i-fuse713-spring-boot-2-camel-infinispan
  - s2i-fuse713-spring-boot-2-camel-rest-3scale
  - s2i-fuse713-spring-boot-2-camel-rest-sql
  - s2i-fuse713-spring-boot-2-camel
  - s2i-fuse713-spring-boot-2-camel-xa
  - s2i-fuse713-spring-boot-2-camel-xml
  - s2i-fuse713-spring-boot-2-cxf-jaxrs
  - s2i-fuse713-spring-boot-2-cxf-jaxws
  - s2i-fuse713-spring-boot-2-cxf-jaxrs-xml
  - s2i-fuse713-spring-boot-2-cxf-jaxws-xml
```

7. ***在 OpenShift 镜像流上升级 Fuse。***

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001

oc replace -n openshift -f ${BASEURL}/fis-image-streams.json
```

8. ***在 OpenShift 快速启动模板上升级 Fuse：***

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json ;
 do
 oc replace -n openshift \
 ${BASEURL}/quickstarts/${template}
 done
```

9. ***升级 Spring Boot 2 快速启动模板：***

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
 spring-boot-2-cxf-jaxrs-template.json \
 spring-boot-2-cxf-jaxws-template.json \
 spring-boot-2-cxf-jaxrs-xml-template.json \
 spring-boot-2-cxf-jaxws-xml-template.json ;
 do oc replace -n openshift \
 ${BASEURL}/quickstarts/${template}
 done
```

10. ***（可选） 查看 OpenShift 模板上的升级 Fuse：***

```
oc get template -n openshift
```

2.4.6. 在 OpenShift 4.x 上调整 Fuse 控制台的性能

默认情况下，Fuse 控制台使用以下 Nginx 设置：

- ***clientBodyBufferSize: 256k***
- ***proxyBuffers: 16 128k***

- *subrequestOutputBufferSize: 10m*

注：有关这些设置的描述，请参阅 Nginx 文档：http://nginx.org/en/docs/dirindex.html

要调整 Fuse 控制台的性能，您可以设置任何 clientBodyBufferSize、proxyBuffers 和 subrequestOutputBufferSize 环境变量。例如，如果您使用 Fuse 控制台监控大量 pod 和路由（例如，共 100 个路由），您可以通过将 Fuse Console 的 subrequestOutputBufferSize 环境变量设置为在 60m 到 100m 之间来解决加载超时问题。

如何设置这些环境变量取决于您在 Openshift 4.x 上安装 Fuse 控制台：

- *使用 Fuse Console Operator*
- *通过使用 Fuse Console 模板*

2.4.6.1. Fuse Console Operator 安装的性能调整

在 Openshift 4.x 上，您可以在部署 Fuse 控制台之前或之后设置 Nginx 性能调优环境变量。如果您随后这样做，OpenShift 会重新部署 Fuse 控制台。

先决条件

- *有 集群管理员 对 OpenShift 集群的访问权限。*
- *您已安装 Fuse Console Operator，如使用 OperatorHub 在 OpenShift 4.x 上安装和部署 Fuse 控制台 中所述。*

流程

您可以在部署 Fuse 控制台之前或之后设置环境变量。

- *在部署 Fuse 控制台前设置环境变量：*

1. *在 OpenShift Web 控制台中，安装有 Fuse Console Operator 的项目中，选择*

Operators > Installed Operators> Red Hat Integration - Fuse Console。

2. *单击 Hawtio 选项卡，然后单击 Create Hawtio。*

3. *在 Create Hawtio 页面中，在 Form 视图中，向下滚动到 Config> Nginx 部分。*

4. *展开 Nginx 部分，然后设置环境变量。例如：*

   - *clientBodyBufferSize: 256k*
   - *proxyBuffers: 16 128k*
   - *subrequestOutputBufferSize: 100m*

5. *保存配置。*

6. *点 Create 以部署 Fuse 控制台。*

7. *部署完成后，打开 Deployments> fuse-console 页面，然后点 Environment 来验证环境变量是否在列表中。*

- *在部署 Fuse 控制台后设置环境变量：*

  1. *在 OpenShift Web 控制台中，打开部署 Fuse 控制台的项目。*

  2. *选择 Operators> Installed Operators> Red Hat Integration - Fuse Console。*

  3. *单击 Hawtio 选项卡，然后单击 fuse-console。*

  4.

4. 选择 **Actions> Edit Hawtio**。

5. 在 **Editor** 窗口中，滚动到 **spec** 部分。

6. 在 **spec** 部分，添加新的 **nginx** 部分并指定一个或多个环境变量，例如：

```
apiVersion: hawt.io/v1alpha1
kind: Hawtio
metadata:
  name: fuse-console
spec:
  type: Namespace
  nginx:
    clientBodyBufferSize: 256k
    proxyBuffers: 16 128k
    subrequestOutputBufferSize: 100m
.
.
.
```

7. 点击 **Save**。

   OpenShift 重新部署 **Fuse** 控制台。

8. 重新部署完成后，打开 **Workloads> Deployments> fuse-console** 页面，然后点 **Environment** 来查看列表中的环境变量。

2.4.6.2. Fuse Console 模板安装的性能调整

在 **Openshift 4.x** 上，您可以在部署 **Fuse** 控制台之前或之后设置 **Nginx** 性能调优环境变量。如果您随后这样做，**OpenShift** 会重新部署 **Fuse** 控制台。

先决条件

- 有 集群管理员 对 **OpenShift** 集群的访问权限。
- 您已在 **OpenShift** 上安装 **Fuse** 控制台模板，如在 **OpenShift 4.x** 服务器 上安装 **Fuse** 镜像流和模板 中所述。

流程

您可以在部署 Fuse 控制台之前或之后设置环境变量。

- *在部署 Fuse 控制台前设置环境变量：*

1. *确定您要使用的 Fuse 控制台模板：*
   - *集群模板(fuse-console-cluster-os4.json)*
   - *带有可配置的 RBAC 的集群模板(fuse-console-cluster-rbac.yml)*
   - *命名空间模板(fuse-console-namespace-os4.json)*
   - *带有可配置的 RBAC (fuse-console-namespace-rbac.yml)的命名空间模板*
2. *编辑您要用于 Fuse 控制台的 Fuse Console 模板的本地副本，使其包含 NGINX_CLIENT_BODY_BUFFER_SIZE、NGINX_PROXY_BUFFERS 和/或 NGINX_SUBREQUEST_OUTPUT_BUFFER_SIZE 环境变量，如下例所示：*

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: fuse-console
spec:
  template:
    spec:
      containers:
      - env:
        - name: NGINX_CLIENT_BODY_BUFFER_SIZE
          value: 256k
        - name: NGINX_PROXY_BUFFERS
          value: 16 128k
        - name: NGINX_SUBREQUEST_OUTPUT_BUFFER_SIZE
          value: 100m
```

3. *保存您的更改。*
4.

按照 在 OpenShift 4.x 中设置 Fuse 控制台 中所述，请按照安装和部署 Fuse 控制台的步骤进行操作。

- 在部署 Fuse 控制台后设置环境变量：

1. 在终端窗口中，登录到 OpenShift 集群。

2. 打开部署 Fuse 控制台的项目。例如，如果在 myfuse 项目中部署了 Fuse 控制台，请使用以下命令：

```
oc project myfuse
```

3. 获取 Fuse 控制台部署的名称：

```
oc get deployments
```

此命令返回在当前项目中运行的部署列表。例如：

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
fuse-console        1/1     1            1           114m
```

4. 运行以下命令，为 Fuse 控制台部署设置环境变量：

```
oc set env dc/fuse-console NGINX_CLIENT_BODY_BUFFER_SIZE="256k"

oc set env dc/fuse-console NGINX_PROXY_BUFFERS="16 128k"

oc set env dc/fuse-console NGINX_SUBREQUEST_OUTPUT_BUFFER_SIZE="10m"
```

OpenShift 重新部署 Fuse 控制台。

5. 重新部署完成后，验证环境变量设置：

   a. 获取 Fuse Console pod 名称：

```
oc get pods
```

b. 运行以下命令来查看环境设置

```
oc exec <fuse-console-podname> -- cat /opt/app-root/etc/nginx.d/nginx-gateway.conf | grep "Performance tuning" -A 3
```

例如，如果 pod 名称是 fuse-console-6646cbbd4c-9rplg，请运行以下命令：

```
oc exec fuse-console-6646cbbd4c-9rplg -- cat /opt/app-root/etc/nginx.d/nginx-gateway.conf | grep "Performance tuning" -A 3
```

2.4.6.3. 在 Fuse 控制台中查看应用程序的性能调整

通过增强的 Fuse 控制台性能调优功能，您可以查看具有大量 MBeans 的应用程序。要使用此功能，请执行以下步骤。

先决条件

- 有 集群管理员 对 OpenShift 集群的访问权限。
- 您已安装 Fuse Console Operator，如使用 OperatorHub 在 OpenShift 4.x 上安装和部署 Fuse 控制台 中所述。

流程

1. 增加应用程序的内存限值。

需要增加内存限值，例如从 256Mi 增加到 512 Mi，因此应用程序在到达 Fuse 控制台前不会造成 OOM 错误崩溃。对于 Fuse 快速入门，请编辑应用程序的 src/main/jkube/deployment.yml 文件。

```
spec:
  template:
    spec:
      containers:
        -
          resources:
            [...]
```

```
        limits:
          cpu: "1.0"
          memory: 512Mi
```

2. 确保 Fuse Console Deployment 或 DeploymentConfig 具有足够的内存限值。如果还不够，将限制从 200Mi 增加到 512Mi。

3. 如果您在 nginx 日志中看到 "too big subrequest 响应 while to client" 错误，请应用 第 2.4.6.1 节 “Fuse Console Operator 安装的性能调整” 部分中提到的解决方案。

2.5. 配置 PROMETHEUS 以监控 OPENSHIFT 上的 FUSE 应用程序

2.5.1. 关于 Prometheus

Prometheus 是一个开源系统和服务监控和警报工具包，可用于监控 Red Hat OpenShift 环境中部署的服务。Prometheus 以给定间隔收集并存储来自配置的服务的指标，评估规则表达式，显示结果，并在指定条件变为 true 时触发警报。

重要

红帽对 Prometheus 的支持仅限于红帽产品文档中提供的设置和配置建议。

要监控 OpenShift 服务，您必须配置每个服务，以向 Prometheus 格式公开端点。此端点是一个 HTTP 接口，它提供指标列表和指标的当前值。Prometheus 定期提取每个目标定义端点，并将收集的数据写入其数据库。Prometheus 在延长时间内收集数据，而不仅仅是针对当前运行的会话。Prometheus 存储数据，以便您可以以图形方式视觉化并在数据上运行查询。

2.5.1.1. Prometheus 查询

在 Prometheus web 界面中，您可以使用 Prometheus Query Language (PromQL) 编写查询来选择和聚合收集的数据。

例如，您可以使用以下查询为带有 http_requests_total 作为指标名称的所有时间序列数据选择 Prometheus 在最后五分钟内记录的所有值：

```
http_requests_total[5m]
```

要进一步定义或过滤查询的结果，请为指标指定一个标签(key:value 对)。例如，您可以使用以下查

询为指标名称 http_requests_total 和作业标签设置为 集成 的所有时间序列数据选择 Prometheus 在最后五分钟内记录的所有值：

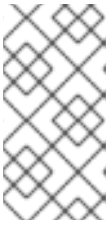
```
http_requests_total{job="integration"}[5m]
```

2.5.1.2. 显示 Prometheus 数据的选项

您可以指定 Prometheus 如何处理查询的结果：

- *在 Prometheus 的表达式浏览器中，将 Prometheus 数据视为表格数据。*
- *通过 Prometheus HTTP API 的外部系统使用 Prometheus 数据。*
- *显示图形中的 Prometheus 数据。*

Prometheus 提供它收集的数据的默认图形视图。如果您希望一个更强大的图形仪表板来查看 Prometheus 数据，Grafana 是一个流行的选择。



注意

Grafana 是社区支持的功能。红帽生产服务级别协议(SLA)不支持部署 Grafana 来监控红帽产品。

您还可以使用 PromQL 语言在 Prometheus 的 Alertmanager 工具中配置警报。

2.5.2. 为 4.14 设置 Prometheus

要设置 Prometheus，请在集群中安装 Prometheus operator 自定义资源定义，然后将 Prometheus 添加到包含 Fuse 应用的 OpenShift 项目中。

先决条件

- *有 集群管理员 对 OpenShift 集群的访问权限。*
- *您已通过在 OpenShift 镜像和模板上安装 Fuse 来准备 OpenShift 集群，如 OpenShift 中*

的 Fuse 指南中所述。

- 您已在集群中创建了 OpenShift 项目，并将 Fuse 应用添加到其中。

流程

1. 使用管理员权限登录到 OpenShift：

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

2. 创建名为 cluster-monitoring-config.yml 的文件：

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: cluster-monitoring-config
 namespace: openshift-monitoring
data:
 config.yaml: |
  enableUserWorkload: true
```

3. 将集群监控配置应用到 "openshift-monitoring" 命名空间：

```
oc create -f cluster-monitoring-config.yml -n openshift-monitoring
```

服务监控器 包含 Prometheus 在 OpenShift Container Platform 上收集给定服务和项目（命名空间）的指标的说明。

1. 安装服务监控器：

```
oc process -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/monitoring/fuse-servicemonitor.yml -p NAMESPACE=<YOUR NAMESPACE> -p FUSE_SERVICE_NAME=<YOUR FUSE SERVICE> | oc apply -f -
```

例如，带有名为 myproject 的 OpenShift 项目（命名空间），其中包含名为 myfuseapp 的 Fuse 应用程序：

Example

```
oc process -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/monitoring/fuse-servicemonitor.yml -p NAMESPACE=myproject -p FUSE_SERVICE_NAME=myfuseapp | oc apply -f -
```

2. *打开 Prometheus 仪表板：*

   a. *登录 OpenShift 控制台。*

   b. *打开您添加 Prometheus 的项目。*

   c. *在左侧窗格中，选择 Administrator 视图并打开 Observe -> Metrics*

   d. *点 Prometheus Hostname URL 打开 Prometheus 仪表板。*

有关 Prometheus 的更多信息，请参阅 Prometheus 文档。

2.5.3. OpenShift 环境变量

要配置应用程序的 Prometheus 实例，您可以设置 表 2.2 “Prometheus 环境变量” 中列出的 OpenShift 环境变量。

表 2.2. Prometheus 环境变量

环境变量	描述	default
AB_PROMETHEUS_HOST	要绑定的主机地址。	**0.0.0.0**
AB_PROMETHEUS_OFF	如果设置，禁用 Prometheus 激活 (echoes 空值)。	启用 Prometheus。
AB_PROMETHEUS_PORT	要使用的端口。	**9779**

环境变量	描述	default
AB_JMX_EXPORTER_CONFIG	使用文件（包括路径）作为 Prometheus 配置文件。	带有 Camel 指标的 /opt/prometheus/prometheus-config.yml 文件。
AB_JMX_EXPORTER_OPTS	附加到 JMX 导出器配置的附加选项。	不适用。

其他资源

有关为 pod 设置环境变量的信息，请参阅 OpenShift 开发人员指南 (https://access.redhat.com/documentation/zh-cn/openshift_container_platform/3.11/html/developer_guide/)。

2.5.4. 控制 Prometheus 监控和收集的指标

默认情况下，Prometheus 使用配置文件 https://raw.githubusercontent.com/jboss-fuse/application-templates/master/prometheus/prometheus-config.yml，其中包含 Camel 公开的所有可能指标。

如果您希望 Prometheus 监控并收集应用程序中的自定义指标（例如，应用程序进程的顺序数），您可以使用自己的配置文件。请注意，您可以识别的指标仅限于 JMX 中提供的指标。

流程

要使用自定义配置文件公开默认 Prometheus 配置没有涵盖的 JMX Bean，请按照以下步骤执行：

1. *创建自定义 Prometheus 配置文件。您可以使用默认文件的内容(prometheus-config.yml https://raw.githubusercontent.com/jboss-fuse/application-templates/master/prometheus/prometheus-config.yml)作为格式指南。*

   您可以将任何名称用于自定义配置文件，例如：my-prometheus-config.yml。

2. *将 prometheus 配置文件（如 my-prometheus-config.yml）添加到应用程序的 src/main/jkube-includes 目录中。*

3. *在应用程序中创建一个 src/main/jkube/deployment.xml 文件，并为 AB_JMX_EXPORTER_CONFIG 环境变量添加一个条目，并将其值设为您的配置文件。例如：*

```
spec:
  template:
    spec:
      containers:
        -
          resources:
            requests:
              cpu: "0.2"
            limits:
              cpu: "1.0"
          env:
          - name: SPRING_APPLICATION_JSON
            value: '{"server":{"tomcat":{"max-threads":1}}}'
          - name: AB_JMX_EXPORTER_CONFIG
            value: "my-prometheus-config.yml"
```

此环境变量适用于 pod 级别的应用程序。

4. *重新构建并部署应用程序。*

2.6. 为 OPENSHIFT 上的 FUSE 使用 METERING

您可以使用 OCP 4 上可用的 Metering 工具从不同的数据源生成 metering 报告。作为集群管理员，您可使用 Metering 来分析集群中的情况。您可以自行编写报告，也可以使用预定义的 SQL 查询来定义如何处理来自现有不同数据源的数据。使用 Prometheus 作为默认数据源，您可以生成 pod、命名空间和大多数其他 Kubernetes 资源的报告。您必须首先在 OpenShift Container Platform 4.x 上使用 Metering 工具在 OpenShift Container Platform 4.x 上安装和配置 Metering Operator。有关 Metering 的更多信息，请参阅 Metering。

注意

IBM Power 系统和 IBM Z 不支持 OpenShift 上的 Fuse 的 metering。

2.6.1. Metering 资源

Metering 具有很多资源，可用于管理 Metering 的部署与安装以及 Metering 提供的报告功能。Metering 使用以下自定义资源定义 (CRD) 来管理：

表 2.3. Metering 资源

名称	描述

名称	描述
MeteringConfig	为部署配置 metering 堆栈。包含用于控制 metering 堆栈各个组件的自定义和配置选项。
Reports	控制要使用的查询、查询运行时间、运行频率以及查询结果的存储位置。
ReportQueries	包含用于对 ReportDataSources 中所含数据进行分析的 SQL 查询。
ReportDataSources	控制 ReportQueries 和 Reports 可用数据。支持配置 metering 中使用的不同数据库的访问权限。

2.6.2. OpenShift 上 Fuse 的 metering 标签

表 2.4. metering Labels

标签	可能的值
com.company	**Red_Hat**
rht.prod_name	**Red_Hat_Integration**
rht.prod_ver	**7.13**
rht.comp	**fuse**
rht.comp_ver	**7.13**
rht.subcomp	**fuse7-java-openshift** **fuse7-eap-openshift** **fuse7-karaf-openshift**
rht.subcomp_t	**infrastructure**

2.7. 使用自定义 GRAFANA 仪表板监控 OPENSHIFT 上的 FUSE

OpenShift Container Platform 4.6 提供了监控仪表板，可帮助您了解集群组件和用户定义的工作负载的状态。

先决条件

- ***您必须在集群中安装并部署了 Prometheus。有关如何在 OpenShift 4 上安装 Grafana 的更***

多信息，请参阅 https://github.com/jboss-fuse/application-templates/blob/master/monitoring/prometheus.md。

- *您必须已安装并配置了 Grafana。*

OpenShift 上的 Fuse 的自定义仪表板

有两个自定义仪表板可用于 OpenShift 上的 Fuse。要使用这些仪表板，您必须在集群上安装和配置 Grafana 和 Prometheus。为 OpenShift 上的 Fuse 提供了两种示例仪表板：您可以从 Fuse Grafana 仪表板导入这些仪表板。

- *Fuse Pod/实例指标仪表板：*

此仪表板从单个 Fuse 应用程序 pod / 实例收集指标。您可以使用 fuse-grafana-dashboard.yml 导入仪表板。OpenShift 上 Fuse Pod 指标仪表板的面板表包括：

表 2.5. Fuse Pod 指标仪表板

标题	图例	查询	描述
进程开始时间	-	process_start_time_seconds{pod="$pod"}*1000	进程启动时的时间
当前内存占用器	-	sum(jvm_memory_bytes_used{pod="$pod", area="heap"})*100/sum(jvm_memory_bytes_max{pod="$pod", area="heap"})	Fuse 当前使用的内存
内存用量	已提交	sum(jvm_memory_bytes_committed{pod="$pod"})	已提交的内存
	使用的	sum(jvm_memory_bytes_used{pod="$pod"})	使用的内存
	max	sum(jvm_memory_bytes_max{pod="$pod"})	最大内存
线程	current	jvm_threads_current{pod="$pod"}	当前线程数量

标题	图例	查询	描述
	daemon	jvm_threads_daemon{pod="$pod"}	守护进程线程数量
	峰值	jvm_threads_peak{pod="$pod"}	峰值线程数
Camel Exchanges / 1m	Exchange Completed / 1m	sum(increase(org_apache_camel_ExchangesCompleted{pod="$pod"}[1m]))	每分钟完成 Camel 交换
	Exchange Failed / 1m	sum(increase(org_apache_camel_ExchangesFailed{pod="$pod"}[1m]))	每分钟失败的 Camel 交换
	Exchange Total / 1m	sum(increase(org_apache_camel_ExchangesTotal{pod="$pod"}[1m]))	每分钟 Camel 交换总数
	Exchange Inflight	sum(org_apache_camel_ExchangesInflight{pod="$pod"})	Camel Exchange 当前正在处理
Camel 处理时间	增量处理时间	sum(org_apache_camel_DeltaProcessingTime{pod="$pod"})	Camel 处理时间的 delta
	最后处理时间	sum(org_apache_camel_LastProcessingTime{pod="$pod"})	最后一个 Camel 处理时间
	最大处理时间	sum(org_apache_camel_MaxProcessingTime{pod="$pod"})	最大 Camel 处理时间
	Min Processing Time	sum(org_apache_camel_MinProcessingTime{pod="$pod"})	最小 Camel 处理时间
	平均处理时间	sum(org_apache_camel_MeanProcessingTime{pod="$pod"})	平均 Camel 处理时间
Camel 服务持续时间	最大持续时间	sum(org_apache_camel_MaxDuration{pod="$pod"})	最大 Camel 服务持续时间

标题	图例	查询	描述
	最小持续时间	sum(org_apache_camel_MinDuration{pod="$pod"})	最低 Camel 服务持续时间
	平均持续时间	sum(org_apache_camel_MeanDuration{pod="$pod"})	平均 Camel 服务持续时间
Camel Failures and Redeliveries	redeliveries	sum(org_apache_camel_Redeliveries{pod="$pod"})	redeliveries 数量
	最后处理时间	sum(org_apache_camel_LastProcessingTime{pod="$pod"})	最后一个 Camel 处理时间
	外部红帽	sum(org_apache_camel_ExternalRedeliveries{pod="$pod"})	外部 redeliveries 数量

- ***Fuse Camel Route Metrics Dashboard:***

此仪表板从 Fuse 应用程序中的单个 Camel 路由收集指标。您可以使用 fuse-grafana-dashboard-routes.yml 导入仪表板。OpenShift 上 Fuse Camel Route 指标仪表板的面板表包括：

表 2.6. Fuse Camel Route 指标仪表板

标题	图例	查询	描述
每秒交换数	-	rate(org_apache_camel_ExchangesTotal{route="\"$route\""}[5m])	每秒 Camel 交换总数
Exchange inflight	-	max(org_apache_camel_ExchangesInflight{route="\"$route\""})	当前正在处理的 Camel 交换数

标题	图例	查询	描述
交换失败率	-	sum (org_apache_camel_ExchangesFailed{route="\"$route\""})/ sum (org_apache_camel_ExchangesTotal{route="\"$route\""})	失败的 Camel 交换百分比
平均处理时间	-	org_apache_camel_MeanProcessingTime{route="\"$route\""}	平均 Camel 处理时间
每秒交换数	Failed	rate(org_apache_camel_ExchangesFailed{route="\"$route\""}[5m])	每秒的交换失败
	完成	rate(org_apache_camel_ExchangesCompleted{route="\"$route\""}[5m])	每秒完成交换
Exchange inflight	Exchange inflight	org_apache_camel_ExchangesInflight{route="\"$route\""}	Camel Exchange 当前正在处理
处理时间	Max	org_apache_camel_MaxProcessingTime{route="\"$route\""}	最大 Camel 处理时间
	平均	org_apache_camel_MeanProcessingTime{route="\"$route\""}	平均 Camel 处理时间
	Min	org_apache_camel_MinProcessingTime{route="\"$route\""}	最小 Camel 处理时间
每秒外部的 Redeliveries	-	rate(org_apache_camel_ExternalRedeliveries{route="\"$route\""}[5m])	每秒外部的 redeliveries
每秒的 redeliveries/redeliveries	-	rate(org_apache_camel_Redeliveries{route="\"$route\""}[5m])	每秒的 redeliveries/redeliveries

标题	图例	查询	描述
每秒处理的故障	-	rate(org_apache_camel_FailuresHandled{route="\"$route\""}[5m])	每秒处理的故障

2.8. 在 OPENSHIFT 3.X 服务器上安装 FUSE 镜像流和模板

*在为 **registry.redhat.io** 配置身份验证后，导入并使用 **OpenShift** 镜像流和模板上的 **Red Hat Fuse**。*

流程

1. *启动 **OpenShift** 服务器。*
2. *以管理员身份登录 **OpenShift** 服务器。*

```
oc login -u system:admin
```

3. *验证您是否使用您为其创建 **docker-registry secret** 的项目。*

```
oc project openshift
```

4. *在 **OpenShift** 镜像流上安装 **Fuse**。*

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001

oc create -n openshift -f ${BASEURL}/fis-image-streams.json
```

5. *安装 **Quickstart** 模板：*

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json ;
```

```
do
oc create -n openshift -f \
${BASEURL}/quickstarts/${template}
done
```

6. **安装 Spring Boot 2 Quickstart 模板：**

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
 spring-boot-2-cxf-jaxrs-template.json \
 spring-boot-2-cxf-jaxws-template.json \
 spring-boot-2-cxf-jaxrs-xml-template.json \
 spring-boot-2-cxf-jaxws-xml-template.json ;
 do oc create -n openshift -f \
 ${BASEURL}/quickstarts/${template}
 done
```

7. **安装 Fuse 控制台的模板。**

```
oc create -n openshift -f ${BASEURL}/fis-console-cluster-template.json
oc create -n openshift -f ${BASEURL}/fis-console-namespace-template.json
```

注意

有关部署 Fuse 控制台的详情，请参阅在 OpenShift 中设置 Fuse 控制台。

8. **安装 Apicurito 模板：**

```
oc create -n openshift -f ${BASEURL}/fuse-apicurito.yml
```

9. （可选） 查看 OpenShift 镜像和模板上安装的 Fuse：

```
oc get template -n openshift
```

2.8.1. 在 OpenShift 3.11 中设置 Fuse 控制台

在 OpenShift 3.11 中，您可以访问 Fuse 控制台：

- 通过将 Fuse 控制台添加到 OpenShift 项目，以便您可以监控项目中所有正在运行的 Fuse 容器。
- 通过将 Fuse 控制台添加到 OpenShift 集群，以便您可以监控集群中的所有项目中正在运行的 Fuse 容器。
- 从特定的 Fuse pod 打开它，以便您可以监控正在运行的 Fuse 容器。

您从命令行部署 Fuse 控制台模板。

注意

要在 Minishift 或 CDK based enviroments 上安装 Fuse 控制台，请按照以下 KCS 文章中所述的步骤操作。

- 要在 Minishift 或 CDK based enviroments 上安装 Fuse 控制台，请参阅 KCS 4998441。
- 如果需要禁用 Jolokia 身份验证，请参阅 KCS 3988671 中描述的临时解决方案。

前提条件

- 如 OpenShift 的 Fuse 指南中所述，在 OpenShift 镜像流和 Fuse 控制台安装 Fuse。

注意

- Fuse 控制台的用户管理由 OpenShift 处理。
- 基于角色的访问控制（在部署后访问 Fuse 控制台的用户）还不适用于 OpenShift 3.11 上的 Fuse。

第 2.8.1.1 节 “在 OpenShift 3.11 上部署 Fuse 控制台”

第 2.8.1.2 节 “从 OpenShift 3.11 上的 Fuse 控制台监控单个 Fuse pod”

2.8.1.1. 在 OpenShift 3.11 上部署 Fuse 控制台

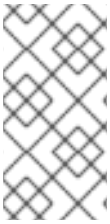
表 2.7 “Fuse 控制台模板” 描述可用于从命令行部署 Fuse 控制台的 OpenShift 3.11 模板，具体取决于 Fuse 应用程序部署的类型。

表 2.7. Fuse 控制台模板

类型	描述
fis-console-cluster-template.json	Fuse 控制台可以发现并连接到在多个命名空间或项目中部署的 Fuse 应用程序。要部署此模板，您必须具有 OpenShift cluster-admin 角色。
fis-console-namespace-template.json	此模板限制 Fuse 控制台访问当前 OpenShift 项目（命名空间），因此充当单个租户部署。若要部署此模板，您必须拥有当前 OpenShift 项目的 admin 角色。

