Red Hat Single Sign-On Continuous Delivery 6 Red Hat Single Sign-On for OpenShift

For Use with Red Hat Single Sign-On Continuous Delivery 6
Abstract

This guide consists of basic information and instructions to get started with Red Hat Single Sign-On Continuous Delivery 6 for OpenShift
# Table of Contents

## CHAPTER 1. INTRODUCTION

1.1. WHAT IS RED HAT SINGLE SIGN-ON? .............................................. 4

## CHAPTER 2. BEFORE YOU BEGIN

2.1. COMPARISON: RED HAT SINGLE SIGN-ON FOR OPENSIFT IMAGE AND RED HAT SINGLE SIGN-ON ... 6
2.2. VERSION COMPATIBILITY AND SUPPORT ........................................ 6
2.3. DEPRECATED IMAGE STREAMS AND APPLICATION TEMPLATES FOR RED HAT SINGLE SIGN-ON FOR OPENSIFT 6
2.4. INITIAL SETUP .............................................................................. 6

## CHAPTER 3. GET STARTED

3.1. USING THE RED HAT SINGLE SIGN-ON FOR OPENSIFT IMAGE STREAMS AND APPLICATION TEMPLATES 7
3.2. PREPARING AND DEPLOYING THE RED HAT SINGLE SIGN-ON FOR OPENSIFT APPLICATION TEMPLATES ... 8
   3.2.1. Configuring Keystores ............................................................ 8
   3.2.2. Generating Secrets .............................................................. 8
   3.2.3. Using the OpenShift Web Console ....................................... 9
   3.2.4. Routes .............................................................................. 10
3.3. BINARY BUILDS ........................................................................ 10
   3.3.1. Deploy Binary Build of EAP 6.4 / 7.1 JSP Service Invocation Application that Authenticates Using Red Hat Single Sign-On 10
      3.3.1.1. Create Red Hat Single Sign-On Realm, Roles, and User for the EAP 6.4 / 7.1 JSP Application 11
      3.3.1.2. Assign user Red Hat Single Sign-On Role to the Realm Management User 12
      3.3.1.3. Prepare Red Hat Single Sign-On Authentication for OpenShift Deployment of the EAP 6.4 / 7.1 JSP Application 13
      3.3.1.4. Deploy Binary Build of the EAP 6.4 / 7.1 JSP Application 14
      3.3.1.5. Access the Application .................................................. 19

## CHAPTER 4. ADVANCED CONCEPTS

4.1. REQUIREMENTS AND DEPLOYING PASSTHROUGH TLS TERMINATION RED HAT SINGLE SIGN-ON TEMPLATES 22
   4.1.1. Preparing the Deployment .................................................... 22
   4.1.2. Creating HTTPS and JGroups Keystores, and Truststore for the Red Hat Single Sign-On Server 22
   4.1.3. Secrets ............................................................................ 24
   4.1.4. Deploying the Chosen Red Hat Single Sign-On Passthrough TLS Template via OpenShift CLI 24
   4.1.5. Accessing the Administrator Console of the Red Hat Single Sign-On Pod 26
4.2. CUSTOMIZING HOSTNAME FOR THE RED HAT SINGLE SIGN-ON SERVER ........................................ 27
4.3. CREATING THE ADMINISTRATOR ACCOUNT FOR RED HAT SINGLE SIGN-ON SERVER 28
   4.3.1. Creating the Administrator Account Using Template Parameters 28
   4.3.2. Creating the Administrator Account via Remote Shell Session to Red Hat Single Sign-On Pod 30
4.4. DEPLOYMENT PROCESS ................................................................. 31
4.5. RED HAT SINGLE SIGN-ON CLIENTS ........................................... 31
   4.5.1. Automatic and Manual Red Hat Single Sign-On Client Registration Methods 32
      4.5.1.1. Automatic Red Hat Single Sign-On Client Registration 32
      4.5.1.2. Manual Red Hat Single Sign-On Client Registration 33
4.6. LIMITATIONS ............................................................................. 34

## CHAPTER 5. TUTORIALS

5.1. EXAMPLE WORKFLOW: PREPARING AND DEPLOYING THE RED HAT SINGLE SIGN-ON FOR OPENSIFT IMAGE 35
   5.1.1. Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment 35
   5.1.2. Deploy Binary Build of EAP 6.4 / 7.1 JSP Service Invocation Application and Secure it Using Red Hat
Single Sign-On

5.1.2.1. Create Red Hat Single Sign-On Realm, Roles, and User for the EAP 6.4 / 7.1 JSP Application
5.1.3. Automatic Database Migration
5.1.4. Manual Database Migration

5.2. EXAMPLE WORKFLOW: MIGRATING ENTIRE RED HAT SINGLE SIGN-ON SERVER DATABASE ACROSS THE ENVIRONMENTS
5.2.1. Deploying the Red Hat Single Sign-On MySQL Application Template
5.2.2. (Optional) Creating additional Red Hat Single Sign-On realm and users to be also exported
5.2.3. Export the Red Hat Single Sign-On database as a JSON file on the OpenShift pod
5.2.4. Retrieve and import the exported JSON file

5.3. EXAMPLE WORKFLOW: CONFIGURING OPENSSHIFT TO USE RED HAT SINGLE SIGN-ON FOR AUTHENTICATION
5.3.1. Configuring Red Hat Single Sign-On Credentials
5.3.2. Configuring OpenShift Master for Red Hat Single Sign-On Authentication
5.3.3. Logging in to OpenShift

5.4. EXAMPLE WORKFLOW: AUTOMATICALLY REGISTERING EAP APPLICATION IN RED HAT SINGLE SIGN-ON WITH OPENID-CONNECT CLIENT
5.4.1. Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment
5.4.2. Preparing the Red Hat Single Sign-On Credentials
5.4.3. Deploy the Red Hat Single Sign-On-enabled JBoss EAP Image
5.4.4. Log in to the JBoss EAP Server Using Red Hat Single Sign-On

5.5. EXAMPLE WORKFLOW: MANUALLY REGISTERING EAP APPLICATION IN RED HAT SINGLE SIGN-ON WITH SAML CLIENT
5.5.1. Preparing the Red Hat Single Sign-On Credentials
5.5.2. Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment
5.5.3. Modifying the secure-saml-deployments File
5.5.4. Configuring SAML Client Registration in the Application web.xml
5.5.5. Deploying the Application

CHAPTER 6. REFERENCE

6.1. ARTIFACT REPOSITORY MIRRORS
6.2. ENVIRONMENT VARIABLES
6.2.1. Information Environment Variables
6.2.2. Configuration Environment Variables
6.2.3. Template variables for all Red Hat Single Sign-On images
6.2.4. Template variables specific to sso-cd-mysql, sso-cd-mysql-persistent, and sso-cd-x509-mysql-persistent
6.2.5. Template variables specific to sso-cd-postgresql, sso-cd-postgresql-persistent, and sso-cd-x509-postgresql-persistent
6.2.6. Template variables specific to sso-cd-mysql-persistent, sso-cd-x509-mysql-persistent, sso-cd-postgresql-persistent, and sso-cd-x509-postgresql-persistent
6.2.7. Template variables for general eap64 and eap71 S2I images
6.2.8. Template variables specific to eap64-sso-s2i and eap71-sso-s2i for automatic client registration
6.2.9. Template variables specific to eap64-sso-s2i and eap71-sso-s2i for automatic client registration with SAML clients
6.3. EXPOSED PORTS
CHAPTER 1. INTRODUCTION

1.1. WHAT IS RED HAT SINGLE SIGN-ON?

Red Hat Single Sign-On is an integrated sign-on solution available as a Red Hat JBoss Middleware for OpenShift containerized image. The Red Hat Single Sign-On for OpenShift image provides an authentication server for users to centrally log in, log out, register, and manage user accounts for web applications, mobile applications, and RESTful web services.

Red Hat offers multiple OpenShift application templates utilizing the Red Hat Single Sign-On for OpenShift image version number 6. These define the resources needed to develop Red Hat Single Sign-On 6 server based deployment and can be split into the following two categories:

- **Templates using HTTPS and JGroups keystores and a truststore for the Red Hat Single Sign-On server, all prepared beforehand. These secure the TLS communication using passthrough TLS termination:**
  - **sso-cd-https**: Red Hat Single Sign-On 6 backed by internal H2 database on the same pod.
  - **sso-cd-mysql**: Red Hat Single Sign-On 6 backed by ephemeral MySQL database on a separate pod.
  - **sso-cd-mysql-persistent**: Red Hat Single Sign-On 6 backed by persistent MySQL database on a separate pod.
  - **sso-cd-postgresql**: Red Hat Single Sign-On 6 backed by ephemeral PostgreSQL database on a separate pod.
  - **sso-cd-postgresql-persistent**: Red Hat Single Sign-On 6 backed by persistent PostgreSQL database on a separate pod.

- **Templates using OpenShift’s internal service serving x509 certificate secrets to automatically create the HTTPS keystore used for serving secure content. The JGroups cluster traffic is authenticated using the AUTH protocol and encrypted using the ASYM_ENCRYPT protocol. The Red Hat Single Sign-On server truststore is also created automatically, containing the /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt CA certificate file, which is used to sign the certificate for HTTPS keystore. Moreover, the truststore for the Red Hat Single Sign-On server is pre-populated with the all known, trusted CA certificate files found in the Java system path. These templates secure the TLS communication using re-encryption TLS termination:**
  - **sso-cd-x509-https**: Red Hat Single Sign-On 6 with auto-generated HTTPS keystore and Red Hat Single Sign-On truststore, backed by internal H2 database. The ASYM_ENCRYPT JGroups protocol is used for encryption of cluster traffic.
  - **sso-cd-x509-mysql-persistent**: Red Hat Single Sign-On 6 with auto-generated HTTPS keystore and Red Hat Single Sign-On truststore, backed by persistent MySQL database. The ASYM_ENCRYPT JGroups protocol is used for encryption of cluster traffic.
  - **sso-cd-x509-postgresql-persistent**: Red Hat Single Sign-On 6 with auto-generated HTTPS keystore and Red Hat Single Sign-On truststore, backed by persistent PostgreSQL database. The ASYM_ENCRYPT JGroups protocol is used for encryption of cluster traffic.

Other templates that integrate with Red Hat Single Sign-On are also available:
The templates contain environment variables specific to Red Hat Single Sign-On that enable automatic Red Hat Single Sign-On client registration when deployed.

CHAPTER 2. BEFORE YOU BEGIN

2.1. COMPARISON: RED HAT SINGLE SIGN-ON FOR OPENS SHIFT IMAGE AND RED HAT SINGLE SIGN-ON

The Red Hat Single Sign-On for OpenShift image version number 6 is based on Red Hat Single Sign-On 6. There are some differences in functionality between the Red Hat Single Sign-On for OpenShift image and Red Hat Single Sign-On:

- The Red Hat Single Sign-On for OpenShift image includes all of the functionality of Red Hat Single Sign-On. In addition, the Red Hat Single Sign-On-enabled JBoss EAP image automatically handles OpenID Connect or SAML client registration and configuration for .war deployments that contain `<auth-method>KEYCLOAK</auth-method>` or `<auth-method>KEYCLOAK-SAML</auth-method>` in their respective `web.xml` files.

2.2. VERSION COMPATIBILITY AND SUPPORT

See the xPaaS part of the OpenShift and Atomic Platform Tested Integrations page for details about OpenShift image version compatibility.

2.3. DEPRECATED IMAGE STREAMS AND APPLICATION TEMPLATES FOR RED HAT SINGLE SIGN-ON FOR OPENS HIFT

**IMPORTANT**

The Red Hat Single Sign-On for OpenShift image version number 7.0 and 7.1 are deprecated and they will no longer receive updates of image and application templates.

To deploy new applications, it is recommended to use the version 7.2 or 6 of the Red Hat Single Sign-On for OpenShift image along with the application templates specific to these image versions.

2.4. INITIAL SETUP

The Tutorials in this guide follow on from and assume an OpenShift instance similar to that created in the OpenShift Primer.

**IMPORTANT**

For information related to updating the existing database when migrating Red Hat Single Sign-On for OpenShift image from previous versions to version 6, see the Updating Existing Database when Migrating Red Hat Single Sign-On for OpenShift Image to a new version section.
3.1. USING THE RED HAT SINGLE SIGN-ON FOR OPENSSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES

IMPORTANT

Red Hat JBoss Middleware for OpenShift images are pulled on demand from the secured Red Hat Registry: registry.redhat.io, which requires authentication. To retrieve content, you will need to log into the registry using the Red Hat account.

To consume container images from registry.redhat.io in shared environments such as OpenShift, it is recommended for an administrator to use a Registry Service Account, also referred to as authentication tokens, in place of an individual person’s Red Hat Customer Portal credentials.

To create a Registry Service Account, navigate to the Registry Service Account Management Application, and log in if necessary.

1. From the Registry Service Accounts page, click Create Service Account.

2. Provide a name for the Service Account, for example registry.redhat.io-sa. It will be prepended with a fixed, random string.
   a. Enter a description for the Service Account, for example Service account to consume container images from registry.redhat.io.
   b. Click Create.

3. After the Service Account was created, click the registry.redhat.io-sa link in the Account name column of the table presented on the Registry Service Accounts page.

4. Finally, click the OpenShift Secret tab, and perform all steps listed on that page.

See the Red Hat Container Registry Authentication article for more information.

To update to the latest Red Hat Single Sign-On for OpenShift images, run the following commands:

1. On your master host(s), ensure that you are logged in as a cluster administrator or a user with project administrator access to the global openshift project.

   $ oc login -u system:admin

2. Run the following commands to update the core set of Red Hat Single Sign-On 6 resources for OpenShift in the openshift project:

   $ for resource in sso-cd-image-stream.json \ 
     sso-cd-https.json \ 
     sso-cd-mysql.json \ 
     sso-cd-mysql-persistent.json \ 
     sso-cd-postgresql.json \ 
     sso-cd-postgresql-persistent.json \ 
     sso-cd-x509-https.json \ 

Run the following command to install the Red Hat Single Sign-On 6 OpenShift image streams in the openshift project:

```
$ oc -n openshift import-image redhat-sso-cd-openshift:6
```

### 3.2. PREPARING AND DEPLOYING THE RED HAT SINGLE SIGN-ON FOR OPENSHIFT APPLICATION TEMPLATES

#### 3.2.1. Configuring Keystores

The Red Hat Single Sign-On for OpenShift image requires two keystores:
- An SSL keystore to provide private and public keys for https traffic encryption.
- A JGroups keystore to provide private and public keys for network traffic encryption between nodes in the cluster.

