Using Red Hat Single Sign-On for OpenShift
Abstract

Guide to using Red Hat Single Sign-On for OpenShift
Table of Contents

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

2. BEFORE YOU BEGIN ..................................................................... 6
   2.1. COMPARISON: RH-SSO FOR OPENSHIFT IMAGE AND RED HAT SINGLE SIGN-ON 6
   2.2. VERSION COMPATIBILITY AND SUPPORT .................................. 6
   2.3. DEPRECATED IMAGE STREAMS AND APPLICATION TEMPLATES FOR RH-SSO FOR OPENSHIFT 6
   2.4. INITIAL SETUP ..................................................................... 6

3. GET STARTED ................................................................................ 7
   3.1. USING THE RH-SSO FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES 7
   3.2. PREPARING AND DEPLOYING THE RH-SSO FOR OPENSHIFT APPLICATION TEMPLATES 7
      3.2.1. Configuring Keystores ....................................................... 7
      3.2.2. Generating Secrets .......................................................... 8
      3.2.3. Creating Administrator Account for Red Hat Single Sign-On Server 8
      3.2.3.1. Creating RH-SSO Administrator Account via Template Parameters 8
      3.2.3.2. Creating RH-SSO Administrator Account via Remote Shell Session to RH-SSO Pod 10
   3.2.4. Using the OpenShift Web Console ........................................ 11
   3.2.5. Routes ............................................................................ 13
   3.2.6. Deployment Process ........................................................... 13
   3.2.7. RH-SSO Clients ................................................................. 13
   3.2.8. Automatic and Manual RH-SSO Client Registration Methods 14
   3.2.9. Automatic RH-SSO Client Registration .................................. 14
   3.2.10. Manual RH-SSO Client Registration .................................... 15
   3.2.11. Limitations ..................................................................... 16
   3.3. BINARY BUILDS .................................................................... 16
      3.3.1. Deploy Binary Build of EAP 6.4 / 7.0 JSP Service Invocation Application that Authenticates Using Red Hat Single Sign-On 16
      3.3.1.1. Create RH-SSO Realm, Roles, and User for the EAP 6.4 / 7.0 JSP Application 17
      3.3.1.2. Assign user RH-SSO Role to the Realm Management User 18
      3.3.1.3. Prepare RH-SSO Authentication for OpenShift Deployment of the EAP 6.4 / 7.0 JSP Application 18
      3.3.1.4. Deploy Binary Build of the EAP 6.4 / 7.0 JSP Application 19
      3.3.1.5. Access the Application .................................................... 26

4. TUTORIALS .................................................................................... 28
   4.1. EXAMPLE WORKFLOW: PREPARING AND DEPLOYING THE RH-SSO FOR OPENSHIFT IMAGE 28
      4.1.1. Preparing RH-SSO Authentication for OpenShift Deployment 28
      4.1.2. Deploying the RH-SSO Application Template 29
   4.2. EXAMPLE WORKFLOW: UPDATING EXISTING DATABASE WHEN MIGRATING RH-SSO FOR OPENSHIFT IMAGE TO A NEW VERSION 31
      4.2.1. Automatic Database Migration .......................................... 31
      4.2.2. Manual Database Migration .............................................. 33
   4.3. EXAMPLE WORKFLOW: MIGRATING ENTIRE RH-SSO SERVER DATABASE ACROSS THE ENVIRONMENTS 41
      4.3.1. Deploying the RH-SSO MySQL Application Template ............ 41
      4.3.2. (Optional) Creating additional RH-SSO realm and users to be also exported 42
      4.3.3. Export the RH-SSO database as a JSON file on the OpenShift pod 43
      4.3.4. Retrieve and import the exported JSON file 43
   4.4. EXAMPLE WORKFLOW: CONFIGURING OPENSHIFT TO USE RH-SSO FOR AUTHENTICATION 45
      4.4.1. Configuring RH-SSO Credentials ....................................... 45
      4.4.2. Configuring OpenShift Master for Red Hat Single Sign-On Authentication 46
4.4.3. Logging in to OpenShift