另外，您可以通过运行以下命令来查看所有模板的参数列表：

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fis-console-namespace-template.json
```



注意

Fuse 控制台模板默认配置端到端加密，以便您的 Fuse 控制台请求是从浏览器到集群服务的安全端到端。

前提条件

- 对于 OpenShift 3.11 上的集群模式，您需要集群管理员角色和集群模式模板。运行以下命令：

```
oc adm policy add-cluster-role-to-user cluster-admin system:serviceaccount:openshift-infra:template-instance-controller
```

流程

从命令行部署 Fuse 控制台：

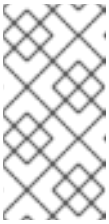
1. *运行以下命令，基于 Fuse 控制台模板创建一个新应用程序（其中 myproject 是项目的名称）：*

- *对于 Fuse Console 集群模板，其中 myhost 是用于访问 Fuse 控制台的主机名：*

```
oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fis-console-cluster-template.json -p ROUTE_HOSTNAME=myhost
```

- *对于 Fuse Console 命名空间 模板：*

```
oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001/fis-console-namespace-template.json
```



注意

您可以省略 命名空间 模板的 route_hostname 参数，因为 OpenShift 会自动生成一个。

2. *运行以下命令，获取 Fuse 控制台部署的状态和 URL：*

```
oc status
```

3. *要从浏览器访问 Fuse 控制台，请使用提供的 URL。*

Example:

+https://fuse-console.192.168.64.12.nip.io.

2.8.1.2. 从 OpenShift 3.11 上的 Fuse 控制台监控单个 Fuse pod

您可以为在 OpenShift 3.11 上运行的 Fuse pod 打开 Fuse 控制台。

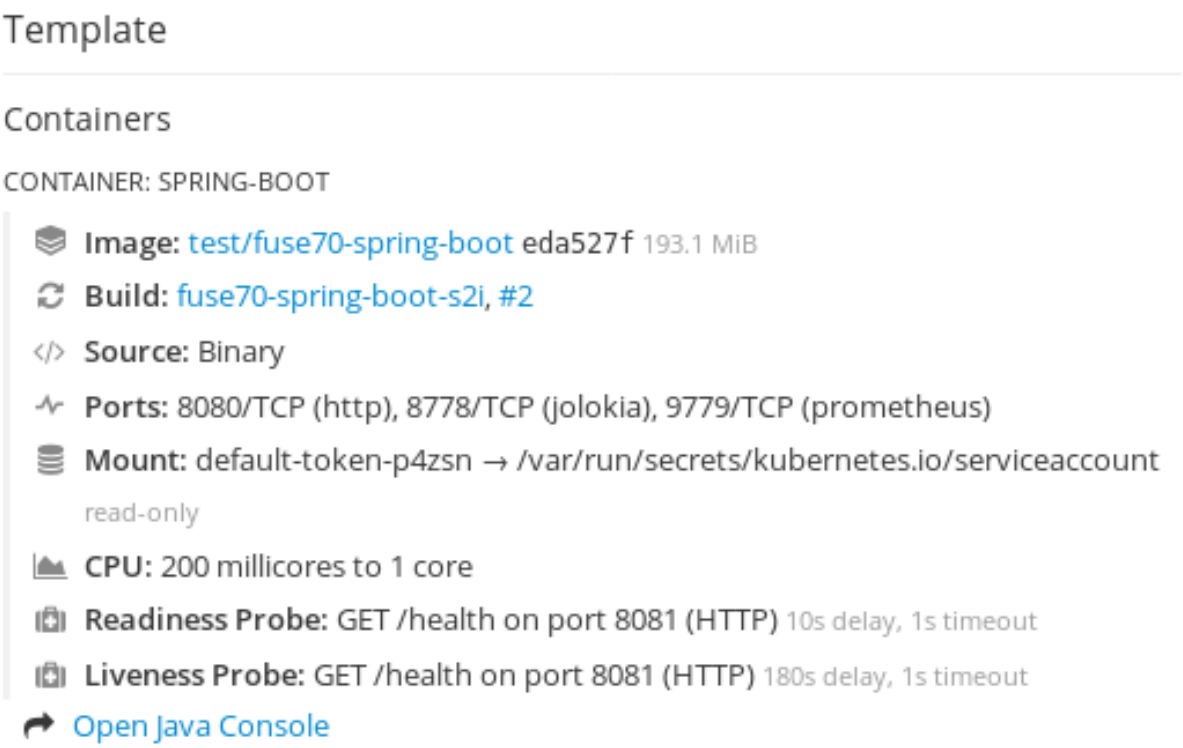
前提条件

- *要将 OpenShift 配置为在 pod 视图中显示到 Fuse Console 的链接，在 OpenShift 镜像上运行 Fuse 的 pod 必须在 name 属性中声明 TCP 端口，设置为 jolokia ：*

```
{
  "kind": "Pod",
  [...]
  "spec": {
    "containers": [
      {
        [...]
        "ports": [
          {
            "name": "jolokia",
            "containerPort": 8778,
            "protocol": "TCP"
          }
```

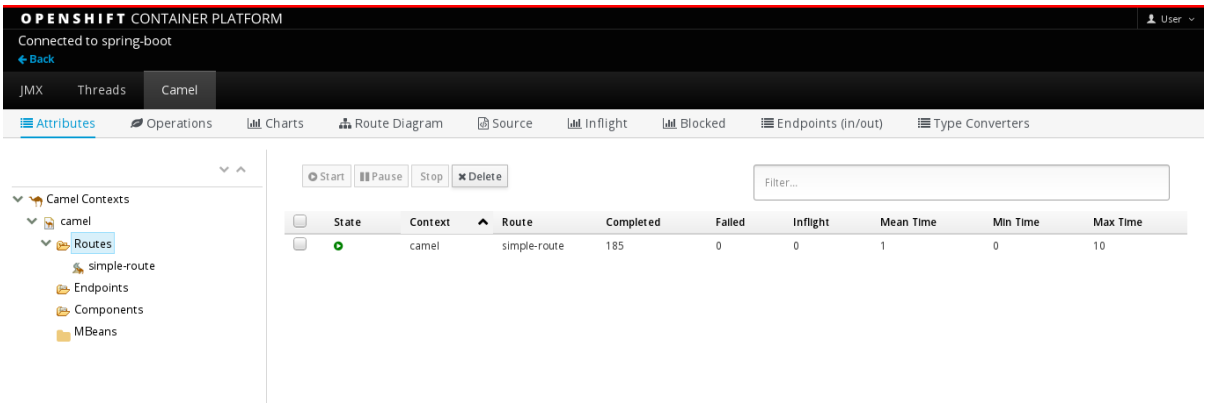
流程

1. *在 OpenShift 项目中的 Applications → Pods 视图中，点 pod 名称查看正在运行的 Fuse pod 的详情。在此页面的右侧，您会看到容器模板的摘要：*



2.

*从此视图中，单击 **Open Java Console** 链接以打开 **Fuse** 控制台。*



第 3 章 在受限环境中的 OPENSHIFT 上安装 FUSE

要在非限制的环境中的 OpenShift 上安装 Fuse，您可以从 registry.redhat.io 中拉取镜像流和模板。在没有或有限的互联网访问的生产环境中，无法实现。本节介绍如何在受限环境中的 OpenShift 上安装 Fuse。

注意

IBM Power Systems、IBM Z 和 LinuxONE 目前不支持在受限环境中安装。

先决条件

- 您已安装并配置了 OpenShift 服务器，以便它可以在受限环境中运行。

3.1. 设置内部 DOCKER REGISTRY

本节介绍如何设置可用于推送或拉取镜像的内部 docker registry。您必须配置可拉取或推送镜像的内部 docker registry。

流程

1. 安装内部 ROOT CA。

```
cd /etc/pki/ca-trust/source/anchors
sudo curl -O https://password.corp.redhat.com/RH-IT-Root-CA.crt
sudo update-ca-trust extract
sudo update-ca-trust update
```

此证书允许系统向 registry 验证其自身。

2. 登录 registry.redhat.io。

```
docker login -u USERNAME -p PASSWORD registry.redhat.io
```

3. 从 registry.redhat.io 拉取 OpenShift 镜像上的 Fuse。

```
docker pull registry.redhat.io/fuse7/fuse-java-openshift-rhel8:1.13
```

```
docker pull registry.redhat.io/fuse7/fuse-java-openshift-jdk11-rhel8:1.13
docker pull registry.redhat.io/fuse7/fuse-karaf-openshift-rhel8:1.13
docker pull registry.redhat.io/fuse7/fuse-console-rhel8:1.13
docker pull registry.redhat.io/fuse7/fuse-apicurito-rhel8:1.13
docker pull registry.redhat.io/fuse7/fuse-apicurito-generator-rhel8:1.13
```

4. *标记拉取的镜像流。*

```
docker tag registry.redhat.io/fuse7/fuse-java-openshift-rhel8:1.13 docker-
registry.upshift.redhat.com/fuse7/fuse-java-openshift-rhel8:1.13
docker tag registry.redhat.io/fuse7/fuse-java-openshift-jdk11-rhel8:1.13 docker-
registry.upshift.redhat.com/fuse7/fuse-java-openshift-jdk11-rhel8:1.13
docker tag registry.redhat.io/fuse7/fuse-karaf-openshift-rhel8:1.13 docker-
registry.upshift.redhat.com/fuse-karaf-openshift-rhel8:1.13
docker tag registry.redhat.io/fuse7/fuse-console-rhel8:1.13 docker-
registry.upshift.redhat.com/fuse7-fuse-console-rhel8:1.13
docker tag registry.redhat.io/fuse7/fuse-apicurito-rhel8:1.13 docker-
registry.upshift.redhat.com/fuse7-fuse-apicurito-rhel8:1.13
docker tag registry.redhat.io/fuse7/fuse-apicurito-generator-rhel8:1.13 docker-
registry.upshift.redhat.com/fuse7-fuse-apicurito-generator-rhel8:1.13
```

5. *将标记的镜像流推送到内部 docker registry。*

```
docker push docker-registry.upshift.redhat.com/fuse7/fuse-java-openshift-rhel8:1.13
docker push docker-registry.upshift.redhat.com/fuse7/fuse-java-openshift-jdk11-rhel8:1.13
docker push docker-registry.upshift.redhat.com/fuse-karaf-openshift-rhel8:1.13
docker push docker-registry.upshift.redhat.com/fuse7-fuse-console-rhel8:1.13
docker push docker-registry.upshift.redhat.com/fuse7-fuse-apicurito-rhel8:1.13
docker push docker-registry.upshift.redhat.com/fuse7-fuse-apicurito-generator-rhel8:1.13
```

3.2. 配置内部 REGISTRY SECRET

设置受限 Docker registry 并推送所有镜像后，需要配置受限的 OpenShift 服务器，以便它可以与内部注册表通信。

流程

1. *以管理员身份登录您的 OpenShift 集群：*

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

2. *打开您要在其中部署 Fuse 的项目：*

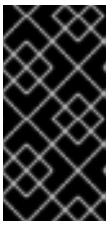
```
oc project myproject
```

3. 使用您的红帽客户门户网站帐户创建一个 docker-registry secret，将 PULL_SECRET_NAME 替换为 psi-internal-registry 创建：

```
oc create secret docker-registry psi-internal-registry \
  --docker-server=docker-registry.redhat.io \
  --docker-username=CUSTOMER_PORTAL_USERNAME \
  --docker-password=CUSTOMER_PORTAL_PASSWORD \
  --docker-email=EMAIL_ADDRESS
```

您应该看到以下输出：

```
secret/psi-internal-registry created
```



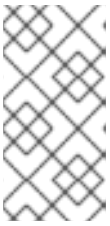
重要

您必须在要向 registry.redhat.io 进行身份验证的每个 OpenShift 项目命名空间中创建此 docker-registry secret。

4. 将机密链接到您的服务帐户，以使用机密拉取镜像。以下示例使用 default 服务帐户、builder 服务帐户和 deployer 服务帐户：

```
oc secrets link default psi-internal-registry
oc secrets link default psi-internal-registry --for=pull
oc secrets link builder psi-internal-registry
oc secrets link builder psi-internal-registry --for=pull
oc secrets link deployer psi-internal-registry
oc secrets link deployer psi-internal-registry --for=pull
```

服务帐户名称必须与 OpenShift 容器集使用的名称匹配。



注意

如果您不想使用红帽用户名和密码来创建 pull secret，您可以使用 registry 服务帐户创建身份验证令牌。

3.3. 在受限环境中的 OPENSHIFT 镜像上安装 FUSE

fis-image-streams.json 文件包含 Red Hat Fuse on OpenShift 的 imageStream 定义。但是，所有

*镜像流都引用 **registry.redhat.io**。您必须更改对 **psi-internal-registry URL** 的所有 **registry.redhat.io** 引用。*

流程

1. *下载 **Red Hat Fuse on OpenShift imagestream json** 文件。*

```
curl -o fis-image-streams.json {BASEURL}
```

2. *打开 **fis-image-streams.json** 文件，找到对 **'registry.redhat.io** 的所有引用。例如：*

```
{
"name": "1.9",
"annotations": {
"description": "Red Hat Fuse 7.13 Karaf S2I images.",
"openshift.io/display-name": "Red Hat Fuse 7.13 Karaf",
"iconClass": "icon-rh-integration",
"tags": "builder,jboss-fuse,java,karaf,xpaas,hidden",
"supports":"jboss-fuse:7.13.0,java:8,xpaas:1.2",
"version": "1.9"
},
"referencePolicy": {
"type": "Local"
},
"from": {
"kind": "DockerImage",
"name": "registry.redhat.io/fuse7/fuse-karaf-openshift-rhel8:1.11"
}
},
```

3. *将文件中的所有 **registry.redhat.io** 引用替换为 **psi-internal-registry** 名称。例如：*

```
{
"name": "1.9",
"annotations": {
"description": "Red Hat Fuse 7.13 Karaf S2I images.",
"openshift.io/display-name": "Red Hat Fuse 7.13 Karaf",
"iconClass": "icon-rh-integration",
"tags": "builder,jboss-fuse,java,karaf,xpaas,hidden",
"supports":"jboss-fuse:7.13.0,java:8,xpaas:1.2",
"version": "1.9"
},
"referencePolicy": {
"type": "Local"
},
"from": {
"kind": "DockerImage",
```

```
"name": "docker-registry.upshift.redhat.com/fuse7/fuse-karaf-openshift-rhel8:1.11"
}
},
```

4. 替换所有引用后，运行以下命令在 OpenShift 镜像流上安装 Fuse：

```
oc create -f fis-image-streams.json -n {namespace}
```

3.4. 使用内部 MAVEN 存储库

在受限环境中，您需要使用不同的 Maven 存储库。您可以使用名为 MAVEN_MIRROR_URL 的模板参数来指定它。您可以使用此 MAVEN_MIRROR_URL 参数从命令行创建新应用程序。

3.4.1. 使用 MAVEN_MIRROR_URL 运行 Spring Boot 应用程序

本例解释了如何使用 MAVEN_MIRROR_URL 部署和运行 Spring Boot 应用程序。

流程

1. 下载 Spring Boot Camel XML 快速入门。

```
oc create -f ./spring-boot-2-camel-xml-template.json -n openshift
```

2. 输入以下命令，使用 MAVEN_MIRROR_URL 参数创建运行 Spring Boot quickstart 模板所需的资源。

在受限环境中，您还需要为本地存储库指定 GIT_REPO 和 GIT_REF 参数。

```
oc new-app s2i-fuse713-spring-boot-2-camel-xml -n {namespace} -p
IMAGE_STREAM_NAMESPACE={namespace} -p MAVEN_MIRROR_URL={Maven mirror
URL} -p GIT_REPO={Git Repo URL} -p GIT_REF={Git branch/tag name}
```

这将为 Quickstart 创建部署配置和构建配置。有关 Quickstart 和创建的资源的默认参数的信息显示在终端上。

3.4.2. 使用 OpenShift Maven 插件运行 Spring Boot 应用程序

本例解释了如何使用内部 Maven 存储库部署并运行带有 OpenShift Maven 插件的 Spring Boot 应用程序。

流程

1. *要使用 OpenShift Maven 插件运行 Quickstart，请从本地存储库下载 Spring Boot 2 camel archetype，然后部署 Quickstart。将 {Maven Mirror URL} 替换为 Maven 镜像存储库 URL。*

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -DarchetypeCatalog={Maven Mirror URL}/archetypes/archetypes-catalog/2.2.0.fuse-7_13_0-00014-redhat-00001/archetypes-catalog-2.2.0.fuse-7_13_0-00014-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-xml-archetype
  -DarchetypeVersion=2.2.0.fuse-7_13_0-00014-redhat-00001
```

2. *archetype 插件切换到交互模式，以提示您输入剩余的字段。*

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse713-spring-boot2
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse713-spring-boot
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

3. *如果上述命令以 BUILD SUCCESS 状态退出，则现在您应该在 fuse713-spring-boot2 子目录下在 OpenShift 项目中有一个新的 Fuse。*

4. *现在，您可以构建和部署 fuse713-spring-boot2 项目。假设您仍然登录到 OpenShift，请更改到 fuse713-spring-boot2 项目的 目录，然后构建和部署项目，如下所示：*

```
cd fuse713-spring-boot2
mvn oc:deploy -Popenshift
```

第 4 章 非管理员用户在 OPENSHIFT 上安装 FUSE

您可以通过创建应用程序并将其部署到 OpenShift，开始在 OpenShift 上使用 Fuse。首先，您需要在 OpenShift 镜像和模板上安装 Fuse。

4.1. 以非 ADMIN 用户身份在 OPENSHIFT 镜像和模板上安装 FUSE

先决条件

- 您可以访问 OpenShift 服务器。它可以是 CDK 或远程 OpenShift 服务器的虚拟 OpenShift 服务器。
- 您已使用 registry.redhat.io 配置了身份验证。

如需更多信息，请参阅：

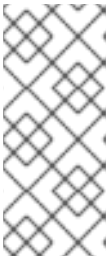
- 使用 registry.redhat.io 进行身份验证，用于容器镜像
- Red Hat CDK 3.17 入门指南

流程

1. 在准备在 OpenShift 项目中构建和部署 Fuse，请登录 OpenShift 服务器，如下所示：

```
oc login -u developer -p developer https://OPENSHIFT_IP_ADDR:8443
```

其中，OPENSHIFT_IP_ADDR 是 OpenShift 服务器的 IP 地址的占位符，因为此 IP 地址并非始终相同。



注意

developer 用户（使用开发人员密码）是 CDK 在虚拟 OpenShift Server 上自动创建的标准帐户。如果您要访问远程服务器，请使用 OpenShift 管理员提供的 URL 和凭据。

2. *创建名为 **test** 的新项目命名空间（假设它尚不存在）。*

```
oc new-project test
```

*如果 **test** 项目命名空间已存在，请切换到它。*

```
oc project test
```

3. *在 **OpenShift** 镜像流上安装 **Fuse**：*

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001

oc create -n test -f ${BASEURL}/fis-image-streams.json
```

*命令输出显示 **OpenShift** 项目的 **Fuse** 镜像流。*

4. *安装 **Quickstart** 模板。*

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json ;
 do
 oc create -n test -f \
 ${BASEURL}/quickstarts/${template}
 done
```

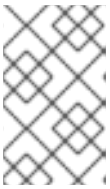
5. *安装 **Spring Boot 2 Quickstart** 模板：*

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
```

```
spring-boot-2-cxf-jaxrs-template.json \
spring-boot-2-cxf-jaxws-template.json \
spring-boot-2-cxf-jaxrs-xml-template.json \
spring-boot-2-cxf-jaxws-xml-template.json ;
do oc create -n openshift -f \
${BASEURL}/quickstarts/${template}
done
```

6. **安装 Fuse 控制台的模板。**

```
oc create -n test -f ${BASEURL}/fis-console-cluster-template.json
oc create -n test -f ${BASEURL}/fis-console-namespace-template.json
```



注意

有关部署 Fuse 控制台的详情，请参阅在 OpenShift 中设置 Fuse 控制台。

7. **（可选）查看 OpenShift 镜像和模板上安装的 Fuse。**

```
oc get template -n test
```

8. **在浏览器中，进入 OpenShift 控制台：**

   a. **使用 https://OPENSHIFT_IP_ADDR:8443，并将 OPENSHIFT_IP_ADDR 替换为您的 OpenShift 服务器的 IP 地址。**

   b. **使用您的凭据（例如，使用用户名 developer 和密码 developer）登录 OpenShift 控制台。**

第 5 章 开发人员入门

5.1. 准备开发环境

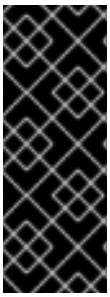
在 OpenShift 项目中开发和测试 Fuse 的根本要求是能够访问 OpenShift 服务器。您有以下基本替代方案：

- 安装 Red Hat CDK
- 远程访问现有 OpenShift 服务器

5.1.1. 在本地机器上安装容器开发套件(CDK)

作为开发人员，如果您想要快速启动，最实用的替代方案是在您的本地机器上安装 Red Hat CDK。使用 CDK，您可以引导在 Red Hat Enterprise Linux (RHEL) 7 上运行 OpenShift 镜像的虚拟机(VM)实例。CDK 的安装由以下关键组件组成：

- 虚拟机(libvirt、VirtualBox 或 Hyper-V)
- Minishift 来启动和管理容器开发环境



重要

Red Hat CDK 仅用于开发目的。它不适用于其他目的，如生产环境，可能无法解决已知的安全漏洞。为了获得对在使用 docker 格式的容器中运行的业务关键型应用程序的完全支持，您需要一个有效的 RHEL 7 或 RHEL Atomic 支持。如需了解更多详细信息，请参阅 对 Red Hat Container Development Kit (CDK)的支持。

先决条件

- Java 版本

在开发人员计算机上，确保您已安装了由 Fuse 7.13 支持的 Java 版本。有关支持的 Java 版本的详情，请参阅 支持的配置。

流程

在本地机器上安装 CDK：

1. 对于 OpenShift 上的 Fuse，我们建议您安装 CDK 的版本 3.17。有关安装和使用 CDK 3.17 的详细信息，请参阅 Red Hat CDK 3.17 入门指南。

2. 按照 为容器镜像验证 registry.redhat.io 中的说明，将 OpenShift 凭证配置为可以访问红帽生态系统目录。

3. 在 OpenShift 镜像和模板上安装了 Fuse，如 第 2 章 管理员入门 所述。

注意

您的 CDK 版本可能预安装了 Fuse on OpenShift 镜像和模板。但是，在配置 OpenShift 凭据后，您必须在 OpenShift 镜像和模板上安装（或更新 Fuse）。

4. 在进行本章中的示例之前，您应该阅读并全面了解 Red Hat CDK 3.17 入门指南。

5.1.2. 远程访问现有 OpenShift 服务器

您的 IT 部门可能已在某些服务器计算机上设置了 OpenShift 集群。在这种情况下，在 OpenShift 中使用 Fuse 时必须满足以下要求：

- 服务器机器必须运行受支持的 OpenShift Container Platform 版本（如 支持的配置页面 中所述）。本指南中的示例针对版本 3.11 进行了测试。

- 要求 OpenShift 管理员在 OpenShift 容器基础镜像和 OpenShift 服务器上的 OpenShift 模板上安装最新的 Fuse。

- 要求 OpenShift 管理员为您创建用户帐户，具有常见的开发人员权限（允许您创建、部署和运行 OpenShift 项目）。

- 询问管理员输入 OpenShift 服务器的 URL （您可以使用它浏览 OpenShift 控制台或使用 oc 命令行客户端连接到 OpenShift）以及帐户的登录凭据。

5.1.3. 安装客户端工具

我们建议您在开发人员机器上安装以下工具：

- Apache Maven 3.6.x：本地构建的 OpenShift 项目需要。从 Apache Maven 下载页面下载相应的软件包。确保您至少安装了 3.6.x 版本（或更新版本），否则 Maven 在构建项目时可能会遇到解决依赖项的问题。

- Git：OpenShift S2I 源工作流需要，并通常建议在 OpenShift 项目中对 Fuse 进行源控制。从 Git Downloads 页面下载适当的软件包。

- OpenShift 客户端：如果您使用 CDK，您可以使用 minishift oc -env 将 oc 二进制文件添加到 PATH 中，这显示您需要键入的命令(oc-env 的输出会根据 OS 和 shell 类型的不同而有所不同)：

```
$ minishift oc-env
export PATH="/Users/john/.minishift/cache/oc/v1.5.0:$PATH"
# Run this command to configure your shell:
# eval $(minishift oc-env)
```

  如需了解更多详细信息，请参阅 CDK 3.17 中的使用 OpenShift Client Binary 指南。

  如果您不使用 CDK，请按照 CLI 参考中的 说明安装 oc 客户端工具。

- （可选） Docker 客户端：高级用户可能会发现安装 Docker 客户端工具（与 OpenShift 服务器上运行的 docker 守护进程通信）会很方便。有关您的操作系统的特定二进制安装的详情，请查看 Docker 安装 站点。

  如需了解更多详细信息，请参阅 CDK 3.17 入门指南 中的重复使用 docker Daemon。

重要

确保安装 oc 工具的版本和与 OpenShift 服务器上运行的 OpenShift 版本兼容的 docker 工具。

5.1.4. 配置 Maven 软件仓库

配置 Maven 存储库，其中包含在本地机器上在 OpenShift 项目中构建 Fuse 所需的 archetypes 和工件。

流程

1. 打开您的 Maven settings.xml 文件，该文件通常位于 ~/.m2/settings.xml （在 Linux 或 macOS 上）或 Documents and Settings\<USER_NAME>\.m2\settings.xml (on Windows)。

2. 添加以下 Maven 存储库。

- Maven Central ：https://repo1.maven.org/maven2
- Red Hat GA 软件仓库 ：https://maven.repository.redhat.com/ga
- 红帽 EA 软件仓库 ：https://maven.repository.redhat.com/earlyaccess/all

您必须将前面的软件仓库同时添加到依赖项软件仓库部分，以及 settings.xml 文件的插件软件仓库部分。

5.2. 在 OPENSHIFT 的 FUSE 上创建和部署应用程序

您可以通过创建应用程序并将其部署到 OpenShift 中，使用以下 OpenShift Source-to-Image (S2I)应用程序开发工作流之一来开始使用 OpenShift 上的 Fuse：

S2I 二进制工作流

带有 二进制源 的构建输入的 S2I。此工作流的特征是，构建部分在开发人员自己的计算机上执行的事实。在本地构建二进制软件包后，此工作流会将二进制软件包传递给 OpenShift。如需了解更多详细信息，请参阅 构建 OpenShift Container Platform 指南中的 Binary Source。

S2I 源工作流

使用 Git 源构建 输入的 S2I。此工作流的特征是构建完全在 OpenShift 服务器上执行的事实。如需了解更多详细信息，请参阅 Build OpenShift Container Platform 指南中的 Git Source。

5.2.1. 使用 S2I 二进制工作流创建和部署应用程序

在本小节中，您将使用 OpenShift S2I 二进制工作流在 OpenShift 项目上创建、构建和部署 Fuse。

注意

使用 JDK11 运行快速入门

如果要在运行时使用基于 JDK11 的镜像，请在编译时使用正确的 JDK11 配置集。使用 JDK11 构建和部署快速入门时，请确保已在构建机器上安装 JDK11，然后使用正确的 JDK11 配置集构建快速入门。

流程

1. 使用 Maven archetype 在 OpenShift 项目中创建一个新的 Fuse。在本例中，我们使用一个 archetype，它会创建一个 Spring Boot Camel 项目示例。打开新的 shell 提示符，并输入以下 Maven 命令之一：

- 访问所有 S2I 快速入门：

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-7_13_0-00014-redhat-00001/archetypes-catalog-2.2.0.fuse-7_13_0-00014-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeVersion=2.2.0.fuse-7_13_0-00014-redhat-00001
```

- 仅访问 spring-boot-2-camel-xml 快速启动：

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-7_13_0-00014-redhat-00001/archetypes-catalog-2.2.0.fuse-7_13_0-00014-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-xml-archetype \
  -DarchetypeVersion=2.2.0.fuse-7_13_0-00014-redhat-00001
```

archetype 插件切换到交互模式，以提示您输入剩余的字段。

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse713-spring-boot
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
```

```
groupId: org.example.fis
artifactId: fuse713-spring-boot
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

出现提示时，为 groupId 值输入 org.example.fis，为 artifactId 值输入 fuse713-spring-boot。接受剩余字段的默认值。

2. 如果上一命令以 BUILD SUCCESS 状态退出，则现在您应该在 fuse713-spring-boot 子目录下在 OpenShift 项目中有一个新的 Fuse。您可以检查 fuse713-spring-boot/src/main/resources/spring/camel-context.xml 文件中的 XML DSL 代码。演示代码定义了一个简单的 Camel 路由，它会持续向日志发送包含随机数字的消息。

3. 在准备在 OpenShift 项目中构建和部署 Fuse，请登录 OpenShift 服务器，如下所示：

