Red Hat JBoss Enterprise Application Platform 8.0

JBoss EAP にデプロイするアプリケーションの開発のスタートガイド

JBoss EAP にデプロイするアプリケーションの作成開始

Last Updated: 2024-02-09

Red Hat JBoss Enterprise Application Platform 8.0 JBoss EAP にデプロイするアプリケーションの開発のスタートガイド

JBoss EAP にデプロイするアプリケーションの作成開始

法律上の通知

概要

Maven をプロジェクト管理ツールとして使用して、JBoss EAP にデプロイするアプリケーションの作成を開始します。アプリケーションは、ベアメタルまたは OpenShift Container Platform 上で実行される JBoss EAP にデプロイします。

目次

JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. **このリンクをクリック** してチケットを作成します。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

新しいプログラミング言語やテクノロジーに慣れるには、"Hello World" アプリケーションを作成するのが最善です。Maven をプロジェクト管理ツールとして使用して、JBoss EAP の "Hello World" アプリケーションを作成できます。

Hello World アプリケーションを作成してデプロイし、デプロイメントをテストするには、次の手順に従います。

ベアメタルデプロイメント

- Hello World アプリケーション用の Maven プロジェクトの作成
- Hello World サーブレットの作成
- ベアメタルインストールへのアプリケーションのデプロイ
- 統合テストに必要な Maven 依存関係とプロファイルの追加
- ベアメタル上で実行されている JBoss EAP にデプロイしたアプリケーションのテスト

OpenShift Container Platform デプロイメント

- Hello World アプリケーション用の Maven プロジェクトの作成
- Hello World サーブレットの作成
- OpenShift Container Platform へのアプリケーションのデプロイ
- 統合テストに必要な Maven 依存関係とプロファイルの追加
- OpenShift Container Platform 上の JBoss EAP にデプロイしたアプリケーションのテスト

第1章 HELLO WORLD アプリケーション用の MAVEN プロジェクトの作成

Maven プロジェクトには **pom.xml** 設定ファイルが含まれており、アプリケーションの作成に必要なディレクトリー構造が含まれています。**pom.xml** 設定ファイルを設定して、アプリケーションの依存関係を追加できます。

Hello World アプリケーション用の Maven プロジェクトを作成するには、次の手順に従います。

- **maven-archetype-webapp** を使用した Maven プロジェクトの作成
- Maven プロジェクトのプロパティーの定義
- Maven プロジェクトのリポジトリーの定義
- Maven プロジェクトの依存関係管理として JBoss EAP BOM をインポートする
- Maven プロジェクトへのプラグイン管理の追加
- Maven プロジェクトの検証

1.1. MAVEN-ARCHETYPE-WEBAPP を使用した MAVEN プロジェクトの作成

maven-archetype-webapp アーキタイプを使用して、JBoss EAP にデプロイするアプリケーションをビルドするための Maven プロジェクトを作成します。Maven には、プロジェクトタイプに特化したテンプレートをベースにプロジェクトを作成できるさまざまなアーキタイプがあります。**maven-archetype-webapp** は、シンプルな Web アプリケーションの開発に必要な構造を持つプロジェクトを作成します。

前提条件

- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. **mvn** コマンドを使用して Maven プロジェクトをセットアップします。このコマンドは、プロジェクトのディレクトリー構造と **pom.xml** 設定ファイルを作成します。

```
$ mvn archetype:generate                \
-DgroupId=org.jboss.as.quickstarts      \(1)
-DartifactId=helloworld                 \(2)
-DarchetypeGroupId=org.apache.maven.archetypes  \(3)
-DarchetypeArtifactId=maven-archetype-webapp    \(4)
-DinteractiveMode=false                 (5)
```

