



Red Hat support for Spring Boot 2.3

Spring Boot 開発者の Dekorate ガイド

Dekorate を使用して、OpenShift とスタンドアロンの RHEL へのデプロイメント用に Spring Boot アプリケーションを自動的に設定します。

Red Hat support for Spring Boot 2.3 Spring Boot 開発者の Dekorate ガイド

Dekorate を使用して、OpenShift とスタンドアロンの RHEL へのデプロイメント用に Spring Boot アプリケーションを自動的に設定します。

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2021 | You need to change the HOLDER entity in the en-US/Dekorater_Guide_for_Spring_Boot_Developers.ent file |.

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.

概要

本ガイドでは、Dekorater を使用してコードからリソースファイルを自動的に生成し、複数の環境へのデプロイメントのために Spring Boot アプリケーションを準備する方法を説明します。

目次

前書き	3
RED HAT ドキュメントへのフィードバック	4
第1章 SPRING BOOT を使用するようにアプリケーションを設定	5
1.1. 前提条件	5
1.2. SPRING BOOT BOM を使用した依存関係バージョンの管理	5
1.3. SPRING BOOT BOM を使用したアプリケーションの親 BOM として使用	7
1.4. 関連情報	8
第2章 SPRING BOOT アプリケーションでの DEKORATE の使用	9
2.1. DEKORATE の概要	9
2.1.1. 関連情報	9
2.2. DEKORATE を使用するためのアプリケーションプロジェクトの設定	9
2.3. DEKORATE を使用したアプリケーション設定のカスタマイズ	10
2.4. SPRING BOOT アプリケーションでのアノテーションレス設定の使用	12
2.5. DEKORATE を使用した OPENSIFT SOURCE-TO-IMAGE ビルドの自動実行	13
2.6. OPENSIFT での SPRING BOOT での DEKORATE の使用	14
2.7. OPENSIFT の DEKORATE 設定プロパティ	16
2.8. SOURCE-TO-IMAGE の DEKORATE 設定プロパティ	20
付録A SOURCE-TO-IMAGE (S2I) ビルドプロセス	23
付録B 追加の SPRING BOOT リソース	24
付録C アプリケーション開発リソース	25
付録D 習熟度レベル	26
Foundational	26
Advanced	26
Expert	26
付録E 用語集	27
E.1. 製品およびプロジェクト名	27
E.2. DEVELOPER LAUNCHER に固有の用語	27

前書き

Dekorate を使用して Spring Boot アプリケーションのコードを処理し、アプリケーションのマニフェストファイルを自動的に生成し、デプロイメントを OpenShift に設定します。

RED HAT ドキュメントへのフィードバック

弊社のドキュメントに関するご意見やご感想をお寄せください。フィードバックをお寄せいただくには、ドキュメントのテキストを強調表示し、コメントを追加できます。

本セクションでは、フィードバックの送信方法を説明します。

前提条件

- Red Hat カスタマーポータルにログインしている。
- Red Hat カスタマーポータルで、**マルチページ HTML** 形式でドキュメントを表示します。

手順

フィードバックを提供するには、以下の手順を実施します。

1. ドキュメントの右上隅にある **フィードバック** ボタンをクリックして、既存のフィードバックを確認します。



注記

フィードバック機能は、**マルチページ HTML** 形式でのみ有効です。

2. フィードバックを提供するドキュメントのセクションを強調表示します。
3. ハイライトされたテキスト近くに表示される **Add Feedback** ポップアップをクリックします。ページの右側のフィードバックセクションにテキストボックスが表示されます。
4. テキストボックスにフィードバックを入力し、**Submit** をクリックします。ドキュメントに関する問題が作成されます。
5. 問題を表示するには、フィードバックビューで問題トラッカーリンクをクリックします。

第1章 SPRING BOOT を使用するようにアプリケーションを設定

Red Hat ビルドの Spring Boot で提供される依存関係を使用するようにアプリケーションを設定します。BOM を使用して依存関係を管理することで、アプリケーションが Red Hat が提供するこれらの依存関係の製品バージョンを常に使用するようにしてください。アプリケーションのルートディレクトリにある **pom.xml** ファイルの Spring Boot BOM (Bill of Materials) アーティファクトを参照します。アプリケーションプロジェクトで BOM を使用することもできます。2 種類の方法を使用できます。

