Red Hat Single Sign-On 7.6

Red Hat Single Sign-On for OpenShift

适用于 Red Hat Single Sign-On 7.6

Last Updated: 2024-06-05

Red Hat Single Sign-On 7.6 Red Hat Single Sign-On for OpenShift

适用于 Red Hat Single Sign-On 7.6

法律通告

摘要

本指南包含 OpenShift 的 Red Hat Single Sign-On 7.6 开始提供的基本信息和说明

目录

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 RED HAT SINGLE SIGN-ON FOR OPENSHIFT 简介

1.1. 什么是红帽单点登录？

Red Hat Single Sign-On 是一个集成登录解决方案，作为用于 OpenShift 容器化镜像的红帽 JBoss 中间件。Red Hat Single Sign-On for OpenShift 镜像提供了一个身份验证服务器，供用户集中登录、注销、注册和管理 Web 应用程序、移动应用程序和 RESTful Web 服务的用户帐户。

Red Hat Single Sign-On for OpenShift 在以下平台上提供：x86_64、IBM Z 和 IBM Power Systems。

1.2. 比较：红帽 OPENSHIFT 镜像单点登录与 RED HAT SINGLE SIGN-ON

Red Hat Single Sign-On for OpenShift 镜像版本号 7.6.9 基于 Red Hat Single Sign-On 7.6.9。Red Hat Single Sign-On for OpenShift 镜像和 Red Hat Single Sign-On 之间有一些重要的区别，应该考虑：

Red Hat Single Sign-On for OpenShift 镜像包括了红帽单点登录的所有功能。另外，启用了 Red Hat Single Sign-On 的 JBoss EAP 镜像会在其相应的 **web.xml** 文件中自动处理带有 **<auth-method>KEYCLOAK</auth-method>** 或 **<auth-method>KEYCLOAK-SAML</auth-method>** 的 ***.war*** 部署中的 OpenID Connect 或 SAML 客户端注册。

1.3. 用于这个软件的模板

红帽通过 Red Hat Single Sign-On for OpenShift 镜像版本号 7.6.9 提供多个 OpenShift 应用程序模板。这些模板定义了开发基于 Red Hat Single Sign-On 7.6.9 服务器的部署所需的资源。模板主要可分为两种类别：直通模板和再加密模板。还存在其他一些杂项模板。

1.3.1. passthrough 模板

这些模板要求预先存在 HTTPS、JGroups 密钥存储和 Red Hat Single Sign-On 服务器的信任存储。它们使用 passthrough TLS 终止保护 TLS 通信。

- ***sso76-ocp3-https***, ***sso76-ocp4-https***: Red Hat Single Sign-On 7.6.9 支持同一 pod 上的内部 H2 数据库。
- ***sso76-ocp3-postgresql***, ***sso76-ocp4-postgresql***: Red Hat Single Sign-On 7.6.9，由独立 pod 上的临时 PostgreSQL 数据库支持。
- ***sso76-ocp3-postgresql-persistent***, ***sso76-ocp4-postgresql-persistent***: Red Hat Single Sign-On 7.6.9 在单独的 pod 上由持久 PostgreSQL 数据库支持。

注意

使用带有 MySQL / MariaDB 数据库的 Red Hat Single Sign-On 的模板已被删除，且自 Red Hat Single Sign-On 版本 7.4 后不可用。

1.3.2. 再加密模板

OpenShift 3.x 和 OpenShift 4.x 提供了独立的再加密模板

1.3.2.1. OpenShift 3.x

OpenShift 3.x 模板使用 service-ca.crt CA 捆绑包文件作为 Service Serving 证书 Secret 的一部分，生成 TLS 证书和密钥来提供安全内容。Red Hat Single Sign-On truststore 也会自动创建，其中包含

/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt CA 证书文件，用于为 HTTPS 密钥存储签名。

Red Hat Single Sign-On 服务器的信任存储预先填充 Java 系统路径中所有已知的可信 CA 证书文件。这些模板使用重新加密 TLS 终止来保护 TLS 通信。JGroups 集群流量使用 **AUTH** 协议进行身份验证，并使用 **ASYM_ENCRYPT** 协议加密。

- ***sso76-ocp3-x509-https***: Red Hat Single Sign-On 7.6.9 带有自动生成的 HTTPS 密钥存储和红帽单点登录信任存储，由内部 H2 数据库支持。
- ***sso76-ocp3-x509-postgresql-persistent***: Red Hat Single Sign-On 7.6.9 带有自动生成的 HTTPS 密钥存储和红帽单点登录信任存储，由持久 PostgreSQL 数据库支持。

1.3.2.2. OpenShift 4.x

OpenShift 4.x 模板使用 内部服务 x509 证书 secret 来自动创建用于提供安全内容的 HTTPS 密钥存储。这些模板使用一个新的服务 CA 捆绑包，它包括了 **service.beta.openshift.io/inject-cabundle=true** ConfigMap 定义。

Red Hat Single Sign-On 服务器的信任存储预先填充 Java 系统路径中所有已知的可信 CA 证书文件。这些模板使用重新加密 TLS 终止来保护 TLS 通信。JGroups 集群流量使用 **AUTH** 协议进行身份验证，并使用 **ASYM_ENCRYPT** 协议加密。

- ***sso76-ocp4-x509-https***: Red Hat Single Sign-On 7.6.9 带有自动生成的 HTTPS 密钥存储和红帽单点登录信任存储，由内部 H2 数据库支持。**ASYM_ENCRYPT** Groups 协议用于加密集群流量。
- ***sso76-ocp4-x509-postgresql-persistent***: Red Hat Single Sign-On 7.6.9 带有自动生成的 HTTPS 密钥存储和红帽单点登录信任存储，由持久 PostgreSQL 数据库支持。**ASYM_ENCRYPT** Groups 协议用于加密集群流量。

1.3.3. 其他模板

还提供了与 Red Hat Single Sign-On 集成的其他模板：

- ***eap64-sso-s2i*** ：已启用红帽 JBoss 企业应用平台 6.4 的红帽单点登录.
- ***eap71-sso-s2i*** ：已启用红帽 JBoss 企业应用平台 7.1 的红帽单点登录.
- Data ***virt63-secure-s2i*** ：已启用红帽 JBoss 数据虚拟化 6.3 的红帽单点登录.

这些模板包含特定于 Red Hat Single Sign-On 的环境变量，在部署时启用 Red Hat Single Sign-On 客户端注册。

其他资源

- 自动和手动红帽单点登录客户端注册方法
- passthrough TLS 终止、OpenShift 3.11
- 再加密 TLS 终止、OpenShift 3.11
- 安全路由, OpenShift 4.11

1.4. 版本兼容性和支持

有关 OpenShift 镜像版本兼容性的详情，请查看 支持的配置 页面。

注意

7.0 和 7.5 之间的 OpenShift 镜像版本 Red Hat Single Sign-On 已被弃用，它们将不再接收镜像和应用程序模板的更新。

要部署新应用，请使用 Red Hat Single Sign-On for OpenShift 镜像的 7.6 版本以及特定于此镜像版本的应用程序模板。

第 2 章 为 OPENSHIFT 配置 RED HAT SINGLE SIGN-ON

2.1. 将 RED HAT SINGLE SIGN-ON 用于 OPENSHIFT 镜像流和应用程序模板

从安全的 Red Hat Registry: registry.redhat.io 中拉取（需要身份验证）的 Red Hat JBoss Middleware for OpenShift 镜像。要获得内容，您需要使用红帽帐户登录到 registry。

要在 OpenShift 等共享环境中使用 ***registry.redhat.io*** 中的容器镜像，建议管理员使用 Registry 服务帐户（也称为身份验证令牌）来代替个人的红帽客户门户网站凭证。

流程

1. 要创建 Registry Service Account，请导航至 Registry Service Account Management Application，如有必要登录。
2. 在 ***Registry Service Accounts*** 页面中，点 ***Create Service Account***。
3. 为服务帐户提供一个名称，如 ***registry.redhat.io-sa***。它将以固定、随机字符串开头。
   a. 输入服务帐户的描述，如 ***服务帐户使用 registry.redhat.io 中的容器镜像***。
   b. 点 ***Create***。
4. 创建 Service Account 后，点 ***Registry Service Accounts*** 页面中的 ***Account name*** 列中的 ***registry.redhat.io-sa*** 链接。
5. 最后，单击 ***OpenShift Secret*** 选项卡，并执行该页面中列出的所有步骤。

如需更多信息，请参阅 Red Hat Container Registry 身份验证 文档。

流程

1. 确保您已作为集群管理员或具有项目管理员访问权限的用户登陆到全局 **openshift** 项目：
2. 根据您的 OpenShift Container Platform 版本选择一个命令。
   a. 如果您在 master 主机（某些）上运行基于 OpenShift Container Platform v3 的集群实例，请执行以下操作：

   ```
   $ oc login -u system:admin
   ```

   b. 如果您正在运行基于 OpenShift Container Platform v4 的集群实例，使用 kubeadmin 户身份登录到 CLI ：

   ```
   $ oc login -u kubeadmin -p password https://openshift.example.com:6443
   ```

3. 运行以下命令，为 **openshift** 项目中的 OpenShift 更新一组 Red Hat Single Sign-On 7.6.9 资源。
   如果使用 OpenShift 3.x 集群，请使用以下命令：

   ```
   $ for resource in sso76-image-stream.json \
     passthrough/ocp-3.x/sso76-ocp3-https.json \
     passthrough/ocp-3.x/sso76-ocp3-postgresql.json \
   ```

```
  passthrough/ocp-3.x/sso76-ocp3-postgresql-persistent.json \
  reencrypt/ocp-3.x/sso76-ocp3-x509-https.json \
  reencrypt/ocp-3.x/sso76-ocp3-x509-postgresql-persistent.json
do
  oc replace -n openshift --force -f \
  https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-image/sso76-dev/templates/${resource}
done
```

如果使用 OpenShift 4.x 集群，请使用下列命令：

```
$ for resource in sso76-image-stream.json \
  passthrough/ocp-4.x/sso76-ocp4-https.json \
  passthrough/ocp-4.x/sso76-ocp4-postgresql.json \
  passthrough/ocp-4.x/sso76-ocp4-postgresql-persistent.json \
  reencrypt/ocp-4.x/sso76-ocp4-x509-https.json \
  reencrypt/ocp-4.x/sso76-ocp4-x509-postgresql-persistent.json
do
  oc replace -n openshift --force -f \
  https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-image/sso76-dev/templates/${resource}
done
```

4. 运行以下命令，在 **openshift** 项目中安装 Red Hat Single Sign-On 7.6.9 OpenShift 镜像流：

```
$ oc -n openshift import-image rh-sso-7/sso76-openshift-rhel8:7.6 --from=registry.redhat.io/rh-sso-7/sso76-openshift-rhel8:7.6 --confirm
```

2.2. 部署 RED HAT SINGLE SIGN-ON 镜像

2.2.1. 准备部署

流程

1. 使用包含 *cluster:admin* 角色的用户登录 OpenShift CLI。
2. 创建一个新项目

```
$ oc new-project sso-app-demo
```

3. 将 **view** 角色添加到 **default** 服务帐户。这可让服务帐户查看 **sso-app-demo** 命名空间中的所有资源，这是管理集群所必需的。

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

2.2.2. 使用应用程序模板部署 Red Hat Single Sign-On 镜像

您可以使用以下接口之一部署模板：

- OpenShift CLI
- OpenShift 3.x Web 控制台

- OpenShift 4.x Web 控制台

2.2.2.1. 使用 OpenShift CLI 部署模板

先决条件

- 执行 将 Red Hat Single Sign-On 用于 OpenShift 镜像流和应用模板 中所述的步骤。

流程

1. 列出可用的红帽单点登录应用程序模板：

```
$ oc get templates -n openshift -o name | grep -o 'sso76.\+' | sort
sso76-ocp3-https
sso76-ocp3-postgresql
sso76-ocp3-postgresql-persistent
sso76-ocp3-x509-https
sso76-ocp3-x509-postgresql-persistent
sso76-ocp4-https
sso76-ocp4-postgresql
sso76-ocp4-postgresql-persistent
sso76-ocp4-x509-https
sso76-ocp4-x509-postgresql-persistent
```

2. 部署所选选项：

```
$ oc new-app --template=sso76-ocp4-x509-https
--> Deploying template "openshift/sso76-ocp4-x509-https" to project sso-app-demo

    Red Hat Single Sign-On 7.6 (Ephemeral)
    ---------
    An example Red Hat Single Sign-On 7 application. For more information about using this
template, see <link xlink:href="https://github.com/jboss-openshift/application-
templates">https://github.com/jboss-openshift/application-templates</link>.

    A new Red Hat Single Sign-On service has been created in your project. The admin
username/password for accessing the master realm using the Red Hat Single Sign-On
console is IACfQO8v/nR7llVSVb4Dye3TNRbXoXhRpAKTmiCRc. The HTTPS keystore used
for serving secure content, the JGroups keystore used for securing JGroups
communications, and server truststore used for securing Red Hat Single Sign-On requests
were automatically created using OpenShift's service serving x509 certificate secrets.

    * With parameters:
      * Application Name=sso
      * JGroups Cluster Password=jg0Rssom0gmHBnooDF3Ww7V4Mu5RymmB #
generated
      * Datasource Minimum Pool Size=
      * Datasource Maximum Pool Size=
      * Datasource Transaction Isolation=
      * ImageStream Namespace=openshift
      * Red Hat Single Sign-On Administrator Username=IACfQO8v # generated
      * Red Hat Single Sign-On Administrator
Password=nR7llVSVb4Dye3TNRbXoXhRpAKTmiCRc # generated
      * Red Hat Single Sign-On Realm=
      * Red Hat Single Sign-On Service Username=
```

```
        * Red Hat Single Sign-On Service Password=
        * Container Memory Limit=1Gi

--> Creating resources ...
    service "sso" created
    service "secure-sso" created
    service "sso-ping" created
    route "sso" created
    route "secure-sso" created
    deploymentconfig "sso" created
--> Success
    Run 'oc status' to view your app.
```

2.2.2.2. 使用 OpenShift 3.x Web 控制台部署模板

先决条件

- 执行 将 Red Hat Single Sign-On 用于 OpenShift 镜像流和应用模板 中所述的步骤。

流程

1. 登录 OpenShift Web 控制台，再选择 **sso-app-demo** 项目空间。
2. 单击 **Add to Project**，然后单击 **Browse Catalog** 以列出默认镜像流和模板。
3. 使用 **Filter by Keyword** 搜索栏，将列表限制为与 *sso* 匹配的列表。您可能需要单击 **中间件**，然后 **集成** 以显示所需的应用程序模板。
4. 选择 Red Hat Single Sign-On 应用模板。本例使用 ***Red Hat Single Sign-On 7.6(Ephemeral)***。
5. 在 **Information** 步骤中点 **Next**。
6. 从 **Add to Project** 下拉菜单中，选择 *sso-app-demo* 项目空间。然后单击**"下一步"**。
7. 在 **绑定** 步骤中选择 **Do not bind**。点 **Create** 来继续。
8. 在 **结果** 步骤中，单击 **Continue to the project overview**链接以验证部署的状态。

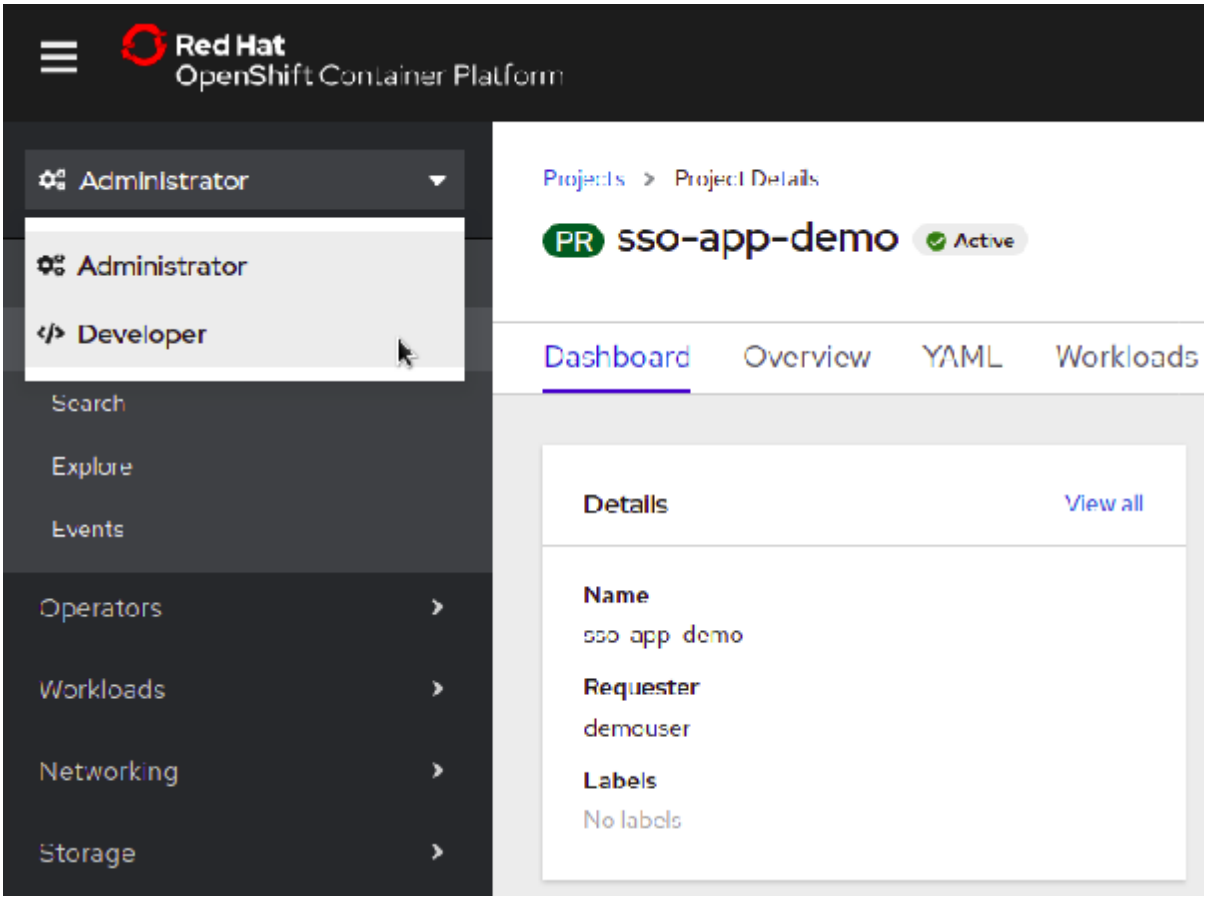
2.2.2.3. 使用 OpenShift 4.x Web 控制台部署模板

先决条件

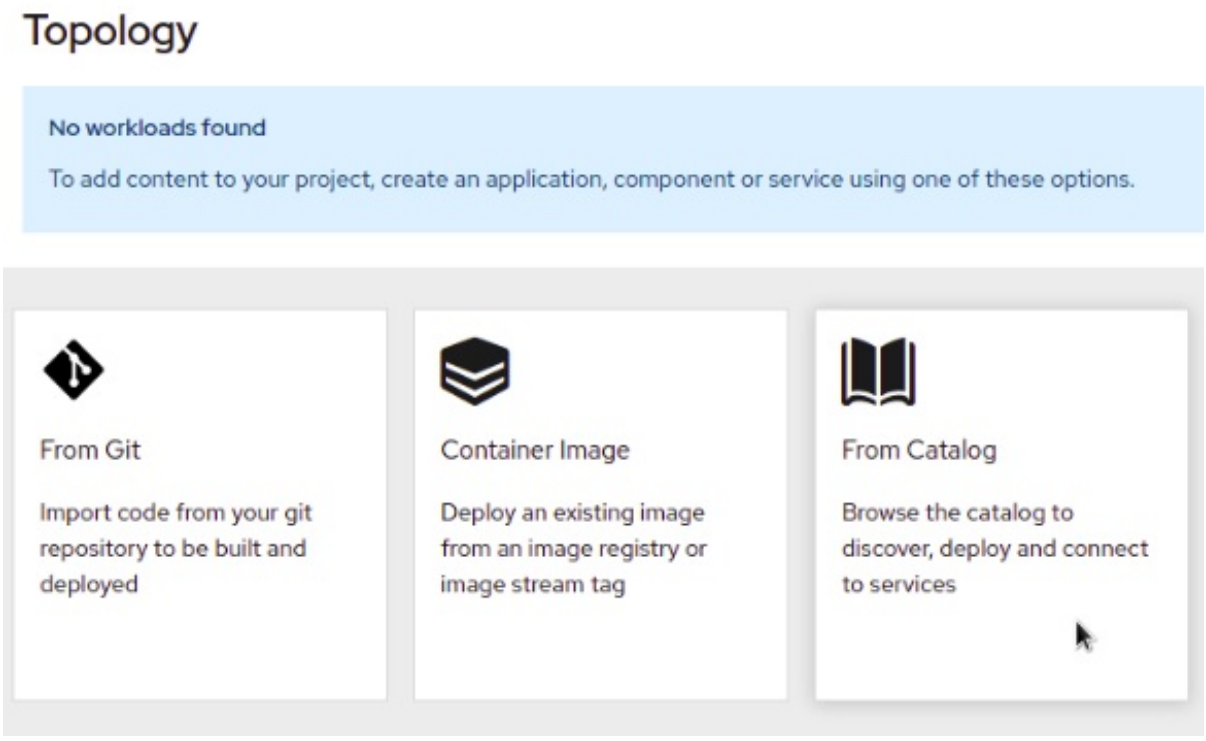
- 执行 将 Red Hat Single Sign-On 用于 OpenShift 镜像流和应用模板 中所述的步骤。