These keystores are expected by the Red Hat Single Sign-On for OpenShift image, even if the application uses only http on a single-node OpenShift instance. Self-signed certificates do not provide secure communication and are intended for internal testing purposes.

![WARNING](https://via.placeholder.com/150)

**WARNING**

For production environments Red Hat recommends that you use your own SSL certificate purchased from a verified Certificate Authority (CA) for SSL-encrypted connections (HTTPS).

See the [JBoss Enterprise Application Platform Security Guide](https://example.com) for more information on how to create a keystore with self-signed or purchased SSL certificates.

#### 3.2.2. Generating Secrets

OpenShift uses objects called **Secrets** to hold sensitive information, such as passwords or keystores. See the [Secrets chapter](https://example.com) in the OpenShift documentation for more information.

The Red Hat Single Sign-On for OpenShift image requires one or more secrets that hold the two keystores described earlier. This provides the necessary authorization to applications in the project.

Use the SSL and JGroups keystore files to create secrets for the project:

```
$ oc secret new <sso-ssl-secret> <ssl.jks>
$ oc secret new <sso-jgroups-secret> <jgroups.jceks>
```
3.2.3. Using the OpenShift Web Console

Log in to the OpenShift web console:

1. Click Add to project to list the default image streams and templates.

2. Use the Filter by keyword search bar to limit the list to those that match sso. You may need to click See all to show the desired application template.

3. Select an application template and configure the deployment parameters as required.

4. Click Create to deploy the application template.

These are some of the more common variables to configure an Red Hat Single Sign-On deployment:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_NAME</td>
<td>The name for the Red Hat Single Sign-On application.</td>
</tr>
<tr>
<td>HOSTNAME_HTTPS</td>
<td>Custom hostname for https service route. Leave blank for default hostname of application-name. project.default-domain-suffix</td>
</tr>
<tr>
<td>HOSTNAME_HTTP</td>
<td>Custom hostname for http service route. Leave blank for default hostname of application-name. project.default-domain-suffix</td>
</tr>
<tr>
<td>HTTPS_KEYSTORE</td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td>HTTPS_PASSWORD</td>
<td>The password for the keystore and certificate.</td>
</tr>
<tr>
<td>HTTPS_SECRET</td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_KEYSTORE</td>
<td>The name of the JGroups keystore file within the secret.</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_PASSWORD</td>
<td>The password for the JGroups keystore and certificate.</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_SECRET</td>
<td>The name of the secret containing the JGroups keystore file.</td>
</tr>
<tr>
<td>SSO_ADMIN_USERNAME</td>
<td>Username of the administrator account for the master realm of the Red Hat Single Sign-On server. Required. If no value is specified, it is auto generated and displayed as an OpenShift instructional message when the template is instantiated.</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO_ADMIN_PASSWORD</td>
<td>Password of the administrator account for the master realm of the Red Hat Single Sign-On server. Required. If no value is specified, it is auto generated and displayed as an OpenShift instructional message when the template is instantiated.</td>
</tr>
<tr>
<td>SSO_REALM</td>
<td>The name of an additional Red Hat Single Sign-On realm to create during deployment.</td>
</tr>
<tr>
<td>SSO_SERVICE_USERNAME</td>
<td>Red Hat Single Sign-On service user name to manage the realm.</td>
</tr>
<tr>
<td>SSO_SERVICE_PASSWORD</td>
<td>Red Hat Single Sign-On service user password.</td>
</tr>
</tbody>
</table>

See the Reference chapter for a more comprehensive list of the Red Hat Single Sign-On environment variables. See the Example Workflow: Preparing and Deploying the Red Hat Single Sign-On for OpenShift Image for an end-to-end example of Red Hat Single Sign-On deployment.

### 3.2.4. Routes

The Red Hat Single Sign-On for OpenShift templates use TLS passthrough termination for routes by default. This means that the destination route receives encrypted traffic without the OpenShift router providing TLS termination. Users do not need the relevant SSL certificate to connect to the Red Hat Single Sign-On login page.

For more information on OpenShift route types, see the Networking chapter of the OpenShift Architecture Guide.

### 3.3. BINARY BUILDS

To deploy existing applications on OpenShift, you can use the binary source capability.

#### 3.3.1. Deploy Binary Build of EAP 6.4 / 7.1 JSP Service Invocation Application that Authenticates Using Red Hat Single Sign-On

The following example uses both app-jee-jsp and service-jee-jaxrs quickstarts to deploy EAP 6.4 / 7.1 JSP service application that authenticates using the Red Hat Single Sign-On.

**Prerequisite:**
IMPORTANT

This guide assumes the Red Hat Single Sign-On for OpenShift image has been previously deployed using one of the following templates:

- sso-cd-mysql
- sso-cd-postgresql
- sso-cd-mysql-persistent
- sso-cd-x509-mysql-persistent
- sso-cd-postgresql-persistent
- sso-cd-x509-postgresql-persistent

3.3.1.1. Create Red Hat Single Sign-On Realm, Roles, and User for the EAP 6.4 / 7.1 JSP Application

The EAP 6.4 / 7.1 JSP service application requires dedicated Red Hat Single Sign-On realm, username, and password to be able to authenticate using Red Hat Single Sign-On. Perform the following steps after the Red Hat Single Sign-On for OpenShift image has been deployed:

Create the Red Hat Single Sign-On Realm

1. Login to the administration console of the Red Hat Single Sign-On server.
   https://secure-sso-sso-app-demo.openshift.example.com/auth/admin
   Use the credentials of the Red Hat Single Sign-On administrator user.

2. Hover your cursor over the realm namespace (default is Master) at the top of the sidebar and click Add Realm.

3. Enter a realm name (this example uses demo) and click Create.

Copy the Public Key

In the newly created demo realm, click the Keys tab, then select Active tab, and copy the public key of type RSA that has been generated.

NOTE

The Red Hat Single Sign-On for OpenShift image version 6 generates multiple keys by default, for example HS256, RS256, or AES. To copy the public key information for the Red Hat Single Sign-On for OpenShift 6 image, click the Keys tab, then select Active tab, and click the Public key button of that row in the keys table, where type of the key matches RSA. Then select and copy the content of the pop-up window that appears.

The information about the public key is necessary later to deploy the Red Hat Single Sign-On-enabled EAP 6.4 / 7.1 JSP application.

Create Red Hat Single Sign-On Roles
NOTE

The service-jee-jaxrs quickstart exposes three endpoints by the service:

- **public** - Requires no authentication.
- **secured** - Can be invoked by users with the user role.
- **admin** - Can be invoked by users with the admin role.

Create **user** and **admin** roles in Red Hat Single Sign-On. These roles will be assigned to an Red Hat Single Sign-On application user to authenticate access to user applications.

1. Click **Roles** in the **Configure** sidebar to list the roles for this realm.

   NOTE

   This is a new realm, so there should only be the default (offline_access and uma_authorization) roles.

2. Click **Add Role**.

3. Enter the role name (**user**) and click **Save**.

Repeat these steps for the **admin** role.

Create the Red Hat Single Sign-On Realm Management User

1. Click **Users** in the **Manage** sidebar to view the user information for the realm.

2. Click **Add User**.

3. Enter a valid **Username** (this example uses the user **appuser**) and click **Save**.

4. Edit the user configuration:

   a. Click the **Credentials** tab in the user space and enter a password for the user (this example uses the password **apppassword**).

   b. Ensure the **Temporary Password** option is set to **Off** so that it does not prompt for a password change later on, and click **Reset Password** to set the user password. A pop-up window prompts for additional confirmation.

3.3.1.2. Assign user Red Hat Single Sign-On Role to the Realm Management User

Perform the following steps to tie the previously created **appuser** with the **user** Red Hat Single Sign-On role:

1. Click **Role Mappings** to list the realm and client role configuration. In **Available Roles**, select the **user** role created earlier, and click **Add selected**.

2. Click **Client Roles**, select **realm-management** entry from the list, select each record in the **Available Roles** list.
NOTE
You can select multiple items at once by holding the Ctrl key and simultaneously clicking the first impersonation entry. While keeping the Ctrl key and the left mouse button pressed, move to the end of the list to the view-clients entry and ensure each record is selected.

3. Click Add selected to assign the roles to the client.

3.3.1.3. Prepare Red Hat Single Sign-On Authentication for OpenShift Deployment of the EAP 6.4 / 7.1 JSP Application

1. Create a new project for the EAP 6.4 / 7.1 JSP application.

   $ oc new-project eap-app-demo

2. Add the view role to the default service account. This enables the service account to view all the resources in the eap-app-demo namespace, which is necessary for managing the cluster.

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

3. The EAP template requires an SSL keystore and a JGroups keystore. This example uses keytool, a package included with the Java Development Kit, to generate self-signed certificates for these keystores.

   a. Generate a secure key for the SSL keystore (this example uses password as password for the keystore).

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

   b. Generate a secure key for the JGroups keystore (this example uses password as password for the keystore).

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

   c. Generate the EAP 6.4 / 7.1 for OpenShift secrets with the SSL and JGroup keystore files.

      $ oc secret new eap-ssl-secret eapkeystore.jks

      $ oc secret new eap-jgroup-secret eapjgroups.jceks

   d. Add the EAP application secret to the default service account.

      $ oc secrets link default eap-ssl-secret eap-jgroup-secret
3.3.1.4. Deploy Binary Build of the EAP 6.4 / 7.1 JSP Application

1. Clone the source code.

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

2. Configure the Red Hat JBoss Middleware Maven repository.

3. Build both the `service-jee-jaxrs` and `app-jee-jsp` applications.
   a. Build the `service-jee-jaxrs` application.

   ```shell
   $ cd keycloak-quickstarts/service-jee-jaxrs/
   $ mvn clean package -DskipTests
   [INFO] Scanning for projects...
   [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INFO] [INF...
IMPORTANT

The **app-jee-jsp** quickstart requires to configure the adapter, and adapter configuration file (**keycloak.json**) to be present at the **config/** directory in the root of the quickstart to successfully build the quickstart. But since this example configures the adapter later via selected environment variables available for the EAP 6.4 / 7.1 for OpenShift image, it is not necessary to specify the form of **keycloak.json** adapter configuration file at this moment.

4. Prepare the directory structure on the local file system.

   Application archives in the **deployments/** subdirectory of the main binary build directory are copied directly to the **standard deployments directory** of the image being built on OpenShift. For the application to deploy, the directory hierarchy containing the web application data must be correctly structured.

   Create main directory for the binary build on the local file system and **deployments/** subdirectory within it. Copy the previously built WAR archives of both the **service-jee-jaxrs** and **app-jee-jsp** quickstarts to the **deployments/** subdirectory:

   ```bash
   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
   ```
NOTE

Location of the standard deployments directory depends on the underlying base image, that was used to deploy the application. See the following table:

<table>
<thead>
<tr>
<th>Name of the Underlying Base Image(s)</th>
<th>Standard Location of the Deployments Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAP for OpenShift 6.4 and 7.1</td>
<td>$JBoss_HOME/standalone/deployments</td>
</tr>
<tr>
<td>Java S2I for OpenShift</td>
<td>/deployments</td>
</tr>
<tr>
<td>JWS for OpenShift</td>
<td>$JWS_HOME/webapps</td>
</tr>
</tbody>
</table>

5. Identify the image stream for EAP 6.4 / 7.1 image.

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

6. Create new binary build, specifying image stream and application name.

```
$ 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"

-->

```bash
JBoss EAP 7.2
-------------
Platform for building and running JavaEE applications on JBoss EAP 7.2

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. Start the binary build. Instruct `oc` executable to use main directory of the binary build we created in previous step as the directory containing binary input for the OpenShift build. In the working directory of `app-jee-jsp` issue the following command.

```
app-jee-jsp$ oc start-build eap-app \
   --from-dir=./sso-eap7-bin-demo/ \n   --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...

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. Create a new OpenShift application based on the build.

```
$ oc new-app eap-app
```
9. Stop all running containers of the EAP 6.4 / 7.1 JSP application in the current namespace.

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

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

10. Further configure the EAP 6.4 / 7.1 JSP application prior the deployment.

a. Configure the application with proper details about the Red Hat Single Sign-On server instance.

WARNING

Ensure to replace the value of \texttt{SSO\_PUBLIC\_KEY} variable below with the actual content of the RSA public key for the \texttt{demo} realm, that has been copied.

\begin{verbatim}
$ 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" \n  -e SSO\_DISABLE\_SSL\_CERTIFICATE\_VALIDATION="true" \n  -e SSO\_USERNAME="appuser" \n  -e SSO\_PASSWORD="apppassword" \n  -e SSO\_REALM="demo" \n  -e SSO\_URL="https://secure-sso-sso-app-demo.openshift.com/auth" \n  -e SSO\_PUBLIC\_KEY="MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkdhXyKx97oIoO6HwnVMiX2EH055Sn+ydsPzbjEvl5F31uVc9uA8dGl6oM8HrnaWWy+i8PvmlARMnhI6x68vJTEc6dOs0P+6A+aExw0coNRp2PDwzszXVWPvPQg3+iytStxu3Icndx+gCOZ\\nxoRqL7rY7zKoQQScGER78Nw6vZDwfe6d/PQ6W4xVEnNytX9KyLFVAE1VvXALygEM/EqYGLmpjw5bMGVKRXnmVo9E88CkFDH8E+aPiApb/gFuI1GJQv+G8sL0R1c8Y3L29F7C81odkBp2yMm3RVFIcG5PTJHqJo/nOtgYlfY4Wyw9mRloY5SyW7O44dXZWlDAQAB" \
  -e SSO\_SECRET="0bb8c399-2501-4fcf-a183-68ac5132868d" 
deploymentconfig "eap-app" updated
\end{verbatim}

b. Configure the application with details about both the SSL and JGroups keystore.