4.5. EXAMPLE WORKFLOW: AUTOMATICALLY REGISTERING EAP APPLICATION IN RH-SSO WITH OPENID-CONNECT CLIENT

4.5.1. Preparing RH-SSO Authentication for OpenShift Deployment
4.5.2. Preparing the RH-SSO Credentials
4.5.3. Deploy the RH-SSO-enabled JBoss EAP Image
4.5.4. Log in to the JBoss EAP Server Using RH-SSO

4.6. EXAMPLE WORKFLOW: MANUALLY REGISTERING EAP APPLICATION IN RH-SSO WITH SAML CLIENT

4.6.1. Preparing the RH-SSO Credentials
4.6.2. Preparing RH-SSO Authentication for OpenShift Deployment
4.6.3. Modifying the secure-saml-deployments File
4.6.4. Configuring SAML Client Registration in the Application web.xml
4.6.5. Deploying the Application

CHAPTER 5. REFERENCE

5.1. ARTIFACT REPOSITORY MIRRORS

5.2. ENVIRONMENT VARIABLES

5.2.1. Information Environment Variables
5.2.2. Configuration Environment Variables
5.2.3. Template variables for all RH-SSO images
5.2.4. Template variables specific to sso71-mysql, sso72-mysql, sso71-mysql-persistent, and sso72-mysql-persistent
5.2.5. Template variables specific to sso71-postgresql, sso72-postgresql, sso71-postgresql-persistent, and sso72-postgresql-persistent
5.2.6. Template variables specific to sso71-mysql-persistent, sso72-mysql-persistent, sso71-postgresql-persistent, and sso72-postgresql-persistent
5.2.7. Template variables for general eap64, eap70, and eap71 S2I images
5.2.8. Template variables specific to eap64-sso-s2i, eap70-sso-s2i, and eap71-sso-s2i for automatic client registration
5.2.9. Template variables specific to eap64-sso-s2i, eap70-sso-s2i, and eap71-sso-s2i for automatic client registration with SAML clients

5.3. EXPOSED PORTS
CHAPTER 1. INTRODUCTION

1.1. WHAT IS RED HAT SINGLE SIGN-ON?

Red Hat Single Sign-On (RH-SSO) 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 RH-SSO for OpenShift image.

For RH-SSO 7.2:

- **sso72-https**: RH-SSO 7.2 backed by internal H2 database on the same pod.
- **sso72-mysql**: RH-SSO 7.2 backed by ephemeral MySQL database on a separate pod.
- **sso72-mysql-persistent**: RH-SSO 7.2 backed by persistent MySQL database on a separate pod.
- **sso72-postgresql**: RH-SSO 7.2 backed by ephemeral PostgreSQL database on a separate pod.
- **sso72-postgresql-persistent**: RH-SSO 7.2 backed by persistent PostgreSQL database on a separate pod.

For RH-SSO 7.1:

- **sso71-https**: RH-SSO 7.1 backed by internal H2 database on the same pod.
- **sso71-mysql**: RH-SSO 7.1 backed by ephemeral MySQL database on a separate pod.
- **sso71-mysql-persistent**: RH-SSO 7.1 backed by persistent MySQL database on a separate pod.
- **sso71-postgresql**: RH-SSO 7.1 backed by ephemeral PostgreSQL database on a separate pod.
- **sso71-postgresql-persistent**: RH-SSO 7.1 backed by persistent PostgreSQL database on a separate pod.

Other templates that integrate with RH-SSO are also available:

- **eap64-sso-s2i**: RH-SSO-enabled Red Hat JBoss Enterprise Application Platform 6.4.
- **eap70-sso-s2i**: RH-SSO-enabled Red Hat JBoss Enterprise Application Platform 7.0.
- **eap71-sso-s2i**: RH-SSO enabled Red Hat JBoss Enterprise Application Platform 7.1.
- **datavirt63-secure-s2i**: RH-SSO-enabled Red Hat JBoss Data Virtualization 6.3.