流程

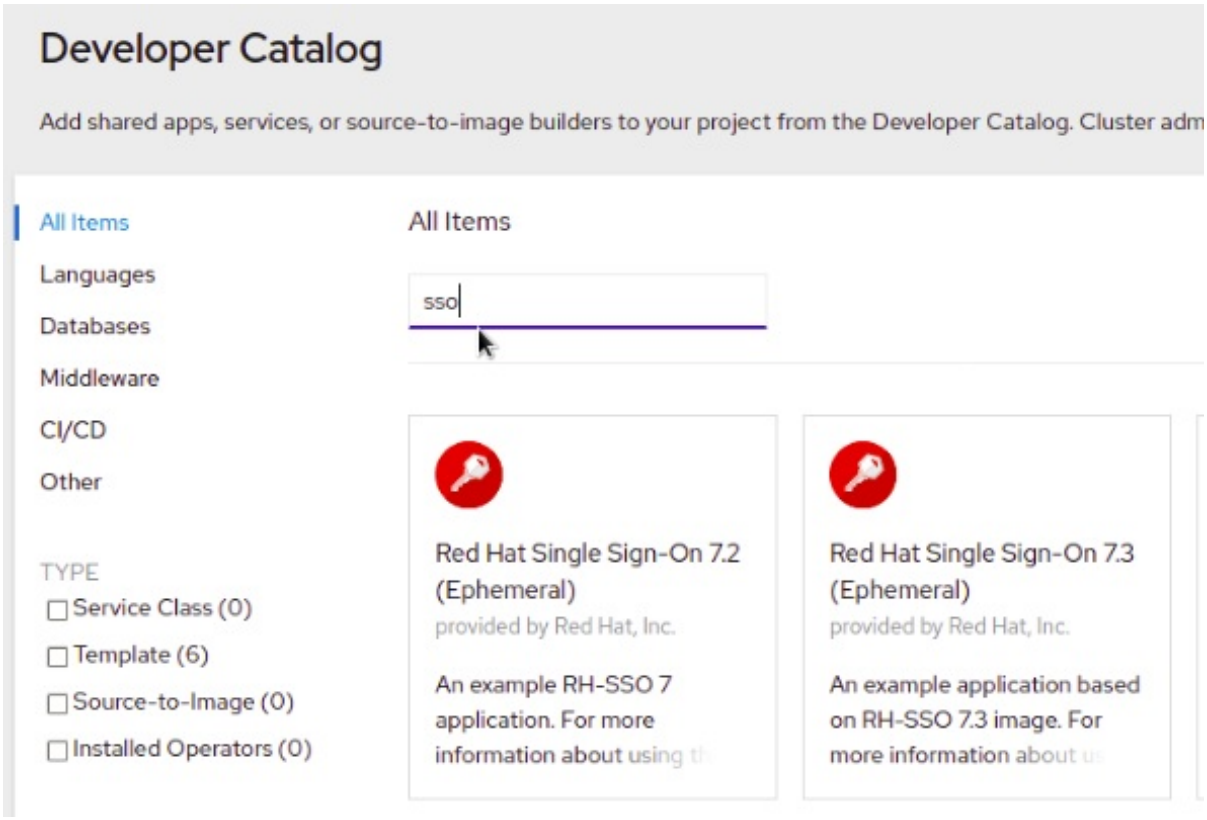
1. 登录 OpenShift Web 控制台，再选择 *sso-app-demo* 项目空间。
2. 在左侧边栏中，单击 **Administrator** 选项卡，然后单击 **< /> Developer**。



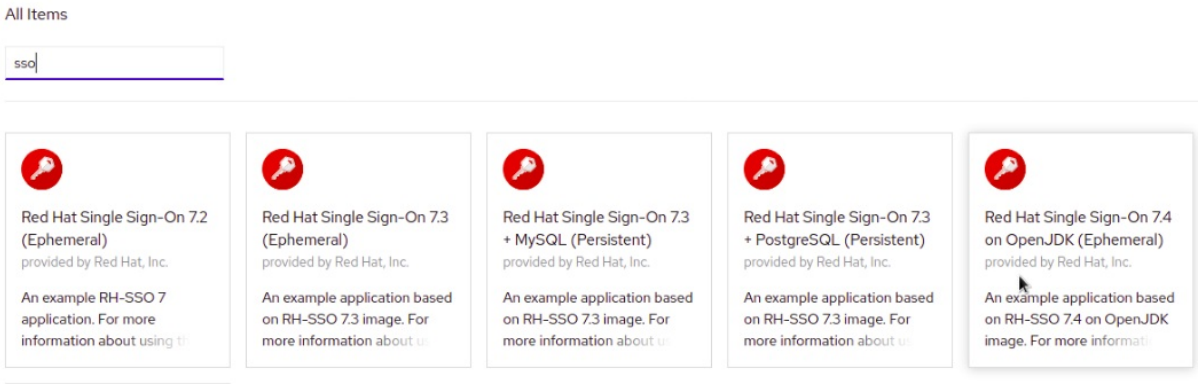
3. 单击 **From Catalog**。



4. 搜索 **sso**。



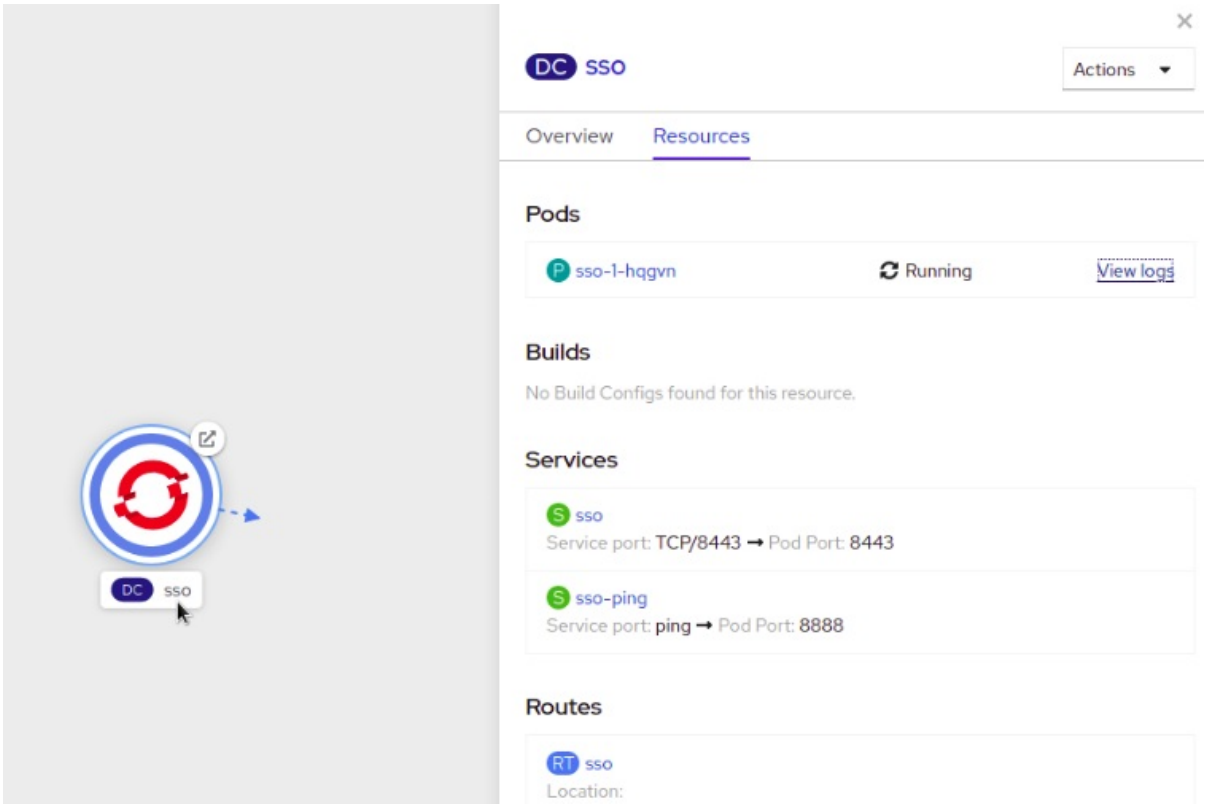
5. 在 **OpenJDK(Ephemeral)上选择一个模板，如 Red Hat Single Sign-On 7.6**。



6. 点 **Instantiate Template**。

7. 如果需要，调整模板参数并点 **Create**。

8. 验证已部署了用于 OpenShift 镜像的 Red Hat Single Sign-On。



2.3. 访问 RED HAT SINGLE SIGN-ON POD 的管理员控制台

流程

1. 部署模板后，识别可用的路由。

```
$ oc get routes
NAME     HOST/PORT
sso      sso-sso-app-demo.openshift.example.com
```

2. 访问红帽单点登录管理控制台。

```
https://sso-sso-app-demo.openshift.example.com/auth/admin
```

3. 提供 管理员帐户 的登录凭据。

第 3 章 执行高级步骤

本章论述了高级流程，如为 Red Hat Single Sign-On 服务器设置密钥存储和信任存储、创建管理员帐户以及可用红帽单点登录客户端注册方法的概述，以及配置集群的指导。

3.1. 部署 PASSTHROUGH TLS 终止模板

您可以使用这些模板进行部署。它们需要 HTTPS、JGroups 密钥存储和红帽单点登录服务器信任存储，因此可用于使用您的自定义 HTTPS、JGroups 密钥存储和红帽单点登录服务器信任存储来实例化 Red Hat Single Sign-On 服务器信任存储。

3.1.1. 准备部署

流程

1. 使用包含 *cluster:admin* 角色的用户登录 OpenShift CLI。
2. 创建一个新项目

```
$ oc new-project sso-app-demo
```

3. 将 **view** 角色添加到 **default** 服务帐户。这可让服务帐户查看 sso-app-demo 命名空间中的所有资源，这是管理集群所必需的。

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

3.1.2. 为红帽单点登录服务器创建 HTTPS 和Groups 密钥存储，以及 Truststore

在此过程中，***openssl*** 工具包用于生成 CA 证书来签署 HTTPS 密钥存储，并为 Red Hat Single Sign-On 服务器创建一个信任存储。***其密钥*** 工具 是 **Java Development Kit 中包含的软件包**，然后用于生成这些密钥存储的自签名证书。

使用重新加密 TLS 终止的 Red Hat Single Sign- On 应用程序模板 **不需要** 或 **预期** HTTPS 和 JGroups 密钥存储和 Red Hat Single Sign-On 服务器信任存储已预先准备好。

再加密模板使用 OpenShift 的内部服务服务证书机密来自动创建 HTTPS 和 JGroups 密钥存储。也会自动创建 Red Hat Single Sign-On 服务器信任存储。它会预先填充 Java 系统路径中找到的所有已知可信 CA 证书文件。

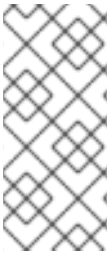
注意

如果要使用现有的 HTTPS /Groups 密钥存储来置备 Red Hat Single Sign-On 服务器，请改为使用一些直通模板。

先决条件

使用 passthrough TLS 终止的红帽单点登录应用程序模板需要部署以下内容：

- 用于加密 https 流量的 HTTPS 密钥存储，
- JGroups 密钥存储 用于为集群中节点之间的通信加密，以及
- Red Hat Single Sign-On 服务器 信任存储用于保护 Red Hat Single Sign-On 请求



注意

对于生产环境，红帽建议您使用从验证证书颁发机构(CA)处购买的 SSL 证书进行 SSL 加密的连接(HTTPS)。

如需了解有关如何使用自签名或购买 SSL 证书的密钥存储的更多信息，请参阅 JBoss Enterprise Application Platform 安全指南。

创建 HTTPS 密钥存储：

流程

1. 生成 CA 证书。选择并记住密码。提供相同的密码，当使用以下 CA 证书对证书签名请求进行签名 时：

```
$ openssl req -new -newkey rsa:4096 -x509 -keyout xpaas.key -out xpaas.crt -days 365 -subj
"/CN=xpaas-sso-demo.ca"
```

2. 为 HTTPS 密钥存储生成一个私钥。提供 **mykeystorepass** 作为密钥存储密码：

```
$ keytool -genkeypair -keyalg RSA -keysize 2048 -dname "CN=secure-sso-sso-app-
demo.openshift.example.com" -alias jboss -keystore keystore.jks
```

3. 为 HTTPS 密钥存储生成证书签名请求。提供 **mykeystorepass** 作为密钥存储密码：

```
$ keytool -certreq -keyalg rsa -alias jboss -keystore keystore.jks -file sso.csr
```

4. 使用 CA 证书签署证书签名请求。提供 用于生成 CA 证书的同一密码 ：

```
$ openssl x509 -req -extfile  <(printf "subjectAltName=DNS:secure-sso-sso-app-
demo.openshift.example.com") -CA xpaas.crt -CAkey xpaas.key -in sso.csr -out sso.crt -days
365 -CAcreateserial
```

注意

要使前面的命令在一个行中正常工作，命令包含进程替换（< **() 语法**）。确保您当前的 shell 环境支持此类语法。否则，您可能会遇到与 **意外令牌 '(' 消息相关的语法错误**。

5. 将 CA 证书导入到 HTTPS 密钥存储中。提供 **mykeystorepass** 作为密钥存储密码。**回复 是 这个证书？[no]:** 问题：

```
$ keytool -import -file xpaas.crt -alias xpaas.ca -keystore keystore.jks
```

6. 将已签名的证书签名请求导入到 HTTPS 密钥存储中。提供 **mykeystorepass** 作为密钥存储密码：

```
$ keytool -import -file sso.crt -alias jboss -keystore keystore.jks
```

为 JGroups 密钥存储生成安全密钥：

提供 **密钥密码** ：

```
$ keytool -genseckey -alias secret-key -storetype JCEKS -keystore jgroups.jceks
```

将 CA 证书导入到新的 Red Hat Single Sign-On 服务器信任存储中：

提供 **mykeystorepass** 作为信任存储密码。**回复 是 这个证书？[no]:** 问题：

```
$ keytool -import -file xpaas.crt -alias xpaas.ca -keystore truststore.jks
```

3.1.3. 创建 secret

流程

您创建名为 secrets 的对象，供 OpenShift 用于存放敏感信息，如密码或密钥存储。

1. 为 HTTPS 和 JGroups 密钥存储创建 secret，以及 上一节中 生成的红帽单点登录服务器信任存储。

```
$ oc create secret generic sso-app-secret --from-file=keystore.jks --from-file=jgroups.jceks --from-file=truststore.jks
```

2. 将这些 secret 链接到默认服务帐户，用于运行 Red Hat Single Sign-On Pod。

```
$ oc secrets link default sso-app-secret
```

其他资源

- 什么是 secret？
- 默认项目服务帐户和角色

3.1.4. 使用 OpenShift CLI 部署 Passthrough TLS 模板

在您创建了 keystores 和 secrets 后，使用 **oc** 命令部署一个透传的 TLS termination 模板。

3.1.4.1. oc 命令指南

在以下 **oc** 命令
中，*SSO_ADMIN_USERNAME*、*SSO_ADMIN_PASSWORD*、*HTTPS_PASSWORD*、*JGROUPS_ENCRYP*
和 *SSO_TRUSTSTORE_PASSWORD* 变量的值与 *sso76-ocp4-https* Red Hat Single Sign-On 应用程序模板中的默认值匹配。

对于生产环境，红帽建议您查阅机构的现场策略，以获取为红帽单点登录服务器的管理员用户帐户生成强用户名和密码的指导，以及红帽单点登录服务器的信任存储。

另外，在创建模板时，使密码与您在创建密钥存储时提供的密码相匹配。如果您使用了不同的用户名或密码，请修改模板中的参数的值以匹配您的环境。

注意

您可以使用以下 ***keytool*** 命令确定与证书关联的别名名称。***keytool*** 是 Java Development Kit 中包含的软件包。

```
$ keytool -v -list -keystore keystore.jks | grep Alias
Enter keystore password:  mykeystorepass
Alias name: xpaas.ca
Alias name: jboss
```

```
$ keytool -v -list -keystore jgroups.jceks -storetype jceks | grep Alias
Enter keystore password:  password
Alias name: secret-key
```

以下命令中的 ***SSO_ADMIN_USERNAME***、***SSO_ADMIN_PASSWORD*** 和 ***SSO_REALM*** 模板参数都是可选的。

3.1.4.2. oc 命令示例

```
$ oc new-app --template=sso76-ocp4-https \
 -p HTTPS_SECRET="sso-app-secret" \
 -p HTTPS_KEYSTORE="keystore.jks" \
 -p HTTPS_NAME="jboss" \
 -p HTTPS_PASSWORD="mykeystorepass" \
 -p JGROUPS_ENCRYPT_SECRET="sso-app-secret" \
 -p JGROUPS_ENCRYPT_KEYSTORE="jgroups.jceks" \
 -p JGROUPS_ENCRYPT_NAME="secret-key" \
 -p JGROUPS_ENCRYPT_PASSWORD="password" \
 -p SSO_ADMIN_USERNAME="admin" \
 -p SSO_ADMIN_PASSWORD="redhat" \
 -p SSO_REALM="demorealm" \
 -p SSO_TRUSTSTORE="truststore.jks" \
 -p SSO_TRUSTSTORE_PASSWORD="mykeystorepass" \
 -p SSO_TRUSTSTORE_SECRET="sso-app-secret"
--> Deploying template "openshift/sso76-ocp4-https" to project sso-app-demo

    Red Hat Single Sign-On 7.6.9 (Ephemeral with passthrough TLS)
    ---------
    An example Red Hat Single Sign-On 7 application. For more information about using this
template, see https://github.com/jboss-openshift/application-templates.

    A new Red Hat Single Sign-On service has been created in your project. The admin
username/password for accessing the master realm via the Red Hat Single Sign-On console is
admin/redhat. Please be sure to create the following secrets: "sso-app-secret" containing the
keystore.jks file used for serving secure content; "sso-app-secret" containing the jgroups.jceks file
used for securing JGroups communications; "sso-app-secret" containing the truststore.jks file used
for securing Red Hat Single Sign-On requests.

    * With parameters:
      * Application Name=sso
      * Custom http Route Hostname=
      * Custom https Route Hostname=
      * Server Keystore Secret Name=sso-app-secret
      * Server Keystore Filename=keystore.jks
      * Server Keystore Type=
```

```
        * Server Certificate Name=jboss
        * Server Keystore Password=mykeystorepass
        * Datasource Minimum Pool Size=
        * Datasource Maximum Pool Size=
        * Datasource Transaction Isolation=
        * JGroups Secret Name=sso-app-secret
        * JGroups Keystore Filename=jgroups.jceks
        * JGroups Certificate Name=secret-key
        * JGroups Keystore Password=password
        * JGroups Cluster Password=yeSppLfp # generated
        * ImageStream Namespace=openshift
        * Red Hat Single Sign-On Administrator Username=admin
        * Red Hat Single Sign-On Administrator Password=redhat
        * Red Hat Single Sign-On Realm=demorealm
        * Red Hat Single Sign-On Service Username=
        * Red Hat Single Sign-On Service Password=
        * Red Hat Single Sign-On Trust Store=truststore.jks
        * Red Hat Single Sign-On Trust Store Password=mykeystorepass
        * Red Hat Single Sign-On Trust Store Secret=sso-app-secret
        * Container Memory Limit=1Gi

--> Creating resources ...
    service "sso" created
    service "secure-sso" created
    service "sso-ping" created
    route "sso" created
    route "secure-sso" created
    deploymentconfig "sso" created
--> Success
    Run 'oc status' to view your app.
```

其他资源

- 透传 TLS 终止

3.2. 自定义 RED HAT SINGLE SIGN-ON 服务器的主机名

hostname SPI 引进了一种灵活方法来为 Red Hat Single Sign-On 服务器配置主机名。默认主机名供应商 **默认**为。这个供应商通过 **原始请求** 供应商提供增强的功能，它现已被弃用。如果没有额外的设置，它会使用请求标头来确定主机名。类似于原始的 **request** 供应商。

有关 **默认** 提供程序的配置选项，请参考" 服务器安装和配置指南 "。**frontendUrl** 选项可以通过 **SSO_FRONTEND_URL** 环境变量进行配置。

注意

为实现向后兼容，如果设置了 **SSO_HOSTNAME**，则会忽略 **SSO_FRONTEND_URL** 设置。

主机名提供程序的另一个选项 **是固定的**，它允许配置固定主机名。后者可确保只能使用有效主机名，并允许内部应用程序通过替代 URL 调用 Red Hat Single Sign-On 服务器。

流程

运行以下命令，为 Red Hat Single Sign-On 服务器设置 **固定** 主机名 SPI 供应商：

1. 将 ***SSO_HOSTNAME*** 环境变量设置为 Red Hat Single Sign-On 服务器所需的主机名，为 OpenShift 镜像部署 Red Hat Single Sign-On。

```
$ oc new-app --template=sso76-ocp4-x509-https \
  -p SSO_HOSTNAME="rh-sso-server.openshift.example.com"
```