```
oc login -u developer -p developer https://OPENSHIFT_IP_ADDR:8443
```

其中，OPENSHIFT_IP_ADDR 是 OpenShift 服务器的 IP 地址的占位符，因为此 IP 地址并非始终相同。

注意

developer 用户（使用 开发人员 密码）是 CDK 在虚拟 OpenShift Server 上自动创建的标准帐户。如果您要访问远程服务器，请使用 OpenShift 管理员提供的 URL 和凭据。

4. 切换到 openshift 项目（如果尚未在 openshift 项目中），如下所示：

```
oc project openshift
```

5. 运行以下命令，以确保 OpenShift 镜像和模板上的 Fuse 已安装，并可以访问它们。

```
oc get template -n openshift
```

如果没有预安装镜像和模板，或者提供的版本已过时，请在 OpenShift 镜像和模板上手动安装（或更新）F。有关如何在 OpenShift 镜像上安装 Fuse 的更多信息，请参阅 第 2 章 管理员入门。

6.

现在，您可以构建和部署 fuse713-spring-boot 项目。假设您仍然登录到 OpenShift，请更改到 fuse713-spring-boot 项目的 目录，然后构建和部署项目，如下所示：

```
cd fuse713-spring-boot
mvn oc:deploy -Popenshift
```

在构建成功后，您应看到如下一些输出：

```
...
[INFO] OpenShift platform detected
[INFO] Using project: openshift
[INFO] Creating a Service from openshift.yml namespace openshift name fuse713-spring-
boot
[INFO] Created Service: target/jkube/applyJson/openshift/service-fuse713-spring-boot.json
[INFO] Using project: openshift
[INFO] Creating a DeploymentConfig from openshift.yml namespace openshift name
fuse713-spring-boot
[INFO] Created DeploymentConfig: target/jkube/applyJson/openshift/deploymentconfig-
fuse713-spring-boot.json
[INFO] Creating Route openshift:fuse713-spring-boot host: null
[INFO] F8: HINT: Use the command `oc get pods -w` to watch your pods start up
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 05:38 min
[INFO] Finished at: 2020-12-04T12:15:06+05:30
[INFO] Final Memory: 63M/688M
[INFO] ------------------------------------------------------------------------
```

注意

第一次运行此命令时，Maven 必须下载许多依赖项，这需要几分钟。后续构建将更快。

7.

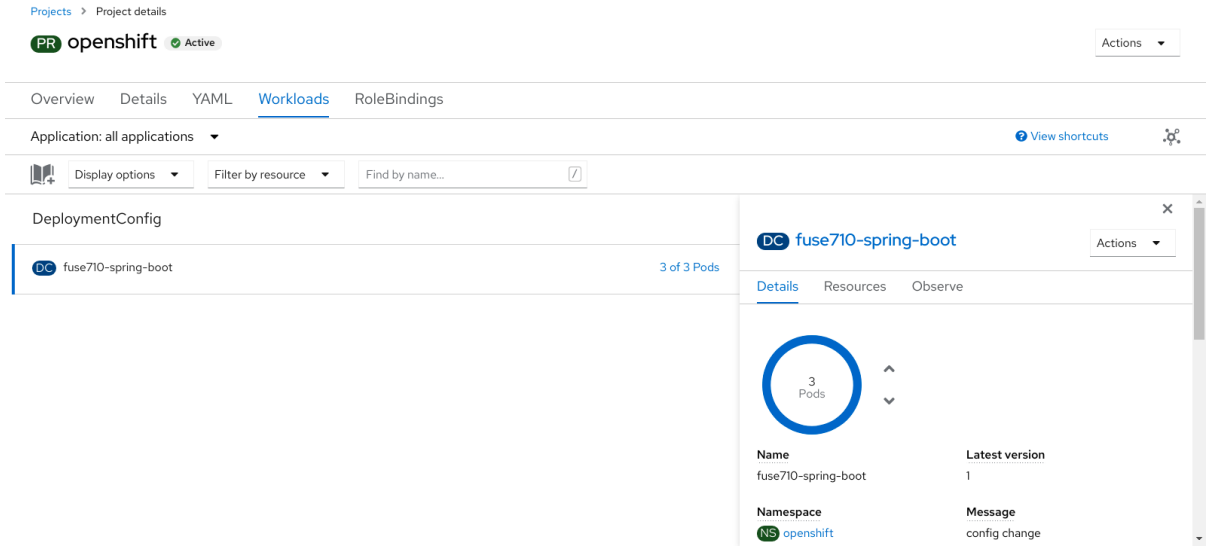
在浏览器中导航到 OpenShift 控制台，并使用您的凭据登录控制台（例如，使用用户名 developer 和密码 developer）。

8.

在左侧面板中，展开 Home，然后单击 Status 以查看 openshift 项目的 Project Status 页面。

9.

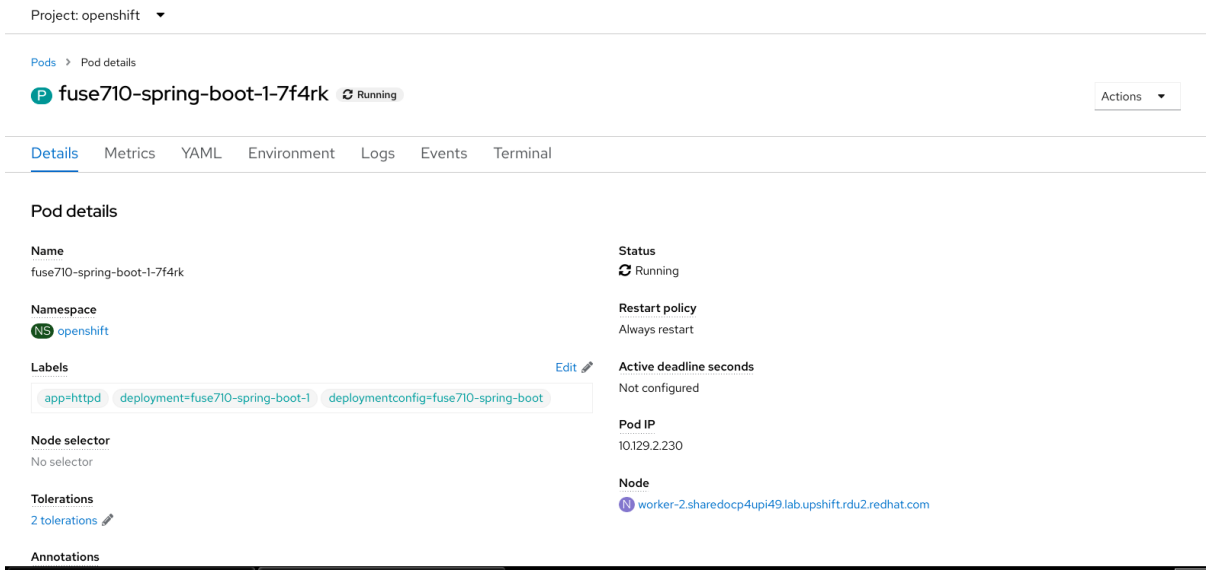
单击 fuse713-spring-boot，以查看 fuse713-spring-boot 应用程序的 Overview 信息页面。



10. *在左侧面板中，展开 **Workloads**。*

11. *单击 **Pods**。此时会显示 **openshift** 项目中所有正在运行的 **pod**。*

12. *单击 **pod Name** （本例中为 **fuse713-spring-boot-xxxxx**）以查看正在运行的 **pod** 的详细信息。*



13. *点 **Logs** 选项卡查看应用程序日志，再向下滚动日志以查找 **Camel** 应用程序生成的随机数字*

日志消息。

```
...
06:45:54.311 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 130
06:45:56.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 898
06:45:58.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 414
06:46:00.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 486
06:46:02.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 093
06:46:04.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 080
```

14. 关闭正在运行的 **pod**，

    a. 在 **openshift** 项目的 **Project Status** 页面中，单击 **fuse713-spring-boot** 应用程序。

    b. 点 **Overview** 选项卡查看应用程序的概述信息页面。

    c. 点 **Desired Count** 旁边的

    图标。此时会显示 **Edit Count** 窗口。

    d. 使用向下箭头缩减至零以停止容器集。

5.2.2. 取消部署并重新部署项目

您可以取消部署或重新部署项目，如下所示：

流程

- 要取消部署项目，请输入以下命令：

```
mvn oc:undeploy
```

- 要重新部署项目，请输入命令：

```
mvn oc:undeploy
mvn oc:deploy -Popenshift
```

5.2.3. 使用 S2I 源工作流创建和部署应用程序

在本小节中，您将使用 OpenShift S2I 源工作流根据模板在 OpenShift 应用上构建和部署 Fuse。此演示的起点是存储在远程 Git 存储库中的快速入门项目。使用 OpenShift 控制台，您将在 OpenShift 服务器中下载、构建和部署此快速入门项目。

流程

1. 按照如下所示，登录 OpenShift 服务器：

```
oc login -u developer -p developer https://OPENSHIFT_IP_ADDR:8443
```

其中，OPENSHIFT_IP_ADDR 是 OpenShift 服务器的 IP 地址的占位符，因为此 IP 地址并非始终相同。

注意

developer 用户（使用 开发人员 密码）是 CDK 在虚拟 OpenShift Server 上自动创建的标准帐户。如果您要访问远程服务器，请使用 OpenShift 管理员提供的 URL 和凭据。

2. 切换到 openshift 项目（如果尚未在 openshift 项目中），如下所示：

```
oc project openshift
```

3. 运行以下命令，以确保已安装 OpenShift 模板上的 Fuse，并可以访问它们。

```
oc get template -n openshift
```

如果没有预安装镜像和模板，或者提供的版本已过时，请在 OpenShift 镜像和模板上手动安装（或更新）F。有关如何在 OpenShift 镜像上安装 Fuse 的更多信息，请参阅 第 2 章 管理员入门。

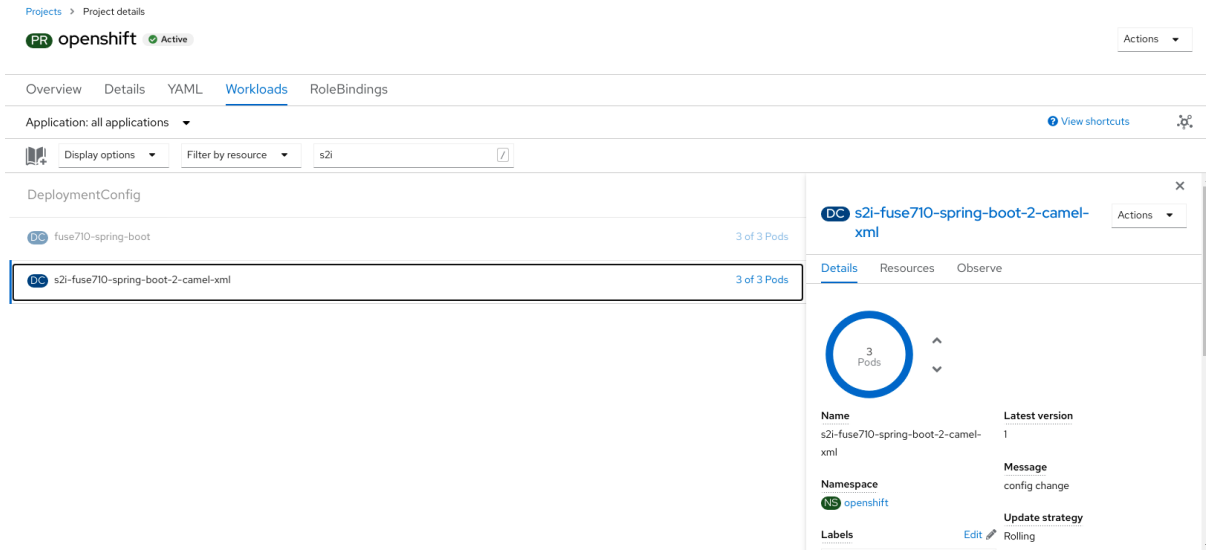
4. 输入以下命令来创建 使用 Spring Boot quickstart 模板运行 Red Hat Fuse 7.13 Camel XML DSL 所需的资源。这将为 Quickstart 创建部署配置和构建配置。有关 Quickstart 和创建的资源的默认参数的信息显示在终端上。

```
oc new-app s2i-fuse7-spring-boot-camel-xml

--> Deploying template "openshift/s2i-fuse7-spring-boot-camel-xml" to project openshift
...
--> Creating resources ...
    imagestream.image.openshift.io "s2i-fuse7-spring-boot-camel-xml" created
    buildconfig.build.openshift.io "s2i-fuse7-spring-boot-camel-xml" created
    deploymentconfig.apps.openshift.io "s2i-fuse7-spring-boot-camel-xml" created
--> Success
    Build scheduled, use 'oc logs -f bc/s2i-fuse7-spring-boot-camel-xml' to track its progress.
    Run 'oc status' to view your app.
```

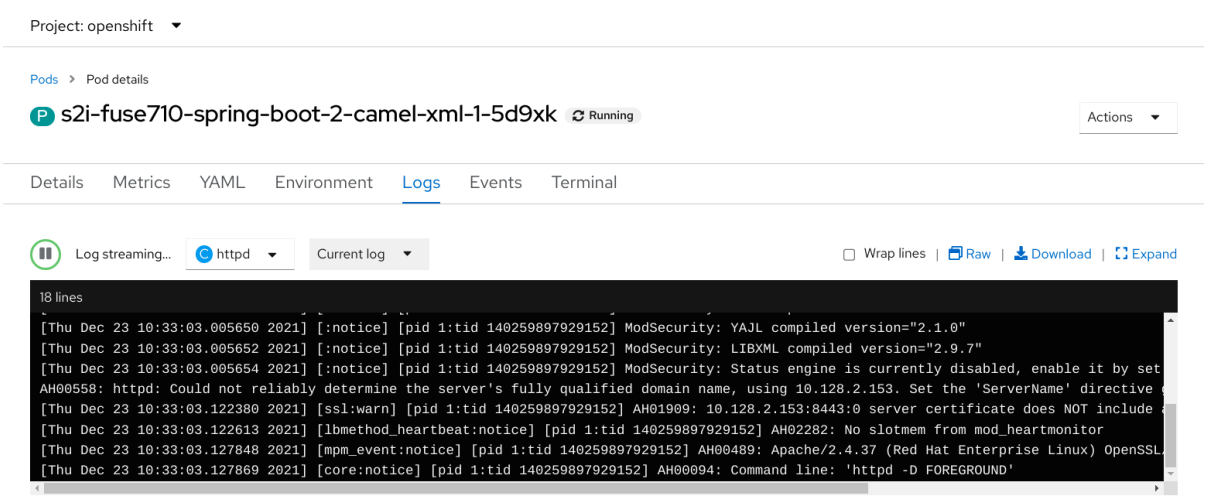
5. 导航到浏览器中的 **OpenShift Web 控制台(https://OPENSHIFT_IP_ADDR，将 OPENSHIFT_IP_ADDR 替换为集群的 IP 地址)，并使用您的凭证（例如，使用用户名 developer和密码 developer ）登录控制台。**

6. 在左侧面板中，展开 **Home**。单击 **Status** 以查看 **Project Status** 页面。此时会显示所选命名空间中的所有现有应用程序（如 **openshift**）。

7. 点 **s2i-fuse7-spring-boot-camel-xml** 查看快速启动的 **Overview** 信息页面。

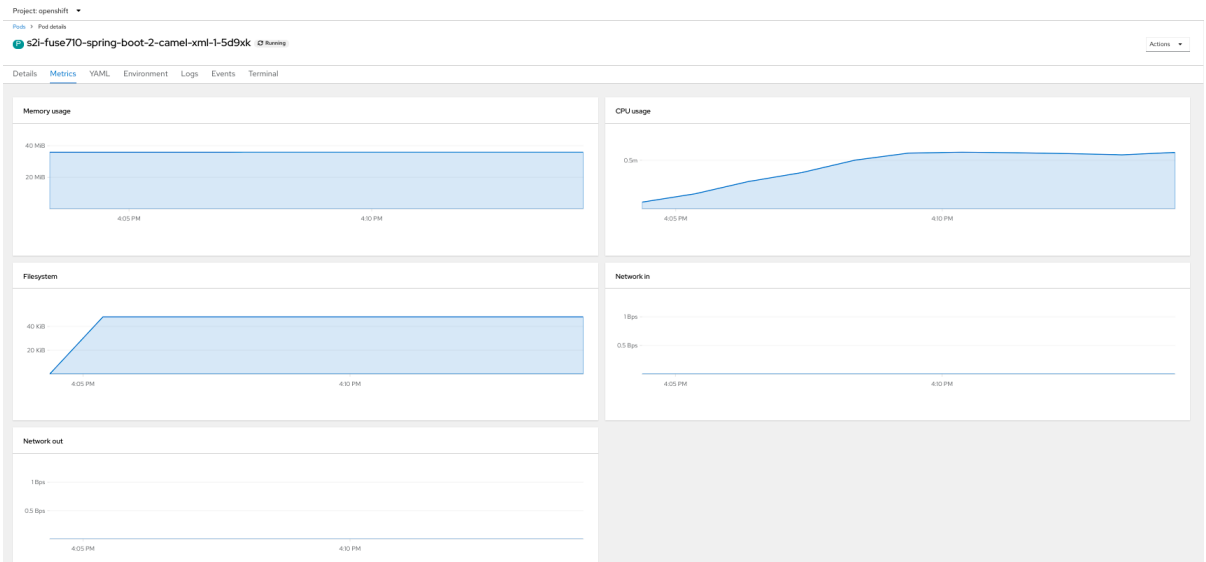


8. 单击 **Resources** 选项卡，然后单击 **View logs** 以查看应用的构建日志。



9. *在左侧面板中，展开 **Workloads**。*

10. *单击 **Pods**，然后单击 **s2i-fuse7-spring-boot-camel-xml-xxxx**。此时会显示应用程序的 **Pod** 详情。*



11. *关闭正在运行的 **pod**，*

a. *在 **openshift** 项目的 **Project Status** 页面中，单击 **s2i-fuse7-spring-boot-camel-xml-xxxx** 应用。*

b.

点 Overview 选项卡查看应用程序的概述信息页面。

c.

点 Desired Count 旁边的

图标。此时会显示 Edit Count 窗口。

d.

使用向下箭头缩减至零以停止容器集。

第 6 章 为 SPRING BOOT 镜像开发应用程序

本章介绍了如何为 Spring Boot 镜像开发应用程序。

6.1. 使用 MAVEN ARCHETYPE 创建 SPRING BOOT 2 项目

此快速入门演示了如何使用 Maven archetypes 创建 Spring Boot 2 项目。

流程

1. 进入系统中的相应目录。

2. 在 shell 提示符中，输入以下 mvn 命令来创建 Spring Boot 2 项目。

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-7_13_0-00014-redhat-00001/archetypes-catalog-2.2.0.fuse-7_13_0-00014-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-xml-archetype \
  -DarchetypeVersion=2.2.0.fuse-7_13_0-00014-redhat-00001
```

archetype 插件切换到交互模式，以提示您输入剩余的字段。

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse713-spring-boot
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse713-spring-boot
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

出现提示时，为 groupId 值输入 org.example.fis，为 artifactId 值输入 fuse713-spring-boot。接受剩余字段的默认值。

3. 如果上述命令以 BUILD SUCCESS 状态退出，则现在您应该在 fuse713-spring-boot 子目录下在 OpenShift 项目中有一个新的 Fuse。

4.

现在，您可以构建和部署 **fuse713-spring-boot** 项目。假设您仍然登录到 **OpenShift**，请更改到 **fuse713-spring-boot** 项目的 目录，然后构建和部署项目，如下所示：

```
cd fuse713-spring-boot
mvn oc:deploy -Popenshift
```

注意

有关可用 **Spring Boot 2 archetypes** 的完整列表，请参阅 **Spring Boot 2 Archetype Catalog**。

6.2. CAMEL SPRING BOOT 应用程序的结构

Camel Spring Boot 应用程序的目录结构如下：

```
├── LICENSE.md
├── pom.xml
├── README.md
├── configuration
│   └── settings.xml
└── src
    ├── main
    │   ├── jkube
    │   │   └── deployment.yml
    │   ├── java
    │   │   └── org
    │   │       └── example
    │   │           └── fis
    │   │               ├── Application.java
    │   │               └── MyTransformer.java
    │   └── resources
    │       ├── application.properties
    │       ├── logback.xml
    │       └── spring
    │           └── camel-context.xml
    └── test
        └── java
            └── org
                └── example
                    └── fis
```

以下文件对于开发应用程序非常重要：

pom.xml

包括其他依赖项。与 Spring Boot 兼容的 Camel 组件在入门版本中可用，如 camel-jdbc-starter 或 camel-infinispan-starter。启动者包含在 pom.xml 中后，它们会自动配置并在引导时使用 Camel 内容注册。用户可以使用 application.properties 文件配置组件的属性。

application.properties

允许您对配置进行外部化，并在不同的环境中使用相同的应用程序代码。详情请参阅 外部化配置

例如，在这个 Camel 应用程序中，您可以配置某些属性，如应用程序名称或 IP 地址等。

application.properties

```
#spring.main.sources=org.example.fos

logging.config=classpath:logback.xml

# the options from org.apache.camel.spring.boot.CamelConfigurationProperties can be configured here
camel.springboot.name=MyCamel

# lets listen on all ports to ensure we can be invoked from the pod IP
server.address=0.0.0.0
management.address=0.0.0.0

# lets use a different management port in case you need to listen to HTTP requests on 8080
management.server.port=8081

# disable all management endpoints except health
endpoints.enabled = false
endpoints.health.enabled = true
```

Application.java

它是运行应用程序的重要文件。作为用户，您将在此处导入 camel-context.xml 文件，以使用 Spring DSL 配置路由。

Application.java 文件指定 @SpringBootApplication 注释，它等同于 @Configuration、@EnableAutoConfiguration 和 @ComponentScan。

Application.java

```
@SpringBootApplication
// load regular Spring XML file from the classpath that contains the Camel XML DSL
@ImportResource({"classpath:spring/camel-context.xml"})
```

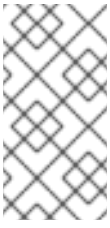
它必须具有运行 Spring Boot 应用程序的主要方法。

Application.java

```
public class Application {
   /**
    * A main method to start this application.
    */
   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }
}
```

camel-context.xml

src/main/resources/spring/camel-context.xml 是开发应用程序的重要文件，因为它包含 Camel 路由。

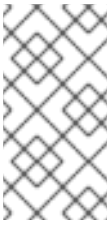


注意

您可以在开发 第一个 Spring Boot 应用程序中找到有关开发 Spring-Boot 应用程序的更多信息

src/main/jkube/deployment.yml

提供与 openshift-maven-plugin 生成的默认 OpenShift 配置文件合并的额外配置。



注意

此文件不是 Spring Boot 应用程序的一部分，但它用于限制 CPU 和内存用量等资源。

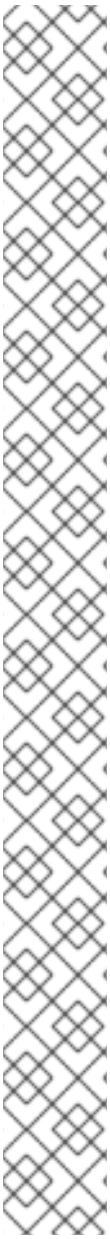
6.3. SPRING BOOT 2 ARCHETYPE 目录

Spring Boot 2 Archetype 目录包括以下示例。

表 6.1. Spring Boot 2 Maven Archetypes

Name	描述
spring-boot-camel-archetype	演示了如何根据 fabric8 Java 基础镜像在 Spring Boot 中使用 Apache Camel。
spring-boot-camel-amq-archetype	演示了如何将 Spring Boot 应用连接到 ActiveMQ 代理，并使用 Kubernetes 或 OpenShift 的两个 Camel 路由之间使用 JMS 消息传递。
spring-boot-camel-drools-archetype	演示如何使用 Apache Camel 将 Kubernetes 或 OpenShift 上运行的 Spring Boot 应用程序与远程 Kie 服务器集成。
spring-boot-camel-infinispan-archetype	演示如何使用 Hot Rod 协议将 Spring Boot 应用程序连接到 JBoss Data Grid 或 Infinispan 服务器。
spring-boot-camel-rest-3scale-archetype	演示如何使用 Camel 的 REST DSL 来公开 RESTful API 并将其公开给 3scale。
spring-boot-camel-rest-sql-archetype	演示如何通过 JDBC 和 Camel 的 REST DSL 使用 SQL 来公开 RESTful API。
spring-boot-camel-xml-archetype	演示了如何通过 Spring XML 配置文件在 Spring Boot 中配置 Camel 路由。
spring-boot-cxf-jaxrs-archetype	演示了如何根据 fabric8 Java 基础镜像在 Spring Boot 中使用 Apache CXF。Quickstart 使用 Spring Boot 配置包含启用了 Swagger 的 CXF JAXRS 端点的应用程序。
spring-boot-cxf-jaxws-archetype	演示了如何根据 fabric8 Java 基础镜像在 Spring Boot 中使用 Apache CXF。quickstart 使用 Spring Boot 配置包含 CXF JAXWS 端点的应用程序。
spring-boot-cxf-jaxrs-xml-archetype	演示如何在 OpenShift 上使用 Apache CXF JAX-RS 和 Spring Boot 2。此快速入门使用 Spring Boot2 启动基于 Spring 配置文件的 CXF 应用，其中包括启用了 Swagger 的 CXF JAXRS 端点。

Name	描述
spring-boot-cxf-jaxws-xml-archetype	演示如何在 OpenShift 上的 Spring Boot 2 中使用 Apache CXF JAX-WS。快速入门使用 Spring Boot2 启动基于 CXF JAXWS 端点的 Spring 配置文件。



注意

以下 Spring Boot 2 Maven archetypes 无法构建和部署到 OpenShift。如需更多信息，请参阅发行注记。

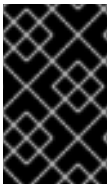
- *spring-boot-camel-archetype*
- *spring-boot-camel-infinspan-archetype*
- *spring-boot-cxf-jaxrs-archetype*
- *spring-boot-cxf-jaxws-archetype*

要临时解决这个问题，在为其中一个快速入门生成 Maven 项目后，编辑项目的 Maven pom.xml 文件以添加以下依赖项：

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>2.4.1</version>
  <scope>test</scope>
</dependency>
```

6.4. SPRING BOOT 的 BOM 文件

Maven Bill of Materials (BOM) 文件的目的是提供一组可正常工作的 Maven 依赖项版本，从而防止您必须为每个 Maven 工件单独定义版本。

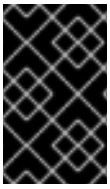


重要

确保您使用基于您正在使用的 Spring Boot 版本的正确 Fuse BOM。

Spring Boot 的 Fuse BOM 提供以下优点：

- 定义 Maven 依赖项的版本，以便在将依赖项添加到 POM 时不需要指定版本。
- 定义一组对特定版本的 Fuse 经过全面测试并支持的策展依赖关系。
- 简化 Fuse 的升级。



重要

红帽仅支持由 Fuse BOM 定义的一组依赖项。

6.5. 合并 BOM 文件

要将 BOM 文件合并到 Maven 项目中，请在项目的 pom.xml 文件中指定 dependencies Management 元素（或者，可能在父 POM 文件中），如两个 Spring Boot 2 的示例所示：

- Spring Boot 2 BOM

Spring Boot 2 BOM

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-springboot-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
```

```
      <scope>import</scope>
    </dependency>
  </dependencies>
 </dependencyManagement>
 ...
</project>
```

在使用依赖项管理机制指定 BOM 后，可以在没有指定工件版本 的情况下将 Maven 依赖项添加到您的 POM 中。例如，要为 camel-hystrix 组件添加依赖项，您可以在 POM 中的 dependencies 元素中添加以下 XML 片段：

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-hystrix-starter</artifactId>
</dependency>
```

请注意，Camel 工件 ID 如何通过 -starter 后缀，将 Camel Hystrix 组件指定为 camel-hystrix-starter，而不是 camel-hystrix。Camel 初学者组件以针对 Spring Boot 环境进行了优化的方式进行打包。

6.6. SPRING BOOT MAVEN 插件

Spring Boot Maven 插件由 Spring Boot 提供，它是一个用于构建和运行 Spring Boot 项目的开发人员实用程序：

- 通过在项目目录中输入 mvn package，为您的 Spring Boot 应用程序 构建 可执行文件 Jar 软件包。构建的输出放置在 Maven 项目的 target/ 子目录中。
- 为方便起见，您可以使用命令 mvn spring-boot:start 运行新构建的应用程序。

要将 Spring Boot Maven 插件合并到项目 POM 文件中，请将插件配置添加到 pom.xml 文件的 project/build/plugins 部分，如下例所示。

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<project ...>
 ...
 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  <!-- configure the versions you want to use here -->
  <fuse.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>

 </properties>
 ...
 <build>
  <plugins>
   <plugin>
    <groupId>org.jboss.redhat-fuse</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>${fuse.version}</version>
    <executions>
     <execution>
      <goals>
       <goal>repackage</goal>
      </goals>
     </execution>
    </executions>
   </plugin>
  </plugins>
 </build>
 ...
</project>
```

