# Red Hat JBoss Enterprise Application Platform 8.0

## 在 OpenShift Container Platform 上使用 JBoss EAP

使用 Red Hat JBoss Enterprise Application Platform for OpenShift 进行开发的指南

Last Updated: 2024-06-30

# Red Hat JBoss Enterprise Application Platform 8.0 在 OpenShift Container Platform 上使用 JBoss EAP

使用 Red Hat JBoss Enterprise Application Platform for OpenShift 进行开发的指南

## 法律通告



## 摘要

使用 Red Hat JBoss Enterprise Application Platform for OpenShift 指南

# 目录

# 提供有关 JBOSS EAP 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

**流程**

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

# 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。

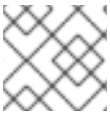
# 第 1 章 什么是 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

Red Hat JBoss Enterprise Application Platform 8.0 (JBoss EAP)是一个基于开放标准构建的中间件平台，符合 Jakarta EE 10 规范。它为高可用性集群、消息传递和分布式缓存等功能提供预配置选项。它包含一个模块化结构，允许您在需要时启用服务，从而提高启动速度。

通过使用基于 Web 的管理控制台和管理命令行界面(CLI)，您可以编写和自动执行任务并避免编辑 XML 配置文件。此外，JBoss EAP 还包含可用于开发、部署和运行安全、可扩展的 Jakarta EE 应用的 API 和开发框架。JBoss EAP 8.0 是 Web Profile、Core Profile 和 Full Platform 规范的 Jakarta EE 10 兼容实现。

## 1.1. JBOSS EAP 如何在 OPENSHIFT 上使用？

红帽提供了在 OpenShift 上使用 JBoss EAP 构建和运行应用程序镜像的容器镜像。



**注意**

红帽不再提供包含 JBoss EAP 的镜像。

## 1.2. 比较：用于 OPENSHIFT 的 JBOSS EAP 和 JBOSS EAP

比较 JBoss EAP 产品与 JBoss EAP for OpenShift 镜像相比，有一些显著区别。以表介绍了 JBoss EAP for OpenShift 当前版本与 JBoss EAP 的比较

**表 1.1. JBoss EAP 和 JBoss EAP for OpenShift 之间的区别**

| JBoss EAP 功能 | OpenShift JBoss EAP 中的状态 | 描述 |
|---|---|---|
| JBoss EAP 管理控制台 | 不包括 | JBoss EAP 管理控制台没有包括在 JBoss EAP for OpenShift 的这个版本中。 |
| JBoss EAP 管理 CLI | 不推荐 | 不建议将 JBoss EAP 管理 CLI 与在容器化环境中运行的 JBoss EAP 一起使用。容器重启时，使用运行中容器中的管理 CLI 进行的任何配置更改都将丢失。可以从 pod 内访问管理 CLI，以进行故障排除。 |
| 受管域 | 不支持 | 虽然不支持 JBoss EAP 受管域，但应用程序的创建和分发是由 OpenShift 中的容器管理的。 |
| 默认根页面 | Disabled | 默认的 root 页面已被禁用，但您可以将自己的应用程序部署在 root 上下文中作为 **ROOT.war**。 |
| 远程消息传递 | 支持 | 支持 Red Hat AMQ 进行 pod 和远程消息传递。ActiveMQ Artemis 仅支持使用 JBoss EAP 实例的单一 pod 中的消息传递，且仅在 Red Hat AMQ 不存在时才启用。 |

| JBoss EAP 功能 | OpenShift JBoss EAP 中的状态 | 描述 |
| --- | --- | --- |
| 事务恢复 | 支持 | EAP 操作器是 OpenShift 4 中唯一进行了测试并被支持的事务选项。有关使用 EAP 操作器恢复事务的更多信息，请参阅 EAP Operator for Safe Transaction Recovery。 |

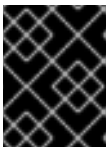
## 1.3. 版本兼容性和支持

JBoss EAP for OpenShift 为 OpenJDK 17 提供镜像。

镜像有两个变体可用：S2I 构建器镜像和运行时镜像。S2I 构建器镜像包含所有必要的工具，可让您在 S2I 构建过程中置备完整的 JBoss EAP 服务器。运行时镜像包含运行 JBoss EAP 但不包含服务器所需的依赖项。服务器在串联构建期间在运行时镜像中安装。

以下修改应用于 JBoss EAP 8.0 for OpenShift 中的镜像。

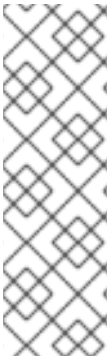
- S2I 构建器镜像不包含已安装的 JBoss EAP 服务器，并在 S2I 构建期间安装 JBoss EAP 8.0 服务器。
- 在 S2I 构建期间，在应用 **pom** 文件中配置 eap-maven-plugin。
- 通过在 S2I 构建期间设置 **GALLEON_PROVISION_FEATURE_PACKS**、**GALLEON_PROVISION_LAYERS** 和 **GALLEON_PROVISION_CHANNELS** 环境变量，使用现有的 JBoss EAP 7.4 应用。
- S2I 构建期间置备的 JBoss EAP 服务器配置文件包含为 OpenShift 自定义的 **standalone.xml** 服务器配置文件。



**重要**

sever 包含 **standalone.xml** 配置文件，而不是用于 JBoss EAP 7.4 的 **standalone-openshift.xml** 配置文件。

- 在镜像内，**JBOSS_HOME** 值为 **/opt/server**。**JBOSS_HOME** 的值是 JBoss EAP 7.4 的 **/opt/eap**。
- 镜像不再存在 **Jolokia 代理**。
- **Prometheus 代理** 没有安装。
- **Python 探测** 不存在。
- 镜像中不再存在 **SSO** 适配器。
- 不存在 **ActiveMQ.rar**。



**注意**

以下发现机制协议已弃用，并被其他协议替代：

- **openshift.DNS_PING** 协议已弃用，并替换为 **dns.DNS_PING** 协议。如果您在自定义 **standalone.xml** 文件中引用 **openshift. DNS_PING** 协议，请将协议替换为 **dns.DNS_PING** 协议。
- **openshift.KUBE_PING** 发现机制协议已弃用，并被 **kubernetes.KUBE_PING** 协议替代。

### 1.3.1. OpenShift 4.x 支持

OpenShift 4.1 中的更改会影响对 Jolokia 的访问，OpenShift 4.x web 控制台中不再提供 Open Java 控制台。

在以前的 OpenShift 版本中，某些 kube-apiserver 代理请求被验证并传递给集群。这个行为现在被视为不安全，因此不再支持以这种方式访问 Jolokia。

由于 OpenShift 控制台的代码库更改，Open Java 控制台的链接不再可用。

### 1.3.2. IBM Z 支持

镜像中没有包括 **libartemis-native** 的 s390x 变体。因此，任何与 AIO 相关的设置都不会考虑。

- **journal-type** ：将 **journal-type** 设为 **ASYNCIO** 没有作用。此属性的值可在运行时默认为 **NIO**。
- **journal-max-io** ：此属性无效。
- **journal-store-enable-async-io**: 此属性无效。

#### 1.3.2.1. 在 OpenShift 上从 JBoss EAP 7.4 升级到 JBoss EAP 8.0

在 OpenShift 上使用 JBoss EAP 7.4 安装的文件 **standalone.xml** 与 JBoss EAP 8.0 及更高版本不兼容。在为 OpenShift 启动 JBoss EAP 8.0 或更高版本的容器前，您必须修改该文件并将其重命名为 **standalone.xml**。

**其他资源**

- 在将 JBoss EAP 7.1 升级到 OpenShift 上的 JBoss EAP 8.0 时，对 **standalone.xml** 的更新。

### 1.3.3. 部署选项

您可以使用 EAP 操作器（特定于 JBoss EAP 的控制器）在 OpenShift 上部署 JBoss EAP Java 应用，该控制器扩展 OpenShift API 以代表 OpenShift 用户创建、配置和管理复杂有状态应用的实例。

**其他资源**

- 如需有关 EAP 操作器的更多信息，请参阅 EAP Operator for Automating Application Deployment on OpenShift。

# 第 2 章 JBOSS EAP 8.0 的 PACKAGE NAMESPACE CHANGE

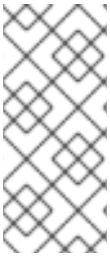
本节提供了 JBoss EAP 8.0 中用于 package 命名空间更改的额外信息。JBoss EAP 8.0 对 Jakarta EE 10 以及 Jakarta EE 10 API 的许多其他实现提供全面支持。用于 JBoss EAP 8.0 的 Jakarta EE 10 支持的重要更改是软件包命名空间更改。

## 2.1. JAVAX 到 JAKARTA 命名空间更改

Jakarta EE 8 和 EE 10 之间的关键区别在于，将 EE API Java 软件包从 **javax** 重命名为 **jakarta**
**prerequisites**。这遵循 Java EE 迁移到 Eclipse Foundation 并建立 Jakarta EE。

适应此命名空间更改是将应用程序从 JBoss EAP 7 迁移到 JBoss EAP 8 的最大任务。要将应用程序迁移到 Jakarta EE 10，您必须完成以下步骤：

- 将 **javax** 软件包中的 EE API 类的任何导入语句或其他源代码使用到 **jakarta** 软件包。
- 更新任何 EE 指定的系统属性或其他以 **javax** 开头的配置属性的名称，以从 **jakarta** 开始。
- 对于任何使用 **java.util.ServiceLoader** 机制启动的 EE 接口或抽象类，请将识别 META-INF/services/javax 的实施类从 **META-INF/services/javax.[rest_of_name]** 改为 **META-INF/services/jakarta.[rest_of_name]**。



**注意**

Red Hat Migration Toolkit 可帮助更新应用程序源代码中的命名空间。如需更多信息，请参阅 如何使用 Red Hat Migration Toolkit for Auto-Migration of a Application to the Jakarta EE 10 Namespace。如果源代码迁移不是一个选项，则 Open Source Eclipse Transformer 项目会提供字节代码转换工具，来将现有 Java 存档从 **javax** 命名空间转换为 **jakarta** 命名空间。

**注意**

这个更改不会影响作为 Java SE 一部分的 **javax** 软件包。

**其他资源**

- 如需更多信息，请参阅 javax 到 jakarta Package Namespace Change。

# 第 3 章 在 OPENSHIFT CONTAINER PLATFORM 上构建并运行 JBOSS EAP 应用程序

您可以按照 Source-to-image (S2I)流程在 JBoss EAP for OpenShift 镜像上构建并运行 Java 应用。

## 3.1. 先决条件

- 已安装并运行 OpenShift 实例。

## 3.2. 准备 OPENSHIFT 以部署应用程序

作为 JBoss EAP 应用程序开发人员，您可以在 OpenShift 上部署应用程序。在以下示例中，kitchensink Quickstart 演示了使用 Jakarta Server Faces、Jakarta Contexts 和 Dependency Injection、Jakarta Enterprise Beans、Jakarta Persistence 和 Jakarta Bean Validation 的 Jakarta EE Web-enabled 数据库应用。**如需更多信息，请参阅 JBoss EAP 8.0 kitchensink Quickstart。按照以下步骤部署应用程序。**

**流程**

1. **使用 oc login 命令登录到您的 OpenShift 实例。**

2. **在 OpenShift 中创建项目。**

   **使用以下命令创建项目：通过项目，您可以独立于其他组组织和管理内容。**

   ```
   $ oc new-project <project_name>
   ```

   **例如，对于 kitchensink Quickstart，使用以下命令创建一个名为 eap-demo 的项目：**

   ```
   $ oc new-project eap-demo
   ```

3. **可选 ：创建密钥存储和 secret。**

   **注意**

   **如果使用 OpenShift 项目中任何启用了 HTTPS 的功能，您必须创建密钥存储和 secret。**

   a. **使用 Java keytool 命令生成密钥存储：**

**警告**

**以下命令生成自签名证书，但在生产环境中，使用您自己的 SSL 证书从验证的证书认证机构(CA)用于 SSL 加密连接(HTTPS)。**

```
$ keytool -genkey -keyalg RSA -alias <alias_name> -keystore <keystore_filename.jks> -validity 360 -keysize 2048
```

**例如，对于 kitchensink Quickstart，请使用以下命令生成密钥存储：**

```
$ keytool -genkey -keyalg RSA -alias eapdemo-selfsigned -keystore keystore.jks -validity 360 -keysize 2048
```

b. **使用以下命令，从您的新密钥存储创建 secret：**

```
$ oc create secret generic <secret_name> --from-file=<keystore_filename.jks>
```

**例如，对于 kitchensink Quickstart，请使用以下命令创建一个 secret：**

```
$ oc create secret generic eap-app-secret --from-file=keystore.jks
```

**其他资源**

- **当在命名空间中添加 Dev Portal 生成的 secret 时，镜像流和模板无法拉取镜像**

### 3.3. 在 OPENSHIFT 中使用 SOURCE-TO-IMAGE 构建应用程序镜像

**按照 Source-to-image (S2I)工作流，为 JBoss EAP 应用构建可重复生成的容器镜像。这些生成的容器镜像包括应用部署和可随时运行 JBoss EAP 服务器。**

**S2I 工作流从 Git 存储库获取源代码，并将它注入到基于您要使用的语言和框架的容器中。在 S2I 工作流完成后，会编译 src 代码，应用将被打包并部署到 JBoss EAP 服务器。**

如需更多信息，请参阅 JBoss EAP S2I 的传统服务器置备。

注意

在 JBoss EAP 中，只有在使用 Jakarta EE 10 开发应用时，才能使用 S2I 镜像。

先决条件

- 您有一个有效的红帽客户帐户。
- 您有一个 Registry 服务帐户。按照红帽客户门户网站中的说明，使用 registry 服务帐户 创建身份验证令牌。
- 您已下载了 OpenShift secret YAML 文件，您可以使用该文件从 Red Hat Ecosystem Catalog 拉取镜像。如需更多信息，请参阅 OpenShift Secret。
- 您使用 oc login 命令登录到 OpenShift。
- 您已安装 Helm。如需更多信息，请参阅安装 Helm。
- 您已在管理 CLI 中输入以下命令来安装 JBoss EAP Helm chart 的存储库：

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

流程

1. 使用以下 YAML 内容，创建名为 helm.yaml 的文件：

```
build:
  uri: https://github.com/jboss-developer/jboss-eap-quickstarts.git
  ref: EAP_8.0.0.GA
  contextDir: helloworld
deploy:
  replicas: 1
```

2. 使用以下命令，将 JBoss EAP 应用部署到 OpenShift：

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

验证

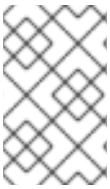
- 使用 curl 访问应用程序。

```
$ curl https://$(oc get route helloworld --template='{{ .spec.host }}')/HelloWorld
```

您将获得 Hello World! 输出，确认应用已经部署。

### 3.4. 在 OPENSHIFT 上部署第三方应用程序

您可以使用编译的 WAR 文件或 EAR 归档为 OpenShift 部署创建应用程序镜像。使用 Dockerfile 将这些存档部署到 JBoss EAP 服务器上，以及包括操作系统、Java 和 JBoss EAP 组件的更新和全面的运行时堆栈。



注意

红帽不提供预构建的 JBoss EAP 服务器镜像。

#### 3.4.1. 使用默认配置置备 JBoss EAP 服务器

您可以使用构建器镜像在 OpenShift 上安装和配置 JBoss EAP 服务器。对于无缝部署，请按照流程置备服务器、传输应用程序文件，并进行必要的自定义。

先决条件

- 您可以访问受支持的 Red Hat JBoss Enterprise Application Platform 容器镜像。例如：
  - registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8
  - registry.redhat.io/jboss-eap-8/eap8-openjdk17-runtime-openshift-rhel8
-

在您的系统上已安装了 podman。使用受支持的 RHEL 上可用的最新 podman 版本。如需更多信息，请参阅 Red Hat JBoss Enterprise Application Platform 8.0 支持的配置。

**流程**

1. 复制提供的以下 Dockerfile 内容：

```
# Use EAP 8 Builder image to create a JBoss EAP 8 server
# with its default configuration

FROM registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest AS
builder

# Set up environment variables for provisioning. (1)
ENV GALLEON_PROVISION_FEATURE_PACKS org.jboss.eap:wildfly-ee-galleon-
pack,org.jboss.eap.cloud:eap-cloud-galleon-pack
ENV GALLEON_PROVISION_LAYERS cloud-default-config
# Specify the JBoss EAP version (2)
ENV GALLEON_PROVISION_CHANNELS org.jboss.eap.channels:eap-8.0


# Run the assemble script to provision the server.
RUN /usr/local/s2i/assemble

# Copy the JBoss EAP 8 server from the builder image to the runtime image.
FROM registry.redhat.io/jboss-eap-8/eap8-openjdk17-runtime-openshift-rhel8:latest AS
runtime

# Set appropriate ownership and permissions.
COPY --from=builder --chown=jboss:root $JBOSS_HOME $JBOSS_HOME

# Steps to add:
# (1) COPY the WAR/EAR to $JBOSS_HOME/standalone/deployments
#       with the jboss:root user. For example:
#     COPY --chown=jboss:root my-app.war $JBOSS_HOME/standalone/deployments
(3)
# (2) (optional) server modification. You can modify EAP server configuration:
#
#       * invoke management operations. For example
#
#       RUN $JBOSS_HOME/bin/jboss-cli.sh --commands="embed-server,/system-
property=Foo:add(value=Bar)"
#
#       First operation must always be embed-server.
#
#       * copy a modified standalone.xml in $JBOSS_HOME/standalone/configuration/
#         for example
#
#     COPY --chown=jboss:root standalone.xml
$JBOSS_HOME/standalone/configuration
```

```
# Ensure appropriate permissions for the copied files.
RUN chmod -R ug+rwX $JBOSS_HOME
```

1. 您可以指定 MAVEN_MIRROR_URL 环境变量，这些变量供镜像内部的 JBoss EAP Maven 插件使用。如需更多信息，请参阅 Artifact repository mirrors。

2. 您不需要为任何次发行版本更新此 Dockerfile。如果要使用特定版本，请在 GALLEON_PROVISION_CHANNELS 环境变量中指定 JBoss EAP 版本。如需更多信息，请参阅 环境变量。

3. 修改复制的 Dockerfile，将 WAR 文件包含在容器中。例如：

```
COPY --chown=jboss:root <my-app.war> $JBOSS_HOME/standalone/deployments
```

将 <myapp.war> 替换为您要添加到镜像的 Web 存档的路径。

2. 使用 podman 构建应用程序镜像：

```
$ podman build -t my-app .
```

执行该命令后，my-app 容器镜像已准备好在 OpenShift 上部署。

3. 将容器镜像上传到以下选项之一：

- 可以从 OpenShift 访问的内部注册表。
- OpenShift registry 通过直接从构建的机器中推送镜像。如需更多信息，请参阅 RHOCP 4 中的如何将容器镜像推送到镜像 registry 中。

4. 从 registry 部署镜像时，请使用 Helm chart、Operator 或 Deployment 等部署策略。选择您首选的方法，并根据要求使用完整镜像 URL 或 ImageStreams。如需更多信息，请参阅使用

Helm chart 在 OpenShift 中构建和部署 JBoss EAP 应用程序。

## 3.5. 使用 OPENID CONNECT 来保护 OPENSHIFT 上的 JBOSS EAP 应用程序

使用 JBoss EAP 原生 OpenID Connect (OIDC)客户端来委托使用外部 OpenID 供应商的身份验证。OIDC 是一个身份层，它允许客户端（如 JBoss EAP）根据 OpenID 供应商执行的身份验证来验证用户的身份。

elytron-oidc-client 子系统和 elytron-oidc-client Galleon 层在 JBoss EAP 中提供原生 OIDC 客户端，以与 OpenID 供应商连接。JBoss EAP 根据您的 OpenID 提供程序配置自动为您的应用程序创建虚拟安全域。

您可以通过三种不同的方式配置 elytron-oidc-client 子系统：

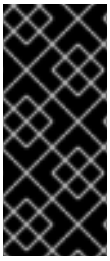
- 在部署中添加 oidc.json。
- 运行 CLI 脚本以配置 elytron-oidc-client 子系统。
- 定义环境变量，以在 OpenShift 上的 JBoss EAP 服务器启动时配置 elytron-oidc-client 子系统。

**注意**

此流程解释了如何使用环境变量配置 elytron-oidc-client 子系统以使用 OIDC 保护应用程序。

### 3.5.1. JBoss EAP 中的 OpenID Connect 配置

当您使用 OpenID 供应商保护应用程序时，您不需要在本地配置任何安全域资源。elytron-oidc-client 子系统在 JBoss EAP 中提供原生 OpenID Connect (OIDC)客户端，以与 OpenID 供应商连接。JBoss EAP 根据您的 OpenID 提供程序配置自动为您的应用程序创建虚拟安全域。



重要

将 OIDC 客户端与红帽构建的 Keycloak 搭配使用。如果可将其他 OpenID 供应商配置为使用 JSON Web 令牌(JWT)的访问令牌，并可配置为使用 RS256、RS384、RS512、ES256、ES384 或 ES512 签名算法。

要启用 OIDC 的使用，您可以配置 elytron-oidc-client 子系统或应用程序本身。JBoss EAP 激活 OIDC 身份验证，如下所示：

- 当您将应用程序部署到 JBoss EAP 时，elytron-oidc-client 子系统会扫描部署，以检测是否需要 OIDC 身份验证机制。
- 如果子系统在 elytron-oidc-client 子系统或应用程序部署描述符中检测到部署的 OIDC 配置，JBoss EAP 为应用启用 OIDC 身份验证机制。
- 如果子系统在两个位置检测到 OIDC 配置，则 elytron-oidc-client 子系统 secure-deployment 属性中的配置优先于应用程序部署描述符中的配置。

其他资源

- OpenID Connect 规格
- OpenID Connect Libraries
- 在红帽构建的 Keycloak 中使用 OpenID Connect 保护应用程序

### 3.5.2. 创建使用 OpenID Connect 保护的应用程序

要创建一个 web 应用，请创建一个具有所需依赖项和目录结构的 Maven 项目。创建一个 Web 应用程序，其中包含一个 servlet，它将返回从登录的用户主体和属性中获取的用户名。如果没有登录的用户，servlet 将返回文本"NO AUTHENTICATED USER"。

先决条件

- 您已安装了 Maven。如需更多信息，请参阅 下载 Apache Maven。

**流程**

1. **使用 mvn 命令建立一个 Maven 项目。该命令创建项目的目录结构以及 pom.xml 配置文件。**

   **语法**

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

   **Example**

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

2. **进入到应用程序根目录：**

   **语法**

```
$ cd <name-of-your-application>
```

   **Example**

```
$ cd simple-webapp-example
```

3. **将生成的 pom.xml 文件的内容替换为以下文本：**

```
<?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.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <version.maven.war.plugin>3.3.2</version.maven.war.plugin>
    <version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>
    <version.server>8.0.0.GA-redhat-00009</version.server>
    <version.bom.ee>${version.server}</version.bom.ee>
  </properties>


  <repositories>
    <repository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
```

```
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>

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

<dependencies>
    <dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.wildfly.security</groupId>
        <artifactId>wildfly-elytron-auth-server</artifactId>
    </dependency>
</dependencies>

<build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>${version.maven.war.plugin}</version>
        </plugin>
        <plugin>
            <groupId>org.jboss.eap.plugins</groupId>
            <artifactId>eap-maven-plugin</artifactId>
            <version>${version.eap.plugin}</version>
            <configuration>
                <channels>
                    <channel>
                        <manifest>
                            <groupId>org.jboss.eap.channels</groupId>
                            <artifactId>eap-8.0</artifactId>
                        </manifest>
                    </channel>
                </channels>
                <feature-packs>
                    <feature-pack>
                        <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>
            <layer>cloud-server</layer>
            <layer>elytron-oidc-client</layer>
          </layers>
          <galleon-options>
              <jboss-fork-embedded>true</jboss-fork-embedded>
          </galleon-options>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>package</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
</build>
</project>
```

**注意**

- **<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin> 是 JBoss EAP Maven 插件的示例版本。有关 JBoss EAP Maven 插件发行版本的更多信息，请参阅 Red Hat Maven 存储库：https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/。**

4. **创建一个用于存储 Java 文件的目录。**

**语法**

```
$ mkdir -p src/main/java/<path_based_on_artifactID>
```

**Example**

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

5. **前往新目录。**

**语法**

```
$ cd src/main/java/<path_based_on_artifactID>
```

**Example**

```
$ cd src/main/java/com/example/app
```

6. **使用以下内容创建一个 securedServlet.java 文件：**

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
import org.wildfly.security.authz.Attributes;
import org.wildfly.security.authz.Attributes.Entry;
/**
 * A simple secured HTTP servlet. It returns the user name and
```