$ 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"
c. Define OpenShift volumes for both the SSL and JGroups secrets created earlier.

```
$ 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

11. Deploy container of the EAP 6.4 / 7.1 JSP application using the modified deployment config.

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

deploymentconfig "eap-app" scaled

12. Expose the service as route.

```
$ 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
```

3.3.1.5. Access the Application
Access the application in your browser using the URL http://eap-app-eap-app-demo.openshift.example.com/app-jsp. You should see output like on the following image:

Perform the following to test the application:

- Click the INVOKE PUBLIC button to access the public endpoint that doesn’t require authentication.
  You should see the Message: public output.

- Click the LOGIN button to be redirected for user authentication to the Red Hat Single Sign-On server instance against the demo realm.
  Specify username and password of the Red Hat Single Sign-On user configured earlier (appuser/apppassword). Click Log in. The look of the application changes as detailed in the following image:

- Click the INVOKE SECURED button to access the secured endpoint.
  You should see the Message: secured output.

- Click the INVOKE ADMIN button to access the admin endpoint.
  You should see 403 Forbidden output.

**NOTE**

The admin endpoint requires users with admin Red Hat Single Sign-On role to invoke properly. Access for the appuser is forbidden because they only have user role privilege, which allows them to access the secured endpoint.
Perform the following steps to add the **appuser** to the **admin** Red Hat Single Sign-On role:

1. Access the administration console of the Red Hat Single Sign-On server’s instance.

   Use the **credentials of the Red Hat Single Sign-On administrator user**.

2. Click **Users** in the **Manage** sidebar to view the user information for the **demo** realm.

3. Click **View all users** button.

4. Click the ID link for the **appuser** or alternatively click the **Edit** button in the **Actions** column.

5. Click the **Role Mappings** tab.

6. Select **admin** entry from the **Available Roles** list in the **Realm Roles** row.

7. Click **Add selected** button to add the **admin** role to the user.

8. Return to EAP 6.4 / 7.1 JSP service application.

9. Click the **LOGOUT** button to reload role mappings for the **appuser**.

10. Click the **LOGIN** button again and provider **appuser** credentials.

11. Click the **INVOKE ADMIN** button again.

   You should see the **Message: admin** output already.
CHAPTER 4. ADVANCED CONCEPTS

These cover additional configuration topics, such as setting up keystores and a truststore for the Red Hat Single Sign-On server, creating an administrator account, an overview of available Red Hat Single Sign-On client registration methods, and guidance on configuring clustering.

4.1. REQUIREMENTS AND DEPLOYING PASSTHROUGH TLS TERMINATION RED HAT SINGLE SIGN-ON TEMPLATES

4.1.1. Preparing the Deployment

Log in to the OpenShift CLI with a user that holds the `cluster:admin` role.

1. Create a new project:

   $ oc new-project sso-app-demo

2. Add the `view` role to the `default` service account. This enables the service account to view all the resources in the sso-app-demo namespace, which is necessary for managing the cluster.

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

4.1.2. Creating HTTPS and JGroups Keystores, and Truststore for the Red Hat Single Sign-On Server

The Red Hat Single Sign-On application templates using passthrough TLS termination require:

- An HTTPS keystore used for encryption of https traffic,
- The JGroups keystore used for encryption of JGroups communications between nodes in the cluster, and
- Red Hat Single Sign-On server truststore used for securing the Red Hat Single Sign-On requests

the Red Hat Single Sign-On for OpenShift image to be deployed properly.

IMPORTANT

The Red Hat Single Sign-On application templates using re-encryption TLS termination do not require or expect the aforementioned HTTPS and JGroups keystores and Red Hat Single Sign-On server truststore to be prepared beforehand. The templates use OpenShift’s internal service serving x509 certificate secrets to automatically create the HTTPS and JGroups keystores. The Red Hat Single Sign-On server truststore is also created automatically, containing the `/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt` CA certificate file, which is used to create these cluster certificates. Moreover, the truststore for the Red Hat Single Sign-On server is pre-populated with the all known, trusted CA certificate files found in the Java system path.

The `openssl` toolkit is used in the following example to generate a CA certificate to sign the HTTPS keystore, and create a truststore for the Red Hat Single Sign-On server. `keytool`, a package included
with the Java Development Kit, is then utilized to generate self-signed certificates for these keystores.

**WARNING**

For production environments Red Hat recommends that you use your own SSL certificate purchased from a verified Certificate Authority (CA) for SSL-encrypted connections (HTTPS).

See the [JBoss Enterprise Application Platform Security Guide](#) for more information on how to create a keystore with self-signed or purchased SSL certificates.

---

**Create the HTTPS keystore:**

a. Generate a CA certificate. Pick and remember the password. Provide identical password, when signing the certificate sign request with the CA certificate below:

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

b. Generate a CA certificate for the HTTPS keystore. Provide `mykeystorepass` as the keystore password:

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

c. Generate a certificate sign request for the HTTPS keystore. Provide `mykeystorepass` as the keystore password:

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

d. Sign the certificate sign request with the CA certificate. Provide the same password that was used to generate the CA certificate:

```
$ openssl x509 -req -CA xpaas.crt -CAkey xpaas.key -in sso.csr -out sso.crt -days 365 -CAcreateserial
```

e. Import the CA certificate into the HTTPS keystore. Provide `mykeystorepass` as the keystore password. Reply `yes` to **Trust this certificate? [no]:** question:

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

f. Import the signed certificate sign request into the HTTPS keystore. Provide `mykeystorepass` as the keystore password:

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

---

**Generate a secure key for the JGroups keystore:**
Provide **password** as the keystore password:

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

**Import the CA certificate into a new Red Hat Single Sign-On server truststore:**

Provide **mykeystorepass** as the truststore password. Reply **yes** to Trust this certificate? [no]:

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

### 4.1.3. Secrets

OpenShift uses objects called **secrets** to hold sensitive information, such as passwords or keystores.

1. Create the secrets for the HTTPS and JGroups keystores, and Red Hat Single Sign-On server truststore, generated in the previous section.

```
$ oc secret new sso-app-secret keystore.jks jgroups.jceks truststore.jks
```

2. Link these secrets to the **default** service account, which is used to run Red Hat Single Sign-On pods.

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

### 4.1.4. Deploying the Chosen Red Hat Single Sign-On Passthrough TLS Template via OpenShift CLI

After the aforementioned **keystores** and **secrets** are created, deploy some of the available **passthrough TLS termination** as follows:

---

**WARNING**

For simplicity, the values of **SSO_ADMIN_USERNAME, SSO_ADMIN_PASSWORD, HTTPS_PASSWORD, JGROUPS_ENCRYPT_PASSWORD**, and **SSO_TRUSTSTORE_PASSWORD** variables in the following command have been chosen to match the default values of the respective parameters of the sso-**cd-https** Red Hat Single Sign-On application template.

For production environments, Red Hat recommends that you consult the on-site policy, specific to your organization for guidance on how to generate sufficiently strong user name and password for the administrator user account of the Red Hat Single Sign-On server, and passwords for the HTTPS and JGroups keystores, and the truststore of the Red Hat Single Sign-On server.

Be aware that **the passwords provided when provisioning the template need to match the passwords provided when creating the keystores**. If using different username and passwords, modify the values of respective **template parameters** as appropriate for your environment.
NOTE

The following commands using the `keytool`, a package included with the Java Development Kit, can be used to determine the names associated with the certificate:

```bash
$ 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
```

Finally, the `SSO_ADMIN_USERNAME`, `SSO_ADMIN_PASSWORD`, and the `SSO_REALM` template parameters in the following command are optional.