- **pom.xml** の `<dependencyManagement>` セクションの [依存関係として](#)。BOM を依存関係として使用する場合、プロジェクトは BOM の `<dependencyManagement>` セクションからすべての Spring Boot 依存関係のバージョン設定を継承します。
- **pom.xml** の `<parent>` セクションで [親 BOM として](#)。BOM を親として使用する場合、プロジェクトの **pom.xml** は、親 BOM から以下の設定値を継承します。
 - `<dependencyManagement>` セクションのすべての Spring Boot 依存関係のバージョン
 - `<pluginManagement>` セクションのバージョンプラグイン
 - `<repositories>` セクションのリポジトリの URL および名前
 - `<pluginRepositories>` セクションに Spring Boot プラグインが含まれるリポジトリの URL および名前

1.1. 前提条件

- **pom.xml** ファイルを使用して設定する Maven ベースのアプリケーションプロジェクト。
- [Red Hat JBoss Middleware General Availability Maven リポジトリへのアクセス](#)。

1.2. SPRING BOOT BOM を使用した依存関係バージョンの管理

製品 BOM を使用して、アプリケーションプロジェクトで Spring Boot 製品依存関係のバージョンを管理します。

手順

1. **dev.snowdrop:snowdrop-dependencies** アーティファクトをプロジェクトの **pom.xml** の `<dependencyManagement>` セクションに追加し、`<type>` 属性値および `<scope>` 属性値を指定します。

```
<project>
...
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>dev.snowdrop</groupId>
      <artifactId>snowdrop-dependencies</artifactId>
      <version>2.3.10.Final-redhat-00004</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
```

```
</dependencyManagement>
```

```
...
```

```
</project>
```

2. 以下のプロパティを追加して、使用している Spring Boot Maven プラグインのバージョンを追跡します。

```
<project>
```

```
...
```

```
<properties>
```

```
<spring-boot-maven-plugin.version>2.3.10.RELEASE</spring-boot-maven-plugin.version>
```

```
</properties>
```

```
...
```

```
</project>
```

3. BOM およびサポートされる Spring Boot Starters および Spring Boot Maven プラグインを含むリポジトリの名前と URL を指定します。

```
<!-- Specify the repositories containing Spring Boot 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>
```

4. **spring-boot-maven-plugin** を Maven がアプリケーションのパッケージ化に使用するプラグインとして追加します。

```
<project>
```

```
...
```

```
<build>
```

```
...
```

```
<plugins>
```

```
...
```

```
<plugin>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-maven-plugin</artifactId>
```

```
<version>${spring-boot-maven-plugin.version}</version>
```

```
<executions>
```

```
<execution>
```

```
<goals>
```

```
<goal>repackage</goal>
```

```
</goals>
```

```

        </execution>
      </executions>
      <configuration>
        <redeploy>true</redeploy>
      </configuration>
    </plugin>
    ...
  </plugins>
  ...
</build>
...
</project>

```

1.3. SPRING BOOT BOM を使用したアプリケーションの親 BOM として使用

以下を自動的に管理します。

- 製品依存関係のバージョン
- Spring Boot Maven プラグインのバージョン
- 製品アーティファクトおよびプラグインが含まれる Maven リポジトリの設定

製品の Spring Boot BOM をプロジェクトの親 BOM として含めて、アプリケーションプロジェクトで使用する内容。この方法では、BOM をアプリケーションの依存関係として使用する代替方法を提供します。

手順

1. **dev.snowdrop:snowdrop-dependencies** アーティファクトを **pom.xml** の **<parent>** セクションに追加します。

```

<project>
  ...
  <parent>
    <groupId>dev.snowdrop</groupId>
    <artifactId>snowdrop-dependencies</artifactId>
    <version>2.3.10.Final-redhat-00004</version>
  </parent>
  ...
</project>

```

2. **spring-boot-maven-plugin** を Maven がアプリケーションを **pom.xml** の **<build>** セクションにパッケージ化するために使用するプラグインとして追加します。プラグインバージョンは、親 BOM により自動的に管理されます。

```

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

