Red Hat build of Eclipse Vert.x 4.3

Eclipse Vert.x Runtime Guide

使用 Eclipse Vert.x 来开发在 OpenShift 和独立 RHEL 上运行的被动、非阻塞、异步应用程序

Last Updated: 2023-04-07

Red Hat build of Eclipse Vert.x 4.3 Eclipse Vert.x Runtime Guide

使用 Eclipse Vert.x 来开发在 OpenShift 和独立 RHEL 上运行的被动、非阻塞、异步应用程序

法律通告

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本指南详细介绍了使用 Eclipse Vert.x 运行时。

目录

前言 3

对红帽文档提供反馈 4

使开源包含更多 5

第 1 章 使用 ECLIPSE VERT.X 进行应用程序开发简介 6
- 1.1. 使用 RED HAT RUNTIMES 应用程序开发概述 6
- 1.2. ECLIPSE VERT.X 概述 6

第 2 章 配置应用程序 9
- 2.1. 将应用程序配置为使用 ECLIPSE VERT.X 9

第 3 章 开发和部署 ECLIPSE VERT.X 运行时应用程序 11
- 3.1. 开发 ECLIPSE VERT.X 应用程序 11
- 3.2. 将 ECLIPSE VERT.X 应用部署到 OPENSHIFT 14
- 3.3. 将 ECLIPSE VERT.X 应用程序部署为独立 RED HAT ENTERPRISE LINUX 17

第 4 章 调试基于 ECLIPSE VERT.X 的应用 20
- 4.1. 远程调试 20
- 4.2. 调试日志记录 22

第 5 章 监控应用程序 26
- 5.1. 在 OPENSHIFT 中访问应用的 JVM 指标 26
- 5.2. 使用 ECLIPSE VERT.X 使用 PROMETHEUS 公开应用程序指标 27

附录 A. SOURCE-TO-IMAGE (S2I)构建过程 30

附录 B. 更新示例应用程序的部署配置 31

附录 C. 配置 JENKINS 自由风格的项目，以使用 OPENSHIFT MAVEN 插件部署应用程序 33
- 后续步骤 34

附录 D. 额外的 ECLIPSE VERT.X 资源 35

附录 E. 应用程序开发资源 36

前言

本指南涵盖了概念以及开发人员使用 Eclipse Vert.x 运行时所需要的实际详细信息。

对红帽文档提供反馈

我们感谢您对我们文档的反馈。要提供反馈，您可以突出显示文档中的文本并添加注释。

本节介绍如何提交反馈。

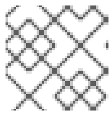
前提条件

- 登录到红帽客户门户网站。
- 在红帽客户门户网站中，以 多页 HTML 格式查看文档。

流程

要提供反馈，请执行以下步骤：

1. 点击文档右上角的 **反馈** 按钮查看现有反馈。



注意

反馈功能仅在多页 HTML 格式中启用。

2. 高亮标记您要提供反馈的文档中的部分。
3. 点击在高亮文本旁边弹出的 **添加反馈**。
文本框会出现在页面右侧的 feedback部分中。
4. 在文本框中输入您的反馈，然后点 **Submit**。
已创建一个文档问题。
5. 要查看问题，请单击反馈视图中的问题跟踪器链接。

使开源包含更多

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

第 1 章 使用 ECLIPSE VERT.X 进行应用程序开发简介

本节介绍使用红帽运行时应用程序开发的基本概念。它还概述 Eclipse Vert.x 运行时。

1.1. 使用 RED HAT RUNTIMES 应用程序开发概述

Red Hat OpenShift 是一个容器应用程序平台，它提供一组云原生运行时。您可以使用运行时在 OpenShift 中开发、构建和部署 Java 或 JavaScript 应用。

使用 Red Hat Runtimes for OpenShift 的应用程序开发包括：

- 一个运行时集合，如 Eclipse Vert.x、T Thorntail、Spring Boot 等，它设计为在 OpenShift 上运行。
- OpenShift 上云原生开发的一个指定方法。

OpenShift 可帮助您管理、保护和自动化应用的部署与监控。您可以将业务问题划分为较小的微服务，并使用 OpenShift 部署、监控和维护微服务。您可以在应用中实施诸如断路器、健康检查和服务发现等模式。

云原生开发充分利用了云计算。

您可通过以下方式构建、部署和管理应用程序：

OpenShift Container Platform
: 红帽内部部署的私有云.

Red Hat CodeReady Studio
: 用于开发、测试和部署应用程序的集成开发环境(IDE)。

本指南提供有关 Eclipse Vert.x 运行时的详细信息。如需有关其他运行时的更多信息，请参阅相关的 运行时文档。

1.2. ECLIPSE VERT.X 概述

Eclipse Vert.x 是用于创建被动、非阻塞和在 Java 虚拟机(JVM)上运行的异步应用程序的工具箱。

Eclipse Vert.x 旨在成为云原生。它允许应用程序使用非常少的线程。这可避免在创建新线程时导致的开销。这可让 Eclipse Vert.x 应用和服务有效使用其内存以及云环境中的 CPU 配额。

在 OpenShift 中使用 Eclipse Vert.x 运行时，可以更轻松地构建被动系统。OpenShift 平台的功能也可用，如滚动更新、服务发现和 canary 部署。借助 OpenShift，您可以在您的应用中实施外部化配置、健康检查、断路器和故障转移等微服务模式。

1.2.1. Eclipse Vert.x 的主要概念

本节介绍了与 Eclipse Vert.x 运行时关联的一些关键概念。它还简要概述被动系统。

云和容器应用程序

云原生应用通常利用微服务构建。它们设计为形成分离组件的分布式系统。这些组件通常在包含大量节点的集群之上在容器内运行。这些应用程序预计会在不需要任何服务停机时间的情况下更新各个组件的故障。基于云原生应用的系统依赖于由底层云平台（如 OpenShift）提供的自动化部署、扩展和管理和维护任务。使用现成的管理和编排工具在集群级别上执行管理和配置任务，而不是在各个机器的水平执行。

被动系统

被动系统(Reactive manifesto 中定义)是一个分布式系统，它具有以下特征：

Elastic

系统在不同的工作负载下保持响应，各个组件根据需要进行扩展和负载均衡，以适应工作负载的不同部分。弹性应用程序无论它们同时收到的请求数量如何，都可提供相同的服务质量。

