Red Hat support for Spring Boot 2.7

Spring Boot Developers 的 Dekorate 指南

使用 Dekorate 自动配置 Spring Boot 应用程序，以部署到 OpenShift 和独立 RHEL

Last Updated: 2023-12-07

Red Hat support for Spring Boot 2.7 Spring Boot Developers 的 Dekorate 指南

使用 Dekorate 自动配置 Spring Boot 应用程序，以部署到 OpenShift 和独立 RHEL

法律通告

摘要

本指南提供有关使用 Dekorate 从您的代码自动生成资源文件的详细信息，并准备您的 Spring Boot 应用以部署到多个环境。

目录

前言

使用 Dekorate 处理 Spring Boot 应用的代码，以自动生成应用清单文件并配置用于部署到 OpenShift 的应用。

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。

先决条件

- 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. 点文档右上角的 **Feedback** 按钮查看现有的反馈。

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. 点在高亮文本旁弹出的 **Add Feedback**。
   文本框中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**。
   文档问题已创建。
5. 要查看问题，请单击反馈视图中的问题跟踪器链接。

第 1 章 在 SPRING BOOT 应用程序中使用 DEKORATE

使用 Dekorate 自动生成应用清单文件，并配置要部署到 OpenShift 的应用。

1.1. DEKORATE 概述

Dekorate 是编译时注解处理器和应用程序资源生成器的集合，由红帽构建的 Spring Boot 提供。它的工作原理是在构建应用程序时在代码中解析注解，并提取配置属性。然后，Dekorate 使用提取的属性值来生成应用配置资源，可用于将应用部署到 Kubernetes 或 OpenShift 集群。

作为开发人员，您可以注解代码，然后使用 Dekorate 在构建应用程序时自动生成应用程序清单，这消除了手动编写用于部署应用程序的资源文件的需求。当应用程序基于丰富的应用程序运行时框架（如 Spring Boot）时，Dekorate 可以直接与框架集成，并从框架提供的 API 中提取配置参数，因此无需注解您的代码。Dekorate 可以通过以下方式自动配置应用程序：

- 在应用程序代码中解析 Dekorate 特定的注解，以获取用于填充清单文件的值和元数据
- 从配置资源中提取信息，如 **application.properties** 或 **application.yaml**
- 从丰富的应用程序框架中获取必要的元数据，并从 **application.properties** 或 **application.yml** 文件中提取配置值。

除了为应用程序生成资源定义外，Dekorate 还提供了 hook，允许您在 OpenShift 集群 Dekorate 上构建和部署您的应用，独立于您编写应用程序的语言，并可与各种构建系统一起使用。Dekorate 由一组作为 Maven BOM 分发的库组成。您可以将库添加为应用程序项目的依赖关系，以将 Dekorate 与应用程序搭配使用。

红帽提供对使用 Dekorate 来生成资源文件和 hook 的支持，您可以使用它来将基于 Spring Boot 部署到 OpenShift Container Platform 的 Java 应用程序。

1.1.1. 其他资源

- OpenShift 的 Dekorate 配置属性参考。
- Source-to-Image 的 Dekorate 配置属性参考 。
- 社区 文档中所有 Dekorate Configuration 属性 的参考。

1.2. 将应用程序项目配置为使用 DEKORATE

将 Dekorate BOM 和 OpenShift Annotations Starter 添加到应用程序项目的 **pom.xml** 文件中。在源文件中包含基本注解，并使用 Maven 打包应用程序，以生成应用程序清单。

先决条件

- 基于 Maven 的 Java 应用程序项目配置为使用 Spring Boot。
- Java JDK 8 或 JDK 11 已安装。
- 已安装 Maven。

流程

1. 将 Dekorate OpenShift Spring Starter 添加到应用程序的 **pom.xml** 文件中，以启用 Dekorates，以接收应用程序源代码和资源文件：

```
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <!-- The OpenShift Spring Starter automatically imports the "io.dekorate:openshift-annotations" dependency. -->
      <groupId>io.dekorate</groupId>
      <artifactId>openshift-spring-starter</artifactId>
      <version>${dekorate.version}</version>
    </dependency>
    ...
  </dependencies>
...
<project>
```

2. 将 **@SpringBootApplication** 注解添加到应用程序项目的主类文件中：

```
package org.acme;

@SpringBootApplication
public class Application {
}
```