```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<executions>
  <execution>
    <goals>
      <goal>repackage</goal>
    </goals>
  </execution>
</executions>
<configuration>
  <redeploy>true</redeploy>
</configuration>
</plugin>
...
</plugins>
...
</build>
...
</project>
```

1.4. 関連情報

- Spring Boot アプリケーションのパッケージ化に関する詳細は、[Spring Boot Maven Plugin](#) のドキュメントを参照してください。

第2章 SPRING BOOT アプリケーションでの DEKORATE の使用

Dekorate を使用して、アプリケーションマニフェストファイルを自動的に生成し、OpenShift へのデプロイメントのためにアプリケーションを設定します。

2.1. DEKORATE の概要

Dekorate は、コンパイル時のアノテーションプロセッサおよび Red Hat ビルドの Spring Boot で提供されるアプリケーションリソースジェネレーターのコレクションです。これは、アプリケーションをビルドして設定プロパティを抽出する際に、コードでアノテーションを解析することで機能します。次に、Dekorate は、展開したプロパティの値を使用して、アプリケーションを Kubernetes または OpenShift クラスターにデプロイするために使用できるアプリケーション設定リソースを生成します。

開発者は、コードにアノテーションを付けた後に Dekorate を使用してアプリケーションのビルド時にアプリケーションマニフェストを自動的に生成できます。これにより、アプリケーションをデプロイするためのリソースファイルを手動で作成する必要がなくなります。アプリケーションが、Spring Boot などのリッチアプリケーションのランタイムフレームワークに基づいていると、Spring Boot、Dekorate はフレームワークと直接統合でき、フレームワークによって提供される API から設定パラメーターを抽出できるため、コードにアノテーションを付ける必要がなくなります。Dekorate は、以下を使用してアプリケーションを自動的に設定できます。

- マニフェストファイルの設定に使用される値およびメタデータを取得するためにアプリケーションコードで Dekorate 固有のアノテーションを解析します。
- **application.properties**、**application.yaml** などの設定リソースから情報の抽出
- リッチアプリケーションフレームワークから必要なメタデータを取得し、**application.properties** ファイルまたは **application.yml** ファイルから設定値を展開します。

アプリケーションのリソース定義を生成するほか、Dekorate はフックを提供し、OpenShift クラスターの Dekorate でアプリケーションをビルドおよびデプロイできます。また、アプリケーションを作成する言語とは独立して機能し、幅広いビルドシステムで使用できます。Dekorate は、Maven BOM として配布されるライブラリーのセットで構成されます。ライブラリーをアプリケーションプロジェクトの依存関係として追加し、アプリケーションと Dekorate を使用することができます。

Red Hat は Dekorate を使用して、Spring Boot をベースとする Java アプリケーションを OpenShift Container Platform にデプロイするのに使用できるリソースファイルとフックの生成をサポートします。

2.1.1. 関連情報

- [OpenShift の Dekorate 設定プロパティ](#) の参照。
- [Source-to-Image の Dekorate 設定プロパティ](#) への参照。
- コミュニティドキュメントの [すべての Dekorate Configuration プロパティ](#) への参照。

2.2. DEKORATE を使用するためのアプリケーションプロジェクトの設定

Dekorate BOM および OpenShift Annotations Starter をアプリケーションプロジェクトの **pom.xml** ファイルに追加します。基本的なアノテーションをソースファイルに追加し、Maven でアプリケーションをパッケージ化してアプリケーションマニフェストを生成します。

前提条件

- [Spring Boot](#) を使用するよう設定された Maven ベースの Java アプリケーションプロジェクト
- Java JDK 8 または JDK 11 がインストールされている。
- Maven がインストールされている。

手順

1. Dekorate OpenShift Spring Starter をアプリケーションの **pom.xml** ファイルに追加して、Dekorate がアプリケーションのソースコードとリソースファイルを処理できるようにします。

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

2. **@SpringBootApplication** アノテーションをアプリケーションプロジェクトのメインクラスファイルに追加します。

```
package org.acme;

@SpringBootApplication
public class Application {
}
```

3. アプリケーションをパッケージ化して、アプリケーションコードおよびリソースファイルを Dekorate で処理します。