(1) **groupID** はプロジェクトを一意に識別するものです。

(2) **artifactID** は生成された **jar** アーカイブの名前です。

(3) **maven-archetype-webapp** の **groupID**。

(4) **maven-archetype-webapp** の **artifactID**。

5 対話モードを開始するのではなく、指定されたパラメーターを使用するように Maven に指示します。

2. 生成されたディレクトリーに移動します。

```
$ cd helloworld
```

3. 生成された **pom.xml** 設定ファイルをテキストエディターで開きます。

4. **pom.xml** 設定ファイルの **<project>** セクション内にある **<name>helloworld Maven Webapp</name>** 行の後ろの内容を削除します。
ファイルが次のようになっていることを確認します。

```
<?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>org.jboss.as.quickstarts</groupId>
   <artifactId>helloworld</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>war</packaging>
   <name>helloworld Maven Webapp</name>

</project>
```

内容を削除したのは、アプリケーションには不要なためです。

次のステップ

- JBoss EAP Hello World アプリケーション用の Maven プロジェクトのプロパティーの定義

1.2. MAVEN プロジェクトのプロパティーの定義

Maven の **pom.xml** 設定ファイルで、プロパティーを値のプレースホルダーとして定義できます。JBoss EAP サーバーの値をプロパティーとして定義し、設定内でその値を一貫して使用します。

前提条件

- Maven プロジェクトを初期化している。
詳細は、JBoss EAP Hello World アプリケーション用の Maven プロジェクトの初期化 を参照してください。

手順

- プロパティー **<version.server>** を、設定したアプリケーションのデプロイ先の JBoss EAP バージョンに定義します。

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

```
    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
      <version.server>8.0.0.GA-redhat-00009</version.server>
    </properties>
</project>
```

次のステップ

- Maven プロジェクトのリポジトリーの定義

1.3. MAVEN プロジェクトのリポジトリーの定義

Maven でダウンロードするアーティファクトとプラグインを検索するアーティファクトリポジトリーとプラグインリポジトリーを定義します。

前提条件

- Maven プロジェクトを初期化している。
  詳細は、JBoss EAP Hello World アプリケーション用の Maven プロジェクトの初期化 を参照してください。

手順

1. アーティファクトリポジトリーを定義します。

```
<project>
  ...
  <repositories>
    <repository>                                          (1)
      <id>jboss-public-maven-repository</id>
      <name>JBoss Public Maven Repository</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
      <layout>default</layout>
    </repository>
    <repository>                                          (2)
      <id>redhat-ga-maven-repository</id>
      <name>Red Hat GA Maven Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
```

```
            <updatePolicy>never</updatePolicy>
        </snapshots>
        <layout>default</layout>
    </repository>
  </repositories>
</project>
```

1. Red Hat GA Maven リポジトリーは、製品化されたすべての JBoss EAP アーティファクトとその他の Red Hat アーティファクトを提供します。
2. JBoss Public Maven リポジトリーは、WildFly Maven プラグインなどのアーティファクトを提供します。

2. プラグインリポジトリーを定義します。

```
<project>
  ...
  <pluginRepositories>
    <pluginRepository>
      <id>jboss-public-maven-repository</id>
      <name>JBoss Public Maven Repository</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
    <pluginRepository>
      <id>redhat-ga-maven-repository</id>
      <name>Red Hat GA Maven Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</project>
```

次のステップ

- Maven プロジェクトの JBoss EAP BOM 依存関係管理のインポート

1.4. MAVEN プロジェクトの依存関係管理として JBOSS EAP BOM をインポートする

JBoss EAP EE With Tools の BOM (Bill of Material) をインポートして、ランタイムの Maven 依存関係のバージョンを管理します。**<dependencyManagement>** セクションで BOM を指定する場合、**provided** スコープで定義されている Maven 依存関係のバージョンを個別に指定する必要はありません。

前提条件

- Maven プロジェクトを初期化している。
  詳細は、JBoss EAP Hello World アプリケーション用の Maven プロジェクトの初期化 を参照してください。

手順

1. **pom.xml** 設定ファイルの properties セクションに BOM バージョンのプロパティーを追加します。

```
<properties>
    ....
    <version.bom.ee>${version.server}</version.bom.ee>
</properties>
```