```bash
$ oc new-app --template=sso-cd-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/sso-cd-https" to project sso-app-demo
```

Red Hat Single Sign-On 6 (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
After the template got deployed, identify the available routes:

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
<th>SERVICES</th>
<th>PORT</th>
<th>TERMINATION</th>
<th>WILDCARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>secure-sso</td>
<td>secure-sso-</td>
<td>sso-app-</td>
<td>secure-sso</td>
<td>&lt;all&gt;</td>
<td>passthrough</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>sso-app-demo.opens</td>
<td>hift.example.com</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sso</td>
<td>sso-sso-app-</td>
<td>demo.opens</td>
<td>sso</td>
<td>&lt;all&gt;</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>sso-app-demo.opens</td>
<td>hift.example.com</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and access the Red Hat Single Sign-On administrator console at:
using the administrator account.

4.2. CUSTOMIZING HOSTNAME FOR THE RED HAT SINGLE SIGN-ON SERVER

The hostname SPI introduced a flexible way to configure the hostname for the Red Hat Single Sign-On server. There are two built-in providers. The first is request, which uses the request headers to determine the hostname. This is the default setting for Red Hat Single Sign-On for OpenShift image. The second is fixed, which allows configuring a fixed hostname. The latter makes sure that only valid hostnames can be used and allows internal applications to invoke Red Hat Single Sign-On server through an alternative URL.

Run the following commands to set the fixed hostname SPI provider for the Red Hat Single Sign-On server:


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

2. Identify the name of the route for the Red Hat Single Sign-On service.

   ```
   $ oc get routes
   ```

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
<th>SERVICES</th>
<th>PORT</th>
<th>TERMINATION</th>
<th>WILDCARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>sso</td>
<td>sso-sso-app-demo.openshift.example.com</td>
<td>sso</td>
<td>&lt;all&gt;</td>
<td>reencrypt</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

3. Change the host: field to match the hostname specified as the value of the SSO_HOSTNAME environment variable above.

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

   If successful, the previous command will return the following output:
4.3. CREATING THE ADMINISTRATOR ACCOUNT FOR RED HAT SINGLE SIGN-ON SERVER

Red Hat Single Sign-On does not provide any pre-configured management account out of the box. This administrator account is necessary for logging into the master realm’s management console and perform server maintenance operations such as, creating realms or users, or registering applications intended to be secured by Red Hat Single Sign-On.

The administrator account can be created:

- By providing values for the `SSO_ADMIN_USERNAME` and `SSO_ADMIN_PASSWORD` parameters, when deploying the Red Hat Single Sign-On application template, or
- By a remote shell session to particular Red Hat Single Sign-On pod, if the Red Hat Single Sign-On for OpenShift image is deployed without an application template.

**NOTE**

Red Hat Single Sign-On allows an initial administrator account to be created via the Welcome Page web form, but only if the Welcome Page is accessed from localhost; this method of administrator account creation is not applicable for the Red Hat Single Sign-On for OpenShift image.

4.3.1. Creating the Administrator Account Using Template Parameters

When deploying Red Hat Single Sign-On application template, the `SSO_ADMIN_USERNAME` and `SSO_ADMIN_PASSWORD` parameters denote the username and password of the Red Hat Single Sign-On server’s administrator account to be created for the master realm.

**NOTE**

*Both of these parameters are required.* If not specified, they are auto generated and displayed as an OpenShift instructional message when the template is instantiated.
IMPORTANT

The lifespan of the Red Hat Single Sign-On server’s administrator account depends upon the storage type used to store the Red Hat Single Sign-On server’s database:

- For an in-memory database mode (sso-cd-https and sso-cd-x509-https templates) the account exists throughout the lifecycle of the particular Red Hat Single Sign-On pod (stored account data is lost upon pod destruction),

- For an ephemeral database mode (sso-cd-mysql and sso-cd-postgresql templates) the account exists throughout the lifecycle of the database pod (even if the Red Hat Single Sign-On pod is destructed, the stored account data is preserved under the assumption that the database pod is still running),

- For persistent database mode (sso-cd-mysql-persistent, sso-cd-x509-mysql-persistent, sso-cd-postgresql-persistent, and sso-cd-x509-postgresql-persistent templates) the account exists throughout the lifecycle of the persistent medium used to hold the database data. This means that the stored account data is preserved even when both the Red Hat Single Sign-On and the database pods are destructed.

It is a common practice to deploy an Red Hat Single Sign-On application template to get the corresponding OpenShift deployment config for the application, and then reuse that deployment config multiple times (every time a new Red Hat Single Sign-On application needs to be instantiated).

WARNING

In the case of ephemeral or persistent database mode, after creating the RH_SSO server’s administrator account, remove the SSO_ADMIN_USERNAME and SSO_ADMIN_PASSWORD variables from the deployment config before deploying new Red Hat Single Sign-On applications.

IMPORTANT

Run the following commands to prepare the previously created deployment config of the Red Hat Single Sign-On application for reuse after the administrator account has been created:

1. Identify the deployment config of the Red Hat Single Sign-On application.

   $ oc get dc -o name
   deploymentconfig/sso
   deploymentconfig/sso-mysql

2. Clear the SSO_ADMIN_USERNAME and SSO_ADMIN_PASSWORD variables setting.

   $ oc set env dc/sso
   -e SSO_ADMIN_USERNAME="" \
   -e SSO_ADMIN_PASSWORD=""
4.3.2. Creating the Administrator Account via Remote Shell Session to Red Hat Single Sign-On Pod

Run following commands to create an administrator account for the *master* realm of the Red Hat Single Sign-On server, when deploying the Red Hat Single Sign-On for OpenShift image directly from the image stream (without the template), after the Red Hat Single Sign-On application pod has been started:

1. Identify the Red Hat Single Sign-On application pod.

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

2. Open a remote shell session to the Red Hat Single Sign-On for OpenShift container.

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

3. Create the Red Hat Single Sign-On server administrator account for the *master* realm at the command line with the `add-user-keycloak.sh` script.

   ```
   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
   ```

   **NOTE**

   The 'sso_admin' / 'sso_password' credentials in the example above are for demonstration purposes only. Refer to the password policy applicable within your organization for guidance on how to create a secure user name and password.

4. Restart the underlying JBoss EAP server instance to load the newly added user account. Wait for the server to restart properly.

   ```
   sh-4.2$ ./jboss-cli.sh --connect ':reload'
   {
     "outcome" => "success",
     "result" => undefined
   }
   ```
5. Log in to the master realm’s administration console of the Red Hat Single Sign-On server using the credentials created in the steps above. In the browser, navigate to 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, and specify the user name and password used to create the administrator user.

4.4. DEPLOYMENT PROCESS

Once deployed, the sso-cd-https and sso-cd-x509-https templates create a single pod that contains both the database and the Red Hat Single Sign-On servers. The sso-cd-mysql, sso-cd-mysql-persistent, sso-cd-x509-mysql-persistent, sso-cd-postgresql, sso-cd-postgresql-persistent, and sso-cd-x509-postgresql-persistent templates create two pods, one for the database server and one for the Red Hat Single Sign-On web server.

After the Red Hat Single Sign-On web server pod has started, it can be accessed at its custom configured hostnames, or at the default hostnames:

- http://sso-<project-name>.<hostname>/auth/admin: for the Red Hat Single Sign-On web server, and

Use the administrator user credentials to log in into the master realm’s administration console.

4.5. RED HAT SINGLE SIGN-ON CLIENTS

Clients are Red Hat Single Sign-On entities that request user authentication. A client can be an application requesting Red Hat Single Sign-On to provide user authentication, or it can be making requests for access tokens to start services on behalf of an authenticated user. See the Managing Clients chapter of the Red Hat Single Sign-On documentation for more information.

Red Hat Single Sign-On provides OpenID-Connect and SAML client protocols. OpenID-Connect is the preferred protocol and utilizes three different access types:

- **public**: Useful for JavaScript applications that run directly in the browser and require no server configuration.
- **confidential**: Useful for server-side clients, such as EAP web applications, that need to perform a browser login.
- **bearer-only**: Useful for back-end services that allow bearer token requests.
It is required to specify the client type in the `<auth-method>` key of the application `web.xml` file. This file is read by the image at deployment. Set the value of `<auth-method>` element to:

- **KEYCLOAK** for the OpenID Connect client.
- **KEYCLOAK-SAML** for the SAML client.

The following is an example snippet for the application `web.xml` to configure an OIDC client:

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

4.5.1. Automatic and Manual Red Hat Single Sign-On Client Registration Methods

A client application can be automatically registered to an Red Hat Single Sign-On realm by using credentials passed in variables specific to the `eap64-sso-s2i`, `eap71-sso-s2i`, and `datavirt63-secure-s2i` templates.

Alternatively, you can manually register the client application by configuring and exporting the Red Hat Single Sign-On client adapter and including it in the client application configuration.

4.5.1.1. Automatic Red Hat Single Sign-On Client Registration

Automatic Red Hat Single Sign-On client registration is determined by Red Hat Single Sign-On environment variables specific to the `eap64-sso-s2i`, `eap71-sso-s2i`, and `datavirt63-secure-s2i` templates. The Red Hat Single Sign-On credentials supplied in the template are then used to register the client to the Red Hat Single Sign-On realm during deployment of the client application.

The Red Hat Single Sign-On environment variables included in the `eap64-sso-s2i`, `eap71-sso-s2i`, and `datavirt63-secure-s2i` templates are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>HOSTNAME_HTTP</code></td>
<td>Custom hostname for http service route. Leave blank for default hostname of <code>&lt;application-name&gt;</code>. <code>&lt;project&gt;</code>.<code>&lt;default-domain-suffix&gt;</code></td>
</tr>
<tr>
<td><code>HOSTNAME_HTTPS</code></td>
<td>Custom hostname for https service route. Leave blank for default hostname of <code>&lt;application-name&gt;</code>. <code>&lt;project&gt;</code>.<code>&lt;default-domain-suffix&gt;</code></td>
</tr>
<tr>
<td><code>SSO_URL</code></td>
<td>The Red Hat Single Sign-On web server authentication address: <code>https://secure-sso-&lt;project-name&gt;</code>./&lt;hostname&gt;/auth</td>
</tr>
<tr>
<td><code>SSO_REALM</code></td>
<td>The Red Hat Single Sign-On realm created for this procedure.</td>
</tr>
<tr>
<td><code>SSO_USERNAME</code></td>
<td>The name of the realm management user.</td>
</tr>
</tbody>
</table>
If the Red Hat Single Sign-On client uses the SAML protocol, the following additional variables need to be configured:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO_SAML_KEYSTORE_SECRET</td>
<td>Secret to use for access to SAML keystore. The default is sso-app-secret.</td>
</tr>
<tr>
<td>SSO_SAML_KEYSTORE</td>
<td>Keystore filename in the SAML keystore secret. The default is keystore.jks.</td>
</tr>
<tr>
<td>SSO_SAML_KEYSTORE_PASSWORD</td>
<td>Keystore password for SAML. The default is mykeystorepass.</td>
</tr>
<tr>
<td>SSO_SAML_CERTIFICATE_NAME</td>
<td>Alias for keys/certificate to use for SAML. The default is jboss.</td>
</tr>
</tbody>
</table>

See Example Workflow: Automatically Registering EAP Application in Red Hat Single Sign-On with OpenID-Connect Client for an end-to-end example of the automatic client registration method using an OpenID-Connect client.

### 4.5.1.2. Manual Red Hat Single Sign-On Client Registration

Manual Red Hat Single Sign-On client registration is determined by the presence of a deployment file in the client application’s ../configuration/ directory. These files are exported from the client adapter in the Red Hat Single Sign-On web console. The name of this file is different for OpenID-Connect and SAML clients:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenID-Connect</td>
<td>../configuration/secure-deployments</td>
</tr>
<tr>
<td>SAML</td>
<td>../configuration/secure-saml-deployments</td>
</tr>
</tbody>
</table>
These files are copied to the Red Hat Single Sign-On adapter configuration section in the standalone-openshift.xml at when the application is deployed.

There are two methods for passing the Red Hat Single Sign-On adapter configuration to the client application:

- Modify the deployment file to contain the Red Hat Single Sign-On adapter configuration so that it is included in the standalone-openshift.xml file at deployment, or
- Manually include the OpenID-Connect keycloak.json file, or the SAML keycloak-saml.xml file in the client application’s ../WEB-INF directory.


4.6. LIMITATIONS

OpenShift does not currently accept OpenShift role mapping from external providers. If Red Hat Single Sign-On is used as an authentication gateway for OpenShift, users created in Red Hat Single Sign-On must have the roles added using the OpenShift Administrator oc adm policy command.

For example, to allow an Red Hat Single Sign-On-created user to view a project namespace in OpenShift:

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

[[Red Hat Single Sign-On-Binary-Builds-Tutorial]] === Example Workflow: Creating OpenShift Application from Existing Maven Binaries and Securing it Using Red Hat Single Sing-On
CHAPTER 5. TUTORIALS

5.1. EXAMPLE WORKFLOW: PREPARING AND DEPLOYING THE RED HAT SINGLE SIGN-ON FOR OPENSIFT IMAGE

5.1.1. Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment

Log in to the OpenShift CLI with a user that holds the `cluster:admin` role.

To deploy existing applications on OpenShift, you can use the binary source capability.

5.1.2. Deploy Binary Build of EAP 6.4 / 7.1 JSP Service Invocation Application and Secure it Using Red Hat Single Sign-On

The following example uses both `app-jee-jsp` and `service-jee-jaxrs` quickstarts to deploy EAP 6.4 / 7.1 JSP service application that authenticates using the Red Hat Single Sign-On.

Prerequisite:

**IMPORTANT**

This guide assumes the Red Hat Single Sign-On for OpenShift image has been previously deployed using one of the following templates:

- `sso-cd-mysql`
- `sso-cd-mysql-persistent`
- `sso-cd-postgresql`
- `sso-cd-postgresql-persistent`
- `sso-cd-x509-mysql-persistent`
- `sso-cd-x509-postgresql-persistent`

5.1.2.1. Create Red Hat Single Sign-On Realm, Roles, and User for the EAP 6.4 / 7.1 JSP Application

The EAP 6.4 / 7.1 JSP service application requires dedicated Red Hat Single Sign-On realm, username, and password to be able to authenticate using Red Hat Single Sign-On. Perform the following steps after the Red Hat Single Sign-On for OpenShift image has been deployed:

Create the Red Hat Single Sign-On Realm

1. Login to the administration console of the Red Hat Single Sign-On server.
   
   https://secure-sso-sso-app-demo.openshift.example.com/auth/admin

   Use the credentials of the Red Hat Single Sign-On administrator user.

2. Hover your cursor over the realm namespace (default is `Master`) at the top of the sidebar and click Add Realm.

3. Enter a realm name (this example uses `demo`) and click Create.
Example Workflow: Updating Existing Database when Migrating Red Hat Single Sign-On for OpenShift Image to a new version

**IMPORTANT**

- Rolling updates from previous versions of Red Hat Single Sign-On for OpenShift to version 6 are not supported as databases and caches are not backward compatible.

- Stop all instances running some of previous versions of the Red Hat Single Sign-On for OpenShift before upgrading. They cannot run concurrently against the same database.

- Pre-generated scripts are not available, they are generated dynamically depending on the database.

Red Hat Single Sign-On 6 can automatically migrate the database schema, or you can choose to do it manually.

**NOTE**

By default the database is automatically migrated when you start Red Hat Single Sign-On 6 for the first time.

5.1.3. Automatic Database Migration

This process assumes that you are running some previous version of the Red Hat Single Sign-On for OpenShift image, backed by MySQL or PostgreSQL database (deployed in ephemeral or persistent mode), running on a separate pod.

**IMPORTANT**

Stop all pods running the previous version of the Red Hat Single Sign-On for OpenShift image before upgrading to Red Hat Single Sign-On 6, as they cannot run concurrently against the same database.

Use the following steps to automatically migrate the database schema:

1. Identify the deployment config used to deploy the containers, running previous version of the Red Hat Single Sign-On for OpenShift image.

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

2. Stop all pods running the previous version of the Red Hat Single Sign-On for OpenShift image in the current namespace.

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

3. Update the image change trigger in the existing deployment config to reference the Red Hat Single Sign-On 6 image.
Start rollout of the new Red Hat Single Sign-On 6 images based on the latest image defined in the image change triggers.

$ oc patch dc/sso --type=json -p '[["op": "replace", "path": "/spec/triggers/0/imageChangeParams/from/name", "value": "redhat-sso-cd-openshift:6"]]' "sso" patched

4. Start rollout of the new Red Hat Single Sign-On 6 images based on the latest image defined in the image change triggers.

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


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

6. (Optional) Verify the database has been successfully updated.

$ 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

5.1.4. Manual Database Migration
IMPORTANT

Pre-generated scripts are not available. They are generated dynamically depending on the database. With Red Hat Single Sign-On 6 one can generate and export these to an SQL file that can be manually applied to the database afterwards. To dynamically generate the SQL migration file for the database:

1. Configure Red Hat Single Sign-On with the correct datasource,

2. Set following configuration options in the `standalone-openshift.xml` file:
   a. `initializeEmpty=false`,
   b. `migrationStrategy=manual`, and
   c. `migrationExport` to the location on the file system of the pod, where the output SQL migration file should be stored (e.g. `migrationExport="${jboss.home.dir}/keycloak-database-update.sql"`).

See [database configuration of Red Hat Single Sign-On 6](#) for further details.

The database migration process handles the data schema update and performs manipulation of the data, therefore, stop all pods running the previous version of the Red Hat Single Sign-On for OpenShift image before dynamic generation of the SQL migration file.

This process assumes that you are running some previous version of the Red Hat Single Sign-On for OpenShift image, backed by MySQL or PostgreSQL database (deployed in ephemeral or persistent mode), running on a separate pod.

Perform the following to generate and get the SQL migration file for the database:

1. Prepare template of OpenShift database migration job to generate the SQL file.

```bash
$ cat job-to-migrate-db-to-sso-cd.yaml.orig
apiVersion: batch/v1
kind: Job
metadata:
  name: job-to-migrate-db-to-sso-cd
spec:
  autoSelector: true
  parallelism: 0
  completions: 1
  template:
    metadata:
      name: job-to-migrate-db-to-sso-cd
    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>>"
```

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-sso-cd
# 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-sso-cd.yaml.orig \
  job-to-migrate-db-to-sso-cd.yaml

2. From deployment config used to run the previous version of the Red Hat Single Sign-On for
OpenShift image, copy the datasource definition and database access credentials to
appropriate places of the template of the database migration job.

Use the following script to copy DB_SERVICE_PREFIX_MAPPING and
TX_DATABASE_PREFIX_MAPPING variable values, together with values of environment
variables specific to particular datasource (<PREFIX>_JNDI, <PREFIX>_USERNAME,
<PREFIX>_PASSWORD, and <PREFIX>_DATABASE) from the deployment config named
sso to the database job migration template named job-to-migrate-db-to-sso-cd.yaml.

NOTE

Although the DB_SERVICE_PREFIX_MAPPING environment variable allows a
comma-separated list of <name><database_type>=<PREFIX> triplets as its
value, this example script accepts only one datasource triplet definition for
demonstration purposes. You can modify the script for handling multiple
datasource definition triplets.

$ 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-sso-cd.yaml"

# Get existing variables of the $SSO_DC_NAME deployment config
# in an array
declare -a SSO_DC_VARS=(

$(oc set env do/${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
# yam 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
# yam 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
        "Successfully updated ${KEY%_VALUE} to:"
        "$VALUE"
Run the script.

```bash
$ 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. Build the Red Hat Single Sign-On 6 database migration image using the pre-configured source and wait for the build to finish.

```bash
$ oc get is -n openshift | grep sso-cd | cut -d ' ' -f 1
redhat-sso-cd-openshift

--context-dir=sso-manual-db-migration
--name=sso-cd-db-migration-image
--> Found image bf45ac2 (7 days old) in image stream "openshift/redhat-sso-cd-openshift" under tag "6" for "redhat-sso-cd-openshift:6"

Red Hat SSO 6
--------------
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 "sso-cd-db-migration-image:latest"
* Use `start-build` to trigger a new build

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

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

4. Update the template of the database migration job (job-to-migrate-db-to-sso-cd.yaml) with reference to the built sso-cd-db-migration-image image.

a. Get the docker pull reference for the image.
1. Get the docker pull reference for the image.

2. Replace the <<SSO_IMAGE_VALUE>> field in the job template with the pull specification.

3. Verify that the field is updated.

4. Instantiate database migration job from the job template.

   ```bash
   $ PULL_REF=$(oc get istag -n $OC_PROJECT -q --no-headers | grep sso-cd-db-migration-image | tr -s '' cut -d '' -f 2)
   $ sed -i "s#<<SSO_IMAGE_VALUE>>#$PULL_REF#g" job-to-migrate-db-to-sso-cd.yaml
   $ oc create -f job-to-migrate-db-to-sso-cd.yaml
   job "job-to-migrate-db-to-sso-cd" created
   
   IMPORTANT
   The database migration process handles the data schema update and performs manipulation of the data, therefore, stop all pods running the previous version of the Red Hat Single Sign-On for OpenShift image before dynamic generation of the SQL migration file.