3. 打包应用程序，以处理带有 Dekorate 的应用程序代码和资源文件

```
mvn clean package
```

4. 导航到包含生成的 OpenShift 清单的 **target/classes/META-INF/dekorate** 目录。

1.3. 使用 DEKORATE 自定义应用程序配置

使用 Dekorate 自定义应用的配置，以便通过 OpenShift 部署

- 在源应用程序中指定注解中的配置参数
- 在 **application.properties** 文件中设置属性

以下示例演示了如何在部署到 OpenShift 时将应用设置为从 2 个副本开始。

先决条件

- 基于 Maven 的 Java 应用程序项目配置为使用 Spring Boot 和 Dekorate
- Java JDK 8 或 JDK 11 已安装。
- 已安装 Maven。

流程

1. 在应用程序的 **pom.xml** 文件中添加 Dekorate OpenShift Annotations 模块作为依赖项：

```
<project>
  ...
```

```
  <dependencies>
    ...
   <dependency>
    <groupId>io.dekorate</groupId>
    <artifactId>openshift-spring-starter</artifactId>
    <version>${dekorate.version}</version>
   </dependency>
    ...
  </dependencies>
...
<project>
```

2. 在部署到 OpenShift 时，配置应用程序启动的默认副本数：

   a. 将 **@OpenshiftApplication** 注解添加到应用程序的主源文件中，并将副本数设置为 2。在构建和部署应用程序时，它会自动从运行的主应用程序容器的 2 个副本开始：

```
package org.acme;

import io.dekorate.openshift.annotation.OpenshiftApplication;

// include the parameter for the number of replicas to
@OpenshiftApplication(replicas=2)
@SpringBootApplication
public class Application {
}
```

   b. 或者，在应用程序的 **application.properties** 文件中设置 **dekorate.openshift.replicas=2** 属性。

   /src/main/resources/application.properties

```
dekorate.openshift.replicas=2
```

3. 打包应用程序：

```
mvn clean package
```

4. 导航到 **target/classes/META-INF/dekorate** 查看由 Dekorate 生成的清单。部署配置 YAML 模板中的副本数量设置为 2：

```
...
spec:
  replicas: 2
  selector:
    matchLabels:
      app: acme
...
```

其他资源

- OpenShift 的 Dekorate 配置属性 概述。

1.4. 在 SPRING BOOT 应用程序中使用无注解配置

通过从 **application.properties** 和 **application.yml** 文件提取配置属性，使用 Dekorate 为 Spring Boot application 项目生成 OpenShift 资源配置文件。这个方法不需要您注解应用程序源，因为 Dekorate 可以从 Spring Boot 和属性文件中的配置参数获取所需的元数据。Annontationless configuration 是 Spring Boot 和 Dekorate 之间的丰富框架集成的功能。

先决条件

- 基于 Maven 的应用项目配置为使用 Spring Boot 和 Dekorate。
- 应用项目中至少有一个类标注为 **@SpringBootApplication** 注释。
- Java JDK 8 或 JDK 11 已安装。
- 已安装 Maven。

流程

1. 在应用程序的 **pom.xml** 文件中添加以下依赖项：

```
<project>
  ...
  <dependencies>
    ...
    <!-- The OpenShift Spring Starter automatically adds "io.dekorate:openshift-annotations"
as a transitive dependency -->
    <dependency>
      <groupId>io.dekorate</groupId>
      <artifactId>openshfit-spring-starter</artifactId>
      <version>${dekorate.version}</version>
    </dependency>
    ...
  </dependencies>
...
<project>
```

2. 将 Dekorate 配置属性添加到项目中的 **application.properties** 或 **application.yml** 文件中。您不必在源文件中添加任何 Dekorate 属性注解。请注意，您仍然可以在源文件中使用注解，但如果您这样做，Dekorate 使用 **application.properties** 或 **application.yml** 文件中提供的参数覆盖注解中提供的参数。

3. 打包应用程序：

```
mvn clean package
```

在构建应用程序 Dekorate 时，会在应用程序项目中的以下资源中解析配置。配置资源会按照增加的优先级顺序解析。这意味着，如果同一配置参数有 2 个不同资源的不同值，则 Dekorate 会使用优先级列表上从较高资源获取的值。例如，如果源中的注解指定了参数值，但为 **application.yml** 中的同一参数指定一个不同的值，Dekorate 会使用它从 **application.yml** 获取的值。Dekorate 按照以下优先级顺序解析您的项目资源：

1. 注解
2. **application.properties**

3. **application.yaml**
4. **application.yml**

4. 导航到 **target/classes/META-INF/dekorate** 目录，其中包含生成的 **openshift.json** 和 **openshift.yml** 清单文件。