```
 * attributes obtained from the logged-in user's Principal. If
 * there is no logged-in user, it returns the text
 * "NO AUTHENTICATED USER".
 */

@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

         Principal user = req.getUserPrincipal();
         SecurityIdentity identity =
SecurityDomain.getCurrent().getCurrentSecurityIdentity();
         Attributes identityAttributes = identity.getAttributes();
         Set <String> keys = identityAttributes.keySet();
         String attributes = "<ul>";

         for (String attr : keys) {
          attributes += "<li> " +  attr + " : " + identityAttributes.get(attr).toString() + "</li>";
         }

         attributes+="</ul>";
         writer.println("<html>");
         writer.println("  <head><title>Secured Servlet</title></head>");
         writer.println("  <body>");
         writer.println("    <h1>Secured Servlet</h1>");
         writer.println("    <p>");
         writer.print(" Current Principal '");
         writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
         writer.print("'");
         writer.print(user != null ? "\n" + attributes : "");
         writer.println("    </p>");
         writer.println("  </body>");
         writer.println("</html>");
        }
    }

}
```

7. **配置应用程序的 web.xml 以保护应用程序资源。**

**Example**

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
```

```
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  metadata-complete="false">

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secured</web-resource-name>
      <url-pattern>/secured</url-pattern>
    </web-resource-collection>

    <auth-constraint>
      <role-name>Users</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>OIDC</auth-method>
  </login-config>

  <security-role>
    <role-name>*</role-name>
  </security-role>
</web-app>
```

在本例中，只有具有角色的 用户才可以访问 应用程序。

### 3.5.3. 在 OpenShift 上部署应用程序

作为 JBoss EAP 应用程序开发人员，您可以在使用 OpenID Connect 子系统的 OpenShift 上部署应用程序，并将其与红帽构建的 Keycloak 服务器集成。按照以下步骤部署应用程序。

**先决条件**

您已在 OpenShift 中配置了带有以下配置的 Keycloak 服务器的红帽构建。如需更多信息，请参阅 红帽构建的 Keycloak Operator。

- 创建名为 JBossEAP 的域。
- 创建名为 demo 的用户。
- 为名为 demo 的用户设置密码。将 Temporary 切换到 OFF，然后单击 Set Password。在确认提示中，单击 Set password。

- 创建名为 **Users** 的角色。
- 将角色 用户分配给用户 **demo**。
- 在 **Client Roles** 字段中，选择您为 **JBoss EAP** 配置的 **realm-management**。
- 将角色 **create-client** 分配给客户端 **realm-management**。

**流程**

1. 将您的应用程序代码部署到 **Git** 存储库。
2. 创建包含 **OIDC** 配置的 **secret**。
   a. 使用以下内容创建名为 **oidc-secret.yaml** 的文件：

```
apiVersion: v1
kind: Secret
metadata:
  name: oidc-secret
type: Opaque
stringData:
  OIDC_PROVIDER_NAME: rh-sso
  OIDC_USER_NAME: demo
  OIDC_USER_PASSWORD: demo
  OIDC_SECURE_DEPLOYMENT_SECRET: mysecret
```

   b. 使用以下命令来创建 **secret**：

```
$ oc apply -f oidc-secret.yaml
```

3. 使用以下内容创建名为 **helm.yaml** 的文件：

```
build:
  uri: [URL TO YOUR GIT REPOSITORY]
```

```
deploy:
  envFrom:
 - secretRef:
      name: oidc-secret
```

4. **使用 JBoss EAP Helm chart 部署示例应用程序：**

```
$ helm install eap-oidc-test-app -f helm.yaml jboss-eap/eap8
```

5. **将环境变量添加到 oidc-secret.yaml 文件中，以配置 OIDC 供应商 URL 和应用程序主机名。**

```
yaml
stringData:
  ...
  OIDC_HOSTNAME_HTTPS: <host of the application>
  OIDC_PROVIDER_URL: https://<host of the SSO provider>/realms/JBossEAP
```

**OIDC_HOSTNAME_HTTPS 的值与以下输出对应：**

```
echo $(oc get route eap-oidc-test-app --template='{{ .spec.host }}')
```

**OIDC_PROVIDER_URL 的值与以下输出对应：**

```
echo https://$(oc get route sso --template='{{ .spec.host }}')/realms/JBossEAP
```

**如果没有设置 OIDC_HOSTNAME_HTTP (S)，则会进行路由发现尝试。若要启用路由发现，OpenShift 用户必须能够列出 路由资源。例如，要创建和将 routeview 角色与 view 用户关联，请使用以下 oc 命令：**

```
$ oc create role <role-name> --verb=list --resource=route

$ oc adm policy add-role-to-user <role-name> <user-name> --role-namespace=<your
namespace>
```

6. **使用 oc apply -f oidc-secret.yaml 更新 secret。**

7. **再次部署应用程序以确保 OpenShift 使用新的环境变量：**

```
$ oc rollout restart deploy eap-oidc-test-app
```

-

验证

1. 在浏览器中，进入到 **https://<eap-oidc-test-app route>/**。

   您将被重定向到红帽构建的 **Keycloak** 登录页面。

2. 访问受保护的 **servlet**。

3. 使用以下凭证登录：

   ```
   username: demo
   password: demo
   ```

   此时会出现包含主体 **ID** 的页面。

### 3.5.4. 基于环境变量的配置

使用这些环境变量在 **OpenShift** 镜像上配置 **JBoss EAP OIDC** 支持。

**表 3.1.** 环境变量

| 环境变量 | 旧的 SSO 环境变量 | 描述 | 必填 | 默认值 |
|---|---|---|---|---|
| OIDC_PROVIDER_NAME | NONE.当使用 **SSO** Passthrough 环境变量时，会在内部设置 "rh-sso" 名称。 | 在使用 OIDC_PROVIDER_NAME 变量时，您必须设置为 **rh-sso**。 | 是 | |
| OIDC_PROVIDER_URL | **$SSO_URL/realms/$SSO_REALM** | 供应商的 URL。 | 是 | |
| OIDC_USER_NAME | SSO_USERNAME | 动态客户端注册需要用户名才能接收令牌。 | 是 | |
| OIDC_USER_PASSWORD | SSO_PASSWORD | 动态客户端注册需要用户密码才能接收令牌。 | 是 | |

| 环境变量 | 旧的 SSO 环境变量 | 描述 | 必填 | 默认值 |
| --- | --- | --- | --- | --- |
| OIDC_SECURE_DEPLOYMENT_SECRET | SSO_SECRET | secure-deployment 子系统和身份验证服务器客户端都知道。 | 否 | |
| OIDC_SECURE_DEPLOYMENT_PRINCIPAL_ATTRIBUTE | SSO_PRINCIPAL_ATTRIBUTE | 配置主体名称的值。 | 否 | **rh-sso** 的默认子 (ID 令牌)。<br>典型的值：preferred_username。 |
| OIDC_SECURE_DEPLOYMENT_ENABLE_CORS | SSO_ENABLE_CORS | 为单点登录应用程序启用 CORS。 | 否 | 默认值为 **False**。 |
| OIDC_SECURE_DEPLOYMENT_BEARER_ONLY | SSO_BEARER_ONLY | 仅接受 bearer 令牌的部署，不支持日志记录。 | 否 | 默认值为 **False**。 |
| OIDC_PROVIDER_SSL_REQUIRED | NONE | 默认为 external，如私有和本地地址，但不支持 https。 | 否 | 外部 |
| OIDC_PROVIDER_TRUSTSTORE | SSO_TRUSTSTORE | 指定 realm **trustore** 文件。如果没有设置，则适配器在处理 HTTPS 请求时无法使用信任管理器。 | 否 | |
| OIDC_PROVIDER_TRUSTSTORE_DIR | SSO_TRUSTSTORE_DIR | 查找 realm **truststore** 的目录。如果没有设置，则适配器在处理 HTTPS 请求时无法使用信任管理器。 | 否 | |
| OIDC_PROVIDER_TRUSTSTORE_PASSWORD | SSO_TRUSTSTORE_PASSWORD | 指定 realm **truststore** 密码。如果没有设置，则适配器在处理 HTTPS 请求时无法使用信任管理器。 | 否 | |
| OIDC_PROVIDER_TRUSTSTORE_CERTIFICATE_ALIAS | SSO_TRUSTSTORE_CERTIFICATE_ALIAS | 指定 realm **trustore** 别名。需要与身份验证服务器交互才能注册客户端。 | 否 | |

| 环境变量 | 旧的 SSO 环境变量 | 描述 | 必填 | 默认值 |
| --- | --- | --- | --- | --- |
| OIDC_DISABLE_SSL_CERTIFICATE_VALIDATION | SSO_DISABLE_SSL_CERTIFICATE_VALIDATION | 与身份验证服务器交互时禁用证书验证，以注册客户端。 | 否 | |
| OIDC_HOSTNAME_HTTP | HOSTNAME_HTTP | 用于非安全路由的主机名。 | 否 | 发现路由。 |
| OIDC_HOSTNAME_HTTPS | HOSTNAME_HTTPS | 用于安全路由的主机名。 | 否 | 发现安全路由。 |
| NONE | SSO_PUBLIC_KEY | Single Sign-On 域的公钥。使用这个选项，公钥由 OIDC 子系统自动检索。 | 否 | 如果设置，则会显示警告信息被忽略。 |

### 3.6. 使用 SAML 保护应用程序

**安全断言标记语言(SAML)充当数据格式和协议，支持在双方之间交换身份验证和授权信息。这两方通常包括身份提供程序和服务提供商。此信息采用包含断言的 SAML 令牌的形式。身份提供程序会发出这些 SAML 令牌，以限制这些主题与服务提供商进行身份验证。主题可将 SAML 令牌与多个服务提供商重复使用，可在 SAML v2 中启用基于浏览器的单点登录。**

**您可以使用 Keycloak SAML 适配器功能 pack 提供的 Galleon 层来保护 web 应用程序。**

**有关 Keycloak SAML 适配器功能 pack 的详情，请参考 Keycloak SAML 适配器功能 pack 来使用 SAML 保护应用程序。**

#### 3.6.1. 使用 SAML 保护应用程序的 Keycloak SAML 适配器功能 pack

**Keycloak SAML 适配器 Galleon pack 是一个 Galleon 功能软件包，其中包括 keycloak-saml 层。使用功能软件包中的 keycloak-saml 层在 JBoss EAP 中安装必要的模块和配置。如果要使用 SAML，则需要使用红帽构建的 Keycloak 作为 Single Sign-On (SSO)的身份供应商。当为 source-to-image (S2I)使用 keycloak-saml SAML 适配器 Galleon 层时，您可以选择使用 SAML 客户端功能来自动注册 Identity Service Provider (IDP)，如红帽构建的 Keycloak。**

#### 3.6.2. 将红帽构建的 Keycloak 配置为 OpenShift 的 SAML 供应商

**Red Hat build of Keycloak 是一个身份和访问管理供应商，用于使用单点登录(SSO)保护 Web 应用程序。它支持 OpenID Connect，它是 OAuth 2.0 和 SAML 的扩展。**

以下流程概述了使用 SAML 保护应用程序所需的基本步骤。如需更多信息，请参阅 红帽构建的 Keycloak 文档。

**先决条件**

- 具有红帽构建的 Keycloak 的管理员访问权限。
- 红帽构建的 Keycloak 正在运行。如需更多信息，请参阅 红帽构建的 Keycloak Operator。
- 您使用 oc login 命令登录到 OpenShift。

**流程**

1. 创建单点登录域、用户和角色。
2. 使用 Java keytool 命令生成密钥和证书：

```
keytool -genkeypair -alias saml-app -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -storepass password -dname "CN=saml-basic-auth,OU=EAP SAML Client,O=Red Hat EAP QE,L=MB,S=Milan,C=IT" -ext ku:c=dig,keyEncipherment -validity 365
```

3. 将密钥存储导入到 Java KeyStore (JKS)格式：

```
keytool -importkeystore -deststorepass password -destkeystore keystore.jks -srckeystore keystore.p12 -srcstoretype PKCS12 -srcstorepass password
```

4. 在 OpenShift 中为密钥存储创建一个 secret：

```
$ oc create secret generic saml-app-secret --from-file=keystore.jks=./keystore.jks --type=opaque
```

**注意**

只有在使用自动 SAML 客户端注册功能时，才需要这些步骤。当 JBoss EAP 以 client-admin 用户身份将新的 SAML 客户端注册到 Keycloak 的红帽构建中时，JBoss EAP 必须将新 SAML 客户端的证书存储在红帽构建的 Keycloak 客户端配置中。这使得 JBoss EAP 能够保持私钥，同时将公共证书存储在红帽构建的 Keycloak 中，后者建立了一个经过身份验证的客户端，用于与红帽构建的 Keycloak 通信。

### 3.6.3. 创建使用 SAML 保护的应用程序

您可以使用安全断言标记语言(SAML)增强 Web 应用程序安全性。SAML 提供有效的用户身份验证和授权，以及单点登录(SSO)功能，使其成为增强 Web 应用程序的可靠选择。

**先决条件**

- 您已安装了 Maven。如需更多信息，请参阅 下载 Apache Maven。

**流程**

1. 使用 mvn 命令设置 Maven 项目。此命令为项目和 pom.xml 配置文件创建目录结构。

**语法**

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

**Example**

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
```

```
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

2. **进入到应用程序根目录：**

**语法**

```
$ cd <name-of-your-application>
```

**Example**

```
$ cd simple-webapp-example
```

3. **将生成的 pom.xml 文件的内容替换为以下文本：**

```
<?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.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
```

```
  <maven.compiler.target>11</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <version.maven.war.plugin>3.3.2</version.maven.war.plugin>
  <version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>
  <version.server>8.0.0.GA-redhat-00009</version.server>
  <version.bom.ee>${version.server}</version.bom.ee>
</properties>

<repositories>
  <repository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>

<pluginRepositories>
  <pluginRepository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </pluginRepository>
</pluginRepositories>

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

<dependencies>
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.wildfly.security</groupId>
    <artifactId>wildfly-elytron-auth-server</artifactId>
  </dependency>
</dependencies>

<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
```

```
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>${version.maven.war.plugin}</version>
        </plugin>
        <plugin>
            <groupId>org.jboss.eap.plugins</groupId>
            <artifactId>eap-maven-plugin</artifactId>
            <version>${version.eap.plugin}</version>
            <configuration>
                <channels>
                    <channel>
                        <manifest>
                            <groupId>org.jboss.eap.channels</groupId>
                            <artifactId>eap-8.0</artifactId>
                        </manifest>
                    </channel>
                </channels>
                <feature-packs>
                    <feature-pack>
                        <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-pack>
                        <location>org.keycloak:keycloak-saml-adapter-galleon-pack</location>
                    </feature-pack>
                </feature-packs>
                <layers>
                    <layer>cloud-server</layer>
                    <layer>keycloak-saml</layer>
                </layers>
                <galleon-options>
                    <jboss-fork-embedded>true</jboss-fork-embedded>
                </galleon-options>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>package</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
  </build>
</project>
```

**注意**

- **<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin> 是 JBoss EAP Maven 插件的示例版本。有关 JBoss EAP Maven 插件发行版本的更多信息，请参阅 Red Hat Maven 存储库：https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/。**

4. **创建一个用于存储 Java 文件的目录。**

**语法**

```
$ mkdir -p src/main/java/<path_based_on_artifactID>
```

**Example**

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

5. **前往新目录。**

**语法**

```
$ cd src/main/java/<path_based_on_artifactID>
```

**Example**

```
$ cd src/main/java/com/example/app
```

6. 创建名为 **SecuredServlet.java** 的文件，其中包含以下设置：

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;
import java.util.Set;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
import org.wildfly.security.authz.Attributes;
/**
 * A simple secured HTTP servlet. It returns the user name and
 * attributes obtained from the logged-in user's Principal. If
 * there is no logged-in user, it returns the text
 * "NO AUTHENTICATED USER".
 */

@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

            Principal user = req.getUserPrincipal();
            SecurityIdentity identity =
SecurityDomain.getCurrent().getCurrentSecurityIdentity();
            Attributes identityAttributes = identity.getAttributes();
            Set <String> keys = identityAttributes.keySet();
            String attributes = "<ul>";

            for (String attr : keys) {
             attributes += "<li> " +  attr + " : " + identityAttributes.get(attr).toString() + "</li>";
            }

            attributes+="</ul>";
            writer.println("<html>");
            writer.println("  <head><title>Secured Servlet</title></head>");
            writer.println("  <body>");
            writer.println("    <h1>Secured Servlet</h1>");
```

```
        writer.println("    <p>");
        writer.print(" Current Principal '");
        writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
        writer.print("'");
        writer.print(user != null ? "\n" + attributes : "");
        writer.println("    </p>");
        writer.println("  </body>");
        writer.println("</html>");
       }
    }

}
```

7. 为 **web.xml 文件创建目录结构：**

```
mkdir -p src/main/webapp/WEB-INF
cd src/main/webapp/WEB-INF
```

8. **配置应用程序的 web.xml 文件来保护应用程序资源。**

**Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>secured</web-resource-name>
            <url-pattern>/secured</url-pattern>
        </web-resource-collection>

        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>KEYCLOAK-SAML</auth-method>
    </login-config>

    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

在本例中，只有具有 user 角色的用户才能访问应用。

验证

创建应用后，将其提交到远程 Git 存储库。

1. 创建 Git 存储库，如 https://github.com/your-username/simple-webapp-example。有关远程存储库和 Git 的更多信息，请参阅 Git 入门 - 关于远程存储库。

2. 在应用程序的根目录中运行以下 Git 命令：

```
git init -b main
git add pom.xml src
git commit -m "First commit"
git remote add origin git@github.com:your-username/simple-webapp-example.git
git remote -v
git push -u origin main
```

这些步骤将您的应用程序提交到远程存储库，使它能在线访问。

### 3.6.4. 在 OpenShift 中构建和部署 SAML 保护的应用程序

您可以使用 JBoss EAP 和单点登录(SSO) Galleon 层在 OpenShift 上构建和部署使用 SAML 保护的应用程序。

先决条件

- 您已安装 Helm。如需更多信息，请参阅安装 Helm。
- 您已创建了 SAML 应用程序项目，并使其在 Git 存储库中访问。
- 您已在管理 CLI 中输入以下命令来安装 JBoss EAP Helm chart 的存储库：

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

-

**流程**

1. 将您的应用程序代码部署到 Git 存储库。

2. 创建包含所需环境变量的 OpenShift secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: saml-secret
type: Opaque
stringData:
  SSO_REALM: "saml-basic-auth"
  SSO_USERNAME: "client-admin"
  SSO_PASSWORD: "client-admin"
  SSO_SAML_CERTIFICATE_NAME: "saml-app"
  SSO_SAML_KEYSTORE: "keystore.jks"
  SSO_SAML_KEYSTORE_PASSWORD: "password"
  SSO_SAML_KEYSTORE_DIR: "/etc/sso-saml-secret-volume"
  SSO_SAML_LOGOUT_PAGE: "/simple-webapp-example"
  SSO_DISABLE_SSL_CERTIFICATE_VALIDATION: "true"
```

3. 将提供的 YAML 内容保存到文件中，如 saml-secret.yaml。

4. 使用以下命令应用保存的 YAML 文件：

```
oc apply -f saml-secret.yaml
```

5. 创建名为 helm.yaml 的文件，其中包含以下设置：

```
build:
  uri: [WEB ADDRESS TO YOUR GIT REPOSITORY]
deploy:
  volumes:
    - name: saml-keystore-volume
      secret:
        secretName: saml-app-secret
  volumeMounts:
    - name: saml-keystore-volume
      mountPath: /etc/sso-saml-secret-volume
      readOnly: true
  envFrom:
    - secretRef:
        name: saml-secret
```

**注意**

以 HTTP 格式指定 Web 地址，如 http://www.redhat.com。如果使用 maven 镜像，请按如下所示指定 Web 地址：

```
build:
  uri: [WEB ADDRESS TO YOUR GIT REPOSITORY]
  env:
    - name: "MAVEN_MIRROR_URL"
      value: "http://..."
```

6. 使用 JBoss EAP Helm chart 部署示例应用程序：

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

7. 将环境变量添加到 saml-secret.yaml 文件中，以配置 Keycloak 服务器 URL 和应用程序路由：

```
stringData:
  ...
  HOSTNAME_HTTPS: <saml-app application route>
  SSO_URL: https://<host of the Keycloak server>
```

将 <saml-app 应用程序 route& gt ; 和 <host of the Keycloak server > 替换为适当的值。

HOSTNAME_HTTPS 的值与以下输出对应：

```
echo $(oc get route saml-app --template='{{ .spec.host }}')
```

SSO_URL 的值对应于以下输出：

```
echo https://$(oc get route sso --template='{{ .spec.host }}')
```

**注意**

如果无法使用这个命令，请使用 oc get routes 来列出可用的路由，并选择到红帽构建的 Keycloak 实例的路由。

8. 使用 oc apply -f saml-secret.yaml 更新 secret。

验证

1. 再次部署应用程序，以确保 OpenShift 使用新的环境变量：

```
$ oc rollout restart deploy saml-app
```

2. 在浏览器中，导航到应用 URL。例如，https://<saml-app route>/simple-webapp-example。

   您将被重定向到红帽构建的 Keycloak 登录页面。

3. 要获取 Web 地址，请使用以下命令访问安全 servlet：

```
echo https://$(oc get route saml-app --template='{{ .spec.host }}')/simple-webapp-example/secured
```

4. 使用以下凭证登录：

```
username: demo
password: demo
```

   此时会显示包含主体 ID 的页面。

您的应用程序现在使用 SAML 进行保护。

### 3.6.5. 创建 SSO 域、用户和角色

您可以在红帽构建的 Keycloak 环境中配置单点登录(SSO)域，定义用户角色并管理访问控制。这些操作可让您增强安全性并简化用户访问管理，确保简化的身份验证体验。这对优化 SSO 设置并改进用户身份验证流程至关重要。

**先决条件**

- 具有红帽构建的 Keycloak 的管理员访问权限。
- 红帽构建的 Keycloak 正在运行。

**流程**

1. 使用 URL 登录红帽 Keycloak 管理控制台： https://<SSO route>/。
2. 在红帽构建的 Keycloak 中创建一个域，例如 saml-basic-auth。然后，您可以使用此域来创建所需的用户、角色和客户端。

   如需更多信息，请参阅创建域。
3. 在 saml-basic-auth 域中创建一个角色。例如，用户。

   如需更多信息，请参阅创建 realm 角色。
4. 创建用户。例如，demo。

   如需更多信息，请参阅 创建用户。
5. 为用户创建密码。例如，demo。

   确保密码不是临时密码。如需更多信息，请参阅 为用户 设置密码。
6. 将 user 角色分配给用于登录访问的 demo 用户。

   如需更多信息，请参阅 分配角色映射。
7. 创建用户。例如，client-admin。

   要在 JBoss EAP 服务器启动时在 Keycloak 服务器中创建 SAML 客户端，您可以使用 client-admin 用户，这需要额外的特权。如需更多信息，请参阅 创建用户。