2. 识别红帽单点登录服务的路由名称。

```
$ oc get routes
NAME     HOST/PORT
sso      sso-sso-app-demo.openshift.example.com
```

3. 更改 **host:** 字段，使其与指定为上述 ***SSO_HOSTNAME*** 环境变量值指定的主机名匹配。

注意

根据需要，在以下命令中调整 **rh-sso-server.openshift.example.com** 值。

```
$ oc patch route/sso --type=json -p '[{"op": "replace", "path": "/spec/host", "value": "rh-sso-server.openshift.example.com"}]'
```

如果成功，上一个命令会返回以下输出：

```
route "sso" patched
```

3.3. 连接到外部数据库

红帽单点登录可以配置为连接到外部（到 OpenShift 集群）数据库。为了达到此目的，您需要修改 **sso-{database name}** Endpoints 对象以指向正确的地址。该流程在 OpenShift 手册中进行了说明。

最简单的方法是从模板创建红帽单点登录，然后修改 Endpoints 对象。您可能还需要更新 DeploymentConfig 中的一些数据源配置变量。完成后，只需推出新的部署即可。

3.4. 集群

3.4.1. 配置 JGroups 发现机制

OpenShift 中的集群是通过以下两种发现机制之一来实现的： **Kubernetes** 或 **DNS**。可以设置它们：

- 通过使用 **< kubernetes.KUBE_PING/> 或 < dns.DNS_PING /> 元素直接在 standalone-openshift. xml** 配置文件中配置 JGroups 协议堆栈，
- 或者，通过指定 **JGROUPS_PING_PROTOCOL** 环境变量，它可以设置为 **dns.DNS_PING** 或 **kubernetes.KUBE_PING**。

OpenShift 4.x 模板配置为使用 **dns.DNS_PING** 机制，并将 **spec.ipFamilyPolicy** 字段设置为 **PreferDualStack**，以启用双栈配置的集群。但是，如果没有为 **JGROUPS_PING_PROTOCOL** 环境变量指定值，则 **kubernetes.KUBE_PING** 是镜像使用的默认选项。

3.4.1.1. 在 单堆栈 配置的集群中配置 DNS_PING

要使 **DNS_PING** 在 **IPv4 或 IPv6 单堆栈集群中** 工作，必须执行以下步骤：

1. **OPENSHIFT_DNS_PING_SERVICE_NAME** 环境变量必须设置为集群的 ping 服务名称。如果没有设置，服务器将充当单一节点集群（一个"集群"）。

2. **OPENSHIFT_DNS_PING_SERVICE_PORT** 环境变量应设置为公开 ping 服务的端口号。**DNS_PING** 协议会尝试从 SRV 记录中识别端口（如果它无法识别端口），则此变量将默认为 8888。

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

   a. 必须命名端口，才能使端口发现正常工作。

   b. 此服务的 **spec.publishNotReadyAddresses** 字段必须设置为 **"true** "。省略此布尔值的设置将导致每个节点在启动过程中组成其自身"集群"，然后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才会检测到）。

在 单堆栈 (IPv4 或 IPv6)集群上用于 DNS_PING 的 ping 服务的定义示例

```
kind: Service
apiVersion: v1
spec:
  clusterIP: None
  ipFamilyPolicy: SingleStack
  ports:
  - name: ping
    port: 8888
  publishNotReadyAddresses: true
  selector:
    deploymentConfig: sso
metadata:
  name: sso-ping
  annotations:
    description: "The JGroups ping port for clustering."
```

3.4.1.2. 在 双栈 配置集群中配置 DNS_PING

此外，为了使 **DNS_PING** 在支持 IPv4 和 IPv6 **地址系列的双网络集群中** 工作，集群 ping 服务的 **spec.ipFamilyPolicy** 字段必须设置为 **PreferDualStack** 或 **RequireDualStack**。此设置可确保 control plane 为配置了双栈的集群中的 ping 服务分配 IPv4 和 IPv6 集群 IP 地址，为 IPv4 和 IPv6 IP 地址启用反向 DNS 查找正常工作，并为 ping 无头服务创建对应的 DNS SRV 记录，如下所示：

使用与 PreferDualStack匹配 spec.ipFamilyPolicy 的 双栈 集群中的 ping 服务 DNS SRV 记录示例

```
$ host -t SRV "${OPENSHIFT_DNS_PING_SERVICE_NAME}"
sso-ping.dual-stack-demo.svc.cluster.local has SRV record 0 50 8888 10-128-0-239.sso-ping.dual-stack-demo.svc.cluster.local.
sso-ping.dual-stack-demo.svc.cluster.local has SRV record 0 50 8888 fd01-0-0-1--b8.sso-ping.dual-stack-demo.svc.cluster.local.
```

用于 双栈 (IPv4 和 IPv6)集群上的 DNS_PING 的 ping 服务的定义示例

```
kind: Service
apiVersion: v1
spec:
  clusterIP: None
```

```
    ipFamilyPolicy: PreferDualStack
    ports:
    - name: ping
      port: 8888
    publishNotReadyAddresses: true
    selector:
      deploymentConfig: sso
metadata:
    name: sso-ping
    annotations:
      description: "The JGroups ping port for clustering."
```

3.4.1.3. 配置 KUBE_PING

要使 **KUBE_PING** 正常工作，必须执行以下步骤：

1. 必须设置 **KUBERNETES_NAMESPACE** 环境变量。如果没有设置，服务器将充当单一节点集群（一个"集群"）。
2. 应设置 **KUBERNETES_LABELS** 环境变量。如果没有设置，则应用程序外的 pod （即使位于命名空间中）会尝试加入它们。
3. 必须向运行 Pod 的服务帐户授予授权，以允许访问 Kubernetes 的 REST api。您在命令行中授予授权。请参阅以下策略命令示例：

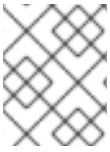
例 3.1. 策略命令

在 **myproject** 命名空间中使用 **default** 服务帐户：

```
oc policy add-role-to-user view system:serviceaccount:myproject:default -n myproject
```

在 **myproject** 命名空间中使用 **sso-service-account** ：

```
oc policy add-role-to-user view system:serviceaccount:myproject:sso-service-account -n myproject
```



注意

因为 **kubernetes.KUBE_PING** 发现机制不需要集群的额外 ping 服务，所以它使用单堆栈和双堆栈配置的集群上上述步骤进行操作。

如需 OpenShift 文档，请参阅 JBoss EAP 的专用部分：

- 探索可用的环境变量以加密 JGroups 流量
- 扩展 pod 的注意事项

3.5. 使用自定义 JDBC 驱动程序

要连接到任何数据库，该数据库的 JDBC 驱动程序必须存在，红帽单点登录。目前，镜像中唯一可用的 JDBC 驱动程序是 PostgreSQL JDBC 驱动程序。对于任何其他数据库，您需要使用自定义 JDBC 驱动程序和 CLI 脚本扩展红帽单点登录镜像，以注册并设置连接属性。以下步骤演示了如何进行此操作，并将

MariaDB 驱动程序作为一个示例进行。相应地更新其他数据库驱动程序的示例。

流程

1. 创建一个空目录。
2. 将 JDBC 驱动程序二进制文件下载到此目录。
3. 在此目录中，创建包含以下内容的新 **Dockerfile** 文件。对于其他数据库，将 **mariadb-java-client-2.5.4.jar** 替换为对应驱动程序的文件名：

```
FROM rh-sso-7/sso76-openshift-rhel8:latest

COPY sso-extensions.cli /opt/eap/extensions/
COPY mariadb-java-client-2.5.4.jar /opt/eap/extensions/jdbc-driver.jar
```

4. 在此目录中，创建包含以下内容的 **sso-extensions.cli** 文件：根据部署需求更新它中的值：

```
batch

set DB_DRIVER_NAME=mariadb
set DB_USERNAME=username
set DB_PASSWORD=password
set DB_DRIVER=org.mariadb.jdbc.Driver
set DB_XA_DRIVER=org.mariadb.jdbc.MariaDbDataSource
set DB_JDBC_URL=jdbc:mariadb://jdbc-host/keycloak
set DB_EAP_MODULE=org.mariadb

set FILE=/opt/eap/extensions/jdbc-driver.jar

module add --name=$DB_EAP_MODULE --resources=$FILE --
dependencies=javax.api,javax.resource.api
/subsystem=datasources/jdbc-driver=$DB_DRIVER_NAME:add( \
  driver-name=$DB_DRIVER_NAME, \
  driver-module-name=$DB_EAP_MODULE, \
  driver-class-name=$DB_DRIVER, \
  driver-xa-datasource-class-name=$DB_XA_DRIVER \
)
/subsystem=datasources/data-source=KeycloakDS:remove()
/subsystem=datasources/data-source=KeycloakDS:add( \
  jndi-name=java:jboss/datasources/KeycloakDS, \
  enabled=true, \
  use-java-context=true, \
  connection-url=$DB_JDBC_URL, \
  driver-name=$DB_DRIVER_NAME, \
  user-name=$DB_USERNAME, \
  password=$DB_PASSWORD \
)

run-batch
```

5. 在这个目录中，通过键入以下命令来构建您的镜像，将 **project/name:tag** 替换为任意名称。**可以使用** Docker 而不是 **podman**。

```
$ podman build -t docker-registry-default/project/name:tag .
```

6. 构建完成后，将镜像推送到 OpenShift 用于部署镜像的 registry。详情请参阅 OpenShift 指南。

7. 如果要将此镜像与您在上一步中构建的自定义 JDBC 驱动程序一起使用，且现有红帽单点登录 OpenShift DeploymentConfig 之前由一些 Red Hat Single Sign-On OpenShift 模板创建，您需要修补 DeploymentConfig 定义。输入以下命令：

```
$ oc patch dc/sso --type=json -p '[{"op": "replace", "path":
"/spec/triggers/0/imageChangeParams/from/name", "value": "sso76-openshift-rhel8-image-
with-custom-jdbc-driver:latest"}]'
"sso" patched
```

此命令假定红帽单点登录镜像与自定义 JDBC 驱动程序的镜像流名称和标签组合为"sso76-openshift-rhel8-image-with-custom-jdbc-driver:latest"。

3.6. 为红帽单点登录服务器创建管理员帐户

Red Hat Single Sign-On 不提供任何预先配置的管理帐户。此管理员帐户需要登录 **主** 域的管理控制台并执行服务器维护操作，如创建域或用户或注册应用程序，旨在由红帽单点登录保护。

可创建管理员帐户：

- 在部署 Red Hat Single Sign-On 应用程序模板时，可以为 ***SSO_ADMIN_USERNAME*** 和 ***SSO_ADMIN_PASSWORD*** 参数 提供值，或者
- 如果在没有应用程序模板的情况下，如果部署了适用于 OpenShift 镜像的 Red Hat Single Sign-On，则通过 远程 shell 会话连接到特定的 红帽单点登录 Pod。

注意

Red Hat Single Sign-On 允许 Welcome Page web 表单创建初始管理员帐户，但只有从本地主机访问 Welcome Page 时；此管理员创建方法不适用于 Red Hat Single Sign-On for OpenShift 镜像。

3.6.1. 使用模板参数创建管理员帐户

在部署 Red Hat Single Sign-On 应用模板时，***SSO_ADMIN_USERNAME*** 和 ***SSO_ADMIN_PASSWORD*** 参数表示为 **master** 域创建 Red Hat Single Sign-On 服务器的管理员帐户的用户名和密码。

这两个参数都是必需的。如果没有指定，则在模板实例化时，它们会自动生成并显示为 OpenShift 指令消息。

Red Hat Single Sign-On 服务器的管理员帐户的期限取决于用来存储 Red Hat Single Sign-On 服务器数据库的存储类型：

- 对于 in-memory 数据库模式(***sso76-ocp3-https***, ***sso76-ocp4-https***, ***sso76-ocp3-x509-https***, 和 ***sso76-ocp4-x509-https*** 模板，帐户在整个特定 Red Hat Single Sign-On pod 生命周期中（在 pod 销毁时会丢失）
- 对于临时数据库模式(***sso76-ocp3-postgresql*** 和 ***sso76-ocp4-postgresql*** 模板)，帐户在整个数据库 pod 生命周期中存在。即使红帽单点登录 pod 被破坏，存储的帐户数据会在假设数据库 pod 仍在运行的情况下保留。
- 对于持久性数据库模式（***sso76-ocp3-postgresql-persistent***, ***sso76-ocp4-postgresql-persistent***, ***sso76-ocp3-x509-postgresql-persistent***, 和 ***sso76-ocp4-x509-postgresql-persistent*** 模板，帐户在整个用于保存数据库数据的持久介质生命周期中存在。这意味着，即使

红帽单点登录和数据库 pod 都被销毁，存储的帐户数据也会保留。

部署 Red Hat Single Sign-On 应用程序模板是一种常见做法，以获取应用程序的对应 OpenShift 部署配置，然后多次重复使用部署配置（每当需要实例化新的红帽单点登录应用程序）。

如果是 **临时或持久数据库模式**，在创建 RH_SSO 服务器的管理员帐户后，在部署新红帽单点登录应用程序前从部署配置中删除 ***SSO_ADMIN_USERNAME*** 和 ***SSO_ADMIN_PASSWORD*** 变量。

流程

运行以下命令，以准备之前创建的红帽单点登录应用程序的部署配置，以便在创建管理员帐户后重复使用：

1. 识别红帽单点登录应用程序的部署配置。

```
$ oc get dc -o name
deploymentconfig/sso
deploymentconfig/sso-postgresql
```

2. 清除 ***SSO_ADMIN_USERNAME*** 和 ***SSO_ADMIN_PASSWORD*** 变量设置。

```
$ oc set env dc/sso \
  -e SSO_ADMIN_USERNAME="" \
  -e SSO_ADMIN_PASSWORD=""
```

3.6.2. 通过到 Red Hat Single Sign-On Pod 的远程 shell 会话创建管理员帐户

在不使用模板的情况下，从镜像流部署 Red Hat Single Sign-On 服务器时，使用以下命令为 Red Hat Single Sign-On 服务器的 **master** realm 创建管理员帐户。

前提条件

- Red Hat Single Sign-On 应用 pod 已启动。

流程

1. 识别 Red Hat Single Sign-On 应用容器集。

```
$ oc get pods
NAME                  READY     STATUS    RESTARTS   AGE
sso-12-pt93n            1/1       Running   0          1m
sso-postgresql-6-d97pf   1/1       Running   0          2m
```

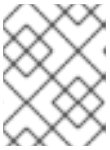
2. 打开到用于 OpenShift 容器的 Red Hat Single Sign-On 的远程 shell 会话。

```
$ oc rsh sso-12-pt93n
sh-4.2$
```

3. 在命令行中，通过 **add-user-keycloak.sh** 脚本为 **master** realm 创建 Red Hat Single Sign-On 服务器管理员帐户。

```
sh-4.2$ cd /opt/eap/bin/
sh-4.2$ ./add-user-keycloak.sh \
        -r master \
```

```
        -u sso_admin \
        -p sso_password
Added 'sso_admin' to '/opt/eap/standalone/configuration/keycloak-add-user.json', restart
server to load user
```



注意

上例中的 'sso_admin' / 'sso_password' 凭证仅用于演示目的。如需有关如何创建安全用户名和密码的指导，请参阅您的机构内的密码策略。

4. 重启底层 JBoss EAP 服务器实例，以加载新添加的用户帐户。等待服务器正确重启。

```
sh-4.2$ ./jboss-cli.sh --connect ':reload'
{
    "outcome" => "success",
    "result" => undefined
}
```

警告

在重新启动服务器时，只需在运行 Red Hat Single Sign-On 容器内重新启动 JBoss EAP 进程，而不是整个容器。这是因为重启整个容器将从头开始重新创建，无需 Red Hat Single Sign-On 服务器 **管理帐户**。

5. 使用上述步骤中创建的凭据，登录红帽单点登录服务器的 **主域** 管理控制台。在浏览器中，进入 **http://sso-<project-name>.<hostname>/auth/admin** for the Red Hat Single Sign-On web server, or to **https://secure-sso-<project-name>.<hostname>/auth/admin** for the encrypted Red Hat Single Sign-On web server，并指定用于创建管理员用户的用户名和密码。

其他资源

- 用于此软件的模板

3.7. 自定义 RED HAT SINGLE SIGN-ON 镜像的默认行为

您可以更改 Red Hat Single Sign-On 镜像的默认行为，如启用 TechPreview 功能或启用调试。本节论述了如何使用 JAVA_OPTS_APPEND 变量进行此更改。

先决条件

此流程假设之前已经 使用以下模板部署了 OpenShift 镜像的 Red Hat Single Sign-On：

- *sso76-ocp3-postgresql*
- *sso76-ocp4-postgresql*
- *sso76-ocp3-postgresql-persistent*
- *sso76-ocp4-postgresql-persistent*

- *sso76-ocp3-x509-postgresql-persistent*
- *sso76-ocp4-x509-postgresql-persistent*

流程

您可以使用 OpenShift Web 控制台或 CLI 更改默认行为。

如果使用 OpenShift Web 控制台，将 JAVA_OPTS_APPEND 变量添加到 sso 部署配置中。例如，要启用 TechPreview 功能，您需要设置 变量，如下所示：

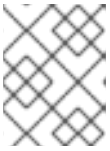
```
JAVA_OPTS_APPEND="-Dkeycloak.profile=preview"
```

如果使用 CLI，请使用以下命令，在使用前提条件下提到的模板部署 Red Hat Single Sign-On pod 时启用 TechPreview 功能。

1. 缩减 Red Hat Single Sign-On pod：

```
$ oc get dc -o name
deploymentconfig/sso
deploymentconfig/sso-postgresql

$ oc scale --replicas=0 dc sso
deploymentconfig "sso" scaled
```



注意

在上一命令中，**sso-postgresql** 会出现 sso-postgresql，因为 PostgreSQL 模板用于部署用于 OpenShift 镜像的 Red Hat Single Sign-On。

2. 编辑部署配置以设置 JAVA_OPTS_APPEND 变量。例如，要启用 TechPreview 功能，您需要设置 变量，如下所示：

```
$ oc env dc/sso -e "JAVA_OPTS_APPEND=-Dkeycloak.profile=preview"
```

3. 扩展 Red Hat Single Sign-On pod：

```
$ oc scale --replicas=1 dc sso
deploymentconfig "sso" scaled
```

4. 测试您选择的 TechPreview 功能。

3.8. 部署过程

部署后，*sso76-ocp3-https*、*sso76-ocp4-https* 模板和 *sso76-ocp3-x509-https* 或 *sso76-ocp4-x509-https* 模板会创建一个包含数据库和红帽单点登录服务器的单个 pod。*sso76-ocp3-postgresql*、*sso76-ocp4-postgresql*、*sso76-ocp3-postgresql-persistent*、*sso76-ocp4-postgresql-persistent* 以及 *sso76-ocp3-x509-postgresql-persistent* 或 *sso76-ocp4-x509-postgresql-persistent* 模板创建两个 pod，一个用于 Red Hat Single Sign-On Web 服务器。

在 Red Hat Single Sign-On web 服务器 pod 启动后，可以在其自定义配置的主机名或默认主机名中访问：

- **http://sso-*<project-name& gt;*. *<hostname>* /auth/admin**: 用于 Red Hat Single Sign-On web 服务器，以及
- **https://secure-sso-*<project-name& gt;*. *<hostname>* /auth/admin**: 用于加密的 Red Hat Single Sign-On web 服务器。

使用 管理员用户凭据 登录 **主** 域的管理控制台。

3.9. RED HAT SINGLE SIGN-ON 客户端

客户端是请求用户身份验证的红帽单点登录实体。客户端可以是请求红帽单点登录以提供用户身份验证的应用，也可以为访问令牌发出请求，以代表经过身份验证的用户启动服务。如需更多信息，请参阅 Red Hat Single Sign-On 文档中的管理客户端章节。

Red Hat Single Sign-On 提供 OpenID-Connect 和 SAML 客户端协议。

OpenID-Connect 是首选协议，它使用三种不同的访问类型：

- **公共** ：对于直接在浏览器中运行的 JavaScript 应用，不需要服务器配置。
- **机密** ：对于需要执行浏览器登录的服务器端客户端，如 EAP Web 应用。
- 仅 **bearer-only** ：对于允许 bearer 令牌请求的后端服务。

需要在应用程序 **web.xml** 文件的 &lt ;**auth-method** > 键中指定客户端类型。在部署时，该文件由镜像读取。将 < **auth-method>** 元素的值设置为：

- 用于 OpenID Connect 客户端的 **KEYCLOAK**。
- 用于 SAML 客户端的 **KEYCLOAK-SAML**。

以下是用来配置 OIDC 客户端的应用程序 **web.xml** 示例片断：

```
...
<login-config>
        <auth-method>KEYCLOAK</auth-method>
</login-config>
...
```

3.9.1. 自动和手动红帽单点登录客户端注册方法

客户端应用可使用特定于 *eap64-sso-s2i*、*eap71-sso-s2i* 和 *datavirt63-secure-s2i* 模板的变量中自动注册到红帽单点登录域。