5. Identify the deployment config used to deploy the containers, running previous version of the Red Hat Single Sign-On for OpenShift image.

   ```bash
   $ oc get dc -o name --selector=application=sso deploymentconfig/ssso
   deploymentconfig/ssso-postgresql
   
   $ oc scale --replicas=0 dc/ssso
   deploymentconfig "ssso" scaled
   
   $ oc get jobs
   NAME                   DESIRED SUCCESSFUL AGE
   job-to-migrate-db-to-sso-cd 1 0 3m
   
   $ oc scale --replicas=1 job/job-to-migrate-db-to-sso-cd
   job "job-to-migrate-db-to-sso-cd" scaled
   
   $ oc get pods
   NAME                        READY STATUS     RESTARTS AGE
   sso-postgresql-1-n5p16     1/1  Running 1 19h
   job-to-migrate-db-to-sso-cd-b87bb 1/1  Running 0 1m
   sso-cd-db-migration-image-1-build 0/1  Completed 0 27m
   ```
NOTE

By default, the database migration job terminates automatically after 600 seconds after the migration file is generated. You can adjust this time period.

9. Get the dynamically generated SQL database migration file from the pod.

```
$ mkdir -p ./db-update
$ oc rsync job-to-migrate-db-to-sso-cd-b87bb:/opt/eap/keycloak-database-update.sql ./db-update
```

10. Inspect the `keycloak-database-update.sql` file for changes to be performed within manual database update to Red Hat Single Sign-On 6 version.

11. Apply the database update manually.

   - Run the following commands if running some previous version of the Red Hat Single Sign-On for OpenShift image, backed by the PostgreSQL database deployed in ephemeral or persistent mode, running on a separate pod:

     i. Copy the generated SQL migration file to the 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. Start a shell session to the PostgreSQL pod.

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

     iii. Use the `psql` tool to apply database update manually.

        ```
        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
        ...
iv. Close the shell session to the PostgreSQL pod. Continue with updating image change trigger step.

- Run the following commands if running some previous version of the Red Hat Single Sign-On for OpenShift image, backed by the MySQL database deployed in ephemeral or persistent mode, running on a separate pod:

  i. Given the pod situation similar to the following:

```
$ oc get pods
NAME                                  READY     STATUS      RESTARTS   AGE
sso-mysql-1-zvhk3                     1/1       Running     0          1h
job-to-migrate-db-to-sso-cd-m202t   1/1       Running     0          11m
sso-cd-db-migration-image-1-build      0/1       Completed   0          13m
```

  ii. Copy the generated SQL migration file to the MySQL pod.

```
$ oc rsync --no-perms=true ./db-update/ sso-mysql-1-zvhk3:/tmp
sending incremental file list
done 1 files
keycloak-database-update.sql
sent 24,718 bytes  received 34 bytes  49,504.00 bytes/sec
total size is 24,594  speedup is 0.99
```

  iii. Start a shell session to the MySQL pod.

```
$ oc rsh sso-mysql-1-zvhk3
sh-4.2$
```

  iv. Use the `mysql` tool to apply database update manually.

```
sh-4.2$ alias mysql="/opt/rh/rh-mysql57/root/bin/mysql"
sh-4.2$ mysql --version
/opt/rh/rh-mysql57/root/bin/mysql Ver 14.14 Distrib 5.7.16, for Linux (x86_64) using EditLine wrapper
sh-4.2$ mysql -D <PREFIX>_DATABASE -u <PREFIX>_USERNAME -p <
/tmp/keycloak-database-update.sql
Enter password:
sh-4.2$ echo $? 
0
```

**IMPORTANT**

Replace `<PREFIX>_USERNAME` and `<PREFIX>_DATABASE` with the actual database credentials retrieved in previous section. Also use value of `<PREFIX>_PASSWORD` as the password for the database, when prompted.
v. Close the shell session to the MySQL pod. Continue with updating image change trigger step.

12. Update the image change trigger in the existing deployment config to reference the Red Hat Single Sign-On 6 image.

   ```
   $ oc patch dc/sso --type=json -p '[{"op": "replace", "path": "/spec/triggers/0/imageChangeParams/from/name", "value": "redhat-sso-cd-openshift:6"}]'
   "sso" patched
   ```

13. Start rollout of the new Red Hat Single Sign-On 6 images based on the latest image defined in the image change triggers.

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


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

5.2. EXAMPLE WORKFLOW: MIGRATING ENTIRE RED HAT SINGLE SIGN-ON SERVER DATABASE ACROSS THE ENVIRONMENTS

This tutorial focuses on migrating the Red Hat Single Sign-On server database from one environment to another or migrating to a different database. It assumes steps described in Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment section have been performed already.

5.2.1. Deploying the Red Hat Single Sign-On MySQL Application Template

1. Log in to the OpenShift web console and select the sso-app-demo project space.

2. Click Add to project to list the default image streams and templates.

3. Use the Filter by keyword search bar to limit the list to those that match sso. You may need to click See all to show the desired application template.

4. Select sso-cd-mysql Red Hat Single Sign-On application template. When deploying the template ensure to keep the SSO_REALM variable unset (default value).

**IMPORTANT**

Export and import of Red Hat Single Sign-On 6 database is triggered at Red Hat Single Sign-On server boot time and its parameters are passed in via Java system properties. This means during one Red Hat Single Sign-On server boot only one of the possible migration actions (either export, or import) can be performed.
WARNING

When the `SSO_REALM` configuration variable is set on the Red Hat Single Sign-On for OpenShift image, a database import is performed in order to create the Red Hat Single Sign-On server realm requested in the variable. For the database export to be performed correctly, the `SSO_REALM` configuration variable cannot be simultaneously defined on such image.

5. Click **Create** to deploy the application template and start pod deployment. This may take a couple of minutes.
   Then access the Red Hat Single Sign-On web console at `https://secure-sso-<sso-app-demo>.<openshift32.example.com>/auth/admin` using the administrator account.

NOTE

This example workflow uses a self-generated CA to provide an end-to-end workflow for demonstration purposes. Accessing the Red Hat Single Sign-On web console will prompt an insecure connection warning.
For production environments, Red Hat recommends that you use an SSL certificate purchased from a verified Certificate Authority.

5.2.2. (Optional) Creating additional Red Hat Single Sign-On realm and users to be also exported

When performing Red Hat Single Sign-On 6 server database export only Red Hat Single Sign-On realms and users currently present in the database will be exported. If the exported JSON file should include also additional Red Hat Single Sign-On realms and users, these need to be created first:

1. Create a new realm
2. Create new users

Upon their creation the database can be exported.

5.2.3. Export the Red Hat Single Sign-On database as a JSON file on the OpenShift pod

1. Get the Red Hat Single Sign-On deployment config and scale it down to zero.

   ```bash
   $ oc get dc -o name deploymentconfig/sso deploymentconfig/sso-mysql
   $ oc scale --replicas=0 dc sso deploymentconfig "sso" scaled
   ```


   ```bash
   $ oc set env dc/sso
   ```
3. Scale the Red Hat Single Sign-On deployment config back up. This will start the Red Hat Single Sign-On server and export its database.

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

4. (Optional) Verify that the export was successful.

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

   $ oc logs sso-4-ejr0k | grep 'Export'
09:24:59,998 INFO  [org.keycloak.services] (ServerService Thread Pool -- 57) KC-SERVICES0035: Export finished successfully

5.2.4. Retrieve and import the exported JSON file

1. Retrieve the JSON file of the Red Hat Single Sign-On database from the pod.

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

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


   **NOTE**

   For importing into an Red Hat Single Sign-On server not running on OpenShift, see the Export and Import section of the RH SSO Server Administration Guide.

Use the administration console of the Red Hat Single Sign-On server to import the resources from previously exported JSON file into the Red Hat Single Sign-On server’s database, when the Red Hat Single Sign-On server is running as a Red Hat Single Sign-On 6 container on OpenShift:

   a. Log into the master realm’s administration console of the Red Hat Single Sign-On server using the credentials used to create the administrator user. In the browser, navigate to 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.
b. At the top of the sidebar choose the name of the Red Hat Single Sign-On realm, the users, clients, realm roles, and client roles should be imported to. This example uses master realm.

c. Click the Import link under Manage section at the bottom of the sidebar.

d. In the page that opens, click Select file and then specify the location of the exported demorealm-export.json JSON file on the local file system.

e. From the Import from realm drop-down menu, select the name of the Red Hat Single Sign-On realm from which the data should be imported. This example uses master realm.

f. Choose which of users, clients, realm roles, and client roles should be imported (all of them are imported by default).

g. Choose a strategy to perform, when a resource already exists (one of Fail, Skip, or Overwrite).

NOTE

The attempt to import an object (user, client, realm role, or client role) fails if object with the same identifier already exists in the current database. Use Skip strategy to import the objects that are present in the demorealm-export.json file, but do not exist in current database.

h. Click Import to perform the import.

NOTE

When importing objects from a non-master realm to master realm or vice versa, after clicking the Import button, it is sometimes possible to encounter an error like the following one:

```
Error! App doesn’t exist in role definitions: realm-management
```

In such cases, it is necessary first to create the missing clients, having the Access Type set to bearer-only. These clients can be created by manual copy of their characteristics from the source Red Hat Single Sign-On server, on which the export JSON file was created, to the target Red Hat Single Sign-On server, where the JSON file is imported. After creation of the necessary clients, click the Import button again.

To suppress the above error message, it is needed to create the missing realm-management client, of the bearer-only Access Type, and click the Import button again.

NOTE

For Skip import strategy, the newly added objects are marked as ADDED and the object which were skipped are marked as SKIPPED, in the Action column on the import result page.
5.3. EXAMPLE WORKFLOW: CONFIGURING OPENSHEET TO USE RED HAT SINGLE SIGN-ON FOR AUTHENTICATION

Configure OpenShift to use the Red Hat Single Sign-On deployment as the authorization gateway for OpenShift. This follows on from Example Workflow: Preparing and Deploying the Red Hat Single Sign-On for OpenShift image, in which Red Hat Single Sign-On was deployed on OpenShift.

This example adds Red Hat Single Sign-On as an authentication method alongside the HTPasswd method configured in the OpenShift Primer. Once configured, both methods will be available for user login to your OpenShift web console.

5.3.1. Configuring Red Hat Single Sign-On Credentials


Create a Realm

1. Hover your cursor over the realm namespace (default is Master) at the top of the sidebar and click Add Realm.
2. Enter a realm name (this example uses OpenShift) and click Create.

Create a User

Create a test user that can be used to demonstrate the Red Hat Single Sign-On-enabled OpenShift login:

1. Click Users in the Manage sidebar to view the user information for the realm.
2. Click Add User.
3. Enter a valid Username (this example uses testuser) and any additional optional information and click Save.
4. Edit the user configuration:
   a. Click the Credentials tab in the user space and enter a password for the user.
   b. Ensure the Temporary Password option is set to Off so that it does not prompt for a password change later on, and click Reset Password to set the user password. A pop-up window prompts for additional confirmation.

Create and Configure an OpenID-Connect Client

See the Managing Clients chapter of the Red Hat Single Sign-On Server Administration Guide for more information.

1. Click Clients in the Manage sidebar and click Create.
2. Enter the **Client ID**. This example uses `openshift-demo`.

3. Select a **Client Protocol** from the drop-down menu (this example uses `openid-connect`) and click **Save**. You will be taken to the configuration **Settings** page of the `openshift-demo` client.

4. From the **Access Type** drop-down menu, select **confidential**. This is the access type for server-side applications.

5. In the **Valid Redirect URIs** dialog, enter the URI for the OpenShift web console, which is `https://openshift.example.com:8443/*` in this example.

The client **Secret** is needed to configure OpenID-Connect on the OpenShift master in the next section. You can copy it now from under the **Credentials** tab. The secret is `<7b0384a2-b832-16c5-9d73-2957842e89h7>` for this example.

### 5.3.2. Configuring OpenShift Master for Red Hat Single Sign-On Authentication

Log in to the OpenShift master CLI. You must have the required permissions to edit the `/etc/origin/master/master-config.yaml` file.

1. Edit the `/etc/origin/master/master-config.yaml` file and find the **identityProviders**. The OpenShift master, which was deployed using the OpenShift Primer, is configured with HTPassword and shows the following:

   ```yaml
   identityProviders:
   - challenge: true
     login: true
     name: htpasswd_auth
     provider:
       apiVersion: v1
       file: /etc/origin/openshift-passwd
       kind: HTPasswdPasswordIdentityProvider
   - 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
   ```
5.3.3. Logging in to OpenShift

Navigate to the OpenShift web console, which in this example is `https://openshift.example.com:8443/console`. The OpenShift login page now has the option to use either `htpasswd_auth` or `rh-sso`. The former is still available because it is present in the `/etc/origin/master/master-config.yaml`.

Select `rh-sso` and log in to OpenShift with the `testuser` user created earlier in Red Hat Single Sign-On. No projects are visible to `testuser` until they are added in the OpenShift CLI. This is the only way to provide user privileges in OpenShift because it currently does not accept external role mapping.