These templates contain environment variables specific to RH-SSO that enable automatic RH-SSO client registration when deployed.
See Automatic and Manual RH-SSO Client Registration Methods for more information.
CHAPTER 2. BEFORE YOU BEGIN

2.1. COMPARISON: RH-SSO FOR OPENSHIFT IMAGE AND RED HAT SINGLE SIGN-ON

The RH-SSO for OpenShift image version number 7.2 is based on Red Hat Single Sign-On 7.2. There are some differences in functionality between the RH-SSO for OpenShift image and Red Hat Single Sign-On:

- The RH-SSO for OpenShift image includes all of the functionality of Red Hat Single Sign-On. In addition, the RH-SSO-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 RH-SSO FOR OPENSIFT

**IMPORTANT**

The RH-SSO for OpenShift image version number 7.0 is deprecated and it will no longer receive updates of image and application templates.

To deploy new applications, it is recommended to use the version 7.1 or 7.2 of the RH-SSO for OpenShift image along with the application templates specific to those 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 RH-SSO for OpenShift image from RH-SSO 7.0 to RH-SSO 7.1, or from RH-SSO 7.1 to RH-SSO 7.2, see the Updating Existing Database when Migrating RH-SSO for OpenShift Image to a new version section.
CHAPTER 3. GET STARTED

3.1. USING THE RH-SSO FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES

Red Hat JBoss Middleware for OpenShift images are pulled on demand from the Red Hat Registry: registry.access.redhat.com. To update to the latest RH-SSO 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 RH-SSO 7.2 resources for OpenShift in the openshift project:

   ```
   $ for resource in sso72-image-stream.json 
   sso72-https.json 
   sso72-mysql-persistent.json 
   sso72-mysql.json 
   sso72-postgresql-persistent.json 
   sso72-postgresql.json
   do
   oc replace -n openshift --force -f 
   https://raw.githubusercontent.com/jboss-openshift/application-templates/ose-v1.4.9/sso/${resource}
   done
   ```

3. Run the following command to install the RH-SSO 7.2 OpenShift image streams in the openshift project:

   ```
   $ oc -n openshift import-image redhat-sso72-openshift:1.0
   ```

3.2. PREPARING AND DEPLOYING THE RH-SSO FOR OPENSHIFT APPLICATION TEMPLATES

3.2.1. Configuring Keystores

The RH-SSO 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 RH-SSO 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

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.

3.2.2. Generating Secrets

OpenShift uses objects called Secrets to hold sensitive information, such as passwords or keystores. See the Secrets chapter in the OpenShift documentation for more information.

The RH-SOO 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:

```bash
$ oc secret new <sso-ssl-secret> <ssl.jks>
$ oc secret new <sso-jgroups-secret> <jgroups.jceks>
```

3.2.3. Creating 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 RH-SOO application template, or

- By a remote shell session to particular RH-SOO pod, if the RH-SOO for OpenShift image is deployed without an application template.

NOTE

Red Hat Single Sign-On allows an initial administrator account creation via the Welcome Page web form. But only if the Welcome Page is accessed from a localhost, this method of administrator account creation is not applicable for RH-SOO for OpenShift image.

3.2.3.1. Creating RH-SOO Administrator Account via Template Parameters
When deploying RH-SSO application template, **SSO_ADMIN_USERNAME** and **SSO_ADMIN_PASSWORD** parameters denote the username and password of the RH-SSO 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 RH-SSO server’s administrator account depends upon the the storage type used to store the RH-SSO server’s database:

- For an in-memory database mode (**sso71-https** and **sso72-https** templates) the account exist throughout the lifecycle of the particular RH-SSO pod (stored account data is lost upon pod destruction),

- For an ephemeral database mode (**sso71-mysql**, **sso71-postgresql**, **sso72-mysql**, and **sso72-postgresql** templates) the account exist throughout the lifecycle of the database pod (even if RH-SSO pod is destructed, the stored account data is preserved under the assumption that the database pod is still running),

- For persistent database mode (**sso71-mysql-persistent**, **sso71-postgresql-persistent**, **sso72-mysql-persistent**, and **sso72-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 RH-SSO and the database pods are destructed.