弹性

即使有任何独立组件失败，系统也会保持响应。在系统中，组件相互隔离。这有助于在失败时快速恢复各个组件。单个组件故障不应影响其他组件的功能。这可防止级联失败，而隔离组件的失败导致其他组件被阻断，并逐渐失败。

响应

快速响应的系统设计为始终以合理的时间内响应请求，以确保提供一致的服务质量。为了保持响应响应，应用程序间的通信通道绝对不会被阻止。

message-Driven

应用的各个组件使用异步消息传递来相互通信。如果事件发生（如鼠标点击或服务上的搜索查询），服务会发送通用频道的消息（即事件总线）。消息反过来由相应的组件发现并由相应的组件处理。

被动系统是分布式系统。它们设计为可将其异步属性用于应用程序开发。

被动编程

被动系统的概念描述了分布式系统的架构，但被动编程是指使应用程序在代码级别重新活跃的做法。被动编程是一种开发模型，用于编写异步和事件驱动的应用程序。在被动应用程序中，代码对事件或消息做出反应。

被动编程有多种实现。例如，使用回调进行简单的实施，使用 Reactive Extensions (Rx)和 coroutines 的复杂实现。

Reactive Extensions (Rx)是 Java 中最成熟的被动编程形式之一。它使用 *RxJava* 库。

1.2.2. Eclipse Vert.x 支持的架构

Eclipse Vert.x 支持以下构架：

- x86_64 (AMD64)
- OpenShift 环境中的 IBM Z (s390x)
- OpenShift 环境中的 IBM Power Systems (ppc64le)

如需有关镜像名称的更多信息，请参阅 Eclipse Vert.x 的部分支持的 Java 镜像 。

1.2.3. 支持联邦信息处理标准(FIPS)

联邦信息处理标准(FIPS)为提高计算机系统和网络之间的安全性和互操作性提供指导和要求。FIPS 140-2 和 140-3 系列适用于硬件和软件级别的加密模块。

联邦信息处理标准(FIPS)出版物 140-2 是美国开发的计算机安全标准。政府和行业工作组来验证加密模块的质量。请参阅 NIST 计算机安全资源中心 上的官方 FIPS 出版物。

Red Hat Enterprise Linux (RHEL)提供了一个集成框架，可启用 FIPS 140-2 合规性系统。在 FIPS 模式中运行时，使用加密库的软件包会根据全局策略自助配置。

要了解合规要求，请参阅 红帽政府标准 页面。

Red Hat build of Eclipse Vert.x 在启用了 FIPS 的 RHEL 系统上运行，并使用 RHEL 提供的 FIPS 认证库。

1.2.3.1. 其他资源

- 有关如何使用启用 FIPS 模式安装 RHEL 的更多信息，请参阅安装启用了 FIPS 模式的 RHEL 8 系统。
- 有关如何在安装 RHEL 后启用 FIPS 模式的更多信息，请参阅 将系统切换到 FIPS 模式。

第 2 章 配置应用程序

本节介绍如何将应用程序配置为使用 Eclipse Vert.x 运行时。

2.1. 将应用程序配置为使用 ECLIPSE VERT.X

当您开始将应用程序配置为使用 Eclipse Vert.x 时，您必须在应用程序的根目录下的 **pom.x BOM (Bill of Materials)构件中引用 Eclipse Vert.** x BOM (Bill of Materials)构件。BOM 用于设置工件的正确版本。

前提条件

- 基于 Maven 的应用程序

流程

1. 打开 **pom.xml** 文件，将 **io.vertx:vertx-dependencies** 构件添加到 < **dependencyManagement>** 部分。将 **类型指定为 pom**，**范围** 指定为 导入。

```
<project>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>io.vertx</groupId>
        <artifactId>vertx-dependencies</artifactId>
        <version>${vertx.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

2. 包含以下属性来跟踪 Eclipse Vert.x 和 Eclipse Vert.x Maven 插件的版本。
属性可用于设置在每个版本中更改的值。例如，产品或插件的版本。

```
<project>
  ...
  <properties>
    <vertx.version>${vertx.version}</vertx.version>
    <vertx-maven-plugin.version>${vertx-maven-plugin.version}</vertx-maven-plugin.version>
  </properties>
  ...
</project>
```

3. 指定 **vertx-maven-plugin** 作为用于打包应用程序的插件：

```
<project>
  ...
  <build>
    <plugins>
      ...
      <plugin>
```

```
          <groupId>io.reactiverse</groupId>
          <artifactId>vertx-maven-plugin</artifactId>
          <version>${vertx-maven-plugin.version}</version>
          <executions>
            <execution>
              <id>vmp</id>
              <goals>
                <goal>initialize</goal>
                <goal>package</goal>
              </goals>
            </execution>
          </executions>
          <configuration>
            <redeploy>true</redeploy>
          </configuration>
        </plugin>
        ...
      </plugins>
    </build>
    ...
  </project>
```

4. 包含 **软件**仓库 **和插件，** 以指定包含工件和插件的存储库以构建应用程序：

```
<project>
...
  <repositories>
    <repository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </pluginRepository>
  </pluginRepositories>
...
</project>
```

其他资源

- 有关打包 Eclipse Vert.x 应用程序的更多信息，请参阅 Vert.x Maven 插件 文档。

第 3 章 开发和部署 ECLIPSE VERT.X 运行时应用程序

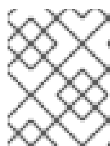
您可以创建一个新的 Eclipse Vert.x 应用程序，并将其部署到 OpenShift 或独立 Red Hat Enterprise Linux。

3.1. 开发 ECLIPSE VERT.X 应用程序

对于基本的 Eclipse Vert.x 应用程序，您需要创建以下内容：

- 包含 Eclipse Vert.x 方法的 Java 类。
- 包含 Maven 构建应用程序所需的信息的 **pom.xml** 文件。

以下流程创建了一个简单的 **Greeting** 应用程序，它将返回 "Greetings!" 作为回答。



注意

要将应用程序构建和部署到 OpenShift，Eclipse Vert.x 4.3 仅支持基于 OpenJDK 8 和 OpenJDK 11 的构建器镜像。不支持 Oracle JDK 和 OpenJDK 9 构建器镜像。

前提条件

- 安装了 OpenJDK 8 或 OpenJDK 11。
- 已安装 Maven。

流程

1. 创建新目录 **myApp**，并导航到它。

   ```
   $ mkdir myApp
   $ cd myApp
   ```

   这是应用的根目录。

2. 在根目录中创建目录结构 **src/main/java/com/example/**，然后导航到.

   ```
   $ mkdir -p src/main/java/com/example/
   $ cd src/main/java/com/example/
   ```

3. 创建包含应用代码的 Java 类文件 **MyApp.java**。

   ```
   package com.example;