   プロパティー **<version.server>** で定義した値が、BOM バージョンの値として使用されます。

2. JBoss EAP BOM の依存関係管理をインポートします。

```
<project>
    ...
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jboss.bom</groupId>                    (1)
                <artifactId>jboss-eap-ee-with-tools</artifactId>    (2)
                <version>${version.bom.ee}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

(1) JBoss EAP が提供する BOM の GroupID。

(2) JBoss EAP が提供する BOM の ArtifactID。この BOM は、サポートされている JBoss EAP Java EE API に加えて、追加の JBoss EAP API JAR およびクライアント BOM、および Arquillian などの開発ツールを提供します。

次のステップ

- Maven プロジェクトへのプラグイン管理の追加

1.5. MAVEN プロジェクトへのプラグイン管理の追加

Maven CLI コマンドに必要なプラグインを取得するために、Maven プラグイン管理セクションを **pom.xml** 設定ファイルに追加します。

前提条件

- Maven プロジェクトを初期化している。

詳細は、JBoss EAP Hello World アプリケーション用の Maven プロジェクトの初期化 を参照してください。

手順

1. **<properties>** セクションで、**wildfly-maven-plugin** および **maven-war-plugin** のバージョンを定義します。

```
<properties>
    ...
    <version.plugin.wildfly>4.1.1.Final</version.plugin.wildfly>
    <version.plugin.war>3.3.2</version.plugin.war>
</properties>
```

2. **<project>** セクション内の **<build>** セクションに **<pluginManagement>** を追加します。

```
<project>
    ...
    <build>
        <pluginManagement>
            <plugins>
                <plugin>  (1)
                    <groupId>org.wildfly.plugins</groupId>
                    <artifactId>wildfly-maven-plugin</artifactId>
                    <version>${version.plugin.wildfly}</version>
                </plugin>
                <plugin>  (2)
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>${version.plugin.war}</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

(1) **wildfly-maven-plugin** を使用すると、**wildfly:deploy** コマンドを使用してアプリケーションを JBoss EAP にデプロイできます。

(2) JDK17+ との互換性を確保するには、war プラグインのバージョンを管理する必要があります。

次のステップ

- Maven プロジェクトの検証

1.6. MAVEN プロジェクトの検証

設定した Maven プロジェクトがビルドされることを確認します。

前提条件

- Maven プロパティーを定義している。

詳細は、Maven プロジェクトのプロパティーの定義 を参照してください。

- Maven リポジトリーを定義している。
  詳細は、Maven プロジェクトのリポジトリーの定義 を参照してください。

- JBoss EAP の BOM (Bill of Material) を依存関係管理としてインポートしている。
  詳細は、Maven プロジェクトの依存関係管理として JBoss EAP BOM をインポートする を参照してください。

- プラグイン管理を追加している。
  詳細は、サーバー Hello World アプリケーション用の Maven プロジェクトにプラグイン管理を追加する を参照してください。

手順

- **pom.xml** に追加した Maven 依存関係をローカルにインストールします。

```
$ mvn package
```

次のような出力が得られます。

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...
```

次のステップ

- Hello World サーブレットの作成

第2章 HELLO WORLD サーブレットの作成

アクセスされたときに "Hello World!" を返すサーブレットを作成します。

この手順では、**<application_home>** は、アプリケーションの **pom.xml** 設定ファイルが含まれるディレクトリーを参照します。

前提条件

- Maven プロジェクトを作成している。
  詳細は、Hello World アプリケーション用の Maven プロジェクトの作成 を参照してください。

手順

1. 必要な依存関係を **pom.xml** 設定ファイルの **<dependencyManagement>** セクションの後に追加します。