It is a common practice to deploy an RH-SSO application template to get the corresponding OpenShift deployment config for the application, and then reuse that deployment config multiple times (every time a new RH-SSO 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 RH-SSO applications.
IMPORTANT

Run the following commands to prepare the previously created deployment config of the RH-SSO application for reuse after the administrator account has been created:

1. Identify the deployment config of the RH-SSO application.

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

2. Clear the `SSO_ADMIN_USERNAME` and `SSO_ADMIN_PASSWORD` variables setting.

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

3.2.3.2. Creating RH-SSO Administrator Account via Remote Shell Session to RH-SSO Pod

Run following commands to create administrator account for the `master` realm of the RH-SSO server, when deploying the RH-SSO for OpenShift image directly from the image stream (without the template), after the RH-SSO application pod has been started:

1. Identify the RH-SSO application pod.

```bash
$ 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 RH-SSO for OpenShift container.

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

3. Create the RH-SSO server administrator account for the `master` realm at the command line with the `add-user-keycloak.sh` script.

```bash
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.

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

**WARNING**

When restarting the server it is important to restart just the JBoss EAP process within the running RH-SSO container, and not the whole container. Because restarting the whole container recreates it from scratch, without the RH-SSO server administration account for the master realm to be created.

5. Log into the master realm’s administration console of the RH-SSO server using the the credentials created in the steps above. In the browser, navigate to http://sso-<project-name>.-<hostname>/auth/admin for the RH-SSO web server, or to https://secure-sso-<project-name>.-<hostname>/auth/admin for the encrypted RH-SSO web server, and specify user name and password used to create the administrator user.

### 3.2.4. 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 RH-SSO deployment:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_NAME</td>
<td>The name for the RH-SSO application.</td>
</tr>
<tr>
<td>HOSTNAME_HTTPS</td>
<td>Custom hostname for https service route. Leave blank for default hostname of</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>HOSTNAME_HTTP</strong></td>
<td>Custom hostname for http service route. Leave blank for default hostname of <code>&lt;application-name&gt;..&lt;project&gt;..&lt;default-domain-suffix&gt;</code></td>
</tr>
<tr>
<td><strong>HTTPS_KEYSTORE</strong></td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td><strong>HTTPS_PASSWORD</strong></td>
<td>The password for the keystore and certificate.</td>
</tr>
<tr>
<td><strong>HTTPS_SECRET</strong></td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_KEYSTORE</strong></td>
<td>The name of the JGroups keystore file within the secret.</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_PASSWORD</strong></td>
<td>The password for the JGroups keystore and certificate.</td>
</tr>
<tr>
<td><strong>JGROUPS_ENCRYPT_SECRET</strong></td>
<td>The name of the secret containing the JGroups keystore file.</td>
</tr>
<tr>
<td><strong>SSO_ADMIN_USERNAME</strong></td>
<td>Username of the administrator account for the master realm of the RH-SSO 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>
</tr>
<tr>
<td><strong>SSO_ADMIN_PASSWORD</strong></td>
<td>Password of the administrator account for the master realm of the RH-SSO 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>
</tr>
<tr>
<td><strong>SSO_REALM</strong></td>
<td>The name of an additional RH-SSO realm to create during deployment.</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_USERNAME</strong></td>
<td>RH-SSO service user name to manage the realm.</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_PASSWORD</strong></td>
<td>RH-SSO service user password.</td>
</tr>
</tbody>
</table>

See the [Reference chapter](#) for a more comprehensive list of the RH-SSO environment variables. See the [Example Workflow: Preparing and Deploying the RH-SSO for OpenShift Image](#) for an end-to-end example of RH-SSO deployment.
3.2.5. Routes

The RH-SSO 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 RH-SSO login page.