   import io.vertx.core.AbstractVerticle;
   import io.vertx.core.Promise;

   public class MyApp extends AbstractVerticle {

       @Override
       public void start(Promise<Void> promise) {
           vertx
               .createHttpServer()
               .requestHandler(r ->
   ```

```
                r.response().end("Greetings!"))
              .listen(8080, result -> {
                if (result.succeeded()) {
                  promise.complete();
                } else {
                  promise.fail(result.cause());
                }
              });
    }
}
```

4. 在应用程序根目录 **myApp** 中创建 **pom.xml** 文件，其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>My Application</name>
  <description>Example application using Vert.x</description>

  <properties>
    <vertx.version>4.3.7.redhat-00002</vertx.version>
    <vertx-maven-plugin.version>1.0.24</vertx-maven-plugin.version>
    <vertx.verticle>com.example.MyApp</vertx.verticle>

    <!-- Specify the JDK builder image used to build your application. -->
    <jkube.generator.from>registry.access.redhat.com/ubi8/openjdk-11</jkube.generator.from>

    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  </properties>

  <!-- Import dependencies from the Vert.x BOM. -->
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>io.vertx</groupId>
        <artifactId>vertx-dependencies</artifactId>
        <version>${vertx.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <!-- Specify the Vert.x artifacts that your application depends on. -->
```

```
  <dependencies>
    <dependency>
      <groupId>io.vertx</groupId>
      <artifactId>vertx-core</artifactId>
    </dependency>
    <dependency>
      <groupId>io.vertx</groupId>
      <artifactId>vertx-web</artifactId>
    </dependency>
  </dependencies>

  <!-- Specify the repositories containing Vert.x artifacts. -->
  <repositories>
    <repository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
  </repositories>

  <!-- Specify the repositories containing the plugins used to execute the build of your
application. -->
  <pluginRepositories>
    <pluginRepository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </pluginRepository>
  </pluginRepositories>

  <!-- Configure your application to be packaged using the Vert.x Maven Plugin. -->
  <build>
    <plugins>
      <plugin>
        <groupId>io.reactiverse</groupId>
        <artifactId>vertx-maven-plugin</artifactId>
        <version>${vertx-maven-plugin.version}</version>
        <executions>
          <execution>
            <id>vmp</id>
            <goals>
              <goal>initialize</goal>
              <goal>package</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

5. 从应用的根目录使用 Maven 构建应用。

```
$ mvn vertx:run
```

6. 验证应用是否正在运行。
   使用 **curl** 或您的浏览器，验证您的应用程序正在 **http://localhost:8080** 中运行。

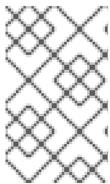
```
$ curl http://localhost:8080
Greetings!
```

附加信息

- 作为建议的做法，您可以配置存活度和就绪度探测，以便在 OpenShift 上运行时为应用启用健康状态监控。

3.2. 将 ECLIPSE VERT.X 应用部署到 OPENSHIFT

要将 Eclipse Vert.x 应用部署到 OpenShift，请在应用中配置 **pom.xml** 文件，然后使用 OpenShift Maven 插件。



注意

Fabric8 Maven 插件不再被支持。使用 OpenShift Maven 插件在 OpenShift 上部署 Eclipse Vert.x 应用。如需更多信息，请参阅从 Fabric8 Maven 插件迁移到 Eclipse JKube 的部分。

您可以通过在 **pom.xml** 文件中替换 **jkube.generator.from** URL 来指定 Java 镜像。镜像在 红帽生态系统目录 中可用。

```
<jkube.generator.from>IMAGE_NAME</jkube.generator.from>
```

例如，使用 OpenJDK 8 的 RHEL 7 的 Java 镜像被指定为：

```
<jkube.generator.from>registry.access.redhat.com/redhat-openjdk-18/openjdk18-
openshift:latest</jkube.generator.from>
```

3.2.1. 支持的 Eclipse Vert.x 的 Java 镜像

Eclipse Vert.x 通过可用于不同操作系统的各种 Java 镜像进行认证和测试。例如，对于带有 OpenJDK 8 或 OpenJDK 11 的 RHEL 7 提供了 Java 镜像。

Eclipse Vert.x 引入了对使用 Red Hat OpenJDK 8 和 Red Hat OpenJDK 11 的 OCI 兼容 通用基本镜像 OpenShift 构建和部署 Eclipse Vert.x 应用程序的支持。

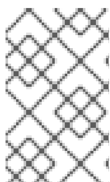
您需要 Docker 或 podman 身份验证来访问红帽生态系统目录中的 RHEL 8 镜像。

下表列出了用于不同架构的 Eclipse Vert.x 支持的容器镜像。这些容器镜像在 红帽生态系统目录 中提供。在目录中，您可以搜索并下载下表中列出的镜像。镜像页面包含访问镜像所需的身份验证过程。

表 3.1. OpenJDK 镜像和架构

JDK (OS)	支持的构架	红帽生态系统目录中可用的镜像
OpenJDK8 (RHEL 7)	x86_64	redhat-openjdk-18/openjdk18-openshift

JDK (OS)	支持的构架	红帽生态系统目录中可用的镜像
OpenJDK11 (RHEL 7)	x86_64	openjdk/openjdk-11-rhel7
OpenJDK8 (RHEL 8)	x86_64	ubi8/openjdk-8-runtime
OpenJDK11 (RHEL 8)	x86_64、IBM Z 和 IBM Power 系统	ubi8/openjdk-11



注意

在 RHEL 7 主机上使用基于 RHEL 8 的容器（例如，OpenShift 3 或 OpenShift 4）具有有限的支持。如需更多信息，请参阅 Red Hat Enterprise Linux Container Compatibility Matrix。

3.2.2. 为 OpenShift 部署准备 Eclipse Vert.x 应用

要将 Eclipse Vert.x 应用程序部署到 OpenShift，它必须包含：

- 在应用程序的 **pom.xml** 文件中启动程序配置集信息。

在以下流程中，使用带有 OpenShift Maven 插件的配置集来构建和部署应用到 OpenShift。

前提条件

- 已安装 Maven。
- Docker 或 podman 身份验证到红帽生态系统目录，以访问 RHEL 8 镜像。

流程

1. 在应用程序根目录下的 **pom.xml** 文件中添加以下内容：