```
<project>
  ...
  <dependencies>
    <dependency>                                    (1)
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <scope>provided</scope>                       (2)
    </dependency>
  </dependencies>
```

(1) **jakarta.servlet-api** 依存関係は、Jakarta Servlet API を提供します。

(2) 依存関係がアプリケーションに含まれないように、スコープを **provided** として定義します。アプリケーションに依存関係を含めない理由は、この依存関係が **jboss-eap-ee-with-tools** BOM によって管理されるためです。このような依存関係は JBoss EAP に組み込まれます。

注記

jboss-eap-ee-with-tools BOM が **<dependencyManagement>** セクションにインポートされるため、依存関係はバージョンなしで定義します。

2. **<application_home>** ディレクトリーに移動します。
3. Java ファイルを保存するディレクトリーを作成します。

```
$ mkdir -p src/main/java/org/jboss/as/quickstarts/helloworld
```

4. 新しいディレクトリーに移動します。

```
$ cd src/main/java/org/jboss/as/quickstarts/helloworld
```

5. "Hello World!" を返すサーブレット **HelloWorldServlet.java** を作成します。

```
package org.jboss.as.quickstarts.helloworld;
```

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/HelloWorld")                    (1)
public class HelloWorldServlet extends HttpServlet {

    static String PAGE_HEADER = "<html><head><title>helloworld</title></head><body>";

    static String PAGE_FOOTER = "</body></html>";

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println(PAGE_HEADER);
        writer.println("<h1> Hello World! </h1>");
        writer.println(PAGE_FOOTER);
        writer.close();
    }
}
```

(1) **@WebServlet("/HelloWorld")** アノテーションは、JBoss EAP に次の情報を提供します。

- このクラスがサーブレットであること。
- このサーブレットを URL "**<application_URL>**/HelloWorld" で利用可能にすること。たとえば、JBoss EAP がローカルホスト上で実行されており、デフォルトの HTTP ポート 8080 でアクセスできる場合、URL は **http://localhost:8080/helloworld/HelloWorld** になります。

6. **<application_home>**/src/main/webapp ディレクトリーに移動します。Maven が作成したファイル "index.jsp" が見つかります。このファイルは、ユーザーがアプリケーションにアクセスしたときに "Hello World!" を出力します。

7. "index.jsp" ファイルの内容を次の内容に置き換えて、Hello World サーブレットにリダイレクトするようにファイルを更新します。

```
<html>
    <head>
        <meta http-equiv="Refresh" content="0; URL=HelloWorld">
    </head>
</html>
```

8. **<application_home>** ディレクトリーに移動します。

9. 次のコマンドを使用して、アプリケーションをコンパイルし、Web アーカイブ (WAR) としてパッケージ化します。

```
$ mvn package
```

出力例

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...
```

次のステップ

- サーバーへのアプリケーションのデプロイ

第3章 サーバーへのアプリケーションのデプロイ

アプリケーションは、ベアメタルまたは OpenShift Container Platform 上で実行されている JBoss EAP サーバーにデプロイできます。

ベアメタル上で実行されている JBoss EAP サーバーにアプリケーションをデプロイするには、次の手順に従います。

- ベアメタルインストールへのアプリケーションのデプロイ

OpenShift Container Platform 上で実行されている JBoss EAP サーバーにアプリケーションをデプロイするには、以下の手順に従います。

- OpenShift Container Platform へのデプロイに向けたアプリケーションの準備
- Helm を使用した OpenShift 上の JBoss EAP へのアプリケーションのデプロイ

3.1. ベアメタルインストールへのアプリケーションのデプロイ

JBoss EAP デプロイプラグインを使用して、アプリケーションを JBoss EAP にデプロイできます。

前提条件

- アプリケーションを作成している。
  詳細は、Hello World サーブレットの作成 を参照してください。
- JBoss EAP が実行されている。

手順

1. アプリケーションのルートディレクトリーに移動します。
   アプリケーションのルートディレクトリーには、**pom.xml** 設定ファイルが含まれています。
2. 次のビルド設定を **pom.xml** 設定ファイルの **<project>** セクションに追加して、アプリケーションアーカイブファイル名を定義します。