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

3.2.6. Deployment Process

Once deployed, the sso71-https and sso72-https templates create a single pod that contains both the database and the RH-SSO servers. The sso71-mysql, sso72-mysql, sso71-mysql-persistent, sso72-mysql-persistent, sso71-postgresql, sso72-postgresql, sso71-postgresql-persistent, and sso72-postgresql-persistent templates create two pods, one for the database server and one for the RH-SSO web server.

After the RH-SSO 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 RH-SSO web server, and

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

3.2.7. RH-SSO Clients

Clients are RH-SSO entities that request user authentication. A client can be an application requesting RH-SSO 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.

RH-SSO 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>
...```

### 3.2.8. Automatic and Manual RH-SSO Client Registration Methods

A client application can be automatically registered to an RH-SSO realm by using credentials passed in variables specific to the `eap64-sso-s2i`, `eap70-sso-s2i`, `eap71-sso-s2i`, and `datavirt63-secure-s2i` templates.

Alternatively, you can manually register the client application by configuring and exporting the RH-SSO client adapter and including it in the client application configuration.

### 3.2.9. Automatic RH-SSO Client Registration

Automatic RH-SSO client registration is determined by RH-SSO environment variables specific to the `eap64-sso-s2i`, `eap70-sso-s2i`, `eap71-sso-s2i`, and `datavirt63-secure-s2i` templates. The RH-SSO credentials supplied in the template are then used to register the client to the RH-SSO realm during deployment of the client application.

The RH-SSO environment variables included in the `eap64-sso-s2i`, `eap70-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;.&lt;project&gt;.&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;.&lt;project&gt;.&lt;default-domain-suffix&gt;</code></td>
</tr>
<tr>
<td><code>SSO_REALM</code></td>
<td>The RH-SSO realm created for this procedure.</td>
</tr>
<tr>
<td><code>SSO_USERNAME</code></td>
<td>The name of the realm management user.</td>
</tr>
<tr>
<td><code>SSO_PASSWORD</code></td>
<td>The password of the user.</td>
</tr>
<tr>
<td><code>SSO_PUBLIC_KEY</code></td>
<td>The public key generated by the realm. It is located in the Keys tab of the Realm Settings in the RH-SSO console.</td>
</tr>
</tbody>
</table>
If the RH-SSO 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><strong>SSO_BEARER_ONLY</strong></td>
<td>If set to true, the OpenID Connect client is registered as bearer-only.</td>
</tr>
<tr>
<td><strong>SSO_ENABLE_CORS</strong></td>
<td>If set to true, the RH-SSO adapter enables Cross-Origin Resource Sharing (CORS).</td>
</tr>
</tbody>
</table>

See [Example Workflow: Automatically Registering EAP Application in RH-SSO with OpenID-Connect Client](#) for an end-to-end example of the automatic client registration method using an OpenID-Connect client.

### 3.2.10. Manual RH-SSO Client Registration

Manual RH-SSO 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 RH-SSO web console. The name of this file is different for OpenID-Connect and SAML clients:

<table>
<thead>
<tr>
<th>OpenID-Connect</th>
<th>../configuration/secure-deployments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAML</td>
<td>../configuration/secure-saml-deployments</td>
</tr>
</tbody>
</table>

These files are copied to the RH-SSO adapter configuration section in the `standalone-openshift.xml` at when the application is deployed.

There are two methods for passing the RH-SSO adapter configuration to the client application:
- Modify the deployment file to contain the RH-SSO 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.

See Example Workflow: Manually Configure an Application to Use RH-SSO Authentication, Using SAML Client for an end-to-end example of the manual RH-SSO client registration method using a SAML client.