8. 为用户创建密码。例如，client-admin。

   确保密码不是临时密码。如需更多信息，请参阅 为用户 设置密码。

9. 从 Client Roles 下拉列表中选择 realm-management。

10. 将角色 create-client、management-clients 和 manage-realm 分配给 client-admin 用户。

    如需更多信息，请参阅 分配角色映射。

### 3.6.6. 用于配置 SAML 子系统的环境变量

您可以通过了解并使用以下变量来优化环境中 Keycloak 服务器的集成。这样可确保应用程序的无缝和安全 Keycloak 设置。

**表 3.2.** 环境变量

| 环境变量 | 描述 | 必填 |
| --- | --- | --- |
| **APPLICATION_NAME** | 用作客户端名称的前缀，派生自部署名称。 | 选填 |
| **HOSTNAME_HTTP** | HTTP OpenShift 路由的自定义主机名。**如果没有设置，则执行路由发现。** | 选填 |
| **HOSTNAME_HTTPS** | HTTPS OpenShift 路由的自定义主机名。**如果没有设置，则执行路由发现。** | 选填 |
| **SSO_DISABLE_SSL_CERTIFICATE_VALIDATION** | 选择 **true** 或 **false** 来启用或禁用 Keycloak 服务器证书验证。当 SSO 服务器生成自签名证书时，请考虑将其设置为 **true**。 | 选填 |
| **SSO_PASSWORD** | 具有与 Keycloak 域交互的具有权限的用户的密码，并创建和注册客户端。例如，**client-admin**。 | True |

| 环境变量 | 描述 | 必填 |
| --- | --- | --- |
| **SSO_REALM** | 用于关联应用客户端的 SSO 域。例如，**saml-basic-auth**。 | 选填 |
| **SSO_SAML_CERTIFICATE_NAME** | SAML 客户端密钥存储中的私钥和证书的别名。例如： **saml-app**。 | True |
| **SSO_SAML_KEYSTORE** | 密钥存储文件的名称。例如，store **.jks**。 | True |
| **SSO_SAML_KEYSTORE_DIR** | 包含客户端密钥存储的目录。例如： **/etc/sso-saml-secret-volume**。 | True |
| **SSO_SAML_KEYSTORE_PASSWORD** | 密钥存储密码。例如，**密码**。 | True |
| **SSO_SAML_LOGOUT_PAGE** | 注销页面。例如，**simple-webapp-example**。 | True |
| **SSO_SAML_VALIDATE_SIGNATURE** | 指定 **true** 来验证签名或 **false** 以不验证它。默认情况下为 true。 | 选填 |
| **SSO_SECURITY_DOMAIN** | 用于保护 undertow 和 **ejb** 子系统的安全域的名称。默认为 **keycloak**。 | 选填 |
| **SSO_TRUSTSTORE** | 包含服务器证书的 truststore 文件名。 | 选填 |
| **SSO_TRUSTSTORE_CERTIFICATE_ALIAS** | truststore 中的证书别名。 | 选填 |
| **SSO_TRUSTSTORE_DIR** | 包含信任存储的目录。 | 选填 |
| **SSO_TRUSTSTORE_PASSWORD** | truststore 和 certificate 的密码。例如，my **keystorepass**。 | 选填 |
| **SSO_URL** | SSO 服务器的 URL。例如，&lt **;SSO 服务器可访问的 route>**。 | True |
| **SSO_USERNAME** | 具有与 Keycloak 域交互以及创建和注册客户端的特权的用户名。例如，**client-admin**。 | True |

### 3.6.7. JBoss EAP 服务器中的路由发现

**您可以使用 JBoss EAP 服务器中的路由发现功能优化服务器的性能并简化指定命名空间中的路由配置。此功能对于提高服务器效率至关重要，以提供更顺畅的操作体验，特别是在未指定 HOSTNAME_HTTPS 变量时。**

如果没有设置 HOSTNAME_HTTPS 变量，JBoss EAP 服务器会自动尝试路由发现。要启用路由发现，您必须创建所需的权限：

```
oc create role routeview --verb=list --resource=route -n YOUR_NAME_SPACE
oc policy add-role-to-user routeview system:serviceaccount:YOUR_NAME_SPACE:default --
role-namespace=YOUR_NAME_SPACE -n YOUR_NAME_SPACE
```

#### 3.6.8. 其他资源

- 红帽构建的 Keycloak 服务器管理指南

### 3.7. 其他资源

- OpenShift Container Platform 入门

# 第 4 章 使用 HELM CHART 在 OPENSHIFT 中构建和部署 JBOSS EAP 应用程序

Helm 是一个开源软件包管理器，允许您在 OpenShift 上构建、部署和维护 JBoss EAP 应用程序。在 JBoss EAP 8.0 中，Helm chart 替换 OpenShift 模板。

## 4.1. HELM CHART 用例

您可以将 Helm chart 与 JBoss EAP 8.0 搭配使用：

- 使用 OpenShift Source-to-Image (S2I)，从 Git 存储库托管的 Maven 项目构建应用程序。
- 在 OpenShift 上部署应用程序镜像，与 OpenShift 集群进行深度集成(TLS 配置、公共路由以公开应用等)。
- 使用 Helm Chart 构建应用程序镜像，并使用 JBoss EAP operator 部署镜像。
- 使用其他方法为 JBoss EAP 构建应用程序镜像，并使用 Helm Chart 部署应用程序镜像。

## 4.2. OPENSHIFT 上 JBOSS EAP 的 HELM CHART 自定义

您可以通过修改包含应用程序特定设置的 YAML 文件来自定义 JBoss EAP 应用程序的 Helm Chart。

在 YAML 文件中，有两个主要部分：

- 构建配置。
- 部署配置。

通过 YAML 视图 选择配置，您可以直接在 OpenShift Development 控制台中编辑 Values 文件，以使用更新的配置升级 Helm 发行版本。

其他资源

- JBoss EAP 8 上的 Helm chart
- 值文件示例

### 4.3. 使用 S2I 置备 JBOSS EAP

使用应用 pom.xml 中的 eap-maven-plugin 来调配您的 JBoss EAP 服务器。

注意

build.s2i.featurePacks,build.s2i.galleonLayers 和 build.s2i.channels 字段已弃用。

### 4.4. 使用 HELM CHART 构建和部署 JBOSS EAP 应用程序

您可以通过配置 构建和部署 值，使用 Helms Chart 构建 JBoss EAP 应用程序。您必须为 Git 存储库提供在构建配置中托管应用程序代码的 Git 存储库的 URL，输出是一个包含构建的应用程序镜像的 ImageStreamTag 资源。

要部署应用程序，您必须提供包含构建的应用程序镜像的 ImageStreamTag 资源。输出是部署的应用和其他相关资源，可用于从 OpenShift 内部和外部访问您的应用。

先决条件

- 您已登录到 OpenShift Development 控制台。
- 您有 JBoss EAP 应用托管在 Git 存储库中。
- 您的应用程序是一个 Maven 项目
- 您已将应用配置为使用 org.jboss.eap.plugins:eap-maven-plugin 来调配 JBoss EAP 8.0 服务器。

流程

1. **从源存储库构建应用程序镜像：**

```
build:
  uri: <git repository URL of your application>
```

2. **可选：在 build 部分输入 secret：**

```
build:
  sourceSecret: <name of secret login to your Git repository>
```

验证

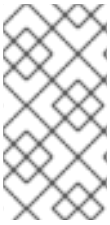
- **如果应用程序已被成功部署，您应该会在 OpenShift Development Console 上的 Helm 发行版本旁边看到部署的徽标。**

**其他资源**

- **使用 Maven 插件调配 JBoss EAP 服务器**

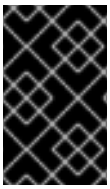
### 4.5. 使用 OPENSHIFT DEVELOPMENT 控制台构建应用程序镜像

**您可以通过在 OpenShift 开发控制台上配置 build 部分，使用 Helm Chart 构建 JBoss EAP 应用程序镜像。**



**注意**

**如果应用程序镜像由另一个机制构建，您可以通过将 build.enabled 字段设置为 false 来跳过 Helm Chart 的构建部分。**



**重要**

**您必须使用引用 Git 存储库的 Git URL 指定 build.url 字段。**

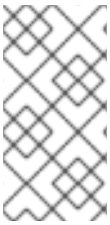
**其他资源**

- **JBoss EAP 8.0 的 Helm chart。**

- 构建应用程序镜像。

### 4.6. 部署应用程序镜像

您可以通过在 OpenShift 开发控制台上配置 deploy 设置，使用 Helm Chart 部署 JBoss EAP 应用程序。



**注意**

如果使用其他机制构建应用程序镜像，您可以通过将 build.deploy 字段设置为 false 来跳过 Helm Chart 的部署配置。

**其他资源**

- OpenShift 开发控制台快速入门.
- 部署应用程序镜像。

#### 4.6.1. 用于 Helm Chart 中的持久性存储的 OpenShift 卷

OpenShift 卷使容器能够存储和共享来自各种来源的数据，包括云存储、网络文件系统(NFS)或主机机器。您可以使用 Helm Chart (OpenShift 软件包管理器)以一致且可重复生成的方式部署应用程序。通过将卷挂载添加到 Helm Chart 中，您可以让应用程序在部署间保留数据。

#### 4.6.2. 使用 Helm chart 挂载卷

此流程解释了如何在 JBoss EAP 8.0 上使用 Helm chart 将 secret 挂载为卷。另外，您还可以使用它来挂载 ConfigMap。此操作可让应用程序安全访问和使用数据，防止它被未经授权的访问或篡改。

例如，通过将机密挂载为卷，存储在机密中的敏感数据将显示为 POD 中挂载该机密的部署中的文件。

**先决条件**

- 您已创建了一个 secret。例如，您已创建了名为 eap-app-secret 的机密，该机密引用 keystore.jks 等文件。

- 您已找出在容器文件系统中挂载 secret 的位置。例如，目录 /etc/jgroups-encrypt-secre-secret-volume 是挂载 secret 文件的位置，如 keystore.jks。

**流程**

1. 在 deploy.volumes 字段中指定卷，并配置要使用的 secret。您必须提供 卷名称 和 secret 的 secretName ：

```
volumes:
 - name: eap-jgroups-keystore-volume
   secret:
     secretName: eap-app-secret
```

2. 在部署配置中使用 deploy.volumeMounts 挂载卷：

```
volumeMounts:
 - name: eap-jgroups-keystore-volume
   mountPath: /etc/jgroups-encrypt-secret-volume
   readOnly: true
```

当 pod 启动时，容器将 keystore.jks 文件挂载到 /etc/jgroups-encrypt-secret-volume/keystore.jks 位置。

**其他资源**

- 卷(Kubernetes 文档)

# 第 5 章 环境变量和模型表达式解析

## 5.1. 先决条件

- 您对如何在操作系统上配置环境变量的一些基本知识。
- 要在 OpenShift Container Platform 中配置环境变量，您必须满足以下先决条件：
  - 已安装 OpenShift 并设置 OpenShift CLI ("oc")。如需有关 oc 的更多信息，请参阅 OpenShift CLI 入门。
  - 您已使用 Helm Chart 将应用程序部署到 OpenShift。如需有关 Helm chart 的更多信息，请参阅 JBoss EAP 的 Helm Charts。

## 5.2. 用于解析管理模型表达式的环境变量

要解决管理模型表达式并在 OpenShift Container Platform 上启动 JBoss EAP 8.0 服务器，您可以在管理命令行界面(CLI)中添加环境变量或设置 Java 系统属性。如果您同时使用两者，JBoss EAP 会观察并使用 Java 系统属性，而非环境变量来解析管理模型表达式。

### 系统属性到环境变量映射

假设您具有此管理表达式 ：${my.example-expr}。当您的 JBoss EAP 服务器尝试解析它时，它会检查名为 my.example-expr 的系统属性。

- 如果您的服务器找到此属性，它将使用其值来解析表达式。
- 如果找不到此属性，您的服务器将继续搜索。

接下来，假设您的服务器找不到系统属性 my.example-expr，它将 my.example-expr 自动更改为所有大写字母，并将不是字母数字的所有字符替换为下划线(_)： MY_EXAMPLE_EXPR。然后，JBoss EAP 会检查具有该名称的环境变量。

- 如果您的服务器找到此变量，它将使用其值来解析表达式。

- 如果找不到此变量，您的服务器将继续搜索。

提示

如果您的原始表达式以前缀 env 开头。 JBoss EAP 通过删除前缀来解析环境变量，然后只查找环境变量名称。例如，对于表达式 env.example，JBoss EAP 会查找 示例 环境变量。

如果这些检查找不到用于解析您的原始表达式的属性或变量，JBoss EAP 会查找表达式是否具有默认值。如果这样做，则默认值会解析表达式。如果没有，则 JBoss EAP 无法解析表达式。

两台服务器的示例

假设在一个服务器上，JBoss EAP 定义此管理资源： < socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:0}">.要运行具有不同端口偏移的第二个服务器，而不是编辑配置文件，请执行以下操作之一：

- 设置 jboss.socket.binding.port-offset Java 系统属性，以解析第二服务器上的值：./standalone.sh -Djboss.socket.binding.port-offset=100。

- 设置 JBOSS_SOCKET_BINDING_PORT_OFFSET 环境变量，以解析第二服务器上的值：JBOSS_SOCKET_BINDING_PORT_OFFSET=100 ./standalone.sh.

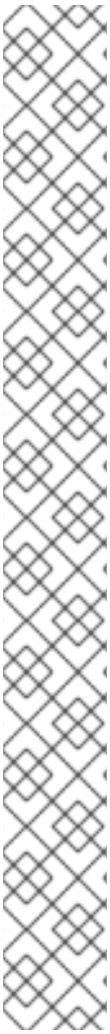
### 5.3. 在 OPENSHIFT CONTAINER PLATFORM 中配置环境变量

使用 JBoss EAP 8.0，您可以配置环境变量来解析管理模型表达式。您还可以使用环境变量来调整您在 OpenShift 上运行的 JBoss EAP 服务器的配置。

在使用 pod 模板的资源上设置环境变量和选项：

```
$ oc set env <object-selection> KEY_1=VAL_1 ... KEY_N=VAL_N [<set-env-options>] [<common-options>]
```

| 选项 | 描述 |
| --- | --- |
| **-e, --env=<KEY>=<VAL>** | 设置给定环境变量的键值对。 |
| **--overwrite** | 确认更新现有环境变量。 |



注意

使用 pod 模板的 Kubernetes 工作负载资源包括：

- Deployment
- ReplicaSet
- StatefulSet
- DaemonSet
- 作业
- CronJob

配置环境变量后，JBoss EAP 管理控制台应在其相关 Pod 的详细信息中显示它们。

其他资源

- 关于 OpenShift CLI
- Red Hat JBoss Enterprise Application Platform 配置指南

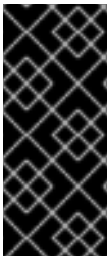
### 5.4. 使用环境变量覆盖管理属性

您知道，您可以使用 Java 系统属性或环境变量解析通过表达式定义的管理属性，但您也可以修改其他属性，即使它们不使用表达式。

为了更轻松地将 JBoss EAP 服务器配置适应您的服务器环境，您可以使用环境变量覆盖任何管理属性的值，而无需编辑配置文件。此功能（从 JBoss EAP 版本 8.0 开始）可用于以下原因：

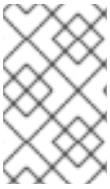
- JBoss EAP 仅为其最常见的管理属性提供表达式。现在，您可以更改没有定义表达式的属性值。

- 有些管理属性将 JBoss EAP 服务器与其他服务（如数据库）连接，您可以提前知道其值，或者您无法存储在配置中的值；例如，在数据库凭证中。通过使用环境变量，您可以在 JBoss EAP 服务器运行时延迟这些属性的配置。



重要

此功能默认为启用，从 JBoss EAP 版本 8.0 OpenShift 运行时镜像开始。要在其他平台上启用它，您必须将 WILDFLY_OVERRIDING_ENV_VARS 环境变量设置为任何值；例如，导出 WILDFLY_OVERRIDING_ENV_VARS=1。



注意

您不能覆盖其 类型为 LIST、OBJECT 或 PROPERTY 的管理属性。

**先决条件**

- 您必须已定义了要覆盖的 management 属性。

**流程**

要使用环境变量覆盖 management 属性，请完成以下步骤：

1. 识别您要更改的资源和属性的路径。例如，将资源 /subsystem=undertow/server=default-server/http-listener=default 的 proxy-address-forwarding 属性的值设置为 true。

2. 通过映射资源地址和管理属性来创建环境变量名称来覆盖此属性，如下所示：

   a. 从资源地址中删除第一个斜杠(/)： /subsystem=undertow/server=default-server/http-listener=default 变为 subsystem=undertow/server=default-server/http-listener=default。

   b. 附加两个下划线(__)和属性名称；例如： subsystem=undertow/server=default-server/http-listener=default__proxy-address-forwarding。

c.

将所有非字母数字字符替换为下划线(_)，并将整个代码行放在所有大写字母中：SUBSYSTEM_UNDERTOW_SERVER_DEFAULT_SERVER_HTTP_LISTENER_DEFAULT__PROXY_ADDRESS_FORWARDING。

3.

设置环境值：SUBSYSTEM_UNDERTOW_SERVER_DEFAULT_SERVER_HTTP_LISTENER_DEFAULT__PROXY_ADDRESS_FORWARDING=true。

**注意**

这些值是必须替换为您的实际配置值的示例。

# 第 6 章 使用 MAVEN 插件调配 JBOSS EAP 服务器

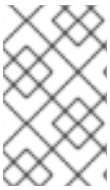
使用 JBoss EAP Maven 插件，您可以通过仅包含在服务器中提供所需功能的 Galleon 层来根据要求配置服务器。

## 6.1. JBOSS EAP MAVEN 插件

JBoss EAP Maven 插件使用 Galleon 修剪功能来减少服务器的大小和内存占用。JBoss EAP Maven 插件支持执行 JBoss EAP CLI 脚本文件以自定义您的服务器配置。CLI 脚本包含用于配置服务器的 CLI 命令列表。

您可以从 Maven 存储库检索最新的 Maven 插件版本，该版本位于 /ga/org/jboss/eap/plugins/eap-maven-plugin 的索引。在 Maven 项目中，pom.xml 文件包含 JBoss EAP Maven 插件的配置。

JBoss EAP Maven 插件在 Maven 执行过程中调配服务器，并将打包的应用（如 WAR）部署到置备的服务器。部署您的应用程序的调配服务器位于 target/server 目录中。JBoss EAP Maven 插件还提供以下功能：



注意

target/server 中的服务器不受支持，仅适用于调试或开发目的。

- 使用 org.jboss.eap:wildfly-ee-galleon-pack 和 org.jboss.eap.cloud:eap-cloud-galleon-pack Galleon feature-pack，它的一些层用于自定义服务器配置文件。
- 将 CLI 脚本命令应用到服务器。
- 支持向服务器安装中添加额外文件，如 密钥存储文件。

## 6.2. 使用 MAVEN 创建 JAKARTA EE 10 应用

创建在访问时输出 "Hello World!" 的应用。

先决条件

- 您已安装了 JDK 17。
- 已安装 Maven 3.6 或更高版本。如需更多信息，请参阅 下载 Apache Maven。

**流程**

1. 设置 Maven 项目。

```
$ mvn archetype:generate \
-DgroupId=GROUP_ID \
-DartifactId=ARTIFACT_ID \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

其中 *GROUP_ID* 是项目的 groupId，*ARTIFACT_ID* 是项目的 artifactId。

2. 要将 Maven 配置为自动管理 jboss-eap-ee BOM 中 Jakarta EE 工件的版本，请将 BOM 添加到 project pom.xml 文件的 & lt;dependencyManagement > 部分。例如：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>8.0.0.GA-redhat-00009</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

**注意**

- <version>A.B.C-redhat-XXXXX</version > 其中 A.B.C 是发行号，XXXXX 是您的 JBoss EAP 实例的构建号。有关 JBoss EAP 版本的版本详情，请参阅 Red Hat Maven 存储库。发行版本和构建号适用于所有 JBoss EAP 版本。https://maven.repository.redhat.com/earlyaccess/all/org/jboss/bom/jboss-eap-ee/.

3. 将由 BOM 管理的 servlet API 工件添加到项目的 pom.xml 文件的 < dependencies > 部分，

如下例所示：

```
<dependency>
  <groupId>jakarta.servlet</groupId>
  <artifactId>jakarta.servlet-api</artifactId>
</dependency>
```

4. 创建包含以下内容的 Java 文件 TestServlet.java，并将该文件保存到 APPLICATION_ROOT/src/main/java/com/example/simple/ 目录中。

```
package com.example.simple;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
@WebServlet(urlPatterns = "/hello")
public class TestServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
      PrintWriter writer = resp.getWriter();
      writer.println("Hello World!");
      writer.close();
  }
}
```

现在，您可以在 JBoss EAP 上部署此应用，或更新此应用，以使用 Maven 插件将其部署到自定义配置的 JBoss EAP 服务器上。

## 6.3. 使用 MAVEN 插件置备 JBOSS EAP 服务器

使用 Maven 插件更新应用的 pom.xml，将其与一起打包并部署到自定义调配的 JBoss EAP 服务器上。然后，您可以在 OpenShift 上的自定义置备的 JBoss EAP 服务器上部署应用。

先决条件

- 确保 JBoss EAP Maven 插件和 JBoss EAP Maven 构件可从您的本地或远程 Maven 存储库访问。
- 您已安装了 JDK 17。

- 您已安装了 Maven。如需更多信息，请参阅 下载 Apache Maven。

**注意**

如果您使用 JDK 17 和 Maven 3.8.5 或更早版本的 Maven 版本，请使用最新的 Maven WAR 插件。

- 您已为 Jakarta EE 10 应用创建了 Maven 项目。如需更多信息，请参阅使用 Maven 创建 Jakarta EE 10 应用。

**流程**

1. 通过将以下内容添加到 pom.xml 文件中，将 Maven 配置为从远程存储库检索 JBoss EAP BOM 和 JBoss EAP Maven 插件：

```
<repositories>
   <repository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
         <enabled>false</enabled>
      </snapshots>
   </repository>
</repositories>
<pluginRepositories>
   <pluginRepository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
         <enabled>false</enabled>
      </snapshots>
   </pluginRepository>
</pluginRepositories>
```

2. 在 pom.xml 文件的 <build > 元素中添加以下内容。您必须指定 JBoss EAP Maven 插件的最新版本。例如：

```
<plugins>
   <plugin>
      <groupId>org.jboss.eap.plugins</groupId>
      <artifactId>eap-maven-plugin</artifactId>
      <version>1.0.0.Final-redhat-00014</version> 1
      <configuration>
         <channels>
```

```
            <channel>
              <manifest>
                <groupId>org.jboss.eap.channels</groupId> (2)
                <artifactId>eap-8.0</artifactId>
              </manifest>
            </channel>
          </channels>
          <feature-packs>
            <feature-pack>
              <location>org.jboss.eap:wildfly-ee-galleon-pack</location> (3)
            </feature-pack>
            <feature-pack>
              <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location> (4)
            </feature-pack>
          </feature-packs>
          <layers>
            <layer>cloud-server</layer> (5)
          </layers>
          <runtime-name>ROOT.war</runtime-name> (6)
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>package</goal> (7)
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
```

(1)

<version>1.0.0.Final-redhat-00014</version > 是 JBoss EAP Maven 插件的示例版本。有关 JBoss EAP Maven 插件发行版本的更多信息，请参阅 Red Hat Maven 存储库：https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/。

(2)

这将指定定义 JBoss EAP 服务器工件的 JBoss EAP 8.0 频道。

(3)

您可以从 JBoss EAP 频道检索此功能 pack 的版本。Galleon 功能包 包括 Galleon 层，如用于置备修剪 JBoss EAP 服务器的 cloud-server。

(4)

此功能软件包调整了云的服务器 Galleon 层。需要使用此功能软件包为 OpenShift 构建应用程序。

5

此 Galleon 层使用在云中运行 JBoss EAP 应用程序时所需的功能置备服务器。

6

使用这个配置选项，您可以在 HTTP 根上下文中注册部署。

7

借助此插件目标，您可以调配服务器、部署应用程序、应用自定义 CLI 脚本，并将自定义内容复制到服务器安装中。

3. 打包应用。

```
$ mvn package
```

目录 target/server 包含可用于调试或开发目的的服务器和应用程序。在 JBoss EAP S2I 构建上下文中，由 JBoss EAP maven-plugin 调配的服务器安装在 /opt/server 位置的 JBoss EAP 镜像中。如需更多信息，请参阅在 OpenShift 中使用 Source-to-Image (S2I)构建应用程序镜像。

注意

如果您使用启用了调试的 mvn package 命令(-X 选项)，请包含属性 -Dorg.slf4j.simpleLogger.log.com.networknt.schema=off 以防止在 schema 验证过程中出现过量调试日志记录。

验证

- 您可以检查生成的服务器配置文件 target/server/standalone/configuration/standalone.xml，其中包含置备的子系统和应用程序部署。

包含部署的 JBoss EAP 服务器已被置备。

其他资源

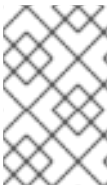
- 可用的 JBoss EAP 层.

- 使用 Maven 存储库。
- Apache Maven 文档中的标准目录布局简介。

### 6.4. GALLEON 置备文件

置备文件是带有名称 provisioning.xml 的 XML 文件，您可以将其存储在 galleon 子目录中。使用它们是在 JBoss EAP Maven 插件中配置功能软件包和层的替代选择。您可以配置 provisioning.xml 文件来微调置备过程。

以下代码演示了一个调配文件内容，您可以使用它来根据 cloud-server 层调配 JBoss EAP 服务器。



注意

JBoss EAP 功能软件包没有版本，在 Maven 插件中从配置的频道中检索版本。

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
    <feature-pack location="org.jboss.eap:wildfly-ee-galleon-pack:">1
        <default-configs inherit="false"/>2
        <packages inherit="false"/>3
    </feature-pack>
    <feature-pack location="org.jboss.eap.cloud:eap-cloud-galleon-pack:
">4
        <default-configs inherit="false"/>
        <packages inherit="false"/>
    </feature-pack>
    <config model="standalone" name="standalone.xml">5
        <layers>
            <include name="cloud-server"/>
        </layers>
    </config>
    <options>6
        <option name="optional-packages" value="passive+"/>
    </options>
</installation>
```