To provide `testuser` `view` privileges for the `sso-app-demo`, use the OpenShift CLI:

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

5.4. EXAMPLE WORKFLOW: AUTOMATICALLY REGISTERING EAP APPLICATION IN RED HAT SINGLE SIGN-ON WITH OPENID-CONNECT CLIENT

This follows on from Example Workflow: Preparing and Deploying the Red Hat Single Sign-On for OpenShift image, in which Red Hat Single Sign-On was deployed on OpenShift. This example prepares Red Hat Single Sign-On realm, role, and user credentials for an EAP project using an OpenID-Connect client.

```
preferredUsername:
 - preferred_username
name:
 - name
email:
 - email
```

a. The Red Hat Single Sign-On `Secret` hash for the `clientSecret` can be found in the Red Hat Single Sign-On web console: **Clients → openshift-demo → Credentials**

b. The endpoints for the `urls` can be found by making a request with the Red Hat Single Sign-On application. For example:

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

The response includes the `authorization_endpoint`, `token_endpoint`, and `userinfo_endpoint`.

c. This example workflow uses a self-generated CA to provide an end-to-end workflow for demonstration purposes. For this reason, the `ca` is provided as `< ca: xpaas.crt >`. This CA certificate must also be copied into the `/etc/origin/master` folder. This is not necessary if using a certificate purchased from a verified Certificate Authority.

2. Save the configuration and restart the OpenShift master:

```
$ systemctl restart atomic-openshift-master
```
client adapter. These credentials are then provided in the EAP for OpenShift template for automatic Red Hat Single Sign-On client registration. Once deployed, the Red Hat Single Sign-On user can be used to authenticate and access JBoss EAP.

NOTE
This example uses a OpenID-Connect client but an SAML client could also be used. See Red Hat Single Sign-On Clients and Automatic and Manual Red Hat Single Sign-On Client Registration Methods for more information on the differences between OpenID-Connect and SAML clients.

5.4.1. Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment

Log in to the OpenShift CLI with a user that holds the `cluster:admin` role.

1. Create a new project:

   $ oc new-project eap-app-demo

2. Add the `view` role to the `default` service account. This enables the service account to view all the resources in the `eap-app-demo` namespace, which is necessary for managing the cluster.

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

3. The EAP template requires an SSL keystore and a JGroups keystore. This example uses `keytool`, a package included with the Java Development Kit, to generate self-signed certificates for these keystores. The following commands will prompt for passwords.

   a. Generate a secure key for the SSL keystore:

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

   b. Generate a secure key for the JGroups keystore:

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

4. Generate the EAP for OpenShift secrets with the SSL and JGroup keystore files:

   $ oc secret new eap-ssl-secret eapkeystore.jks
   $ oc secret new eap-jgroup-secret eapjgroups.jceks

5. Add the EAP secret to the `default` service account:

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

5.4.2. Preparing the Red Hat Single Sign-On Credentials


Create a Realm
1. Hover your cursor over the realm namespace at the top of the sidebar and click *Add Realm*.

2. Enter a realm name (this example uses `eap-demo`) and click **Create**.

**Copy the Public Key**

In the newly created `eap-demo` realm, click the **Keys** tab and copy the generated public key. This example uses the variable `<realm-public-key>` for brevity. This is used later to deploy the Red Hat Single Sign-On-enabled JBoss EAP image.

**Create a Role**

Create a role in Red Hat Single Sign-On with a name that corresponds to the JEE role defined in the `web.xml` of the example EAP application. This role is assigned to an Red Hat Single Sign-On **application user** to authenticate access to user applications.

1. Click **Roles** in the **Configure** sidebar to list the roles for this realm. This is a new realm, so there should only be the default `offline_access` role.

2. Click **Add Role**.

3. Enter the role name (this example uses the role `eap-user-role`) and click **Save**.

**Create Users and Assign Roles**

Create two users: - Assign the **realm management user** the `realm-management` roles to handle automatic Red Hat Single Sign-On client registration in the Red Hat Single Sign-On server. - Assign the **application user** the JEE role, created in the previous step, to authenticate access to user applications.

Create the **realm management user**:

1. Click **Users** in the **Manage** sidebar to view the user information for the realm.

2. Click **Add User**.

3. Enter a valid **Username** (this example uses the user `eap-mgmt-user`) and click **Save**.

4. Edit the user configuration. Click the **Credentials** tab in the user space and enter a password for the user. After the password has been confirmed you can click **Reset Password** to set the user password. A pop-up window prompts for additional confirmation.

5. Click **Role Mappings** to list the realm and client role configuration. In the **Client Roles** drop-down menu, select `realm-management` and add all of the available roles to the user. This provides the user Red Hat Single Sign-On server rights that can be used by the JBoss EAP image to create clients.

Create the **application user**:

1. Click **Users** in the **Manage** sidebar to view the user information for the realm.

2. Click **Add User**.

3. Enter a valid **Username** and any additional optional information for the **application user** and click **Save**.

4. Edit the user configuration. Click the **Credentials** tab in the user space and enter a password for the user. After the password has been confirmed you can click **Reset Password** to set the user password. A pop-up window prompts for additional confirmation.
5. Click **Role Mappings** to list the realm and client role configuration. In **Available Roles**, add the role created earlier.

### 5.4.3. Deploy the Red Hat Single Sign-On-enabled JBoss EAP Image

1. Return to the OpenShift web console and click **Add to project** to list the default image streams and templates.

2. Use the **Filter by keyword** search bar to limit the list to those that match **sso**. You may need to click **See all** to show the desired application template.

3. Select the **eap71-sso-s2i** image to list all of the deployment parameters. Include the following Red Hat Single Sign-On parameters to configure the Red Hat Single Sign-On credentials during the EAP build:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_NAME</td>
<td>sso</td>
</tr>
<tr>
<td>HOSTNAME_HTTPS</td>
<td>secure-sample-jsp.eap-app-demo.openshift32.example.com</td>
</tr>
<tr>
<td>HOSTNAME_HTTP</td>
<td>sample-jsp.eap-app-demo.openshift32.example.com</td>
</tr>
<tr>
<td>SOURCE_REPOSITORY_URL</td>
<td><a href="https://repository-example.com/developer/application">https://repository-example.com/developer/application</a></td>
</tr>
<tr>
<td>SSO_URL</td>
<td><a href="https://secure-sso-sso-app-demo.openshift32.example.com/auth">https://secure-sso-sso-app-demo.openshift32.example.com/auth</a></td>
</tr>
<tr>
<td>SSO_REALM</td>
<td>eap-demo</td>
</tr>
<tr>
<td>SSO_USERNAME</td>
<td>eap-mgmt-user</td>
</tr>
<tr>
<td>SSO_PASSWORD</td>
<td>password</td>
</tr>
<tr>
<td>SSO_PUBLIC_KEY</td>
<td>&lt;realm-public-key&gt;</td>
</tr>
<tr>
<td>HTTPS_KEYSTORE</td>
<td>eapkeystore.jks</td>
</tr>
<tr>
<td>HTTPS_PASSWORD</td>
<td>password</td>
</tr>
<tr>
<td>HTTPS_SECRET</td>
<td>eap-ssl-secret</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_KEYSTORE</td>
<td>eapjgroups.jceks</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_PASSWORD</td>
<td>password</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_SECRET</td>
<td>eap-jgroup-secret</td>
</tr>
</tbody>
</table>
4. Click **Create** to deploy the JBoss EAP image.

It may take several minutes for the JBoss EAP image to deploy.

### 5.4.4. Log in to the JBoss EAP Server Using Red Hat Single Sign-On

1. Access the JBoss EAP application server and click **Login**. You are redirected to the Red Hat Single Sign-On login.

2. Log in using the Red Hat Single Sign-On user created in the example. You are authenticated against the Red Hat Single Sign-On server and returned to the JBoss EAP application server.

### 5.5. EXAMPLE WORKFLOW: MANUALLY REGISTERING EAP APPLICATION IN RED HAT SINGLE SIGN-ON WITH SAML CLIENT

This follows on from **Example Workflow: Preparing and Deploying the Red Hat Single Sign-On for OpenShift image**, in which Red Hat Single Sign-On was deployed on OpenShift.

This example prepares Red Hat Single Sign-On realm, role, and user credentials for an EAP project and configures an EAP for OpenShift deployment. Once deployed, the Red Hat Single Sign-On user can be used to authenticate and access JBoss EAP.

**NOTE**

This example uses a SAML client but an OpenID-Connect client could also be used. See [Red Hat Single Sign-On Clients](Red_Hat_Single_Sign_On_Continuous_Delivery_6-Red_Hat_Single_Sign_On_for_OpenShift#red_hat_single_sign_on_clients) and [Automatic and Manual Red Hat Single Sign-On Client Registration Methods](Red_Hat_Single_Sign_On_Continuous_Delivery_6-Red_Hat_Single_Sign_On_for_OpenShift#automatic_and_manual_red_hat_single_sign_on_client_registration_methods) for more information on the differences between SAML and OpenID-Connect clients.

### 5.5.1. Preparing the Red Hat Single Sign-On Credentials

Log in to the encrypted Red Hat Single Sign-On web server at `https://secure-sso-<project-name>.<hostname>/auth/admin` using the **administrator account** created during the Red Hat Single Sign-On deployment.

**Create a Realm**

1. Hover your cursor over the realm namespace (default is **Master**) at the top of the sidebar and click **Add Realm**.

2. Enter a realm name (this example uses `saml-demo`) and click **Create**.

**Copy the Public Key**

In the newly created `saml-demo` realm, click the **Keys** tab and copy the generated public key. This example uses the variable `realm-public-key` for brevity. This is needed later to deploy the Red Hat Single Sign-On-enabled JBoss EAP image.

**Create a Role**

Create a role in Red Hat Single Sign-On with a name that corresponds to the JEE role defined in the `web.xml` of the example EAP application. This role will be assigned to a Red Hat Single Sign-On **application user** to authenticate access to user applications.
1. Click Roles in the Configure sidebar to list the roles for this realm. This is a new realm, so there should only be the default offline_access role.

2. Click Add Role.

3. Enter the role name (this example uses the role saml-user-role) and click Save.

Create Users and Assign Roles

Create two users: - Assign the realm management user the realm-management roles to handle automatic Red Hat Single Sign-On client registration in the Red Hat Single Sign-On server. - Assign the application user the JEE role, created in the previous step, to authenticate access to user applications.

Create the realm management user:

1. Click Users in the Manage sidebar to view the user information for the realm.

2. Click Add User.

3. Enter a valid Username (this example uses the user app-mgmt-user) and click Save.

4. Edit the user configuration. Click the Credentials tab in the user space and enter a password for the user. After the password has been confirmed you can click Reset Password to set the user password. A pop-up window prompts for additional confirmation.

Create the application user:

1. Click Users in the Manage sidebar to view the user information for the realm.

2. Click Add User.

3. Enter a valid Username and any additional optional information for the application user and click Save.

4. Edit the user configuration. Click the Credentials tab in the user space and enter a password for the user. After the password has been confirmed you can click Reset Password to set the user password. A pop-up window prompts for additional confirmation.

5. Click Role Mappings to list the realm and client role configuration. In Available Roles, add the role created earlier.

Create and Configure a SAML Client

Clients are Red Hat Single Sign-On entities that request user authentication. This example configures a SAML client to handle authentication for the EAP application. This section saves two files, keystore.jks and keycloak-saml-subsystem.xml that are needed later in the procedure.

Create the SAML Client:

1. Click Clients in the Configure sidebar to list the clients in the realm. Click Create.

2. Enter a valid Client ID. This example uses sso-saml-demo.

3. In the Client Protocol drop-down menu, select saml.

4. Enter the Root URL for the application. This example uses https://demoapp-eap-app-demo.openshift32.example.com.
Configure the SAML Client:

In the **Settings** tab, set the **Root URL** and the **Valid Redirect URLs** for the new **sso-saml-demo** client:

1. For the **Root URL**, enter the same address used when creating the client. This example uses https://demoapp-eap-app-demo.openshift32.example.com.

2. For the **Valid Redirect URLs**, enter an address for users to be redirected to at when they log in or out. This example uses a redirect address relative to the root https://demoapp-eap-app-demo.openshift32.example.com/*. 

Export the SAML Keys:

1. Click the **SAML Keys** tab in the **sso-saml-demo** client space and click **Export**.

2. For this example, leave the **Archive Format** as **JKS**. This example uses the default **Key Alias** of **sso-saml-demo** and default **Realm Certificate Alias** of **saml-demo**.

3. Enter the **Key Password** and the **Store Password**. This example uses **password** for both.

4. Click **Download** and save the **keystore-saml.jks** file for use later.

5. Click the **sso-saml-demo** client to return to the client space ready for the next step.

Download the Client Adapter:

1. Click **Installation**.

2. Use the **Format Option** drop-down menu to select a format. This example uses **Keycloak SAML Wildfly/JBoss Subsystem**.

3. Click **Download** and save the file **keycloak-saml-subsystem.xml**.

The **keystore-saml.jks** will be used with the other EAP keystores in the next section to create an OpenShift secret for the EAP application project. Copy the **keystore-saml.jks** file to an OpenShift node.

The **keycloak-saml-subsystem.xml** will be modified and used in the application deployment. Copy it into the **/configuration** folder of the application as **secure-saml-deployments**.

### 5.5.2. Preparing Red Hat Single Sign-On Authentication for OpenShift Deployment

Log in to the OpenShift CLI with a user that holds the **cluster:admin** role.

1. Create a new project:
   
   $ oc new-project eap-app-demo

2. Add the **view** role to the **default** service account. This enables the service account to view all the resources in the **eap-app-demo** namespace, which is necessary for managing the cluster.

   $ oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default
3. The EAP template requires an SSL keystore and a JGroups keystore. This example uses keytool, a package included with the Java Development Kit, to generate self-signed certificates for these keystores. The following commands will prompt for passwords.

   a. Generate a secure key for the SSL keystore:

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

   b. Generate a secure key for the JGroups keystore:

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

4. Generate the EAP for OpenShift secrets with the SSL and JGroup keystore files:

   ```
   $ oc secret new eap-ssl-secret eapkeystore.jks
   $ oc secret new eap-jgroup-secret eapjgroups.jceks
   ```

5. Add the EAP application secret to the EAP service account created earlier:

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

### 5.5.3. Modifying the secure-saml-deployments File

The keycloak-saml-subsystem.xml, exported from the Red Hat Single Sign-On client in a previous section, should have been copied into the /configuration folder of the application and renamed secure-saml-deployments. EAP searches for this file when it starts and copies it to the standalone-openshift.xml file inside the Red Hat Single Sign-On SAML adapter configuration.