```
<!-- Specify the JDK builder image used to build your application. -->
<properties>
  <jkube.generator.from>registry.access.redhat.com/redhat-openjdk-18/openjdk18-
openshift:latest</jkube.generator.from>
</properties>

...

<profiles>
    <profile>
      <id>openshift</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.eclipse.jkube</groupId>
            <artifactId>openshift-maven-plugin</artifactId>
            <version>1.1.1</version>
            <executions>
              <execution>
                <goals>
```

```
            <goal>resource</goal>
            <goal>build</goal>
            <goal>apply</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
  </build>
 </profile>
</profiles>
```

2. 替换 **pom.xml** 文件中的 **jkube.generator.from** 属性，以指定您要使用的 OpenJDK 镜像。
   - x86_64 架构
     - RHEL 7 with OpenJDK 8

       ```
       <jkube.generator.from>registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift:latest</jkube.generator.from>
       ```

     - RHEL 7 with OpenJDK 11

       ```
       <jkube.generator.from>registry.access.redhat.com/openjdk/openjdk-11-rhel7:latest</jkube.generator.from>
       ```

     - RHEL 8 with OpenJDK 8

       ```
       <jkube.generator.from>registry.access.redhat.com/ubi8/openjdk-8:latest</jkube.generator.from>
       ```

   - x86_64、s390x (IBM Z)和 ppc64le (IBM Power Systems)架构
     - RHEL 8 with OpenJDK 11

       ```
       <jkube.generator.from>registry.access.redhat.com/ubi8/openjdk-11:latest</jkube.generator.from>
       ```

3.2.3. 使用 OpenShift Maven 插件将 Eclipse Vert.x 应用部署到 OpenShift

要将 Eclipse Vert.x 应用程序部署到 OpenShift，您必须执行以下操作：

- 登录您的 OpenShift 实例。
- 将应用部署到 OpenShift 实例。

前提条件

- 已安装 **oc** CLI 客户端。
- 已安装 Maven。

流程

1. 使用 **oc** 客户端登录您的 OpenShift 实例。

```
$ oc login ...
```

2. 在 OpenShift 实例中创建一个新项目。

```
$ oc new-project MY_PROJECT_NAME
```

3. 使用应用的根目录中的 Maven 将应用部署到 OpenShift。应用程序的根目录包含 **pom.xml** 文件。

```
$ mvn clean oc:deploy -Popenshift
```

此命令使用 OpenShift Maven 插件在 OpenShift 上启动 S2I 进程，并启动容器集。

4. 验证部署。

a. 检查应用程序的状态，并确保您的 Pod 正在运行。

```
$ oc get pods -w
NAME                         READY   STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa          1/1     Running     0          58s
MY_APP_NAME-s2i-1-build      0/1     Completed   0          2m
```

MY_APP_NAME-1-aaaaa pod 在完全部署并启动后应处于 **Running** 状态。

具体 pod 名称会有所不同。

b. 确定 pod 的路由。

路由信息示例

```
$ oc get routes
NAME          HOST/PORT                                        PATH      SERVICES
PORT     TERMINATION
MY_APP_NAME       MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME       MY_APP_NAME      8080
```

pod 的路由信息为您提供了用于访问它的基本 URL。

在这个示例中， **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** 是用于访问应用程序的基本 URL。

c. 验证您的应用是否在 OpenShift 中运行。

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
Greetings!
```

3.3. 将 ECLIPSE VERT.X 应用程序部署为独立 RED HAT ENTERPRISE LINUX

要将 Eclipse Vert.x 应用部署到独立 Red Hat Enterprise Linux，请在应用中配置 **pom.xml** 文件，使用 Maven 打包并使用 **java -jar** 命令进行部署。

前提条件

- 安装了 RHEL 7 或 RHEL 8。

3.3.1. 为独立 Red Hat Enterprise Linux 部署准备 Eclipse Vert.x 应用程序

要将 Eclipse Vert.x 应用部署到独立 Red Hat Enterprise Linux，您必须首先使用 Maven 打包应用程序。

前提条件

- 已安装 Maven。

流程

1. 将以下内容添加到应用程序的根目录中的 **pom.xml** 文件中：

```
...
<build>
  <plugins>
    <plugin>
      <groupId>io.reactiverse</groupId>
      <artifactId>vertx-maven-plugin</artifactId>
      <version>1.0.24</version>
      <executions>
        <execution>
          <id>vmp</id>
          <goals>
            <goal>initialize</goal>
            <goal>package</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
...
```

2. 使用 Maven 打包您的应用。

```
$ mvn clean package
```

生成的 JAR 文件位于 **目标** 目录中。

3.3.2. 使用 jar 将 Eclipse Vert.x 应用部署到独立 Red Hat Enterprise Linux 中

要将 Eclipse Vert.x 应用部署到独立 Red Hat Enterprise Linux，请使用 **java -jar** 命令。

前提条件

- 安装了 RHEL 7 或 RHEL 8。
- 安装了 OpenJDK 8 或 OpenJDK 11。
- 带有该应用的 JAR 文件。

流程

1. 使用 应用部署 JAR 文件。

```
$ java -jar my-app-fat.jar
```

2. 验证部署。
使用 **curl** 或浏览器验证您的应用程序正在 **http://localhost:8080** 中运行：

```
$ curl http://localhost:8080
```