### 3.2.11. Limitations

OpenShift does not currently accept OpenShift role mapping from external providers. If RH-SSO is used as an authentication gateway for OpenShift, users created in RH-SSO must have the roles added using the OpenShift Administrator oadm policy command.

For example, to allow an RH-SSO-created user to view a project namespace in OpenShift:

```bash
oadm policy add-role-to-user view <user-name> -n <project-name>
```

### 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 / 7.0 JSP Service Invocation Application that Authenticated Using Red Hat Single Sign-On

The following example uses both app-jee-jsp and service-jee-jaxrs quickstarts to deploy EAP 6.4 / 7.0 JSP service application that authenticates using the Red Hat Single Sign-On.

**Prerequisite:**

```
IMPORTANT
This guide assumes the RH-SSO for OpenShift image has been previously deployed using one of the following templates:

- sso71-mysql
- sso72-mysql
- sso71-postgresql
- sso72-postgresql
- sso71-mysql-persistent
- sso72-mysql-persistent
- sso71-postgresql-persistent
- sso72-postgresql-persistent
```
3.3.1.1. Create RH-SSO Realm, Roles, and User for the EAP 6.4 / 7.0 JSP Application

The EAP 6.4 / 7.0 JSP service application requires dedicated RH-SSO realm, username, and password to be able to authenticate using Red Hat Single Sign-On. Perform the following steps after the RH-SSO for OpenShift image has been deployed:

Create the RH-SSO Realm

1. Login to the administration console of the RH-SSO server.
   
   \[\text{https://secure-sso-sso-app-demo.openshift.example.com/auth/admin}\]

   Use the credentials of the RH-SSO administrator user.

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

3. Enter a realm name (this example uses \textbf{demo}) and click \textbf{Create}.

Copy the Public Key

In the newly created \textbf{demo} realm, click the \textbf{Keys} tab and copy the public key that has been generated.

\begin{itemize}
  \item RH-SSO 7.1 and RH-SSO 7.2 images generate two keys by default:
  \begin{itemize}
    \item RSA key, and
    \item HMAC key
  \end{itemize}

  To copy the public key information for the RH-SSO 7.1 or RH-SSO 7.2 image, click the \textbf{Public key} button of the \textbf{RSA} row of the keys table. Then select and copy the content of the pop-up window that appears.
\end{itemize}

The information about the public key is necessary to deploy the RH-SSO-enabled EAP 6.4 / 7.0 JSP application.

Create RH-SSO Roles

\begin{itemize}
  \item The \texttt{service-jee-jaxrs} quickstart exposes three endpoints by the service:
    \begin{itemize}
      \item \texttt{public} - Requires no authentication.
      \item \texttt{secured} - Can be invoked by users with the \texttt{user} role.
      \item \texttt{admin} - Can be invoked by users with the \texttt{admin} role.
    \end{itemize}

  \item Create \texttt{user} and \texttt{admin} roles in RH-SSO. These roles will be assigned to an RH-SSO application user to authenticate access to user applications.
\end{itemize}

1. Click \textbf{Roles} in the \textbf{Configure} sidebar to list the roles for this realm.
NOTE

This is a new realm, so there should only be the default roles:

- `offline_access` and `uma_authorization` role for the RH-SSO 7.1 and RH-SSO 7.2 images.

2. Click **Add Role**.

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

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

**Create the RH-SSO 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 RH-SSO Role to the Realm Management User**

Perform the following steps to tie the previously created **appuser** with the **user** RH-SSO 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.

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

**3.3.1.3. Prepare RH-SSO Authentication for OpenShift Deployment of the EAP 6.4 / 7.0 JSP Application**

1. Create a new project for the EAP 6.4 / 7.0 JSP application.
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.

```bash
$ 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).

      ```bash
      $ 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).

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

   c. Generate the EAP 6.4 / 7.0 for OpenShift secrets with the SSL and JGroup keystore files.

      ```bash
      $ 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.

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

### 3.3.1.4. Deploy Binary Build of the EAP 6.4 / 7.0 JSP Application

1. Clone the source code.

   ```bash
   $ 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.
b. **Comment out** the **app-jee-jsp/config/keycloak.json** requirement of the **maven-enforcer-plugin** plugin and build the **app-jee-jsp** application.