第 7 章 在 SPRING BOOT 中运行 APACHE CAMEL 应用程序

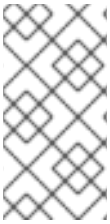
Apache Camel Spring Boot 组件为 Spring Boot 自动配置 Camel 上下文。Camel 上下文的自动配置会自动检测到 Spring 上下文中提供的 Camel 路由，并将生成者模板、消费者模板以及类型转换器注册为 Bean。Apache Camel 组件包含一个 Spring Boot starter 模块，允许您使用 starters 开发 Spring Boot 应用程序。

7.1. CAMEL SPRING BOOT 组件简介

每个 Camel Spring Boot 应用程序都必须使用项目的 pom.xml 中的 dependencyManagement 元素来指定依赖项的产品化版本。这些依赖项在 Red Hat Fuse BOM 中定义，并受特定版本的 Red Hat Fuse 的支持。您可以为额外的启动者省略 version number 属性，以便不覆盖 BOM 中的版本。如需更多信息，请参阅 Quickstart pom。

Example

```
<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>org.jboss.redhat-fuse</groupId>
   <artifactId>fuse-springboot-bom</artifactId>
   <version>${fuse.version}</version>
   <type>pom</type>
   <scope>import</scope>
  </dependency>
 </dependencies>
</dependencyManagement>
```



注意

camel-spring-boot jar 包含 spring.factories 文件，该文件允许您将依赖项添加到 classpath 中，以便 Spring Boot 自动配置 Camel 上下文。

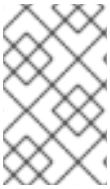
7.2. CAMEL SPRING BOOT STARTER 模块简介

Starters 是应该在 Spring Boot 应用程序中使用的 Apache Camel 模块。每个 Camel 组件都有一个 camel-xxx-starter 模块（在 第 7.3 节 “没有入门模块的 Camel 组件列表” 部分列出了几个例外）。

开始者满足以下要求：

- 使用与 IDE 工具兼容的原生 Spring Boot 配置系统允许自动配置组件。
- 允许自动配置数据格式和语言。
- 管理传输日志记录依赖项，以与 Spring Boot 日志记录系统集成。
- 包括额外的依赖项并对齐传输的依赖关系，以最大程度降低创建正常工作的 Spring Boot 应用程序的工作量。

每个入门程序在 test /camel-itest-spring-boot 中都有自己的集成测试，用于验证与 Spring Boot 的当前发行版本的兼容性。



注意

如需了解更多详细信息，请参阅链接： Apache Camel Spring-Boot 示例。

7.3. 没有入门模块的 CAMEL 组件列表

由于兼容性问题，以下组件没有初学者模块：

- camel-blueprint （只针对 OSGi 引入）
- camel-cdi （仅限 CDI 引入）
- camel-core-osgi （仅限 OSGi 修改）
- camel-ejb （仅限 JUnit 修改）

- camel-eventadmin （仅限 OSGi 引入）
- camel-ibatis （包含camel-mybatis-starter ）
- camel-jclouds
- camel-mina （包含camel-mina2-starter ）
- camel-paxlogging （仅限 OSGi 修改）
- camel-quartz （包含camel-quartz2-starter ）
- camel-spark-rest
- camel-openapi-java (camel-openapi-java-starter)

7.4. 使用 CAMEL SPRING BOOT STARTER

Apache Camel 提供了一个初学者模块，可让您快速开始开发 Spring Boot 应用程序。

流程

1. 将以下依赖项添加到 Spring Boot pom.xml 文件中：

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-boot-starter</artifactId>
</dependency>
```

2. 使用 Camel 路由添加类，如下面的代码片段中所示。将这些路由添加到类路径中后，路由会自动启动。

```
package com.example;
```

```
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("timer:foo")
          .to("log:bar");
    }
}
```

3. 可选。要让主线程阻止，以便 Camel 保持工作，请执行以下操作之一：

   a. 包含 spring-boot-starter-web 依赖项，

   b. 或将 camel.springboot.main-run-controller=true 添加到 application.properties 或 application.yml 文件中。

   您可以使用 camel.springboot mdadm 属性自定义 application.properties 或 application. yml 文件中的 Camel 应用程序。

4. 可选。要使用 bean 的 ID 名称引用自定义 bean，请在 src/main/resources/application.properties （或 application.yml）文件中配置选项。以下示例显示了 xslt 组件如何使用 bean ID 来引用自定义 bean。

   a. 请参阅 id myExtensionFactory 的自定义 bean。

   ```
   camel.component.xslt.saxon-extension-functions=myExtensionFactory
   ```

   b. 然后，使用 Spring Boot @Bean 注释创建自定义 bean。

   ```
   @Bean(name = "myExtensionFactory")
   public ExtensionFunctionDefinition myExtensionFactory() {
     }
   ```

   或者，对于 Jackson ObjectMapper，在 camel-jackson data-format 中：

```
camel.dataformat.json-jackson.object-mapper=myJacksonMapper
```

7.5. 关于 SPRING BOOT 的 CAMEL 上下文自动配置

Camel Spring Boot 自动配置提供 CamelContext 实例，并创建一个 SpringCamelContext。它还初始化并执行该上下文的关机。此 Camel 上下文在 camelContext bean 名称下的 Spring 应用程序上下文中注册，您可以像其他 Spring bean 一样访问它。您可以访问 camelContext，如下所示。

Example

```
@Configuration
public class MyAppConfig {

  @Autowired
  CamelContext camelContext;

  @Bean
  MyService myService() {
    return new DefaultMyService(camelContext);
  }

}
```

7.6. SPRING BOOT APPLICATIONS 中的自动探测 CAMEL 路由

Camel 自动配置从 Spring 上下文收集所有 RouteBuilder 实例，并将它们自动注入到 CamelContext 中。这简化了使用 Spring Boot 启动程序创建新的 Camel 路由的过程。您可以创建路由，如下所示：

Example

将 @Component annotated 类添加到您的 classpath 中。

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("jms:invoices").to("file:/invoices");
  }

}
```

或者在您的 @Configuration 类中创建一个新的路由 RouteBuilder bean。

```
@Configuration
public class MyRouterConfiguration {

  @Bean
  RoutesBuilder myRouter() {
    return new RouteBuilder() {

      @Override
      public void configure() throws Exception {
        from("jms:invoices").to("file:/invoices");
      }

    };
  }

}
```

7.7. 为 CAMEL SPRING BOOT 自动配置配置 CAMEL 属性

Spring Boot 自动配置连接到 Spring Boot 外部配置，如属性占位符、OS 环境变量或带有 Camel 属性的系统属性。

流程

1. *在 application.properties 文件中定义属性：*

```
route.from = jms:invoices
```

或者，将 Camel 正确设置为系统属性，例如：

```
java -Droute.to=jms:processed.invoices -jar mySpringApp.jar
```

2. *按照如下所示，将配置的属性用作 Camel 路由中的占位符：*

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("{{route.from}}").to("{{route.to}}");
```

```
    }

}
```

7.8. 配置自定义 CAMEL 上下文

要对 Camel Spring Boot 自动配置创建的 CamelContext bean 执行操作，请在 Spring 上下文中注册 CamelContextConfiguration 实例。

流程

- 在 Spring 上下文中注册 CamelContextConfiguration 实例，如下所示。

```
@Configuration
public class MyAppConfig {

 ...

 @Bean
 CamelContextConfiguration contextConfiguration() {
  return new CamelContextConfiguration() {
    @Override
    void beforeApplicationStart(CamelContext context) {
      // your custom configuration goes here
    }
  };
 }

}
```

在 Spring 上下文启动前，CamelContext Configuration 和 beforeApplicationStart (CamelContext) 方法被调用，因此传递给此回调的 CamelContext 实例会被完全自动配置。您可以将许多 CamelContextConfiguration 实例添加到 Spring 上下文中，并且所有实例都将被执行。

7.9. 在自动配置的 CAMELCONTEXT 中禁用 JMX

要在自动配置的 CamelContext 中禁用 JMX，您可以使用 camel.springboot.jmxEnabled 属性启用 JMX。

流程

- 在 application.properties 文件中添加以下属性，并将其设置为 false ：

```
camel.springboot.jmxEnabled = false
```

7.10. 将自动配置的消费者和制作者模板注入 SPRING 管理的 BEAN

Camel 自动配置提供预配置的 ConsumerTemplate 和 ProducerTemplate 实例。您可以将它们注入到 Spring 管理的 Bean 中。

Example

```
@Component
public class InvoiceProcessor {

  @Autowired
  private ProducerTemplate producerTemplate;

  @Autowired
  private ConsumerTemplate consumerTemplate;
  public void processNextInvoice() {
    Invoice invoice = consumerTemplate.receiveBody("jms:invoices", Invoice.class);
    ...
    producerTemplate.sendBody("netty-http:http://invoicing.com/received/" + invoice.id());
  }

}
```

默认情况下，消费者模板和制作者模板随端点缓存大小设置为 1000。您可以通过将以下 Spring 属性设置为所需缓存大小来更改这些值，例如：

```
camel.springboot.consumerTemplateCacheSize = 100
camel.springboot.producerTemplateCacheSize = 200
```

7.11. 关于 SPRING 上下文中自动配置的 TYPECONVERTER

Camel 自动配置在 Spring 上下文中注册一个名为 typeConverter 的 TypeConverter 实例。

Example

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public long parseInvoiceValue(Invoice invoice) {
    String invoiceValue = invoice.grossValue();
    return typeConverter.convertTo(Long.class, invoiceValue);
  }

}
```

7.12. SPRING 类型转换 API 网桥

Spring 由强大的 类型转换 API 组成。Spring API 与 Camel 类型转换器 API 类似。由于两个 API Camel Spring Boot 之间的相似性会自动注册桥接转换器(SpringTypeConverter)，它委托给 Spring 转换 API。这意味着开箱即用的 Camel 将对待与 Camel 类似的 Spring Converters。

这可让您使用 Camel TypeConverter API 访问 Camel 和 Spring 转换器，如下所示：

Example

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public UUID parseInvoiceId(Invoice invoice) {
    // Using Spring's StringToUUIDConverter
    UUID id = invoice.typeConverter.convertTo(UUID.class, invoice.getId());
  }

}
```

这里，Spring Boot 将转换委托给应用程序上下文中提供的 Spring ConversionService 实例。如果没有 ConversionService 实例，Camel Spring Boot 自动配置会创建一个 ConversionService 实例。

7.13. 禁用类型转换功能

要禁用 Camel Spring Boot 类型转换功能，请将 camel.springboot.typeConversion 属性设置为 false。当此属性设置为 false 时，自动配置不会注册类型转换器实例，且不会启用将类型转换为 Spring Boot 类型转换 API。

流程

- 要禁用 Camel Spring Boot 组件的类型转换功能，将 camel.springboot.typeConversion 属性设置为 false，如下所示：

```
camel.springboot.typeConversion = false
```

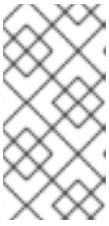
7.14. 在类路径中添加 XML 路由以进行自动配置

默认情况下，Camel Spring Boot 组件会自动探测，并包含 camel 目录中的 classpath 中的 Camel XML 路由。您可以使用配置选项配置目录名称或禁用此功能。

流程

- 在类路径中配置 Camel Spring Boot XML 路由，如下所示：

```
// turn off
camel.springboot.xmlRoutes = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRoutes = classpath:com/foo/routes/*.xml
```



注意

XML 文件应该定义 Camel XML 路由元素而不是 CamelContext 元素，例如：

```
<routes xmlns="http://camel.apache.org/schema/spring">
    <route id="test">
        <from uri="timer://trigger"/>
        <transform>
            <simple>ref:myBean</simple>
        </transform>
        <to uri="log:out"/>
    </route>
</routes>
```

使用 Spring XML 文件

要将 Spring XML 文件与 <camelContext> 搭配使用，您可以在 Spring XML 文件或 application.properties 文件中配置 Camel 上下文。要设置 Camel 上下文的名称并打开流缓存，请在 application.properties 文件中添加以下内容：

```
camel.springboot.name = MyCamel
camel.springboot.stream-caching-enabled=true
```

7.15. 为自动配置添加 XML REST-DSL 路由

Camel Spring Boot 组件会自动探测并嵌入 camel-rest 目录下的 classpath 中添加的 Camel Rest-DSL XML 路由。您可以使用配置选项配置目录名称或禁用此功能。

流程

- 在 classpath 中配置 Camel Spring Boot Rest-DSL XML 路由，如下所示：

```
// turn off
camel.springboot.xmlRests = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRests = classpath:com/foo/rests/*.xml
```

注意

Rest-DSL XML 文件应该定义 Camel XML REST 元素，而不是 CamelContext 元素，例如：

```
<rests xmlns="http://camel.apache.org/schema/spring">
  <rest>
    <post uri="/persons">
      <to uri="direct:postPersons"/>
    </post>
    <get uri="/persons">
      <to uri="direct:getPersons"/>
    </get>
    <get uri="/persons/{personId}">
      <to uri="direct:getPersionId"/>
    </get>
    <put uri="/persons/{personId}">
      <to uri="direct:putPersionId"/>
    </put>
    <delete uri="/persons/{personId}">
      <to uri="direct:deletePersionId"/>
```

```
    </delete>
  </rest>
</rests>
```

7.16. 使用 CAMEL SPRING BOOT 测试

当 Camel 在 Spring Boot 上运行时，Spring Boot 会自动嵌入 Camel 及其所有路由，这些路由使用 @Component 标注。使用 Spring Boot 测试时，使用 @SpringBootTest 而不是 @ContextConfiguration 来指定要使用的配置类。

当您在不同的 RouteBuilder 类中有多个 Camel 路由时，Camel Spring Boot 组件会在运行应用程序时自动嵌入所有这些路由。因此，当您希望只从一个 RouteBuilder 类测试路由时，您可以使用以下模式包含或排除要启用的 RouteBuilders：

- java-routes-include-pattern: 用于包括与模式匹配的 RouteBuilder 类。
- java-routes-exclude-pattern: 用于排除与模式匹配的 RouteBuilder 类。exclude 优先于 include。

流程

1. 将单元测试类中的 include 或 exclude 模式指定为 @SpringBootTest 注释的属性，如下所示：

```
@RunWith(CamelSpringBootRunner.class)
@SpringBootTest(classes = {MyApplication.class);
  properties = {"camel.springboot.java-routes-include-pattern=**/Foo*"})
public class FooTest {
```

在 FooTest 类中，include 模式是 **/Foo*，它代表 Ant 样式模式。在这里，模式以双星号开头，该星号与任何前导软件包名称匹配。/foo* 表示类名称必须以 Foo 开头，例如 FooRoute。

2. 使用以下 maven 命令运行测试：

```
mvn test -Dtest=FooTest
```

其它资源

- *编写组件*
- *组件*
- *端点*
- *开始使用*

第 8 章 在 OPENSHIFT 上的 FUSE 上运行 SOAP 到 REST BRIDGE QUICKSTART FOR SPRING BOOT 2

此快速入门演示了如何使用 Camel 的 REST DSL 来公开后端 SOAP API。简单的 camel 路由可以桥接 REST 调用旧 SOAP 服务。安全性适用于由 RH SSO 支持的 REST 端点和 SOAP 端点。前端 REST API 通过 OAuth 和 OpenID Connect 保护，客户端将使用 Resource Owner Password Credentials OAuth2 模式从 RH SSO 获取 JWT 访问令牌，并使用此令牌访问 REST 端点。

先决条件

- 已安装并配置了 OCP 4.1 或更高版本。
- 已安装 RH SSO 7.4 或更高版本。
- 已安装 3Scale 2.8 或更高版本。
- 您已将身份验证配置为 registry.redhat.io。如需更多信息，请参阅配置 Red Hat Container Registry 身份验证。

流程

以下小节解释了如何在 OpenShift 上的 Fuse 上运行和部署 SOAP 到 REST 网桥快速启动。

1. 启动 OpenShift 服务器。由于我们需要安装 RH SSO 镜像(2 个 pod)和 3Scale 镜像(15 个 pod)，因此我们需要在强大的机器上启动 OpenShift 服务器，选项 --memory 8GB --cpus 4。我们还需要在过期时间内发布安全令牌，因此我们还需要添加 timezone 选项。确保 Openshift 集群使用与本地机器相同的时区（默认为使用 UTC 时区）。

2. 将 cluster-admin 角色添加到用户 developer。

```
$ oc login -u system:admin
$ oc adm policy add-cluster-role-to-user cluster-admin developer
$ oc login -u developer
$ oc project openshift
```

此快速入门部署在 openshift 命名空间中（这是涉及的模板的默认配置）和 RH SSO 镜像的需求，因此我们需要向用户 开发人员 添加 cluster-admin 角色。

3.

创建一个机密，并将它链接到 serviceaccounts。

```
$ oc create secret docker-registry camel-bridge --docker-server=registry.redhat.io \
  --docker-username=USERNAME \
  --docker-password=PASSWORD \
  --docker-email=EMAIL_ADDRESS
$ oc secrets link default camel-bridge --for=pull
$ oc secrets link builder camel-bridge
```

4.

添加 RH SSO 镜像流，并使用模板 sso74-x509-postgresql-persistent 安装 RH SSO。

```
$ for resource in sso74-image-stream.json \
   sso74-https.json \
   sso74-postgresql.json \
   sso74-postgresql-persistent.json \
   sso74-x509-https.json \
   sso74-x509-postgresql-persistent.json
 do
   oc create -f \
   https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-
image/sso74-dev/templates/${resource}
 done

$ oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default

$ oc new-app --template=sso74-x509-postgresql-persistent
```

验证 RH SSO 镜像可从 openshift 命名空间可用，然后使用模板 sso74-x509-postgresql-persistent 安装 RH SSO。此模板可以平均保存 RH SSO 配置，因此配置会在 Openshift 服务器重启后保留。

5.

在服务器上成功安装了 RH SSO 镜像后，您可以看到控制台上的输出，如下所示：

```
A new persistent RH-SSO service (using PostgreSQL) has been created in your project. The
admin username/password for accessing the master realm via the RH-SSO console is
tprYtXP1/nEjf7fojv11FmhJ5eaqadoh0SI2gvlls. The username/password for accessing the
PostgreSQL database "root" is userqxe/XNYRjL74CrJEWW7HiSYEdH5FMKVSDytx. The
HTTPS keystore used for serving secure content, the JGroups keystore used for securing
JGroups communications, and server truststore used for securing RH-SSO requests were
automatically created via OpenShift's service serving x509 certificate secrets.

    * With parameters:
      * Application Name=sso
      * Custom RH-SSO Server Hostname=
      * JGroups Cluster Password=1whGRnsAWu162u0e4P6jNpLn5ysJLWjg # generated
      * Database JNDI Name=java:jboss/datasources/KeycloakDS
      * Database Name=root
```

```
* Datasource Minimum Pool Size=
* Datasource Maximum Pool Size=
* Datasource Transaction Isolation=
* PostgreSQL Maximum number of connections=
* PostgreSQL Shared Buffers=
* Database Username=userqxe # generated
* Database Password=XNYRjL74CrJEWW7HiSYEdH5FMKVSDytx # generated
* Database Volume Capacity=1Gi
* ImageStream Namespace=openshift
* RH-SSO Administrator Username=tprYtXP1 # generated
* RH-SSO Administrator Password=nEjf7fojv11FmhJ5eaqadoh0SI2gvlls # generated
* RH-SSO Realm=
* RH-SSO Service Username=
* RH-SSO Service Password=
* PostgreSQL Image Stream Tag=10
* Container Memory Limit=1Gi
```

6. *请注意用于访问 **RH SSO 管理控制台的 Username/Password。例如，***

```
* RH-SSO Administrator Username=tprYtXP1 # generated
* RH-SSO Administrator Password=nEjf7fojv11FmhJ5eaqadoh0SI2gvlls # generated
```

7. ***在 3scale 项目中安装 3scale 模板。***

```
$ oc new-project 3scale
$ oc create secret docker-registry threescale-registry-auth --docker-server=registry.redhat.io
--docker-server=registry.redhat.io \
  --docker-username=USERNAME \
  --docker-password=PASSWORD \
  --docker-email=EMAIL_ADDRESS
$ oc secrets link default threescale-registry-auth --for=pull
$ oc secrets link builder threescale-registry-auth
$ oc new-app --param WILDCARD_DOMAIN="OPENSHIFT_IP_ADDR.nip.io" -f
https://raw.githubusercontent.com/3scale/3scale-amp-openshift-
templates/2.8.0.GA/amp/amp-eval-tech-preview.yml
```

openshift 上的 3scale 安装将启动 15 个 pod，因此需要为 3scale 创建新项目。您还需要一个新的 threescale-registry-auth （使用此名称创建 secret，因为它以 3scale 模板编写）secret 用于 3scale。您可以重复使用 camel-bridge secret 中的 USERNAME/PASSWORD。我们有意使用 amp-eval-tech-preview.yml 模板，因为它没有明确指定硬件资源，因此可以在本地机器/laptop 上运行。

8. ***在 Openshift 上成功安装了 3scale 模板后，您可以看到控制台上的输出，如下所示：***

```
3scale API Management
    ---------
    3scale API Management main system (Evaluation)
```

```
Login on https://3scale-admin.192.168.64.33.nip.io as admin/b6t784nt

* With parameters:
  * AMP_RELEASE=2.8
  * APP_LABEL=3scale-api-management
  * TENANT_NAME=3scale
  * RWX_STORAGE_CLASS=null
  * AMP_BACKEND_IMAGE=registry.redhat.io/3scale-amp2/backend-rhel7:3scale2.8
  * AMP_ZYNC_IMAGE=registry.redhat.io/3scale-amp2/zync-rhel7:3scale2.8
  * AMP_APICAST_IMAGE=registry.redhat.io/3scale-amp2/apicast-gateway-
rhel8:3scale2.8
  * AMP_SYSTEM_IMAGE=registry.redhat.io/3scale-amp2/system-rhel7:3scale2.8
  * ZYNC_DATABASE_IMAGE=registry.redhat.io/rhscl/postgresql-10-rhel7
  * MEMCACHED_IMAGE=registry.redhat.io/3scale-amp2/memcached-rhel7:3scale2.8
  * IMAGESTREAM_TAG_IMPORT_INSECURE=false
  * SYSTEM_DATABASE_IMAGE=registry.redhat.io/rhscl/mysql-57-rhel7:5.7
  * REDIS_IMAGE=registry.redhat.io/rhscl/redis-32-rhel7:3.2
  * System MySQL User=mysql
  * System MySQL Password=mrscfh4h # generated
  * System MySQL Database Name=system
  * System MySQL Root password.=xbi0ch3i # generated
  * WILDCARD_DOMAIN=192.168.64.33.nip.io
  * SYSTEM_BACKEND_USERNAME=3scale_api_user
  * SYSTEM_BACKEND_PASSWORD=kraji167 # generated
  * SYSTEM_BACKEND_SHARED_SECRET=8af5m6gb # generated
  *
SYSTEM_APP_SECRET_KEY_BASE=726e63427173e58cbb68a63bdc60c7315565d6acd037c
aedeeb0050ecc0e6e41c3c7ec4aba01c17d8d8b7b7e3a28d6166d351a6238608bb84aa5d5b2d
c02ae60 # generated
  * ADMIN_PASSWORD=b6t784nt # generated
  * ADMIN_USERNAME=admin
  * ADMIN_EMAIL=
  * ADMIN_ACCESS_TOKEN=k055jof4itblvwwn # generated
  * MASTER_NAME=master
  * MASTER_USER=master
  * MASTER_PASSWORD=buikudum # generated
  * MASTER_ACCESS_TOKEN=xa7wkt16 # generated
  * RECAPTCHA_PUBLIC_KEY=
  * RECAPTCHA_PRIVATE_KEY=
  * SYSTEM_REDIS_URL=redis://system-redis:6379/1
  * SYSTEM_MESSAGE_BUS_REDIS_URL=
  * SYSTEM_REDIS_NAMESPACE=
  * SYSTEM_MESSAGE_BUS_REDIS_NAMESPACE=
  * Zync Database PostgreSQL Connection Password=efyJdRccBbYcWtWl # generated
  * ZYNC_SECRET_KEY_BASE=dcmNGWtrjCReuJlQ # generated
  * ZYNC_AUTHENTICATION_TOKEN=3FKMAije3V3RWQQ8 # generated
  * APICAST_ACCESS_TOKEN=2ql8txu4 # generated
  * APICAST_MANAGEMENT_API=status
  * APICAST_OPENSSL_VERIFY=false
  * APICAST_RESPONSE_CODES=true
  * APICAST_REGISTRY_URL=http://apicast-staging:8090/policies
```

9. *请注意可以访问 **3scale 管理控制台的 Username/Password。***

```
* ADMIN_PASSWORD=b6t784nt # generated
* ADMIN_USERNAME=admin
```

10. **配置 RH SSO。**

    a. 在 RH SSO 安装后，使用控制台中显示的用户名/密码从 https://sso-openshift.OPENSHIFT_IP_ADDR.nip.io/auth 登录 RH SSO 管理控制台。

    b. 单击页面左上角的 Add Realm 按钮。

    c. 在 Add Realm 页面上，选择 Import Select file 按钮。

    d. 从目录中选择 ./src/main/resources/keycloak-config/realm-export-new.json，这将为本例导入预定义必要的 realm/client/user/role。

11. **配置 3Scale API 网关。**

    a. 在 3Scale 安装后，使用控制台中显示的用户名/密码从 https://3scale-admin.OPENSHIFT_IP_ADDR.nip.io/p/admin/dashboard 登录 3Scale 管理控制台。

    b. 在创建新产品时，请选择 Define manually，并将 camel-security-bridge 用于 Name 和 System name。

    c. 在创建新后端时，将 camel-security-bridge 用于 Name 和 System name，而 私有基本 URL 应该为 http://spring-boot-camel-soap-rest-bridge-openshift.OPENSHIFT_IP_ADDR.nip.io/。

    d. 将新创建的后端添加到新创建的产品。

    e. 添加映射规则 Verb:POST Pattern:/。

    f. 在创建应用程序计划时，将 camel-security-bridge 用于 Name 和 System name。

g. 在创建应用程序时，选择新的创建的 camel-security-bridge 应用程序计划。创建应用程序后，记下 API 凭据。使用这些凭证访问 3scale 网关。例如，

```
User Key  bdfb53fe9b426fbf21428fd116035798
```

h. 编辑新创建的 camel-security-bridge 项目，并在 Dashboard 中从 camel-security-bridge 中发布它。

i. 进入 Integration > Settings。选择 As HTTP Headers 作为 Credentials 位置。

j. 在 Dashboard 中的 camel-security-bridge 中，进入 Integration > Configuration，并提升 Staging APIcast 和 Production APIcast。

12. 导航到包含提取的 Quickstart 应用程序的目录（例如，my_openshift/spring-boot-camel-soap-rest-bridge）。

```
$ cd my_openshift/spring-boot-camel-soap-rest-bridge
```

13. 构建和部署项目到 OpenShift 集群。

```
$ mvn clean oc:deploy -Popenshift -DJAVA_OPTIONS="-Dsso.server=https://sso-openshift.OPENSHIFT_IP_ADDR.nip.io -Dweather.service.host=${your local ip}"
```

我们需要在 openshift 上将两个属性传递给 camel-soap-rest-bridge 镜像。一个是 openshift 上的 RH SSO 服务器地址，它是 https://sso-openshift.OPENSHIFT_IP_ADDR.nip.io。另一个是后端 soap 服务器。在这个快速入门中，我们在本地机器上运行 backend soap 服务器，因此将机器的本地 IP 地址作为 -Dweather.service.host 传递。（这必须是 localhost 或 127.0.0.1 以外的 ip 地址。）

14. 在您的浏览器中，导航到 OpenShift 控制台中的 openshift 项目。等待 spring-boot-camel-soap-rest-bridge 的 pod 启动。

15. 在项目的 Overview 页面中，导航到 spring-boot-camel-soap-rest-bridge 应用程序的详情页面部署：
https://OPENSHIFT_IP_ADDR:8443/console/project/openshift/browse/pods/spring-boot-camel-soap-rest-bridge-NUMBER_OF_DEPLOYMENT?tab=details。

16. *切换到 Logs 选项卡，从 Camel 查看日志。*

17. *Access OpenApi API.*

本例使用 context-path camelcxf/openapi 使用 openapi 提供服务的 API 文档。您可以从 Web 浏览器访问 API 文档，地址为 http://spring-boot-camel-soap-rest-bridge-openshift.OPENSHIFT_IP_ADDR.nip.io/camelcxf/openapi/openapi.jsonn。

第 9 章 在带有 XA 事务的 SPRING BOOT 上运行 CAMEL 服务

Spring Boot Camel XA 事务快速入门演示了如何在 Spring-Boot 上运行 Camel 服务，它支持两个外部事务资源（一个 JMS 资源(A-MQ)和数据库(PostgreSQL)上的 XA 事务处理）。这些外部资源由 OpenShift 提供，它必须在运行此快速入门前启动。

9.1. STATEFULSET 资源

此快速入门使用 OpenShift StatefulSet 资源来确保事务管理器的唯一性，并且需要 PersistentVolume 来存储事务日志。应用程序支持在 StatefulSet 资源上扩展。每个实例都有 自己的进程 恢复管理器。特殊的控制器保证在应用程序缩减时，所有实例都会终止，完成其所有工作，而无需离开待处理的事务。如果恢复管理器无法在终止前清除所有待处理的工作，则控制器会回滚缩减操作。此快速入门使用 Spring Boot Narayana 恢复控制器。

9.2. SPRING BOOT NARAYANA 恢复控制器

Spring Boot Narayana 恢复控制器允许在终止前清理待处理的事务来安全地处理 StatefulSet 的缩减阶段。如果执行缩减操作，且 pod 在终止后没有清理，则会恢复前面的副本数，从而有效地取消缩减操作。

StatefulSet 的所有 pod 都需要访问一个共享卷，用于存储属于 StatefulSet 的每个 pod 的终止状态。StatefulSet 的 pod-0 会定期检查状态，并在存在不匹配时将 StatefulSet 扩展至正确的大小。

要使恢复控制器正常工作，需要编辑当前命名空间的权限（角色绑定包含在发布到 OpenShift 的资源集中）。可以使用 CLUSTER_RECOVERY_ENABLED 环境变量禁用恢复控制器。在这种情况下，服务帐户上不需要任何特殊权限，但任何缩减操作都可能会在终止的 pod 上离开待处理的事务，而无需通知。

9.3. 配置 SPRING BOOT NARAYANA 恢复控制器

以下示例演示了如何配置 Narayana 以使用恢复控制器在 OpenShift 上工作。

流程

1. 这是一个 application.properties 文件示例。替换 Kubernetes yaml 描述符中的以下选项。