第 4 章 调试基于 ECLIPSE VERT.X 的应用

这部分包含有关在本地和远程部署中调试基于 Eclipse Vert.x 的应用的信息。

4.1. 远程调试

要远程调试应用，您必须首先将其配置为在调试模式中启动，然后为它附加一个调试器。

4.1.1. 以调试模式在本地启动应用程序

调试基于 Maven 的项目的方法之一是在指定调试端口的同时手动启动应用，然后将远程调试器连接到该端口。此方法至少适用于以下应用程序部署：

- 使用 **mvn vertx:debug** 目标启动应用程序时。这将启动启用调试的应用程序。

前提条件

- 基于 Maven 的应用程序

流程

1. 在控制台中，导航到您的应用程序的目录。
2. 启动应用程序并使用 **-Ddebug.port** 参数指定 debug 端口：

```
$ mvn vertx:debug -Ddebug.port=$PORT_NUMBER
```

   这里的 **$PORT_NUMBER** 是您选择的未使用端口号。记住此号码用于远程调试器配置。

   使用 **-Ddebug.suspend=true** 参数使应用程序等待等待调试器启动。

4.1.2. 在 OpenShift 中启动应用程序处于调试模式

要在 OpenShift 上远程调试基于 Eclipse Vert.x 的应用，您必须将容器内的 **JAVA_DEBUG** 环境变量设置为 **true**，并配置端口转发，以便您可以从远程调试程序连接到您的应用。

前提条件

- 在 OpenShift 上运行的应用。
- 已安装 **oc** 二进制文件。
- 在目标 OpenShift 环境中执行 **oc port-forward** 命令的功能。

流程

1. 使用 **oc** 命令，列出可用的部署配置：

```
$ oc get dc
```

2. 将应用的部署配置中的 **JAVA_DEBUG** 环境变量设置为 **true**，它将 JVM 配置为打开端口号 **5005** 以进行调试。例如：

```
$ oc set env dc/MY_APP_NAME JAVA_DEBUG=true
```

3. 如果应用程序没有设置为在配置更改时自动重新部署，请重新部署应用程序。例如：

```
$ oc rollout latest dc/MY_APP_NAME
```

4. 配置本地机器到应用程序 pod 的端口转发：

   a. 列出当前运行的 pod，并找到包含您的应用程序的 pod:

   ```
   $ oc get pod
   NAME                    READY   STATUS    RESTARTS  AGE
   MY_APP_NAME-3-1xrsp     0/1     Running   0         6s
   ...
   ```

   b. 配置端口转发：

   ```
   $ oc port-forward MY_APP_NAME-3-1xrsp $LOCAL_PORT_NUMBER:5005
   ```

   在这里，**$LOCAL_PORT_NUMBER** 是本地机器上您选择的未使用端口号。记住此号码用于远程调试器配置。

5. 完成调试后，取消设置应用程序 pod 中的 **JAVA_DEBUG** 环境变量。例如：

```
$ oc set env dc/MY_APP_NAME JAVA_DEBUG-
```

其他资源

如果要从默认值更改 debug 端口，则可设置 **JAVA_DEBUG_PORT** 环境变量，即 **5005**。

4.1.3. 将远程调试器附加到应用程序

当应用程序被配置为调试时，为它附加您选择的远程调试程序。在本指南中，涵盖了 Red Hat CodeReady Studio，但在使用其他程序时的步骤类似。

前提条件

- 在本地或 OpenShift 上运行的应用，并配置了调试。
- 应用程序侦听调试的端口号。
- Red Hat CodeReady Studio installed on your machine.您可以从 Red Hat CodeReady Studio 下载页面 下载。

流程

1. 启动 Red Hat CodeReady Studio。
2. 为应用程序创建新调试配置：

   a. 点 **Run→Debug Configuration**。

   b. 在配置列表中，双击 **Remote Java 应用。**这会创建一个新的远程调试配置。

c. 在 **Name** 字段中输入适合配置的名称。

d. 在 **Project** 字段中输入应用程序目录的路径。为方便起见，您可以使用 **Browse…** 按钮。

e. 将 **Connection Type** 字段设置为 *Standard (Socket Attach)* （如果尚未连接）。

f. 将 **Port** 字段设置为应用程序侦听调试的端口号。

g. 点 **应用。**

3. 点 Debug Configuration 窗口中的 **Debug** 按钮开始调试。
要在首次时间后快速启动您的调试配置，请点击 **Run→Debug History**，并从列表中选择配置。

其他资源

- 在红帽知识库中使用 JBoss Developer Studio 调试 OpenShift Java 应用 。
Red Hat CodeReady Studio 之前称为 JBoss Developer Studio。
- OpenShift 博客中的 调试 Java 应用 OpenShift 和 Kubernetes 文章.

4.2. 调试日志记录

Eclipse Vert.x 提供了一个内置日志记录 API。Eclipse Vert.x 的默认日志实现使用 Java JDK 提供的 **java.util.logging** 库。另外，Eclipse Vert.x 允许您使用不同的日志记录框架，如 *Log4J* (Eclipse Vert.x 支持 Log4J v1 和 v2)或 *SLF4J*。

4.2.1. 使用 java.util.logging为 Eclipse Vert.x 应用程序配置日志记录

使用 **java.util.logging** 为 Eclipse Vert.x 应用程序配置调试日志记录：

- 在 **application.properties** 文件中设置 **java.util.logging.config.file** 系统属性。此变量的值必须与 **java.util.logging** 配置文件的 名称对应。这样可确保 **LogManager** 在应用程序启动时初始化 **java.util.logging**。
- 另外，还可在 Maven 项目的类路径中添加带有 **vertx-default-jul-logging.properties** 名称的 **java.util.logging** 配置文件。Eclipse Vert.x 将使用该文件来配置应用程序启动时 **java.util.logging**。

通过 Eclipse Vert.x，您可以使用 **LogDelegateFactory** 指定自定义日志记录后端，它为 **Log4J**、**Log4J2** 和 **SLF4J** 库提供预建的实施。与默认情况下 Java 中包含的 **java.util.logging** 不同，其他后端需要您将相应的库指定为应用程序的依赖项。