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

mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------------------

[INFO] Building Keycloak Quickstart: service-jee-jaxrs 3.1.0.Final
[INFO] ---------------------------------------------------------------------

...[INFO] -------------------------------------

[INFO] BUILD SUCCESS
[INFO] -------------------------------------

[INFO] Total time: 2.153 s
[INFO] Finished at: 2017-06-26T12:06:12+02:00
[INFO] Final Memory: 25M/241M

-------------------------------------

b. Comment out the **app-jee-jsp/config/keycloak.json** requirement of the **maven-enforcer-plugin** plugin and build the **app-jee-jsp** application.

```bash
$ cd ../app-jee-jsp/

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

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

app-jee-jsp$ mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------------------

[INFO] Building Keycloak Quickstart: app-jee-jsp 3.1.0.Final
[INFO] ---------------------------------------------------------------------

...[INFO] -------------------------------------

[INFO] BUILD SUCCESS
[INFO] -------------------------------------

[INFO] Total time: 3.018 s
[INFO] Final Memory: 35M/310M

-------------------------------------
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 / 7.0 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
   |__ deployments
   
   1 directory, 2 files
   ```
Location of the standard deployments directory depends on the underlying base image, that was used to deploy the application. See the following table:

## Table 3.1. Standard Location of the Deployments Directory

<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 7.0</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 / 7.0 image.

$ oc get is -n openshift | grep eap | cut -d ' ' -f 1
jboss-eap64-openshift
jboss-eap70-openshift
jboss-eap71-openshift

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

**NOTE**

Replace `--image-stream=jboss-eap70-openshift` parameter with the `--image-stream=jboss-eap64-openshift` one in the following oc command to deploy the JSP application on top of JBoss EAP 6.4 for OpenShift image.

$ oc new-build --binary=true \
--image-stream=jboss-eap70-openshift \
--name=eap-app
--> Found image 31895a4 (3 months old) in image stream "openshift/jboss-eap70-openshift" under tag "latest" for "jboss-eap70-openshift"

JBoss EAP 7.0
------------
Platform for building and running JavaEE applications on JBoss EAP 7.0

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
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/ \
--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
```

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

```
eap-app-demo/eap-app-1:aa2574d9
```
Platform for building and running JavaEE applications on JBoss EAP 7.0

Tags: builder, javaee, eap, eap7

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

--> Creating resources ...
   deploymentconfig "eap-app" created
   service "eap-app" created

--> Success
Run 'oc status' to view your app.

9. Stop all running containers of the EAP 6.4 / 7.0 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 / 7.0 JSP application prior the deployment.

   a. Configure the application with proper details about the RH-SSO server instance.

   $ oc set env dc/eap-app \
   -e HOSTNAME_HTTP="eap-app-eap-app-demo.openshift.example.com" \ 
   -e HOSTNAME_HTTPS="secure-eap-app-eap-app-demo.openshift.example.com" \ 
   -e SSO_DISABLE_SSL_CERTIFICATE_VALIDATION="true" \ 
   -e SSO_USERNAME="appuser" \ 
   -e SSO_PASSWORD="apppassword" \ 
   -e SSO_REALM="demo" \ 
   -e SSO_URL="https://secure-sso-sso-app-demo.openshift.example.com/auth" \ 
   -e SSO_PUBLIC_KEY="MIIBIjANBgkqhkiG9w0BAQEFAA0CAQ8AMIIBCgKCAQEAkdhXyKx97oIo06HwnV/MiX2EH055Sn+ydsPzbjIevI5F31UvUco9uA8dGl6oM8HrnaWv+"

  **WARNING**

  Ensure to replace the value of **SSO_PUBLIC_KEY** variable below with the actual content of the RSA public key for the demo realm, that has been copied.
b. Configure the application with details about both the SSL and JGroups keystore.

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

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