1

此元素指示置备过程调配从 JBoss EAP 频道检索的 JBoss EAP 功能 pack。

2

此元素指示置备过程排除默认配置。您可以在 JBoss EAP 服务器安装中检索默认配置，如 standalone.xml 和 standalone-ha.xml。当您从 JBoss EAP Maven 插件置备 JBoss EAP 服务器时，请根据配置的 Galleon 用户生成单一服务器配置。将选项设置为 false 可防止生成任何其他服务器配置。default-configs 和 软件包 不支持设置 inherit=true。

3

此元素指示置备过程排除默认软件包。

4

此元素指示调配流程调配 JBoss EAP 云功能 pack。子元素指示进程排除默认配置和默认软件包。

5

此元素指示置备过程创建自定义独立配置。该配置包括 JBoss EAP 功能 pack 中定义的 cloud-server 基础层，并通过 JBoss EAP 云功能打包为 OpenShift 进行调优。

6

此元素指示调配流程优化 JBoss EAP 模块的调配。

### 6.5. MAVEN 插件配置属性

您可以通过设置以下配置参数列表来配置 eap-maven-plugin Maven 插件：

表 6.1. Maven 插件配置属性

| Name | 类型 | 描述 |
|---|---|---|

| Name | 类型 | 描述 |
| --- | --- | --- |
| Channels | list | 频道 YAML 文件引用列表。频道文件包含 JBoss EAP 服务器工件的版本。有两种方法可以识别频道 YAML 文件。<br>● 如果您将频道 YAML 文件工件部署到带有频道分类器的 Maven 存储库中，您可以使用其 Maven 协调来识别它：**groupId**、**artifactId** 和可选版本。如果没有设置 version，它将使用最新的频道版本。例如：<br>`<channels>`<br>`<channel>`<br>`<manifest>`<br>`<groupId>org.jboss.eap.channels</groupId>`<br>`<artifactId>eap-8.0</artifactId>`<br>`</manifest>`<br>`</channel>`<br>`</channels>`<br>● 您可以使用 URL 检索频道 YAML 文件。例如：<br>`<channels>`<br>`<channel>`<br>`<manifest>`<br>`<url>file:///foo/my-manifest.yaml</url>`<br>`</manifest>`<br>`</channel>`<br>`</channels>` |
| **excluded-layers** | list | 要排除的 Galleon 层列表。当设置了 **feature-pack-location** 或 feature packs 时，您可以使用它。使用系统属性 **wildfly.provisioning.layers.excluded** 提供要排除的、以逗号分隔的层列表。 |
| **extra-server-content-dirs** | list | 将内容复制到调配的服务器的目录列表。您可以使用到目录的绝对路径或相对路径。相对路径必须相对于项目基础目录。 |
| **feature-packs** | list | 要安装的功能软件包配置列表，您可以和层组合使用。使用系统属性 **wildfly.provisioning.feature-packs** 提供以逗号分隔的功能软件包列表。 |

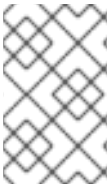
| Name | 类型 | 描述 |
| --- | --- | --- |
| **filename** | 字符串 | 要部署的应用程序的文件名。默认值为 **${project.build.finalName}.${project.packaging}**。在异常情况下，**ejb** 打包结果为 **.jar** 扩展名。例如，war 打包期间 **$[project.packaging]** 的值为 **war**，在 ejb 打包期间 **$[project.packaging]** 的值是 **ejb**，这不是有效的 jar 扩展。这些情形需要 **.jar** 扩展。 |
| **Galleon-options** | Map | 在置备服务器时，您可以设置特定的 Galleon 选项。如果您要在同一 Maven 会话中构建大量服务器，您必须将 **jboss-fork-embedded** 选项设置为 **true** 以 fork Galleon 置备和 CLI 脚本执行。例如：<br><br>`<galleon-options>`<br>`  <jboss-fork-embedded>true</jboss-fork-embedded>`<br>`</galleon-options>` |
| 层 | list | 要置备的 Galleon 层列表。当设置了 **feature-pack-location** 或 feature packs 时，您可以使用它。使用系统属性 **wildfly.provisioning.layers** 提供以逗号分隔的层列表。 |
| **layers-configuration-file-name** | 字符串 | 从层生成的配置文件的名称。默认值为 **standalone.xml**。如果没有配置层，则无法设置此参数。 |
| **log-provisioning-time** | 布尔值 | 指定在置备结束时是否记录置备时间。默认值为 **false**。 |
| **name** | 字符串 | 用于部署的名称。 |
| **offline-provisioning** | 布尔值 | 指定在插件解析工件时是否使用离线模式。在离线模式中，插件使用本地 Maven 存储库进行工件解析。默认值为 **false**。 |
| **overwrite-provisioned-server** | 布尔值 | 如果要从 **provisioningDir** 中删除引用的现有服务器并置备一个新的服务器，请将其设为 **true**。如果没有，则将其设置为 **false**。默认值为 **false**。 |

| Name | 类型 | 描述 |
| --- | --- | --- |
| **packaging-scripts** | list | 要执行的 CLI 脚本和命令的列表。如果脚本文件不是绝对的，它必须相对于项目基础目录。使用以下方法配置 CLI 执行：<br><br>`<packaging-scripts>`<br>`<packaging-script>`<br>`<scripts>`<br>`<script>../scripts/script1.cli</script>`<br>`</scripts>`<br>`<commands>`<br>`<command>/system-property=foo:add(value=bar)</command>`<br>`</commands>`<br>`<properties-files>`<br>`<property-file>my-properties.properties</property-file>`<br>`</properties-files>`<br>`<java-opts>`<br>`<java-opt>-Xmx256m</java-opt>`<br>`</java-opts>`<br>`<!-- Expressions resolved during server execution -->`<br>`<resolve-expressions>false</resolve-expressions>`<br>`</packaging-script>`<br>`</packaging-scripts>` |
| **provisioning-dir** | 字符串 | 要在其中置备服务器的目录的路径。它可以是绝对路径或相对于 **buildDir** 的路径。默认情况下，服务器被调配到 **target/server** 目录中。默认值为 **server**。 |
| **provisioning-file** | File | 要使用的 **provisioning.xml** 文件的路径。当设置了 feature packs 配置项目和层配置项时，您无法使用它。如果置备文件路径不是绝对的，它必须相对于项目基础目录。默认值为 **${project.basedir}/galleon/provisioning.xml**。 |
| **record-provisioning-state** | 布尔值 | 指定是否在 **.galleon** 目录中记录置备状态。默认值为 **false**。 |
| **runtime-name** | 字符串 | 部署的 runtime-name。默认值为部署文件名，如 **myapp.war**。您可以将此参数设置为 **ROOT.war**，以获取在 HTTP root 上下文中注册的部署。 |
| **server-config** | 字符串 | 部署期间要使用的服务器配置的名称。如果设置了 **layers-configuration-file-name**，部署会部署到从 **layers-configuration-file-name** 引用的配置中。默认值为 **standalone.xml**。 |

| Name | 类型 | 描述 |
| --- | --- | --- |
| **skip** | 布尔值 | 如果您希望跳过目标，请将其设为 **true**。如果没有，则将其设置为 **false**。默认值为 **false**。 |
| **stdout** | 字符串 | 指明如何为创建的 CLI 进程处理 **stdout** 和 **stderr**。如果值被定义，**stderr** 会重定向到 **stdout**，除非值为 none。默认情况下，**stdout** 和 **stderr** 流从当前进程继承。您可以从以下选项将设置改为一：<br>• **none** 表示不应使用 **stderr** 和 **stdout**。<br>• **用于重定向到当前进程的 system.out 或 System.err**。<br>• 任何其它值都假定为文件的路径，**stdout** 和 **stderr** 将写入该文件。 |

## 6.6. 如何为 JBOSS EAP 8.0 启用对 EAP-DATASOURCES-GALLEON-PACK 的支持

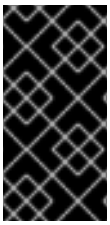
**eap-data-sources-galleon-pack Galleon 功能 pack 允许您置备一个可连接到数据库的 JBoss EAP 8.0 服务器。**



**注意**

**并非所有数据库都被支持。**

**另外，此功能 pack 为各种数据库提供 JDBC 驱动程序和数据源，这些数据库可以和 JBoss EAP 8.0 Galleon 功能软件包一起置备。这个功能软件包中定义的 Galleon 层是 decorator 层。这意味着除了 JBoss EAP 基础层外，还需要调配它们。**



**重要**

**datasources-web-server 基础层是置备由功能软件包定义的 Galleon 层时使用的最小基本层。**

**其他资源**

- **JBoss EAP 8.0 定义层文档**

## 6.7. 支持的驱动程序和数据源

对于 Galleon 功能软件包支持的每个数据库，它提供同时构建的 Galleon 层，它们是：

- postgresql-driver
- postgresql-datasource
- mssqlserver-datasource
- mssqlserver-driver
- Oracle-datasource
- oracle-driver

表 6.2. 支持的驱动程序和数据源

| 层 | 描述 |
| --- | --- |
| postgresql-driver | 这会为驱动程序安装 JBoss EAP 模块，并将驱动程序资源添加到服务器配置中的 data sources 子系统。 |
| postgresql-datasource | 这基于 **postgresql-driver** Galleon 层构建，以添加数据源。 |

注意

- 功能软件包中没有包括特定的驱动程序版本。在置备服务器前，您必须指定驱动程序版本。

  Example

  ```
  POSTGRESQL_DRIVER_VERSION="42.2.19"
  ```

- 特定于驱动程序的环境变量在其特定的驱动程序文档中定义。

其他资源

- 配置 Microsoft SQL Server 驱动程序和数据源的环境变量
- 配置 Oracle 驱动程序和数据源的环境变量
- 配置 PostgreSQL 驱动程序和数据源的环境变量

## 6.8. 使用 JBOSS EAP MAVEN 插件置备具有 JDBC 驱动程序和数据源的服务器

您可以使用 Galleon 功能软件包在 OpenShift 中置备 JBoss EAP 服务器。

注意

此流程仅演示如何在 OpenShift 上为 JBoss EAP 8.0 置备 JBoss EAP 服务器。

先决条件

- 已安装 OpenShift 并设置 OpenShift CLI ("oc")。如需更多信息，请参阅 OpenShift CLI 入门。
- 您已经了解如何使用 JBoss EAP maven 插件。如需更多信息，请参阅 JBoss EAP Maven 插件。

流程

- 将数据源 feature pack 的 Maven 协调(GroupId 和 artifactId)添加到 JBoss EAP maven 插件配置中。

```
<channels>
  <channel>
    <groupId>org.jboss.eap.channels</groupId>
    <artifactId>eap-8.0</artifactId>
  </channel>
</channels>
<feature-packs>
  <feature-pack>
    <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
  </feature-pack>
```

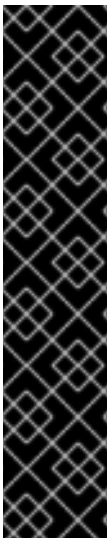
```
  <feature-pack>
    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap:eap-datasources-galleon-pack</location>
  </feature-pack>
</feature-packs>
<layers>
  <!-- Base layer -->
  <layer>jaxrs-server</layer>
    <!-- The postgresql datasource layer -->
  <layer>postgresql-datasource</layer>
</layers>
```

**其他资源**

- 用于 todo-backend 快速启动的 JBoss EAP 8 Beta Maven 插件配置

# 第 7 章 配置 JBOSS EAP 服务器和应用程序

JBoss EAP for OpenShift 镜像已预先配置用于 Java 应用程序的基本用途。但是，您可以在镜像中配置 JBoss EAP 实例。推荐的方法是使用 OpenShift S2I 进程，并在 Helm chart 中设置环境变量来调整 JVM。



**重要**

当容器重启或终止时，对正在运行的容器所做的任何配置更改都将丢失。

这包括使用传统 JBoss EAP 安装中包含的脚本进行的任何配置更改，如 add-user.sh 或 management CLI。

强烈建议您将 OpenShift S2I 进程与环境变量一起使用，对 JBoss EAP for OpenShift 镜像内的 JBoss EAP 实例进行任何配置更改。

## 7.1. JVM 默认内存设置

您可以使用以下环境变量来修改自动计算的 JVM 设置：请注意，只有在定义了有效容器内存限值时，才会自动计算默认内存大小时使用这些变量。

| 环境变量 | 描述 |
| --- | --- |
| **JAVA_INITIAL_MEM_RATIO** | 这个环境变量现已弃用。对应于 JVM 参数 **-XX:InitialRAMPercentage**。默认情况下不指定，并将在以后的版本中删除。您需要直接在 **JAVA_OPTS** 中指定 **--XX:InitialRAMPercentage**。<br>  **注意**<br>您不再需要设置 **JAVA_INITIAL_MEM_RATIO=0** 以禁用自动计算。因为没有为这个环境变量提供默认值。 |
| **JAVA_MAX_MEM_RATIO** | 配置 **-XX:MaxRAMPercentage** JVM 选项的环境变量。将最大堆大小设置为 Java 虚拟机可用内存总量的百分比。默认值为 80%。设置 **JAVA_MAX_MEM_RATIO=0** 会禁用这个默认值。 |

| 环境变量 | 描述 |
| --- | --- |
| **JAVA_OPTS** | 为 JVM 提供附加选项的环境变量，如 **JAVA_OPTS=-Xms512m -Xmx1024m**<br><br>**注意**<br><br>如果您为 **-Xms** 设置了一个值，则忽略 **-XX:InitialRAMPercentage** 选项。如果您为 **-Xmx** 设置值，则忽略 **-XX:MaxRAMPercentage** 选项。 |
| **JAVA_MAX_INITIAL_MEM** | 这个环境变量现已弃用。使用 **JAVA_OPTS** 提供 '-Xms' 选项，如 **JAVA_OPTS=-Xms256m** |

### 7.2. JVM 垃圾回收设置

**OpenShift 的 EAP 镜像包括垃圾回收和垃圾回收日志记录的设置**

垃圾回收设置

```
-XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4
-XX:AdaptiveSizePolicyWeight=90 -XX:+ExitOnOutOfMemoryError
```

垃圾回收日志设置

```
-Xlog:gc*:file=/opt/server/standalone/log/gc.log:time,uptimemillis:filecount=5,filesize=3M
```

### 7.3. JVM 环境变量

**使用这些环境变量为 OpenShift 镜像在 EAP 中配置 JVM。**

**表 7.1. JVM 环境变量**

| 变量名称 | 示例 | 默认值 | JVM 设置 | 描述 |
| --- | --- | --- | --- | --- |

| 变量名称 | 示例 | 默认值 | JVM 设置 | 描述 |
| --- | --- | --- | --- | --- |
| JAVA_OPTS | -verbose:class | 没有默认值 | Multiple | 传递给 **java** 命令的 JVM 选项。<br><br>使用 **JAVA_OPTS_APPEND** 配置额外的 JVM 设置。如果使用 **JAVA_OPTS**，则在服务器 JVM 设置中不会添加一些不可配置的默认值。您必须明确添加这些设置。<br><br>使用 **JAVA_OPTS** 禁用容器脚本默认添加的某些设置。禁用的设置包括：<br><br>● -XX:MetaspaceSize=96M<br>● -Djava.net.preferIPv4Stack=true<br>● -Djboss.modules.system.pkgs=jdk.nashorn.api,com.sun.crypto.provider<br>● -Djava.awt.headless=true<br><br>如果使用 **JAVA_OPTS** 配置其他设置，请添加这些默认值。 |
| JAVA_OPTS_APPEND | -Dsome.property=value | 没有默认值 | Multiple | 用户指定的 Java 选项，用于附加到 **JAVA_OPTS** 中生成的选项。 |
| JAVA_MAX_MEM_RATIO | 50 | 80 | -Xmx | 当 **JAVA_OPTS** 中没有指定 **-Xmx** 选项时，请使用此变量。此变量的值用于根据容器的限制计算默认最大堆内存大小。如果在没有内存约束的容器中使用了此变量，则变量不会起作用。如果在具有内存约束的容器中使用了此变量，则 **-Xmx** 的值设置为容器可用内存的指定比例。默认值 *50* 代表 50% 用作可用内存的上限。要跳过最大内存计算，请将此变量的值设置为 *0*。无 **-Xmx** 选项将添加到 **JAVA_OPTS**。 |
| JAVA_INITIAL_MEM_RATIO | 25 | -Xms | -Xms | 在 **JAVA_OPTS** 中没有指定 **-Xms** 选项时，使用此变量。此变量的值用于根据最大堆内存计算默认初始堆内存大小。如果在没有内存约束的容器中使用了此变量，则变量不会起作用。如果在具有内存约束的容器中使用了此变量， **-Xms** 的值会被设置为 **-Xmx** 内存的指定比例。默认值为 *25*，表示最大内存的 25% 用作初始堆大小。要跳过初始内存的计算，请将此变量的值设置为 *0*。没有 **-Xms** 选项会被添加到 **JAVA_OPTS**。 |

| 变量名称 | 示例 | 默认值 | JVM 设置 | 描述 |
| --- | --- | --- | --- | --- |
| JAVA_MAX_INITIAL_MEM | 4096 | 4096 | -Xms | **JAVA_MAX_INITIAL_MEM** 环境变量现已弃用，使用 **JAVA_OPTS** 提供 -Xms 选项。例如： JAVA_OPTS=-Xms256m |
| JAVA_DIAGNOSTICS | true | false（禁用） | **-Xlog:gc:utctime -XX:NativeMemoryTracking=summary** | 将这个变量的值设置为 **true**，以在事件发生时在标准输出中包含诊断信息。如果在 **JAVA_DIAGNOSTICS** 已定义为 **true** 的环境中将此变量定义为 **true**，则仍然包含诊断。 |
| DEBUG | true | false | -agentlib:jdwp=transport=dt_socket,address=$DEBUG_PORT,server=y,suspend=n | 启用远程调试。 |
| DEBUG_PORT | 8787 | 8787 | -agentlib:jdwp=transport=dt_socket,address=$DEBUG_PORT,server=y,suspend=n | 指定用于调试的端口。 |
| GC_MIN_HEAP_FREE_RATIO | 20 | 10 | -XX:MinHeapFreeRatio | 垃圾回收后可用的最小堆百分比，以避免扩展。 |
| GC_MAX_HEAP_FREE_RATIO | 40 | 20 | -XX:MaxHeapFreeRatio | 垃圾回收后空闲的最大堆百分比，以避免缩小。 |
| GC_TIME_RATIO | 4 | 4 | -XX:GCTimeRatio | 指定垃圾回收之外花费的时间（例如，应用程序执行花费的时间）与垃圾回收所花费的时间比。 |
| GC_ADAPTIVE_SIZE_POLICY_WEIGHT | 90 | 90 | -XX:AdaptiveSizePolicyWeight | 提供给当前垃圾回收时间的权重，与之前的垃圾回收时间相比。 |
| GC_METASPACE_SIZE | 20 | 96 | -XX:MetaspaceSize | 初始元空间大小。 |
| GC_MAX_METASPACE_SIZE | 100 | 没有默认值 | -XX:MaxMetaspaceSize | 最大元空间大小。 |

| 变量名称 | 示例 | 默认值 | JVM 设置 | 描述 |
| --- | --- | --- | --- | --- |
| GC_CONTAINER_OPTIONS | -XX:+UseG1GC | -XX:-UseParallelGC | -XX:-UseParallelGC | 指定要使用的 Java 垃圾回收。变量的值通过使用 Java Runtime Environment (JRE)命令行选项来指定。指定的 JRE 命令会覆盖默认值。 |

**以下环境变量已弃用：**

- **JAVA_OPTIONS ：使用 JAVA_OPTS。**
- **INITIAL_HEAP_PERCENT: 使用 JAVA_INITIAL_MEM_RATIO。**
- **CONTAINER_HEAP_PERCENT: 使用 JAVA_MAX_MEM_RATIO。**

### 7.4. 默认数据源

**JBoss EAP 8.0 不提供数据源 ExampleDS。**

**有些快速入门需要这个数据源：**

- **cmt**
- **thread-racing**

**客户开发的应用程序可能还需要 ExampleDS 数据源。**

**如果您需要默认数据源，请使用 ENABLE_GENERATE_DEFAULT_DATASOURCE 环境变量，在调配 JBoss EAP 服务器时包含它。**

```
ENABLE_GENERATE_DEFAULT_DATASOURCE=true
```

**注意**

只有在使用 **cloud-default-config galleon** 层时，此环境变量才能正常工作。

# 第 8 章 JBOSS EAP FOR OPENSHIFT 中的功能修剪

修剪服务器可减少调配服务器的安全风险，或者减少内存占用，使其更适合微服务容器。

在构建包含 JBoss EAP 的镜像时，您可以控制要包含在镜像中的 JBoss EAP 功能和子系统。当您在 Source-to-Image (S2I)构建过程中创建新应用程序时，您可以使用 JBoss EAP Maven 插件进行此操作。如需更多信息，请参阅使用 Maven 插件置备 JBoss EAP 服务器。

注意

在 S2I 构建过程中，您可以使用以下环境变量而不是 JBoss EAP Maven 插件：

- GALLEON_PROVISION_FEATURE_PACKS
- GALLEON_PROVISION_LAYERS
- GALLEON_PROVISION_CHANNELS

## 8.1. 可用的 JBOSS EAP 层

红帽提供了基础层和 decorator 层，允许您在 OpenShift 中自定义您的 JBoss EAP 服务器。基本层提供核心功能，decorator 层增强了基础层。

任何置备层都不支持以下 Jakarta EE 规格：

- Jakarta Server Faces 2.3
- Jakarta Enterprise Beans 3.2
- Jakarta XML Web Services 2.3

### 8.1.1. 基本层

每个基础层包括典型的服务器用户用例的核心功能。

**datasources-web-server**

此层包含一个 servlet 容器，以及配置数据源的功能。

以下是 datasources-web-server 中默认包含的 JBoss EAP 子系统：

- core-management
- 数据源
- deployment-scanner
- ee
- elytron
- io
- jca
- jmx
- logging
- naming
- request-controller

- **security-manager**
- **Transactions**
- **undertow**

**此层支持以下 Jakarta EE 规格：**

- **Jakarta JSON Processing 1.1**
- **Jakarta JSON Binding 1.0**
- **Jakarta Servlet 4.0**
- **Jakarta Expression Language 3.0**
- **Jakarta Server Pages 2.3**
- **Jakarta Standard Tag Library 1.2**
- **Jakarta Concurrency 1.1**
- **Jakarta Annotations 1.3**
- **Jakarta XML Binding 2.3**
- **Jakarta Debugging Support for Other Languages 1.0**

- Jakarta Transactions 1.3
- Jakarta Connectors 1.7

**jaxrs-server**

此层通过以下 JBoss EAP 子系统增强了 datasources-web-server 层：

- jaxrs
- weld
- jpa

此层还将基于 Infinispan 的第二级实体添加本地缓存到容器。

除了 datasources-web-server 层支持的那些层，还需要以下 Jakarta EE 规格：

- jakarta 上下文和依赖注入 2.0
- Jakarta Bean Validation 2.0
- Jakarta Interceptors 1.2
- Jakarta RESTful Web Services 2.1
- Jakarta Persistence 2.2

**cloud-server**

此层通过以下 JBoss EAP 子系统增强了 jaxrs-server 层：

- resource-adapters
- messaging-activemq （远程代理消息传递，非嵌入式消息传递）

此层还在 jaxrs-server 层中添加以下可观察性功能：

- 原生健康
- 原生指标

除了 jaxrs-server 层支持的外，该层还支持以下 Jakarta EE 规格：

- Jakarta Security 1.0

**cloud-default-config**

此层根据 standalone-ha.xml 使用服务器配置调配服务器，并包含子系统配置 messaging-activemq。在 contrary 上，不包括 modcluster 和 core-management 子系统配置。这被配置为在云中使用。此外，将安装所有 JBoss EAP 服务器 JBoss 模块。