```
mvn clean package
```

4. 生成された OpenShift マニフェストが含まれる **target/classes/META-INF/dekorate** ディレクトリに移動します。

2.3. DEKORATE を使用したアプリケーション設定のカスタマイズ

Dekorate を使用して、OpenShift でのデプロイメント用にアプリケーションの設定をカスタマイズします。

- アプリケーションのソースのアノテーションで構成パラメーターを指定
- **application.properties** ファイルでのプロパティの設定

以下の例は、OpenShift へのデプロイ時に 2 つのレプリカで始まるようにアプリケーションを設定する方法を示しています。

前提条件

- [Spring Boot](#) および [Dekorater](#) を使用するよう設定された Maven ベースの Java アプリケーションプロジェクト
- Java JDK 8 または JDK 11 がインストールされている。
- Maven がインストールされている。

手順

1. Dekorater OpenShift Annotations モジュールを、アプリケーションの **pom.xml** ファイルの依存関係として追加します。

```
<project>
...
<dependencies>
...
<dependency>
  <groupId>io.dekorater</groupId>
  <artifactId>openshift-spring-starter</artifactId>
</dependency>
...
</dependencies>
...
</project>
```

2. OpenShift へのデプロイ時に、アプリケーションが開始するデフォルトのレプリカ数を設定します。
 - a. **@OpenshiftApplication** アノテーションをアプリケーションのメインソースファイルに追加し、レプリカ数を 2 に設定します。アプリケーションのビルドとデプロイ時に、実行中のメインのアプリケーションコンテナの 2 つのレプリカで自動的に起動します。

```
package org.acme;

import io.dekorater.openshift.annotation.OpenshiftApplication;

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

- b. または、アプリケーションの **application.properties** ファイルに **dekorater.openshift.replicas=2** プロパティを設定します。

/src/main/resources/application.properties

```
dekorater.openshift.replicas=2
```

3. アプリケーションをパッケージ化します。

```
mvn clean package
```

4. **target/classes/META-INF/dekorate** に移動し、Dekorate が生成したマニフェストを表示します。デプロイメント設定 YAML テンプレートのレプリカ数は 2 に設定されます。

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

関連情報

- [OpenShift の Dekorate 設定プロパティ](#) の概要

2.4. SPRING BOOT アプリケーションでのアノテーションレス設定の使用

Dekorate を使用して、**application.properties** ファイルおよび **application.yml** ファイルから **dekorate** 設定プロパティを抽出して、Spring Boot アプリケーションプロジェクトの OpenShift リソース設定ファイルを生成します。Dekorate は、Spring Boot から必要なメタデータとプロパティファイルから設定パラメーターを取得できるため、このメソッドにはアプリケーションソースにアノテーションを付ける必要はありません。アノテーションのない設定は、Spring Boot と Dekorate の間のリッチフレームワーク統合機能です。

前提条件

- [Spring Boot](#) および [Dekorate](#) を使用するよう設定された Maven ベースのアプリケーションプロジェクト
- **@SpringBootApplication** アノテーションが付けられたアプリケーションプロジェクトの 1 つ以上のクラス。
- Java JDK 8 または JDK 11 がインストールされている。
- Maven がインストールされている。

手順

1. アプリケーションの **pom.xml** ファイルに以下の依存関係を追加します。

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



```
</dependencies>
```

```
...
```

```
<project>
```

2. Dekorater 設定プロパティをプロジェクトの **application.properties** ファイルまたは **application.yml** ファイルに追加します。Dekorater プロパティアノテーションをソースファイルに追加する必要はありません。ソースファイルのアノテーションを使用できますが、これを行う場合は、アノテーションで提供されるパラメーターを **application.properties** ファイルまたは **application.yml** ファイルにあるパラメーターで上書きします。
3. アプリケーションをパッケージ化します。

```
mvn clean package
```

アプリケーションの Dekorater をビルドすると、アプリケーションプロジェクト内の以下のリソースの設定が解析されます。設定リソースは優先順に解析されます。つまり、異なるタイプの 2 つの異なるリソースが同じ設定パラメーターに異なる値を持っている場合、Dekorater は優先度のリストにあるリソースから取得した値を使用します。たとえば、ソースのアノテーションがパラメーター値を指定するものの、**application.yml** の同じパラメーターに異なる値が指定されている場合、Dekorater は **application.yml** から取得する値を使用します。Dekorater は、以下の優先順位でプロジェクトリソースを解析します。

1. アノテーション
 2. **application.properties**
 3. **application.yaml**
 4. **application.yml**
4. 生成された **openshift.json** および **openshift.yml** マニフェストファイルが含まれる **target/classes/META-INF/dekorater** ディレクトリーに移動します。