```
<build>
    ...
    <finalName>${project.artifactId}</finalName>  (1)
</build>
```

(1) デプロイメントの名前をプロジェクトのアーティファクト ID に設定します。

3. JBoss EAP デプロイプラグインを使用してアプリケーションをビルドおよびデプロイします。

```
$ mvn package wildfly:deploy
```

検証

- ブラウザーでアドレス **http://localhost:8080/helloworld/** に移動します。
  http://localhost:8080/helloworld/HelloWorld にリダイレクトされ、次のメッセージが表示されます。

```
Hello World!
```

次のステップ

- JBoss EAP にデプロイしたアプリケーションのテスト

3.2. OPENSHIFT CONTAINER PLATFORM へのアプリケーションのデプロイ

source-to-image (S2I) ワークフローを使用して、OpenShift Container Platform 上の JBoss EAP にアプリケーションをデプロイできます。S2I ワークフローは、Git リポジトリーからソースコードを取得し、使用する言語とフレームワークをベースとするコンテナーに挿入します。S2I ワークフローが完了すると、**src** コードがコンパイルされ、アプリケーションがパッケージ化されて JBoss EAP サーバーにデプロイされます。

3.2.1. OpenShift Container Platform へのデプロイに向けたアプリケーションの準備

OpenShift Container Platform は、Git リポジトリーでホストされるアプリケーションを使用します。アプリケーションを OpenShift にデプロイするには、まずアプリケーションを Git リポジトリーにプッシュする必要があります。その後、JBoss EAP Helm チャートを使用してアプリケーションのデプロイを設定できます。

前提条件

- アプリケーションを作成している。
  詳細は、Hello World サーブレットの作成 を参照してください。
- Git リポジトリーを作成している。

手順

1. アプリケーションがローカル Git リポジトリーにまだない場合は、アプリケーションをローカル Git リポジトリーに移動します。

```
$ mv -r helloworld/ <your_git_repo>
```

2. **pom.xml** 設定ファイルで次のプロパティーを定義します。

```
<properties>
    ...
    <version.plugin.eap>1.0.0.Final-redhat-00013</version.plugin.eap> (1)
</properties>
```

(1) **<version.plugin.eap>** は、JBoss EAP Maven プラグインのバージョンを定義します。

3. JBoss EAP Maven プラグインを **<project>** セクション内の **<build>** セクションの **<pluginManagement>** に追加します。

```
<project>
    ...
    <build>
        <pluginManagement>
```

```
        <plugins>
          ...
          <plugin>
            <groupId>org.jboss.eap.plugins</groupId>
            <artifactId>eap-maven-plugin</artifactId>
            <version>${version.plugin.eap}</version>
          </plugin>
        </plugins>
      </pluginManagement>
    </build>
</project>
```

4. **pom.xml** 設定ファイルにプロファイル "openshift" を作成します。
このプロファイルで、OpenShift Container Platform へのデプロイに必要なプラグイン、機能パック、およびレイヤーを定義します。

```
<profiles>
  <profile>
    <id>openshift</id>
    <build>
      <plugins>
        <plugin>
          <groupId>org.jboss.eap.plugins</groupId>
          <artifactId>eap-maven-plugin</artifactId> (1)
          <configuration>
            <channels>
              <channel>
                <manifest>
                  <groupId>org.jboss.eap.channels</groupId>
                  <artifactId>eap-8.0</artifactId>
                </manifest>
              </channel>
            </channels>
            <feature-packs>
              <feature-pack> (2)
                <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
              </feature-pack>
              <feature-pack>
                <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
              </feature-pack>
            </feature-packs>
            <layers> (3)
              <layer>cloud-server</layer>
            </layers>
            <name>ROOT.war</name> (4)
          </configuration>
          <executions>
            <execution>
              <goals>
                <goal>package</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
```