**ee-core-profile-server**

ee-core-profile-server 层使用 Jakarta EE 10 Core Profile 置备服务器。Core Profile 为提供核心 JBoss EAP 服务器功能和 Jakarta EE API 的用户提供了一个小型配置文件。ee-core-profile-server 层最适合小型运行时，如云原生应用和微服务。

### 8.1.2. decorator 层

Decorator 层不会独立使用。您可以使用基本层配置一个或多个 decorator 层，以提供额外的功能。

**Observability（可观察性）**

这个 decorator 层在置备的服务器中添加了以下可观察性功能：

- 原生健康

- 原生指标

注意

此层内置在 cloud-server 层。您不需要将此层添加到 cloud-server 层。

**web-clustering**

此层将嵌入式基于 Infinispan 的 Web 会话集群添加到调配的服务器。

### 8.2. 在 JBOSS EAP 中置备用户开发的层

除了红帽提供的配置层外，您还可以配置您开发的自定义层。

**流程**

1. 使用 Galleon Maven 插件构建自定义层。

   如需更多信息，请参阅准备 Maven 项目。

2. 将自定义层部署到可访问的 Maven 存储库。

3. 您可以使用自定义 Galleon 功能打包环境变量在 S2I 镜像构建过程中自定义 Galleon 功能包和层。

   有关自定义 Galleon 功能包和层的更多信息，请参阅在 S2I 构建过程中使用自定义 Galleon 功能包。

4. 可选：创建一个自定义配置文件来引用用户定义的层，并支持的 JBoss EAP 层并将其存储在应用目录中。

   有关创建自定义置备文件的更多信息，请参阅 Galleon 置备文件。

5. 运行 S2I 流程以在 OpenShift 中调配 JBoss EAP 服务器。

如需更多信息，请参阅在 S2I 构建过程中使用自定义 Galleon 功能包。

### 8.2.1. 为 JBoss EAP 构建并使用自定义 Galleon 层

自定义 Galleon 层打包在 Galleon 功能包中，旨在使用 JBoss EAP 8.0 运行。

在 Openshift 中，您可以构建和使用包含要置备的层的 Galleon 功能包，例如，JBoss EAP 8.0 服务器的 MariaDB 驱动程序和数据源。层包含服务器中安装的内容。层可以更新服务器 XML 配置文件，并将内容添加到服务器安装中。

本节介绍了如何构建和使用包含层的 Galleon 功能包，以置备 OpenShift 中的 JBoss EAP 8.0 服务器的 MariaDB 驱动程序和数据源。

#### 8.2.1.1. 准备 Maven 项目

Galleon feature-packs 使用 Maven 创建。此流程包括创建新的 Maven 项目的步骤。

**流程**

1. 运行以下命令来创建一个新的 Maven 项目：

```
mvn archetype:generate -DarchetypeGroupId=org.codehaus.mojo.archetypes -
DarchetypeArtifactId=pom-root -DgroupId=org.jboss.eap.demo -DartifactId=mariadb-galleon-
pack -DinteractiveMode=false
```

2. 导航到 mariadb-galleon-pack 目录，并更新 pom.xml 文件，使其包含 Red Hat Maven 存储库：

```
<repositories>
  <repository>
    <id>redhat-ga</id>
    <name>Redhat GA</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </repository>
</repositories>
```

3. 更新 pom.xml 文件，以添加 JBoss EAP Galleon feature-pack 和 MariaDB 驱动程序的依赖项：

-

```
<dependencies>
  <dependency>
    <groupId>org.jboss.eap</groupId>
    <artifactId>wildfly-ee-galleon-pack</artifactId>
    <version>8.0.0.GA-redhat-00010</version>
    <type>zip</type>
  </dependency>
  <dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.7.2</version>
  </dependency>
</dependencies>
```

**注意**

- **<version>A.B.C-redhat-XXXXX</version > 其中 A.B.C 是发行号，XXXXX 是您的 JBoss EAP 实例的构建号。有关 JBoss EAP 版本的版本详情，请参阅 Red Hat Maven 存储库。发行版本和构建号适用于所有 JBoss EAP 版本。https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/wildfly-ee-galleon-pack/.**

4. **更新 pom.xml 文件，使其包含用于构建 Galleon feature-pack 的 Maven 插件：**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.wildfly.galleon-plugins</groupId>
      <artifactId>wildfly-galleon-maven-plugin</artifactId>
      <version>6.4.8.Final-redhat-00001</version>
      <executions>
        <execution>
          <id>mariadb-galleon-pack-build</id>
          <goals>
            <goal>build-user-feature-pack</goal>
          </goals>
          <phase>compile</phase>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

#### 8.2.1.2. 添加 feature-pack 内容

**这个过程帮助您在自定义 Galleon 功能包中添加层，例如 feature-pack，包括 MariaDB 驱动程序和数据源层。**

**先决条件**

- 您已创建了一个 Maven 项目。如需了解更多详细信息，请参阅准备 Maven 项目。

**流程**

1. 在自定义 feature-pack Maven 项目中创建目录 src/main/resources，例如，请参阅 准备 Maven 项目。此目录是包含 feature-pack 内容的根目录。

2. 创建 src/main/resources/modules/org/mariadb/jdbc/main 目录。

3. 在 主目录中，使用以下内容创建一个名为 module.xml 的文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="org.mariadb.jdbc" xmlns="urn:jboss:module:1.8">
  <resources>
    <artifact name="${org.mariadb.jdbc:mariadb-java-client}"/> 1
  </resources>
  <dependencies> 2
    <module name="java.se"/>
    <module name="jakarta.transaction.api"/>
    <module name="jdk.net"/>
  </dependencies>
</module>
```

1. MariaDB 驱动程序 groupId 和 artifactId.在调配时，会安装实际的驱动程序 JAR 文件。驱动程序的版本从 pom.xml 文件引用。

2. 适用于 MariaDB 驱动程序的 JBoss 模块模块依赖项。

4. 创建 src/main/resources/layers/standalone/ 目录。这是 Galleon feature-pack 定义的所有层的根目录。

5. 创建 src/main/resources/layers/standalone/mariadb-driver 目录。

6. **在 mariadb-driver 目录中，使用以下内容创建 layer-spec.xml 文件：**

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="mariadb-driver">
  <feature spec="subsystem.datasources"> (1)
    <feature spec="subsystem.datasources.jdbc-driver">
      <param name="driver-name" value="mariadb"/>
      <param name="jdbc-driver" value="mariadb"/>
      <param name="driver-xa-datasource-class-name"
value="org.mariadb.jdbc.MariaDbDataSource"/>
      <param name="driver-module-name" value="org.mariadb.jdbc"/>
    </feature>
  </feature>
  <packages> (2)
    <package name="org.mariadb.jdbc"/>
  </packages>
</layer-spec>
```

(1) **使用名为 *MariaDB* 的 JDBC 驱动程序更新 datasources 子系统配置，由模块 org.mariadb.jdbc 来实施。**

(2) **包含置备层时安装的驱动程序类的 JBoss 模块模块。**

**mariadb-driver 层使用通过 JBoss 模块模块 实施的 JDBC 驱动程序的配置更新 datasources 子系统。**

7. **创建 src/main/resources/layers/standalone/mariadb-datasource 目录。**

8. **在 mariadb-datasource 目录中，使用以下内容创建 layer-spec.xml 文件：**

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="mariadb-datasource">
  <dependencies>
    <layer name="mariadb-driver"/> (1)
  </dependencies>
  <feature spec="subsystem.datasources.data-source"> (2)
    <param name="data-source" value="MariaDBDS"/>
    <param name="jndi-name"
value="java:jboss/datasources/${env.MARIADB_DATASOURCE:MariaDBDS}"/>
    <param name="connection-url"
value="jdbc:mariadb://${env.MARIADB_HOST:localhost}:${env.MARIADB_PORT:3306}/${env.
```

```
MARIADB_DATABASE}"/> 3
    <param name="driver-name" value="mariadb"/>
    <param name="user-name" value="${env.MARIADB_USER}"/> 4
    <param name="password" value="${env.MARIADB_PASSWORD}"/>
  </feature>
</layer-spec>
```

1

这个依赖项在置备数据源时强制置备 MariaDB 驱动程序。在调配该层时，会自动调配层所有层。

2

使用名为 *MariaDBDS* 的数据源更新 datasources 子系统配置。

3

数据源的名称、主机、端口和数据库值从环境变量 *MARIADB_DATASOURCE*、*MARIADB_HOST*、*MARIADB_PORT* 和 *MARIADB_DATABASE*（服务器启动时设置）解析。

4

用户名和密码值可从环境变量 *MARIADB_USER* 和 *MARIADB_PASSWORD* 解析。

9. 运行以下命令来构建 Galleon 功能包：

```
mvn clean install
```

文件 target/mariadb-galleon-pack-1.0-SNAPSHOT.zip 被创建。

#### 8.2.1.3. 在 S2I 构建过程中使用自定义 Galleon 功能包

在 OpenShift S2I 构建过程中，必须向 Maven 构建提供自定义功能包。这通常是通过将自定义 feature-pack 部署为工件（例如 org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT ）来实现的。

**注意**

有关为自定义 Galleon 功能打包使用配置 JBoss EAP S2I 镜像的更多信息，请参阅使用 高级环境变量配置 Galleon。

**先决条件**

- 已安装 oc 命令行
- 已登陆到一个 OpenShift 集群
- 您已配置了对 Red Hat Container registry 的访问。如需更多信息，请参阅 Red Hat Container Registry。
- 您已创建了自定义 Galleon 功能包。如需更多信息，请参阅准备 Maven 项目。

**流程**

1. 运行以下命令来启动 MariaDB 数据库。本例使用 MariaDB 镜像 mariadb-105-rhel7。您必须使用最新支持的 MariaDB 镜像版本。请参阅 红帽生态系统目录 以获取有关 MariaDB 镜像 的更多信息。

```
oc new-app -e MYSQL_USER=admin -e MYSQL_PASSWORD=admin -e
MYSQL_DATABASE=mariadb registry.redhat.io/rhscl/mariadb-105-rhel7
```

   OpenShift 服务 mariadb-101-rhel7 已创建并启动。

2. 通过自定义 feature-pack Maven 构建生成的 feature-pack Maven 构建，在 Maven 项目目录 mariadb-galleon-pack 中运行以下命令来创建一个 secret：

```
oc create secret generic mariadb-galleon-pack --from-file=target/mariadb-galleon-pack-1.0-
SNAPSHOT.zip
```

   secret mariadb-galleon-pack 已创建。在启动 S2I 构建时，此机密用于在 pod 中挂载 feature-pack .zip 文件，使文件在服务器调配阶段可用。

#### 8.2.1.4. 导入 JBoss EAP 8 镜像流

您可以按照以下步骤导入 JBoss EAP 8.0 镜像流。

**流程**

1. 导入 JBoss EAP 8.0 镜像流：

```
oc import-image jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest --
from=registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest
--confirm
```

#### 8.2.1.4.1. 使用 JBoss EAP maven 插件创建 S2I 构建

eap-maven-plugin 已配置了对 JBoss EAP galleon feature-pack、JBoss EAP 云 galleon feature-pack 和 mariadb galleon feature-pack 的引用。请参阅 pom.xml 的提取：

```
<feature-packs>
  <feature-pack>
    <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-pack>
    <location>org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT</location> (1)
  </feature-pack>
</feature-packs>
<layers>
  <layer>jaxrs-server</layer>
  <layer>mariadb-datasource</layer> (2)
</layers>
```

(1) 需要 mariadb feature-pack 版本。它在 JBoss EAP 8 配置的频道中没有解决。

(2) mariadb-datasource 层。

流程

1. 运行以下命令来创建 S2I 构建：

```
oc new-build eap8-openjdk17-builder-openshift-rhel8:latest~https://github.com/jboss-
container-images/jboss-eap-8-openshift-image#EAP_8.0.0 \
--context-dir=examples/eap/custom-layers/application \
--build-secret=mariadb-galleon-pack:/tmp/demo-maven-
repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT \ (1)
--name=mariadb-app-build
```

1

mariadb-galleon-pack 机密挂载到 /tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT 目录中。

**其他资源**

如需更多信息，请参阅 JBoss EAP 8.0 演示示例。

#### 8.2.1.4.2. 使用旧的 S2I 调配功能创建 S2I 构建

您可以使用 openshift-legacy 配置集来配置 S2I 构建，以便您可以置备服务器。

**流程**

1. 运行以下命令来创建新的 OpenShift 构建：

```
oc new-build eap8-openjdk17-builder-openshift-rhel8:latest~https://github.com/jboss-container-images/jboss-eap-8-openshift-image#EAP_8.0.0 \
--context-dir=examples/eap/custom-layers/application \
--env=GALLEON_PROVISION_CHANNELS="org.jboss.eap.channels:eap-8.0" \ 1
--env=GALLEON_PROVISION_FEATURE_PACKS="org.jboss.eap:wildfly-ee-galleon-pack,org.jboss.eap.cloud:eap-cloud-galleon-pack,org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT" \ 2
--env=GALLEON_PROVISION_LAYERS="jaxrs-server,mariadb-datasource" \ 3
--env=GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO="/tmp/demo-maven-repository" \ 4
--env=MAVEN_ARGS="-Popenshift-legacy" \ 5
--build-secret=mariadb-galleon-pack:/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT \ 6
--name=mariadb-app-build
```

1

此环境变量在置备过程中使用 JBoss EAP 8.0 频道。

2

此环境变量引用 JBoss EAP 8.0 feature-pack、云 feature-pack 和 mariadb feature-pack。

3

这个环境变量引用您要用来置备服务器的 Galleon 层集合。jaxrs-server 是基础服务器层 mariadb-datasource 是我们的自定义层，将 mariadb 驱动程序和新数据源引入服务

器安装。

4

这指向包含 mariadb feature-pack 的本地 maven 存储库的位置。

5

此环境变量重新定义 MAVEN_ARGS 来启用 openshift-legacy 配置集。

6

mariadb-galleon-pack 机密挂载到 /tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT 目录中。

注意

此目录路径符合 Maven 存储库工件协调到路径映射。

#### 8.2.1.4.3. 启动构建

您可以通过创建新构建来创建 mariadb-app-build 镜像。

流程

1. 从之前创建的同一 OpenShift 构建中启动一个新构建，并运行以下命令：

```
oc start-build mariadb-app-build
```

成功执行命令后，会创建镜像 mariadb-app-build。

#### 8.2.1.4.4. 创建新部署

您可以通过提供将数据源绑定到正在运行的 MariaDB 数据库所需的环境变量来创建新部署

流程

1. 运行以下命令来创建新部署：

```
oc new-app --name=mariadb-app mariadb-app-build \
--env=MARIADB_PORT=3306 \
--env=MARIADB_USER=admin \
--env=MARIADB_PASSWORD=admin \
--env=MARIADB_HOST=mariadb-105-rhel7 \
--env=MARIADB_DATABASE=mariadb  \
--env=MARIADB_DATASOURCE=Demo (1)
```

(1) 该演示要求数据源命名为 **Demo**

**注意**

有关自定义 **Galleon** 功能包环境变量的详情，请参阅 **自定义 Galleon 功能软件包环境变量**。

2. 公开 **mariadb-app** 应用程序，运行以下命令：

```
oc expose svc/mariadb-app
```

3. 要创建新任务，请运行以下命令：

```
curl -X POST http://$(oc get route mariadb-app --template='{{ .spec.host }}')/tasks/title/foo
```

4. 要访问任务列表，请运行以下命令：

```
curl http://$(oc get route mariadb-app --template='{{ .spec.host }}')
```

添加的任务显示在浏览器中。

### 8.2.2. 使用高级环境变量配置 Galleon

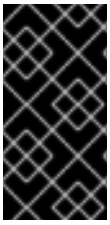
您可以使用高级自定义 **Galleon** 功能软件包环境变量来自定义在 **S2I** 镜像构建过程中存储自定义 **Galleon** 功能软件包和层的位置。这些高级自定义 **Galleon** 功能 **pack** 环境变量如下：

- **GALLEON_DIR=<path>**，将默认的 **<project_root_dir>/galleon** 目录路径覆盖为 **<project_root_dir>/<GALLEON_DIR>**。

- GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO=<path>, 将 <project root dir>/galleon/repository 目录路径改为一个到 Maven 本地存储库缓存目录的绝对路径。这个软件仓库包含自定义 Galleon 功能软件包。

您必须在与 Maven local-cache 文件系统配置兼容的子目录中找到 Galleon feature pack 归档文件。例如，在 path-to-repository/org/examples/my-feature-pack/1.0.0.Final/my-feature-pack/1.0.0 中查找 org.examples:my-feature-pack:1.0.0.Final.zip 路径。

您可以通过在 <project_root>/<GALLEON_DIR> 目录中创建一个 settings.xml 文件来配置您的 Maven 项目设置。GALLEON_DIR 的默认值为 <project_root_dir>/galleon。Maven 使用该文件为您的应用程序置备自定义 Galleon 功能软件包。如果您没有创建 settings.xml 文件，Maven 将使用 S2I 镜像创建的默认 settings.xml 文件。



重要

不要在 settings.xml 文件中指定本地 Maven 存储库位置，因为 S2I 构建器镜像指定了本地 Maven 存储库的位置。S2I 构建器镜像在 S2I 构建过程中使用此位置。

其他资源

- 自定义 Galleon 功能软件包环境变量。

### 8.2.3. 自定义 Galleon 功能 pack 环境变量

您可以使用以下自定义 Galleon 功能软件包环境变量来自定义如何使用 JBoss EAP S2I 镜像。

表 8.1. 自定义 Galleon 功能软件包环境变量的描述

| 环境变量 | 描述 |
| --- | --- |
| GALLEON_DIR=<path> | 其中 *<path>* 是相对于应用程序项目根目录的目录。您的 *<path>* 目录包含您的可选 Galleon 自定义内容，如 **settings.xml** 文件和本地 Maven 存储库缓存。这个缓存包含自定义 Galleon 功能软件包。<br><br>目录默认为 **galleon**。 |
| GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO=<path> | *<path>* 是包含自定义 feature-packs 的 Maven 本地存储库目录的绝对路径。目录默认为 **galleon/repository**。 |

| 环境变量 | 描述 |
| --- | --- |
| GALLEON_PROVISION_FEATURE_PACKS=<list_of_galleon_feature_packs> | *<list_of_galleon_feature_packs>* 是 Maven 协调标识的自定义 Galleon 功能包的逗号分隔列表。列出的功能软件包必须与构建器镜像中存在的 JBoss EAP 8.0 服务器版本兼容。<br><br>您可以使用 **GALLEON_PROVISION_LAYERS** 环境变量来为您的服务器设置 Galleon 层（由自定义功能包定义）。 |

# 第 9 章 在 OPENSHIFT CONTAINER PLATFORM 上部署您的 JBOSS EAP 应用程序

## 9.1. 用于在 OPENSHIFT 上自动化应用程序部署的 JBOSS EAP OPERATOR

EAP 操作器是扩展 OpenShift API 的 JBoss EAP 特定控制器。您可以使用 EAP 操作器创建、配置、管理和无缝升级复杂有状态应用的实例。

EAP 操作器在集群中管理多个 JBoss EAP Java 应用实例。它还通过在缩减副本前验证所有事务完成，并将 pod 标记为 clean 以进行终止，确保应用程序集群中的安全事务恢复。EAP 操作器使用 StatefulSet 来适当处理 Jakarta Enterprise Beans 远程和事务恢复处理。StatefulSet 确保持久性存储和网络主机名稳定性，即使 pod 重启后也是如此。

您必须使用 OperatorHub 安装 EAP 操作器，供 OpenShift 集群管理员用于发现、安装和升级操作器。

在 OpenShift Container Platform 4 中，您可以使用 Operator Lifecycle Manager (OLM)安装、更新和管理所有 Operator 以及在多个集群中运行的关联服务的生命周期。

OLM 在 OpenShift Container Platform 4 中默认运行。它帮助集群管理员对集群上运行的操作器进行安装、升级和授予访问权。OpenShift Container Platform Web 控制台为集群管理员提供管理界面，用于安装 Operator，以及授予特定项目访问权限以使用集群中可用的 Operator 目录。

如需有关 Operator 和 OLM 的更多信息，请参阅 OpenShift 文档。

### 9.1.1. 使用 Web 控制台安装 EAP Operator

作为 JBoss EAP 集群管理员，您可以使用 OpenShift Container Platform Web 控制台从 Red Hat OperatorHub 安装 EAP operator。然后，您可以将 EAP 操作器订阅到一个或多个命名空间，供集群上的开发人员使用。

在使用 Web 控制台安装 EAP operator 之前，您必须注意以下几个点：

- Installation Mode ：选择 All namespaces on the cluster (default)，以在所有命名空间上安装 Operator，或者选择单独的命名空间（如果可用），仅在所选命名空间中安装 Operator。
- 更新频道： 如果 EAP 操作器可以通过多个频道获得，您可以选择您要订阅的频道。例如，

要通过 stable 频道部署（如果可用），则从列表中选择这个选项。

- 批准策略：您可以选择自动或手动更新。如果选择自动更新 EAP 操作器，则当有新版本 Operator 可用时，Operator Lifecycle Manager(OLM)会自动升级正在运行的 EAP 实例。如果选择手动更新，则当有新版 Operator 可用时，OLM 会创建更新请求。然后，您必须手动批准更新请求，以便 Operator 更新至新版本。

**注意**

以下流程可能会根据 OpenShift Container Platform Web 控制台中的修改而有所变化。有关最新和最准确的流程，请参阅 *OpenShift Container Platform* 指南中的使用 Web 控制台从 OperatorHub 安装 部分。

**先决条件**

- 使用具有 cluster-admin 权限的账户访问 OpenShift Container Platform 集群。

**流程**

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。
2. 向下滚动或在 Filter by keyword 框中输入 EAP，以查找 EAP 操作器。
3. 选择 JBoss EAP operator 并单击 Install。
4. 在 Create Operator Subscription 页面：
   a. 任选以下一项：
      - All namespaces on the cluster(default)，将 operator 安装至默认 openshift-operators 命名空间，以便供集群中的所有命名空间监视和使用。该选项并非始终可用。
      - 集群中的特定命名空间会将 Operator 安装到您选择的特定命名空间中。Operator 仅可在该单一命名空间中使用。

b. 选择一个 **Update Channel**。

c. 如前面所述，选择自动或手动批准策略。

5. 点 **Subscribe** 使 EAP Operator 可供此 OpenShift Container Platform 集群上的所选命名空间使用。

a. 如果选择了手动批准策略，订阅的升级状态将保持在 **Upgrading**，直至您审核并批准了它的安装计划。批准 **Install Plan** 页面中的安装计划后，订阅升级状态将变为 **Up to date**。

b. 如果您选择了一个自动批准策略，则升级状态会在不干预的情况下变为 **Up to date**。

6. 在订阅的升级状态变为 **Up to date** 后，选择 **Operators** → **Installed Operators** 来验证 EAP ClusterServiceVersion(CSV)是否显示为 **up**，其 **Status** 会在相关命名空间中变为 **InstallSucceeded**。

**注意**

对于 **All namespaces...** 安装模式，**openshift-operators** 命名空间中显示的状态为 **InstallSucceeded**。在其他命名空间中，显示的状态是 **Copied**。如果 **Status** 字段没有更改为 **InstallSucceeded**，请检查 **openshift-operators** 项目（如果选择了 **A specific namespace...** 安装模式）中的 **openshift-operators** 项目中的 pod 的日志，这会在 **Workloads** → **Pods** 页面中报告问题以便进一步排除故障。

### 9.1.2. 使用 CLI 安装 EAP Operator

作为 **JBoss EAP** 集群管理员，您可以使用 **OpenShift Container Platform CLI** 从 **Red Hat OperatorHub** 安装 **EAP operator**。然后，您可以将 **EAP** 操作器订阅到一个或多个命名空间，供集群上的开发人员使用。

使用 **CLI** 从 **OperatorHub** 安装 **EAP Operator** 时，请使用 **oc** 命令创建 **Subscription** 对象。

**先决条件**

- 可以使用具有 **cluster-admin** 权限的账户访问 **OpenShift Container Platform** 集群。

- 您已在本地系统中安装了 **oc** 工具。

**流程**

1. 查看 **OperatorHub** 中集群可用的 **Operator** 列表：