4.2.2. 在 Eclipse Vert.x 应用中添加日志输出。

1. 要向应用程序添加日志记录，请创建一个 **io.vertx.core.logging.Logger** ：

```
Logger logger = LoggerFactory.getLogger(className);

logger.info("something happened");
logger.error("oops!", exception);
logger.debug("debug message");
logger.warn("warning");
```

小心

日志记录后端使用不同的格式来代表参数化消息中的可替换令牌。如果您依赖参数化日志记录方法，您将无法在不更改代码的情况下切换日志后端。

4.2.3. 为应用程序指定自定义日志框架

如果您不希望 Eclipse Vert.x 使用 **java.util.logging**，将 **io.vertx.core.logging.Logger** 配置为使用不同的日志记录框架，如 **Log4J** 或 **SLF4J** ：

1. 将 **vertx.logger-delegate-factory-class-name** 系统属性的值设置为实现 **LogDelegateFactory** 接口的类名称。Eclipse Vert.x 为以下库提供预建的实现，其对应的预定义类名称如下：

程序库	类名称
Log4J v1	**io.vertx.core.logging.Log4jLogDelegateFactory**
Log4J v2	**io.vertx.core.logging.Log4j2LogDelegateFactory**
SLF4J	**io.vertx.core.logging.SLF4JLogDelegateFactory**

使用自定义库实施日志记录时，请确保相关的 **Log4J** 或 **SLF4J** jars 包含在应用程序的依赖项中。

小心

通过 Eclipse Vert.x 提供的 *Log4J* v1 委派不支持参数化消息。*Log4J v2* 和 *SLF4J* 的委派都使用 **{}** 语法。**java.util.logging** 委派依赖于使用 **{n}** 语法的 **java.text.MessageFormat**。

4.2.4. 为 Eclipse Vert.x 应用程序配置 Netty 日志记录。

Netty 是一个库，供 VertX 用于管理应用程序中异步网络通信。

Netty：

- 能够快速轻松地开发网络应用程序，如协议服务器和客户端。
- 简化和简化网络编程，如 TCP 和 UDP 套接字服务器开发。
- 提供统一的 API，用于管理阻塞和非阻塞连接。

Netty 不依赖于使用系统属性的外部日志记录配置。相反，它会基于对项目中的 Netty 类可见的日志记录库实施日志配置。Netty 会尝试按照以下顺序使用库：

1. **SLF4J**
2. **Log4J**
3. **java.util.logging** 作为回退选项

您可以通过在应用程序 **的主** 方法的开头添加以下代码，将 **io.netty.util.internal.logging.InternalLoggerFactory** 设置为特定日志记录器：

```
// Force logging to Log4j
InternalLoggerFactory.setDefaultFactory(Log4JLoggerFactory.INSTANCE);
```

4.2.5. 访问 OpenShift 上的调试日志

启动您的应用并与之交互，以查看 OpenShift 中的调试语句。

前提条件

- 已安装并验证 **oc** CLI 客户端。
- 启用调试日志记录的基于 Maven 的应用。

流程

1. 将应用程序部署到 OpenShift：

   ```
   $ mvn clean oc:deploy -Popenshift
   ```

2. 查看日志：
   1. 使用您的应用程序获取 pod 的名称：

      ```
      $ oc get pods
      ```

   2. 开始监视日志输出：

      ```
      $ oc logs -f pod/MY_APP_NAME-2-aaaaa
      ```

      保持打开日志输出的终端窗口，以便您可以观察日志输出。

3. 与应用程序交互：
   例如，以下命令基于一个示例 REST API 级别 0 应用程序，其中 debug logging 设置为在 **/api/greeting** 方法中记录 **消息** 变量：
   1. 获取应用程序的路由：

      ```
      $ oc get routes
      ```

   2. 在应用程序的 **/api/greeting** 端点上发出 HTTP 请求：

      ```
      $ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
      ```

4. 返回到包含 Pod 日志的窗口，并检查日志中的调试日志消息。

   ```
   ...
   Feb 11, 2017 10:23:42 AM io.openshift.MY_APP_NAME
   INFO: Greeting: Hello, Sarah
   ...
   ```

5. 要禁用 debug 日志记录，请更新日志记录配置文件，如 **src/main/resources/vertx-default-jul-logging.properties**，删除您的类的日志配置并重新部署您的应用程序。

第 5 章 监控应用程序

本节介绍监控 OpenShift 上运行的基于 Eclipse Vert.x 的应用。

5.1. 在 OPENSHIFT 中访问应用的 JVM 指标

5.1.1. 在 OpenShift 中使用 Jolokia 访问 JVM 指标

Jolokia 是内置的轻量级解决方案，可通过 OpenShift 上的 HTTP 访问 JMX (Java 管理扩展)指标。Jolokia 允许您通过 HTTP 网桥访问 JMX 收集的 CPU、存储和内存使用情况数据。Jolokia 使用 REST 接口和 JSON 格式的消息有效负载。它因其比较高速度和低资源要求而对云应用程序进行监控。

对于基于 Java 的应用，OpenShift Web 控制台提供了集成 *hawt.io* 控制台，该控制台由运行应用程序的 JVM 收集和显示所有相关指标输出。

先决条件

- 已验证 **oc** 客户端
- 在 OpenShift 上的项目中运行的基于 Java 的应用容器
- 最新 JDK 1.8.0 镜像

流程

1. 列出项目中容器器集的部署配置，再选择与应用程序对应的部署配置。

```
oc get dc
```

```
NAME         REVISION   DESIRED   CURRENT   TRIGGERED BY
MY_APP_NAME   2          1         1         config,image(my-app:6)
...
```

2. 打开运行您的应用程序的 pod 的 YAML 部署模板，进行编辑。

```
oc edit dc/MY_APP_NAME
```

3. 在模板的 **ports** 部分添加以下条目并保存您的更改：

```
...
spec:
  ...
  ports:
  - containerPort: 8778
    name: jolokia
    protocol: TCP
  ...
...
```