```
# Cluster
cluster.nodename=1
cluster.base-dir=./target/tx
```

```
# Transaction Data
spring.jta.transaction-manager-id=${cluster.nodename}
spring.jta.log-dir=${cluster.base-dir}/store/${cluster.nodename}

# Narayana recovery settings
snowdrop.narayana.openshift.recovery.enabled=true
snowdrop.narayana.openshift.recovery.current-pod-name=${cluster.nodename}
# You must enable resource filtering in order to inject the Maven artifactId
snowdrop.narayana.openshift.recovery.statefulset=${project.artifactId}
snowdrop.narayana.openshift.recovery.status-dir=${cluster.base-dir}/status
```

2. **您需要一个共享卷来存储与终止相关的事务和信息。它可以挂载到 StatefulSet yaml 描述符中，如下所示。**

```
apiVersion: apps/v1
kind: StatefulSet
#...
spec:
#...
  template:
#...
    spec:
      containers:
      - env:
        - name: CLUSTER_BASE_DIR
          value: /var/transaction/data
          # Override CLUSTER_NODENAME with Kubernetes Downward API (to use `pod-0`,
`pod-1` etc. as tx manager id)
        - name: CLUSTER_NODENAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
#...
        volumeMounts:
        - mountPath: /var/transaction/data
          name: the-name-of-the-shared-volume
#...
```

Camel 扩展 for Spring Boot Narayana Recovery Controller

如果在 Spring Boot 应用程序上下文中找到 Camel，则 Camel 上下文会在清除所有待处理的事务前自动停止。

9.4. 在 OPENSHIFT 上运行 CAMEL SPRING BOOT XA QUICKSTART

此流程演示了如何在运行的单一节点 OpenShift 集群上运行 Quickstart。

流程

1. 下载 **Camel Spring Boot XA** 项目。

```
git clone --branch spring-boot-camel-xa-7.13.0.fuse-7_13_0-00011-redhat-00001
https://github.com/jboss-fuse/spring-boot-camel-xa
```

2. 导航到 **spring-boot-camel-xa** 目录，再运行以下命令：

```
mvn clean install
```

3. 登录 **OpenShift** 服务器。

```
oc login -u developer -p developer
```

4. 创建名为 **test** 的新项目命名空间（假设它尚不存在）。

```
oc new-project test
```

如果 **test** 项目命名空间已存在，请切换到它。

```
oc project test
```

5. 安装依赖项。

- 使用用户名 **username** 和密码 **Thepassword1!** 安装 **postgresql**。

```
oc new-app --param=POSTGRESQL_USER=theuser --
param=POSTGRESQL_PASSWORD='Thepassword1!' --
env=POSTGRESQL_MAX_PREPARED_TRANSACTIONS=100 --template=postgresql-
persistent
```

- 使用用户名用户名 和密码 **Thepassword1!** 安装 **A-MQ** 代理。

```
oc new-app --param=MQ_USERNAME=theuser --
param=MQ_PASSWORD='Thepassword1!' --template=amq63-persistent
```

6. 为事务日志创建持久性卷声明。

```
oc create -f persistent-volume-claim.yml
```

7. 构建和部署快速入门。

```
mvn oc:deploy -Popenshift
```

8. 扩展至所需副本数。

```
oc scale statefulset spring-boot-camel-xa --replicas 3
```

注： pod 名称用作事务管理器 ID (spring.jta.transaction-manager-id 属性)。当前实施还限制事务管理器 ID 的长度。请注意：

- StatefulSet 的名称是事务系统的标识符，因此不得更改它。
- 您应该为 StatefulSet 命名，以便所有 pod 名称的长度都小于或等于 23 个字符。OpenShift 使用惯例创建 Pod 名称： <statefulset-name>-0, <statefulset-name>-1 等。Narayana 最好避免有多个具有相同 id 的恢复管理器，因此当 pod 名称超过限制时，最后 23 字节被用作事务管理器 id （在剥离一些字符（如 - ）后。

9. 快速启动运行后，使用以下命令获取基础服务 URL。

```
NARAYANA_HOST=$(oc get route spring-boot-camel-xa -o jsonpath={.spec.host})
```

9.5. 测试成功 XA 事务

以下工作流演示了如何测试成功的 XA 事务。

流程

1. 获取 audit_log 表中的消息列表。

```
curl -w "\n" http://$NARAYANA_HOST/api/
```

2. 列表在启动时为空。现在，您可以放置第一个元素。

```
curl -w "\n" -X POST http://$NARAYANA_HOST/api/?entry=hello
```

等待一段时间后会得到新列表。

```
curl -w "\n" http://$NARAYANA_HOST/api/
```

3. 新列表包含两个消息，即 hello 和 hello-ok。hello-ok 确认消息已发送到传出队列，然后记录。您可以添加多个信息并查看日志。

9.6. 测试失败的 XA 事务

以下工作流演示了如何测试失败的 XA 事务。

流程

1. 发送名为 fail 的消息。

```
curl -w "\n" -X POST http://$NARAYANA_HOST/api/?entry=fail
```

2. 等待一段时间后会得到新列表。

```
curl -w "\n" http://$NARAYANA_HOST/api/
```

3. 此消息会在路由末尾生成一个异常，以便始终回滚事务。您应该不会在 audit_log 表中找到消息的任何追踪。

第 10 章 将 CAMEL 应用程序与 A-MQ 代理集成

本教程介绍了如何使用 A-MQ 镜像部署快速入门。

10.1. 构建和部署 SPRING BOOT CAMEL A-MQ 快速入门

此快速入门演示了如何将 Spring Boot 应用程序连接到 AMQ Broker，并在使用 OpenShift 上的 Fuse 的两个 Camel 路由之间使用 JMS 消息传递。

先决条件

- 确保 AMQ Broker 已安装并运行，如 在 OpenShift 上部署 AMQ Broker 所述。
- 确保 OpenShift 正确运行，并且 OpenShift 中已安装了 Fuse 镜像流。请参阅管理员入门。
- 确保为 fuse 配置 Maven 存储库，请参阅配置 Maven 存储库。

流程

1. 以开发者身份登录 OpenShift 服务器。

```
oc login -u developer -p developer
```

2. 为 Quickstart 创建一个新项目，例如：

```
oc new-project quickstart
```

3. 使用 Maven archetype 检索 Quickstart 项目：

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-
catalog/2.2.0.fuse-sb2-790047-redhat-00004/archetypes-catalog-2.2.0.fuse-sb2-790047-
redhat-00004-archetype-catalog.xml -DarchetypeGroupId=org.jboss.fuse.fis.archetypes -
DarchetypeArtifactId=spring-boot-camel-amq-archetype -DarchetypeVersion=2.2.0.fuse-sb2-
790047-redhat-00004
```

4. 导航到 quickstart 目录 fuse713-spring-boot-camel-amq。

```
cd fuse713-spring-boot-camel-amq
```

5. 运行以下命令，将配置文件应用到 AMQ Broker。这些配置文件创建 AMQ Broker 用户和队列，它们都有 admin 权限。

```
oc login -u admin -p admin

oc apply -f src/main/resources/k8s
```

6. 为应用程序创建 ConfigMap，例如：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: spring-boot-camel-amq-config
  namespace: quickstarts
data:
  service.host: 'fuse-broker-amqps-0-svc'
  service.port.amqp: '5672'
  service.port.amqps: '5671'
```

7. 使用 Step 3 中的 ImageStream 运行 mvn 命令，将 Quickstart 部署到 OpenShift 服务器：

```
mvn oc:deploy -Popenshift -Djkube.generator.fromMode=istag -
Djkube.generator.from=openshift/fuse-java-openshift:1.13
```

8. 验证 Quickstart 是否已成功运行：

   a. 导航到浏览器中的 OpenShift Web 控制台(https://OPENSHIFT_IP_ADDR，将 OPENSHIFT_IP_ADDR 替换为集群的 IP 地址)，并使用您的凭证（例如，使用用户名 developer 和密码 developer）登录控制台。

   b. 在左侧面板中，展开 Home，然后单击 Status 以查看 openshift 项目的 Project Status 页面。

   c. 点 fuse713-spring-boot-camel-amq 查看 Quickstart 的 Overview 信息页面。

d.

在左侧面板中，展开 **Workloads**。

e.

单击 **Pods**，然后单击 **fuse713-spring-boot-camel-amq-xxxxx**。此时会显示 **Quickstart** 的 **pod** 详情。

f.

点 **Logs** 查看应用程序的日志。

输出显示消息被成功发送。

```
10:17:59.825 [Camel (camel) thread #10 - timer://order] INFO  generate-order-route - Generating order order1379.xml
10:17:59.829 [Camel (camel) thread #8 - JmsConsumer[incomingOrders]] INFO  jms-cbr-route - Sending order order1379.xml to the UK
10:17:59.829 [Camel (camel) thread #8 - JmsConsumer[incomingOrders]] INFO  jms-cbr-route - Done processing order1379.xml
10:18:02.825 [Camel (camel) thread #10 - timer://order] INFO  generate-order-route - Generating order order1380.xml
10:18:02.829 [Camel (camel) thread #7 - JmsConsumer[incomingOrders]] INFO  jms-cbr-route - Sending order order1380.xml to another country
10:18:02.829 [Camel (camel) thread #7 - JmsConsumer[incomingOrders]] INFO jms-cbr-route - Done processing order1380.xml
```

9.

要查看 **Web** 界面上的路由，请点 **Open Java Console** 并检查 **AMQ** 队列中的信息。

第 11 章 将 SPRING BOOT 与 KUBERNETES 集成

Spring Cloud Kubernetes 插件目前使您能够集成 Spring Boot 和 Kubernetes 的以下功能：

- Spring Boot 外部配置
- Kubernetes ConfigMap
- Kubernetes Secret

11.1. SPRING BOOT 外部化配置

在 Spring Boot 中，外部化 配置是可让您将来自外部源的配置值注入 Java 代码的机制。在 Java 代码中，注入通常由使用 @Value 注释（注入单个字段）或 @ConfigurationProperties 注释（注入 Java bean 类的多个属性）启用。

配置数据可能会来自各种不同的源（或 属性源）。特别是，配置属性通常在项目的 application.properties 文件中设置（如果愿意，则为 application.yaml 文件）。

11.1.1. Kubernetes ConfigMap

Kubernetes ConfigMap 是一种可为已部署的应用提供配置数据的机制。ConfigMap 对象通常在 YAML 文件中定义，然后上传到 Kubernetes 集群，使配置数据可供已部署的应用程序使用。

11.1.2. Kubernetes Secret

Kubernetes 机密 是一种向已部署的应用提供敏感数据（如密码、证书等）的机制。

11.1.3. Spring Cloud Kubernetes 插件

Spring Cloud Kubernetes 插件实现 Kubernetes 和 Spring Boot 之间的集成。在原则上，您可以使用 Kubernetes API 从 ConfigMap 访问配置数据。但是，直接将 Kubernetes ConfigMap 与 Spring Boot 外部化配置机制集成更加方便，因此 Kubernetes ConfigMap 的行为是 Spring Boot 配置的替代属性源。这基本上是 Spring Cloud Kubernetes 插件提供的。

11.1.4. 启用带有 Kubernetes 集成的 Spring Boot

您可以通过将作为 Maven 依赖项添加到 pom.xml 文件来启用 Kubernetes 集成。

流程

1. 通过在 Spring Boot Maven 项目的 pom.xml 文件中添加以下 Maven 依赖项来启用 Kubernetes 集成。

```
<project ...>
 ...
 <dependencies>
  ...
  <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-kubernetes-config</artifactId>
  </dependency>
  ...
 </dependencies>
 ...
</project>
```

2. 以完成集成，

- 在您的 Java 源代码中添加一些注解
- 创建 Kubernetes ConfigMap 对象
- 修改 OpenShift 服务帐户权限，以允许应用程序读取 ConfigMap 对象。

其他资源

- 如需了解更多详细信息，请参阅为 ConfigMap 属性源运行教程。

11.2. 为 CONFIGMAP 属性源运行教程

以下教程允许您试验设置 Kubernetes Secret 和 ConfigMap。如 Enabling Spring Boot with Kubernetes Integration 所述启用 Spring Cloud Kubernetes 插件，将 Kubernetes 配置对象与 Spring Boot Externalized 配置集成。

11.2.1. 运行 Spring Boot Camel Config quickstart

以下教程基于 spring-boot-camel-config-archetype Maven archetype，它可让您设置 Kubernetes Secret 和 ConfigMap。

流程

1. 打开一个新的 shell 提示符，再输入以下 Maven 命令以创建简单的 Camel Spring Boot 项目。

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-7_13_0-00014-redhat-00001/archetypes-catalog-2.2.0.fuse-7_13_0-00014-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-config-archetype \
  -DarchetypeVersion=2.2.0.fuse-7_13_0-00014-redhat-00001
```

archetype 插件切换到互动模式，提示您输入剩余的字段：

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse713-configmap
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse713-configmap
version: 1.0-SNAPSHOT
package: org.example.fis
Y: : Y
```

出现提示时，为 groupId 值输入 org.example.fis，为 artifactId 值输入 fuse713-configmap。接受剩余字段的默认值。

2. 登录 OpenShift，并切换到要部署应用的 OpenShift 项目。例如，要以 developer 用户身份登录并部署到 openshift 项目，请输入以下命令：

```
oc login -u developer -p developer
oc project openshift
```

3. 在命令行中，切换到新的 fuse713-configmap 项目目录，并为此应用创建 Secret 对象。

```
cd fuse713-configmap
oc create -f sample-secret.yml
```

注意

在部署应用程序前，需要创建 Secret 对象，否则部署的容器进入 wait 状态，直到 Secret 可用为止。如果随后创建 Secret，容器将退出等待状态。有关如何设置 Secret 对象的更多信息，请参阅设置 Secret。

4. 构建和部署快速入门应用程序。在 fuse713-configmap 项目的顶部级别输入：

```
mvn oc:deploy -Popenshift
```

5. 查看应用日志，如下所示：

   a. 导航到浏览器中的 OpenShift Web 控制台(https://OPENSHIFT_IP_ADDR，将 OPENSHIFT_IP_ADDR 替换为集群的 IP 地址)，并使用您的凭证（例如，使用用户名 developer和密码 developer ）登录控制台。

   b. 在左侧面板中，展开 Home。单击 Status 以查看 Project Status 页面。此时会显示所选命名空间中的所有现有应用程序（如 openshift）。

   c. 单击 fuse713-configmap，以查看 Quickstart 的 Overview 信息页面。

   d. 在左侧面板中，展开 Workloads。

   e. 单击 Pods，然后单击 fuse713-configmap-xxxx。此时会显示应用程序的 Pod 详情。

   f. 点 Logs 选项卡查看应用程序日志。

6. 默认接收者列表在 src/main/resources/application.properties 中配置，将生成的消息发送到两个 dummy 端点： direct:async-queue 和 direct:file。这会导致以下信息写入应用程序日志中：

```
5:44:57.377 [Camel (camel) thread #0 - timer://order] INFO  generate-order-route -
```

```
Generating message message-44, sending to the recipient list
15:44:57.378 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ----> message-44 pushed to an async queue (simulation)
15:44:57.379 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ----> Using username 'myuser' for the async queue
15:44:57.380 [Camel (camel) thread #0 - timer://order] INFO  target-route--file - ----> message-44 written to a file
```

7. **在使用 ConfigMap 对象更新 fuse713-configmap 应用配置前，您必须提供 fuse713-configmap 应用程序权限来查看 OpenShift ApiServer 中的数据。输入以下命令为 fuse713-configmap 应用程序的服务帐户授予 view 权限：**

```
oc policy add-role-to-user view system:serviceaccount:openshift:qs-camel-config
```

注意

服务帐户通过 system:serviceaccount:PROJECT_NAME:SERVICE_ACCOUNT_NAME 指定。fis-config 部署描述符定义 SERVICE_ACCOUNT_NAME 为 qs-camel-config。

8. **要查看操作中的实时重新加载功能，请按如下所示创建 ConfigMap 对象：**

```
oc create -f sample-configmap.yml
```

新 ConfigMap 覆盖正在运行的应用程序中 Camel 路由的接收者列表，将其配置为将生成的消息发送到 三个 虚拟端点： direct:async-queue,direct:file, direct:mail。如需有关 ConfigMap 对象的更多信息，请参阅设置 ConfigMap。这会导致以下信息写入应用程序日志中：

```
16:25:24.121 [Camel (camel) thread #0 - timer://order] INFO  generate-order-route - Generating message message-9, sending to the recipient list
16:25:24.124 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ----> message-9 pushed to an async queue (simulation)
16:25:24.125 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ----> Using username 'myuser' for the async queue
16:25:24.125 [Camel (camel) thread #0 - timer://order] INFO  target-route--file - ----> message-9 written to a file (simulation)
16:25:24.126 [Camel (camel) thread #0 - timer://order] INFO  target-route--mail - ----> message-9 sent via mail
```

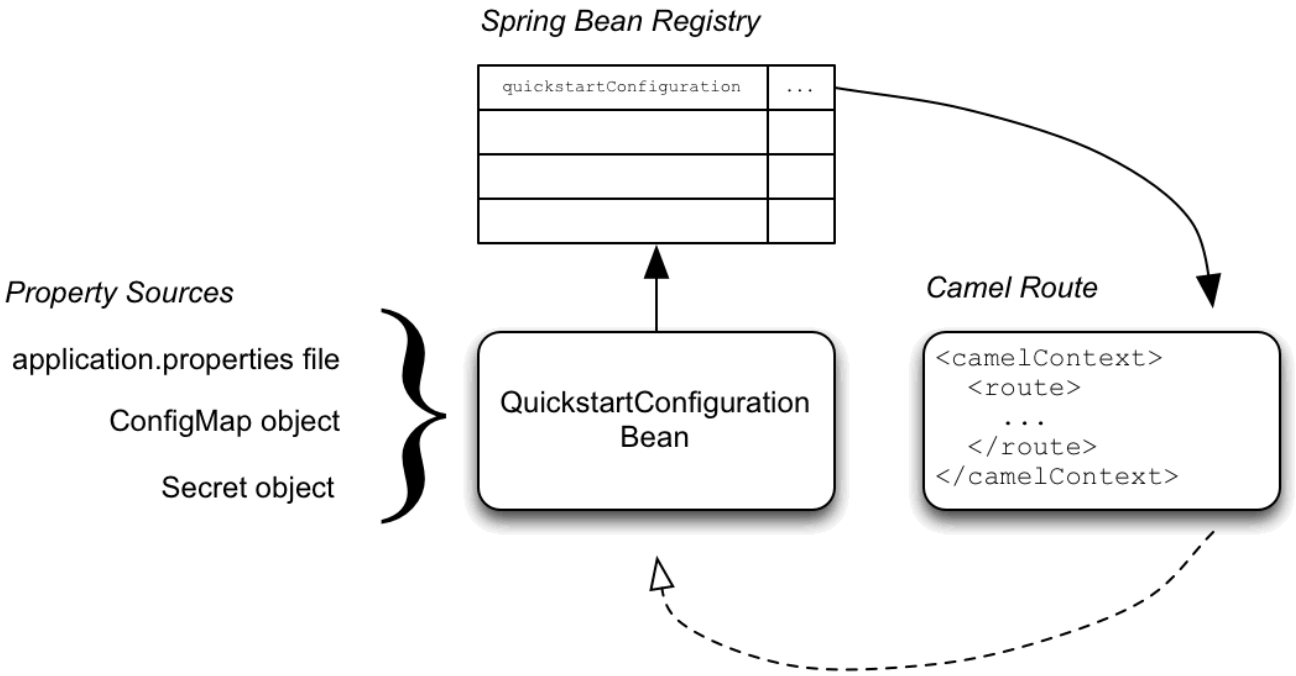
11.2.2. 配置属性 bean

配置属性 bean 是一个常规 Java Bean，可通过注入接收配置设置。它提供了 Java 代码与外部配置机

制之间的基本接口。

外部化配置和 Bean Registry

下图显示了 Spring Boot Externalized Configuration 在 spring-boot-camel-config quickstart 中的工作方式。



配置机制有以下主要部分：

属性源

提供注入到配置中的属性设置。默认属性源是应用程序的 application.properties 文件，这可以选择被 ConfigMap 对象或 Secret 对象覆盖。

配置属性 Bean

从属性源接收 configuraton 更新。配置属性 bean 是由 @Configuration 和 @ConfigurationProperties 注释的 Java bean decorated。

Spring bean registry

使用 requisite 注解时，在 Spring bean registry 中注册了配置属性 bean。

与 Camel bean registry 集成

Camel bean registry 会自动与 Spring bean registry 集成，以便在 Camel 路由中引用注册的 Spring Bean。

QuickstartConfiguration 类

fuse713-configmap 项目的配置属性 bean 定义为 QuickstartConfiguration Java 类（在 src/main/java/org/example/fis/ 目录下），如下所示：

```
package org.example.fis;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration (1)
@ConfigurationProperties(prefix = "quickstart") (2)
public class QuickstartConfiguration {

    /**
     * A comma-separated list of routes to use as recipients for messages.
     */
    private String recipients; (3)

    /**
     * The username to use when connecting to the async queue (simulation)
     */
    private String queueUsername; (4)

    /**
     * The password to use when connecting to the async queue (simulation)
     */
    private String queuePassword; (5)

    // Setters and Getters for Bean properties
    // NOT SHOWN
    ...
}
```

(1) @Configuration 注释会导致 QuickstartConfiguration 类实例化并在 Spring 中注册，作为 ID 为 quickstartConfiguration 的 bean。这自动使 bean 可以从 Camel 访问。例如，target-route-queue 路由可以使用 Camel 语法 ${bean:quickstartConfiguration?method=getQueueUsername} 来访问 queueUserName 属性。

(2) @ConfigurationProperties 注释定义了一个前缀 Quickstart，必须在属性源中定义属性值时使用。例如，属性文件将 recipients 属性引用为 quickstart.recipients。

(3)

4

queueUsername 属性可从属性源注入。

5

queuePassword 属性可从属性源注入。

11.2.3. 设置 Secret

此快速入门中的 Kubernetes Secret 是以标准的方式设置的，除了一个额外的步骤外：Spring Cloud Kubernetes 插件必须使用 Secret 的挂载路径进行配置，以便可以在运行时读取 Secret。设置 Secret：

1. *创建示例 Secret 对象*
2. *为 Secret 配置卷挂载*
3. *配置 spring-cloud-kubernetes 以读取 Secret 属性*

Secret 对象示例

quickstart 项目提供了一个示例 Secret sample-secret.yml，如下所示。Secret 对象中的属性值始终采用 base64 编码（使用 base64 命令行工具）。当 Secret 挂载到 pod 文件系统中时，这些值会自动解码回纯文本。

sample-secret.yml 文件

```
apiVersion: v1
kind: Secret
metadata: 1
  name: camel-config
type: Opaque
data:
  # The username is 'myuser'
  quickstart.queue-username: bXI1c2VyCg== 2
  quickstart.queue-password: MWYyZDFlMmU2N2Rm 3
```

1

metadata.name: 标识 Secret。OpenShift 系统的其他部分使用此标识符来引用 Secret。

2

quickstart.queue-username ：它被注入到 quickstartConfiguration bean 的 queueUsername 属性中。该值必须 采用 base64 编码。

3

quickstart.queue-password ：它被注入到 quickstartConfiguration bean 的 queuePassword 属性中。该值必须 采用 base64 编码。

注意

Kubernetes 不允许在 CamelCase 中定义属性名称（它需要所有小写的属性名称）。要临时解决这个问题，请使用连字符的 queue-username，它 Spring Boot 与 queueUsername 匹配。这利用 Spring Boot 的 relaxed 绑定规则 进行外部化配置。

为 Secret 配置卷挂载

应用程序必须配置为在运行时加载 Secret，方法是将 Secret 配置为卷挂载。应用程序启动后，Secret 属性在文件系统的指定位置变为可用。应用的 deployment.yml 文件位于 src/main/jkube/ 目录下，它定义了 Secret 的卷挂载。

deployment.yml 文件

```
spec:
  template:
    spec:
      serviceAccountName: "qs-camel-config"
      volumes: 1
        - name: "camel-config"
          secret:
            # The secret must be created before deploying this application
            secretName: "camel-config"
      containers:
        -
          volumeMounts: 2
            - name: "camel-config"
```

```
          readOnly: true
          # Mount the secret where spring-cloud-kubernetes is configured to read it
          # see src/main/resources/bootstrap.yml
          mountPath: "/etc/secrets/camel-config"
        resources:
#         requests:
#           cpu: "0.2"
#           memory: 256Mi
#         limits:
#           cpu: "1.0"
#           memory: 256Mi
        env:
        - name: SPRING_APPLICATION_JSON
          value: '{"server":{"undertow":{"io-threads":1, "worker-threads":2 }}}'
```

1

在 volumes 部分中，部署声明一个名为 camel-config 的新卷，它引用名为 camel-config 的 Secret。

2

在 volumeMounts 部分中，部署声明一个新的卷挂载，它引用 camel-config 卷，并指定 Secret 卷应挂载到 pod 文件系统的路径 /etc/secrets/camel-config 中。

配置 spring-cloud-kubernetes 以读取 Secret 属性

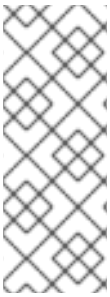
要将 secret 与 Spring Boot 外部化配置集成，Spring Cloud Kubernetes 插件必须配置有 secret 的挂载路径。Spring Cloud Kubernetes 从指定位置读取 secret，并将其提供给 Spring Boot 作为属性源。Spring Cloud Kubernetes 插件由 bootstrap.yml 文件中的设置进行配置，该文件位于 quickstart 项目中的 src/main/resources 下。

bootstrap.yml 文件

```
# Startup configuration of Spring-cloud-kubernetes
spring:
  application:
    name: camel-config
  cloud:
    kubernetes:
      reload:
        # Enable live reload on ConfigMap change (disabled for Secrets by default)
        enabled: true
      secrets:
        paths: /etc/secrets/camel-config
```

-

spring.cloud.kubernetes.secrets.paths 属性指定 pod 中 secret 卷挂载的路径列表。



注意

bootstrap.properties 文件（或 bootstrap.yml 文件）的行为与 application.properties 文件类似，但会在应用程序启动的早期阶段载入。在 bootstrap.properties 文件中设置与 Spring Cloud Kubernetes 插件相关的属性更为可靠。

11.2.4. 设置 ConfigMap

除了创建 ConfigMap 对象并正确设置 view 权限外，与 Spring Cloud Kubernetes 集成还需要将 ConfigMap 的 metadata.name 与项目 bootstrap.yml 文件中配置的 spring.application.name 属性的值匹配。设置 ConfigMap：

- 创建示例 ConfigMap 对象
- 设置视图权限
- 配置 Spring Cloud Kubernetes 插件

ConfigMap 对象示例

quickstart 项目提供了一个 ConfigMap sample-configmap.yml 示例。

```
kind: ConfigMap
apiVersion: v1
metadata: (1)
  # Must match the 'spring.application.name' property of the application
  name: camel-config
data:
  application.properties: | (2)
    # Override the configuration properties here
    quickstart.recipients=direct:async-queue,direct:file,direct:mail (3)
```

(1)

2

data.application.properties：本节列出了可覆盖使用应用程序部署的原始 application.properties 文件中的设置的属性设置。

3

quickstart.recipients ：它被注入到 quickstartConfiguration bean 的 recipients 属性中。

设置查看权限

如 Secret 的 deployment.yml 文件中所示， serviceAccountName 在项目的 deployment.yml 文件中设置为 qs-camel-config。因此，您需要输入以下命令来启用 Quickstart 应用的查看权限（假设它部署到 test 项目命名空间中）：