或者，您可以通过配置和导出 Red Hat Single Sign-On 客户端适配器并包含在客户端应用程序配置中来手动注册客户端应用程序。

3.9.1.1. 自动 Red Hat Single Sign-On 客户端注册

自动红帽单点登录客户端注册由特定于 *eap64-sso-s2i*、*eap71-sso-s2i* 和 *datavirt63-secure-s2i* 模板的 Red Hat Single Sign-On 环境变量决定。然后，使用模板中提供的 Red Hat Single Sign-On 凭证，在部署客户端应用期间将客户端注册到 Red Hat Single Sign-On 域。

eap64-sso-s2i、*eap71-sso-s2i* 和 *datavirt63-secure-s2i* 中包含的 Red Hat Single Sign-On 环境变量包括：

变量	描述
HOSTNAME_HTTP	http 服务路由的自定义主机名。将空保留为默认主机名 <application-name>.<project>.<default-domain-suffix>
HOSTNAME_HTTPS	https 服务路由的自定义主机名。将空保留为默认主机名 <application-name>.<project>.<default-domain-suffix>
SSO_URL	Red Hat Single Sign-On web 服务器验证地址：https://secure-sso- <*project-name*> . <*hostname*>/auth
SSO_REALM	为此流程创建的红帽单点登录域。
SSO_USERNAME	*域管理用户* 的名称。
SSO_PASSWORD	用户的密码。
SSO_PUBLIC_KEY	域生成的公钥。它位于 Red Hat Single Sign-On 控制台的 **Realm Settings** 的 **Keys** 选项卡中。
SSO_BEARER_ONLY	如果设置为 **true**，则 OpenID Connect 客户端将注册为 bearer-only。
SSO_ENABLE_CORS	如果设置为 **true**，Red Hat Single Sign-On 适配器启用 Cross-Origin Resource Sharing(CORS)。

如果 Red Hat Single Sign-On 客户端使用 SAML 协议，则需要配置以下额外变量：

变量	描述
SSO_SAML_KEYSTORE_SECRET	用于访问 SAML 密钥存储的机密。默认值为 *sso-app-secret*。
SSO_SAML_KEYSTORE	SAML 密钥存储机密中的密钥存储文件名。默认为 *keystore.jks*。
SSO_SAML_KEYSTORE_PASSWORD	SAML 的密钥存储密码。默认为 *mykeystorepass*。
SSO_SAML_CERTIFICATE_NAME	SAML 所用密钥/证书的别名。默认为 *jboss*。

请参阅 示例：在带有 OpenID-Connect 客户端的红帽单点登录中自动注册 EAP 应用程序，以获取使用 OpenID -Connect 客户端自动客户端注册方法的端到端示例。

3.9.1.2. 手动注册红帽单点登录客户端注册

手动注册红帽单点登录客户端注册由客户端应用的 *../configuration/* 目录中的部署文件决定。这些文件从红帽单点登录 Web 控制台中的客户端适配器导出。此文件的名称是 OpenID-Connect 和 SAML 客户端的不同：

OpenID-Connect	*../configuration/secure-deployments*
SAML	*../configuration/secure-saml-deployments*

这些文件被复制到部署应用程序时，位于 的 *standalone-openshift.xml* 中的 Red Hat Single Sign-On adapter 配置部分。

可以通过两种方法将 Red Hat Single Sign-On 适配器配置传递给客户端应用程序：

- 修改部署文件以包含 Red Hat Single Sign-On 适配器配置，以便在部署时将其包含在 *standalone-openshift.xml* 文件中，或者
- 手动包含 OpenID-Connect *keycloak.json* 文件，或将 SAML *keycloak-saml.xml* 文件包含在客户端应用程序的 ... **/WEB-INF** 目录中。

请参阅 示例工作流：手动将应用配置为使用 Red Hat Single Sign-On 身份验证，使用 SAML 客户端作为手动红帽单点登录客户端注册方法的端到端示例。

3.10. 将 RED HAT SINGLE SIGN-ON VAULT 与 OPENSHIFT SECRET 搭配使用

Red Hat Single Sign-On 管理支持的几个字段从外部 vault 获取 secret 的值，请参阅 服务器管理指南。以下示例演示了如何在 OpenShift 中设置基于文件的纯文本库，并将其设置为用于获取 SMTP 密码。

流程

1. 使用 ***SSO_VAULT_DIR*** 环境变量为 vault 指定目录。您可以直接在部署配置的环境中直接引入 ***SSO_VAULT_DIR*** 环境变量。也可以通过在模板中的相应位置添加以下代码片段来包含在模板中：

```
"parameters": [
    ...
    {
        "displayName": "RH-SSO Vault Secret directory",
        "description": "Path to the RH-SSO Vault directory.",
        "name": "SSO_VAULT_DIR",
        "value": "",
        "required": false
    }
    ...
]

env: [
    ...
    {
        "name": "SSO_VAULT_DIR",
        "value": "${SSO_VAULT_DIR}"
```

```
    }
    ...
]
```

注意

只有在设置 ***SSO_VAULT_DIR*** 环境变量时，才会配置纯文本 vault 提供程序。

2. 在 OpenShift 集群中创建 secret：

```
$ oc create secret generic rhsso-vault-secrets --from-literal=master_smtp-password=mySMTPPsswd
```

3. 使用 **${SSO_VAULT_DIR}** 将卷作为路径挂载到部署配置中。对于已在运行的部署：

```
$ oc set volume dc/sso --add --mount-path=${SSO_VAULT_DIR} --secret-name=rhsso-vault-secrets
```

4. 创建 pod 后，您可以使用 Red Hat Single Sign-On 配置中的自定义字符串来引用该 secret。例如，对于在本教程中创建的 **mySMTPP** swd secret，您可以在 smtp 密码配置中的 **master** 域中使用 **${vault.smtp-password}**，并会被使用时由 **mySMTPPswd** 替换。

3.11. 限制

OpenShift 目前不接受来自外部提供程序的 OpenShift 角色映射。如果 Red Hat Single Sign-On 用作 OpenShift 的身份验证网关，则在 Red Hat Single Sign-On 中创建的用户必须使用 OpenShift Administrator **oc adm policy** 命令添加角色。

例如，允许 Red Hat Single Sign-On- created 用户在 OpenShift 中查看项目命名空间：

```
$ oc adm policy add-role-to-user view <user-name> -n <project-name>
```

第 4 章 教程

本章中的教程假定您通过执行 OpenShift Container Platform 集群安装，与创建的 OpenShift 实例类似。

4.1. 为 OPENSHIFT 镜像版本更新新红帽单点登录的数据库

请注意，以下与更新相关的点：

- 从以前的 Red Hat Single Sign-On 版本升级到 7.6.9 的滚动更新不被支持，因为数据库和缓存不兼容。
- 用于 Red Hat Single Sign-On for OpenShift 7.6.9 的模板需要 PostgreSQL 服务器版本 13。如果您有一个过时的 PostgreSQL 版本，请在更新数据库前更新 PostgreSQL 版本。
- 升级前，OpenShift 的 Red Hat Single Sign-On 版本中的实例无法运行。它们不能针对同一数据库同时运行。
- 生成的脚本不可用。它们根据数据库动态生成。

您有三种更新数据库的选择：

- 如果您有一个过时的 PostgreSQL 服务器版本，升级 PostgreSQL 服务器，然后迁移数据库。
- 允许 Red Hat Single Sign-On 7.6.9 自动迁移数据库架构
- 手动更新数据库

注意

默认情况下，在第一次启动 Red Hat Single Sign-On 7.6.9 时，数据库会自动迁移。

4.1.1. PostgreSQL 版本升级和数据库迁移

Red Hat Single Sign-On 7.6.9 模板中存在的 PostgreSQL 服务器可能与之前使用的版本不同。例如，请考虑这种情况：

- 您使用 PostgreSQL 服务器的版本 10，运行带有 PostgreSQL 容器集的 Red Hat Single Sign-On for OpenShift 容器镜像。
- Red Hat Single Sign-On 7.6.9 容器镜像要求 PostgreSQL pod 使用 PostgreSQL 服务器的版本 13。

以下流程描述了如何将 PostgreSQL 版本升级到 13，然后迁移数据库。

流程

1. 首先执行数据库级别备份。

```
$ oc rsh <POSTGRE-SQL-POD> pg_dump -C <DATABASE> rhsso_db.bak
```

2. 缩减 **sso** 容器集。

```
$ oc scale dc/sso --replicas=0
```

3. 编辑 **dc/sso-postgresql**.

```
$ oc edit dc/sso-postgresql
```

将 **ImageStreamTag** 切换到 **:postgresql:13-el8**。

```
- imageChangeParams:
    automatic: true
    containerNames:
    - sso-postgresql
    from:
      kind: ImageStreamTag
      name: postgresql:13-el8
      namespace: openshift
```

4. 等待 **sso-postgresql** pod 正在运行并稳定。

5. 确保 pod **sso-postgresql** 具有正确的版本。

```
$ oc rsh dc/sso-postgresql /bin/bash -c "psql --version"
psql (PostgreSQL) 13.5
```

6. 取消设置变量 **POSTGRESQL_UPGRADE**，并再次部署 **sso-postgresql** pod。

```
$ oc set env dc/sso-postgresql POSTGRESQL_UPGRADE-
```

7. 再次等待 **sso-posgresql** pod 正在运行并稳定。

8. 运行以下命令，为 **openshift** 项目中的 OpenShift 更新一组 Red Hat Single Sign-On 7.6.9 资源：
如果使用 OpenShift 3.x 集群，请使用以下命令：

```
$ for resource in sso76-image-stream.json \
  passthrough/ocp-3.x/sso76-ocp3-https.json \
  passthrough/ocp-3.x/sso76-ocp3-postgresql.json \
  passthrough/ocp-3.x/sso76-ocp3-postgresql-persistent.json \
  reencrypt/ocp-3.x/sso76-ocp3-x509-https.json \
  reencrypt/ocp-3.x/sso76-ocp3-x509-postgresql-persistent.json
do
  oc replace -n openshift --force -f \
  https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-image/sso76-dev/templates/${resource}
done
```

如果使用 OpenShift 4.x 集群，请使用下列命令：

```
$ for resource in sso76-image-stream.json \
  passthrough/ocp-4.x/sso76-ocp4-https.json \
  passthrough/ocp-4.x/sso76-ocp4-postgresql.json \
  passthrough/ocp-4.x/sso76-ocp4-postgresql-persistent.json \
  reencrypt/ocp-4.x/sso76-ocp4-x509-https.json \
  reencrypt/ocp-4.x/sso76-ocp4-x509-postgresql-persistent.json
do
  oc replace -n openshift --force -f \
```

```
  https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-image/sso76-dev/templates/${resource}
done
```

9. 运行以下命令，在 **openshift** 项目中安装 Red Hat Single Sign-On 7.6.9 OpenShift 镜像流：

```
$ oc -n openshift import-image rh-sso-7/sso76-openshift-rhel8:7.6 --from=registry.redhat.io/rh-sso-7/sso76-openshift-rhel8:7.6 --confirm
```

10. 更新现有部署配置中的镜像更改触发器，以引用 Red Hat Single Sign-On 7.6.9 镜像。

```
$ oc patch dc/sso --type=json -p '[{"op": "replace", "path": "/spec/triggers/0/imageChangeParams/from/name", "value": "sso76-openshift-rhel8:7.6"}]'
"sso" patched
```

11. 根据镜像更改触发器中定义的最新镜像，开始推出新的 Red Hat Single Sign-On 7.6.9 镜像。

```
$ oc rollout latest dc/sso
```

12. 将 **sso** pod 扩展至一个副本。

注意

您可能希望从 **dc/sso** 中暂时增加存活和就绪度探测阈值和值（以秒为单位）。此步骤在第一次引导时执行数据库升级，这可能需要一些时间。

```
$ oc scale --replicas=1 dc/sso
```

注意

如果您有多个副本，请考虑扩展至一个副本。在 Red Hat Single Sign-On 启动后，您可以将 缩减回原始副本数。

4.1.2. 自动数据库迁移

此过程假设您运行早期版本的 OpenShift 镜像，由运行在独立 pod 中的 PostgreSQL 数据库（以临时或永久模式部署）为 OpenShift 镜像提供支持。

先决条件

- 执行 为 OpenShift 部署准备红帽单点登录身份验证 中介绍的步骤。

流程

使用以下步骤自动迁移数据库模式：

1. 识别用于部署容器的部署配置，这些配置在 OpenShift 镜像的红帽单点登录之前版本上运行。

```
$ oc get dc -o name --selector=application=sso
deploymentconfig/sso
deploymentconfig/sso-postgresql
```

2. 在当前命名空间中停止运行之前版本的红帽单点登录的所有 Pod。它们不能针对同一数据库同时运行。

```
$ oc scale --replicas=0 dc/sso
deploymentconfig "sso" scaled
```

3. 更新现有部署配置中的镜像更改触发器，以引用 Red Hat Single Sign-On 7.6.9 镜像。

```
$ oc patch dc/sso --type=json -p '[{"op": "replace", "path":
"/spec/triggers/0/imageChangeParams/from/name", "value": "sso76-openshift-rhel8:7.6"}]'
"sso" patched
```

4. 根据镜像更改触发器中定义的最新镜像，开始推出新的 Red Hat Single Sign-On 7.6.9 镜像。

```
$ oc rollout latest dc/sso
deploymentconfig "sso" rolled out
```

5. 使用修改后的部署配置部署 Red Hat Single Sign-On 7.6.9 容器。

```
$ oc scale --replicas=1 dc/sso
deploymentconfig "sso" scaled
```

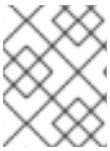
6. （可选）验证数据库是否已成功更新。

```
$ oc get pods --selector=application=sso
NAME                  READY     STATUS    RESTARTS   AGE
sso-4-vg21r           1/1       Running   0          1h
sso-postgresql-1-t871r   1/1       Running   0          2h
```

```
$ oc logs sso-4-vg21r | grep 'Updating'
11:23:45,160 INFO
[org.keycloak.connections.jpa.updater.liquibase.LiquibaseJpaUpdaterProvider]
(ServerService Thread Pool -- 58) Updating database. Using changelog META-INF/jpa-
changelog-master.xml
```

4.1.3. 手动数据库迁移

数据库迁移过程更新了数据架构，并执行数据的操作。此过程还会停止所有运行之前为 OpenShift 镜像的 Red Hat Single Sign-On 版本的所有 Pod，然后再动态生成 SQL 迁移文件。



注意

此过程假设您正在为由 PostgreSQL 数据库（以临时或永久模式部署）支持的 OpenShift 镜像运行早期版本，并在单独的 pod 上运行。

流程

为脚本生成准备环境。

1. 使用正确的数据源配置 Red Hat Single Sign-On 7.6.9，

2. 在 **standalone-openshift.xml** 文件中设置以下配置选项：

   a. **initializeEmpty=false**

a. **initializeEmpty=false**,

b. **migrationStrategy=manual**，以及

c. **migrationExport** 到 pod 的文件系统的位置，输出 SQL 迁移文件应存储在这里（例如 **migrationExport="${jboss.home.dir}/keycloak-database-update.sql"**）。

其他资源

- 数据库配置

流程

执行以下步骤为数据库生成 SQL 迁移文件：

1. 准备 OpenShift 数据库迁移作业模板，以生成 SQL 文件。

```
$ cat job-to-migrate-db-to-sso76.yaml.orig
apiVersion: batch/v1
kind: Job
metadata:
  name: job-to-migrate-db-to-sso76
spec:
  autoSelector: true
  parallelism: 0
  completions: 1
  template:
    metadata:
      name: job-to-migrate-db-to-sso76
    spec:
      containers:
      - env:
        - name: DB_SERVICE_PREFIX_MAPPING
          value: <<DB_SERVICE_PREFIX_MAPPING_VALUE>>
        - name: <<PREFIX>>_JNDI
          value: <<PREFIX_JNDI_VALUE>>
        - name: <<PREFIX>>_USERNAME
          value: <<PREFIX_USERNAME_VALUE>>
        - name: <<PREFIX>>_PASSWORD
          value: <<PREFIX_PASSWORD_VALUE>>
        - name: <<PREFIX>>_DATABASE
          value: <<PREFIX_DATABASE_VALUE>>
        - name: TX_DATABASE_PREFIX_MAPPING
          value: <<TX_DATABASE_PREFIX_MAPPING_VALUE>>
        - name: <<SERVICE_HOST>>
          value: <<SERVICE_HOST_VALUE>>
        - name: <<SERVICE_PORT>>
          value: <<SERVICE_PORT_VALUE>>
        image: <<SSO_IMAGE_VALUE>>
        imagePullPolicy: Always
        name: job-to-migrate-db-to-sso76
        # Keep the pod running after the SQL migration
        # file was generated, so we can retrieve it
        command:
          - "/bin/bash"
```

```
        - "-c"
        - "/opt/eap/bin/openshift-launch.sh || sleep 600"
      restartPolicy: Never
```

```
$ cp job-to-migrate-db-to-sso76.yaml.orig \
    job-to-migrate-db-to-sso76.yaml
```

2. 从用于运行 OpenShift 镜像的 Red Hat Single Sign-On 的早期版本的部署配置，将数据源定义和数据库访问凭证复制到适合数据库迁移作业模板的位置。
使用以下脚本，将 **DB_SERVICE_PREFIX_MAPPING** 和 **TX_DATABASE_PREFIX_MAPPING** 变量值，以及特定于特定数据源（**<PREFIX>_JNDI**, **<PREFIX>_USERNAME**, **<PREFIX>_PASSWORD**, 和 **<PREFIX>_DATABASE**）的环境变量从名为 **sso** 的部署配置复制到名为 **job-to-migrate-db-to-sso76.yaml** 的数据库作业迁移模板。

注意

虽然 **DB_SERVICE_PREFIX_MAPPING** 环境变量允许 用逗号分开 < **name>-<database_type>=<PREFIX** > triplets 列表，但本例脚本只接受一个用于演示目的的数据源 triplet 定义。您可以修改用于处理多个数据源定义脚本。

```
$ cat mirror_sso_dc_db_vars.sh
#!/bin/bash

# IMPORTANT:
#
# If the name of the SSO deployment config differs from 'sso'
# or if the file name of the YAML definition of the migration
# job is different, update the following two variables
SSO_DC_NAME="sso"
JOB_MIGRATION_YAML="job-to-migrate-db-to-sso76.yaml"

# Get existing variables of the $SSO_DC_NAME deployment config
# in an array
declare -a SSO_DC_VARS=( \
  $(oc set env dc/${SSO_DC_NAME} --list \
  | sed '/^#/d') \
)

# Get the PREFIX used in the names of environment variables
PREFIX=$( \
  grep -oP 'DB_SERVICE_PREFIX_MAPPING=[^ ]+' \
  <<< "${SSO_DC_VARS[@]}" \
)
PREFIX=${PREFIX##*=}

# Substitute:
# * <<PREFIX>> with actual $PREFIX value and
# * <<PREFIX with "<<$PREFIX" value
# The order in which these replacements are made is important!
sed -i "s#<<PREFIX>>#${PREFIX}#g" ${JOB_MIGRATION_YAML}
sed -i "s#<<PREFIX#<<${PREFIX}#g" ${JOB_MIGRATION_YAML}

# Construct the array of environment variables
# specific to the datasource
```

```
declare -a DB_VARS=(JNDI USERNAME PASSWORD DATABASE)

# Prepend $PREFIX to each item of the datasource array
DB_VARS=( "${DB_VARS[@]/#/${PREFIX}_}" )

# Add DB_SERVICE_PREFIX_MAPPING and TX_DATABASE_PREFIX_MAPPING
# variables to datasource array
DB_VARS=( \
  "${DB_VARS[@]}" \
  DB_SERVICE_PREFIX_MAPPING \
  TX_DATABASE_PREFIX_MAPPING \
)

# Construct the SERVICE from DB_SERVICE_PREFIX_MAPPING
SERVICE=$( \
  grep -oP 'DB_SERVICE_PREFIX_MAPPING=[^ ]' \
  <<< "${SSO_DC_VARS[@]}" \
)
SERVICE=${SERVICE#*=}
SERVICE=${SERVICE%=*}
SERVICE=${SERVICE^^}
SERVICE=${SERVICE//-/_}

# If the deployment config contains <<SERVICE>>_SERVICE_HOST
# and <<SERVICE>>_SERVICE_PORT variables, add them to the
# datasource array. Their values also need to be propagated into
# yaml definition of the migration job.
HOST_PATTERN="${SERVICE}_SERVICE_HOST=[^ ]"
PORT_PATTERN="${SERVICE}_SERVICE_PORT=[^ ]"
if
  grep -Pq "${HOST_PATTERN}" <<< "${SSO_DC_VARS[@]}" &&
  grep -Pq "${PORT_PATTERN}" <<< "${SSO_DC_VARS[@]}"
then
  DB_VARS=( \
    "${DB_VARS[@]}" \
    "${SERVICE}_SERVICE_HOST" \
    "${SERVICE}_SERVICE_PORT" \
  )
# If they are not defined, delete their placeholder rows in
# yaml definition file (since if not defined they are not
# expanded which make the yaml definition invalid).
else
  for KEY in "HOST" "PORT"
  do
    sed -i "/SERVICE_${KEY}/d" ${JOB_MIGRATION_YAML}
  done
fi

# Substitute:
# * <<SERVICE_HOST>> with ${SERVICE}_SERVICE_HOST and
# * <<SERVICE_HOST_VALUE>> with <<${SERVICE}_SERVICE_HOST_VALUE>>
# The order in which replacements are made is important!
# Do this for both "HOST" and "PORT"
for KEY in "HOST" "PORT"
do
  PATTERN_1="<<SERVICE_${KEY}>>"
```

```
  REPL_1="${SERVICE}_SERVICE_${KEY}"
  sed -i "s#${PATTERN_1}#${REPL_1}#g" ${JOB_MIGRATION_YAML}
  PATTERN_2="<<SERVICE_${KEY}_VALUE>>"
  REPL_2="<<${SERVICE}_SERVICE_${KEY}_VALUE>>"
  sed -i "s#${PATTERN_2}#${REPL_2}#g" ${JOB_MIGRATION_YAML}
done

# Propagate the values of the datasource array items into
# yaml definition of the migration job
for VAR in "${SSO_DC_VARS[@]}"
do
  IFS=$'=' read KEY VALUE <<< $VAR
  if grep -q $KEY <<< ${DB_VARS[@]}
  then
    KEY+="_VALUE"
    # Enwrap integer port value with double quotes
    if [[ ${KEY} =~ ${SERVICE}_SERVICE_PORT_VALUE ]]
    then
      sed -i "s#<<${KEY}>>#\"${VALUE}\"#g" ${JOB_MIGRATION_YAML}
    # Character values do not need quotes
    else
      sed -i "s#<<${KEY}>>#${VALUE}#g" ${JOB_MIGRATION_YAML}
    fi
    # Verify that the value has been successfully propagated.
    if
      grep -q '(JNDI|USERNAME|PASSWORD|DATABASE)' <<< "${KEY}" &&
      grep -q "<<PREFIX${KEY#${PREFIX}}" ${JOB_MIGRATION_YAML} ||
      grep -q "<<${KEY}>>" ${JOB_MIGRATION_YAML}
    then
      echo "Failed to update value of ${KEY%_VALUE}! Aborting."
      exit 1
    else
      printf '%-60s%-40s\n' \
            "Successfully updated ${KEY%_VALUE} to:" \
            "$VALUE"
    fi
  fi
done
```