4. 重新部署运行应用程序的 pod。

```
oc rollout latest dc/MY_APP_NAME
```

pod 使用更新的部署配置重新部署，并公开端口 **8778**。

5. 登录 OpenShift Web 控制台。

6. 在侧边栏中，进入 *Applications* > *Pods*，然后点运行应用程序的 pod 的名称。

7. 在 pod 详情屏幕中，单击 *Open Java Console* 以访问 hawt.io 控制台。

其他资源

- Hawt.io 文档

5.2. 使用 ECLIPSE VERT.X 使用 PROMETHEUS 公开应用程序指标

Prometheus 连接到受监控的应用程序来收集数据，应用程序不会向服务器发送指标。

前提条件

- 集群中运行的 Prometheus 服务器

流程

1. 将 **vertx-micrometer** 和 **vertx-web** 依赖项包含到应用程序的 **pom.xml** 文件中：

pom.xml

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-micrometer-metrics</artifactId>
</dependency>
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-web</artifactId>
</dependency>
```

2. 从 3.5.4 开始，公开 Prometheus 的指标需要在自定义 **Launcher** 类中配置 Eclipse Vert.x 选项。在 custom **Launcher** 类中，覆盖 **beforeStartingVertx** 和 **afterStartingVertx** 方法来配置指标引擎，例如：

CustomLauncher.java 文件示例

```
package org.acme;

import io.micrometer.core.instrument.Meter;
import io.micrometer.core.instrument.config.MeterFilter;
import io.micrometer.core.instrument.distribution.DistributionStatisticConfig;
import io.micrometer.prometheus.PrometheusMeterRegistry;
import io.vertx.core.Vertx;
import io.vertx.core.VertxOptions;
import io.vertx.core.http.HttpServerOptions;
import io.vertx.micrometer.MicrometerMetricsOptions;
import io.vertx.micrometer.VertxPrometheusOptions;
import io.vertx.micrometer.backends.BackendRegistries;
```

```
public class CustomLauncher extends Launcher {

  @Override
  public void beforeStartingVertx(VertxOptions options) {
    options.setMetricsOptions(new MicrometerMetricsOptions()
      .setPrometheusOptions(new VertxPrometheusOptions().setEnabled(true)
        .setStartEmbeddedServer(true)
        .setEmbeddedServerOptions(new HttpServerOptions().setPort(8081))
        .setEmbeddedServerEndpoint("/metrics"))
      .setEnabled(true));
  }

  @Override
  public void afterStartingVertx(Vertx vertx) {
    PrometheusMeterRegistry registry = (PrometheusMeterRegistry)
BackendRegistries.getDefaultNow();
    registry.config().meterFilter(
      new MeterFilter() {
        @Override
        public DistributionStatisticConfig configure(Meter.Id id, DistributionStatisticConfig config)
{
          return DistributionStatisticConfig.builder()
            .percentilesHistogram(true)
            .build()
            .merge(config);
        }
    });
}
```

3. 创建自定义 **Verticle** 类，并覆盖 **start** 方法来收集指标。例如，使用 **Timer** 类测量执行时间：

CustomVertxApp.java 文件示例

```
package org.acme;

import io.micrometer.core.instrument.MeterRegistry;
import io.micrometer.core.instrument.Timer;
import io.vertx.core.AbstractVerticle;
import io.vertx.core.Vertx;
import io.vertx.core.VertxOptions;
import io.vertx.core.http.HttpServerOptions;
import io.vertx.micrometer.backends.BackendRegistries;

public class CustomVertxApp extends AbstractVerticle {

  @Override
  public void start() {
    MeterRegistry registry = BackendRegistries.getDefaultNow();
    Timer timer = Timer
      .builder("my.timer")
      .description("a description of what this timer does")
      .register(registry);

    vertx.setPeriodic(1000, l -> {
      timer.record(() -> {
```

```
        // Do something

      });
    });
  }
}
```

4. 在应用程序的 **pom .xml 文件中设置 < vertx.verticle > 和 <vertx.launcher** > 属性以指向您的自定义类：

```
<properties>
  ...
  <vertx.verticle>org.acme.CustomVertxApp</vertx.verticle>
  <vertx.launcher>org.acme.CustomLauncher</vertx.launcher>
  ...
</properties>
```

5. 启动应用程序：

```
$ mvn vertx:run
```

6. 多次调用 traced 端点：

```
$ curl http://localhost:8080/
Hello
```

7. 等待至少 15 秒进行集合，并在 Prometheus UI 中看到指标：

    1. 在 http://localhost:9090/ 上打开 Prometheus UI，然后在 *Expression* 框中输入 **hello**。

    2. 从建议中选择 **application:hello_count** 示例，再单击 *Execute*。

    3. 在显示的表中，您可以看到调用资源方法的次数。

    4. 或者，选择 **application:hello_time_mean_seconds** 以查看所有调用的平均值时间。

    请注意，您创建的所有指标都带有 **application:** 前缀。还有其他指标，作为 Eclipse MicroProfile Metrics 规格要求，由 Eclipse Vert.x 自动公开。这些指标以 **base:** 和 **vendor:** 前缀，并公开有关应用在其中运行的 JVM 的信息。

其他资源

- 有关在 Eclipse Vert.x 中使用 Micrometer 指标的更多信息，请参阅 Eclipse Vert.x} Micrometer Metrics。

附录 A. SOURCE-TO-IMAGE (S2I)构建过程

Source-to-Image (S2I)是一种构建工具，用于从带有应用程序源的在线 SCM 存储库生成可重复生成的 Docker 格式容器镜像。借助 S2I 构建，您可以轻松地将应用程序的最新版本提供给生产环境，缩短构建时间、减少资源和网络使用量，提高安全性，以及许多其他优势。OpenShift 支持多种 构建策略和输入源。

如需更多信息，请参阅 OpenShift Container Platform 文档中的 Source-to-Image (S2I)构建 章节。

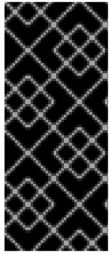
您必须向 S2I 流程提供三个元素来汇编最终容器镜像：

- 托管在在线 SCM 存储库中的应用源，如 GitHub。
- S2I Builder 镜像，作为汇编镜像的基础，提供应用程序的运行生态系统。
- 另外，您还可以提供 S2I 脚本使用 的环境变量和参数。

这个过程会根据 S2I 脚本中指定的指令将应用程序源和依赖项注入 Builder 镜像，并生成运行汇编的应用程序的 Docker 格式容器镜像。如需更多信息，请查看 S2I 构建要求，构建选项 以及如何 OpenShift Container Platform 文档中的构建工作 部分。

附录 B. 更新示例应用程序的部署配置

示例应用的部署配置包含与在 OpenShift 中部署和运行应用相关的信息，如路由信息或就绪度探测位置。示例应用的部署配置存储在一组 YAML 文件中。对于使用 OpenShift Maven 插件的示例，YAML 文件位于 **src/main/jkube/** 目录中。例如，使用 Nodeshift，YAML 文件位于 **.nodeshift** 目录中。



重要

OpenShift Maven 插件和 Nodeshift 使用的部署配置文件不一定是完整的 OpenShift 资源定义。OpenShift Maven 插件和 Nodeshift 都可以获取部署配置文件，并添加一些缺少的信息来创建完整的 OpenShift 资源定义。由 OpenShift Maven 插件生成的资源定义位于 **target/classes/META-INF/jkube/** 目录中。由 Nodeshift 生成的资源定义位于 **tmp/nodeshift/resource/** 目录中。

前提条件

- 现有示例项目。
- 已安装 **oc** CLI 客户端。

流程

1. 编辑现有的 YAML 文件，或使用您的配置更新创建额外的 YAML 文件。
   - 例如，如果您的示例已经配置了 **readinessProbe** 的 YAML 文件，您可以将 **路径** 值改为不同的可用路径来检查就绪度：