```
oc policy add-role-to-user view system:serviceaccount:test:qs-camel-config
```

配置 Spring Cloud Kubernetes 插件

Spring Cloud Kubernetes 插件由 bootstrap.yml 文件中的以下设置进行配置。

spring.application.name

这个值必须与 ConfigMap 对象的 metadata.name 匹配（例如，在 Quickstart 项目中的 sample-configmap.yml 中定义）。默认为应用程序。

spring.cloud.kubernetes.reload.enabled

把它设置为 true 可动态重新加载 ConfigMap 对象。

有关支持的属性的详情，请参阅 PropertySource Reload Configuration Properties。

11.3. 使用 CONFIGMAP PROPERTYSOURCE

Kubernetes 具有将配置传递给应用程序的 ConfigMap 的概念。Spring cloud Kubernetes 插件提供与 ConfigMap 集成，使配置映射可以被 Spring Boot 访问。

启用时 ConfigMap PropertySource 将查找由应用程序命名的 ConfigMap （请参阅 spring.application.name）。如果找到该映射，它将读取其数据并执行以下操作：

- 应用单个属性
- apply Property Named application.yaml
- apply Property Named application.properties

11.3.1. 应用单个属性

假设我们有一个名为 demo 的 Spring Boot 应用程序，它使用属性读取其线程池配置。

- pool.size.core
- pool.size.max

这可以以 YAML 格式外部化到配置映射：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: demo
data:
  pool.size.core: 1
  pool.size.max: 16
```

11.3.2. 应用 application.yaml ConfigMap 属性

对于大多数情况，单个属性可以正常工作，但有时我们找到 YAML 更为方便。在这种情况下，我们使用名为 application.yaml 的单个属性，并将 YAML 嵌入到其中：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: demo
data:
  application.yaml: |-
    pool:
      size:
        core: 1
        max:16
```

-

11.3.3. 应用 application.properties ConfigMap 属性

您还可以在 Spring Boot application.properties 文件的样式中定义 ConfigMap 属性。在这种情况下，我们使用名为 application.properties 的单个属性，并列出其中的属性设置：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: demo
data:
  application.properties: |-
    pool.size.core: 1
    pool.size.max: 16
```

11.3.4. 部署 ConfigMap

要部署 ConfigMap 并使它可以被 Spring Boot 应用程序访问，请执行以下步骤。

流程

1. 在 Spring Boot 应用程序中，使用外部配置机制 来访问 ConfigMap 属性源。例如，通过为带有 @Configuration 注释的 Java bean 标注，bean 的属性值可能会由 ConfigMap 注入。

2. 在项目的 bootstrap.properties 文件（或 bootstrap.yaml 文件）中，设置 spring.application.name 属性以匹配 ConfigMap 的名称。

3. 对与应用程序关联的服务帐户启用 查看 权限（默认情况下，这是名为 default的服务帐户）。例如，将 view 权限添加到 default 服务帐户：

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project -q)
```

11.4. 使用 SECRETS PROPERTYSOURCE

Kubernetes 具有用于存储敏感数据的 机密 的概念，如密码、OAuth 令牌等。Spring cloud Kubernetes 插件提供与 Secret 集成，使 secret 可以被 Spring Boot 访问。

启用时的 Secrets 属性源将从以下源查找 Secret ：如果找到 secret，则会将其数据提供给应用程序。

1. *从 secret 挂载中递归读取*

2. *在应用程序后命名（请参阅 spring.application.name）*

3. *匹配一些标签*

请注意，默认情况下，不启用 通过 API （点 2 和 3）消耗 Secret。

11.4.1. 设置 Secret 的示例

假设我们有一个名为 demo 的 Spring Boot 应用程序，它使用属性来读取其 ActiveMQ 和 PostreSQL 配置。

```
amq.username
amq.password
pg.username
pg.password
```

这些 secret 可以外部化为 YAML 格式的 Secret ：

ActiveMQ Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: activemq-secrets
  labels:
    broker: activemq
type: Opaque
data:
  amq.username: bXI1c2VyCg==
  amq.password: MWYyZDFlMmU2N2Rm
```

PostreSQL Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secrets
  labels:
    db: postgres
type: Opaque
```

```
data:
  pg.username: dXNlcgo=
  pg.password: cGdhZG1pbgo=
```

11.4.2. 使用 secret

您可以以多种方式选择要使用的 Secret：

- 通过列出映射 secret 的目录：

```
-Dspring.cloud.kubernetes.secrets.paths=/etc/secrets/activemq,etc/secrets/postgres
```

如果您的所有 secret 都映射到一个通用 root，您可以设置它们，如下所示：

```
-Dspring.cloud.kubernetes.secrets.paths=/etc/secrets
```

- 通过设置命名 secret：

```
-Dspring.cloud.kubernetes.secrets.name=postgres-secrets
```

- 通过定义标签列表：

```
-Dspring.cloud.kubernetes.secrets.labels.broker=activemq
-Dspring.cloud.kubernetes.secrets.labels.db=postgres
```

11.4.3. Secrets PropertySource 的配置属性

您可以使用以下属性来配置 Secrets 属性源：

spring.cloud.kubernetes.secrets.enabled

启用 Secrets 属性源。type 是 布尔值，默认为 true。

spring.cloud.kubernetes.secrets.name

设置要查找的机密的名称。类型为 String，默认为 ${spring.application.name}。

spring.cloud.kubernetes.secrets.labels

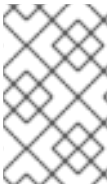
设置用于查找 secret 的标签。此属性的行为 基于映射的绑定。type 是 java.util.Map，默认为 null。

spring.cloud.kubernetes.secrets.paths

设置挂载 secret 的路径。此属性的行为 基于集合的绑定。type 是 java.util.List，默认为 null。

spring.cloud.kubernetes.secrets.enableApi

通过 API 启用/禁用消耗 secret。type 是 布尔值，默认为 false。



注意

出于安全原因，通过 API 访问 secret 可能会受到限制，首选将 secret 挂载到 POD。

11.5. 使用 PROPERTYSOURCE RELOAD

有些应用程序可能需要检测外部属性源的更改，并更新其内部状态以反映新的配置。Spring Cloud Kubernetes 的 reload 功能可以在相关的 ConfigMap 或 Secret 发生变化时触发应用程序重新加载。

11.5.1. 启用 PropertySource Reload

Spring Cloud Kubernetes 的 PropertySource reload 功能默认为禁用。

流程

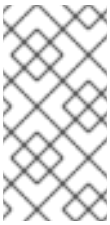
1. 导航到 Quickstart 项目的 src/main/resources 目录，再打开 bootstrap.yml 文件。
2. 更改配置属性 spring.cloud.kubernetes.reload.enabled=true。

11.5.2. PropertySource Reload 级别

以下级别的 reload 支持属性 spring.cloud.kubernetes.reload.strategy ：

refresh

（默认） 仅重新加载标有 @ConfigurationProperties 或 @RefreshScope 的配置 Bean。此重新加载级别利用 Spring Cloud 上下文的刷新功能。



注意

当重新加载策略设置为 刷新 时，PropertySource reload 功能只能用于 简单 属性（即不是集合）。在运行时无法更改由集合支持的属性。

restart_context

整个 Spring ApplicationContext 已安全重启。使用新配置重新创建 Bean。

shutdown

Spring ApplicationContext 被关闭，以激活容器重启。使用这个级别时，请确保所有非守护进程线程的生命周期都绑定到 ApplicationContext，并且将复制控制器或副本集配置为重启 pod。

11.5.3. PropertySource Reload 示例

以下示例说明了在启用重新加载功能时会发生什么。

流程

1. *假定使用默认设置(刷新 模式)启用了重新加载功能。配置映射更改时将刷新以下 bean：*

```
@Configuration
@ConfigurationProperties(prefix = "bean")
public class MyConfig {

    private String message = "a message that can be changed live";

    // getter and setters

}
```

2. *要查看发生的更改，请创建另一个 bean 以定期打印消息，如下所示。*

```
@Component
public class MyBean {

    @Autowired
    private MyConfig config;

    @Scheduled(fixedDelay = 5000)
    public void hello() {
```

```
        System.out.println("The message is: " + config.getMessage());
    }
}
```

3. 您可以使用 ConfigMap 更改应用程序打印的消息，如下所示。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: reload-example
data:
  application.properties: |-
    bean.message=Hello World!
```

与 pod 关联的 Config Map 中名为 bean.message 的属性的任何更改都会反映在程序的输出中。

11.5.4. PropertySource Reload 操作模式

重新载入功能支持两种操作模式：

event

（默认）监视使用 Kubernetes API (Web 套接字)的 ConfigMap 或 secret 中的更改。任何事件都将在配置上生成重新检查，并在发生更改时重新加载。需要服务帐户的 view 角色来侦听配置映射更改。更高级别角色（例如：secret 需要编辑）（默认不会监控secrets）。

轮询

定期从配置映射和 secret 重新创建配置，以查看它是否已改变。可以使用属性 spring.cloud.kubernetes.reload.period 配置轮询周期，默认为 15 秒。它要求角色与被监控的属性源相同。例如，这意味着，对已挂载的 secret 源的文件使用轮询不需要特定的特权。

11.5.5. PropertySource Reload 配置属性

以下属性可用于配置重新载入功能：

spring.cloud.kubernetes.reload.enabled

启用监控属性源和配置重新加载。type 是 布尔值，默认为 false。

spring.cloud.kubernetes.reload.monitoring-config-maps

允许在配置映射中监控更改。type 是 布尔值，默认为 true。

spring.cloud.kubernetes.reload.monitoring-secrets

允许监控 secret 中的更改。type 是 布尔值，默认为 false。

spring.cloud.kubernetes.reload.strategy

触发重新加载时使用的策略(刷新、restart_context、shutdown)。type 是 Enum，默认为 refresh。

spring.cloud.kubernetes.reload.mode

指定如何侦听属性源(事件、轮询)。type 是 Enum，默认为 事件。

spring.cloud.kubernetes.reload.period

使用 轮询 策略时验证更改的周期（毫秒）。类型为 Long，默认为 15000。

请注意以下几点：

- *spring.cloud.kubernetes.reload gem 属性不应在 ConfigMap 或 Secret 中使用。在运行时更改这些属性可能会导致意外的结果；*
- *删除属性或整个配置映射在使用 刷新 级别时不会恢复 Bean 的原始状态。*

第 12 章 为 KARAF 镜像开发应用程序

本教程介绍了如何为 Karaf 镜像创建和部署应用。

12.1. 使用 MAVEN ARCHETYPE 创建 KARAF 项目

要使用 Maven archetype 创建 Karaf 项目，请按照以下步骤操作：

流程

1. 进入系统中的相应目录。

2. 启动 Maven 命令以创建 Karaf 项目

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-7_13_0-00014-redhat-00001/archetypes-catalog-2.2.0.fuse-7_13_0-00014-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=karaf-camel-log-archetype \
  -DarchetypeVersion=2.2.0.fuse-7_13_0-00014-redhat-00001
```

3. archetype 插件切换到交互模式来提示您输入剩余的字段

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse713-karaf-camel-log
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse713-karaf-camel-log
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

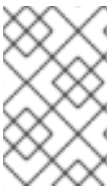
出现提示时，为 groupId 值输入 org.example.fis，为 artifactId 值输入 fuse713-karaf-camel-log。接受剩余字段的默认值。

4. 如果上述命令以 BUILD SUCCESS 状态退出，则现在您应该在 fuse713-karaf-camel-log 子目录下在 OpenShift 项目中有一个新的 Fuse。

5.

现在，您可以构建和部署 **fuse713-karaf-camel-log** 项目。假设您仍已登录到 **OpenShift**，请更改到 **fuse713-karaf-camel-log** 项目的目录，然后构建并部署项目，如下所示：

```
cd fuse713-karaf-camel-log
mvn oc:deploy -Popenshift
```



注意

有关可用 **Karaf archetypes** 的完整列表，请参阅 **Karaf Archetype Catalog**。

12.2. CAMEL KARAF 应用程序的结构

Camel Karaf 应用程序的目录结构如下：

```
├── pom.xml 1
├── README.md
├── configuration
│   └── settings.xml
└── src
    ├── main
    │   ├── jkube
    │   │   └── deployment.yml 2
    │   ├── java
    │   │   └── org
    │   │       └── example
    │   │           └── fis
    │   └── resources
    │       ├── assembly
    │       │   └── etc
    │       │       └── org.ops4j.pax.logging.cfg 3
    │       └── OSGI-INF
    │           └── blueprint
    │               └── camel-log.xml 4
    └── test
        └── java
            └── org
                └── example
                    └── fis
```

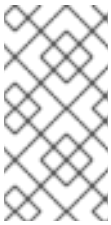
以下文件对于开发 **Karaf** 应用程序非常重要：

1

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
</dependency>
```

2

src/main/jkube/deployment.yml ：提供与 openshift-maven-plugin 生成的默认 OpenShift 配置文件合并的额外配置。



注意

此文件不作为 Karaf 应用的一部分使用，但在所有快速入门中都用来限制 CPU 和内存用量等资源。

3

org.ops4j.pax.logging.cfg: Demonstrates 如何自定义日志级别，将日志级别设置为 DEBUG，而不是默认的 INFO。

4

camel-log.xml：包含应用程序的源代码。

12.3. KARAF ARCHETYPE 目录

Karaf archetype 目录包含以下示例。

表 12.1. Karaf Maven Archetypes

Name	描述
karaf-camel-amq-archetype	演示了如何使用 Camel amq 组件向 Apache ActiveMQ 消息代理发送和接收消息。
karaf-camel-log-archetype	演示一个简单的 Apache Camel 应用程序，该应用程序将消息记录到服务器每 5 秒记录一次。
karaf-camel-rest-sql-archetype	演示如何通过 JDBC 和 Camel 的 REST DSL 使用 SQL 来公开 RESTful API。

Name	描述
karaf-cxf-rest-archetype	演示如何使用 CXF 创建 RESTful (JAX-RS) Web 服务，并通过 OSGi HTTP 服务公开该服务。

12.4. 使用 FABRIC8 KARAF 功能

Fabric8 为 Apache Karaf 提供支持，使开发用于 Kubernetes 的 OSGi 应用变得更加简单。

Fabric8 的主要功能如下：

- ***用于解析蓝图 XML 文件中的占位符的不同策略。***
- ***环境变量***
- ***系统属性***
- ***服务***
- ***Kubernetes ConfigMap***
- ***Kubernetes Secret***
- ***使用 Kubernetes 配置映射动态更新 OSGi 配置管理。***
- ***为 OSGi 服务提供 Kubernetes 热检查。***

12.4.1. 添加 Fabric8 Karaf 功能

要使用这些功能，请将 fabric8-karaf-features 依赖项添加到项目 POM 文件。

流程

1. *打开项目的 pom.xml 文件，并添加 fabric8-karaf-features 依赖项。*

```
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-karaf-features</artifactId>
  <version>${fabric8.version}</version>
  <classifier>features</classifier>
  <type>xml</type>
</dependency>
```

fabric8 karaf 功能将安装到 Karaf 服务器中。

12.4.2. 添加 Fabric8 Karaf 核心捆绑包功能

bundle fabric8-karaf-core 提供 Blueprint 和 ConfigAdmin 扩展使用的功能。

流程

1. *打开项目的 pom.xml，并将 fabric8-karaf-core 添加到 startupFeatures 部分。*

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-core</feature>
  ...
</startupFeatures>
```

这将在自定义 Karaf 分发中添加 fabric8-karaf-core 功能。

12.4.3. 设置属性 Placeholder 服务选项

bundle fabric8-karaf-core 使用以下接口导出服务 PlaceholderResolver ：

```
public interface PlaceholderResolver {
    /**
     * Resolve a placeholder using the strategy indicated by the prefix
     *
     * @param value the placeholder to resolve
     * @return the resolved value or null if not resolved
     */
    String resolve(String value);
```

```
    /**
     * Replaces all the occurrences of variables with their matching values from the resolver
using the given source string as a template.
     *
     * @param source the string to replace in
     * @return the result of the replace operation
     */
    String replace(String value);

    /**
     * Replaces all the occurrences of variables within the given source builder with their
matching values from the resolver.
     *
     * @param value the builder to replace in
     * @rerurn true if altered
     */
    boolean replaceIn(StringBuilder value);

    /**
     * Replaces all the occurrences of variables within the given dictionary
     *
     * @param dictionary the dictionary to replace in
     * @rerurn true if altered
     */
    boolean replaceAll(Dictionary<String, Object> dictionary);

    /**
     * Replaces all the occurrences of variables within the given dictionary
     *
     * @param dictionary the dictionary to replace in
     * @rerurn true if altered
     */
    boolean replaceAll(Map<String, Object> dictionary);
}
```

PlaceholderResolver 服务充当不同属性占位符解析策略的收集器。默认情况下提供的解析策略列在表 解析策略 中。要设置属性占位符服务选项，您可以使用系统属性或环境变量或两者。

流程

1. 要访问 OpenShift 上的 ConfigMap，服务帐户需要查看权限。为服务帐户添加查看权限。

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project
-q)
```

2. 将 secret 挂载到 Pod，以便通过 API 访问 secret。

3. Pod 上作为卷挂载提供的 secret 映射到名为 secret 的目录，如下所示

```
containers:
 -
  env:
  - name: FABRIC8_K8S_SECRETS_PATH
    value: /etc/secrets
    volumeMounts:
  - name: activemq-secret-volume
    mountPath: /etc/secrets/activemq
    readOnly: true
  - name: postgres-secret-volume
    mountPath: /etc/secrets/postgres
    readOnly: true

volumes:
 - name: activemq-secret-volume
 secret:
 secretName: activemq
 - name: postgres-secret-volume
  secret:
 secretName: postgres
```

12.4.4. 添加自定义属性占位符解析器

您可以添加自定义占位符解析器来支持特定的需求，如自定义加密。您还可以使用 PlaceholderResolver 服务将解析器提供给 Blueprint 和 ConfigAdmin。

流程

1. 将以下 mvn 依赖项添加到项目 pom.xml 中。

pom.xml

```
---
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-karaf-core</artifactId>
</dependency>
---
```

2. 实施 PropertiesFunction 接口，并使用 SCR 将其注册为 OSGi 服务。

```
import io.fabric8.karaf.core.properties.function.PropertiesFunction;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.ConfigurationPolicy;
import org.apache.felix.scr.annotations.Service;

@Component(
    immediate = true,
    policy = ConfigurationPolicy.IGNORE,
    createPid = false
)
@Service(PropertiesFunction.class)
public class MyPropertiesFunction implements PropertiesFunction {
    @Override
    public String getName() {
        return "myResolver";
    }

    @Override
    public String apply(String remainder) {
        // Parse and resolve remainder
        return remainder;
    }
}
```

3. 您可以在配置管理中引用解析器，如下所示：

属性

```
my.property = $[myResolver:value-to-resolve]
```

12.4.5. 解析策略列表

PlaceholderResolver 服务充当不同属性占位符解析策略的收集器。默认情况下提供的解析策略列在表中。

1. 解析策略列表

prefix	Example	描述
env	**ENV:JAVA_HOME**	从 OS 环境变量中查找属性。
'sys	**sys:java.version**	从 Java JVM 系统属性中查找属性。
'service	**service:amq**	使用服务命名约定，从 OS 环境变量中查找属性。
service.host	**service.host:amq**	使用服务命名规则从 OS 环境变量中查找属性，仅返回主机名部分。
service.port	**service.port:amq**	使用服务命名约定，从 OS 环境变量中查找属性，仅返回端口部分。
k8s:map	**k8s:map:myMap/myKey**	从 Kubernetes ConfigMap （通过 API）中查找属性
k8s:secret	**k8s:secret:amq/password**	从 Kubernetes Secret （通过 API 或卷挂载）中查找属性

12.4.6. Property Placeholder 服务选项列表

属性占位符服务支持以下选项：

1. ***属性占位符服务选项列表***

Name	default	描述
fabric8.placeholder.prefix	$[	占位符前缀
fabric8.placeholder.suffix	]	占位符后缀
fabric8.k8s.secrets.path	null	以逗号分隔的路径列表，其中 secret 被映射
fabric8.k8s.secrets.api.enabled	false	通过 API 启用/禁用消耗 secret

12.5. 添加 FABRIC8 KARAF 配置管理员支持

12.5.1. 添加 Fabric8 Karaf 配置管理员支持

您可以将 Fabric8 Karaf 配置管理员支持添加到自定义 Karaf 分发中。

流程

- 打开项目的 pom.xml，并将 fabric8-karaf-cm 添加到 startupFeatures 部分。

pom.xml

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-cm</feature>
  ...
</startupFeatures>
```

12.5.2. 添加 ConfigMap 注入

fabric8-karaf-cm 提供了一个 ConfigAdmin 网桥，可在 Karaf 的 ConfigAdmin 中注入 ConfigMap 值。

流程

1. 要通过 ConfigAdmin 网桥添加，ConfigMap 必须使用 karaf.pid 标记。karaf.pid 值对应于组件的 pid。例如，

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: myconfig
  labels:
    karaf.pid: com.mycompany.bundle
data:
  example.property.1: my property one
  example.property.2: my property two
```

2. 要定义配置，您可以使用单个属性名称。个别属性适用于大多数情况。它与 karaf/etc 中的 pid 文件相同。例如，

```
kind: ConfigMap
```

```
apiVersion: v1
metadata:
  name: myconfig
  labels:
    karaf.pid: com.mycompany.bundle
data:
  com.mycompany.bundle.cfg: |
    example.property.1: my property one
    example.property.2: my property two
```

12.5.3. 配置插件

fabric8-karaf-cm 提供了一个 ConfigurationPlugin，它解析配置属性占位符。

要启用使用 fabric8-karaf-cm 插件的属性替换，您必须将 Java 属性 fabric8.config.plugin.enabled 设置为 true。例如，您可以使用 Karaf 镜像中的 JAVA_OPTIONS 环境变量设置此属性，如下所示：

```
JAVA_OPTIONS=-Dfabric8.config.plugin.enabled=true
```

12.5.4. config Property Placeholders

配置属性占位符示例如下所示：

my.service.cfg

```
amq.usr = $[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/username]
amq.pwd = $[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/password]
amq.url = tcp://$[env+service:ACTIVEMQ_SERVICE_NAME]
```

my-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0"
        xsi:schemaLocation="
          http://www.osgi.org/xmlns/blueprint/v1.0.0
```

```
            https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
            http://camel.apache.org/schema/blueprint
            http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">

  <cm:property-placeholder persistent-id="my.service" id="my.service" update-strategy="reload"/>

  <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
    <property name="userName"  value="${amq.usr}"/>
    <property name="password"  value="${amq.pwd}"/>
    <property name="brokerURL" value="${amq.url}"/>
  </bean>
</blueprint>
```

12.5.5. Fabric8 Karaf Config Admin 选项

Fabric8 Karaf Config Admin 支持以下选项：

Name	default	描述
fabric8.config.plugin.enabled	false	enable ConfigurationPlugin
fabric8.cm.bridge.enabled	true	启用 ConfigAdmin 网桥
fabric8.config.watch	true	启用查看 ConfigMap 更改
fabric8.config.merge	false	在 ConfigAdmin 中启用合并 ConfigMap 值
fabric8.config.meta	true	在 ConfigAdmin 网桥中启用注入 ConfigMap meta
fabric8.pid.label	karaf.pid	定义 ConfigAdmin 网桥查找的标签（即，需要选择的 ConfigMap 必须具有该标签；该值决定了它所关联的 PID 的值）

Name	default	描述
fabric8.pid.filters	empty	为 ConfigAdmin 网桥定义额外的条件，以选择 ConfigMap。支持的语法是： ● 不同标签中的条件由 "," 分隔，并相互间预期。 ● 在标签内，分号(;)被视为 OR，并可用作标签值中条件的分隔符。 例如，一个过滤器（如 -Dfabric8.pid.filters=appName=A;B,database.name=my.oracle.datasource ）转换为 "give me all ConfigMaps，值为 A 或 B，标签 database.name 等于 my.oracle.datasource"。

重要

ConfigurationPlugin 需要 Aries Blueprint CM 1.0.9 或更高版本。

12.6. 添加 FABRIC8 KARAF 蓝图支持

fabric8-karaf-blueprint 使用来自 fabric8-karaf-core 的 Aries PropertyEvaluator 和属性占位符来解析 Blueprint XML 文件中的占位符。

流程

- *要在自定义 Karaf 分发中包含蓝图支持的功能，请将 fabric8-karaf-blueprint 添加到 pom.xml 项目的 startupFeatures 部分。*

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-blueprint</feature>
  ...
</startupFeatures>
```

Example

fabric8 evaluator 支持串联的 evaluators，如 ${env+service:MY_ENV_VAR}。您需要根据环境变量解析 MY_ENV_VAR 变量。然后，使用服务功能解析结果。例如，

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.2.0"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0
             https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
             http://camel.apache.org/schema/blueprint
             http://camel.apache.org/schema/blueprint/camel-blueprint.xsd
             http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.3.0
             http://aries.apache.org/schemas/blueprint-ext/blueprint-ext-1.3.xsd">

  <ext:property-placeholder evaluator="fabric8" placeholder-prefix="$[" placeholder-suffix="]"/>

  <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
    <property name="userName"
value="$[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/username]"/>
    <property name="password"
value="$[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/password]"/>
    <property name="brokerURL" value="tcp://$[env+service:ACTIVEMQ_SERVICE_NAME]"/>
  </bean>
</blueprint>
```

重要

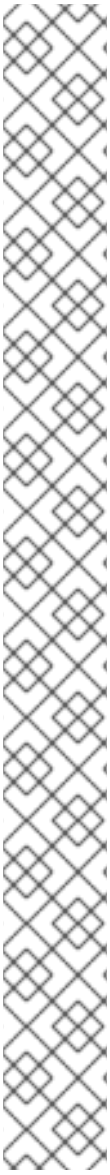
嵌套属性占位符替换需要 Aries Blueprint Core 1.7.0 或更高版本。

12.7. 启用 FABRIC8 KARAF 健康检查

建议将 fabric8-karaf-checks 安装为启动功能。启用后，您的 Karaf 服务器可以公开这些 URL：

- **http://0.0.0.0:8181/readiness-check and**
- **http://0.0.0.0:8181/health-check**

它们可供 Kubernetes 用于就绪度和存活度探测。



注意

这些 URL 只有在满足以下条件时，才会以 HTTP 200 状态代码响应：

- OSGi 框架已启动。
- 所有 OSGi 捆绑包都已启动。
- 所有引导功能都已安装。
- 所有部署的 BluePrint 捆绑包都处于创建的状态。
- 所有部署的 SCR 捆绑包都处于 active、registered 或 factory 状态。
- 所有 Web 捆绑包都部署到 Web 服务器。
- 所有创建的 Camel 上下文都处于 started 状态。

流程

1. 打开项目的 pom.xml，并在 startupFeatures 部分中添加 fabric8-karaf-checks 功能。

pom.xml

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-checks</feature>
  ...
</startupFeatures>
```

oc:resources 目标将检测您使用 fabric8-karaf-checks 功能，并将 Kubernetes for readiness 和 liveness 探测自动添加到您的容器的配置中。

12.7.1. 配置健康检查

默认情况下，fabric8-karaf-checks 端点注册到在端口 8181 上运行的内置 HTTP 服务器引擎 (Undertow)。为了避免在容器中运行 HTTP 进程的健康和就绪度检查请求被阻止，端点可以注册到单独的 Undertow 容器中。

这些检查可以通过设置以下属性在 etc/io.fabric8.checks.cfg 文件中配置：

- httpPort: 如果指定了此属性并且是有效的端口号，则 readiness-check 和 health-check 端点将注册到 Undertow 服务器的单独的实例中
- readinessCheckPath 和 healthCheckPath 属性允许您配置可用于就绪度和健康检查的实际 URI。默认情况下，这些值与前面的值相同。

注意

这些属性可以在启动 Fuse-Karaf 后更改，但也可在 etc/io.fabric8.checks.cfg 文件中指定，该文件是希望拥有 fabric8-karaf-checks 功能的一部分，供希望使用 fabric8-karaf-checks 功能。

以下示例演示了在 etc/io.fabric8.checks.cfg 文件中配置健康和就绪属性：

Example

```
httpPort = 8182
readinessCheckPath = /readiness-check
healthCheckPath = /health-check
```

12.8. 添加自定义健康检查

您可以提供额外的自定义健康检查，以防止 Karaf 服务器在准备好处理请求前收到用户流量。要启用自定义健康检查，您需要实现 io.fabric8.karaf.checks.HealthChecker 或 io.fabric8.karaf.checks.ReadinessChecker 接口，并在 OSGi registry 中注册这些对象。

流程

- 将以下 mvn 依赖项添加到项目 pom.xml 文件中。

pom.xml

```
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-karaf-checks</artifactId>
</dependency>
```