```
$ oc get packagemanifests -n openshift-marketplace | grep eap
NAME        CATALOG              AGE
...
eap         Red Hat Operators    43d
...
```

2. 创建一个 **Subscription** 对象 **YAML** 文件（例如，**eap-operator-sub.yaml**）来向 **EAP operator** 订阅命名空间。以下是 **Subscription** 对象 **YAML** 文件示例：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: eap
  namespace: openshift-operators
spec:
  channel: stable
  installPlanApproval: Automatic
  name: eap (1)
  source:  redhat-operators (2)
  sourceNamespace: openshift-marketplace
```

(1) 要订阅的 **Operator** 的名称。

(2) **EAP Operator** 由 **redhat-operators CatalogSource** 提供。

有关频道和批准策略的详情，请查看此流程的 **Web 控制台** 版本。

3. 从 **YAML** 文件创建 **Subscription** 对象：

```
$ oc apply -f eap-operator-sub.yaml
$ oc get csv -n openshift-operators
NAME              DISPLAY     VERSION   REPLACES   PHASE
```

```
eap-operator.v1.0.0   JBoss EAP   1.0.0              Succeeded
```

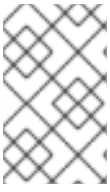
EAP 操作器已经安装成功。此时，OLM 知道 EAP 操作器。Operator 的 ClusterServiceVersion(CSV)出现在目标命名空间中，由 EAP 操作器提供的 API 可用于创建。

### 9.1.3. 使用 EAP 操作器在 OpenShift 上部署 Java 应用

EAP 操作员有助于在 OpenShift 上自动化 Java 应用程序部署。如需有关 EAP 操作器 API 的信息，请参阅 EAP Operator：API 信息。

**先决条件**

- 已安装 EAP operator。有关安装 EAP 操作器的更多信息，请参阅使用 Web 控制台安装 EAP 操作器， 以及使用 CLI 安装 EAP 操作器。
- 您已使用 JBoss EAP for OpenShift Source-to-Image (S2I)构建镜像，构建了 user 应用的 Docker 镜像。
- 如果应用程序的 CustomResourceDefinition(CRD)文件引用了一个 Secret 对象，则已创建了 Secret。有关创建新 Secret 对象的更多信息，请参阅创建 Secret。
- 如果您的应用程序的 CRD 文件引用了 ConfigMap，则已创建了 ConfigMap。有关创建 ConfigMap 的详情，请参考 创建 ConfigMap。
- 如果选择这样做，则已从 standalone.xml 文件创建了 ConfigMap。有关从 standalone.xml 文件创建 ConfigMap 的详情，请参考从 standalone.xml 文件创建 ConfigMap。



**注意**

JBoss EAP 8.0 不支持从 ConfigMap 提供 standalone.xml 文件。

**流程**

1. 打开 Web 浏览器并登录到 OperatorHub。

2. 选择您要用于 Java 应用程序的 Project 或命名空间。

3. 进入到 Installed Operator，再选择 JBoss EAP operator。

4. 在 Overview 选项卡上，单击 Create Instance 链接。

5. 指定应用程序镜像详情。

应用镜像指定包含 Java 应用的 Docker 镜像。该镜像必须使用 JBoss EAP for OpenShift Source-to-Image (S2I)构建镜像来构建。如果 applicationImage 字段与 imagestreamtag 对应，对镜像的任何更改都会触发应用程序的自动升级。

您可以为 OpenShift 应用程序镜像提供以下 JBoss EAP 参考：

- 镜像的名称：mycomp/myapp
- tag: mycomp/myapp:1.0
- 摘要：mycomp/myapp:@sha256:0af38bc38be93116b6a1d86a9c78bd14cd527121970899d719baf78e5dc7bfd2
- imagestreamtag: my-app:latest

6. 指定应用程序的大小。例如：

```
spec:
  replicas:2
```

7. 使用 env spec 配置应用程序环境。环境变量 可以直接来自值，如 POSTGRESQL_SERVICE_HOST 或来自 Secret 对象，如 POSTGRESQL_USER。例如：

```
spec:
```

```
env:
- name: POSTGRESQL_SERVICE_HOST
  value: postgresql
- name: POSTGRESQL_SERVICE_PORT
  value: '5432'
- name: POSTGRESQL_DATABASE
  valueFrom:
    secretKeyRef:
      key: database-name
      name: postgresql
- name: POSTGRESQL_USER
  valueFrom:
    secretKeyRef:
      key: database-user
      name: postgresql
- name: POSTGRESQL_PASSWORD
  valueFrom:
    secretKeyRef:
      key: database-password
      name: postgresql
```

8. **完成与应用程序部署相关的以下可选配置：**

- **指定服务器数据目录的存储要求。如需更多信息，请参阅为应用程序配置持久性存储。**

- **指定您在 WildFlyServerSpec 中创建的 Secret 名称，将其挂载为运行应用的 pod 中的卷。例如：**

```
spec:
  secrets:
    - my-secret
```

**Secret 挂载到 /etc/secrets/<secret name& gt;，每个键/值存储为一个文件。文件的名称是键，内容是值。Secret 作为一个卷在 pod 中挂载。以下示例演示了可用于查找键值的命令：**

```
$ ls /etc/secrets/my-secret/
my-key  my-password
$ cat /etc/secrets/my-secret/my-key
devuser
$ cat /etc/secrets/my-secret/my-password
my-very-secure-pasword
```

**注意**

修改 Secret 对象可能会导致项目不一致。红帽建议创建一个与旧内容相同的新对象，而不是修改现有 Secret 对象。然后，您可以根据需要更新内容，并将 operator 自定义资源(CR)中的引用从 old 改为 new。这被视为一个新的 CR 更新，pod 被重新加载。

- 指定您在 WildFlyServerSpec 中创建的 ConfigMap 名称，将其挂载为运行应用程序的 pod 中的卷。例如：

```
spec:
  configMaps:
  - my-config
```

ConfigMap 挂载于 /etc/configmaps/<configmap name& gt;，每个键/值都存储为一个文件。文件的名称是键，内容是值。ConfigMap 作为一个卷在 pod 中挂载。查找键值：

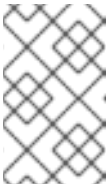
```
$ ls /etc/configmaps/my-config/
key1 key2
$ cat /etc/configmaps/my-config/key1
value1
$ cat /etc/configmaps/my-config/key2
value2
```

**注意**

修改 ConfigMap 可能会导致项目不一致。红帽建议创建一个与旧内容相同的新 ConfigMap，而不是修改现有 ConfigMap。然后，您可以根据需要更新内容，并将 operator 自定义资源(CR)中的引用从 old 改为 new。这被视为一个新的 CR 更新，pod 被重新加载。

- 如果您选择拥有自己的独立 ConfigMap，请提供 ConfigMap 的名称以及 standalone.xml 文件的密钥：

```
standaloneConfigMap:
  name: clusterbench-config-map
  key: standalone.xml
```



**注意**

JBoss EAP 8.0 不支持从 standalone.xml 文件创建 ConfigMap。

- 如果要在 OpenShift 中禁用默认的 HTTP 路由创建，请将 disableHTTPRoute 设置为 true ：

```
spec:
  disableHTTPRoute: true
```

#### 9.1.3.1. 创建 secret

如果应用程序的 CustomResourceDefinition (CRD)文件引用 Secret，则必须在使用 EAP operator 在 OpenShift 上部署应用程序前创建 Secret。

流程

- 创建 Secret ：

```
$ oc create secret generic my-secret --from-literal=my-key=devuser --from-literal=my-password='my-very-secure-pasword'
```

#### 9.1.3.2. 创建 configMap

如果应用程序的 CustomResourceDefinition (CRD)文件引用了 spec.ConfigMaps 字段中的 ConfigMap，则必须在使用 EAP operator 在 OpenShift 上部署应用程序前创建 ConfigMap。

流程

- 创建 configmap：

```
$ oc create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2
configmap/my-config created
```

#### 9.1.3.3. 从 standalone.xml 文件创建 configMap

您可以创建自己的 JBoss EAP 独立配置，而不使用来自 JBoss EAP for OpenShift Source-to-Image (S2I)的应用镜像中的一个。standalone.xml 文件必须放在可由 Operator 访问的 ConfigMap 中。

**注意**

**JBoss EAP 8.0 不支持从 ConfigMap 提供 standalone.xml 文件。**

**流程**

- **从 standalone.xml 文件创建 ConfigMap ：**

```
$ oc create configmap clusterbench-config-map --from-file
examples/clustering/config/standalone.xml
configmap/clusterbench-config-map created
```

#### 9.1.3.4. 为应用程序配置持久性存储

**如果您的应用程序需要一些数据的持久性存储，如在 pod 重启后必须保留的事务或消息日志，请配置存储规格。如果存储 spec 为空，则应用程序的每个 pod 都会使用 EmptyDir 卷。但是，这个卷在对应的 pod 停止后不会保留。**

**流程**

1. **指定 volumeClaimTemplate 来配置资源要求，以存储 JBoss EAP 独立数据目录。模板的名称派生自 JBoss EAP 的名称。对应的卷以 ReadWriteOnce 访问模式挂载。**

```
spec:
  storage:
    volumeClaimTemplate:
      spec:
        resources:
          requests:
            storage: 3Gi
```

**满足此存储要求的持久性卷挂载到 /eap/standalone/data 目录中。**

### 9.1.4. 使用 EAP 操作器查看应用的指标

**您可以使用 EAP 操作器查看 OpenShift 上部署的应用的指标。**

**当集群管理员在项目中启用指标监控时，EAP 操作器会自动显示 OpenShift 控制台中的指标。**

**先决条件**

- 您的集群管理员已为项目启用了监控。如需更多信息，请参阅 为用户定义的项目启用监控。

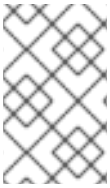
流程

1. 在 OpenShift Container Platform Web 控制台中进入 Monitoring → Metrics。

2. 在 Metrics 屏幕上，在文本框中键入应用程序的名称以选择您的应用程序。您的应用程序的指标会出现在屏幕上。

### 9.1.5. 使用 Web 控制台卸载 EAP operator

您可以从集群中删除或卸载 EAP Operator，您可以删除订阅来将其从订阅的命名空间中删除。您还可以删除 EAP Operator 的 ClusterServiceVersion(CSV)和部署。



注意

为确保数据一致性和安全性，请在卸载 EAP 操作器前将集群中的 pod 数量缩减为 0。

您可以使用 Web 控制台卸载 EAP 操作器。

警告

如果您决定删除整个 wildflyserver 定义(oc delete wildflyserver <deployment_name>)，则不会启动事务恢复过程，无论未完成的事务是什么，pod 都会被终止。此操作中未完成的结果可能会阻止您稍后启动的数据更改。其他涉及事务性企业所涉及的 JBoss EAP 实例的数据变化也会被利用这个 wildflyserver 进行远程调用。

流程

1. 在 Operators → Installed Operators 页面中，选择 JBoss EAP。

2. 在 Operator Details 页面的右侧，从 Actions 下拉菜单中选择 Uninstall Operator。

3. 如果要删除所有安装相关组件，则在看到 Remove Operator Subscription 窗口提示时，勾选 Also completely remove the Operator from the selected namespace 复选框。这会删除 CSV，并删除与 Operator 关联的 pod、部署、自定义资源定义(CRD)和自定义资源(CR)。

4. 单击 Remove。EAP 操作器将停止运行，并且不再接收更新。

### 9.1.6. 使用 CLI 卸载 JBoss EAP operator

您可以从集群中删除或卸载 EAP Operator，您可以删除订阅来将其从订阅的命名空间中删除。您还可以删除 EAP Operator 的 ClusterServiceVersion(CSV)和部署。

**注意**

为确保数据一致性和安全性，请在卸载 EAP 操作器前将集群中的 pod 数量缩减为 0。

您可以使用命令行卸载 EAP 操作器。

使用命令行时，您可以通过从目标命名空间中删除订阅和 CSV 来卸载 Operator。

**警告**

如果您决定删除整个 wildflyserver 定义(oc delete wildflyserver <deployment_name>)，则不会启动事务恢复过程，无论未完成的事务是什么，pod 都会被终止。此操作中未完成的结果可能会阻止您稍后启动的数据更改。其他涉及事务性企业所涉及的 JBoss EAP 实例的数据变化也会被利用这个 wildflyserver 进行远程调用。

**流程**

1. 在 currentCSV 字段中检查 EAP operator 订阅的当前版本：

```
$ oc get subscription eap-operator -n openshift-operators -o yaml | grep currentCSV
  currentCSV: eap-operator.v1.0.0
```

2. 删除 EAP Operator 的订阅：

```
$ oc delete subscription eap-operator -n openshift-operators
subscription.operators.coreos.com "eap-operator" deleted
```

3. 使用上一步中的 currentCSV 值删除目标命名空间中 EAP Operator 的 CSV：

```
$ oc delete clusterserviceversion eap-operator.v1.0.0 -n openshift-operators
clusterserviceversion.operators.coreos.com "eap-operator.v1.0.0" deleted
```

### 9.1.7. 用于安全事务恢复的 JBoss EAP operator

JBoss EAP 操作器在终止应用程序集群前确保数据一致性。要做到这一点，Operator 会在缩减副本前验证所有事务是否都完成，并将 pod 标记为 干净 以终止。

这意味着，如果要在没有数据不一致的情况下安全地删除部署，您必须首先将 pod 数量缩减为 0，等待所有 pod 终止，然后只删除 wildflyserver 实例。

**警告**

如果您决定删除整个 wildflyserver 定义(oc delete wildflyserver <deployment_name>)，则不会启动事务恢复过程，无论未完成的事务是什么，pod 都会被终止。此操作中未完成的结果可能会阻止您稍后启动的数据更改。其他涉及事务性企业所涉及的 JBoss EAP 实例的数据变化也会被利用这个 wildflyserver 进行远程调用。

当 scaledown 进程启动 pod 状态(oc get pod <pod_name>)时，pod 仍标记为 Running，因为 pod 必须完成所有未完成的事务，包括针对它的远程企业级 Bean 调用。

如果要监控 scaledown 进程的状态，请观察 wildflyserver 实例的状态。如需更多信息，请参阅监控扩展进程。有关缩减期间 pod 状态的信息，请参阅扩展过程中的 Pod 状态。

#### 9.1.7.1. stable 网络主机名的 StatefulSets

管理 wildflyserver 的 EAP 操作器将创建一个 StatefulSet 作为管理 JBoss EAP pod 的底层对象。

StatefulSet 是管理有状态应用程序的工作负载 API 对象。它管理一组 pod 的部署和扩展，并为这些 pod 的排序和唯一性提供保证。

StatefulSet 确保集群中的 pod 以预定义的顺序命名。它还确保 pod 终止遵循相同的顺序。例如，pod-1 有一个交易的结果，因此处于 SCALING_DOWN_RECOVERY_DIRTY 状态。即使 pod-0 处于 SCALING_DOWN_CLEAN 状态，它也不会在 pod-1 之前终止。在 pod-1 清理 并终止前，pod-0 会一直处于 SCALING_DOWN_CLEAN 状态。但是，即使 pod-0 处于 SCALING_DOWN_CLEAN 状态，也不会收到任何新请求，且实际闲置。

注意

减少 StatefulSet 的副本大小或删除 pod 本身无效，并会恢复此类更改。

#### 9.1.7.2. 监控 scaledown 进程

如果要监控 scaledown 进程的状态，您必须观察 wildflyserver 实例的状态。有关 scaledown 期间不同 pod 状态的更多信息，请参阅扩展 过程中 Pod 状态。

流程

- 观察 scaledown 进程的状态：

```
oc describe wildflyserver <name>
```

  - WildFlyServer.Status.Scalingdown Pods 和 WildFlyServer.Status.Replicas 字段显示 active 和 non-active pod 的整体状态。
  - Scalingdown Pods 字段显示所有未完成事务时要终止的 pod 数量。
  - WildFlyServer.Status.Replicas 字段显示当前运行的 pod 数量。

- WildFlyServer.Spec.Replicas 字段显示处于 ACTIVE 状态的 pod 数量。

- 如果没有 pod 在 scaledown 处理 WildFlyServer.Status.Replicas 和 WildFlyServer.Spec.Replicas 字段中的 pod 数量。

#### 9.1.7.2.1. scaledown 期间的 Pod 状态

下表描述了 scaledown 期间的不同 pod 状态：

**表 9.1. Pod 状态描述**

| Pod 状态 | 描述 |
| --- | --- |
| ACTIVE | pod 处于活跃状态并处理请求。 |
| SCALING_DOWN_RECOVERY_INVESTIGATION | pod 即将缩减。缩减流程正在调查 JBoss EAP 中的事务状态。 |
| SCALING_DOWN_RECOVERY_DIRTY | JBoss EAP 包含一些不完整的事务。在清理前 pod 无法被终止。事务恢复过程在 JBoss EAP 中定期运行，它会等待事务完成。 |
| SCALING_DOWN_CLEAN | pod 由事务缩减处理处理，并标记为 **clean** 以从集群中移除。 |

### 9.1.7.3. 在处理人员的交易期间缩减

当事务的结果未知时，无法自动事务恢复。然后，您必须手动恢复您的事务。

**先决条件**

- 您的 pod 的状态一直处于 SCALING_DOWN_RECOVERY_DIRTY 中。

**流程**

1. 使用 CLI 访问您的 JBoss EAP 实例。

2. 解决事务对象存储中的所有 heuristics 事务记录。如需更多信息，请参阅 *JBoss EAP 管理事务中的恢复 Heuristic Outcomes*。

3. 从企业 bean 客户端恢复文件夹中删除所有记录。

   a. 从 pod enterprise bean 客户端恢复目录中删除所有文件：

```
$JBOSS_HOME/standalone/data/ejb-xa-recovery
oc exec <podname> rm -rf $JBOSS_HOME/standalone/data/ejb-xa-recovery
```

4. pod 的状态更改为 SCALING_DOWN_CLEAN，pod 被终止。

#### 9.1.7.4. 配置 transactions 子系统，以使用 JDBC 存储进行事务日志

如果系统不提供用于存储 事务日志 的文件系统，请使用 JBoss EAP S2I 镜像来配置 JDBC 对象存储。

重要

当 JBoss EAP 部署为可引导 JAR 时，S2I 环境变量不可用。在这种情况下，您必须创建一个 Galleon 层或配置 CLI 脚本来进行必要的配置更改。

JDBC 对象存储可使用环境变量 TX_DATABASE_PREFIX_MAPPING 设置。此变量具有与 DB_SERVICE_PREFIX_MAPPING 相同的结构。

前提条件

- 您已根据环境变量的值创建了数据源。
- 您已确保数据库和写入权限的一致性，在数据库和 JDBC 对象存储通信 的事务管理器 之间存在。如需更多信息，请参阅配置 JDBC 数据源

流程

- 通过 S2I 环境变量设置和配置 JDBC 对象存储。

示例

```
# Narayana JDBC objectstore configuration via s2i env variables
- name: TX_DATABASE_PREFIX_MAPPING
  value: 'PostgresJdbcObjectStore-postgresql=PG_OBJECTSTORE'
- name: POSTGRESJDBCOBJECTSTORE_POSTGRESQL_SERVICE_HOST
  value: 'postgresql'
- name: POSTGRESJDBCOBJECTSTORE_POSTGRESQL_SERVICE_PORT
  value: '5432'
- name: PG_OBJECTSTORE_JNDI
  value: 'java:jboss/datasources/PostgresJdbc'
- name: PG_OBJECTSTORE_DRIVER
  value: 'postgresql'
- name: PG_OBJECTSTORE_DATABASE
  value: 'sampledb'
- name: PG_OBJECTSTORE_USERNAME
  value: 'admin'
- name: PG_OBJECTSTORE_PASSWORD
  value: 'admin'
```

验证

- **您可以通过检查 standalone.xml 配置文件 oc rsh <podname> cat /opt/server/standalone/configuration/standalone.xml 来验证数据源配置和事务子系统配置。**

  **预期输出：**

```
<datasource jta="false" jndi-name="java:jboss/datasources/PostgresJdbcObjectStore" pool-name="postgresjdbcobjectstore_postgresqlObjectStorePool"
    enabled="true" use-java-context="true" statistics-enabled="${wildfly.datasources.statistics-enabled:${wildfly.statistics-enabled:false}}">
    <connection-url>jdbc:postgresql://postgresql:5432/sampledb</connection-url>
    <driver>postgresql</driver>
    <security>
        <user-name>admin</user-name>
        <password>admin</password>
    </security>
</datasource>

<!-- under subsystem urn:jboss:domain:transactions -->
<jdbc-store datasource-jndi-name="java:jboss/datasources/PostgresJdbcObjectStore">
     <!-- the pod name was named transactions-xa-0 -->
    <action table-prefix="ostransactionsxa0"/>
    <communication table-prefix="ostransactionsxa0"/>
    <state table-prefix="ostransactionsxa0"/>
</jdbc-store>
```

**其他资源**

- 有关使用管理控制台或管理 CLI 创建数据源的更多信息，请参阅 JBoss EAP *配置指南*中的创建数据源。

### 9.1.8. 使用 pod 横向自动扩展 HPA 自动扩展 pod

通过 EAP 操作器，您可以使用 pod 横向自动扩展 HPA 根据属于该 EAP 应用的 pod 收集的指标自动增加或减少 EAP 应用的规模。

**注意**

使用 HPA 确保在 pod 缩减时仍处理事务恢复。

**流程**

1. 配置资源：

```
apiVersion: wildfly.org/v1alpha1
kind: WildFlyServer
metadata:
  name: eap-helloworld
spec:
  applicationImage: 'eap-helloworld:latest'
  replicas: 1
  resources:
    limits:
      cpu: 500m
      memory: 2Gi
    requests:
      cpu: 100m
      memory: 1Gi
```

**重要**

您必须为 pod 中的容器指定资源限值和请求，以便自动扩展按预期工作。

2. 创建 Horizontal pod 自动缩放器：

```
oc autoscale wildflyserver/eap-helloworld --cpu-percent=50 --min=1 --max=10
```

验证

- 您可以通过检查副本来验证 HPA 行为。副本数量会根据工作负载的增加或减少而进行相应的增加或减少。

```
oc get hpa -w
NAME            REFERENCE                     TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
eap-helloworld   WildFlyServer/eap-helloworld   217%/50%   1         10        1          4s
eap-helloworld   WildFlyServer/eap-helloworld   217%/50%   1         10        4          17s
eap-helloworld   WildFlyServer/eap-helloworld   133%/50%   1         10        8          32s
eap-helloworld   WildFlyServer/eap-helloworld   133%/50%   1         10        10         47s
eap-helloworld   WildFlyServer/eap-helloworld   139%/50%   1         10        10         62s
eap-helloworld   WildFlyServer/eap-helloworld   180%/50%   1         10        10         92s
eap-helloworld   WildFlyServer/eap-helloworld   133%/50%   1         10        10         2m2s
```

**其他资源**

- 使用 pod 横向自动扩展自动扩展 pod

### 9.1.9. OpenShift 上的 Jarkarta 企业 Bean remoting

#### 9.1.9.1. Jakarta Enterprise Beans 在 openShift 上的远程

要使 JBoss EAP 能够在 OpenShift 上的不同 JBoss EAP 集群之间正确地使用企业 bemoting 调用，您必须了解 OpenShift 上的企业 bean remoting 配置选项。

**注意**

在 OpenShift 上部署时，请考虑使用 EAP 操作器。EAP 操作员使用 StatefulSet 来处理企业 bean remoting 和事务恢复处理。StatefulSet 确保持久性存储和网络主机名稳定性，即使 pod 重启后也是如此。

当使用带有事务传播的企业 bean 远程调用联系 JBoss EAP 实例时，需要网络主机名稳定性。即使 pod 重启，JBoss EAP 实例必须可以被同一主机名访问。事务管理器（有状态组件）将持久事务数据绑定到特定的 JBoss EAP 实例。由于事务日志绑定到特定的 JBoss EAP 实例，它必须在同一实例中完成。

要在使用 JDBC 事务日志存储时防止数据丢失，请确保您的数据库提供了数据一致性的读取和写入。当数据库通过多个实例水平扩展时，一致数据读取和写入非常重要。

企业 bean 远程调用者有两个选项来配置远程调用：

- 定义远程出站连接。如需更多信息，请参阅配置远程出站连接。

- 对远程服务器上的 bean 使用编程 JNDI 查找。如需更多信息，请参阅使用远程 Jakarta Enterprise Beans 客户端。

您必须重新配置代表目标节点地址的值，具体取决于企业 bean 远程调用配置方法。

注意

远程调用的目标企业 bean 的名称必须是第一个 pod 的 DNS 地址。

StatefulSet 的行为取决于 pod 的排序。pod 以预定义的顺序命名。例如，如果您将应用扩展为三个副本，则您的 pod 具有名称，如 eap-server-0、eap-server-1 和 eap-server-2。