2.5. DEKORATE を使用した OPENSIFT SOURCE-TO-IMAGE ビルドの自動実行

Maven でアプリケーションをコンパイルした後に、Dekorater を使用して OpenShift コンテナイメージビルドを自動的に実行できます。

Dekorater を使用して Source-to-Image ビルドを自動的にトリガーする機能は [テクノロジーレビュー](#) として利用できます。Red Hat は、実稼働環境でのこの機能の使用に対するサポートを提供していません。

前提条件

- [Spring Boot](#) および [Dekorater](#) を使用するよう設定された Maven ベースのアプリケーションプロジェクト
- `@SpringBootApplication` アノテーションがプロジェクトのソースファイルに追加されている。
- Java JDK 8 または JDK 11 がインストールされている。
- Maven がインストールされている。
- **oc** コマンドラインツールがインストールされている

- **oc** コマンドラインツールを使用して OpenShift クラスターにログインしている。

手順

1. Dekorate OpenShift Spring Starter を依存関係としてアプリケーションの **pom.xml** ファイルに追加します。このモジュールは、すべての Dekorate OpenShift Starters に推移的な依存関係として含まれていることに注意してください。

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

2. アプリケーションをビルドしてデプロイします。Maven がアプリケーションをコンパイルした後にコンテナイメージビルドを実行するための **-Ddekorate.build=true** プロパティを含めます。Source-to-Image ビルドを自動的に実行する機能は [テクノロジープレビュー](#) として提供されることに注意してください。

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

Maven でアプリケーションをコンパイルした後に、コマンドラインから Source-to-image ビルドを手動で実行することもできます。

```
# 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
```

2.6. OPENSIFT での SPRING BOOT での DEKORATE の使用

以下の例は、方法を示しています。

1. アプリケーションで **openshift-spring-starter** を使用できます。
2. Dekorate は、アプリケーションのタイプを自動的に識別し、OpenShift サービスルートおよびプローブを随時設定できます。
3. Maven がアプリケーションをコンパイルした後に、source-to-image ビルドをトリガーするようにアプリケーションを設定できます。
4. 前提条件
 - [Spring Boot](#) および [Dekorate](#) を使用するよう設定された Maven ベースのアプリケーションプロジェクト
 - **@SpringBootApplication** アノテーションがプロジェクトのソースファイルに追加されました。

- Java JDK 8 または JDK 11 がインストールされている。
- Maven がインストールされている。
- **oc** コマンドラインツールがインストールされている
- **oc** コマンドラインツールを使用して OpenShift クラスターにログインしている。

手順

1. Dekorate Spring Starter は、アプリケーションプロジェクトの **pom.xml** ファイルの依存関係として追加します。

pom.xml

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

2. **@SpringBootApplication** アノテーションを **Main.java** クラスに追加します。これにより、アプリケーションをコンパイルする際に source-to-image ビルドを開始できます。

/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. Rest コントローラーをアプリケーションに追加します。

/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 スターターによって提供される Spring アプリケーションプロセッサは Rest コントローラーを自動的に検出し、アプリケーションタイプを Web アプリケーションとして識別します。Web アプリケーションの場合、Dekorate は OpenShift アプリケーションテンプレートを自動的に生成し、以下を設定します。

- アプリケーションの OpenShift Service ルート
 - アプリケーションのルートでサービスを公開します。
 - Liveness および Readiness プローブの設定
4. アプリケーションをビルドしてデプロイします。Maven がアプリケーションをコンパイルした後に source-to-image ビルドを自動的に実行するには、**-Ddekorate.deploy=true** プロパティを含めます。

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

2.7. OPENSIFT の DEKORATE 設定プロパティ

以下の表に記載されているプロパティは、Dekorate が OpenShift へのデプロイメントに使用する値を設定します。Dekorate は、これらのプロパティで指定した値を使用して、アプリケーションプロジェクトに生成された Deployment Configuration およびアプリケーションリソースファイルを設定します。各プロパティは、特定のプロパティのテーブルに記載されているデータ型の値を受け入れます。一部のプロパティには、これらの属性に値を指定しない場合に Dekorate が使用するデフォルト値があります。これらのプロパティは、アプリケーションプロジェクトの **application.properties** ファイルで設定できます。