注意

在 OSGi 注册表中创建和注册对象的最简单方法是使用 SCR。

Example

一个执行健康检查的示例，以确保您有一些可用磁盘空间，如下所示：

```
import io.fabric8.karaf.checks.*;
import org.apache.felix.scr.annotations.*;
import org.apache.commons.io.FileSystemUtils;
import java.util.Collections;
import java.util.List;

@Component(
    name = "example.DiskChecker",
    immediate = true,
    enabled = true,
    policy = ConfigurationPolicy.IGNORE,
    createPid = false
)
@Service({HealthChecker.class, ReadinessChecker.class})
public class DiskChecker implements HealthChecker, ReadinessChecker {

    public List<Check> getFailingReadinessChecks() {
        // lets just use the same checks for both readiness and health
```

```
        return getFailingHeathChecks();
    }

    public List<Check> getFailingHealthChecks() {
        long free = FileSystemUtils.freeSpaceKb("/");
        if (free < 1024 * 500) {
            return Collections.singletonList(new Check("disk-space-low", "Only " + free + "kb of
disk space left."));
        }
        return Collections.emptyList();
    }
}
```

第 13 章 为 JBOSS EAP 镜像开发应用程序

要在 JBoss EAP 上开发 Fuse 应用程序，替代方案是使用 S2I 源工作流为使用 EAP 的 Red Hat Camel CDI 创建 OpenShift 项目。

先决条件

- 确保 OpenShift 正确运行，并且 OpenShift 中已安装了 Fuse 镜像流。请参阅管理员入门。
- 确保为 fuse 配置 Maven 存储库，请参阅配置 Maven 存储库。

13.1. 使用 S2I 源工作流创建 JBOSS EAP 项目

要在 JBoss EAP 上开发 Fuse 应用程序，替代方案是使用 S2I 源工作流为使用 EAP 的 Red Hat Camel CDI 创建 OpenShift 项目。

流程

1. 将 view 角色添加到 default 服务帐户，以启用集群。这会授予用户对 default 服务帐户的 view 访问权限。每个项目都需要服务帐户来运行构建、部署和其他容器集。在 shell 提示符中输入以下 oc client 命令：

```
oc login -u developer -p developer
oc policy add-role-to-user view -z default
```

2. 查看 OpenShift 模板上已安装的 Fuse。

```
oc get template -n openshift
```

3. 输入以下命令来创建 使用 EAP quickstart 运行 Red Hat Fuse 7.13 Camel CDI 所需的资源。它为快速入门创建部署配置和构建配置。终端会显示有关 Quickstart 和创建的资源的信息。

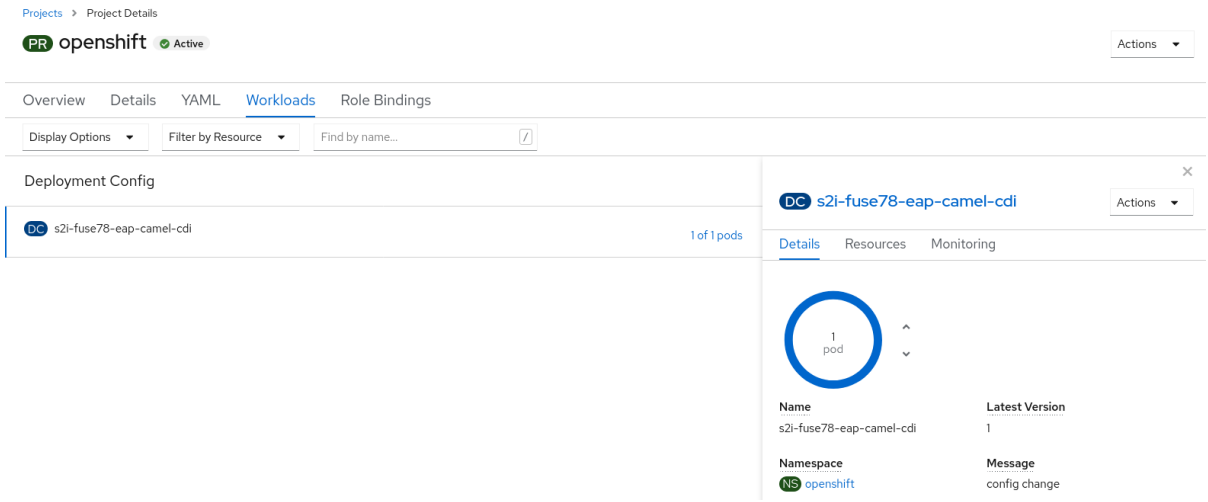
```
oc new-app s2i-fuse7-eap-camel-cdi

--> Creating resources ...
    service "s2i-fuse7-eap-camel-cdi" created
    service "s2i-fuse7-eap-camel-cdi-ping" created
    route.route.openshift.io "s2i-fuse7-eap-camel-cdi" created
    imagestream.image.openshift.io "s2i-fuse7-eap-camel-cdi" created
```

```
    buildconfig.build.openshift.io "s2i-fuse7-eap-camel-cdi" created
    deploymentconfig.apps.openshift.io "s2i-fuse7-eap-camel-cdi" created
--> Success
    Access your application via route 's2i-fuse7-eap-camel-cdi-OPENSHIFT_IP_ADDR'
    Build scheduled, use 'oc logs -f bc/s2i-fuse7-eap-camel-cdi' to track its progress.
    Run 'oc status' to view your app.
```

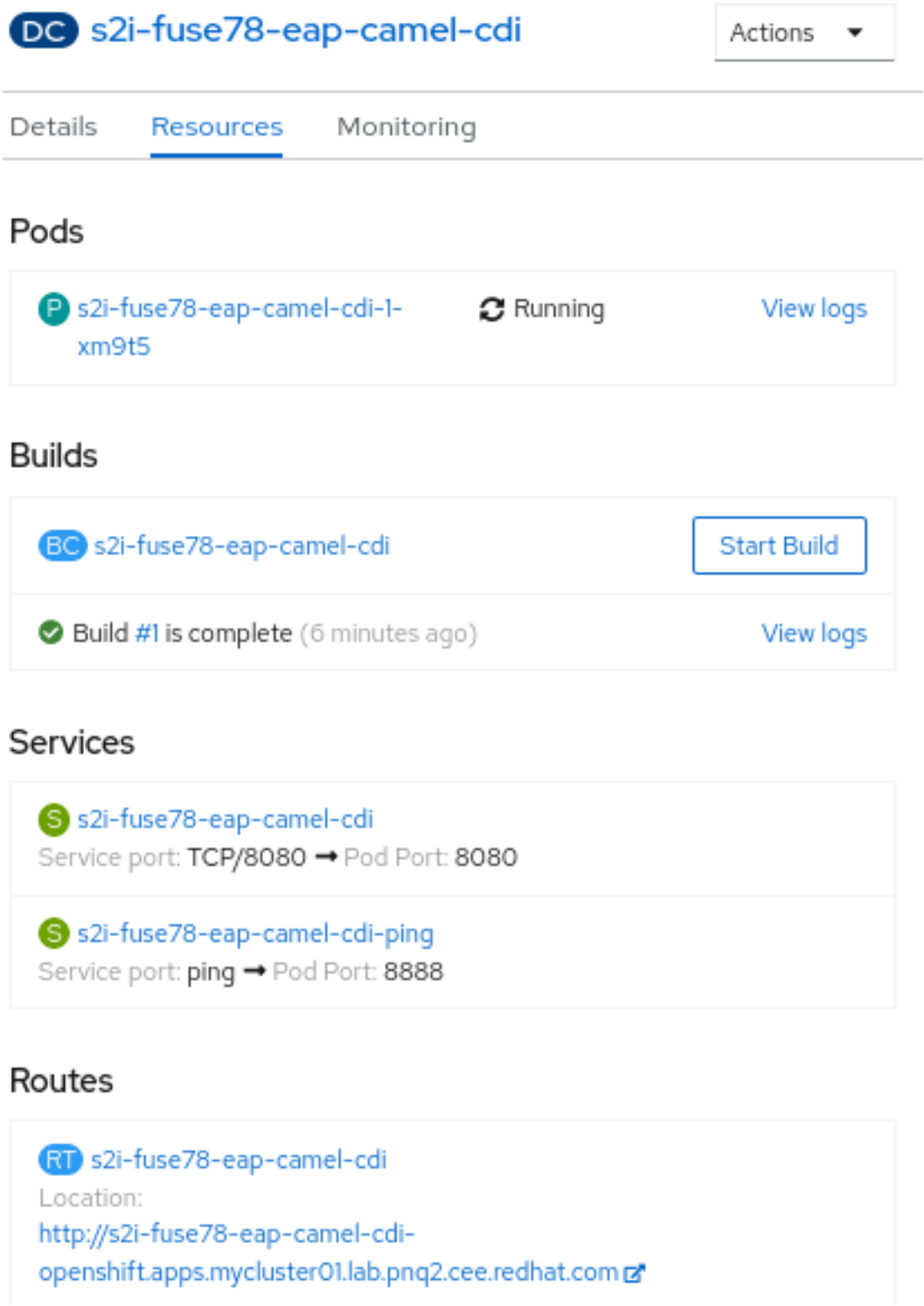
4. 导航到浏览器中的 OpenShift Web 控制台(https://OPENSHIFT_IP_ADDR，将 OPENSHIFT_IP_ADDR 替换为集群的 IP 地址)，并使用您的凭证（例如，使用用户名 developer和密码 developer ）登录控制台。

5. 在左侧面板中，展开 Home。单击 Status 以查看 Project Status 页面。此时会显示所选命名空间中的所有现有应用程序（如 openshift）。

6. 单击 s2i-fuse7-eap-camel-cdi，以查看 Quickstart 的 Overview 信息页面。



7. 点 Resources 选项卡，然后点 Routes 部分显示的链接来访问应用程序。



链接的格式为 http://s2i-fuse7-eap-camel-cdi-OPENSHIFT_IP_ADDR。在浏览器中显示类似如下的消息：

> *Hello world from 172.17.0.3*

8. *您还可以使用 URL 中的 name 参数指定名称。例如，如果您在浏览器中输入 URL http://s2i-fuse7-eap-camel-cdi-openshift.apps.cluster-name.openshift.com/?name=jdoe，您会看到*

响应：

```
Hello jdoe from 172.17.0.3
```

9. *点 View Logs 查看应用程序的日志。*

10. *关闭正在运行的 pod，*

    a. *单击 Overview 选项卡，以返回到应用的概览信息页面。*

    b. *点 Desired Count 旁边的*

    图标。此时会显示 Edit Count 窗口。

    c. *使用向下箭头缩减至零以停止容器集。*

13.2. JBOSS EAP 应用程序的结构

您可以在以下位置找到带有 EAP 示例的 Red Hat Fuse 7.13 Camel CDI 的源代码：

```
https://github.com/wildfly-extras/wildfly-camel-examples/tree/wildfly-camel-examples-5.2.0.fuse-720021/camel-cdi
```

Camel on EAP 应用程序的目录结构如下：

```
├── pom.xml
├── README.md
├── configuration
│   └── settings.xml
└── src
    └── main
        ├── java
        │   └── org
        │       └── wildfly
        │           └── camel
        │               └── examples
        │                   └── cdi
        │                       └── camel
```

```
        ├── MyRouteBuilder.java
        ├── SimpleServlet.java
        └── SomeBean.java
    └── webapp
        └── WEB-INF
            └── beans.xml
```

以下文件对于开发 JBoss EAP 应用程序非常重要：

pom.xml

包括其他依赖项。

13.3. JBOSS EAP QUICKSTART 模板

以下 S2I 模板为 JBoss EAP 上的 Fuse 提供：

表 13.1. JBoss EAP S2I 模板

Name	描述
带有 EAP 的 JBoss Fuse 7.13 Camel A-MQ (**eap-camel-amq-template**)	演示使用 camel-activemq 组件连接到 OpenShift 中运行的 AMQ 消息代理。假设代理已部署。
带有 EAP 的 Red Hat Fuse 7.13 Camel CDI (**eap-camel-cdi-template**)	演示使用 camel-cdi 组件将 CDI Bean 与 Camel 路由集成。
Red Hat Fuse 7.13 CXF JAX-RS with EAP (**eap-camel-cxf-jaxrs-template**)	演示使用 camel-cxf 组件来生成和使用 JAX-RS REST 服务。
带有 EAP 的 Red Hat Fuse 7.13 CXF JAX-WS (**eap-camel-cxf-jaxws-template**)	演示使用 camel-cxf 组件来生成和使用 JAX-WS Web 服务。

第 14 章 在 OPENSHIFT 上的 FUSE 中使用持久性存储

OpenShift 应用的 Fuse 基于 OpenShift 容器，容器没有持久文件系统。每次启动应用程序时，都会在具有不可变 Docker 格式的镜像的新容器中启动。因此，当容器停止时，文件系统中的任何持久数据都会丢失。但是，应用程序需要将一些状态作为数据存储存储在持久存储中，有时应用程序会共享对常见数据存储的访问。OpenShift 平台支持将外部存储置备为持久存储。

14.1. 关于卷和卷类型

OpenShift 允许 Pod 和容器作为由多个本地或网络附加存储端点支持的文件系统 挂载卷。

卷类型包括：

- emptyDir （空目录）：这是默认卷类型。它是在本地主机上创建 pod 时分配的目录。它不会在服务器间复制，当您删除目录的 pod 时，也会复制它。
- ConfigMap ：它是一个目录，其内容填充了来自名为 configmap 的键值对。
- hostpath （主机目录）：它是在任何主机上具有特定路径的目录，需要提升的特权。
- Secret （挂载 secret）：Secret 卷将命名 secret 挂载到提供的目录中。
- PersistentVolumeClaim 或 pvc （持久性卷声明）：这会将容器中的卷目录链接到按名称分配的持久性卷声明。持久卷声明是分配存储的请求。请注意，如果您的声明没有绑定，您的 pod 不会启动。

卷在 Pod 级别配置，只能使用 hostPath 直接访问外部存储。因此，更难以管理作为 hostPath 卷的多个 Pod 的共享资源的访问权限。

14.2. 关于 PERSISTENTVOLUME

PersistentVolume 允许集群管理员置备集群范围的存储，这些存储由 NFS、Ceph RBD、AWS Elastic Block Store (EBS)等各种类型的网络存储提供支持。PersistentVolume 还指定容量、访问模式和循环策略。这允许来自多个项目的 pod 访问持久性存储，而无需担心底层资源的性质。

请参阅 配置持久性存储 以创建各种 PersistentVolume。

14.3. 配置持久性卷

您可以通过创建配置文件来置备持久性卷。然后，可通过创建 PersistentVolume 声明来访问此存储。

流程

1. 使用以下示例配置，创建名为 pv.yaml 的配置文件。这会将主机上的路径置备为名为 pv001 的 PersistentVolume。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 2Mi
  hostPath:
    path: /data/pv0001/
```

此处的主机路径为 /data/pv0001，存储容量限制为 2MB。例如，在使用 OpenShift CDK 时，它将从托管 OpenShift 集群的虚拟机调配 /data/pv0001 目录。

2. 创建 PersistentVolume。

```
oc create -f pv.yaml
```

3. 验证 PersistentVolume 的创建。这将列出 OpenShift 集群中配置的所有 PersistentVolume ：

```
oc get pv
```

14.4. 创建 PERSISTENTVOLUMECLAIMS

PersistentVolume 将存储端点作为 OpenShift 集群中命名实体公开。要从项目访问此存储，必须创建 PersistentVolumeClaims 来访问 PersistentVolume。使用特定访问模式的自定义存储声明为每个项目

创建 PersistentVolumeClaim。

流程

- *以下示例配置为 1MB 的存储创建一个名为 pvc0001 的声明，其对名为 pv0001 的 PersistentVolume 具有读写访问。*

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc0001
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Mi
```

14.5. 在 POD 中使用持久性卷

Pod 使用卷挂载来定义文件系统挂载位置和卷，以定义参考 PersistentVolumeClaims。

流程

1. *如下所示，创建示例容器配置，它将 PersistentVolumeClaim pvc0001 挂载到文件系统中的 /usr/share/data。*

```
spec:
  template:
    spec:
      containers:
        - volumeMounts:
          - name: vol0001
            mountPath: /usr/share/data
      volumes:
        - name: vol0001
          persistentVolumeClaim:
            claimName: pvc0001
```

应用程序写入目录 /usr/share/data 的任何数据现在都会在容器重启后保留。

2. *在 OpenShift 应用的 Fuse 中的 src/main/jkube/deployment.yml 文件中添加此配置，并使用以下命令创建 OpenShift 资源：*

```
mvn oc:resource-apply
```

3. *验证创建的 DeploymentConfiguration 是否具有卷挂载和卷。*

```
oc describe deploymentconfig <application-dc-name>
```

对于 OpenShift 快速入门上的 Fuse，将 & lt;application-dc-name& gt; 替换为 Maven 项目名称，如 spring-boot-camel。

第 15 章 在 OPENSHIFT 上修补 FUSE

您可能需要执行一个或多个以下任务，将 Fuse on OpenShift 产品设置为最新的补丁级别：

对 OpenShift 镜像上的 Fuse 进行补丁

更新 OpenShift 服务器上的 OpenShift 镜像上的 Fuse，以便新应用程序构建基于 Fuse 基础镜像的补丁版本。

使用 BOM 补丁应用程序依赖项

更新项目 POM 文件中的依赖项，以便应用程序使用 Maven 工件的补丁版本。

对 OpenShift 模板上的 Fuse 进行补丁

更新 OpenShift 服务器上的 Fuse on OpenShift 模板，以便在 OpenShift 模板中使用 Fuse 创建的新项目使用 Maven 工件的补丁版本。

15.1. BOM 和 MAVEN 依赖项的重要备注

在 OpenShift 上的 Fuse 中，应用程序完全使用从 Red Hat Maven 存储库下载的 Maven 工件构建。因此，为了修补应用程序代码，您需要做的都是编辑项目的 POM 文件，将 Maven 依赖项更改为在 OpenShift 补丁版本中使用适当的 Fuse。

升级 OpenShift 上 Fuse 的所有 Maven 依赖项非常重要，以便您的项目使用同一补丁版本中的所有依赖项。OpenShift 项目的 Fuse 由一组精心构建和测试的 Maven 工件组成。如果您尝试混合和匹配来自 OpenShift 补丁级别 的不同 Fuse 的 Maven 工件，则可能会最终使用未经测试且不受红帽不支持的配置。避免这种情况的最简单方法是，在 Maven 中使用 Bill of Materials (BOM)文件，该文件定义了 OpenShift 上 Fuse 支持的所有 Maven 工件的版本。当您更新 BOM 文件版本时，会自动更新项目的 POM 中 OpenShift Maven 工件上所有 Fuse 的版本。

由 Fuse 在 OpenShift Maven archetype 或 OpenShift 模板上的 Fuse 生成的 POM 文件具有标准布局，它使用 BOM 文件并定义某些所需插件的版本。建议您在您自己的应用程序中坚持这个标准布局，因为这样可更轻松地修补和升级应用程序的依赖项。

15.2. 在 OPENSHIFT 镜像上修补 FUSE

OpenShift 镜像上的 Fuse 独立于主要 Fuse 产品进行更新。如果 OpenShift 镜像上的 Fuse 需要任何补丁，则更新的镜像将在 OpenShift 镜像流上的标准 Fuse 上提供，您可以从 registry.redhat.io 下载更新的镜像。OpenShift 上的 Fuse 提供以下镜像流（由 OpenShift 镜像流名称标识）：

- ***fuse-java-openshift-rhel8***
- ***fuse-java-openshift-jdk11-rhel8***
- ***fuse-karaf-openshift-rhel8***
- ***fuse-eap-openshift-jdk8-rhel7***
- ***fuse-eap-openshift-jdk11-rhel8***
- ***fuse-console-rhel8***
- ***fuse-apicurito-generator-rhel8***
- ***fuse-apicurito-rhel8***

流程

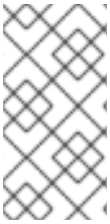
1. ***OpenShift 镜像流上的 Fuse 通常安装在 OpenShift 服务器上的 openshift 项目中。要检查 OpenShift 上 OpenShift 镜像上的 Fuse 状态，请以管理员身份登录 OpenShift，并输入以下命令：***

```
$ oc get is -n openshift
NAME                        DOCKER REPO                                  TAGS
UPDATED
fuse-console-rhel8          172.30.1.1:5000/openshift/fuse7/fuse-console-rhel8
1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
fuse7-eap-openshift-jdk8-rhel7   172.30.1.1:5000/openshift/fuse7/fuse-eap-openshift-jdk8-
rhel7       1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
fuse7-eap-openshift-jdk11-rhel8  172.30.1.1:5000/openshift/fuse7/fuse-eap-openshift-jdk11-
rhel8       1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
fuse7-java-openshift-rhel8       172.30.1.1:5000/openshift/fuse7/fuse-java-openshift-rhel8
1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
fuse7-java-openshift-jdk11-rhel8 172.30.1.1:5000/openshift/fuse7/fuse-java-openshift-jdk11-
rhel8       1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
fuse7-karaf-openshift-rhel8      172.30.1.1:5000/openshift/fuse7/fuse-karaf-openshift-rhel8
1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
fuse-apicurito-generator-rhel8   172.30.1.1:5000/openshift/fuse7/fuse-apicurito-generator-
```

```
rhel8   1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13    About an hour ago
apicurito-ui-rhel8              172.30.1.1:5000/openshift/fuse7/apicurito-ui-rhel8
1.6,1.7,1.8,1.9,1.10,1.11,1.12,1.13   About an hour ago
```

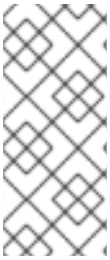
2. 现在，您可以一次更新每个镜像流：

```
oc import-image -n openshift fuse7/fuse7-java-openshift-rhel8:1.13
oc import-image -n openshift fuse7/fuse7-java-openshift-jdk11-rhel8:1.13
oc import-image -n openshift fuse7/fuse7-karaf-openshift-rhel8:1.13
oc import-image -n openshift fuse7/fuse7-eap-openshift-jdk8-rhel7:1.13
oc import-image -n openshift fuse7/fuse7-eap-openshift--jdk11-rhel8:1.13
oc import-image -n openshift fuse7/fuse7-console-rhel8:1.13
oc import-image -n openshift fuse7/apicurito-ui-rhel8:1.13
oc import-image -n openshift fuse7/fuse-apicurito-generator-rhel8:1.13
```



注意

镜像流中的版本标签格式为 **1.9-<BUILDNUMBER& gt;**。当您将标签指定为 **1.9** 时，您将在 **1.9** 流中获取最新的构建。



注意

您还可以配置 **Fuse** 应用程序，以便在 **OpenShift** 镜像上的新 **Fuse** 可用时自动触发重新构建。详情请参阅 **Builds OpenShift Container Platform documentation_** 中的 **Triggering and modify build** 部分。

15.3. 在 OPENSHIFT 模板上修补 FUSE

您必须将 **OpenShift** 模板上的 **Fuse** 更新至最新的补丁级别，以确保使用正确的补丁依赖项构建基于模板的新项目。

流程

1. 您需要管理员特权才能在 **OpenShift** 模板上更新 **Fuse**。以管理员身份登录到 **OpenShift** 服务器，如下所示：

```
oc login URL -u ADMIN_USER -p ADMIN_PASS
```

其中 **URL** 是 **OpenShift** 服务器和 **ADMIN_USER** 的 **URL**，**ADMIN_PASS** 是 **OpenShift** 服务器上管理员帐户的凭据。

2. *在 OpenShift 模板上安装补丁的 Fuse。在命令提示符下输入以下命令：*

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-amq-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-cdi-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-cxf-jaxrs-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-cxf-jaxws-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-camel-amq-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-camel-log-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-camel-rest-sql-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-cxf-rest-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-amq-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-config-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-drools-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-infinispan-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-xml-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-cxf-jaxrs-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-cxf-jaxws-template.json
```

注意

BASEURL 指向 Git 存储库的 GA 分支，该分支存储 Quickstart 模板，并且始终具有 HEAD 的最新模板。因此，每次运行前面的命令时，都将获得模板的最新版本。

15.4. 使用 BOM 对应用程序的依赖项进行补丁

如果您的应用程序 pom.xml 文件被配置为使用新样式的 BOM，请按照本节中的说明升级 Maven 依赖项。

15.4.1. 更新 Spring Boot 应用程序中的依赖项

以下代码片段显示了 OpenShift 上 Fuse 中 Spring Boot 应用程序的 POM 文件标准布局，突出显示一些重要属性设置：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <fuse.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>
    ...
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-springboot-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
  <build>
    ...
    <plugins>
      <!-- Core plugins -->
      ...
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        ...
        <version>${fuse.version}</version>
      </plugin>
    </plugins>
  </build>

  <profiles>
    <profile>
      <id>openshift</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.jboss.redhat-fuse</groupId>
            <artifactId>openshift-maven-plugin</artifactId>
            ...
            <version>${fuse.version}</version>
          </plugin>
        </plugins>
      </build>
```

```
    </profile>
  </profiles>
</project>
```

当涉及修补或升级应用程序时，以下版本设置非常重要：

fuse.version

定义 new-style fuse-springboot-bom BOM 的版本，以及 openshift-maven-plugin 插件的版本和 spring-boot-maven-plugin 插件。

15.4.2. 更新 Karaf 应用中的依赖项

以下代码片段显示了 OpenShift 上 Fuse 中 Karaf 应用程序的 POM 文件标准布局，突出显示一些重要属性设置：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <fuse.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>
    ...
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-karaf-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>karaf-maven-plugin</artifactId>
        <version>${fuse.version}</version>
        ...
      </plugin>
      ...
      <plugin>
```

```
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>openshift-maven-plugin</artifactId>
        <version>${fuse.version}</version>
        ...
      </plugin>
    </plugins>
  </build>

</project>
```

当涉及修补或升级应用程序时，以下版本设置非常重要：

fuse.version

定义 new-style fuse-karaf-bom BOM 的版本，以及 openshift-maven-plugin 插件的版本和 karaf-maven-plugin 插件。

15.4.3. 更新 JBoss EAP 应用中的依赖项

以下代码片段显示了 OpenShift 上 Fuse 中 JBoss EAP 应用程序的 POM 文件标准布局，突出显示一些重要属性设置：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <fuse.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>
    ...
  </properties>

  <!-- Dependency Management -->
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-eap-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

当涉及修补或升级应用程序时，以下版本设置非常重要：

fuse.version

*定义 **fuse-eap-bom** BOM 文件的版本（替换 **old-style wildfly-camel-bom** BOM 文件）。通过将 **BOM** 版本更新至特定的补丁版本，您也会有效地更新 **JBoss EAP Maven** 依赖项上的所有 **Fuse**。*

15.5. 可用的 BOM 版本

*下表显示了与 **Red Hat Fuse** 的不同补丁版本对应的新风格的 **BOM** 版本。*

表 15.1. Red Hat Fuse Releases 和 Corresponding New-Style BOM 版本

Red Hat Fuse 发行版本	**org.jboss.redhat-fuse** BOM Version
Red Hat Fuse 7.13 GA	7.13.0.fuse-7_13_0-00012-redhat-00001

*要将应用程序 **POM** 升级到特定的 **Red Hat Fuse** 补丁版本，请将 **fuse.version** 属性设置为对应的 **BOM** 版本。*

第 16 章 在 OPENSHIFT 上卸载 FUSE

要在 OpenShift 上卸载 Fuse，请使用 oc delete 命令从 registry.redhat.io 中删除镜像流和模板。

16.1. 在 OPENSHIFT 4.X 服务器上卸载 FUSE 镜像流和模板

流程

+ 查找您的版本的 BASEURL，并将其定义为在以下命令中使用的变量。

+

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-7_13_0-00014-redhat-00001
```

1. 删除 Spring Boot 2 quickstart 模板。

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
 spring-boot-2-cxf-jaxrs-template.json \
 spring-boot-2-cxf-jaxws-template.json \
 spring-boot-2-cxf-jaxrs-xml-template.json \
 spring-boot-2-cxf-jaxws-xml-template.json ;
 do oc delete -n openshift -f \
 ${BASEURL}/quickstarts/${template}
 done
```

2. 删除 OpenShift 快速启动模板上的 Fuse。

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json ;
```

```
do
oc delete -n openshift -f \
${BASEURL}/quickstarts/${template}
done
```

3. **删除镜像流。**

```
oc delete -n openshift -f ${BASEURL}/fis-image-streams.json
```

4. **删除 Samples Operator 中的项目。**

编辑 Samples Operator 的配置：

```
oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

5. **从 skippedImagestreams 和 skippedTemplates 部分删除 Fuse 和 Spring Boot 2 模板。**

内置镜像流

对于常见的用例，一些镜像流和模板内置了。它们由 **Sample Operator** 管理，因此您无法手动删除它们。您可以在卸载时忽略它们。

内置镜像流使用清单中的 **samples.operator.openshift.io/managed: "true"** 标签进行配置，以便您可以验证它是否使用 **oc get** 和 **grep** 命令进行管理。

Example