EAP 操作器也使用 无头服务 来确保将特定的 DNS 主机名分配给 pod。如果应用使用 EAP 操作器，则使用名称（如 eap-server-headless ）创建一个无头服务。在本例中，第一个 pod 的 DNS 名称为 eap-server-0.eap-server-headless。

使用主机名 eap-server-0.eap-server-headless 确保企业 bean 调用达到连接到集群的任何 EAP 实例。bootstrap 连接用于初始化 Jakarta Enterprise Beans 客户端，它会在下一步中收集 EAP 集群的结构。

#### 9.1.9.1.1. 在 OpenShift 上配置 Jakarta Enterprise Beans

您必须配置作为企业级补救的调用者的 JBoss EAP 服务器。目标服务器必须配置一个具有权限的用户，才能接收企业 bean 远程调用。

**先决条件**

- 您已使用 EAP 操作器和支持的 OpenShift S2I 镜像，用于在 OpenShift 中部署和管理 JBoss EAP 应用实例。

- 集群设置正确。有关 JBoss EAP 集群的更多信息，请参阅 集群 部分。

**流程**

1. **在目标服务器上创建一个用户，并有权接收企业 bean 远程调用：**

```
$JBOSS_HOME/bin/add-user.sh
```

2. **配置调用 JBoss EAP 应用服务器。**

a. **使用自定义配置功能，在 $JBOSS_HOME/standalone/configuration 中创建 eap-config.xml 文件。如需更多信息，请参阅 自定义配置。**

b. **使用 wildfly.config.url 属性配置调用器 JBoss EAP 应用服务器：**

```
JAVA_OPTS_APPEND="-
Dwildfly.config.url=$JBOSS_HOME/standalone/configuration/eap-config.xml"
```

**注意**

**如果您的配置使用以下示例，请将 >>PASTE_..._HERE<< 替换为您配置的用户名和密码。**

**配置示例**

```
<configuration>
  <authentication-client xmlns="urn:elytron:1.0">
    <authentication-rules>
      <rule use-configuration="jta">
        <match-abstract-type name="jta" authority="jboss" />
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="jta">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <providers>
          <use-service-loader />
        </providers>
        <set-user-name name="PASTE_USER_NAME_HERE" />
        <credentials>
          <clear-password password="PASTE_PASSWORD_HERE" />
        </credentials>
        <set-mechanism-realm name="ApplicationRealm" />
      </configuration>
```

```
    </authentication-configurations>
  </authentication-client>
</configuration>
```

# 第 10 章 故障排除

Pod 可能会因为许多原因重启，但 JBoss EAP pod 重启的常见原因可能包括 OpenShift 资源约束，特别是内存不足问题。如需有关 OpenShift pod 驱除的更多信息，请参阅 OpenShift 文档。

## 10.1. POD 重启故障排除

默认情况下，将 OpenShift 模板的 JBoss EAP 配置为在遇到内存不足问题等情况时自动重新启动受影响的容器。以下步骤可帮助您诊断和排除内存不足和其他 pod 重启问题。

1. 获取有问题的 pod 的名称。

   您可以使用以下命令查看 pod 名称，以及每个 pod 重启的次数。

   ```
   $ oc get pods
   ```

2. 要诊断 Pod 重启的原因，您可以检查之前 Pod 的 JBoss EAP 日志或 OpenShift 事件。

   a. 要查看以上 pod 的 JBoss EAP 日志，请使用以下命令：

   ```
   oc logs --previous POD_NAME
   ```

   b. 要查看 OpenShift 事件，请使用以下命令：

   ```
   $ oc get events
   ```

3. 如果 Pod 由于资源问题而重启，您可以尝试修改 OpenShift pod 配置，以增加其 资源请求和限值。有关配置 pod 计算资源的更多信息，请参阅 OpenShift 文档。

## 10.2. 使用 JBOSS EAP 管理 CLI 故障排除

JBoss EAP 管理 CLI( *EAP_HOME*/bin/jboss-cli.sh )可从容器内访问，以进行故障排除。

重要

不建议使用 JBoss EAP 管理 CLI 在运行的 pod 中进行配置更改。容器重启时，使用运行中容器中的管理 CLI 进行的任何配置更改都将丢失。

要为 OpenShift 进行 JBoss EAP 的配置更改，请参阅配置 JBoss EAP 服务器和应用程序。

1. 首先打开与正在运行的 pod 的远程 shell 会话。

```
$ oc rsh POD_NAME
```

2. 在远程 shell 会话中运行以下命令启动 JBoss EAP 管理 CLI：

```
$ /opt/server/bin/jboss-cli.sh
```

### 10.3. 在 JBOSS EAP 8 上将 HELM CHART 从 1.0.0 更新至 1.1.0 时进行故障排除

将 Helm Chart 升级到 JBoss EAP 8 的最新版本时可能会出现错误。如果您在升级 Helm Chart 前修改了 immutable 字段，则升级过程中可能会显示以下出错信息：

```
UPGRADE FAILED: cannot patch "<helm-release-name>" with kind Deployment:
Deployment.apps "<helm-release-name>" is invalid: spec.selector: Invalid value:
v1.LabelSelector{MatchLabels:map[string]string{"app.kubernetes.io/instance":"<helm-
release-name>", "app.kubernetes.io/name":"<helm-release-name>"}, MatchExpressions:
[]v1.LabelSelectorRequirement(nil)}: field is immutable
```

要解决这个问题，在运行 helm upgrade < helm-release-name> 命令前，通过运行 oc delete deployment <helm-release-name> 命令来删除部署资源。

# 第 11 章 OPENSHIFT CONTAINER PLATFORM 的参考信息

本节中的内容来自此应用程序镜像的工程文档。内容作为开发目的参考，并在产品文档范围之外进行测试。

## 11.1. 信息环境变量

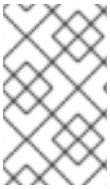
以下环境变量旨在向镜像提供信息，且不应由用户修改：

**表 11.1. 信息环境变量**

| 变量名称 | 描述和值 |
| --- | --- |
| JBOSS_IMAGE_NAME | 镜像名称。<br><br>Values:<br><br>• **jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8** (JDK 17 / RHEL 8) |
| JBOSS_IMAGE_VERSION | 镜像版本。<br><br>值：镜像版本号。如需最新的值，请参阅 Red Hat Container Catalog：<br><br>• JDK 17 / RHEL 8 |
| JBOSS_MODULES_SYSTEM_PKGS | 以逗号分隔的 JBoss EAP 系统模块软件包列表，供应用程序使用。<br><br>Value: **jdk.nashorn.api** |
| STI_BUILDER | 为 **jee** 项目类型提供 OpenShift S2I 支持。<br><br>值 ： **jee** |

## 11.2. 配置环境变量

您可以配置以下环境变量来调整镜像，而无需重新构建镜像。



**注意**

有关此处未列出的其他环境变量，请参阅 **JBoss EAP 文档**。

**表 11.2. 配置环境变量**

| 变量名称 | 描述 |
| --- | --- |
| CLI_GRACEFUL_SHUTDOWN | 如果设置为任何非零长度值，则镜像将阻止使用 **TERM** 信号关闭，并且需要使用 JBoss EAP 管理 CLI 执行 **shutdown** 命令。<br><br>示例 value: **true** |
| CONTAINER_HEAP_PERCENT | 将最大 Java 堆大小设置为可用容器内存的百分比。<br><br>示例值： **0.5** |
| CUSTOM_INSTALL_DIRECTORIES | 以逗号分隔的目录列表，用于在 S2I 过程中为镜像安装和配置工件。<br><br>示例值： **custom,shared** |
| DEFAULT_JMS_CONNECTION_FACTORY | 此值用于为 Jakarta Messaging 连接工厂指定默认 JNDI 绑定，如 **jms-connection-factory='java:jboss/DefaultJMSConnectionFactory'**。<br><br>示例值： **java:jboss/DefaultJMSConnectionFactory** |
| ENABLE_ACCESS_LOG | 启用将信息记录到标准输出频道。<br><br>使用以下方法实施访问信息记录：<br><br>● JBoss EAP 6.4 OpenShift 镜像使用自定义 JBoss Web 访问日志 Valve。<br><br>● JBoss EAP for OpenShift 镜像使用 JBoss EAP 7.4 *开发指南中的* Undertow **AccessLogHandler**。<br><br>默认值为 **false**。 |
| INITIAL_HEAP_PERCENT | 将初始 Java 堆大小设置为最大堆大小的百分比。<br><br>示例值： **0.5** |
| JAVA_OPTS_APPEND | 服务器启动选项.<br><br>示例值： **-Dfoo=bar** |
| JBOSS_MODULES_SYSTEM_PKGS_APPEND | 以逗号分隔的软件包名称列表，该名称将附加到 **JBOSS_MODULES_SYSTEM_PKGS** 环境变量。<br><br>示例值： **org.jboss.byteman** |

| 变量名称 | 描述 |
| --- | --- |
| JGROUPS_CLUSTER_PASSWORD | 用于验证节点的密码，以便允许加入 JGroups 集群。**必需**，在使用 **ASYM_ENCRYPT** JGroups 集群网络流量加密协议时。如果没有设置，则禁用身份验证，集群通信不会加密，并发出警告。可选，在使用 **SYM_ENCRYPT** JGroups 集群流量加密协议时。<br><br>示例值： **mypassword** |
| JGROUPS_ENCRYPT_KEYSTORE | 使用 **SYM_ENCRYPT** JGroups 集群流量加密协议时指定的 secret 中的密钥存储文件的名称。如果没有设置，集群通信不会加密，并发出警告。<br><br>示例值： **jgroups.jceks** |
| JGROUPS_ENCRYPT_KEYSTORE_DIR | 挂载包含密钥存储的 secret 的目录路径。<br><br>示例值： **/etc/jgroups-encrypt-secret-volume** |
| JGROUPS_ENCRYPT_NAME | 使用 **SYM_ENCRYPT** JGroups 集群流量加密协议时与服务器证书关联的名称。如果没有设置，集群通信不会加密，并发出警告。<br><br>示例值： **jgroups** |
| JGROUPS_ENCRYPT_PASSWORD | 使用 **SYM_ENCRYPT** JGroups 集群流量加密协议时用于访问密钥存储和证书的密码。如果没有设置，集群通信不会加密，并发出警告。<br><br>示例值： **mypassword** |
| JGROUPS_ENCRYPT_PROTOCOL | JGroups 协议用于加密集群流量。可以是 **SYM_ENCRYPT** 或 **ASYM_ENCRYPT**。<br><br>默认为 **SYM_ENCRYPT**。<br><br>示例值： **ASYM_ENCRYPT** |
| JGROUPS_PING_PROTOCOL | 用于节点发现的 JGroups 协议。可以是 **dns.DNS_PING** 或 **kubernetes.KUBE_PING**。 |
| MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION | 为了向后兼容，设置为 **true**，从而将 **MyQueue** 和 **MyTopic** 用作物理目的地名称默认值，而不是 **queue/MyQueue** 和 **topic/MyTopic**。 |
| OPENSHIFT_DNS_PING_SERVICE_NAME | 为 DNS 发现机制公开服务器上的 ping 端口的服务名称。<br><br>示例值： **eap-app-ping** |
| OPENSHIFT_DNS_PING_SERVICE_PORT | DNS 发现机制的 ping 端口号。如果没有指定，则会尝试从该服务的 SRV 记录发现端口号，否则会使用默认的 **8888**。<br><br>默认值为 **8888**。 |

| 变量名称 | 描述 |
| --- | --- |
| OPENSHIFT_KUBE_PING_LABELS | 为 Kubernetes 发现机制集群标签选择器。<br>示例值： **app=eap-app** |
| OPENSHIFT_KUBE_PING_NAMESPACE | 为 Kubernetes 发现机制集群项目命名空间。<br>示例值： **myproject** |
| SCRIPT_DEBUG | 如果设置为 **true**，请确保使用 **-x** 选项执行 Bash 脚本，在执行命令时打印命令及其参数。 |

### 11.3. 公开的端口

**表 11.3. 公开的端口**

| 端口号 | 描述 |
| --- | --- |
| 8443 | HTTPS |

### 11.4. DATASOURCES

**数据源根据某些环境变量的值自动创建。**

**最重要的环境变量是 DB_SERVICE_PREFIX_MAPPING，因为它定义了数据源的 JNDI 映射。这个变量允许的值是以逗号分隔的 *POOLNAME-DATABASETYPE=PREFIX* triplets 列表，其中：**

- ***POOLNAME* 用作数据源中的 pool-name。**
- ***DATABASETYPE* 是要使用的数据库驱动程序。**
- ***PREFIX* 是用来配置数据源的环境变量名称中使用的前缀。**

#### 11.4.1. 数据源的 JNDI 映射

**对于每个 *POOLNAME-DATABASETYPE=PREFIX* triplet 在 DB_SERVICE_PREFIX_MAPPING 环境变量中定义的，启动脚本会创建一个单独的数据源，在运行镜像时执行。**

注意

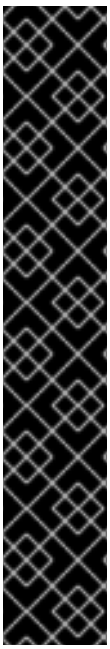
DB_SERVICE_PREFIX_MAPPING 的第一部分（在等号之前）应为小写。

*DATABASETYPE* 决定数据源的驱动程序。

有关配置驱动程序的更多信息，请参阅 模块、驱动程序和通用部署。JDK 8 镜像具有 postgresql 和 mysql 的驱动程序。

警告

不要将任何特殊字符用于 *POOLNAME* 参数。



数据库驱动程序

对使用带有 JBoss EAP 的红帽提供的内部数据源驱动程序的支持现已弃用。红帽建议您为 JBoss EAP 应用使用从数据库供应商获取的 JDBC 驱动程序。

JBoss EAP for OpenShift 镜像不再提供以下内部数据源：

- MySQL
- PostgreSQL

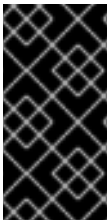
有关安装驱动程序的更多信息，请参阅 模块、驱动程序和通用部署。

有关使用 JBoss EAP 配置 JDBC 驱动程序的更多信息，请参阅 JBoss EAP *配置指南*中的 JDBC 驱动程序。

请注意，如果想将它们添加到置备的服务器中，您还可以创建一个自定义层来安装这些驱动程序和数据源。

#### 11.4.1.1. 数据源配置环境变量

要配置其他数据源属性，请使用以下环境变量：



**重要**

务必将以下变量名称中的 ***POOLNAME***、***DATABASETYPE*** 和 ***PREFIX*** 的值替换为适当的值。本节和 **Datasources** 部分中描述了这些可替换值。

| 变量名称 | 描述 |
| --- | --- |
| *POOLNAME_DATABASETYPE*_SERVICE_HOST | 定义在数据源的 **connection-url** 属性中使用的数据库服务器的主机名或 IP 地址。<br><br>示例值： **192.168.1.3** |
| *POOLNAME_DATABASETYPE*_SERVICE_PORT | 定义数据源的端口。<br><br>示例值： **5432** |
| *PREFIX*_BACKGROUND_VALIDATION | 当设置为 **true** 数据库连接时，在使用前会在后台线程中定期验证。默认为 **false**，表示默认情况下启用 **validate-on-match** 方法。 |
| *PREFIX*_BACKGROUND_VALIDATION_MILLIS | 指定验证的频率，以毫秒为单位，当启用 **background-validation** 数据库连接验证机制时 (***PREFIX*_BACKGROUND_VALIDATION** 变量设为 **true**)。默认值为 **10000**。 |
| *PREFIX*_CONNECTION_CHECKER | 指定连接检查器类，用于验证正在使用的特定数据库的连接。<br><br>示例值： **org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker** |
| *PREFIX*_DATABASE | 定义数据源的数据库名称。<br><br>示例值： **myDatabase** |
| *PREFIX*_DRIVER | 为数据源定义 Java 数据库驱动程序。<br><br>示例值： **postgresql** |

| 变量名称 | 描述 |
| --- | --- |
| *PREFIX*_EXCEPTION_SORTER | 指定用于在致命数据库连接例外后正确检测和清理的异常分类器类。<br><br>示例值：**org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter** |
| *PREFIX*_JNDI | 定义数据源的 JNDI 名称。默认为 **java:jboss/datasources/*POOLNAME_DATABASETYPE***，其中 ***POOLNAME*** 和 ***DATABASETYPE*** 从上述 triplet 获取。如果要覆盖默认生成的 JNDI 名称，则此设置很有用。<br><br>示例值：**java:jboss/datasources/test-postgresql** |
| *PREFIX*_JTA | 为非 XA 数据源定义 Jakarta Transactions 选项。XA datasources 已默认已经支持 Jakarta 事务。<br><br>默认值为 **true**。 |
| *PREFIX*_MAX_POOL_SIZE | 定义数据源的最大池大小选项。<br><br>示例值 **：20** |
| *PREFIX*_MIN_POOL_SIZE | 定义数据源的最小池大小选项。<br><br>示例值：**1** |
| *PREFIX*_NONXA | 将数据源定义为非 XA 数据源。默认值为 **false**。 |
| *PREFIX*_PASSWORD | 定义数据源的密码。<br><br>示例值：**password** |
| *PREFIX*_TX_ISOLATION | 为数据源定义 java.sql.Connection 事务隔离级别。<br><br>示例值：**TRANSACTION_READ_UNCOMMITTED** |
| *PREFIX*_URL | 定义数据源的连接 URL。<br><br>示例值：**jdbc:postgresql://localhost:5432/postgresdb** |
| *PREFIX*_USERNAME | 定义数据源的用户名。<br><br>示例值：**admin** |

#### 11.4.1.2. 例子

这些示例演示了 DB_SERVICE_PREFIX_MAPPING 环境变量创建的值如何影响数据源创建。

#### 11.4.1.2.1. 单个映射

考虑值 test-postgresql=TEST。

这会创建一个数据源，其名称为 java:jboss/datasources/test_postgresql。此外，所有所需的设置（如 password 和 username）都应作为带有 TEST_ prefix 的环境变量提供，如 TEST_USERNAME 和 TEST_PASSWORD。

#### 11.4.1.2.2. 多个映射

您可以指定多个数据源映射。

**注意**

始终使用逗号分隔多个数据源映射。

对于 DB_SERVICE_PREFIX_MAPPING 环境变量，请考虑以下值： cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL.

这会创建以下两个数据源：

1. java:jboss/datasources/test_mysql
2. java:jboss/datasources/cloud_postgresql

然后，您可以使用 TEST_MYSQL 前缀来配置 MySQL 数据源的用户名和密码等内容，如 TEST_MYSQL_USERNAME。对于 PostgreSQL 数据源，请使用 CLOUD_ 前缀，如 CLOUD_USERNAME。

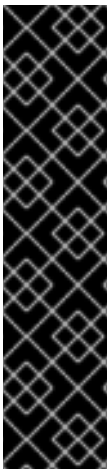
## 11.5. 集群

### 11.5.1. 配置 JGroups Discovery Mechanism

要在 OpenShift 上启用 JBoss EAP 集群，请在 JBoss EAP 配置中配置 JGroups 协议堆栈，以使用 kubernetes.KUBE_PING 或 dns.DNS_PING 发现机制。

虽然您可以使用自定义 standalone.xml 配置文件，但建议您使用 环境变量 在镜像构建中配置 JGroups。

以下说明使用环境变量为 OpenShift 镜像配置 JBoss EAP 的发现机制。



重要

如果您使用 Helm Chart 在 JBoss EAP for OpenShift 镜像上部署应用程序，则默认发现机制为 dns.DNS_PING。

dns.DNS_PING 和 kubernetes.KUBE_PING 发现机制相互兼容。不能从两个独立的子集群构成超级集群，一个使用 dns.DNS_PING 机制进行发现，另一个使用 kubernetes.KUBE_PING 机制。同样，在执行滚动升级时，在源和目标集群中，发现机制需要相同。

#### 11.5.1.1. 配置 KUBE_PING

使用 KUBE_PING JGroups 发现机制：

1. JGroups 协议堆栈必须配置为使用 KUBE_PING 作为发现机制。

   您可以将 JGROUPS_PING_PROTOCOL 环境变量设置为 kubernetes.KUBE_PING ：

   ```
   JGROUPS_PING_PROTOCOL=kubernetes.KUBE_PING
   ```

2. KUBERNETES_NAMESPACE 环境变量必须设置为 OpenShift 项目名称。例如：

   ```
   KUBERNETES_NAMESPACE=PROJECT_NAME
   ```

3. 应设置 KUBERNETES_LABELS 环境变量。这应该 与服务级别 上设置的标签匹配。如果没有设置，则应用程序以外的 pod （在您的命名空间中）将尝试加入。例如：

```
KUBERNETES_LABELS=application=APP_NAME
```

4. 必须向服务帐户授予在 下运行 pod 的授权，才能访问 Kubernetes 的 REST API。这可通过 OpenShift CLI 完成。以下示例使用当前项目命名空间中的 default 服务帐户：

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

在项目命名空间中使用 eap-service-account ：

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):eap-service-account -n $(oc project -q)
```

**注意**

如需有关向服务帐户添加策略的更多信息，请参阅准备 OpenShift 以部署应用程序。

#### 11.5.1.2. 配置 DNS_PING

使用 DNS_PING JGroups 发现机制：

1. JGroups 协议堆栈必须配置为使用 DNS_PING 作为发现机制。

   您可以将 JGROUPS_PING_PROTOCOL 环境变量设置为 dns.DNS_PING ：

```
JGROUPS_PING_PROTOCOL=dns.DNS_PING
```

2. OPENSHIFT_DNS_PING_SERVICE_NAME 环境变量必须设置为集群的 ping 服务名称。

```
OPENSHIFT_DNS_PING_SERVICE_NAME=PING_SERVICE_NAME
```

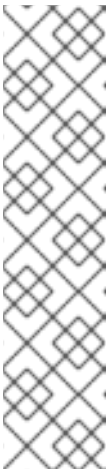
3. OPENSHIFT_DNS_PING_SERVICE_PORT 环境变量应设置为公开 ping 服务的端口号。DNS_PING 协议尝试从 SRV 记录中识别端口，否则默认为 8888。

```
OPENSHIFT_DNS_PING_SERVICE_PORT=PING_PORT
```

4. 必须定义公开 ping 端口的 ping 服务。这个服务应该是无头(ClusterIP=None)，且必须具有以下内容：

   a. 端口必须命名为。

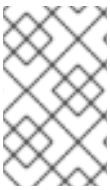
   b. 该服务必须使用 service.alpha.kubernetes.io/tolerate-unready-endpoints 和 publishNotReadyAddresses 属性标注，两者都设置为 true。



注意

- 使用 service.alpha.kubernetes.io/tolerate-unready-endpoints 和 publishNotReadyAddresses 属性来确保 ping 服务在两个更新的 OpenShift 版本中正常工作。
- 省略这些注解会导致每个节点在启动过程中组成自己的 "cluster of one"。然后，每个节点会在启动后将其集群合并到其他节点的集群中，因为其他节点在启动后不会被检测到。

```
kind: Service
apiVersion: v1
spec:
  publishNotReadyAddresses: true
  clusterIP: None
  ports:
  - name: ping
    port: 8888
  selector:
    deploymentConfig: eap-app
metadata:
  name: eap-app-ping
  annotations:
    service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"
    description: "The JGroups ping port for clustering."
```



注意

DNS_PING 不需要对服务帐户进行任何修改，并使用默认权限。

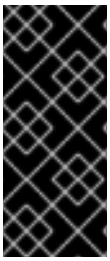
### 11.5.2. 配置 JGroups 以加密集群流量

要在 OpenShift 上为 JBoss EAP 加密集群流量，您必须在 JBoss EAP 配置中配置 JGroups 协议堆

栈，以使用 SYM_ENCRYPT 或 ASYM_ENCRYPT 协议。

虽然您可以使用自定义 standalone.xml 配置文件，但建议您使用 环境变量 在镜像构建中配置 JGroups。

以下说明使用环境变量为 JBoss EAP for OpenShift 镜像配置集群流量加密的协议。



重要

SYM_ENCRYPT 和 ASYM_ENCRYPT 协议不相互兼容。不能从两个独立的子集群构成超级集群，一个使用 SYM_ENCRYPT 协议加密集群流量，另一个使用 ASYM_ENCRYPT 协议。同样，在执行滚动升级时，对源和目标集群的协议需要相同。

#### 11.5.2.1. 配置 SYM_ENCRYPT

使用 SYM_ENCRYPT 协议加密 JGroups 集群流量：

1. JGroups 协议堆栈必须配置为使用 SYM_ENCRYPT 作为加密协议。

   您可以将 JGROUPS_ENCRYPT_PROTOCOL 环境变量设置为 SYM_ENCRYPT ：

   ```
   JGROUPS_ENCRYPT_PROTOCOL=SYM_ENCRYPT
   ```

2. JGROUPS_ENCRYPT_KEYSTORE_DIR 环境变量必须设置为挂载包含密钥存储的机密的目录路径。例如：

   ```
   JGROUPS_ENCRYPT_KEYSTORE_DIR=/etc/jgroups-encrypt-secret-volume
   ```

3. JGROUPS_ENCRYPT_KEYSTORE 环境变量必须设置为指定 secret 中的密钥存储文件的名称。如果没有设置，集群通信不会加密，并发出警告。例如：

   ```
   JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
   ```

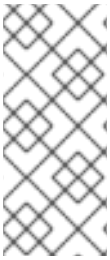
4. JGROUPS_ENCRYPT_NAME 环境变量必须设置为与服务器证书关联的名称。如果没有设置，集群通信不会加密，并发出警告。例如：