表2.1 OpenShift の Dekorate アプリケーションプロパティ

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.openshift.port-of	文字列	アプリケーションが属するコンポーネントのコレクション名。このプロパティの値は、デプロイメント設定やサービスなど、アプリケーションに含まれる他の Kubernetes リソースの名前で使用されます。	このプロパティの値を指定しない場合、Dekorate はアプリケーションの Maven プロジェクトで使用する groupId の名前をデフォルト値として使用します。

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.openshift.name	文字列	アプリケーションの名前 このプロパティの値は、デプロイメント設定やサービスなど、アプリケーションに含まれる他の Kubernetes リソースの名前で使用されます。	このプロパティの値を指定しない場合、Dokorate は、アプリケーションの Maven プロジェクトに使用する artifactId の名前をデフォルト値として使用します。
dekorate.openshift.version	文字列	アプリケーションのバージョン。 このプロパティの値は、デプロイメント設定やサービスなど、アプリケーションに含まれるすべての Kubernetes リソースの名前で使用されます。	このプロパティの値を指定しない場合、Dokorate はアプリケーションの Maven プロジェクトで指定する version をデフォルト値として使用します。
dekorate.openshift.init-containers	Container[]	アプリケーションで使用する init コンテナを指定します。	
dekorate.openshift.labels	Label[]	アプリケーションのすべてのリソースに追加するカスタムラベルを指定します。	
dekorate.openshift.annotations	Annotation[]	アプリケーションのすべてのリソースに追加するカスタムアノテーションを指定します。	
dekorate.openshift.env-vars	Env[]	アプリケーション用に作成したすべてのコンテナに定義する環境変数を指定します。	
dekorate.openshift.working-dir	文字列	アプリケーションコンテナの作業ディレクトリを指定します。	
dekorate.openshift.command	String[]	コンテナで使用するコマンドを指定します。	
dekorate.openshift.arguments	String[]	コンテナで使用するカスタムコマンドライン引数を指定します。	

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.openshift.replicas	int	アプリケーションのデプロイ時に作成するアプリケーションコンテナのレプリカ数を指定します。	1
dekorate.openshift.service-account	文字列	アプリケーションが使用するサービスアカウントの名前を指定します。	
dekorate.openshift.host	文字列	アプリケーションが実行しているホストノードの名前	
dekorate.openshift.ports	Port[]	提供するサービスが公開されるネットワークポート	
dekorate.openshift.service-type	ServiceType	アプリケーション用に生成されるサービスのタイプ	ClusterIP
dekorate.openshift.pvc-volumes	PersistentVolumeClaim Volume[]	アプリケーションのすべてのコンテナに割り当てる永続ボリューム要求 (PVC)	
dekorate.openshift.secret-volumes	SecretVolume[]	アプリケーションのすべてのコンテナに割り当てるシークレットボリューム	
dekorate.openshift.config-map-volumes	ConfigMapVolume[]	アプリケーションのすべてのコンテナに割り当てる ConfigMap ボリューム	
dekorate.openshift.git-repo-volumes	GitRepoVolume[]	アプリケーションのすべてのコンテナに割り当てる Git リポジトリボリューム	
dekorate.openshift.aws-elastic-block-store-volumes	AwsElasticBlockStoreVolume[]	アプリケーションのすべてのコンテナに割り当てる AWS Elastic Block Store ボリューム	

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.openshift.azure-disk-volumes	AzureDiskVolume[]	アプリケーションのすべてのコンテナに割り当てる Microsoft Azure ディスクボリューム	
dekorate.openshift.azure-file-volumes	AzureFileVolume[]	アプリケーションのすべてのコンテナに割り当てる Azure ファイルボリューム	
dekorate.openshift.mounts	Mount[]	アプリケーションのすべてのコンテナに割り当てるマウント	
dekorate.openshift.image-pull-policy	ImagePullPolicy	アプリケーションのデプロイ時に使用するイメージプルポリシーを指定します。	IfNotPresent
dekorate.openshift.image-pull-secrets	String[]	アプリケーションのデプロイ時に使用するイメージプルシークレットポリシーを指定します。	
dekorate.openshift.liveness-probe	プローブ	アプリケーションコンテナの Liveness プローブの設定	
dekorate.openshift.readiness-probe	プローブ	アプリケーションコンテナの Readiness プローブの設定	
dekorate.openshift.request-resources	ResourceRequirements	アプリケーションコンテナが要求するリソースの量の指定	
dekorate.openshift.limit-resources	ResourceRequirements	アプリケーションコンテナのリソース制限を設定します。	
dekorate.openshift.sidecars	Container[]	サイドカーコンテナとしてデプロイするコンテナの指定	
dekorate.openshift.expose	boolean	デプロイ後にアプリケーションの Route を公開するかどうかを設定します。	false