```
        </build>
    </profile>
</profiles>
```

1. **wildfly-maven-plugin** は、アプリケーションがデプロイされた JBoss EAP インスタンスを OpenShift Container Platform 上にプロビジョニングするための JBoss EAP プラグインです。
2. **feature-packs** は、機能パック (サーバーを動的にプロビジョニングする機能を含む zip ファイル) を定義します。この場合、機能パック **org.wildfly:wildfly-galleon-pack** および **org.wildfly.cloud:wildfly-cloud-galleon-pack** が必要です。
3. **layers** は、プロビジョニングされたサーバーに含めるレイヤーを (設定した機能パックから) 定義します。各レイヤーは、単独で、または他のレイヤーと組み合わせてインストールできる 1 つ以上のサーバー機能を特定します。ここでは、クラウドサーバーに適した JBoss EAP の基本機能のみをプロビジョニングする **cloud-server** レイヤーを選択します。
4. **<name>ROOT.war</name>**: 生成されるアプリケーションの Web アーカイブ (WAR) の名前を定義します。**ROOT.war** を指定すると、アプリケーションはサーバーのルートパスにデプロイされます。指定しないと、**<name/>** 相対パスにデプロイされます。

5. アプリケーションがコンパイルされることを確認します。

```
$ mvn package -Popenshift
```

6. 変更をリポジトリーにプッシュします。

次のステップ

- Helm を使用した OpenShift 上の JBoss EAP へのアプリケーションのデプロイ

3.2.2. Helm を使用した OpenShift 上の JBoss EAP へのアプリケーションのデプロイ

JBoss EAP Helm チャートを使用して、Helm を使用してアプリケーションを設定し、OpenShift 上の JBoss EAP にデプロイします。

前提条件

- OpenShift Container Platform にデプロイするアプリケーションの準備が完了している。
  詳細は、OpenShift Container Platform へのデプロイに向けたアプリケーションの準備 を参照してください。
- OpenShift Container Platform でプロジェクトを作成している。
  詳細は、プロジェクトの使用 を参照してください。
- OpenShift CLI (**oc**) をインストールしている。
  詳細は、OpenShift CLI のインストール を参照してください。
- マシンから OpenShift Container Platform にログインしている。
  詳細は、OpenShift CLI へのログイン を参照してください。
- Helm をインストールしている。
  詳細は、Installing Helm を参照してください。

手順

1. アプリケーションのルートディレクトリーに **charts** というディレクトリーを作成し、そこに移動します。アプリケーションのルートディレクトリーは、**pom.xml** 設定ファイルを含んでいるディレクトリーです。

```
$ mkdir charts; cd charts
```

2. 次の内容を含む **helm.yaml** ファイルを作成します。

```
build:
  uri: https://github.com/<user>/<repository>.git  1
  ref: <branch_name>                                2
  contextDir: helloworld                            3
deploy:
  replicas: 1                                       4
```

1. OpenShift Container Platform にデプロイするアプリケーションを含む Git リポジトリーの URL を指定します。
2. アプリケーションを含む Git ブランチを指定します。
3. アプリケーションが存在するディレクトリーを指定します。
4. 作成する Pod の数を指定します。

3. Helm で JBoss EAP リポジトリーを設定します。
   - まだ JBoss EAP リポジトリーを Helm に追加していない場合は、追加します。

```
$ helm repo add jboss-eap https://jbossas.github.io/eap-charts/
```

   - すでに JBoss EAP リポジトリーを Helm に追加している場合は、それを更新します。

```
$ helm repo update jboss-eap
```

4. Helm を使用してアプリケーションをデプロイします。

```
$ helm install helloworld -f helm.yaml jboss-eap/eap8
```

デプロイが完了するまでに数分かかる場合があります。

検証

1. デプロイメントへのルートの URL を取得します。

```
$ APPLICATION_URL=https://$(oc get route helloworld --template='{{ .spec.host }}') &&
echo "" &&
echo "Application URL: $APPLICATION_URL"
```

2. ブラウザーで "Application URL" に移動します。
"/HelloWorld" パスのサーブレットにリダイレクトされ、次のメッセージが表示されます。