运行脚本。

```
$ chmod +x ./mirror_sso_dc_db_vars.sh
$ ./mirror_sso_dc_db_vars.sh
Successfully updated DB_SERVICE_PREFIX_MAPPING to:          sso-postgresql=DB
Successfully updated DB_JNDI to:                            java:jboss/datasources/KeycloakDS
Successfully updated DB_USERNAME to:                        userxOp
Successfully updated DB_PASSWORD to:                        tsWNhQHK
Successfully updated DB_DATABASE to:                        root
Successfully updated TX_DATABASE_PREFIX_MAPPING to:         sso-postgresql=DB
```

3. 使用 预配置的源构建 Red Hat Single Sign-On 7.6.9 数据库迁移镜像，并等待构建完成。

```
$ oc get is -n openshift | grep sso76 | cut -d ' ' -f1
sso76-openshift-rhel8
```

```
$ oc new-build sso76-openshift-rhel8:7.6~https://github.com/iankko/openshift-
examples.git#KEYCLOAK-8500 \
  --context-dir=sso-manual-db-migration \
  --name=sso76-db-migration-image
--> Found image bf45ac2 (7 days old) in image stream "openshift/sso76-openshift-rhel8"
under tag "7.6" for "sso76-openshift-rhel8:7.6"

    Red Hat SSO 7.6.9
    ----------
    Platform for running Red Hat SSO

    Tags: sso, sso7, keycloak

    * A source build using source code from https://github.com/iankko/openshift-
examples.git#KEYCLOAK-8500 will be created
      * The resulting image will be pushed to image stream "sso76-db-migration-image:latest"
      * Use 'start-build' to trigger a new build

--> Creating resources with label build=sso76-db-migration-image ...
    imagestream "sso76-db-migration-image" created
    buildconfig "sso76-db-migration-image" created
--> Success
    Build configuration "sso76-db-migration-image" created and build triggered.
    Run 'oc logs -f bc/sso76-db-migration-image' to stream the build progress.
```

```
$ oc logs -f bc/sso76-db-migration-image --follow
Cloning "https://github.com/iankko/openshift-examples.git#KEYCLOAK-8500" ...
...
Push successful
```

4. 更新数据库迁移作业(**job-to-migrate-db-to-sso76.yaml**)的模板并引用构建的 **sso76-db-migration-image** 镜像。

   a. 获取镜像的 docker pull 参考。

   ```
   $ PULL_REF=$(oc get istag -n $(oc project -q) --no-headers | grep sso76-db-migration-
   image | tr -s ' ' | cut -d ' ' -f 2)
   ```

   b. 将作业模板中的 <<SSO_IMAGE_VALUE> 字段替换为 pull 规格。

   ```
   $ sed -i "s#<<SSO_IMAGE_VALUE>>#$PULL_REF#g" job-to-migrate-db-to-sso76.yaml
   ```

   c. 验证该字段是否已更新。

5. 从作业模板实例化数据库迁移作业。

```
$ oc create -f job-to-migrate-db-to-sso76.yaml
job "job-to-migrate-db-to-sso76" created
```

重要

数据库迁移过程处理数据架构更新并执行数据操作，因此停止为 OpenShift 镜像运行之前版本的 Red Hat Single Sign-On 的所有 pod，然后再动态生成 SQL 迁移文件。

6. 识别用于部署容器的部署配置，这些配置在 OpenShift 镜像的红帽单点登录之前版本上运行。

```
$ oc get dc -o name --selector=application=sso
deploymentconfig/sso
deploymentconfig/sso-postgresql
```

7. 在当前命名空间中停止运行之前版本的红帽单点登录的所有 Pod。

```
$ oc scale --replicas=0 dc/sso
deploymentconfig "sso" scaled
```

8. 运行数据库迁移作业并等待 pod 正确运行。

```
$ oc get jobs
NAME                        DESIRED   SUCCESSFUL   AGE
job-to-migrate-db-to-sso76   1         0           3m
```

```
$ oc scale --replicas=1 job/job-to-migrate-db-to-sso76
job "job-to-migrate-db-to-sso76" scaled
```

```
$ oc get pods
NAME                              READY     STATUS      RESTARTS   AGE
sso-postgresql-1-n5p16            1/1       Running     1          19h
job-to-migrate-db-to-sso76-b87bb  1/1       Running     0          1m
sso76-db-migration-image-1-build  0/1       Completed   0          27m
```

注意

默认情况下，在生成迁移文件后数据库迁移作业会在 **600 秒后自动**终止。您可以调整这个时间。

9. 从 pod 获取动态生成的 SQL 数据库迁移文件。

```
$ mkdir -p ./db-update
$ oc rsync job-to-migrate-db-to-sso76-b87bb:/opt/eap/keycloak-database-update.sql ./db-update
receiving incremental file list
keycloak-database-update.sql

sent 30 bytes  received 29,726 bytes  59,512.00 bytes/sec
total size is 29,621  speedup is 1.00
```

10. 检查 **keycloak-database-update.sql** 文件，以了解有关在手动数据库更新中对 Red Hat Single Sign-On 7.6.9 版本所做的更改。

11. 手动应用数据库更新。

- 运行以下命令，如果为 OpenShift 镜像运行一些早期版本的 Red Hat Single Sign-On，则由以临时或持久模式部署的 PostgreSQL 数据库提供支持，并在单独的 pod 上运行：

  i. 将生成的 SQL 迁移文件复制到 PostgreSQL pod。

```
$ oc rsync --no-perms=true ./db-update/ sso-postgresql-1-n5p16:/tmp
```

```
sending incremental file list

sent 77 bytes  received 11 bytes  176.00 bytes/sec
total size is 26,333  speedup is 299.24
```

ii. 启动与 PostgreSQL pod 的 shell 会话。

```
$ oc rsh sso-postgresql-1-n5p16
sh-4.2$
```

iii. 使用 **psql** 工具手动应用数据库更新。

```
sh-4.2$ alias psql="/opt/rh/rh-postgresql95/root/bin/psql"
sh-4.2$ psql --version
psql (PostgreSQL) 9.5.4
sh-4.2$ psql -U <PREFIX>_USERNAME -d <PREFIX>_DATABASE -W -f
/tmp/keycloak-database-update.sql
Password for user <PREFIX>_USERNAME:
INSERT 0 1
INSERT 0 1
...
```

重要

将 **<PREFIX>_USERNAME** 和 **<PREFIX>_DATABASE** 替换为 上一节中检索的实际数据库凭证。在提示时，使用 **<PREFIX>_PASSWORD** 作为数据库的密码。

iv. 关闭与 PostgreSQL pod 的 shell 会话。继续 更新镜像更改触发器步骤。

12. 更新现有部署配置中的镜像更改触发器，以引用 Red Hat Single Sign-On 7.6.9 镜像。

```
$ oc patch dc/sso --type=json -p '[{"op": "replace", "path":
"/spec/triggers/0/imageChangeParams/from/name", "value": "sso76-openshift-rhel8:7.6"}]'
"sso" patched
```

13. 根据镜像更改触发器中定义的最新镜像，开始推出新的 Red Hat Single Sign-On 7.6.9 镜像。

```
$ oc rollout latest dc/sso
deploymentconfig "sso" rolled out
```

14. 使用修改后的部署配置部署 Red Hat Single Sign-On 7.6.9 容器。

```
$ oc scale --replicas=1 dc/sso
deploymentconfig "sso" scaled
```

4.2. 在环境间迁移 RED HAT SINGLE SIGN-ON 服务器的数据库

本教程着重介绍将 Red Hat Single Sign-On 服务器数据库从一个环境迁移到另一个环境或迁移到不同的数据库。

导出和导入 Red Hat Single Sign-On 7.6.9 数据库会在 Red Hat Single Sign-On 服务器引导时触发，其参数通过 Java 系统属性传递。这意味着在一个红帽单点登录服务器引导期间，仅执行其中一个可能的迁移操作、*导出* 或 *导入*。

4.2.1. 部署 Red Hat Single Sign-On PostgreSQL 应用程序模板

先决条件

- 已执行 为 OpenShift Deployment 准备红帽单点登录身份验证 部分中描述的步骤。

流程

1. 登录 OpenShift Web 控制台，再选择 **sso-app-demo** 项目空间。
2. 单击 **Add to project**，以列出默认镜像流和模板。
3. 使用 **Filter by keyword** 搜索栏，将列表限制为与 *sso* 匹配的列表。您可能需要点击 **See all** 以显示所需的应用程序模板。
4. 选择 ***sso76-ocp4-postgresql*** Red Hat Single Sign-On 应用程序模板。在部署模板时，请确保 **保留 *SSO_REALM* 变量未设置**（默认值）。

警告

当在 Red Hat Single Sign-On 上为 OpenShift 镜像设置了 ***SSO_REALM*** 配置变量时，将执行数据库导入，以创建 变量所请求的红帽单点登录服务器域。要使数据库导出正确执行，无法在此类镜像上同时定义 ***SSO_REALM*** 配置变量。

5. 点击 **Create** 来部署应用程序模板并启动 Pod 部署。这可能需要几分钟时间。
然后，使用 管理员帐户 访问位于 **https://secure-sso- *<sso-app-demo>* . *<openshift32.example.com>* /auth/admin** 的 Red Hat Single Sign-On Web 控制台。

注意

这个示例工作流使用自生成的 CA 为演示提供一个端到端工作流。访问 Red Hat Single Sign-On Web 控制台将出现不安全的连接警告。
对于生产环境，红帽建议您使用从验证的证书颁发机构购买的 SSL 证书。

其他资源

- 导入和导出数据库

4.2.2. （可选）创建要导出的额外域和用户

在执行 Red Hat Single Sign-On 7.6.9 服务器数据库导出时，只会导出当前数据库中的域和用户。如果导出的 JSON 文件还应包含其他红帽单点登录域和用户，则需要创建它们。使用这些程序。

1. 创建一个 realm

2. 创建用户

在创建之后，可以导出数据库。

其他资源

- 导入和导出数据库

4.2.3. 将 Red Hat Single Sign-On 数据库导出为 OpenShift pod 上的 JSON 文件

先决条件

- 创建新的域和用户。

流程

1. 获取红帽单点登录部署配置并将其缩减为零。

```
$ oc get dc -o name
deploymentconfig/sso
deploymentconfig/sso-postgresql

$ oc scale --replicas=0 dc sso
deploymentconfig "sso" scaled
```

2. 指示在 Red Hat Single Sign-On 7.6.9 服务器上部署的 Red Hat Single Sign-On 7.6.9 服务器，以便在 Red Hat Single Sign-On 服务器引导时执行数据库导出。

```
$ oc set env dc/sso \
  -e "JAVA_OPTS_APPEND= \
    -Dkeycloak.migration.action=export \
    -Dkeycloak.migration.provider=singleFile \
    -Dkeycloak.migration.file=/tmp/demorealm-export.json"
```

3. 备份红帽单点登录部署配置。这将启动 Red Hat Single Sign-On 服务器并导出其数据库。

```
$ oc scale --replicas=1 dc sso
deploymentconfig "sso" scaled
```

4. （可选）验证导出是否成功。

```
$ oc get pods
NAME                  READY     STATUS    RESTARTS   AGE
sso-4-ejr0k           1/1       Running   0          27m
sso-postgresql-1-ozzl0   1/1       Running   0          4h

$ oc logs sso-4-ejr0k | grep 'Export'
09:24:59,503 INFO  [org.keycloak.exportimport.singlefile.SingleFileExportProvider]
(ServerService Thread Pool -- 57) Exporting model into file /tmp/demorealm-export.json
09:24:59,998 INFO  [org.keycloak.services] (ServerService Thread Pool -- 57) KC-
SERVICES0035: Export finished successfully
```

4.2.4. 检索并导入导出的 JSON 文件

流程

1. 从 pod 检索 Red Hat Single Sign-On 数据库的 JSON 文件。

```
$ oc get pods
NAME                 READY    STATUS    RESTARTS   AGE
sso-4-ejr0k          1/1      Running   0          2m
sso-postgresql-1-ozzl0   1/1      Running   0          4h

$ oc rsync sso-4-ejr0k:/tmp/demorealm-export.json .
```

2. （可选）将 Red Hat Single Sign-On 数据库的 JSON 文件导入到另一个环境中运行的红帽单点登录服务器。

注意

有关导入到 OpenShift 上未运行的 Red Hat Single Sign-On 服务器，请参阅 导入和导出数据库。

当 Red Hat Single Sign-On 服务器作为 Red Hat Single Sign-On 7.6.9 容器运行时，使用 Admin Console Export/Import 功能将资源从之前导出的 JSON 文件导入到 Red Hat Single Sign-On 服务器数据库中。

a. 使用用于创建管理员用户的凭据，登录红帽单点登录服务器的 **主域** 管理控制台。在浏览器中，导航到 Red Hat Single Sign **-On web 服务器的 http://sso-<project-name>.<hostname>/auth/admin**，或指向加密的红帽 Single Sign **-On web 服务器的 https://secure-sso-<project-name>.<hostname>/auth/admin**。

b. 在侧边栏的顶部，选择 Red Hat Single Sign-On 域的名称、用户、客户端、域角色和客户端角色。本例使用 **master** realm。

c. 单击侧栏底部的 **Manage** 部分下的 **Import** 链接。

d. 在打开的页面中，单击 **Select file**，然后指定本地文件系统中导出的 **demorealm-export.json** JSON 文件的位置。

e. 在 **Import from realm** 下拉菜单中选择应从其中导入数据的 Red Hat Single Sign-On 域的名称。本例使用 **master** realm。

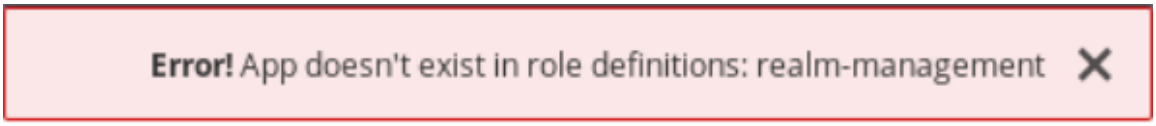
f. 选择应导入哪些用户、客户端、域角色和客户端角色（所有这些角色都默认导入）。

g. 选择要执行的策略，即资源已存在（ **Fail**、**Skip** 或 **Overwrite**之一）。

注意

如果当前数据库中已存在具有相同标识符的对象，尝试导入对象（用户、客户端、域角色或客户端角色）。使用 **Skip** 策略导入 **demorealm-export.json** 文件中存在的对象，但不存在于当前数据库中。

h. 单击 **Import** 以执行导入。
当将对象从非 master 域导入到 **master** 域时，点 **Import** 按钮后，有时可能会出现类似以下的错误：



在这种情况下，首先需要创建缺少的客户端，将 **Access Type** 设为 **bearer-only**。这些客户端可通过从源红帽单点登录服务器（创建导出 JSON 文件）到目标红帽单点登录服务器（其中导入 JSON 文件）的特征来创建。创建必要的客户端后，再次单击 **导入** 按钮。

要绕过 上述 错误消息，需要创建缺少的 **realm-management** 客户端（使用 **bearer-only Access Type** ），然后再次单击 **Import** 按钮。

对于 **Skip** import 策略，新添加的对象被标记为 **ADDED** 和跳过的对象，在导入结果页面的 **Action** 列中被标记为 **SKIPPED**。

如果选择（ **覆盖** 策略），Admin Console 导入允许您**覆盖** 资源。在生产环境中，请谨慎使用此功能。

4.3. 配置 OPENSHIFT 3.11 以使用 RED HAT SINGLE SIGN-ON 进行身份验证

配置 OpenShift 3.11，以使用红帽单点登录部署作为 OpenShift 的授权网关。

本例添加了 Red Hat Single Sign-On 作为验证方法，以及安装 OpenShift Container Platform 集群期间配置的身份提供程序。配置后，红帽单点登录方法也将可用（与配置的身份提供程序）用于用户登录 OpenShift Web 控制台。

其他资源

- 用户身份供应商
- 安装 OpenShift Container Platform 集群

4.3.1. 配置红帽单点登录凭证

先决条件

- 已执行 为 OpenShift Deployment 准备红帽单点登录身份验证 部分中描述的步骤。

流程

在 **https://secure-sso-*sso-app-demo* 登录加密的红帽单点登录 Web 服务器。*openshift32.example.com*/auth/admin** 使用 Red Hat Single Sign-On 部署期间创建的 xref:sso-administrator-setup[administrator 帐户。

创建一个 Realm

1. 将光标悬停在域命名空间（默认为 **主**栏的顶部），然后单击 **Add Realm**。
2. 输入 realm 名称（本示例使用 *OpenShift*）并点 **Create**。

创建用户

创建一个测试用户，用于演示启用了 Red Hat Single Sign-On 的 OpenShift 登录：

1. 单击 **Manage** sidebar 中的 **Users**，以查看域的用户信息。

2. 单击 **添加用户**。
3. 输入一个有效的 **Username** （本示例使用 *testuser*）和任何其他可选信息，**然后单击保存**。
4. 编辑用户配置：
   a. 单击用户空间中的 **Credentials** 选项卡，然后输入用户的密码。
   b. 确保将 **Temporary Password** 选项设为 **Off**，以便它不会稍后提示输入密码，然后单击 **Reset Password** 来设置用户密码。弹出窗口提示进行额外的确认。

创建并配置 OpenID-Connect Client

1. 单击 **Manage** 栏中的 **Clients**，再单击 **Create**。
2. 输入 **客户端 ID**。这个示例使用 *openshift-demo*。
3. 从下拉菜单中选择 **Client Protocol** （本例使用 **openid-connect**）并点 **Save**。您将进入 *openshift-demo* 客户端的配置设置页面。
4. 从 **Access Type** 下拉菜单中选择 **confidential**。这是服务器端应用程序的访问类型。
5. 在 **Valid Redirect URIs** 对话框中，输入 OpenShift Web 控制台的 URI，本例中为 *https://openshift.example.com:8443/**。

在下一节中，需要客户端 **Secret** 在 OpenShift 主控机上配置 OpenID-Connect。现在，您可以在 **Credentials** 标签页下复制它。secret 是 <*7b0384a2-b832-16c5-9d73-2957842e89h7*>。

其他资源

- 管理 OpenID Connect 和 SAML 客户端

4.3.2. 为红帽单点登录身份验证配置 OpenShift Master

登录 OpenShift 主控机 CLI。

先决条件

您必须具有编辑 **/etc/origin/master/master-config.yaml** 文件的权限。

流程

1. 编辑 **/etc/origin/master/master-config.yaml** 文件并找到 **identityProviders** 部分。例如，如果 OpenShift 主控机配置了 HTPassword 身份提供程序，则 **identityProviders** 部分将类似如下：

```
identityProviders:
- challenge: true
  login: true
  name: htpasswd_auth
  provider:
    apiVersion: v1
    file: /etc/origin/openshift-passwd
    kind: HTPasswdPasswordIdentityProvider
```