1. Open the /configuration/secure-saml-deployments file in a text editor.

2. Replace the YOUR-WAR.war value of the secure-deployment name tag with the application .war file. This example uses sso-saml-demo.war.

3. Replace the SPECIFY YOUR LOGOUT PAGE! value of the logout page tag with the url to redirect users when they log out of the application. This example uses /index.jsp.

4. Delete the <PrivateKeyPem> and <CertificatePem> tags and keys and replace it with keystore information:

   ```
   ...<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>
   ```

   The mount path of the keystore-saml.jks (in this example /etc/eap-secret-volume/keystore-saml.jks) can be specified in the application template with the parameter EAP_HTTPS_KEYSTORE_DIR.

   The aliases and passwords for the PrivateKey and the Certificate were configured when the SAML Keys were exported from the Red Hat Single Sign-On client.
5. Delete the second `<CertificatePem>` tag and key and replace it with the the realm certificate information:

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

The certificate alias and password were configured when the SAML Keys were exported from the Red Hat Single Sign-On client.

6. Save and close the `/configuration/secure-saml-deployments` file.

### 5.5.4. Configuring SAML Client Registration in the Application `web.xml`

The client type must also be specified by the `<auth-method>` key in the application `web.xml`. This file is read by the image at deployment.

Open the application `web.xml` file and ensure it includes the following:

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

### 5.5.5. Deploying the Application

You do not need to include any Red Hat Single Sign-On configuration for the image because that has been configured in the application itself. Navigating to the application login page redirects you to the Red Hat Single Sign-On login. Log in to the application through Red Hat Single Sign-On using the `application user` user created earlier.
CHAPTER 6. REFERENCE

6.1. ARTIFACT REPOSITORY MIRRORS

A repository in Maven holds build artifacts and dependencies of various types (all the project jars, library jar, plugins or any other project specific artifacts). It also specifies locations from where to download artifacts from, while performing the S2I build. Besides using central repositories, it is a common practice for organizations to deploy a local custom repository (mirror).

Benefits of using a mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Ability to have greater control over the repository content.
- Possibility to share artifacts across different teams (developers, CI), without the need to rely on public servers and repositories.
- Improved build times.

Often, a repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at http://10.0.0.1:8080/repository/internal/, the S2I build can then use this manager by supplying the MAVEN_MIRROR_URL environment variable to the build configuration of the application as follows:

1. Identify the name of the build configuration to apply MAVEN_MIRROR_URL variable against:

   $ oc get bc -o name
   buildconfig/sso

2. Update build configuration of sso with a MAVEN_MIRROR_URL environment variable

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

3. Verify the setting

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

4. Schedule new build of the application

   NOTE

   During application build, you will notice that Maven dependencies are pulled from the repository manager, instead of the default public repositories. Also, after the build is finished, you will see that the mirror is filled with all the dependencies that were retrieved and used during the build.

A repository in Maven holds build artifacts and dependencies of various types (all the project jars, library jar, plugins or any other project specific artifacts). It also specifies locations from where to download
artifacts from, while performing the S2I build. Besides using central repositories, it is a common practice for organizations to deploy a local custom repository (mirror).

Benefits of using a mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Ability to have greater control over the repository content.
- Possibility to share artifacts across different teams (developers, CI), without the need to rely on public servers and repositories.
- Improved build times.

Often, a repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at http://10.0.0.1:8080/repository/internal/, the S2I build can then use this manager by supplying the MAVEN_MIRROR_URL environment variable to the build configuration of the application as follows:

1. Identify the name of the build configuration to apply MAVEN_MIRROR_URL variable against:

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

2. Update build configuration of sso with a MAVEN_MIRROR_URL environment variable

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

3. Verify the setting

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

4. Schedule new build of the application

   **NOTE**

   During application build, you will notice that Maven dependencies are pulled from the repository manager, instead of the default public repositories. Also, after the build is finished, you will see that the mirror is filled with all the dependencies that were retrieved and used during the build.

6.2. ENVIRONMENT VARIABLES

6.2.1. Information Environment Variables

The following information environment variables are designed to convey information about the image and should not be modified by the user:

Table 6.1. Information Environment Variables
### 6.2.2. Configuration Environment Variables

Configuration environment variables are designed to conveniently adjust the image without requiring a rebuild, and should be set by the user as desired.

#### Table 6.2. Configuration Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB_JOLOKIA_AUTH_OPENSIF</td>
<td>Switch on client authentication for OpenShift TLS communication. The value of this parameter can be a relative distinguished name which must be contained in a presented client’s certificate. Enabling this parameter will automatically switch Jolokia into https communication mode. The default CA cert is set to /var/run/secrets/kubernetes.io/serviceaccount/ca.crt.</td>
<td>true</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>AB_JOLOKIA_CONFIG</td>
<td>If set uses this file (including path) as Jolokia JVM agent properties</td>
<td>/opt/jolokia/custom.properties</td>
</tr>
<tr>
<td></td>
<td>(as described in Jolokia’s reference manual). If not set, the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/opt/jolokia/etc/jolokia.properties file will be created using the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>settings as defined in this document, otherwise the rest of the settings</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in this document are ignored.</td>
<td></td>
</tr>
<tr>
<td>AB_JOLOKIA_DISCOVERY_ENABLE</td>
<td>Enable Jolokia discovery. Defaults to false.</td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_HOST</td>
<td>Host address to bind to. Defaults to 0.0.0.0.</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>AB_JOLOKIA_HTTPS</td>
<td>Switch on secure communication with https. By default self-signed server</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>certificates are generated if no serverCert configuration is given in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AB_JOLOKIA_OPTS. NOTE: If the values is set to an empty string, https is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>turned off. If the value is set to a non empty string, https is turned on.</td>
<td></td>
</tr>
<tr>
<td>AB_JOLOKIA_ID</td>
<td>Agent ID to use ($HOSTNAME by default, which is the container id).</td>
<td>openjdk-app-1-xqlsj</td>
</tr>
<tr>
<td>AB_JOLOKIA_OFF</td>
<td>If set disables activation of Jolokia (i.e. echos an empty value). By</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>default, Jolokia is enabled. NOTE: If the values is set to an empty string,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>https is turned off. If the value is set to a non empty string, https is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>turned on.</td>
<td></td>
</tr>
<tr>
<td>AB_JOLOKIA_OPTS</td>
<td>Additional options to be appended to the agent configuration. They should be</td>
<td>backlog=20</td>
</tr>
<tr>
<td></td>
<td>given in the format &quot;key=value, key=value, ...&lt;200b&gt;&quot;</td>
<td></td>
</tr>
<tr>
<td>AB_JOLOKIA_PASSWORD</td>
<td>Password for basic authentication. By default authentication is</td>
<td>mypassword</td>
</tr>
<tr>
<td></td>
<td>switched off.</td>
<td></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>AB_JOLOKIA_PASSWORD_RANDOM</td>
<td>If set, a random value is generated for AB_JOLOKIA_PASSWORD, and it is saved in the /opt/jolokia/etc/jolokia.pw file.</td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_PORT</td>
<td>Port to use (Default: 8778).</td>
<td>5432</td>
</tr>
<tr>
<td>AB_JOLOKIA_USER</td>
<td>User for basic authentication. Defaults to jolokia.</td>
<td>myusername</td>
</tr>
<tr>
<td>CONTAINER_CORE_LIMIT</td>
<td>A calculated core limit as described in CFS Bandwidth Control.</td>
<td>2</td>
</tr>
<tr>
<td>GC_ADAPTIVE_SIZE_POLICY_WEIGHT</td>
<td>The weighting given to the current Garbage Collection (GC) time versus previous GC times.</td>
<td>90</td>
</tr>
<tr>
<td>GC_MAX_HEAP_FREE_RATIO</td>
<td>Maximum percentage of heap free after GC to avoid shrinking.</td>
<td>40</td>
</tr>
<tr>
<td>GC_MAX_METASPACE_SIZE</td>
<td>The maximum metaspace size.</td>
<td>100</td>
</tr>
<tr>
<td>GGC_TIME_RATIO_MIN_HEAP_FREE_RATIO</td>
<td>Minimum percentage of heap free after GC to avoid expansion.</td>
<td>20</td>
</tr>
<tr>
<td>GC_TIME_RATIO</td>
<td>Specifies the ratio of the time spent outside the garbage collection (for example, the time spent for application execution) to the time spent in the garbage collection.</td>
<td>4</td>
</tr>
<tr>
<td>JAVA_DIAGNOSTICS</td>
<td>Set this to get some diagnostics information to standard out when things are happening.</td>
<td>true</td>
</tr>
<tr>
<td>JAVA_INITIAL_MEM_RATIO</td>
<td>This is used to calculate a default initial heap memory based the maximal heap memory. The default is 100% of the maximal heap is used for the initial heap size. You can skip this mechanism by setting this value to 0 in which case no -Xms option is added.</td>
<td>100</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>JAVA_MAX_MEM_RATIO</td>
<td>It is used to calculate a default maximal heap memory based on a containers restriction. If used in a Docker container without any memory constraints for the container then this option has no effect. If there is a memory constraint then <code>-Xmx</code> is set to a ratio of the container available memory as set here. The default is 50 which means 50% of the available memory is used as an upper boundary. You can skip this mechanism by setting this value to 0 in which case no <code>-Xmx</code> option is added.</td>
<td>40</td>
</tr>
<tr>
<td>JAVA_OPTS_APPEND</td>
<td>Server startup options.</td>
<td>-</td>
</tr>
<tr>
<td>MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION</td>
<td>For backwards compatability, set to true to use MyQueue and MyTopic as physical destination name defaults instead of queue/MyQueue and topic/MyTopic.</td>
<td>false</td>
</tr>
<tr>
<td>OPENSPLIT_KUBE_PING_LABELS</td>
<td>Clustering labels selector.</td>
<td>app=sso-app</td>
</tr>
<tr>
<td>OPENSPLIT_KUBE_PING_NAMESPACE</td>
<td>Clustering project namespace.</td>
<td>myproject</td>
</tr>
<tr>
<td>SCRIPT_DEBUG</td>
<td>If set to true, ensures that the bash scripts are executed with the <code>-x</code> option, printing the commands and their arguments as they are executed.</td>
<td>true</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>SSO_ADMIN_PASSWORD</td>
<td>Password of the administrator account for the master realm of the Red Hat Single Sign-On server. <strong>Required.</strong> If no value is specified, it is auto generated and displayed as an OpenShift Instructional message when the template is instantiated.</td>
<td>adm-password</td>
</tr>
<tr>
<td>SSO_ADMIN_USERNAME</td>
<td>Username of the administrator account for the master realm of the Red Hat Single Sign-On server. <strong>Required.</strong> If no value is specified, it is auto generated and displayed as an OpenShift Instructional message when the template is instantiated.</td>
<td>admin</td>
</tr>
<tr>
<td>SSO_HOSTNAME</td>
<td>Custom hostname for the Red Hat Single Sign-On server. <strong>Not set by default.</strong> If not set, the request hostname SPI provider, which uses the request headers to determine the hostname of the Red Hat Single Sign-On server is used. If set, the fixed hostname SPI provider, with the hostname of the Red Hat Single Sign-On server set to the provided variable value, is used. See dedicated Customizing Hostname for the Red Hat Single Sign-On Server section for additional steps to be performed, when SSO_HOSTNAME variable is set.</td>
<td>rh-sso-server.openshift.example.com</td>
</tr>
<tr>
<td>SSO_REALM</td>
<td>Name of the realm to be created in the Red Hat Single Sign-On server if this environment variable is provided.</td>
<td>demo</td>
</tr>
<tr>
<td>SSO_SERVICE_PASSWORD</td>
<td>The password for the Red Hat Single Sign-On service user.</td>
<td>mgmt-password</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_USERNAME</strong></td>
<td>The username used to access the Red Hat Single Sign-On service. This is used by clients to create the application client(s) within the specified Red Hat Single Sign-On realm. This user is created if this environment variable is provided.</td>
<td>sso-mgmtuser</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE</strong></td>
<td>The name of the truststore file within the secret.</td>
<td>truststore.jks</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE_DIR</strong></td>
<td>Truststore directory.</td>
<td>/etc/sso-secret-volume</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE_PASSWORD</strong></td>
<td>The password for the truststore and certificate.</td>
<td>mykeystorepass</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE_SECRET</strong></td>
<td>The name of the secret containing the truststore file. Used for sso-truststore-volume volume.</td>
<td>truststore-secret</td>
</tr>
</tbody>
</table>

Available application templates for Red Hat Single Sign-On for OpenShift can combine the aforementioned configuration variables with common OpenShift variables (for example `APPLICATION_NAME` or `SOURCE_REPOSITORY_URL`), product specific variables (e.g. `HORNETQ_CLUSTER_PASSWORD`), or configuration variables typical to database images (e.g. `MYSQL_FT_MAX_WORD_LEN`) yet. All of these different types of configuration variables can be adjusted as desired to achieve the deployed Red Hat Single Sign-On-enabled application will align with the intended use case as much as possible. The list of configuration variables, available for each category of application templates for Red Hat Single Sign-On-enabled applications, is described below.