1.5. 使用 DEKORATE 自动执行 OPENSHIFT SOURCE-TO-IMAGE 构建

在使用 Maven 编译应用程序后，您可以使用 Dekorate 自动执行 OpenShift 容器镜像构建。

请注意，使用 Dekorate 自动触发 Source-to-image 构建的 功能作为技术预览提供。红帽不支持在生产环境中使用此功能。

先决条件

- 基于 Maven 的应用项目配置为使用 Spring Boot 和 Dekorate。
- **@SpringBootApplication** 注解添加到项目中的源文件中。
- Java JDK 8 或 JDK 11 已安装。
- 已安装 Maven。
- 已安装 **oc** 命令行工具。
- 使用 **oc** 命令行工具登录到 OpenShift 集群。

流程

1. 将 Dekorate OpenShift Spring Starter 作为依赖项添加到应用程序的 **pom.xml** 文件中。请注意，此模块作为传输依赖项包含在所有 Dekorate OpenShift Starters 中：

```
<project>
 ...
 <dependencies>
   ...
  <dependency>
   <groupId>io.dekorate</groupId>
   <artifactId>openshift-spring-starter</artifactId>
   <version>${dekorate.version}</version>
  </dependency>
  ...
 </dependencies>
...
<project>
```

2. 构建并部署您的应用。包含 **-Ddekorate.build=true** 属性，用于在 Maven 编译应用程序后执行容器镜像构建。请注意，自动执行 Source-to-image 构建的 功能作为技术预览提供。

```
$ mvn clean install -Ddekorate.build=true
```

在使用 Maven 编译应用程序后，您还可以从命令行手动执行 Source-to-image 构建：

```
# Process your application YAML template that is generated by Dekorate:
```

```
$ oc apply -f target/classes/META-INF/dekorate/openshift.yml
# Execute the Source-to-image build and deploy your application to the OpenShift cluster:
$ oc start-build example --from-dir=./target --follow
```

1.6. 在 OPENSHIFT 中使用 DEKORATE 和 SPRING BOOT

以下示例演示了如何进行：

1. 您可以在应用程序中使用 **openshift-spring-stater**。
2. Dekorate 可以自动识别应用程序的类型，并相应地配置 OpenShift 服务路由和探测。
3. 您可以将应用程序设置为在 Maven 编译应用程序后触发 Source-to-image 构建。
4. 先决条件
   - 基于 Maven 的应用项目配置为使用 Spring Boot 和 Dekorate。
   - **@SpringBootApplication** 注解添加到项目中的源文件中。
   - Java JDK 8 或 JDK 11 已安装。
   - 已安装 Maven。
   - 安装了 **oc** 命令行工具。
   - 使用 **oc** 命令行工具登录到 OpenShift 集群。

流程

1. 在应用程序项目的 **pom.xml** 文件中添加 Dekorate Spring Starter 作为依赖项。

   pom.xml

```
<project>
 ...
 <dependencies>
   ...
  <dependency>
   <groupId>io.dekorate</groupId>
   <artifactId>openshift-spring-starter</artifactId>
   <version>${dekorate.version}</version>
  </dependency>
  ...
 </dependencies>
...
<project>
```

2. 将 **@SpringBootApplication** 注释添加到您的 **Main.java** 类。这可让 source-to-image 构建在编译应用程序时启动：

   /src/main/java/io/dekorate/example/sbonopenshift/Main.java

```
package io.dekorate.example.sbonopenshift;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {

  public static void main(String[] args) {
    SpringApplication.run(Main.class, args);
  }

}
```

3. 在应用程序中添加 Rest 控制器：

/src/main/java/io/dekorate/example/sbonopenshift/Controller.java

```
package io.dekorate.example.sbonopenshift;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Controller {

  @RequestMapping("/")
  public String hello() {
    return "Hello world";
  }
}
```

Dekorate Spring starter 提供的 Spring 应用程序处理器会自动检测到 Rest 控制器，并将应用程序类型标识为 Web 应用。对于 Web 应用程序，Dekorate 会自动生成 OpenShift 应用程序模板并配置：

- 应用程序的 OpenShift Service 路由
- 在应用程序的路由上公开服务
- 配置存活度和就绪度探测设置

4. 构建和部署您的应用程序。包含 **-Ddekorate.deploy=true** 属性，以在 Maven 编译应用程序后自动执行 source-to-image 构建。

```
mvn clean install -Ddekorate.deploy=true
```

1.7. OPENSHIFT 的 DEKORATE 配置属性

下表中列出的属性设置 Dekorate 用来配置要部署到 OpenShift 的应用的值。Dekorate 使用这些属性中指定的值来填充为应用程序项目生成的 Deployment Configuration 和 application 资源文件。每个属性接受特定属性表中列出的数据类型值。如果没有为这些属性指定值，则某些属性有一个默认值，则 Dekorate 使用。您可以在应用程序项目的 **application.properties** 文件中设置这些属性。