将 Red Hat Single Sign-On 作为二级身份提供程序添加类似以下代码片段的内容：

```
- name: rh_sso
  challenge: false
  login: true
  mappingMethod: add
  provider:
    apiVersion: v1
    kind: OpenIDIdentityProvider
    clientID: openshift-demo
    clientSecret: 7b0384a2-b832-16c5-9d73-2957842e89h7
    ca: xpaas.crt
    urls:
      authorize: https://secure-sso-sso-app-
demo.openshift32.example.com/auth/realms/OpenShift/protocol/openid-connect/auth
      token: https://secure-sso-sso-app-
demo.openshift32.example.com/auth/realms/OpenShift/protocol/openid-connect/token
      userInfo: https://secure-sso-sso-app-
demo.openshift32.example.com/auth/realms/OpenShift/protocol/openid-connect/userinfo
    claims:
      id:
      - sub
      preferredUsername:
      - preferred_username
      name:
      - name
      email:
      - email
```

a. **clientSecret** 的 Red Hat Single Sign-On **Secret** hash 可在 Red Hat Single Sign-On Web 控制台中找到： **Clients → *openshift-demo* → Credentials**

b. 可以通过使用红帽单点登录应用程序发出请求来查找 **urls** 的端点。例如：

```
<curl -k https://secure-sso-sso-app-
demo.openshift32.example.com/auth/realms/OpenShift/.well-known/openid-configuration
| python -m json.tool>
```

响应包括 **authorization_endpoint**、**token_endpoint** 和 **userinfo_endpoint**。

c. 这个示例工作流使用自生成的 CA 为演示提供一个端到端工作流。因此，**ca** 作为 <*ca: xpaas.crt*> 提供。此 CA 证书还必须复制到 **/etc/origin/master** 文件夹中。如果使用从验证的证书颁发机构购买的证书，则不需要这样做。

2. 保存配置并重启 OpenShift master：

```
$ systemctl restart atomic-openshift-master
```

4.3.3. 登录 OpenShift

流程

1. 导航到 OpenShift Web 控制台，在这个示例中为 *https://openshift.example.com:8443/console*。
OpenShift 登录页面现在提供使用 **htpasswd_auth** 或 **rh-sso** 身份提供程序登录的选项？前者仍然可用，因为它存在于 **/etc/origin/master/master-config.yaml** 中。

2. 选择 **rh-sso**，再使用之前在红帽单点登录中创建的 *testuser* 用户登录 OpenShift。
在 OpenShift CLI 中添加项目之前，项目对 *testuser* 可见。这是在 OpenShift 中提供用户特权的唯一方法，因为它目前不接受外部角色映射。

3. 要为 **sso-app-demo** 提供 *testuser* 视图 特权，请使用 OpenShift CLI：

```
$ oc adm policy add-role-to-user view testuser -n sso-app-demo
```

4.4. 从 MAVEN 二进制文件创建 OPENSHIFT 应用程序，并使用 RED HAT SINGLE SIGN-ON 对其进行保护

要在 OpenShift 上部署现有应用，您可以使用二进制源功能。

4.4.1. 部署 EAP 6.4 / 7.1 JSP 服务二进制构建并使用 Red Hat Single Sign-On vocation 应用和保护它

以下示例使用 app-jee-jsp 和 service-jee-jaxrs 快速入门部署 EAP 6.4 / 7.1 JSP 服务应用程序，使用 Red Hat Single Sign-On 进行身份验证。

先决条件

- 在此之前，已使用以下模板之一部署了 Red Hat Single Sign-On for OpenShift 镜像：
- *sso76-ocp3-postgresql*
- *sso76-ocp3-postgresql-persistent*
- *sso76-ocp3-x509-postgresql-persistent*
- *sso76-ocp4-postgresql*
- *sso76-ocp4-postgresql-persistent*
- *sso76-ocp4-x509-postgresql-persistent*

4.4.1.1. 为 EAP 6.4 / 7.1 JSP 应用程序创建红帽单点登录、角色和用户

EAP 6.4 / 7.1 JSP 服务应用需要专门的红帽单点登录域、用户名和密码才能使用红帽单点登录进行身份验证。在为 OpenShift 镜像部署了 Red Hat Single Sign-On 后执行以下步骤：

创建 Red Hat Single Sign-On Realm

1. 登录红帽单点登录服务器的管理控制台。
https://secure-sso-sso-app-demo.openshift.example.com/auth/admin

   使用 Red Hat Single Sign-On 管理员用户 的凭据 。

2. 将光标悬停在域命名空间（默认为 **主**栏的顶部），然后单击 **Add Realm**。

3. 输入 realm 名称（本示例使用 **demo**）并点 **Create**。

复制公钥

在新创建的 **演示域中**，单击 **Keys** 选项卡，然后选择 **Active** 选项卡，然后复制生成的 **RSA** 的公钥。

注意

Red Hat Single Sign-On for OpenShift 镜像版本 7.6.9 默认生成多个密钥，如 **HS256**、**RS256** 或 **AES**。要复制 Red Hat Single Sign-On for OpenShift 7.6.9 镜像的公钥信息，请点 **Keys** 选项卡，然后选择 **Active** 选项卡，然后点击 **密钥** 表中的该行的公钥按钮，其中键的类型与 **RSA** 匹配。然后选择并复制出现的弹出窗口的内容。

之后，需要部署 启用了 Red Hat Single Sign-On-enabled EAP 6.4 / 7.1 JSP 应用程序的信息。

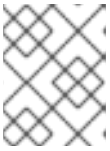
创建红帽单点登录角色

service-jee-jaxrs Quickstart 通过服务公开三个端点：

- **public** - 不需要身份验证。
- **secured** - 可以被具有 **user** 角色的用户调用。
- **admin** - 可以由具有 **admin** 角色的用户调用。

在 红帽单点登录中创建用户和 **admin** 角色。这些角色将分配给 Red Hat Single Sign-On 应用用户，以验证对用户应用的访问权限。

1. 单击 **Configure** 边栏中的 **Roles**，以列出此域的角色。



注意

这是一个新域，因此只有默认值（**offline_access** 和 **uma_authorization**）角色。

2. 点 **Add Role**。
3. 输入角色名称（**用户**），**然后单击保存**。

对 **admin** 角色重复这些步骤。

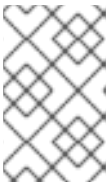
创建 Red Hat Single Sign-On Realm 管理用户

1. 单击 **Manage** sidebar 中的 **Users**，以查看域的用户信息。
2. 点 **Add User**。
3. 输入一个有效的 **Username** （本例使用用户 **appuser**）并点 **Save**。
4. 编辑用户配置：
   a. 单击用户空间中的 **Credentials** 选项卡，然后为用户输入一个密码（本例使用密码 **apppassword**）。
   b. 确保将 **Temporary Password** 选项设为 **Off**，以便它不会稍后提示输入密码，然后单击 **Reset Password** 来设置用户密码。弹出窗口将提示您确认。

4.4.1.2. 将 user 角色分配给 realm 管理用户

执行以下步骤将之前创建的 **appuser 用户与** Red Hat Single Sign-On 角色关联：

1. 单击 **Role Mappings**，以列出 realm 和 client 角色配置。在 **Available Roles** 中，选择之前创建的 **用户角色**，然后点击 **Add selected>**。
2. 点 **Client Roles**，从列表中选择 **realm-management** 条目，在 **Available Roles** 列表中选择每个记录。



注意

您可以通过保留 **Ctrl** 键并同时点击第一个 **模拟** 条目来选择多个项目。虽然保留 **Ctrl** 键和鼠标左键，但按鼠标右键移动至 **view-clients** 条目的结尾，并确保每个记录已被选中。

3. 点 **Add selected** > 将角色分配给客户端。

4.4.1.3. 为 EAP 6.4 / 7.1 JSP 应用进行 OpenShift 部署准备红帽单点登录身份验证

流程

1. 为 EAP 6.4 / 7.1 JSP 应用创建一个新项目。

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

2. 将 **view** 角色添加到 **default** 服务帐户。这可让服务帐户查看 **eap-app-demo** 命名空间中的所有资源，这是管理集群所必需的。

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

3. EAP 模板需要 SSL 密钥存储和 JGroups 密钥存储。本例使用 **keytool**（Java Development Kit 中包含的软件包）为这些密钥存储生成自签名证书。
   a. 为 SSL 密钥存储生成安全密钥（本例使用 **password** 作为密钥存储的密码）。

```
$ keytool -genkeypair \
-dname "CN=secure-eap-app-eap-app-demo.openshift.example.com" \
-alias https \
-storetype JKS \
-keystore eapkeystore.jks
```

   b. 为 JGroups 密钥存储生成安全密钥（本例使用 **password** 作为密钥存储的密码）。

```
$ keytool -genseckey \
-alias jgroups \
-storetype JCEKS \
-keystore eapjgroups.jceks
```

   c. 使用 SSL 和 JGroup 密钥存储文件为 OpenShift 机密生成 EAP 6.4 / 7.1。

```
$ oc create secret generic eap-ssl-secret --from-file=eapkeystore.jks
```

```
$ oc create secret generic eap-jgroup-secret --from-file=eapjgroups.jceks
```

   d. 将 EAP 应用机密添加到 **default** 服务帐户。

```
$ oc secrets link default eap-ssl-secret eap-jgroup-secret
```

4.4.1.4. 部署 EAP 6.4 / 7.1 JSP 应用程序的二进制构建

流程

1. 克隆源代码。

```
$ git clone https://github.com/keycloak/keycloak-quickstarts.git
```

配置 Red Hat JBoss Middleware Maven 软件仓库

2. 同时构建 service-jee-jaxrs 和 app-jee-jsp 应用。

a. 构建 **service-jee-jaxrs** 应用。

```
$ cd keycloak-quickstarts/service-jee-jaxrs/
```

```
$ mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Keycloak Quickstart: service-jee-jaxrs 3.1.0.Final
[INFO] ------------------------------------------------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.153 s
[INFO] Finished at: 2017-06-26T12:06:12+02:00
[INFO] Final Memory: 25M/241M
[INFO] ------------------------------------------------------------------------
```

b. **注释掉 maven-enforcer-plugin** 插件的 **app-jee-jsp/config/keycloak.json** 要求并构建 **app-jee-jsp** 应用。

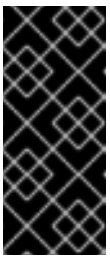
```
service-jee-jaxrs]$ cd ../app-jee-jsp/
```

```
app-jee-jsp]$ sed -i /\<executions\>/s/^/\<\!--/ pom.xml
```

```
app-jee-jsp]$ sed -i '/\(<\/executions>\)/a\-->' pom.xml
```

```
app-jee-jsp]$ mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Keycloak Quickstart: app-jee-jsp 3.1.0.Final
[INFO] ------------------------------------------------------------------------
...
[INFO] Building war: /tmp/github/keycloak-quickstarts/app-jee-jsp/target/app-jsp.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.018 s
[INFO] Finished at: 2017-06-26T12:22:25+02:00
[INFO] Final Memory: 35M/310M
[INFO] ------------------------------------------------------------------------
```



重要

app-jee-jsp Quickstart 要求您配置适配器，且适配器配置文件 (**keycloak.json**)位于 quickstart 根目录下的 **config/** 目录中，才能成功构建快速启动。但本例稍后会通过 EAP 6.4 / 7.1 为 OpenShift 镜像提供的所选环境变量来配置适配器，因此目前不需要指定 **keycloak.json** 适配器配置文件的形式。

4. 在本地文件系统中准备目录结构。
主二进制构建 **目录的 deployment/** 子目录中的应用存档直接复制到 OpenShift 上构建的镜像的标准部署目录中。对于要部署的应用，必须正确构建包含 Web 应用数据的目录层次结构。

   在本地文件系统和 **部署/** 子目录中为二进制构建创建主目录。将 **service-jee-jaxrs** 和 **app-jee-jsp** Quickstart 的之前构建 WAR 存档复制到 **deployments/** 子目录：

```
app-jee-jsp]$ ls
config  pom.xml  README.md  src  target
```

```
app-jee-jsp]$ mkdir -p sso-eap7-bin-demo/deployments
```

```
app-jee-jsp]$ cp target/app-jsp.war sso-eap7-bin-demo/deployments/
```

```
app-jee-jsp]$ cp ../service-jee-jaxrs/target/service.war sso-eap7-bin-demo/deployments/
```

```
app-jee-jsp]$ tree sso-eap7-bin-demo/
sso-eap7-bin-demo/
|__ deployments
    |__ app-jsp.war
    |__ service.war

1 directory, 2 files
```

注意

标准部署目录的位置取决于用于部署应用的底层基础镜像。请参见以下表：

表 4.1. 部署目录的标准位置

底层基础镜像的名称	部署目录的标准位置
用于 OpenShift 6.4 和 7.1 的 EAP	*$JBOSS_HOME/standalone/deployments*
用于 OpenShift 的 Java S2I	*/deployments*
用于 OpenShift 的 JWS	*$JWS_HOME/webapps*

5. 识别 EAP 6.4 / 7.1 镜像的镜像流。

```
$ oc get is -n openshift | grep eap | cut -d ' ' -f 1
jboss-eap64-openshift
jboss-eap71-openshift
```

6. 新建二进制构建，指定镜像流和应用程序名称。

注意

将 **--image-stream=jboss-eap71-openshift** 参数替换为以下 oc 命令中的 **--image-stream=jboss-eap64-openshift**，以在 OpenShift 镜像的 JBoss EAP 6.4 之上部署 JSP 应用程序。

```
$ oc new-build --binary=true \
--image-stream=jboss-eap71-openshift \
--name=eap-app
--> Found image 31895a4 (3 months old) in image stream "openshift/jboss-eap71-openshift"
under tag "latest" for "jboss-eap71-openshift"

    JBoss EAP 7.4
    -------------
    Platform for building and running Jakarta EE applications on JBoss EAP 7.4

    Tags: builder, javaee, eap, eap7

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream "eap-app:latest"
      * A binary build was created, use 'start-build --from-dir' to trigger a new build

--> Creating resources with label build=eap-app ...
    imagestream "eap-app" created
    buildconfig "eap-app" created
--> Success
```

7. 启动二进制构建。指示 **oc** 可执行文件使用在上一步中创建的二进制构建 的主目录作为 包含 OpenShift 构建二进制输入的目录。在 **app-jee-jsp** 的工作目录中，发出以下命令。

```
app-jee-jsp]$ oc start-build eap-app \
--from-dir=./sso-eap7-bin-demo/ \
--follow
Uploading directory "sso-eap7-bin-demo" as binary input for the build ...
build "eap-app-1" started
Receiving source from STDIN as archive ...
Copying all war artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all ear artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all rar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all jar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all war artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
'/home/jboss/source/deployments/app-jsp.war' -> '/opt/eap/standalone/deployments/app-
jsp.war'
'/home/jboss/source/deployments/service.war' ->
'/opt/eap/standalone/deployments/service.war'
Copying all ear artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
Copying all rar artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
Copying all jar artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
Pushing image 172.30.82.129:5000/eap-app-demo/eap-app:latest ...
Pushed 6/7 layers, 86% complete
Pushed 7/7 layers, 100% complete
Push successful
```

8. 基于构建创建新的 OpenShift 应用。

```
$ oc new-app eap-app
--> Found image 6b13d36 (2 minutes old) in image stream "eap-app-demo/eap-app" under
tag "latest" for "eap-app"

    eap-app-demo/eap-app-1:aa2574d9
    -------------------------------
    Platform for building and running Jakarta EE applications on JBoss EAP 7.4

    Tags: builder, javaee, eap, eap7

    * This image will be deployed in deployment config "eap-app"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by service "eap-app"
      * Other containers can access this service through the hostname "eap-app"

--> Creating resources ...
    deploymentconfig "eap-app" created
    service "eap-app" created
--> Success
    Run 'oc status' to view your app.
```

9. 在当前命名空间中停止 EAP 6.4 / 7.1 JSP 应用程序的所有容器。

```
$ oc get dc -o name
deploymentconfig/eap-app
```

```
$ oc scale dc/eap-app --replicas=0
deploymentconfig "eap-app" scaled
```

10. 在部署前，进一步配置 EAP 6.4 / 7.1 JSP 应用程序。

    a. 针对 Red Hat Single Sign-On 服务器实例，配置应用程序。

> **警告**
>
> 确保将以下 ***SSO_PUBLIC_KEY*** 变量的值替换为 **demo** 域的 RSA 公钥的实际内容，该变量的值已 复制。

```
$ oc set env dc/eap-app \
-e HOSTNAME_HTTP="eap-app-eap-app-demo.openshift.example.com" \
-e HOSTNAME_HTTPS="secure-eap-app-eap-app-demo.openshift.example.com" \
-e SSO_DISABLE_SSL_CERTIFICATE_VALIDATION="true" \
-e SSO_USERNAME="appuser" \
-e SSO_PASSWORD="apppassword" \
-e SSO_REALM="demo" \
-e SSO_URL="https://secure-sso-sso-app-demo.openshift.example.com/auth" \
-e
SSO_PUBLIC_KEY="MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkdhXyK
x97oIoO6HwnV/MiX2EHO55Sn+ydsPzbjJevI5F31UvUco9uA8dGl6oM8HrnaWWv+i8PvmIa
RMhhl6Xs68vJTEc6d0soP+6A+aExw0coNRp2PDwvzsXVWPvPQg3+iytStxu3Icndx+gC0ZY
nxoRqL7rY7zKcQBScGEr78Nw6vZDwfe6d/PQ6W4xVErNytX9KyLFVAE1VvhXALyqEM/E
qYGLmpjw5bMGVKRXnhmVo9E88CkFDH8E+aPiApb/gFul1GJOv+G8ySLoR1c8Y3L29F7
C81odkVBp2yMm3RVFIGSPTjHqjO/nOtqYIfY4Wyw9mRIoY5SyW7044dZXRwIDAQAB"
\
-e SSO_SECRET="0bb8c399-2501-4fcd-a183-68ac5132868d"
deploymentconfig "eap-app" updated
```

    b. 为应用配置 SSL 和 JGroups 密钥存储的详细信息。

```
$ oc set env dc/eap-app \
-e HTTPS_KEYSTORE_DIR="/etc/eap-secret-volume" \
-e HTTPS_KEYSTORE="eapkeystore.jks" \
-e HTTPS_PASSWORD="password" \
-e JGROUPS_ENCRYPT_SECRET="eap-jgroup-secret" \
-e JGROUPS_ENCRYPT_KEYSTORE_DIR="/etc/jgroups-encrypt-secret-volume" \
-e JGROUPS_ENCRYPT_KEYSTORE="eapjgroups.jceks" \
-e JGROUPS_ENCRYPT_PASSWORD="password"
deploymentconfig "eap-app" updated
```

    c. 为之前创建的 SSL 和 JGroups 机密定义 OpenShift 卷。

```
$ oc volume dc/eap-app --add \
```

```
--name="eap-keystore-volume" \
--type=secret \
--secret-name="eap-ssl-secret" \
--mount-path="/etc/eap-secret-volume"
deploymentconfig "eap-app" updated
```

```
$ oc volume dc/eap-app --add \
--name="eap-jgroups-keystore-volume" \
--type=secret \
--secret-name="eap-jgroup-secret" \
--mount-path="/etc/jgroups-encrypt-secret-volume"
deploymentconfig "eap-app" updated
```

d. 配置应用的部署配置，以在 **默认的** OpenShift 服务帐户（默认设置）下运行应用容器集。

```
$ oc patch dc/eap-app --type=json \
-p '[{"op": "add", "path": "/spec/template/spec/serviceAccountName", "value": "default"}]'
"eap-app" patched
```

11. 使用修改后的部署配置部署 EAP 6.4 / 7.1 JSP 应用的容器。

```
$ oc scale dc/eap-app --replicas=1
deploymentconfig "eap-app" scaled
```

12. 将服务作为路由公开。

```
$ oc get svc -o name
service/eap-app
```

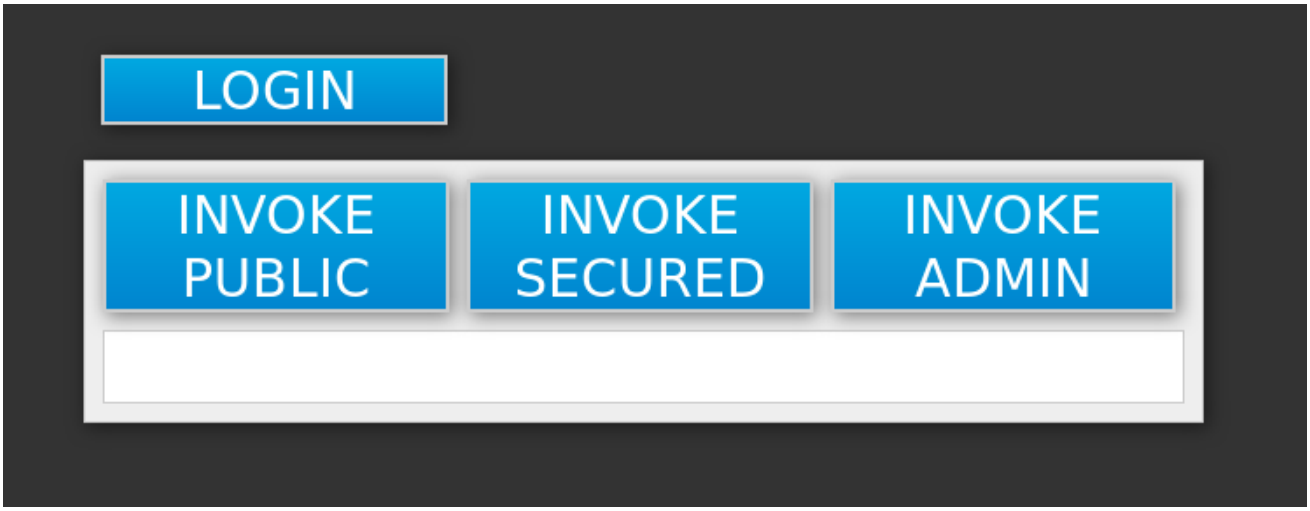
```
$ oc get route
No resources found.
```

```
$ oc expose svc/eap-app
route "eap-app" exposed
```

```
$ oc get route
NAME      HOST/PORT                                  PATH      SERVICES   PORT
TERMINATION   WILDCARD
eap-app   eap-app-eap-app-demo.openshift.example.com             eap-app    8080-tcp
None
```