```
Hello World!
```

次のステップ

- JBoss EAP にデプロイしたアプリケーションのテスト

第4章 JBOSS EAP にデプロイしたアプリケーションのテスト

JBoss EAP にデプロイした Hello World アプリケーションが動作していることを確認するために、統合テストを追加できます。

ベアメタル上で実行されている JBoss EAP サーバーにデプロイしたアプリケーションのテストを追加するには、以下の手順に従います。

- 統合テストに必要な Maven 依存関係とプロファイルの追加
- アプリケーションをテストするためのテストクラスの作成
- ベアメタル上で実行されている JBoss EAP にデプロイしたアプリケーションのテスト

OpenShift Container Platform 上で実行されている JBoss EAP サーバーにデプロイしたアプリケーションのテストを追加するには、以下の手順に従います。

- 統合テストに必要な Maven 依存関係とプロファイルの追加
- アプリケーションをテストするためのテストクラスの作成
- OpenShift Container Platform 上の JBoss EAP にデプロイしたアプリケーションのテスト

4.1. 統合テストに必要な MAVEN 依存関係とプロファイルの追加

アプリケーションの統合テストを作成するには、必要な Maven 依存関係を追加します。

前提条件

- Maven プロジェクトを作成している。
  詳細は、Hello World アプリケーション用の Maven プロジェクトの作成 を参照してください。

手順

1. **pom.xml** 設定ファイルで次のプロパティーを定義します。

```
<properties>
  ...
  <version.plugin.failsafe>3.2.2</version.plugin.failsafe>
</properties>
```

2. テストに必要な依存関係を追加します。

```
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

3. プロファイルを定義して、統合テストに必要なプラグインを追加します。

```
<project>
  ...
  <profiles>
  ...
    <profile>
      <id>integration-testing</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-failsafe-plugin</artifactId> (1)
            <version>${version.plugin.failsafe}</version>
            <configuration>
              <includes>
                <include>**/HelloWorldServletIT</include> (2)
              </includes>
            </configuration>
            <executions>
              <execution>
                <goals>
                  <goal>integration-test</goal>
                  <goal>verify</goal>
                </goals>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
</project>
```

(1) 統合テストを実行するための Maven プラグイン。

(2) アプリケーションをテストする Java クラスの名前。

次のステップ

- アプリケーションをテストするためのテストクラスの作成

4.2. アプリケーションをテストするためのテストクラスの作成

OpenShift Container Platform 上の JBoss EAP でアプリケーションがデプロイおよび実行されていることを検証する統合テストを作成します。このテストでは、アプリケーションの Web ページの HTTP GET が 200 OK を返すことを確認します。

この手順では、**<application_home>** は、アプリケーションの **pom.xml** 設定ファイルが含まれるディレクトリーを参照します。

前提条件

- アプリケーションを JBoss EAP にデプロイしている。

詳細は、アプリケーションのビルドとサーバーへのデプロイ を参照してください。

- JUnit テストに必要な Maven 依存関係を追加している。
  詳細は、統合テストに必要な Maven 依存関係とプロファイルの追加 を参照してください。

手順

1. **<application_home>** ディレクトリーに移動します。
2. テストクラスを格納するディレクトリーを作成します。

```
$ mkdir -p src/test/java/org/jboss/as/quickstarts/helloworld
```

3. 新しいディレクトリーに移動します。

```
$ cd src/test/java/org/jboss/as/quickstarts/helloworld
```

4. デプロイメントをテストする Java クラス **HelloWorldServletIT.java** を作成します。

```
package org.jboss.as.quickstarts.helloworld;

import org.junit.Test;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;
import static org.junit.Assert.assertEquals;

public class HelloWorldServletIT {

    private static final String DEFAULT_SERVER_HOST = "http://localhost:8080/helloworld";
(1)

    @Test
    public void testHTTPEndpointIsAvailable() throws IOException, InterruptedException,
URISyntaxException {
        String serverHost = System.getProperty("server.host");
        if (serverHost == null) {
            serverHost = DEFAULT_SERVER_HOST;
        }
        final HttpRequest request = HttpRequest.newBuilder()
                .uri(new URI(serverHost+"/HelloWorld"))
                .GET()
                .build();                                                    (2)
        final HttpClient client = HttpClient.newBuilder()
                .followRedirects(HttpClient.Redirect.ALWAYS)
                .connectTimeout(Duration.ofMinutes(1))
                .build();                                                    (3)
        final HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString()); (4)
```

```
        assertEquals(200, response.statusCode());                    5
    }
}
```