```
JGROUPS_ENCRYPT_NAME=jgroups
```

5. **JGROUPS_ENCRYPT_PASSWORD 环境变量必须设置为用于访问密钥存储和证书的密码。如果没有设置，集群通信不会加密，并发出警告。例如：**

```
JGROUPS_ENCRYPT_PASSWORD=mypassword
```

#### 11.5.2.2. 配置 ASYM_ENCRYPT



**注意**

**JBoss EAP 8.0 包括 ASYM_ENCRYPT 协议的新版本。以前版本的协议已弃用。如果您指定了 JGROUPS_CLUSTER_PASSWORD 环境变量，则会使用已弃用的协议版本，并在 pod 日志中输出警告信息。**

**要使用 ASYM_ENCRYPT 协议加密 JGroups 集群流量，请将 ASYM_ENCRYPT 指定为加密协议，并将它配置为使用 elytron 子系统中配置的密钥存储。**

```
-e JGROUPS_ENCRYPT_PROTOCOL="ASYM_ENCRYPT" \
-e JGROUPS_ENCRYPT_NAME="encrypt_name" \
-e JGROUPS_ENCRYPT_PASSWORD="encrypt_password" \
-e JGROUPS_ENCRYPT_KEYSTORE="encrypt_keystore" \
-e JGROUPS_CLUSTER_PASSWORD="cluster_password"
```

### 11.5.3. 扩展 pod 的注意事项

**根据 JGroups 中的发现机制，启动节点将搜索现有集群协调器节点。如果在给定超时中没有找到协调器节点，则起始节点会假定它是第一个成员，并占用协调器状态。**

**当多个节点同时启动时，它们假定是创建具有多个分区的分割集群的第一个成员。例如，使用 DeploymentConfig API 扩展从 0 到 2 个 pod 可能会导致分割集群创建。要避免这种情况，您需要启动第一个 pod，然后扩展到所需的 pod 数量。**

**注意**

**默认情况下，JBoss EAP Operator 使用 StatefulSet API，它会按顺序创建 pod，即防止创建分割集群。**

## 11.6. 原生健康检查

JBoss EAP for OpenShift 镜像实施 OpenShift 中包含的存活度和就绪度探测。如需更多信息，请参阅 *OpenShift Container Platform 开发人员指南* 中的 存活度和就绪度探测。

下表显示了这些健康检查要传递所需的值。如果状态不是下面找到的值，则检查会失败，并根据镜像的重启策略重启镜像。

**表 11.4. 存活度和就绪度检查**

| 已执行的测试 | Liveness | 就绪 |
| --- | --- | --- |
| 服务器状态 | 任何状态 | 运行中 |
| 引导错误 | 无 | 无 |
| 部署状态 [a] | 不适用或没有 **failed** 项 | 不适用或没有 **failed** 项 |
| 原生健康检查 | **UP** | **UP** |

[a] N/A 仅在没有部署时才有效状态。

## 11.7. 消息传递

### 11.7.1. 配置外部 Red Hat AMQ Broker

您可以使用环境变量为 OpenShift 镜像配置 JBoss EAP，以连接到外部 Red Hat AMQ 代理。

## 11.8. 安全域

若要配置新的安全域，用户必须定义 SECDOMAIN_NAME 环境变量。

这会导致在 环境变量后创建名为 的安全域。用户也可以定义以下环境变量来自定义域：

**表 11.5. 安全域**

| 变量名称 | 描述 |
| --- | --- |
| SECDOMAIN_NAME | 定义一个额外的安全域。<br>示例值： **myDomain** |

| 变量名称 | 描述 |
| --- | --- |
| SECDOMAIN_PASSWORD_STACKING | 如果定义，则启用了 **password-stacking** 模块选项，并设置为值 **useFirstPass**。<br><br>示例 value: **true** |
| SECDOMAIN_LOGIN_MODULE | 要使用的登录模块。<br><br>默认为 **UsersRoles** |
| SECDOMAIN_USERS_PROPERTIES | 包含用户定义的属性文件的名称。<br><br>默认为 **users.properties** |
| SECDOMAIN_ROLES_PROPERTIES | 包含角色定义的属性文件的名称。<br><br>默认为 **roles.properties** |

### 11.9. HTTPS 环境变量

| 变量名称 | 描述 |
| --- | --- |
| HTTPS_NAME | 如果与 **HTTPS_PASSWORD** 和 **HTTPS_KEYSTORE** 一起定义，请启用 HTTPS 并设置 SSL 名称。<br><br>如果您使用 **keytool -genkey** 命令创建它，这应该是您的密钥存储的别名名称。<br><br>示例值： **example.com** |
| HTTPS_PASSWORD | 如果与 **HTTPS_NAME** 和 **HTTPS_KEYSTORE** 一起定义，请启用 HTTPS 并设置 SSL 密钥密码。<br><br>示例值： **passw0rd** |
| HTTPS_KEYSTORE | 如果与 **HTTPS_PASSWORD** 和 **HTTPS_NAME** 一起定义，请启用 HTTPS，并将 SSL 证书密钥文件设置为 ***EAP_HOME*/standalone/configuration** 下的相对路径<br><br>示例值： **ssl.key** |

### 11.10. 管理环境变量

**表 11.6. 管理环境变量**

| 变量名称 | 描述 |
| --- | --- |
| ADMIN_USERNAME | 如果定义了这个和 **ADMIN_PASSWORD**，则用于 JBoss EAP 管理用户名。<br><br>示例值： **eapadmin** |
| ADMIN_PASSWORD | 指定 **ADMIN_USERNAME** 的密码。<br><br>示例值： **passw0rd** |

### 11.11. S2I

**该镜像包括 S2I 脚本和 Maven。Maven 目前仅支持将应当部署到 OpenShift 上基于 JBoss EAP 的容器（或相关的/级镜像）的应用程序的构建工具。**

**目前只支持 WAR 部署。**

#### 11.11.1. 自定义配置

**可以为镜像添加自定义配置文件。放入 configuration/ 目录中的所有文件都将复制到 *EAP_HOME*/standalone/configuration/ 中。例如，要覆盖镜像中使用的默认配置，只需将自定义 standalone.xml 添加到 configuration/ 目录中。有关这样的部署，请参阅示例。**

##### 11.11.1.1. 自定义模块

**可以添加自定义模块。modules/ 目录中的所有文件将复制到 *EAP_HOME*/modules/ 中。有关这样的部署，请参阅示例。**

#### 11.11.2. Deployment Artifacts

**默认情况下，将部署源 target 目录中的工件。要从不同的目录部署，请在 BuildConfig 定义中设置 ARTIFACT_DIR 环境变量。ARTIFACT_DIR 是一个以逗号分隔的列表。例如：ARTIFACT_DIR=app1/target,app2/target,app3/target**

#### 11.11.3. 工件存储库镜像

**Maven 中的存储库包含各种类型的构建工件和依赖项，例如，所有项目 JAR、库 JAR、插件或任何其他项目特定工件。它还指定在执行 S2I 构建时下载工件的位置。除了使用中央存储库外，组织还会部署本**

地自定义镜像存储库是一种常见的做法。

使用镜像的好处包括：

- 同步镜像的可用性，其地理位置更接近且更快。
- 能够更好地控制存储库内容。
- 跨不同团队（开发人员、CI）共享工件，而无需依赖公共服务器和存储库。
- 改进了构建时间。

通常，存储库管理器可以充当镜像的本地缓存。假设存储库管理器已在 https://10.0.0.1:8443/repository/internal/ 部署并可访问，S2I 构建可以通过向应用的构建配置提供 MAVEN_MIRROR_URL 环境变量来使用该管理器，如下所示：

1. 识别要针对应用 MAVEN_MIRROR_URL 变量的构建配置的名称。

```
oc get bc -o name
buildconfig/eap
```

2. 使用 MAVEN_MIRROR_URL 环境变量更新 eap 的构建配置。

```
oc env bc/eap MAVEN_MIRROR_URL="https://10.0.0.1:8443/repository/internal/"
buildconfig "eap" updated
```

3. 验证设置。

```
oc env bc/eap --list
# buildconfigs eap
MAVEN_MIRROR_URL=https://10.0.0.1:8443/repository/internal/
```

4. 调度应用的新构建。

**注意**

在应用构建过程中，您会注意到 Maven 依赖项是从存储库管理器拉取的，而不是从默认的公共存储库拉取。另外，在构建完成后，您会看到镜像已填充构建期间检索和使用的所有依赖项。

#### 11.11.3.1. 安全工件存储库镜像 URL

为了防止通过 Maven 存储库进行"man-in-the-middle"攻击，JBoss EAP 需要使用安全 URL 进行工件存储库镜像 URL。

URL 应该指定安全 http("https")和安全端口。

默认情况下，如果您指定了非安全 URL，则返回错误。您可以使用属性 -Dinsecure.repositories=WARN 来覆盖此行为。

### 11.11.4. 脚本

**run**

此脚本使用 openshift-launch.sh 脚本，该脚本使用 standalone.xml 配置配置和启动 JBoss EAP。

**assemble**

此脚本使用 Maven 构建源，创建软件包(WAR)，并将它移到 *EAP_HOME*/standalone/deployments 目录。

### 11.11.5. 自定义脚本

您可以在启动 pod 时添加自定义脚本，然后再启动 JBoss EAP。

您可以在启动 pod 时添加有效的脚本，包括 CLI 脚本。

从镜像启动 JBoss EAP 时可使用两个选项，包括脚本：

- 挂载要作为 postconfigure.sh 执行的 configmap

- 在提名安装目录中添加一个 **install.sh** 脚本

#### 11.11.5.1. 挂载 configmap 以执行自定义脚本

当您要在运行时将自定义脚本挂载到现有镜像（换句话说，一个已构建的镜像）时，挂载 configmap。

挂载 configmap：

1. 使用您要包含在 **postconfigure.sh** 中的内容创建 configmap。

   例如，在项目根目录中创建一个名为 **extensions** 的目录，使其包含 **postconfigure.sh** 和 **extensions.cli** 的脚本，并运行以下命令：

   ```
   $ oc create configmap jboss-cli --from-file=postconfigure.sh=extensions/postconfigure.sh --
   from-file=extensions.cli=extensions/extensions.cli
   ```

2. 通过部署控制器(dc)将 configmap 挂载到 pod 中。

   ```
   $ oc set volume dc/eap-app --add --name=jboss-cli -m /opt/server/extensions -t configmap --
   configmap-name=jboss-cli --default-mode='0755' --overwrite
   ```

**postconfigure.sh 示例**

```
#!/usr/bin/env bash
set -x
echo "Executing postconfigure.sh"
$JBOSS_HOME/bin/jboss-cli.sh --file=$JBOSS_HOME/extensions/extensions.cli
```

**extensions.cli 示例**

```
embed-server --std-out=echo  --server-config=standalone.xml
:whoami
quit
```

-

#### 11.11.5.2. 使用 install.sh 执行自定义脚本

当您要在镜像构建时将脚本包含在镜像中时，请使用 install.sh。

使用 install.sh 执行自定义脚本：

1. 在 s2i 构建期间使用的项目的 git 存储库中，创建名为 .s2i 的目录。

2. 在 s2i 目录中，添加名为 environment 的文件，其内容如下：

```
$ cat .s2i/environment
CUSTOM_INSTALL_DIRECTORIES=extensions
```

3. 创建名为 extensions 的目录。

4. 在 extensions 目录中，使用类似以下内容的内容创建 postconfigure.sh 文件（将占位符代码替换为适合您环境的代码）：

```
$ cat extensions/postconfigure.sh
#!/usr/bin/env bash
echo "Executing patch.cli"
$JBOSS_HOME/bin/jboss-cli.sh --file=$JBOSS_HOME/extensions/some-cli-example.cli
```

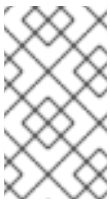
5. 在 extensions 目录中，使用类似以下内容的内容创建 install.sh 文件（将占位符代码替换为适合您环境的相应代码）：

```
$ cat extensions/install.sh
#!/usr/bin/env bash
set -x
echo "Running $PWD/install.sh"
injected_dir=$1
# copy any needed files into the target build.
cp -rf ${injected_dir} $JBOSS_HOME/extensions
```

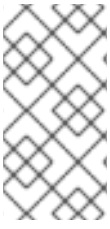
### 11.11.6. 环境变量

**您可以通过向 s2i build 命令提供环境变量来影响构建的执行方式。可以提供的环境变量有：**

**表 11.7. s2i 环境变量**

| 变量名称 | 描述 |
| --- | --- |
| ARTIFACT_DIR | 此目录中的 **.war**、**.ear** 和 **.jar** 文件将复制到 **deployments/** 目录中。<br><br>示例值： **target** |
| ENABLE_GENERATE_DEFAULT_DATASOURCE | 可选。当包含在值 **true** 中时，服务器将使用默认的数据源进行调配。否则，不会包含默认数据源。 |
| GALLEON_PROVISION_LAYERS | 可选。指示 S2I 流程调配指定的层。该值是一个要调配的、以逗号分隔的层列表，包括一个基础层和任意数量的 decorator 层。<br><br>示例值： **jaxrs** |
| GALLEON_PROVISION_CHANNELS | 这是以逗号分隔的 JBoss EAP 频道清单列表。JBoss EAP 频道清单通过 **groupid:artifactId:[version]** 标识。<br><br>  <br>**注意**<br><br>版本是可选的，这意味着将检索最新的频道清单。对于 JBoss EAP 8.0，请使用此频道 **org.jboss.eap.channels:eap-8.0**。 |
| GALLEON_PROVISION_FEATURE_PACKS | 构建环境变量，为您的 S2I 镜像指定自定义 Galleon 功能 pack。例如： **org.jboss.eap:wildfly-ee-galleon-pack:[version],org.jboss.eap.cloud:eap-cloud-galleon-pack:[version]**.<br><br>  <br>**注意**<br><br>当您设置 **GALLEON_PROVISION_CHANNELS=org.jboss.eap.channels:eap-8.0** 时，不需要 feature-packs 版本。 |
| HTTP_PROXY_HOST | 用于 Maven 的 HTTP 代理的主机名或 IP 地址。<br><br>示例值： **192.168.1.1** |
| HTTP_PROXY_PORT | 用于 Maven 的 HTTP 代理的 TCP 端口。<br><br>示例值： **8080** |

| 变量名称 | 描述 |
| --- | --- |
| HTTP_PROXY_USERNAME | 如果提供给 **HTTP_PROXY_PASSWORD**，请为 HTTP 代理使用凭证。<br><br>示例值： **myusername** |
| HTTP_PROXY_PASSWORD | 如果由 **HTTP_PROXY_USERNAME** 提供，请为 HTTP 代理使用凭证。<br><br>示例值： **mypassword** |
| HTTP_PROXY_NONPROXYHOSTS | 如果提供，配置的 HTTP 代理将忽略这些主机。<br><br>示例值： **some.example.org\|*.example.net** |
| MAVEN_ARGS | 覆盖在构建期间提供给 Maven 的参数。<br><br>示例值： **-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package** |
| MAVEN_ARGS_APPEND | 在构建期间附加提供给 Maven 的用户参数。<br><br>示例值： **-Dfoo=bar** |
| MAVEN_MIRROR_URL | 用于配置 Maven Mirror/repository Manager 的 URL。<br><br>示例值： **https://10.0.0.1:8443/repository/internal/**<br><br>请注意，指定的 URL 应安全。详情请参阅 安全工件存储库镜像 URL。 |
| MAVEN_CLEAR_REPO | （可选）在构建后清除本地 Maven 存储库。<br><br>如果镜像中存在的服务器与本地缓存高度耦合，则不会删除缓存并输出警告。<br><br>示例 value: **true** |
| APP_DATADIR | 如果定义，则复制数据文件的源中的目录。<br><br>示例值： **mydata** |
| DATA_DIR | 复制 **$APP_DATADIR** 的镜像中数据的目录。<br><br>示例值： ***EAP_HOME*/data** |



**注意**

**如需更多信息，请参阅在 OpenShift Container Platform 上构建并运行 JBoss EAP 应用程序，它使用 Maven 和 JBoss EAP 中包含的 S2I 脚本作为 OpenShift 镜像。**

## 11.12. 不支持的事务恢复场景

- OpenShift 不支持 JTS 事务。
- OpenShift 不支持 XTS 事务。
- OpenShift 不支持一些第三方用于事务完成和崩溃恢复流程的 XATerminator 接口。
- 不支持通过 JBoss Remoting 事务处理。

**注意**

使用 EAP operator 支持通过 JBoss Remoting 传播的事务。

## 11.13. 包括的 JBOSS 模块

下表列出了 JBoss EAP for OpenShift 镜像中包含的 JBoss 模块。

表 11.8. 包括的 JBoss 模块

| JBoss 模块 |
| --- |
| org.jboss.as.clustering.common |
| org.jboss.as.clustering.jgroups |
| org.jboss.as.ee |
| org.jgroups |
| org.openshift.ping |
| net.oauth.core |

## 11.14. EAP OPERATOR：API 信息

EAP Operator 引进了以下 API：

### 11.14.1. WildFlyServer

**WildFlyServer 定义自定义 JBoss EAP 资源。**

**表 11.9. WildFlyServer**

| 字段 | 描述 | Scheme | 必填 |
|---|---|---|---|
| **metadata** | 标准对象元数据 | ObjectMeta v1 meta | false |
| **spec** | JBoss EAP 部署所需的规格。 | **WildFlyServerSpec** | true |
| **status** | 最近观察到的 JBoss EAP 部署 状态。只读。 | WildFlyServerStatus | false |

### 11.14.2. WildFlyServerList

**WildFlyServerList 定义 JBoss EAP 部署的列表。**

**表 11.10. 表**

| 字段 | 描述 | Scheme | 必填 |
|---|---|---|---|
| **metadata** | 标准列表的元数据 | metav1.ListMeta | false |
| **items** | **WildFlyServer** 列表 | **WildFlyServer** | true |

### 11.14.3. WildFlyServerSpec

**WildFlyServerSpec 是 JBoss EAP 资源所需行为的规范。**

**它使用一个带有 pod 规格的 StatefulSet，它挂载由 /opt/jboss/wildfly/standalone/data 的存储指定的卷。**

**表 11.11. WildFlyServerSpec**

| 字段 | 描述 | Scheme | 必填 |
|---|---|---|---|
| **applicationImage** | 要部署的应用程序镜像的名称 | 字符串 | false |

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **replicas** | 应用程序所需的副本数 | int32] | true |
| **standaloneConfigMap** | spec 指定如何从 **ConfigMap** 读取独立配置。 | **StandaloneConfigMapSpec** | false |
| **资源** | **resources** spec 用于指定 Stateful Set 的 request 或 limits。如果省略，则使用命名空间默认值。 | **资源** | false |
| **securityContext** | **securityContext** spec，用于定义由 Stateful Set 创建的 pod 容器的特权和访问控制设置。如果省略，则使用默认权限。如需更多信息，请参阅 securityContext | ***corev1.SecurityContext** | false |
| **storage** | 存储 spec 以指定如何使用存储。如果省略，则使用 **EmptyDir** （不会在 pod 重启后保留数据） | **StorageSpec** | false |
| **serviceAccountName** | 用来运行 JBoss EAP pod 的 ServiceAccount 名称 | 字符串 | false |
| **envFrom** | 来自 **configMap** 或 secret 的容器中的环境变量列表 | **corev1.EnvFromSource** | false |
| **env** | 容器中存在的环境变量列表 | corev1.EnvVar | false |
| **secrets** | 要作为容器中的卷挂载的 secret 名称列表。每个 secret 都作为只读卷挂载到 **/etc/secrets/<secret name>** | 字符串 | false |
| **configMaps** | 要作为容器中的卷挂载的 **ConfigMap** 名称列表。每个 **ConfigMap** 在 **/etc/configmaps/<config map name>** 下挂载为一个只读卷。 | 字符串 | false |

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **disableHTTPRoute** | 禁用创建到应用程序服务的 HTTP 端口的路由（如果省略则为false） | 布尔值 | false |
| **sessionAffinity** | 如果同一客户端 IP 的连接每次都传递到相同的 JBoss EAP 实例/pod（如果忽略，则为错误） | 布尔值 | false |

### 11.14.4. Resources

**Resources 定义 WildflyServer 资源的资源。如果没有定义 Resources 字段，或者 Request 或 Limits 为空，则此资源会从 StatefulSet 中删除。这个资源的描述是一个标准容器资源，它使用 corev1.ResourceRequirements 方案。**

### 11.14.5. StorageSpec

**StorageSpec 为 WildFlyServer 资源定义存储。如果没有定义 EmptyDir 或 volumeClaimTemplate，则会使用默认的 EmptyDir。**

**EAP Operator 使用这个 StorageSpec 的信息配置 StatefulSet，以挂载专用于 JBoss EAP 的 standalone/data 目录的卷，以保留自己的数据。例如，事务日志）。如果使用了 EmptyDir，则该数据不会在 pod 重启后保留。如果部署在 JBoss EAP 上的应用依赖于事务，请指定 volumeClaimTemplate，以便在 pod 重启时重复使用相同的持久性卷。**

**表 11.12. 表**

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **emptyDir** | JBoss EAP **StatefulSet** 使用的 **EmptyDirVolumeSource** | corev1.EmptyDirVolumeSource | false |

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **volumeClaimTemplate** | **用于配置** 资源要求的 PersistentVolumeClaim 规格，以存储 JBoss EAP 独立数据目录。模板的名称派生自 **WildFlyServer** 名称。对应的卷以 **ReadWriteOnce** 访问模式挂载。 | corev1.PersistentVolumeClaim | false |

### 11.14.6. StandaloneConfigMapSpec

**StandaloneConfigMapSpec 定义 JBoss EAP 单机配置如何从 ConfigMap 读取。如果省略，JBoss EAP 会使用其镜像中的 standalone.xml 配置。**

**表 11.13. StandaloneConfigMapSpec**

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **name** | 包含独立配置 XML 文件的 **ConfigMap** 名称。 | 字符串 | true |
| key | 值为独立配置 XML 文件的 **ConfigMap** 键。如果省略，spec 会查找 **standalone.xml** 键。 | 字符串 | false |

### 11.14.7. WildFlyServerStatus

**WildFlyServerStatus 是 JBoss EAP 部署的最新观察状态。只读。**

**表 11.14. WildFlyServerStatus**

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **replicas** | 应用程序的实际副本数 | int32 | true |
| **selector** | HorizontalPodAutoscaler 使用的 pod 选择器 | 字符串 | true |
| **主机** | 路由到应用程序 HTTP 服务的主机 | 字符串 | true |
| **pods** | pod 的状态 | PodStatus | true |

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **scalingdownPods** | 缩减清理进程的 pod 数量 | int32 | true |

### 11.14.8. PodStatus

**PodStatus 是运行 JBoss EAP 应用的 pod 的最新观察状态。**

**表 11.15. PodStatus**

| 字段 | 描述 | Scheme | 必填 |
| --- | --- | --- | --- |
| **name** | pod 的名称 | 字符串 | true |
| **podIP** | 分配给 pod 的 IP 地址 | 字符串 | true |
| **state** | 缩减过程中 pod 的状态。状态默认为 ACTIVE，这意味着它服务于请求。 | 字符串 | false |

***更新于 2024-02-08***