4.4.1.5. 访问应用程序

在浏览器中使用 URL **http://eap-app-eap-app-demo.openshift.example.com/app-jsp** 访问应用程序。您应该会在以下镜像中看到类似如下的输出：

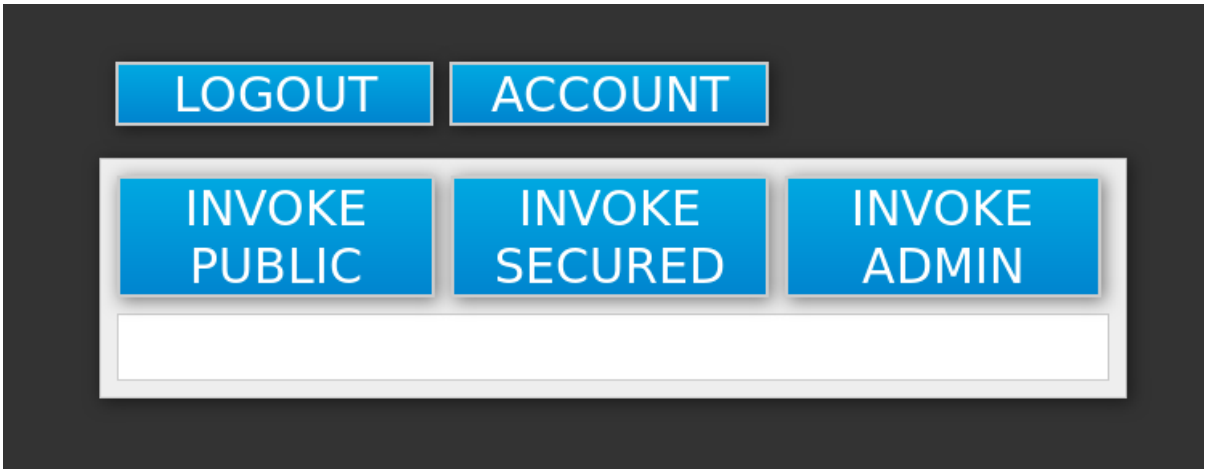


流程

执行以下操作来测试应用程序：

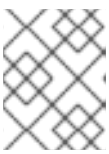
1. 单击 **INVOKE PUBLIC** 按钮，以访问不需要身份验证的 **公共端点**。
   您应看到 **Message: public** 输出。

2. 单击 **LOGIN** 按钮，以针对 **演示** 域将用户身份验证重定向到红帽单点登录服务器实例。
   指定之前配置的 Red Hat Single Sign-On 用户的用户名和密码(**appuser** / **apppassword**)。点**登录**。查看应用程序变化，具体如下镜像：



3. 单击 **INVOKE SECURED** 按钮来访问安全 **端点**。
   您应该看到 **Message: secure output**。

4. 点 **INVOKE ADMIN** 按钮访问 **admin** 端点。
   您应该看到 **403 Forbidden** 输出。



注意

admin 端点需要 **admin** Red Hat Single Sign-On 角色来调用属性。禁止 **appuser** 的访问，因为它们只有 **user** 角色特权，这允许他们访问 **secured** 端点。

流程

执行以下步骤将 **appuser** 添加到 **admin** Red Hat Single Sign-On 角色：

1. 访问红帽单点登录服务器的管理控制台。

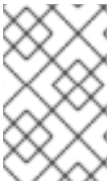
https://secure-sso-sso-app-demo.openshift.example.com/auth/admin.

使用 Red Hat Single Sign-On 管理员用户 的凭据 。

2. 单击 **Manage** sidebar 中的 **Users**，以查看 **demo** 域的用户信息。
3. 点 **View all users** 按钮。
4. 点 **appuser** 的 ID 链接，或者点 **Actions** 列中的 **Edit** 按钮。
5. 点 **Role Mappings** 选项卡。
6. 在 **Realm Roles** 行中，选择 **Available Roles** 列表中的 **admin** 条目。
7. 点 **Add selected** > 按钮将 **admin** 角色添加到用户。
8. 返回到 EAP 6.4 / 7.1 JSP 服务应用程序。
http://eap-app-eap-app-demo.openshift.example.com/app-jsp.
9. 单击 **LOGOUT** 按钮，以重新加载 **appuser** 的角色映射。
10. 再次单击 **LOGIN** 按钮和 provider **appuser** 凭据。
11. 再次单击 **INVOKE ADMIN** 按钮。
您应该看到 **Message: admin** output already。

4.5. 在带有 OPENID-CONNECT 客户端的 RED HAT SINGLE SIGN-ON 中自动注册 EAP 应用程序

本例使用 OpenID-Connect 客户端适配器为 EAP 项目准备红帽单点登录、角色和用户凭据。然后，EAP 中为 OpenShift 模板提供了这些凭证，以自动注册红帽单点登录客户端注册。部署后，可以使用 Red Hat Single Sign-On 用户进行验证和访问 JBoss EAP。



注意

这个示例使用 OpenID-Connect 客户端，但也可以使用 SAML 客户端。如需有关 OpenID-Connect 和 SAML 客户端之间的区别的更多信息，请参阅 Red Hat Single Sign-On Clients 和 Automatic and Manual Red Hat Single Sign-On Client Registration Methods 。

先决条件

- 已执行 为 OpenShift Deployment 准备红帽单点登录身份验证 部分中描述的步骤。

4.5.1. 为 OpenShift 部署准备红帽单点登录身份验证

使用包含 *cluster:admin* 角色的用户登录 OpenShift CLI。

1. 创建一个新项目

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

2. 将 **view** 角色添加到 **default** 服务帐户。这可让服务帐户查看 **eap-app-demo** 命名空间中的所有资源，这是管理集群所必需的。

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

3. EAP 模板需要 SSL 密钥存储和 JGroups 密钥存储。
本例使用 **keytool**（Java Development Kit 中包含的软件包）为这些密钥存储生成自签名证书。
以下命令将提示输入密码。

   a. 为 SSL 密钥存储生成安全密钥：

   ```
   $ keytool -genkeypair -alias https -storetype JKS -keystore eapkeystore.jks
   ```

   b. 为 JGroups 密钥存储生成安全密钥：

   ```
   $ keytool -genseckey -alias jgroups -storetype JCEKS -keystore eapjgroups.jceks
   ```

4. 使用 SSL 和 JGroup 密钥存储文件为 OpenShift 机密生成 EAP：

```
$ oc create secret generic eap-ssl-secret --from-file=eapkeystore.jks
$ oc create secret generic eap-jgroup-secret --from-file=eapjgroups.jceks
```

5. 将 EAP secret 添加到 **default** 服务帐户：

```
$ oc secrets link default eap-ssl-secret eap-jgroup-secret
```

4.5.2. 准备红帽单点登录凭证

使用在红帽单点登录部署期间创建的 管理员帐户 登录到加密的红帽单点登录 Web 服务器 **https://secure-sso-*<project-name>*.*<hostname>*/auth/admin**。

流程

创建一个 Realm

1. 将光标悬停在侧边栏顶部的 realm 命名空间上，然后单击 **Add Realm**。
2. 输入 realm 名称（本例使用 *eap-demo*），然后点 **Create**。

复制公钥

在新创建的 *eap-demo* 域中，单击 **Keys** 选项卡并复制生成的公钥。这个示例为 brevity 使用变量 &*lt;realm-public-key&* gt;。这用于部署启用了 Red Hat Single Sign-On 的 JBoss EAP 镜像。

创建角色

在红帽单点登录中创建角色，其名称对应于示例 EAP 应用的 **web.xml** 中定义的 JEE 角色。此角色分配到 Red Hat Single Sign-On *应用用户*，以验证对用户应用的访问权限。

1. 单击 **Configure** 边栏中的 **Roles**，以列出此域的角色。这是一个新域，因此只有默认的 *offline_access* 角色。
2. 点 **Add Role**。
3. 输入角色名称（本例使用 role *eap-user-role*），然后单击 **Save**。

创建用户和分配角色

创建两个用户： - 为 **realm-management** 角色分配 *域管理用户*，以便在 Red Hat Single Sign-On 服务器中处理自动的 Red Hat Single Sign-On 客户端注册。- 分配在上一步中创建的*应用程序用户* JEE 角色，以验证对用户应用程序的访问权限。

创建 *realm 管理用户* ：

1. 单击 **Manage** sidebar 中的 **Users**，以查看域的用户信息。
2. 单击 **添加用户**。
3. 输入一个有效的 **Username** （本例使用用户 *eap-mgmt-user*）并单击 **Save**。
4. 编辑用户配置。单击用户空间中的 **Credentials** 选项卡，然后输入用户的密码。确认密码后，您可以单击 **重置密码** 以设置用户密码。弹出窗口提示进行额外的确认。
5. 单击 **Role Mappings**，以列出 realm 和 client 角色配置。在 **Client Roles** 下拉菜单中，选择 **realm-management** 并将所有可用的角色添加到用户。这提供了红帽单点登录服务器权限，供 JBoss EAP 镜像用于创建客户端。

创建应用程序用户 ：

1. 单击 **Manage** sidebar 中的 **Users**，以查看域的用户信息。
2. 单击 **添加用户**。
3. 输入一个有效的 **Username** 以及 *应用程序用户* 的额外可选信息，然后点 **Save**。
4. 编辑用户配置。单击用户空间中的 **Credentials** 选项卡，然后输入用户的密码。确认密码后，您可以单击 **重置密码** 以设置用户密码。弹出窗口提示进行额外的确认。
5. 单击 **Role Mappings**，以列出 realm 和 client 角色配置。在 **Available Roles** 中，添加之前创建的角色。

4.5.3. 部署启用了 Red Hat Single Sign-On 的 JBoss EAP 镜像

流程

1. 返回到 OpenShift Web 控制台，点 **Add to project** 列出默认镜像流和模板。
2. 使用 **Filter by keyword** 搜索栏，将列表限制为与 *sso* 匹配的列表。您可能需要点击 **See all** 以显示所需的应用程序模板。
3. 选择 ***eap71-sso-s2i*** 镜像，以列出所有部署参数。包含以下 Red Hat Single Sign-On 参数，以在 EAP 构建期间配置 Red Hat Single Sign-On 凭证：

变量	示例值
APPLICATION_NAME	*sso*
HOSTNAME_HTTPS	*secure-sample-jsp.eap-app-demo.openshift32.example.com*
HOSTNAME_HTTP	*sample-jsp.eap-app-demo.openshift32.example.com*

变量	示例值
SOURCE_REPOSITORY_URL	*https://repository-example.com/developer/application*
SSO_URL	*https://secure-sso-sso-app-demo.openshift32.example.com/auth*
SSO_REALM	*eap-demo*
SSO_USERNAME	*eap-mgmt-user*
SSO_PASSWORD	*password*
SSO_PUBLIC_KEY	*<realm-public-key>*
HTTPS_KEYSTORE	*eapkeystore.jks*
HTTPS_PASSWORD	*password*
HTTPS_SECRET	*EAP-ssl-secret*
JGROUPS_ENCRYPT_KEYSTORE	*eapjgroups.jceks*
JGROUPS_ENCRYPT_PASSWORD	*password*
JGROUPS_ENCRYPT_SECRET	*EAP-jgroup-secret*

4. 单击 **Create** 以部署 JBoss EAP 镜像。

部署 JBoss EAP 映像可能需要几分钟时间。

4.5.4. 使用红帽单点登录登录 JBoss EAP 服务器

流程

1. 访问 JBoss EAP 应用服务器，然后单击 **登录**。您将被重定向到 Red Hat Single Sign-On 登录。
2. 使用在示例中创建的 Red Hat Single Sign-On 用户登录。您已通过 Red Hat Single Sign-On 服务器进行身份验证，并返回给 JBoss EAP 应用服务器。

4.6. 在 RED HAT SINGLE SIGN-ON 中手动将 EAP 应用程序注册到 SAML 客户端

本例为 EAP 项目准备红帽单点登录域、角色和用户凭据，并为 OpenShift 部署配置 EAP。部署后，可以使用 Red Hat Single Sign-On 用户进行验证和访问 JBoss EAP。

注意

这个示例使用 SAML 客户端，但也可以使用 OpenID-Connect 客户端。如需有关 OpenID-Connect 和 SAML 客户端之间的区别的更多信息，请参阅 Red Hat Single Sign-On Clients 和 Automatic and Manual Red Hat Single Sign-On Client Registration Methods 。

先决条件

- 已执行 为 OpenShift Deployment 准备红帽单点登录身份验证 部分中描述的步骤。

4.6.1. 准备红帽单点登录凭证

流程

使用在红帽单点登录部署期间创建的 管理员帐户 登录到加密的红帽单点登录 Web 服务器 **https://secure-sso-*<project-name>*.*<hostname>*/auth/admin**。

创建一个 Realm

1. 将光标悬停在域命名空间（默认为 **主**栏的顶部），然后单击 **Add Realm**。
2. 输入 realm 名称（本例使用 *saml-demo*），然后点 **Create**。

复制公钥

在新创建的 *saml-demo* 域中，单击 **Keys** 选项卡并复制生成的公钥。本例使用变量 *realm-public-key* 进行 brevity。之后需要部署启用了 Red Hat Single Sign-On 的 JBoss EAP 镜像。

创建角色

在红帽单点登录中创建角色，其名称对应于示例 EAP 应用的 **web.xml** 中定义的 JEE 角色。此角色将分配给 Red Hat Single Sign-On *应用用户*，以验证对用户应用的访问权限。

1. 单击 **Configure** 边栏中的 **Roles**，以列出此域的角色。这是一个新域，因此只有默认的 *offline_access* 角色。
2. 点 **Add Role**。
3. 输入角色名称（本例使用 role *saml-user-role*）并点 **Save**。

创建用户和分配角色

创建两个用户： - 为 **realm-management** 角色分配 *域管理用户*，以便在 Red Hat Single Sign-On 服务器中处理自动的 Red Hat Single Sign-On 客户端注册。 - 分配在上一步中创建的*应用程序用户* JEE 角色，以验证对用户应用程序的访问权限。

创建 *realm 管理用户* ：

1. 单击 **Manage** sidebar 中的 **Users**，以查看域的用户信息。
2. 单击 **添加用户**。
3. 输入一个有效的 **Username** （本示例使用用户 *app-mgmt-user*）并点 **Save**。
4. 编辑用户配置。单击用户空间中的 **Credentials** 选项卡，然后输入用户的密码。确认密码后，您可以单击 **重置密码** 以设置用户密码。弹出窗口提示进行额外的确认。

创建应用程序用户：

1. 单击 **Manage** sidebar 中的 **Users**，以查看域的用户信息。
2. 单击 **添加用户**。
3. 输入一个有效的 **Username** 以及 *应用程序用户* 的额外可选信息，然后点 **Save**。
4. 编辑用户配置。单击用户空间中的 **Credentials** 选项卡，然后输入用户的密码。确认密码后，您可以单击 **重置密码** 以设置用户密码。弹出窗口提示进行额外的确认。
5. 单击 **Role Mappings**，以列出 realm 和 client 角色配置。在 **Available Roles** 中，添加之前创建的角色。

创建并配置 SAML 客户端：

客户端是请求用户身份验证的红帽单点登录实体。本例配置 SAML 客户端，以处理 EAP 应用的身份验证。本节保存两个文件 **keystore.jks** 和 **keycloak-saml-subsystem.xml**，它们会在以后的过程中需要。

创建 SAML 客户端：

1. 单击 **Configure** 边栏中的 **Clients**，以列出域中的客户端。点 **Create**。
2. 输入一个有效 **的客户端 ID**。这个示例使用 *sso-saml-demo*。
3. 在 **Client Protocol** 下拉菜单中选择 **saml**。
4. 输入应用程序的 **Root URL**。这个示例使用 *https://demoapp-eap-app-demo.openshift32.example.com*。
5. 点击 **Save**。

配置 SAML 客户端：

在 **Settings** 选项卡中，为新的 ***sso-saml-demo*** 客户端设置 **Root URL** 和 **Valid Redirect URL**：

1. 对于 **Root URL**，请输入创建客户端时使用的相同地址。这个示例使用 *https://demoapp-eap-app-demo.openshift32.example.com*。
2. 对于 **有效的重定向 URL**，在登录或注销时输入要重定向到的用户的地址。这个示例使用相对于 root *https://demoapp-eap-app-demo.openshift32.example.com/** 的重定向地址。

导出 SAML 密钥：

1. 点 *sso-saml-demo* 客户端空间中的 **SAML Keys** 标签页，点 **Export**。
2. 在本例中，将 **归档格式** 保留为 **JKS**。这个示例使用 *sso-saml-demo* 的默认 **Key Alias** 和 default **Realm Certificate Alias** of *saml-demo*。
3. 输入密钥密码 **，** 并输入 **"存储密码** "。这个示例使用 *password*。
4. 单击 **Download** 并保存 **密钥存储-saml.jks** 文件，以便稍后使用。
5. 单击 ***sso-saml-demo*** 客户端，以返回到客户端空间以供下一步使用。

下载客户端适配器：

1. 点击 **Installation**。

2. 使用 **Format Option** 下拉菜单选择格式。本例使用 **Keycloak SAML Wildfly/JBoss subsystem**。

3. 点 **Download** 并保存 **keycloak-saml-subsystem.xml** 文件。

keystore-saml.jks 将与下一部分中的其他 EAP 密钥存储一起使用，以创建 EAP 应用项目的 OpenShift 机密。将 **密钥存储-saml.jks** 文件复制到 OpenShift 节点。
keycloak-saml-subsystem.xml 将修改并在应用程序部署中使用。将它复制到应用的 **/configuration** 文件夹作为 **secure-saml-deployments**。

4.6.2. 为 OpenShift 部署准备红帽单点登录身份验证

使用包含 *cluster:admin* 角色的用户登录 OpenShift CLI。

流程

1. 创建一个新项目

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

2. 将 **view** 角色添加到 **default** 服务帐户。这可让服务帐户查看 **eap-app-demo** 命名空间中的所有资源，这是管理集群所必需的。

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

3. EAP 模板需要 SSL 密钥存储和 JGroups 密钥存储。
本例使用 **keytool**（Java Development Kit 中包含的软件包）为这些密钥存储生成自签名证书。以下命令将提示输入密码。

   a. 为 SSL 密钥存储生成安全密钥：

```
$ keytool -genkeypair -alias https -storetype JKS -keystore eapkeystore.jks
```

   b. 为 JGroups 密钥存储生成安全密钥：

```
$ keytool -genseckey -alias jgroups -storetype JCEKS -keystore eapjgroups.jceks
```

4. 使用 SSL 和 JGroup 密钥存储文件为 OpenShift 机密生成 EAP：

```
$ oc create secret generic eap-ssl-secret --from-file=eapkeystore.jks
$ oc create secret generic eap-jgroup-secret --from-file=eapjgroups.jceks
```

5. 将 EAP 应用程序 secret 添加到之前创建的 EAP 服务帐户中：

```
$ oc secrets link default eap-ssl-secret eap-jgroup-secret
```

4.6.3. 修改 secure-saml-deployments 文件

先决条件

- **keycloak-saml-subsystem.xml** 从上一节中的 Red Hat Single Sign-On 客户端导出，应该已复制到应用程序的 **/configuration** 文件夹中，并重命名了 **secure-saml-deployments**。EAP 会在启动该文件并将其复制到 Red Hat Single Sign-On SAML 适配器配置中的 **standalone-**

openshift.xml 文件时搜索此文件。

流程

1. 在文本编辑器中打开 /configuration/secure-saml-deployments 文件。
2. 将 secure-deployment name 标签的 YOUR-WAR.war 值替换为应用程序的 .war 文件。本例使用 *sso-saml-demo.war*。
3. 将 logout page 标签的 SPECIFY YOUR LOGOUT PAGE! 值替换为在登出应用程序时将用户重新导向的 url。这个示例使用 /index.jsp。
4. 删除 <PrivateKeyPem> 和 <CertificatePem> 标签和密钥，并将它替换为密钥存储信息：

```
...
<Keys>
  <Key signing="true">
    <KeyStore file= "/etc/eap-secret-volume/keystore-saml.jks" password="password">
      <PrivateKey alias="sso-saml-demo" password="password"/>
      <Certificate alias="sso-saml-demo"/>
    </KeyStore>
  </Key>
</Keys>
```

keystore-saml.jks 的挂载路径（本例中为 */etc/eap-secret-volume/keystore-saml.jks*）可使用 EAP_HTTPS_KEYSTORE_DIR 参数在应用程序模板中指定。
当从 Red Hat Single Sign-On 客户端导出 SAML 密钥时，配置的 PrivateKey 和 Certificate 的别名和密码。