### 6.2.3. Template variables for all Red Hat Single Sign-On images

**Table 6.3. Configuration Variables Available For All Red Hat Single Sign-On Images**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>APPLICATION_NAME</code></td>
<td>The name for the application.</td>
</tr>
<tr>
<td><code>DB_MAX_POOL_SIZE</code></td>
<td>Sets xa-pool/max-pool-size for the configured datasource.</td>
</tr>
<tr>
<td><code>DB_TX_ISOLATION</code></td>
<td>Sets transaction-isolation for the configured datasource.</td>
</tr>
<tr>
<td><code>DB_USERNAME</code></td>
<td>Database user name.</td>
</tr>
<tr>
<td><code>HOSTNAME_HTTP</code></td>
<td>Custom hostname for http service route. Leave blank for default hostname, e.g.: &lt;application-name&gt;, &lt;project&gt;, &lt;default-domain-suffix&gt;.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HOSTNAME_HTTPS</td>
<td>Custom hostname for https service route. Leave blank for default hostname, e.g.: <code>&lt;application-name&gt;.&lt;project&gt;.&lt;default-domain-suffix&gt;</code>.</td>
</tr>
<tr>
<td>HTTPS_KEYSTORE</td>
<td>The name of the keystore file within the secret. If defined along with HTTPS_PASSWORD and HTTPS_NAME, enable HTTPS and set the SSL certificate key file to a relative path under $JBOSS_HOME/standalone/configuration.</td>
</tr>
<tr>
<td>HTTPS_KEYSTORE_TYPE</td>
<td>The type of the keystore file (JKS or JCEKS).</td>
</tr>
<tr>
<td>HTTPS_NAME</td>
<td>The name associated with the server certificate (e.g. jboss). If defined along with HTTPS_PASSWORD and HTTPS_KEYSTORE, enable HTTPS and set the SSL name.</td>
</tr>
<tr>
<td>HTTPS_PASSWORD</td>
<td>The password for the keystore and certificate (e.g. mykeystorepass). If defined along with HTTPS_NAME and HTTPS_KEYSTORE, enable HTTPS and set the SSL key password.</td>
</tr>
<tr>
<td>HTTPS_SECRET</td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td>IMAGE_STREAM_NAMESPACE</td>
<td>Namespace in which the ImageStreams for Red Hat Middleware images are installed. These ImageStreams are normally installed in the openshift namespace. You should only need to modify this if you’ve installed the ImageStreams in a different namespace/project.</td>
</tr>
<tr>
<td>JGROUPS_CLUSTER_PASSWORD</td>
<td>JGroups cluster password.</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_KEYSTORE</td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_NAME</td>
<td>The name associated with the server certificate (e.g. secret-key).</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_PASSWORD</td>
<td>The password for the keystore and certificate (e.g. password).</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_SECRET</td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td>SSO_ADMIN_USERNAME</td>
<td>Username of the administrator account for the master realm of the Red Hat Single Sign-On server. Required. If no value is specified, it is auto generated and displayed as an OpenShift instructional message when the template is instantiated.</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSO_ADMIN_PASSWORD</strong></td>
<td>Password of the administrator account for the master realm of the Red Hat Single Sign-On server. Required. If no value is specified, it is auto generated and displayed as an OpenShift instructional message when the template is instantiated.</td>
</tr>
<tr>
<td><strong>SSO_REALM</strong></td>
<td>Name of the realm to be created in the Red Hat Single Sign-On server if this environment variable is provided.</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_USERNAME</strong></td>
<td>The username used to access the Red Hat Single Sign-On service. This is used by clients to create the application client(s) within the specified Red Hat Single Sign-On realm. This user is created if this environment variable is provided.</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_PASSWORD</strong></td>
<td>The password for the Red Hat Single Sign-On service user.</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE</strong></td>
<td>The name of the truststore file within the secret.</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE_SECRET</strong></td>
<td>The name of the secret containing the truststore file. Used for <code>sso-truststore-volume</code> volume.</td>
</tr>
<tr>
<td><strong>SSO_TRUSTSTORE_PASSWORD</strong></td>
<td>The password for the truststore and certificate.</td>
</tr>
</tbody>
</table>

### 6.2.4. Template variables specific to sso-cd-mysql, sso-cd-mysql-persistent, and sso-cd-x509-mysql-persistent

Table 6.4. Configuration Variables Specific To Red Hat Single Sign-On-enabled MySQL Applications With Ephemeral Or Persistent Storage

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB_USERNAME</strong></td>
<td>Database user name.</td>
</tr>
<tr>
<td><strong>DB_PASSWORD</strong></td>
<td>Database user password.</td>
</tr>
<tr>
<td><strong>DB_JNDI</strong></td>
<td>Database JNDI name used by application to resolve the datasource, e.g. <code>java:/jboss/datasources/mysql</code></td>
</tr>
<tr>
<td><strong>MYSQL_AIO</strong></td>
<td>Controls the <code>innodb_use_native_aio</code> setting value if the native AIO is broken.</td>
</tr>
<tr>
<td><strong>MYSQL_FT_MAX_WORD_LEN</strong></td>
<td>The maximum length of the word to be included in a FULLTEXT index.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>MYSQL_FT_MIN_WORD_LEN</td>
<td>The minimum length of the word to be included in a FULLTEXT index.</td>
</tr>
<tr>
<td>MYSQL_LOWER_CASE_TABLE_NAMES</td>
<td>Sets how the table names are stored and compared.</td>
</tr>
<tr>
<td>MYSQL_MAX_CONNECTIONS</td>
<td>The maximum permitted number of simultaneous client connections.</td>
</tr>
</tbody>
</table>

### 6.2.5. Template variables specific to sso-cd-postgresql, sso-cd-postgresql-persistent, and sso-cd-x509-postgresql-persistent

Table 6.5. Configuration Variables Specific To Red Hat Single Sign-On-enabled PostgreSQL Applications With Ephemeral Or Persistent Storage

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_USERNAME</td>
<td>Database user name.</td>
</tr>
<tr>
<td>DB_PASSWORD</td>
<td>Database user password.</td>
</tr>
<tr>
<td>DB_JNDI</td>
<td>Database JNDI name used by application to resolve the datasource, e.g. java:/jboss/datasources/postgresql</td>
</tr>
<tr>
<td>POSTGRESQL_MAX_CONNECTIONS</td>
<td>The maximum number of client connections allowed. This also sets the maximum number of prepared transactions.</td>
</tr>
<tr>
<td>POSTGRESQL_SHARED_BUFFERS</td>
<td>Configures how much memory is dedicated to PostgreSQL for caching data.</td>
</tr>
</tbody>
</table>

### 6.2.6. Template variables specific to sso-cd-mysql-persistent, sso-cd-x509-mysql-persistent, sso-cd-postgresql-persistent, and sso-cd-x509-postgresql-persistent

Table 6.6. Configuration Variables Specific To Red Hat Single Sign-On-enabled MySQL / PostgreSQL Applications With Persistent Storage

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOLUME_CAPACITY</td>
<td>Size of persistent storage for database volume.</td>
</tr>
</tbody>
</table>

### 6.2.7. Template variables for general eap64 and eap71 S2I images

Table 6.7. Configuration Variables For EAP 6.4 and EAP 7 Applications Built Via S2I
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>APPLICATION_NAME</strong></td>
<td>The name for the application.</td>
</tr>
<tr>
<td><strong>ARTIFACT_DIR</strong></td>
<td>Artifacts directory.</td>
</tr>
<tr>
<td><strong>AUTO_DEPLOY_EXPLODED</strong></td>
<td>Controls whether exploded deployment content should be automatically deployed.</td>
</tr>
<tr>
<td><strong>CONTEXT_DIR</strong></td>
<td>Path within Git project to build; empty for root project directory.</td>
</tr>
<tr>
<td><strong>GENERIC_WEBHOOK_SECRET</strong></td>
<td>Generic build trigger secret.</td>
</tr>
<tr>
<td><strong>GITHUB_WEBHOOK_SECRET</strong></td>
<td>GitHub trigger secret.</td>
</tr>
<tr>
<td><strong>HORNETQ_CLUSTER_PASSWORD</strong></td>
<td>HornetQ cluster administrator password.</td>
</tr>
<tr>
<td><strong>HORNETQ_QUEUES</strong></td>
<td>Queue names.</td>
</tr>
<tr>
<td><strong>HORNETQ_TOPICS</strong></td>
<td>Topic names.</td>
</tr>
<tr>
<td><strong>HOSTNAME_HTTP</strong></td>
<td>Custom host name for http service route. Leave blank for default host name, e.g.: <code>&lt;application-name&gt;.&lt;project&gt;.&lt;default-domain-suffix&gt;</code>.</td>
</tr>
<tr>
<td><strong>HOSTNAME_HTTPS</strong></td>
<td>Custom host name for https service route. Leave blank for default host name, e.g.: <code>&lt;application-name&gt;.&lt;project&gt;.&lt;default-domain-suffix&gt;</code>.</td>
</tr>
<tr>
<td><strong>HTTPS_KEYSTORE_TYPE</strong></td>
<td>The type of the keystore file (JKS or JCEKS).</td>
</tr>
<tr>
<td><strong>HTTPS_KEYSTORE</strong></td>
<td>The name of the keystore file within the secret. If defined along with <strong>HTTPS_PASSWORD</strong> and <strong>HTTPS_NAME</strong>, enable HTTPS and set the SSL certificate key file to a relative path under <code>$JBOSS_HOME/standalone/configuration</code>.</td>
</tr>
<tr>
<td><strong>HTTPS_NAME</strong></td>
<td>The name associated with the server certificate (e.g. jboss). If defined along with <strong>HTTPS_PASSWORD</strong> and <strong>HTTPS_KEYSTORE</strong>, enable HTTPS and set the SSL name.</td>
</tr>
<tr>
<td><strong>HTTPS_PASSWORD</strong></td>
<td>The password for the keystore and certificate (e.g. mykeystorepass). If defined along with <strong>HTTPS_NAME</strong> and <strong>HTTPS_KEYSTORE</strong>, enable HTTPS and set the SSL key password.</td>
</tr>
<tr>
<td><strong>HTTPS_SECRET</strong></td>
<td>The name of the secret containing the keystore file.</td>
</tr>
</tbody>
</table>
Namespace in which the ImageStreams for Red Hat Middleware images are installed. These ImageStreams are normally installed in the openshift namespace. You should only need to modify this if you’ve installed the ImageStreams in a different namespace/project.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IMAGE_STREAM_NAMESPACE</strong></td>
<td>Namespace in which the ImageStreams for Red Hat Middleware images are installed. These ImageStreams are normally installed in the openshift namespace. You should only need to modify this if you’ve installed the ImageStreams in a different namespace/project.</td>
</tr>
<tr>
<td><strong>JGROUPS_CLUSTER_PASSWORD</strong></td>
<td>JGroups cluster password.</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_KEYSTORE</strong></td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_NAME</strong></td>
<td>The name associated with the server certificate (e.g. secret-key).</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_PASSWORD</strong></td>
<td>The password for the keystore and certificate (e.g. password).</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_SECRET</strong></td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td><strong>SOURCE_REPOSITORY_REF</strong></td>
<td>Git branch/tag reference.</td>
</tr>
<tr>
<td><strong>SOURCE_REPOSITORY_URL</strong></td>
<td>Git source URI for application.</td>
</tr>
</tbody>
</table>

### 6.2.8. Template variables specific to eap64-sso-s2i and eap71-sso-s2i for automatic client registration

Table 6.8. Configuration Variables For EAP 6.4 and EAP 7 Red Hat Single Sign-On-enabled Applications Built Via S2I

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSO_URL</strong></td>
<td>Red Hat Single Sign-On server location.</td>
</tr>
<tr>
<td><strong>SSO_REALM</strong></td>
<td>Name of the realm to be created in the Red Hat Single Sign-On server if this environment variable is provided.</td>
</tr>
<tr>
<td><strong>SSO_USERNAME</strong></td>
<td>The username used to access the Red Hat Single Sign-On service. This is used to create the application client(s) within the specified Red Hat Single Sign-On realm. This should match the <strong>SSO_SERVICE_USERNAME</strong> specified through one of the sso-cd- templates.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSO_PASSWORD</td>
<td>The password for the Red Hat Single Sign-On service user.</td>
</tr>
<tr>
<td>SSO_PUBLIC_KEY</td>
<td>Red Hat Single Sign-On public key. Public key is recommended to be passed into the template to avoid man-in-the-middle security attacks.</td>
</tr>
<tr>
<td>SSO_SECRET</td>
<td>The Red Hat Single Sign-On client secret for confidential access.</td>
</tr>
<tr>
<td>SSO_SERVICE_URL</td>
<td>Red Hat Single Sign-On service location.</td>
</tr>
<tr>
<td>SSO_TRUSTSTORE_SECRET</td>
<td>The name of the secret containing the truststore file.</td>
</tr>
<tr>
<td></td>
<td>Used for sso-truststore-volume volume.</td>
</tr>
<tr>
<td>SSO_TRUSTSTORE</td>
<td>The name of the truststore file within the secret.</td>
</tr>
<tr>
<td>SSO_TRUSTSTORE_PASSWORD</td>
<td>The password for the truststore and certificate.</td>
</tr>
<tr>
<td>SSO_BEARER_ONLY</td>
<td>Red Hat Single Sign-On client access type.</td>
</tr>
<tr>
<td>SSO_DISABLE_SSL_CERTIFICATE_VALIDATION</td>
<td>If true SSL communication between EAP and the Red Hat Single Sign-On Server is insecure (i.e. certificate validation is disabled with curl)</td>
</tr>
<tr>
<td>SSO_ENABLE_CORS</td>
<td>Enable CORS for Red Hat Single Sign-On applications.</td>
</tr>
</tbody>
</table>

6.2.9. Template variables specific to eap64-sso-s2i and eap71-sso-s2i for automatic client registration with SAML clients

Table 6.9. Configuration Variables For EAP 6.4 and EAP 7 Red Hat Single Sign-On-enabled Applications Built Via S2I Using SAML Protocol

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO_SAML_CERTIFICATE_NAME</td>
<td>The name associated with the server certificate.</td>
</tr>
<tr>
<td>SSO_SAML_KEYSTORE_PASSWORD</td>
<td>The password for the keystore and certificate.</td>
</tr>
<tr>
<td>SSO_SAML_KEYSTORE</td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td>SSO_SAML_KEYSTORE_SECRET</td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td>SSO_SAML_LOGOUT_PAGE</td>
<td>Red Hat Single Sign-On logout page for SAML applications.</td>
</tr>
</tbody>
</table>
### 6.3. EXPOSED PORTS

<table>
<thead>
<tr>
<th>Port Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8443</td>
<td>HTTPS</td>
</tr>
<tr>
<td>8778</td>
<td>Jolokia monitoring</td>
</tr>
</tbody>
</table>