```
spec:
  template:
    spec:
      containers:
        readinessProbe:
          httpGet:
            path: /path/to/probe
            port: 8080
            scheme: HTTP
...
```

   - 如果现有 YAML 文件中没有配置 **readinessProbe**，您也可以在使用 **readinessProbe** 配置在同一目录中创建一个新的 YAML 文件。
2. 使用 Maven 或 npm 部署示例的更新版本。
3. 验证您的配置更新是否显示在您的示例中部署的版本中。

```
$ oc export all --as-template='my-template'

apiVersion: template.openshift.io/v1
kind: Template
metadata:
  creationTimestamp: null
  name: my-template
objects:
- apiVersion: template.openshift.io/v1
  kind: DeploymentConfig
```

```
...
spec:
  ...
  template:
    ...
    spec:
      containers:
        ...
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /path/to/different/probe
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 60
          periodSeconds: 30
          successThreshold: 1
          timeoutSeconds: 1
        ...
```

其他资源

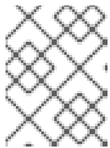
如果您使用基于 Web 控制台或 **oc** CLI 客户端直接更新应用程序的配置，请导出并把这些更改添加到您的 YAML 文件中。使用 **oc export all** 命令来显示您已部署的应用程序的配置。

附录 C. 配置 JENKINS 自由风格的项目，以使用 OPENSHIFT MAVEN 插件部署应用程序

与使用本地主机的 OpenShift Maven 插件类似来部署应用，您可以将 Jenkins 配置为使用 Maven 和 OpenShift Maven 插件来部署应用。

前提条件

- 访问 OpenShift 集群。
- 在同一 OpenShift 集群上运行的 Jenkins 容器镜像。
- 在 Jenkins 服务器上安装和配置了 JDK 和 Maven。
- 配置为使用 **pom.xml** 中的 OpenShift Maven 插件的应用，并使用 RHEL 基础镜像构建。



注意

要将应用程序构建和部署到 OpenShift，Eclipse Vert.x 4.3 仅支持基于 OpenJDK 8 和 OpenJDK 11 的构建器镜像。不支持 Oracle JDK 和 OpenJDK 9 构建器镜像。

pom.xml示例

```
<properties>
  ...
  <jkube.generator.from>registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift:latest</jkube.generator.from>
</properties>
```

- GitHub 中提供的应用源。

流程

1. 为您的应用程序创建一个新的 OpenShift 项目：
   a. 打开 OpenShift Web 控制台并登录。
   b. 单击 *Create Project* 以创建新的 OpenShift 项目。
   c. 输入项目信息并点 *Create*。
2. 确保 Jenkins 能够访问该项目。
   例如，如果您为 Jenkins 配置了服务帐户，请确保该帐户编辑了应用的项目的访问权限。
3. **在 Jenkins 服务器上创建新的可用 Jenkins 项目**：
   a. 单击 ***New Item***。

b. 输入名称，选择 *Freestyle project*，然后单击 *OK*。

c. 在 *Source Code Management* 下，选择 *Git* 并添加应用程序的 GitHub url。

d. 在 *Build* 下，选择 *Add build step* 并选择 Invoke top-level Maven 目标。

e. 将以下内容添加到 *Goals* 中：

```
clean oc:deploy -Popenshift -Djkube.namespace=MY_PROJECT
```

将 MY_PROJECT 替换为应用的 OpenShift 项目的名称。

a. 点 *Save*。

4. 从 Jenkins 项目主页，单击 *Build Now*，以验证您的应用构建并部署到 OpenShift 项目。

   您也可以通过在应用的 OpenShift 项目中打开路由来验证您的应用是否已部署。

后续步骤

- 考虑添加 GITSCM 轮询 或使用 Poll SCM 构建触发器。这些选项可让构建在每次将新提交推送到 GitHub 存储库时运行。
- 考虑添加在部署前执行测试的构建步骤。

附录 D. 额外的 ECLIPSE VERT.X 资源

- Reactive Manifesto
- Eclipse Vert.x 项目
- Action 中的 Vert.x
- 用于 Reactive Programming 的 Eclipse Vert.x
- 在 Java 中构建 Reactive 微服务
- Developers 的 Eclipse Vert.x Cheat Sheet
- Vert.x - 从零到(micro)-hero
- Red Hat Summit 2017 Talk - Reactive Programming with Eclipse Vert.x
- 2017 年红帽峰会 - Reactive Systems with Eclipse Vert.x 和红帽 OpenShift
- 使用 Eclipse Vert.x 和 OpenShift 的实时主动系统

附录 E. 应用程序开发资源

有关使用 OpenShift 应用程序开发的更多信息，请参阅：

- OpenShift 互动学习门户

要减少网络负载并缩短应用程序的构建时间，请在 OpenShift Container Platform 上为 Maven 设置 Nexus 镜像：

- 为 Maven 设置 Nexus 镜像