5. 删除第二个 <CertificatePem> 标签和密钥，并将其替换为 realm 证书信息：

```
...
<Keys>
  <Key signing="true">
    <KeyStore file="/etc/eap-secret-volume/keystore-saml.jks" password="password">
      <Certificate alias="saml-demo"/>
    </KeyStore>
  </Key>
</Keys>
...
```

从 Red Hat Single Sign-On 客户端导出 SAML 密钥时，会配置证书别名和密码。

6. 保存并关闭 /configuration/secure-saml-deployments 文件。

4.6.4. 在应用程序 web.xml中配置 SAML 客户端注册

客户端类型还必须由应用程序 web.xml 中的 <auth-method> 键指定。在部署时，该文件由镜像读取。

打开应用程序 web.xml 文件，并确保它包括以下内容：

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

```
</login-config>
...
```

4.6.5. 部署应用程序

您不需要包含镜像的 Red Hat Single Sign-On 配置，因为应用程序本身中已经配置。导航到应用程序登录页面，将您重定向到 Red Hat Single Sign-On 登录。使用之前创建的 *应用用户*，通过 Red Hat Single Sign-On 登录应用程序。

第 5 章 参考

5.1. 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（所有项目 jars、库 jar、插件或任何其他项目特定工件）。它还指定在执行 S2I 构建时从中下载工件的位置。除了使用中央存储库外，组织也是一种常见做法，部署本地自定义存储库(mirror)。

使用镜像的好处有：

- 同步镜像的可用性，这种镜像更为严格且更快。
- 能够对存储库内容有更大的控制。
- 可能会在不同的团队（developers、CI）共享工件，而无需依赖公共服务器和存储库。
- 改进构建时间。

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经在 ***http://10.0.0.1:8080/repository/internal/*** 进行了部署并可访问，然后构建 S2I 构建可通过以下流程为应用程序的构建配置提供 **MAVEN_MIRROR_URL** 环境变量：

流程

1. 识别构建配置的名称，以应用 **MAVEN_MIRROR_URL** 变量。

```
$ oc get bc -o name
buildconfig/sso
```

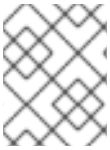
2. 使用 **MAVEN_MIRROR_URL** 环境变量更新 **sso** 的构建配置。

```
$ oc set env bc/sso \
  -e MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"
buildconfig "sso" updated
```

3. 验证设置。

```
$ oc set env bc/sso --list
# buildconfigs sso
MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/
```

4. 调度应用的新构建。



注意

在应用构建期间，您将注意到从存储库管理器中提取 Maven 依赖项，而不是默认的公共存储库。另外，在构建完成后，您会看到镜像已填充并在构建期间使用的所有依赖项。

5.2. 环境变量

5.2.1. 信息环境变量

以下信息环境变量旨在提供有关镜像的信息，且不应由用户修改：

表 5.1. 信息环境变量

变量名称	描述	示例值
AB_JOLOKIA_AUTH_OPENSHIFT	-	*true*
AB_JOLOKIA_HTTPS	-	*true*
AB_JOLOKIA_PASSWORD_RANDOM	-	*true*
JBOSS_IMAGE_NAME	镜像名称，与 "name" 标签相同。	*rh-sso-7/sso76-openshift-rhel8*
JBOSS_IMAGE_VERSION	镜像版本，与 "version" 标签相同。	*7.6*
JBOSS_MODULES_SYSTEM_PKGS	-	*org.jboss.logmanager,jdk.nashorn.api*

5.2.2. 配置环境变量

配置环境变量旨在方便调整镜像，而无需重新构建，应根据需要为用户设置。

表 5.2. 配置环境变量

变量名称	描述	示例值
AB_JOLOKIA_AUTH_OPENSHIFT	切换到 OpenShift TLS 通信的客户端身份验证。此参数的值可以是相对可区分名称，它必须包含在出示客户端的证书中。启用此参数将自动将 Jolokia 切换到 https 通信模式。默认 CA 证书设置为 **/var/run/secrets/kubernetes.io/serviceaccount/ca.crt**。	*true*
AB_JOLOKIA_CONFIG	如果设置使用此文件（包括路径）作为 Jolokia JVM 代理属性（如 Jolokia 的 参考手册中所述）。如果没有设置，则将使用本文档中定义的设置创建 **/opt/jolokia/etc/jolokia.properties** 文件，否则忽略本文档中的其他设置。	*/opt/jolokia/custom.properties*
AB_JOLOKIA_DISCOVERY_ENABLED	启用 Jolokia 发现。默认为 ***false***。	*true*

变量名称	描述	示例值
AB_JOLOKIA_HOST	要绑定到的主机地址。默认为 *0.0.0.0*。	*127.0.0.1*
AB_JOLOKIA_HTTPS	切换到安全与 https 的通信。默认情况下，如果在 *AB_JOLOKIA_OPTS* 中未提供 serverCert 配置，则会生成自签名证书。*注意：如果将值设置为空字符串，则 **关闭** https。如果值被设置为非空字符串，则在 **上打开** https。*	*true*
AB_JOLOKIA_ID	要使用的代理 ID（默认为 $HOSTNAME，这是容器 ID）。	*openjdk-app-1-xqlsj*
AB_JOLOKIA_OFF	如果设置禁用 Jolokia 激活（即回显空值）。默认情况下启用 Jolokia。*注意：如果将值设置为空字符串，则 **关闭** https。如果值被设置为非空字符串，则在 **上打开** https。*	*true*
AB_JOLOKIA_OPTS	要附加到代理配置的额外选项。它们应该采用 **"key=value, key=value, ...<200b> "**	*backlog=20*
AB_JOLOKIA_PASSWORD	用于基本身份验证的密码。默认情况下关闭身份验证。	*mypassword*
AB_JOLOKIA_PASSWORD_RANDOM	如果设置，会为 *AB_JOLOKIA_PASSWORD* 生成随机值，并保存在 */opt/jolokia/etc/jolokia.pw* 文件中。	*true*
AB_JOLOKIA_PORT	要使用的端口（默认为 *8778*）。	*5432*
AB_JOLOKIA_USER	用于基本身份验证的用户。默认为 *jolokia*。	*myusername*
CONTAINER_CORE_LIMIT	CFS Bandwidth Control 中描述的计算内核限制。	*2*
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	赋予当前的 Garbage Collection(GC)时间与之前的 GC 时间的权重。	*90*

变量名称	描述	示例值
GC_MAX_HEAP_FREE_RATIO	GC 年后释放的最大堆百分比以避免缩小。	*40*
GC_MAX_METASPACE_SIZE	最大元空间大小。	*100*
GC_TIME_RATIO_MIN_HEAP_FREE_RATIO	GC 年后释放的最小堆百分比以避免扩展。	*20*
GC_TIME_RATIO	指定垃圾回收外部花费的时间比率（例如，应用程序执行所花费的时间）与垃圾回收中花费的时间。	*4*
JAVA_DIAGNOSTICS	把它设置为在发生事件时将一些诊断信息设置为标准输出。	*true*
JAVA_INITIAL_MEM_RATIO	这用于计算基于 maximal 堆内存的默认初始堆内存。默认值为 100，表示将 maximal 堆的 100% 用于初始堆大小。您可以通过将此值设置为 0 来跳过此机制，在添加 no **-Xms** 选项时。	*100*
JAVA_MAX_MEM_RATIO	它用于根据容器限制计算默认最大堆内存。如果在 Docker 容器中使用没有容器内存约束的情况，则此选项无效。如果存在内存限制，则 **-Xmx** 会按照此处设定的容器可用内存比率。默认值为 50，表示可用内存的 50% 用作上限。您可以通过将此值设置为 0 来跳过此机制，在添加了 **-Xmx** 选项时。	*40*
JAVA_OPTS_APPEND	服务器启动选项。	*-Dkeycloak.migration.action=export -Dkeycloak.migration.provider=dir -Dkeycloak.migration.dir=/tmp*
MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION	为了向后兼容，请将 设置为 true，以使用 **MyQueue** 和 **MyTopic** 作为物理目的地名称默认值，而不是 **queue/MyQueue** 和 **topic/MyTopic**。	*false*
OPENSHIFT_KUBE_PING_LABELS	集群标签选择器。	*app=sso-app*
OPENSHIFT_KUBE_PING_NAMESPACE	集群项目命名空间。	*myproject*

变量名称	描述	示例值
SCRIPT_DEBUG	如果设置为 **true**，则 ensurses 使用 **-x** 选项执行 bash 脚本，并在执行这些命令时打印命令及其参数。	*true*
SSO_ADMIN_PASSWORD	Red Hat Single Sign-On 服务器的 **master** realm 的管理员账户的密码。**必需。**如果没有指定值，则在模板实例化时，它会自动生成并显示为 OpenShift 指令消息。	*adm-password*
SSO_ADMIN_USERNAME	Red Hat Single Sign-On 服务器的 **master** realm 的管理员账户的用户名。**必需。**如果没有指定值，则在模板实例化时，它会自动生成并显示为 OpenShift 指令消息。	*admin*
SSO_HOSTNAME	Red Hat Single Sign-On 服务器的自定义主机名。**默认没有设置**。如果没有设置，**request** 主机名 SPI 供应商使用请求标头来确定使用的 Red Hat Single Sign-On 服务器的主机名。如果设置，则使用 **固定** 主机名 SPI 供应商，并将 Red Hat Single Sign-On 服务器的主机名设置为提供的变量值。当设置了 *SSO_HOSTNAME* 变量时，请参阅 Red Hat Single Sign-On Server 的专用自定义主机名部分。	*rh-sso-server.openshift.example.com*
SSO_REALM	如果提供了此环境变量，要在 Red Hat Single Sign-On 服务器中创建的域名称。	*demo*
SSO_SERVICE_PASSWORD	Red Hat Single Sign-On 服务用户的密码。	*mgmt-password*
SSO_SERVICE_USERNAME	用于访问 Red Hat Single Sign-On 服务的用户名。客户端用于在指定的红帽单点登录域内创建应用程序客户端。如果提供了此环境变量，则会创建此用户。	*sso-mgmtuser*
SSO_TRUSTSTORE	secret 中 truststore 文件的名称。	*truststore.jks*
SSO_TRUSTSTORE_DIR	信任存储目录。	*/etc/sso-secret-volume*
SSO_TRUSTSTORE_PASSWORD	truststore 和证书的密码。	*mykeystorepass*

变量名称	描述	示例值
SSO_TRUSTSTORE_SECRET	包含信任存储文件的 secret 名称。用于 *sso-truststore-volume* 卷。	***truststore-secret***

适用于 OpenShift 的红帽单点登录可用的 应用模板 可以将 上述配置变量 与常见 OpenShift 变量（如 ***APPLICATION_NAME*** 或 ***SOURCE_REPOSITORY_URL***）相结合，还具有特定于产品的变量（例如，***HORNETQ_CLUSTER_PASSWORD***）或将典型的配置变量与数据库镜像（例如 ***POSTGRESQL_MAX_CONNECS***）合并。所有这些不同类型的配置变量都可根据需要调整，以实现部署的 Red Hat Single Sign-On-enabled 应用将会尽可能与预期用例保持一致。以下介绍了与支持红帽单点登录的应用程序模板相关的配置变量列表，如下所述。

5.2.3. 所有红帽单点登录镜像的模板变量

表 5.3. 可用于所有红帽单点登录镜像的配置变量

变量	描述
APPLICATION_NAME	应用程序的名称。
DB_MAX_POOL_SIZE	为配置的数据源设置 xa-pool/max-pool-size。
DB_TX_ISOLATION	为配置的数据源设置 transaction-isolation。
DB_USERNAME	数据库用户名。
HOSTNAME_HTTP	http 服务路由的自定义主机名。保留默认主机名的空白，例如：< *application-name*>.<*project*>.<*default-domain-suffix*>。
HOSTNAME_HTTPS	https 服务路由的自定义主机名。保留默认主机名的空白，例如：< *application-name*>.<*project*>.<*default-domain-suffix*>。
HTTPS_KEYSTORE	机密中的密钥存储文件的名称。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 *$JBOSS_HOME/standalone/configuration* 下的相对路径。
HTTPS_KEYSTORE_TYPE	密钥存储文件的类型（JKS 或 JCEKS）。
HTTPS_NAME	与服务器证书关联的名称（如 *jboss*）。如果与 ***HTTPS_PASSWORD*** 和 ***HTTPS_KEYSTORE*** 一起定义，请启用 HTTPS 并设置 SSL 名称。
HTTPS_PASSWORD	密钥存储和证书的密码（如 *mykeystorepass*）。如果与 ***HTTPS_NAME*** 和 ***HTTPS_KEYSTORE*** 一起定义，请启用 HTTPS 并设置 SSL 密钥密码。

变量	描述
HTTPS_SECRET	包含密钥存储文件的机密的名称。
IMAGE_STREAM_NAMESPACE	安装 Red Hat Middleware 镜像的 ImageStreams 的命名空间。这些 ImageStreams 通常安装在 *openshift* 命名空间中。如果您在不同命名空间/项目中安装了 ImageStreams 时，才需要修改它。
JGROUPS_CLUSTER_PASSWORD	JGroup 集群密码。
JGROUPS_ENCRYPT_KEYSTORE	机密中的密钥存储文件的名称。
JGROUPS_ENCRYPT_NAME	与服务器证书关联的名称（如 *secret-key*）。
JGROUPS_ENCRYPT_PASSWORD	密钥存储和证书的密码（如 *password* ）。
JGROUPS_ENCRYPT_SECRET	包含密钥存储文件的机密的名称。
SSO_ADMIN_USERNAME	Red Hat Single Sign-On 服务器的 **master** realm 的管理员账户的用户名。**必需。**如果没有指定值，则在模板实例化时，它会自动生成并显示为 OpenShift 指令消息。
SSO_ADMIN_PASSWORD	Red Hat Single Sign-On 服务器的 **master** realm 的管理员账户的密码。**必需。**如果没有指定值，则在模板实例化时，它会自动生成并显示为 OpenShift 指令消息。
SSO_REALM	如果提供了此环境变量，要在 Red Hat Single Sign-On 服务器中创建的域名称。
SSO_SERVICE_USERNAME	用于访问 Red Hat Single Sign-On 服务的用户名。客户端用于在指定的红帽单点登录域内创建应用程序客户端。如果提供了此环境变量，则会创建此用户。
SSO_SERVICE_PASSWORD	Red Hat Single Sign-On 服务用户的密码。
SSO_TRUSTSTORE	secret 中 truststore 文件的名称。
SSO_TRUSTSTORE_SECRET	包含信任存储文件的 secret 名称。用于 ***sso-truststore-volume*** 卷。
SSO_TRUSTSTORE_PASSWORD	truststore 和证书的密码。

5.2.4. 特定于 *sso76-ocp3-postgresql*、*sso76-ocp4-postgresql*、*sso76-ocp3-postgresql-persistent*、*sso76-ocp4-postgresql-persistent*、*sso76-ocp3-x509-postgresql-persistent*、sso76-postgresql-persistent 和 *sso76-ocp4-x509-*

*postgresql-persistent*的模板变量

表 5.4. 特定于红帽单点登录的 PostgreSQL 应用程序使用临时的持久性存储

变量	描述
DB_USERNAME	数据库用户名。
DB_PASSWORD	数据库用户密码。
DB_JNDI	应用程序用来解析数据源的数据库 JNDI 名称，如 *java:/jboss/datasources/postgresql*
POSTGRESQL_MAX_CONNECTIONS	允许的最大客户端连接数。这也设定了准备的最大事务数量。
POSTGRESQL_SHARED_BUFFERS	配置专用于 PostgreSQL 的内存来缓存数据。

5.2.5. 常规 eap64 和 eap71 S2I 镜像的模板变量

表 5.5. EAP 6.4 和 EAP 7 Applications Built Via S2I 的配置变量

变量	描述
APPLICATION_NAME	应用程序的名称。
ARTIFACT_DIR	工件目录。
AUTO_DEPLOY_EXPLODED	控制是否应该自动部署展开的部署内容。
CONTEXT_DIR	要构建的 Git 项目中的路径；为 root 项目目录为空。
GENERIC_WEBHOOK_SECRET	通用构建触发器机密。
GITHUB_WEBHOOK_SECRET	GitHub 触发器 secret。
HORNETQ_CLUSTER_PASSWORD	HornetQ 集群管理员密码。
HORNETQ_QUEUES	队列名称。
HORNETQ_TOPICS	主题名称。
HOSTNAME_HTTP	http 服务路由的自定义主机名。保留默认主机名的空白，例如：< *application-name*>.<*project*>.<*default-domain-suffix*>。

变量	描述
HOSTNAME_HTTPS	https 服务路由的自定义主机名。保留默认主机名的空白，例如：< *application-name*>.<*project*>.<*default-domain-suffix*>。
HTTPS_KEYSTORE_TYPE	密钥存储文件的类型（JKS 或 JCEKS）。
HTTPS_KEYSTORE	机密中的密钥存储文件的名称。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 *$JBOSS_HOME/standalone/configuration* 下的相对路径。
HTTPS_NAME	与服务器证书关联的名称（如 *jboss*）。如果与 ***HTTPS_PASSWORD*** 和 ***HTTPS_KEYSTORE*** 一起定义，请启用 HTTPS 并设置 SSL 名称。
HTTPS_PASSWORD	密钥存储和证书的密码（如 *mykeystorepass*）。如果与 ***HTTPS_NAME*** 和 ***HTTPS_KEYSTORE*** 一起定义，请启用 HTTPS 并设置 SSL 密钥密码。
HTTPS_SECRET	包含密钥存储文件的机密的名称。
IMAGE_STREAM_NAMESPACE	安装 Red Hat Middleware 镜像的 ImageStreams 的命名空间。这些 ImageStreams 通常安装在 *openshift* 命名空间中。如果您在不同命名空间/项目中安装了 ImageStreams 时，才需要修改它。
JGROUPS_CLUSTER_PASSWORD	JGroup 集群密码。
JGROUPS_ENCRYPT_KEYSTORE	机密中的密钥存储文件的名称。
JGROUPS_ENCRYPT_NAME	与服务器证书关联的名称（如 *secret-key*）。
JGROUPS_ENCRYPT_PASSWORD	密钥存储和证书的密码（如 *password* ）。
JGROUPS_ENCRYPT_SECRET	包含密钥存储文件的机密的名称。
SOURCE_REPOSITORY_REF	Git 分支/标签引用。
SOURCE_REPOSITORY_URL	应用程序的 Git 源 URI。

5.2.6. 特定于 *eap64-sso-s2i* 和 *eap71-sso-s2i* 的模板变量用于自动客户端注册

表 5.6. 适用于 EAP 6.4 和 EAP 7 的配置变量，红帽单点登录应用程序内置了 Via S2I

变量	描述
SSO_URL	红帽单点登录服务器位置.
SSO_REALM	如果提供了此环境变量，要在 Red Hat Single Sign-On 服务器中创建的域名称。
SSO_USERNAME	用于访问 Red Hat Single Sign-On 服务的用户名。这可用于在指定的 Red Hat Single Sign-On 域中创建应用程序客户端。这应当与通过 **sso76-** 模板之一指定的 *SSO_SERVICE_USERNAME* 匹配。
SSO_PASSWORD	Red Hat Single Sign-On 服务用户的密码。
SSO_PUBLIC_KEY	红帽单点登录公钥.建议将公钥传递到模板以避免中间人安全攻击。
SSO_SECRET	红帽单点登录客户端机密机密访问。
SSO_SERVICE_URL	红帽单点登录服务位置.
SSO_TRUSTSTORE_SECRET	包含信任存储文件的 secret 名称。用于 ***sso-truststore-volume*** 卷。
SSO_TRUSTSTORE	secret 中 truststore 文件的名称。
SSO_TRUSTSTORE_PASSWORD	truststore 和证书的密码。
SSO_BEARER_ONLY	Red Hat Single Sign-On 客户端访问类型。
SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	如果 EAP 与 Red Hat Single Sign-On 服务器之间的 true SSL 通信不安全（例如，使用 curl 禁用证书验证）
SSO_ENABLE_CORS	为红帽单点登录应用程序启用 CORS。

5.2.7. 特定于 *eap64-sso-s2i* 和 *eap71-sso-s2i* 的模板变量，用于自动使用 SAML 客户端进行客户端注册

表 5.7. 用于 EAP 6.4 和 EAP 7 的配置变量，红帽单点登录应用程序使用 SAML 协议构建了 Via S2I

变量	描述
SSO_SAML_CERTIFICATE_NAME	与服务器证书关联的名称。
SSO_SAML_KEYSTORE_PASSWORD	密钥存储和证书的密码。

变量	描述
SSO_SAML_KEYSTORE	机密中的密钥存储文件的名称。
SSO_SAML_KEYSTORE_SECRET	包含密钥存储文件的机密的名称。
SSO_SAML_LOGOUT_PAGE	SAML 应用程序的 Red Hat Single Sign-On logout 页面。

5.3. 公开端口

端口号	描述
8443	HTTPS
8778	Jolokia 监控