表 1.1. OpenShift 的 Dekorate 应用属性

属性	数据类型	描述	默认值（如果适用）
dekorate.openshift.part-of	字符串	应用程序所属组件集合的名称。此属性的值用于应用程序包含的其他 Kubernetes 资源的名称，如 Deployment 配置和服务。	如果没有为此属性指定值，则 Dekorate 将使用您在应用程序的 Maven 项目中所用 **groupId** 的名称作为默认值。
dekorate.openshift.name	字符串	应用程序的名称。此属性的值用于应用程序包含的其他 Kubernetes 资源的名称，如 Deployment 配置和服务。	如果没有为此属性指定值，则 Dekorate 将使用应用程序 Maven 项目的 **artifactId** 的名称作为默认值。
dekorate.openshift.version	字符串	应用程序的版本。此属性的值用于应用程序包含的所有 Kubernetes 资源的名称，如 Deployment 配置和服务。	如果没有为此属性指定值，则 Dekorate 将使用您在包含应用程序的 Maven 项目中指定的版本作为默认值。
dekorate.openshift.init-containers	Container[]	指定您要在应用程序中使用的 init 容器	
dekorate.openshift.labels	Label[]	指定要添加到应用程序中的所有资源中的自定义标签	
dekorate.openshift.annotations	Annotation[]	指定要添加到应用程序中的所有资源的自定义注解	
dekorate.openshift.env-vars	Env[]	指定您要为应用程序创建的所有容器定义的环境变量	
dekorate.openshift.working-dir	字符串	指定应用程序容器的工作目录	
dekorate.openshift.command	String[]	指定您要在容器中使用的命令	
dekorate.openshift.arguments	String[]	指定您要在容器中使用的自定义命令行参数	
dekorate.openshift.replicas	int	指定在部署应用程序时您要创建的应用程序容器副本数	**1**

属性	数据类型	描述	默认值（如果适用）
dekorate.openshift.service-account	字符串	指定应用程序使用的 Service 帐户的名称	
dekorate.openshift.host	字符串	运行应用程序的主机节点的名称	
dekorate.openshift.ports	port[]	您提供的服务所公开的网络端口	
dekorate.openshift.service-type	ServiceType	为应用程序生成的服务类型	**ClusterIP**
dekorate.openshift.pvc-volumes	PersistentVolumeClaimVolume[]	要附加到应用程序所有容器的持久性卷声明	
dekorate.openshift.secret-volumes	SecretVolume[]	要附加到应用程序所有容器的 secret 卷	
dekorate.openshift.config-map-volumes	ConfigMapVolume[]	要附加到应用程序所有容器的 ConfigMap 卷	
dekorate.openshift.git-repo-volumes	GitRepoVolume[]	您要附加到应用程序所有容器的 Git 存储库卷	
dekorate.openshift.aws-elastic-block-store-volumes	AwsElasticBlockStoreVolume[]	您要附加到应用程序所有容器的 AWS Elastic Block Store 卷	
dekorate.openshift.azure-disk-volumes	AzureDiskVolume[]	要附加到应用程序所有容器的 Microsoft Azure 磁盘卷	
dekorate.openshift.azure-file-volumes	AzureFileVolume[]	您要附加到应用程序所有容器的 Azure 文件卷	
dekorate.openshift.mounts	Mount[]	将您要附加到应用程序的所有容器挂载	
dekorate.openshift.image-pull-policy	ImagePullPolicy	指定部署应用程序时希望的镜像拉取策略	**IfNotPresent**
dekorate.openshift.image-pull-secrets	String[]	指定部署应用程序时要使用的镜像 pull secret 策略	
dekorate.openshift.liveness-probe	probe	为您的应用程序容器设置存活度探测	

属性	数据类型	描述	默认值（如果适用）
dekorate.openshift.readiness-probe	probe	为您的应用程序容器设置就绪度探测	
dekorate.openshift.request-resources	ResourceRequirements	指定应用程序容器需要的资源量	
dekorate.openshift.limit-resources	ResourceRequirements	为应用程序容器设置资源限值	
dekorate.openshift.sidecars	Container[]	指定您要部署为 sidecar 的容器	
dekorate.openshift.expose	布尔值	设置是否在部署后为应用程序公开路由	**false**
dekorate.openshift.headless	布尔值	设置您是否希望生成的服务执行无头	**false**
dekorate.openshift.auto-deploy-enabled	布尔值	设置应用程序是否在生成部署 hook 时自动部署。在应用程序上设置此属性需要您在 **application.properties** 文件中硬编码其值。如果要避免硬编码其值，请不要设置此属性。使用 Maven 部署应用程序时使用 **-Ddekorate.deploy=true** 选项	**false**
dekorate.openshift.deployment-kind	字符串	要使用的部署资源的种类。支持的值包括 **DeploymentConfig**、**Deployment**、**StatefulSet**、**Job**、Job 和 **CronJob** 默认为第一个。	**DeploymentConfig**