```
]$ oc get is fuse7-eap-openshift -n openshift -o yaml | grep
'samples.operator.openshift.io/managed'
    samples.operator.openshift.io/managed: "true"
]$
```

附录 A. SPRING BOOT MAVEN 插件

Spring Boot Maven 插件在 Maven 中提供 Spring Boot 支持，并允许您打包可执行文件 jar 或 war 归档并运行应用程序 原位。

A.1. SPRING BOOT MAVEN 插件目标

Spring Boot Maven 插件包括以下目标：

- spring-boot:run 运行 Spring Boot 应用程序。
- spring-boot:repackage 将您的 .jar 和 .war 文件重新打包为可执行文件。
- spring-boot:start 和 spring-boot:stop 均用于管理 Spring Boot 应用程序的生命周期。
- spring-boot:build-info 生成可由 Actuator 使用的构建信息。

A.2. 使用 SPRING BOOT MAVEN 插件

您可以在以下位置找到有关如何使用 Spring Boot 插件的通用说明：https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/#using。以下示例演示了为 Spring Boot 使用 spring-boot-maven-plugin。

- Spring Boot 2 示例

有关 Spring Boot Maven 插件的更多信息，请参阅 https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/ 链接。

A.2.1. 为 Spring Boot 2 使用 Spring Boot Maven 插件

以下示例演示了为 Spring Boot 2 使用 spring-boot-maven-plugin。

Example

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.redhat.fuse</groupId>
  <artifactId>spring-boot-camel</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <!-- configure the Fuse version you want to use here -->
    <fuse.bom.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.bom.version>

    <!-- maven plugin versions -->
    <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
    <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
  </properties>


  <build>
    <defaultGoal>spring-boot:run</defaultGoal>

    <plugins>
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>${fuse.bom.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>repackage</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>

  <repositories>
    <repository>
      <id>redhat-ga-repository</id>
      <url>https://maven.repository.redhat.com/ga</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
    <repository>
      <id>redhat-ea-repository</id>
      <url>https://maven.repository.redhat.com/earlyaccess/all</url>
```

```
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>

    <pluginRepositories>
        <pluginRepository>
            <id>redhat-ga-repository</id>
            <url>https://maven.repository.redhat.com/ga</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>redhat-ea-repository</id>
            <url>https://maven.repository.redhat.com/earlyaccess/all</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
</project>
```

附录 B. 使用 KARAF MAVEN 插件

karaf-maven-plugin 可让您创建 Karaf 服务器汇编，它是 Karaf 容器的微服务风格打包。完成的汇编包含 Karaf 安装的所有基本组件（包括 etc/、data/、lib 和系统目录的内容），但会缩减到运行应用程序所需的最小裸机。

B.1. MAVEN 依赖项

karaf-assembly 项目中的 Maven 依赖项是功能存储库（分类 功能）或 kar 归档。

- 功能存储库安装在 maven 结构化系统/内部存储库中。
- Kar 归档在服务器之上解包其内容，并安装了包含的功能存储库。

Maven 依赖项范围

依赖项的 Maven 范围决定了其功能存储库是否列在 features 服务配置文件 etc/org.apache.karaf.features.cfg （在 featuresRepositories 属性下）。这些范围是：

- compile （默认）：仓库中的所有功能（或 kar 归档）将安装到 startup.properties 中。功能存储库不在功能服务配置文件中列出。
- runtime: 作为 karaf-maven-plugin 中的引导阶段。
- provided: 作为 karaf-maven-plugin 中的安装阶段。

B.2. KARAF MAVEN 插件配置

karaf-maven-plugin 定义与 Maven 范围相关的三个阶段。插件配置通过引用安装的功能软件仓库来控制如何使用这些元素安装功能：

- 启动阶段： etc/startup.properties

  在这个阶段，启动功能、启动配置集和启动捆绑包用于准备包含在 etc/startup.properties 中的捆绑包列表。这将导致功能捆绑包列在适当起始级别的 etc/startup.properties 中，并将捆绑

包复制到 系统 内部存储库中。您可以使用 feature_name 或 feature_name/feature_version 格式，例如 < startupFeature>foo</startupFeature& gt;。

- Boot stage: etc/org.apache.karaf.features.cfg

  此阶段管理 featuresRepositories 属性中的 featuresBoot 属性和软件仓库中可用的功能。这将导致功能名称添加到 features 服务配置文件中 boot-features，以及复制到 系统 内部存储库中的功能中的所有捆绑包。您可以使用 feature_name 或 feature_name/feature_version 格式，例如 < bootFeature>bar</bootFeature& gt;。

- 安装阶段：

  此阶段将工件安装到 ${karaf.home}/${karaf.default.repository} 中。这将导致功能中的所有捆绑包安装到 系统 内部存储库中。因此在运行时，可以安装该功能而无需访问外部存储库。您可以使用 feature_name 或 feature_name/feature_version 格式，例如 < installedFeature>baz</installedFeature& gt;。

- 库

  该插件接受 library 元素，该元素可以具有一个或多个指定库 URL 的 library 子元素。

  Example

```
<libraries>
   <library>mvn:org.postgresql/postgresql/9.3-1102-jdbc41;type:=endorsed</library>
</libraries>
```

B.3. 自定义的 KARAF ASSEMBLY

创建 Karaf 服务器汇编的建议方法是使用 karaf- maven-plugin 提供的 karaf:assembly 目标。这会从项目的 pom.xml 文件中的 Maven 依赖项组装服务器。karaf-maven-plugin 配置中指定的捆绑包（或功能）和 pom.xml 中的 &lt ;dependencies > 部分指定的依赖项都可以进入自定义的 karaf assembly。

- 对于 kar

kar 类型的依赖项将作为启动(scope=compile)、boot (scope=runtime)或在 karaf-maven-plugin 中添加(scope=provided) kars kars。kars 被解压缩到工作目录（目标/装配）和功能 XML，并搜索功能 XML，并用作附加功能存储库（与给定 kar 阶段相等的阶段）。

- 对于 features.xml

具有 功能 分类器的依赖项将用作启动(scope=compile)、boot (scope=runtime)或在 karaf-maven-plugin 中安装(scope=provided)存储库。不需要显式添加在 kar 中找到的功能存储库。

- 对于 jar 和 bundle

与 bundle 或 jar 类型的依赖项将用作启动(scope=compile)、boot (scope=runtime)或在 karaf-maven-plugin 中安装(scope=provided)捆绑包。

B.3.1. Karaf:assembly 目标

您可以使用 karaf-maven-plugin 提供的 karaf:assembly 目标创建 Karaf 服务器装配。此目标从项目 POM 中的 Maven 依赖项组装一个微服务风格的服务器组。对于 OpenShift 项目中的 Fuse，我们建议您将 karaf:assembly 目标绑定到 Maven 安装阶段。该项目使用捆绑包打包，项目本身会在 bootBundles 元素中列出它，从而安装到 Karaf 容器中。

注意

在启动阶段仅包含 karaf 框架功能等必要元素，因为它将进入 etc/startup.properties，在此阶段 karaf 功能服务不会完全启动。将其他元素延迟到启动阶段。

Example

以下示例显示了快速启动中典型的 Maven 配置：

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>karaf-maven-plugin</artifactId>
  <version>${fuse.version}</version>
  <extensions>true</extensions>
  <executions>
    <execution>
      <id>karaf-assembly</id>
      <goals>
        <goal>assembly</goal>
```

```
      </goals>
      <phase>install</phase>
    </execution>
  </executions>
  <configuration>

    <karafVersion>{karafMavenPluginVersion}</karafVersion>
    <useReferenceUrls>true</useReferenceUrls>
    <archiveTarGz>false</archiveTarGz>
    <includeBuildOutputDirectory>false</includeBuildOutputDirectory>
    <startupFeatures>
      <feature>karaf-framework</feature>
          </startupFeatures>
          <bootFeatures>
      <feature>shell</feature>
      <feature>jaas</feature>
      <feature>aries-blueprint</feature>
      <feature>camel-blueprint</feature>
      <feature>fabric8-karaf-blueprint</feature>
      <feature>fabric8-karaf-checks</feature>
    </bootFeatures>
    <bootBundles>
      <bundle>mvn:${project.groupId}/${project.artifactId}/${project.version}</bundle>
    </bootBundles>
  </configuration>
</plugin>
```

附录 C. OPENSHIFT MAVEN 插件

OpenShift Maven 插件用于为 OpenShift 构建和部署 Java 应用。它将 Java 应用程序引入 OpenShift。它提供了一个紧密集成到 maven，并从已提供的构建配置中获益。它侧重于三个任务：

- 构建 S2I 镜像
- 创建 OpenShift 资源
- 在 OpenShift 上部署应用程序

C.1. 关于 OPENSHIFT MAVEN 插件

OpenShift Maven 插件具有以下功能：

- 处理 S2I 镜像，因此继承其灵活且强大的配置。
- 支持这两个 OpenShift 描述符
- 带有二进制源的 OpenShift Docker 构建（作为直接镜像构建针对 Docker 守护进程的替代方案）
- mulitple 配置风格：
  - 零配置用于快速提升默认值，将预先选择。
  - 以 XML 语法在插件配置内进行内联配置。
  - 插件增强的实际部署描述符的外部配置模板。

- 灵活的自定义：
  - 生成器分析 Maven 构建并为某些系统(spring-boot、普通 java、karaf)生成的自动 Docker 镜像配置。
  - 增强程序通过额外的信息（如 SCM 标签）来扩展 OpenShift 资源描述符，并可添加诸如服务的默认对象。
  - 生成器和 Enrichers 可以单独配置并合并到配置集中。

C.2. 构建镜像

oc:build 目标用于创建包含应用程序的 Docker 格式镜像。然后，稍后可以在 Kubernetes 或 OpenShift 上部署它们。此插件使用 maven-assembly-plugin 中的 assembly 描述符格式来指定添加到镜像中的内容。然后，可以通过 oc:push 推送到公共或私有 Docker registry。oc:watch 目标允许您对代码更改做出反应，以自动重新创建镜像或将新工件复制到正在运行的容器中。

C.3. KUBERNETES 和 OPENSHIFT 资源

可以使用 oc:resource 创建 Kubernetes 和 OpenShift 资源描述符。这些文件打包在 Maven 工件中，并可使用 oc:apply 部署到正在运行的编配平台中。

配置

配置有四个级别：

- zero-Config 模式有助于根据 pom.xml 文件中的内容（如使用哪个基础镜像或要公开的端口）做出一些非常有用的决策。它用于启动事情，并使快速入门应用程序保持小且 tidy。
- XML 插件配置模式与 docker-maven-plugin 提供的内容类似。它允许使用 IDE 支持类型安全配置，但只提供了可能的资源描述符功能的子集。
- Kubernetes 和 OpenShift 资源片段是用户提供的 YAML 文件，可由插件增强。这允许专家用户使用具有所有功能的普通配置文件，但也能够添加特定于项目的构建信息并避免样板代码。
-

Docker Compose 用于在 OpenShift 集群上启动 docker compose 部署。这至少需要了解 OpenShift 部署过程。

C.4. 安装 OPENSHIFT MAVEN 插件

此插件位于 Maven central 存储库下，并可连接到预集成阶段，如下所示。默认情况下，Maven 只搜索 org.apache.maven.plugins 和 org.codehaus.mojo 软件包中的插件。要解析 JKube 插件目标的供应商，请编辑 ~/.m2/settings.xml 文件并添加 org.eclipse.jkube 命名空间，以便 <pluginGroups> 配置。

流程

- 要将 OpenShift Maven 插件连接到 pre-integration 阶段，请将以下内容添加到 ~/.m2/settings.xml 文件中：

```
<settings>
    ...

    <pluginGroups>
      <pluginGroup>org.jboss.redhat-fuse</pluginGroup>
    </pluginGroups>

    ...
</settings>

<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>openshift-maven-plugin</artifactId>
  <version>${fuse.version}</version>

  <configuration>
    ....
    <images>
      <!-- A single's image configuration -->
      <image>
        ...
        <build>
         ....
        </build>
      </image>
      ....
    </images>
  </configuration>

  <!-- Connect oc:resource, oc:build and oc:helm to lifecycle phases -->
  <executions>
    <execution>
      <id>jkube</id>
      <goals>
        <goal>resource</goal>
```

```
        <goal>build</goal>
        <goal>helm</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

C.5. 了解 OPENSHIFT MAVEN 插件构建目标

构建目标用于创建和管理 Kubernetes 和 OpenShift 构建工件，如 Docker 格式的镜像或 S2I 构建。

表 C.1. OpenShift Maven 插件构建目标

目标	描述
oc:resource	创建 Kubernetes 或 OpenShift 资源描述符。生成的资源位于 target/classes/META-INF/jkube/openshift 目录中。
oc:build	构建镜像。
oc:push	将镜像推送到 registry。要推送的 registry 默认是 docker.io，但可以指定为镜像名称的一部分。
oc:apply	将资源应用到正在运行的集群。此目标与 oc:deploy 类似，但不执行完整的部署周期。

C.6. 了解 OPENSHIFT MAVEN 插件开发目标

开发目标用于向开发集群部署资源描述符。另外，帮助您管理开发集群的生命周期。

表 C.2. OpenShift Maven 插件开发目标

目标	描述
oc:deploy	在创建并构建应用程序后，将资源描述符部署到集群中。与 oc:apply 相同，但它在后台运行。
oc:undeploy	取消部署并移除集群中的资源描述符。
oc:log	显示正在运行的应用的日志。
oc:debug	启用远程调试。
oc:watch	观察文件更改并执行重建和重新部署。

附录 D. CAMEL MAVEN 插件

您可以使用 camel-maven 插件从源代码验证所有 Camel 端点。这可让您在运行 Camel 应用程序或单元测试前确保端点有效。

D.1. CAMEL MAVEN 插件目标

在源代码中验证 Camel 端点使用

- camel:validate ：此目标验证 Maven 项目源代码以识别无效的 camel 端点 URI。

D.2. 将 CAMEL-MAVEN 插件添加到项目中

您可以将 camel-maven 插件添加到项目的 pom.xml 文件中。

流程

1. 要启用插件，请将以下内容添加到 pom.xml 文件中：

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>camel-maven-plugin</artifactId>
  <version>${fuse.bom.version}</version>
</plugin>
```

2. 从命令行或 Java 编辑器运行验证目标。

```
mvn camel:validate
```

自动运行插件

您可以启用插件作为构建的一部分自动运行，以捕获任何错误。阶段 决定插件何时运行。

在以下示例中，插件在阶段 process-classes 中运行，后者在编译主源代码后运行。

Example

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>camel-maven-plugin</artifactId>
  <version>7.13.0.fuse-7_13_0-00012-redhat-00001</version>
  <executions>
    <execution>
      <phase>process-classes</phase>
        <goals>
          <goal>validate</goal>
        </goals>
    </execution>
  </executions>
</plugin>
```

验证测试源代码

您可以通过将阶段更改为 process-test-classes，配置 maven 插件以验证测试源代码：

Example

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>camel-maven-plugin</artifactId>
  <version>7.13.0.fuse-7_13_0-00012-redhat-00001</version>
  <executions>
    <execution>
      <configuration>
        <includeTest>true</includeTest>
      </configuration>
        <phase>process-test-classes</phase>
          <goals>
            <goal>validate</goal>
          </goals>
    </execution>
  </executions>
</plugin>
```

D.3. 在任何 MAVEN 项目中运行目标

您还可以在任何 Maven 项目中运行验证目标，而无需将插件添加到 pom.xml 文件中。您需要使用其完全限定名称来指定插件。

流程

- 要在 Apache Camel 的 camel-example-cdi 插件上运行目标，请运行以下命令：

```
$cd camel-example-cdi
$mvn org.jboss.redhat-fuse:camel-maven-plugin:7.13.0.fuse-7_13_0-00012-redhat-00001:validate
```

这将显示以下输出：

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Camel :: Example :: CDI 2.16.2
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- fabric8-camel-maven-plugin:2.3.80:validate (default-cli) @ camel-example-cdi ---
[INFO] Endpoint validation success: (4 = passed, 0 = invalid, 0 = incapable, 0 = unknown components)
[INFO] Simple validation success: (0 = passed, 0 = invalid)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

成功传递验证后，您可以验证四个端点。以下示例演示了如何验证和在需要时正确验证 camel 端点。

Example

让我们假定您在源代码中的 Camel 端点 URI 中做了拼写错误，例如：

1. 正确的 Camel 端点 URI 如下所示：

```
@Uri("timer:foo?period=5000")
```

2. 您可以进行更改以在 句点 选项中包含拼写错误，例如：

```
@Uri("timer:foo?perid=5000")
```

3. *再次运行验证目标。*

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Camel :: Example :: CDI 2.16.2
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- org.jboss.redhat-fuse:camel-maven-plugin:7.13.0.fuse-7_13_0-00012-redhat-
00001:validate (default-cli) @ camel-example-cdi ---
[WARNING] Endpoint validation error at:
org.apache.camel.example.cdi.MyRoutes(MyRoutes.java:32)

 timer:foo?perid=5000

                perid    Unknown option. Did you mean: [period]


[WARNING] Endpoint validation error: (3 = passed, 1 = invalid, 0 = incapable, 0 = unknown
components)
[INFO] Simple validation success: (0 = passed, 0 = invalid)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

如上所示，camel 端点 URI 中的错误会显示。

D.4. 选项

maven 插件支持以下选项，您可以从命令行（使用 -D 语法）配置，或者在 < configuration > 标签的 pom.xml 文件中定义。

表 D.1. Camel Maven 插件选项

参数	默认	描述	注释
downloadVersion	true	允许从互联网下载 Camel 目录版本。	如果项目使用了与插件 default 不同的 Camel 版本。
failOnError	false	如果找到无效的 Camel 端点，则失败。	默认情况下，插件会在 **WARN** 级别记录错误。
logUnparseable	false	无法解析的日志端点 URI，因此无法验证。	
includeJava	true	在 Camel 端点验证中包含 Java 文件。	另请参见 **包括** 和 **excludes**。

参数	默认	描述	注释
includeXML	true	在 Camel 端点验证中包含 XML 文件。	另请参见 **包括** 和 **excludes**。
includeTest	false	在验证中包含测试源代码。	
includes	-	通配符和正则表达式模式，用于过滤要包含在验证中的 java 和 xml 文件名。	可以使用逗号分隔多个值。
excludes	-	通配符和正则表达式模式，来过滤 java 和 xml 文件，以便从验证中排除。	可以使用逗号分隔多个值。
ignoreUnknownComponent	true	忽略未知组件。	
ignoreIncapable	true	忽略能够解析端点 URI。	
ignoreLenientProperties	true	忽略使用 lenient 属性的组件，以便它们可以有自定义参数。	默认情况下，URI 验证在不属于组件的属性上失败。（例如，用于在端点 URI 中提供查询参数的 HTTP 组件。）
showAll	false	显示所有端点和简单表达式。	显示无效和有效性。

D.5. 验证包括 TEST

如果您有 Maven 项目，则也可以运行插件来验证单元测试源代码中的端点。

流程

- ***使用 -D 传递选项：***

```
$cd myproject

$mvn org.jboss.redhat-fuse:camel-maven-plugin:7.13.0.fuse-7_13_0-00012-redhat-00001:validate -DincludeTest=true
```

附录 E. 自定义 JVM 环境变量

您可以使用 JVM 环境变量在 OpenShift 镜像上设置 Fuse 的所有选项。

E.1. 在 OPENJDK 8 中使用 S2I JAVA 构建器镜像

使用 S2I Java 构建器镜像，您可以直接运行结果，而无需使用任何其他应用服务器。此 S2I 镜像适合具有扁平类路径（包括 fat jars）的微服务。

您可以在 OpenShift 镜像上使用 Fuse 时配置 Java 选项。对于 JVM 选项，您可以使用环境变量 JAVA_OPTIONS。另外，为提供给应用程序的参数提供 JAVA_ARGS。

E.2. 使用带有 OPENJDK 8 的 S2I KARAF 构建器镜像

S2I Karaf 构建器镜像可与 OpenShift 的源到镜像工作流一起使用，以构建基于 maven 项目的自定义组件。

流程

- 使用以下命令来使用 S2I 工作流。

```
s2i build <git repo url> registry.redhat.io/fuse7/fuse-karaf-openshift:1.13 <target image name>
docker run <target image name>
```

E.2.1. 配置 Karaf4 汇编

maven 项目构建的 Karaf4 装配的位置可以通过多种方式提供。

- 输出目录中的默认 assembly 文件 lftar.gz
- 在 sti 或 oc 命令中使用 -e 标志
- 通过在项目源下的 .sti/environment 中设置 FUSE_ASSEMBLY 属性

E.2.2. 自定义 Maven 构建

可以自定义 maven 构建。MAVEN_ARGS 环境变量可以设置为更改行为。默认情况下，MAVEN_ARGS 设置如下：

```
`Karaf4: install karaf:assembly karaf:archive -DskipTests -e`
```

E.3. 构建时间环境变量

以下是用于在构建期间影响 S2I Java 和 Karaf 构建器镜像的环境变量。

- MAVEN_ARGS ：调用 maven 时使用的参数，替换默认软件包。
- MAVEN_ARGS_APPEND: Additional Maven 参数，可用于添加临时参数，如 -X 或 -am -pl。
- ARTIFACT_DIR ：目标/的路径，其中为多模块构建创建 jar 文件。它们被添加到 ${MAVEN_ARGS} 中。
- ARTIFACT_COPY_ARGS ：在将输出目录中的工件复制到应用程序目录时使用的参数。用于指定哪些工件将成为镜像的一部分。
- MAVEN_CLEAR_REPO: 如果设置，请在构建工件后删除 Maven 存储库。这可用于将应用镜像保持小，但可防止增量构建。默认值为 false。

E.4. 运行时间环境变量

您可以使用以下环境变量来影响 run 脚本。

- JAVA_APP_DIR ：应用程序所在的目录。应用程序中的所有路径都相对于目录。
- JAVA_LIB_DIR ：此目录包含 Java jar 文件以及可选的 classpath 文件，其中包含 classpath。作为单行类路径(colon separated)或列出行尾的 jar 文件。但是，如果没有设置，则 JAVA_LIB_DIR 与 JAVA_APP_DIR 目录相同。

- *JAVA_OPTIONS ：在调用 java 时添加的选项。*
- *JAVA_MAX_MEM_RATIO ：当 JAVA_OPTIONS 中不提供 -Xmx 选项时，会使用它。这用于根据容器限制计算默认的 maximal heap Memory。如果在没有容器任何内存约束的 Docker 容器中使用，则此选项不会起作用。*
- *JAVA_MAX_CORE ：它手动限制可用的内核数，用于计算特定默认值，如垃圾收集器线程的数量。如果设置为 0，则无法根据内核数执行基本 JVM 调优。*
- *JAVA_DIAGNOSTICS ：将其设置为在出现问题时获取一些诊断信息。*
- *JAVA_MAIN_CLASS ：用作 java 的参数的主要类。当您使用此环境变量时，$JAVA_APP_DIR 目录中的所有 jar 文件都添加到 classpath 中，在 $JAVA_LIB_DIR 目录中。*
- *JAVA_APP_JAR ：包含适当清单的 jar 文件，以便您可以使用 java -jar 开始。但是，如果没有提供，则会设置 $JAVA_MAIN_CLASS。在所有情况下，此 jar 文件都添加到 classpath 中。*
- *JAVA_APP_NAME ：用于进程的名称。*
- *JAVA_CLASSPATH ：要使用的类路径。如果没有提供，启动脚本会检查文件 ${JAVA_APP_DIR}/classpath，并将其内容用作 classpath。如果此文件不存在，则应用程序目录中的所有 jar 都添加到 (classes:${JAVA_APP_DIR}/*) 下。*
- *JAVA_DEBUG ：如果设置，则将开启远程调试。*
- *JAVA_DEBUG_PORT ：用于远程调试的端口。默认值为 5005。*

E.5. JOLOKIA 配置

您可以使用 Jolokia 中的以下环境变量：

-

AB_JOLOKIA_OFF ：如果设置，则禁用 Jolokia 激活(echos 空值)。默认情况下启用 Jolokia。

- *AB_JOLOKIA_CONFIG ：如果设置，请使用文件（包括路径）作为 Jolokia JVM 代理属性。但是，如果没有设置，将使用设置创建 /opt/jolokia/etc/jolokia.properties。*
- *AB_JOLOKIA_HOST: Host address to bind （默认值为 0.0.0.0）*
- *AB_JOLOKIA_PORT: 使用的端口（默认值为 8778）*
- *AB_JOLOKIA_USER: 用于基本身份验证的用户。默认情况下，它是 jolokia。*
- *AB_JOLOKIA_PASSWORD: 用于基本身份验证的密码。默认情况下关闭身份验证。*
- *AB_JOLOKIA_PASSWORD_RANDOM ：生成值，并使用 /opt/jolokia/etc/jolokia.pw 文件中写入。*
- *AB_JOLOKIA_HTTPS ：与 HTTPS 进行安全通信切换。默认情况下，如果在 AB_JOLOKIA_OPTS 中未提供 serverCert 配置，则会生成自签名服务器证书。*
- *AB_JOLOKIA_ID:要使用的代理 ID*
- *AB_JOLOKIA_DISCOVERY_ENABLED ：启用 Jolokia 发现。默认值为 false。*
- *AB_JOLOKIA_OPTS: 附加到代理配置中的附加选项。选项以 key=value 格式提供。*

以下是与各种环境集成的一个选项：

- *AB_JOLOKIA_AUTH_OPENSHIFT ：切换 OpenShift TSL 通信的客户端身份验证。确保此参数的值必须存在于客户端证书中。如果启用此参数，它将自动将 Jolokia 切换到 HTTPS 通信模式。默认 CA 证书设置为 /var/run/secrets/kubernetes.io/serviceaccount/ca.crt。*

可以通过将变量 JAVA_ARGS 设置为对应的值来提供应用程序参数。

附录 F. 调整 JVM 以在 LINUX 容器中运行

当您允许 JVM ergonomics 为垃圾收集器、堆大小和运行时编译器设置默认值时，Linux 容器中运行的 Java 进程的行为与预期一样。当您在不执行任何调优参数的 Java 应用程序时，java -jar mypplication-fat.jar urllib-osgithe JVM 会自动根据主机限制设置几个参数，而不是 容器限制。

本节提供有关在 Linux 容器中打包 Java 应用程序的信息，以便考虑计算默认值时容器的限制。

F.1. 调整 JVM

当前的 Java JVM 的生成不是容器感知型，因此它们根据物理主机的大小而非容器的大小来分配资源。例如，JVM 通常将主机上的物理内存的 最大堆大小 设置为 1/4。在大型主机上，此值可以轻松地超过为容器定义的内存限值，如果容器限制在运行时超过容器，OpenShift 将终止应用。

要解决这个问题，您可以在 OpenShift 基础镜像上使用 Fuse，它们能够了解在受限容器中运行的 Java JVM，并在未手动完成的情况下自动调整最大堆大小。它提供了为运行应用程序的 JVM 设置最大内存限值和核心限制的解决方案。对于 OpenShift 镜像上的 Fuse，它可以：

- 根据容器内核设置 CICompilerCount
- 当容器内存限制低于 300MB 时禁用 C2 JIT 编译器
- 以下 300MB 时，对默认堆大小使用容器内存限值的一体

F.2. FUSE ON OPENSHIFT 镜像的默认行为

在 OpenShift 上的 Fuse 中，应用构建的基础镜像可以是 Java 镜像（用于 Spring Boot 应用程序）或 Karaf 镜像（用于 Karaf 应用程序）。在 OpenShift 镜像上的 Fuse 执行一个脚本，该脚本读取容器限制，并使用这些限制作为分配资源的基础。默认情况下，脚本将以下资源分配给 JVM：

- 容器内存限值的 50%
- 容器核心限制的 50%

这有一些例外。对于 Karaf 和 Java 镜像，当物理内存低于 300MB 阈值时，堆大小将恢复为默认堆大小的一次性，而不是一对一对一的。

F.3. 在 OPENSHIFT 镜像上自定义 FUSE 调整

该脚本设置 CONTAINER_MAX_MEMORY 和 CONTAINER_CORE_LIMIT 环境变量，由自定义应用读取以调优其内部资源。另外，您可以指定以下运行时环境变量，供您自定义运行应用程序的 JVM 上的设置：

- JAVA_OPTIONS
- JAVA_MAX_MEM_RATIO

要显式自定义限制，您可以在 Maven 项目中编辑 deployment.yml 文件来设置 JAVA_MAX_MEM_RATIO 环境变量。

Example

```
spec:
  template:
    spec:
      containers:
        -
          resources:
            requests:
              cpu: "0.2"
               memory: 256Mi
            limits:
              cpu: "1.0"
               memory: 256Mi
          env:
          - name: JAVA_MAX_MEM_RATIO
            value: 60
```

F.4. 调整第三方库

红帽建议您为 Jetty 等任何第三方 Java 库自定义限制。如果您没有手动自定义限制，这些库将使用给定的默认限制。启动脚本会公开一些环境变量来描述应用程序可使用的容器限制：

CONTAINER_CORE_LIMIT

计算的内核限制

CONTAINER_MAX_MEMORY

提供给容器的内存限值