1. アプリケーションが実行されている URL。この値は **sever.host** が定義されていない場合に使用されます。
2. アプリケーション URI の HttpRequest インスタンスを作成します。
3. アプリケーションに要求を送信し、アプリケーションからの応答を受信するための HttpClient を作成します。
4. アプリケーションから応答を取得します。
5. アプリケーションから返される応答が "200" であることをテストして、アプリケーションが到達可能であることを示します。

次のステップ

- ベアメタル上で実行されている JBoss EAP サーバーにデプロイしたアプリケーションをテストするには、次の手順に従います。
  - ベアメタル上で実行されている JBoss EAP にデプロイしたアプリケーションのテスト
- OpenShift Container Platform 上で実行されている JBoss EAP サーバーにデプロイしたアプリケーションをテストするには、次の手順に従います。
  - OpenShift Container Platform 上の JBoss EAP にデプロイしたアプリケーションのテスト

4.3. ベアメタル上で実行されている JBOSS EAP にデプロイしたアプリケーションのテスト

ベアメタル上で実行されている JBoss EAP にデプロイしたアプリケーションをテストします。

前提条件

- テストクラスを作成している。
  詳細は、アプリケーションをテストするためのテストクラスの作成 を参照してください。
- テストするアプリケーションを JBoss EAP にデプロイしている。
- JBoss EAP が実行されている。

手順

1. **<application_home>** ディレクトリーに移動します。
2. **verify** コマンドを使用して **integration-testing** プロファイルを指定し、統合テストを実行します。

```
$ mvn verify -Pintegration-testing
```

出力例

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.2.2:verify (default) @ helloworld ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.982 s
[INFO] Finished at: 2023-11-22T14:53:54+05:30
[INFO] ------------------------------------------------------------------------
```

4.4. OPENSHIFT CONTAINER PLATFORM 上の JBOSS EAP にデプロイしたアプリケーションのテスト

OpenShift Container Platform 上の JBoss EAP にデプロイしたアプリケーションをテストします。

前提条件

- テストクラスを作成している。
  詳細は、アプリケーションをテストするためのテストクラスの作成 を参照してください。

手順

1. 変更を Git リポジトリーにプッシュします。
2. **<application_home>** ディレクトリーに移動します。
3. **verify** コマンドを使用して、**integration-testing** プロファイルをアクティブ化し、アプリケーションへの URL を指定してテストを実行します。

```
$ mvn verify -Pintegration-testing -Dserver.host=https://$(oc get route helloworld --
template='{{ .spec.host }}')
```

注記

このテストでは、SSL/TLS を使用して、デプロイされたアプリケーションに接続します。したがって、テストを実行するマシンによって証明書が信頼されている必要があります。

証明書を信頼するには、証明書を Java トラストストアに追加する必要があります。

例

```
$ keytool -trustcacerts -keystore _<path-to-java-truststore>_ -storepass
_<trust-store-password>_ -importcert -alias _<alias-for-the-certificate>_ -file
_<path-to-certificate>_/_<certificate-name>_
```

出力例

```
[INFO] Running org.jboss.as.quickstarts.helloworld.HelloWorldServletIT
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.345 s -- in
org.jboss.as.quickstarts.helloworld.HelloWorldServletIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.2.2:verify (default) @ helloworld ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.984 s
[INFO] Finished at: 2023-11-30T15:51:22+05:30
[INFO] ------------------------------------------------------------------------
```

改訂日時: 2024-02-08