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.openshift.headless	boolean	生成するサービスがヘッドレスを実行するかどうかを設定します。	false
dekorate.openshift.autom-deploy-enabled	boolean	デプロイフックを生成する際にアプリケーションが自動的にデプロイされるかどうかを設定します。このプロパティをアプリケーションで設定するには、 application.properties ファイルで値をハードコーディングする必要があります。値をハードコーディングしないようにする場合、このプロパティは設定しないでください。代わりに、Maven を使用してアプリケーションをデプロイする場合に - Ddekorate.deploy=true オプションを使用します。	false

2.8. SOURCE-TO-IMAGE の DEKORATE 設定プロパティ

以下の表に記載されているプロパティは、Depkorate が Source-to-Image (s2i) を設定してアプリケーション用にビルドするために使用する値を設定します。これらのプロパティは、アプリケーションプロジェクトの **application.properties** ファイルで設定できます。

表2.2 S2i の Dekorate 設定プロパティ

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.s2i.enabled	boolean	アプリケーション用の s2i ビルドフックの生成を有効にします。	true
dekorate.s2i.registry	文字列	ビルドするイメージのレジストリー名を指定します。	

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.s2i.group	文字列	アプリケーションのグループ ID を指定します。この値は、構築する docker イメージのユーザー名として使用されます。	
dekorate.s2i.name	文字列	アプリケーションの名前を指定します。この値は、ビルドするイメージの名前として使用されます。	
dekorate.s2i.version	文字列	アプリケーションのバージョン。この値は、ビルドするイメージのタグとして使用されます。	
dekorate.s2i.image	文字列	ビルドするイメージの完全な参照を指定します。このプロパティを設定すると、 group 、 name 、および version プロパティの値が上書きされます。	
dekorate.s2i.docker-file	文字列	アプリケーションプロジェクトのルートディレクトリからの Dockerfile への相対パスを指定します。	Dockerfile
dekorate.s2i.builder-image	文字列	使用する S2i ビルダーイメージの名前を指定します。	fabric8/s2i-java:2.3
dekorate.s2i.build-env-vars	Env[]	s2i ビルドの環境変数を設定します。	
dekorate.s2i.auto-push-enabled	boolean	true の場合、s2i はイメージのビルド時に、指定したレジストリーにイメージを自動的にプッシュします。	false

プロパティ	データタイプ	詳細	デフォルト値 (該当する場合)
dekorate.s2i.auto-build-enabled	boolean	true の場合、s2i はアプリケーションがコンパイルされるとビルドフックを自動的に登録します。	false
dekorate.s2i.auto-deploy-enabled	boolean	true の場合、デプロイフックを生成する際にアプリケーションが自動的にデプロイされます。このプロパティをアプリケーションで設定するには、 application.properties ファイルで値をハードコーディングする必要があります。値をハードコーディングしないようにする場合、このプロパティは設定しないでください。代わりに、Maven を使用してアプリケーションをデプロイする場合に - Ddekorate.deploy=true オプションを使用します。	false

付録A SOURCE-TO-IMAGE (S2I) ビルドプロセス

[Source-to-Image](#) (S2I) は、アプリケーションソースのあるオンライン SCM リポジトリから再現可能な Docker 形式のコンテナイメージを生成するビルドツールです。S2I ビルドを使用すると、ビルド時間を短縮し、リソースおよびネットワークの使用量を減らし、セキュリティを改善し、その他の多くの利点を使用して、アプリケーションの最新バージョンを実稼働に簡単に配信できます。OpenShift は、複数の [ビルドストラテジーおよび入力ソース](#) をサポートします。

詳細は、OpenShift Container Platform ドキュメントの「[Source-to-Image \(S2I\) ビルド](#)」の章を参照してください。