1.8. SOURCE-TO-IMAGE 的 DEKORATE 配置属性

下表中列出的属性设置 Dekorate 用来配置 Source-to-Image (s2i)的值来为您的应用程序构建。您可以在应用程序项目的 **application.properties** 文件中设置这些属性。

表 1.2. S2i 的 Dekorate 配置属性

属性	数据类型	描述	默认值（如果适用）
dekorate.s2i.enabled	布尔值	为您的应用程序启用 s2i 构建 hook 生成	**true**
dekorate.s2i.registry	字符串	指定您要构建的镜像的 registry 名称	
dekorate.s2i.group	字符串	指定应用程序的组 ID。这个值将作为您构建的 docker 镜像中的用户名使用	
dekorate.s2i.name	字符串	指定应用程序的名称。这个值被用作您构建的镜像的名称。	
dekorate.s2i.version	字符串	应用程序的版本。这个值被用作您构建的镜像标签。	
dekorate.s2i.image	字符串	指定对您要构建的镜像的完整引用。设置后，此属性覆盖 组、**name** 和 **version** 属性的值。	
dekorate.s2i.docker-file	字符串	指定从应用程序项目的根目录到 Dockerfile 的相对路径	**Docker**
dekorate.s2i.builder-image	字符串	指定您要使用的 S2i 构建器镜像的名称	**registry.access.redhat.com/ubi8/openjdk-8:1.3**
dekorate.s2i.build-env-vars	Env[]	为 s2i 构建设置环境变量	
dekorate.s2i.auto-push-enabled	布尔值	为 **true** 时，s2i 会在构建镜像时自动将镜像推送到指定的 registry。	**false**
dekorate.s2i.auto-build-enabled	布尔值	为 **true** 时，s2i 会在编译应用程序时自动注册构建 hook	**false**

属性	数据类型	描述	默认值（如果适用）
dekorate.s2i.auto-deploy-enabled	布尔值	为 **true** 时，应用程序会在生成部署 hook 时自动部署。在应用程序上设置此属性需要您在 **application.properties** 文件中硬编码其值。如果要避免硬编码其值，请不要设置此属性。使用 Maven 部署应用程序时使用 **-Ddekorate.deploy=true** 选项	**false**

附录 A. SOURCE-TO-IMAGE (S2I)构建过程

Source-to-Image (S2I)是一种构建工具，用于通过应用源从在线 SCM 存储库生成可重复生成的 Docker 格式容器镜像。借助 S2I 构建，您可以轻松地将最新版本的应用交付至生产中，构建时间更短、资源和网络使用、提高安全性以及许多其他优点。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。

您必须为 S2I 进程提供三个元素，以编译最终容器镜像：

- 托管在在线 SCM 存储库中的应用源，如 GitHub。
- S2I Builder 镜像，作为编译镜像的基础，并提供运行应用程序的生态系统。
- 另外，您还可以提供 S2I 脚本使用的环境变量和参数。

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

附录 B. 其他 SPRING BOOT 资源

- OpenShift 架构概述
- Spring Cloud Kubernetes
- Spring Boot 项目
- Spring Framework 项目
- OpenShift Spring Boot Lab microservicess

附录 C. 应用程序开发资源

如需有关使用 OpenShift 进行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

附录 D. 精度级别

每个可用示例都教授需要某些最小知识的概念。这个要求因示例而异。最低要求和概念以多个原则进行组织。除了此处描述的级别外，还需要特定于每个示例的其他信息。

Base Base

根据基础精通的原则，一般不需要预先了解相关主题的信息；它们可提供关键元素、概念和术语的一般意识和演示。除非在示例描述中直接提及，否则没有特殊要求。

Advanced

使用高级示例时，假设是除了 Kubernetes 和 OpenShift 外，您熟悉示例主题区域的通用概念和术语。您还必须能够自行执行基本任务，例如配置服务和应用程序，或者管理网络。如果示例需要某个服务，但配置该服务不在示例范围内，假设您具有正确配置它的知识，并且文档中仅描述服务生成的状态。

专家

专家示例需要最高级别的主题知识。您应当根据基于功能的文档和手册执行许多任务，并且文档在最复杂的场景中是旨在执行的任务。