最終的なコンテナイメージをアセンブルするには、S2I プロセスに 3 つの要素を指定する必要があります。

- GitHub などのオンライン SCM リポジトリでホストされるアプリケーションソース。
- S2I Builder イメージ。アセンブルされたイメージの基盤となり、アプリケーションを実行しているエコシステムを提供します。
- 必要に応じて、[S2I スクリプト](#) によって使用される環境変数およびパラメーターを指定することもできます。

このプロセスは、S2I スクリプトで指定された指示に従ってアプリケーションソースおよび依存関係を Builder イメージに挿入し、アセンブルされたアプリケーションを実行する Docker 形式のコンテナイメージを生成します。詳細は、OpenShift Container Platform ドキュメントの「[S2I build requirements](#)」、「[build options](#)」および「[how builds work](#)」を参照してください。

付録B 追加の **SPRING BOOT** リソース

- [OpenShift Architecture Overview](#)
- [Spring Boot Microservices On Red Hat OpenShift Container Platform 3](#)
- [Spring Cloud Kubernetes](#)
- [Spring Boot プロジェクト](#)
- [Spring Framework プロジェクト](#)
- [OpenShift Spring Boot Lab Microservices](#)
- [Fabric8 を使用した Spring Boot アプリケーションの作成](#)
- [Fabric8 Maven プラグイン](#)

付録C アプリケーション開発リソース

OpenShift でのアプリケーション開発に関する詳細は、以下を参照してください。

- [OpenShift インタラクティブラーニングポータル](#)

ネットワークの負荷を削減し、アプリケーションのビルド時間を短縮するには、Minishift または CDK に Maven の Nexus ミラーを設定します。

- [Maven 用の Nexus ミラーリングの設定](#)

付録D 習熟度レベル

利用可能な各例は、特定の最小知識を必要とする概念について言及しています。この要件は例によって異なります。最小要件と概念は、いくつかの上達度レベルで構成されています。ここで説明するレベルの他に、各例に固有の追加情報が必要になる場合があります。

Foundational

Foundational と評価された例は、通常、サブジェクトに関する事前の知識を必要としません。それらは、重要な要素、概念、および用語の一般的な認識およびデモンストレーションを提供します。この例の説明に直接記載されているものを除き、特別な要件はありません。

Advanced

Advanced サンプルを使用する場合は、Kubernetes および OpenShift に加えて、例のサブジェクトエリアの一般的な概念および用語に精通していることを前提としています。サービスやアプリケーションの設定、ネットワークの管理など、独自の基本的なタスクを実行できるようにする必要があります。この例ではサービスが必要で、設定がサンプルの範囲に含まれていない場合は、適切に設定する知識があり、サービスの結果として生じる状態のみがドキュメントに記載されていることを前提とします。

Expert

Expert サンプルは、サブジェクトに関する最高レベルの知識が必要です。機能ベースのドキュメントとマニュアルに基づいて多くのタスクを実行することが期待されており、ドキュメントは最も複雑なシナリオを対象としています。

付録E 用語集

E.1. 製品およびプロジェクト名

Developer Launcher (developers.redhat.com/launch)

Developer Launcher (developers.redhat.com/launch) は、Red Hat が提供するスタンドアロンの入門エクスペリエンスです。これは、OpenShift でのクラウドネイティブ開発の開始に役立ちます。これには、OpenShift にダウンロード、ビルド、およびデプロイできる機能のサンプルアプリケーションが含まれます。

Minishift または CDK

Minishift を使用してマシンで実行している OpenShift クラスター。

E.2. DEVELOPER LAUNCHER に固有の用語

例

アプリケーション仕様 (例: REST API を持つ Web サービス)。

この例では、通常、実行する言語やプラットフォームを指定していません。説明には意図された機能のみが含まれます。

アプリケーションの例

特定の **ランタイム** に対する特定の **サンプル** の言語固有の実装。サンプルアプリケーションは、**サンプルカタログ** に記載されています。

たとえば、サンプルアプリケーションは Thorntail ランタイムを使用して実装される REST API を持つ Web サービスです。

サンプルカタログ

サンプルアプリケーションに関する情報が含まれる Git リポジトリ。

ランタイム

サンプルアプリケーション を実行するプラットフォーム。たとえば、Thorntail または Eclipse Vert.x などです。