```bash
$ 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
```

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

d. Configure the deployment config of the application to run application pods under the default OpenShift service account (default setting).

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

11. Deploy container of the EAP 6.4 / 7.0 JSP application using the modified deployment config.

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

12. Expose the service as route.
$ oc get svc -o name
service/eap-app

$ oc get route
No resources found.

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

$ oc get route
<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>eap-app</td>
<td>eap-app-eap-app-demo.openshift.example.com</td>
<td></td>
</tr>
<tr>
<td>eap-app</td>
<td>8080-tcp</td>
<td></td>
</tr>
</tbody>
</table>

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:

![Login and Invoke Public Buttons](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 RH-SSO server instance against the **demo** realm. Specify username and password of the RH-SSO 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** RH-SSO 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** RH-SSO role:

1. Access the administration console of the RH-SSO server’s instance.  
   `<https://secure-sso-sso-app-demo.openshift.example.com/auth/admin>`

   Use the credentials of the RH-SSO 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 / 7.0 JSP service application.  
   `<http://eap-app-eap-app-demo.openshift.example.com/app-jsp>`

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. TUTORIALS

4.1. EXAMPLE WORKFLOW: PREPARING AND DEPLOYING THE RH-SSO FOR OCP IMAGE

4.1.1. Preparing RH-SSO Authentication for OpenShift Deployment

Log in to the OpenShift CLI with a user that holds the \textit{cluster:admin} role.

1. Create a new project:

   $ oc new-project sso-app-demo

2. Add the \texttt{view} role to the \texttt{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

3. The RH-SSO template requires an SSL keystore and a JGroups keystore.

   \textbf{NOTE}

   OpenShift does not permit login authentication from self-signed certificates. For demonstration purposes, this example uses \texttt{openssl} to generate a CA certificate to sign the SSL keystore and create a truststore. This truststore is also included in the creation of the secret, and specified in the RH-SSO template.

   This example also uses \texttt{keytool}, a package included with the Java Development Kit, to generate self-signed certificates for these keystores.

   \textbf{WARNING}

   For production environments, it's recommended that you use your own SSL certificate purchased from a verified Certificate Authority (CA) for SSL-encrypted connections (HTTPS).

The following commands will prompt for passwords.

a. Generate a CA certificate:

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

b. Generate a Certificate for the SSL keystore:

   ```
$ keytool -genkeypair -keyalg RSA -keysize 2048 -dname "CN=secure-sso-sso-app-demo.openshift32.example.com" -alias sso-https-key -keystore sso-https.jks

c. Generate a Certificate Sign Request for the SSL keystore:

$ keytool -certreq -keyalg rsa -alias sso-https-key -keystore sso-https.jks -file sso.csr

d. Sign the Certificate Sign Request with 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 into the SSL keystore:

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

f. Import the signed Certificate Sign Request into the SSL keystore:

$ keytool -import -file sso.crt -alias sso-https-key -keystore sso-https.jks

g. Import the CA into a new truststore keystore:

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

h. Generate a secure key for the JGroups keystore:

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

4. Create the keystore secrets with the JGroups keystore, and with the SSL keystore and truststore. The truststore is included in this example because it was necessary to sign the example SSL keystore.

$ oc secret new sso-jgroup-secret jgroups.jceks
$ oc secret new sso-ssl-secret sso-https.jks truststore.jks

5. Link the secrets to the default service account:

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

4.1.2. Deploying the RH-SSO Application Template

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

2. Click **Add to Project**, then **Browse Catalog** 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 **Middleware**, then **Integration** to show the desired application template.

4. Select an RH-SSO application template. This example uses **sso72-https**. Specify the application name, hostnames, and SSL keystore, JGroups keystore, and secret information:

<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-sso-app-demo.openshift32.example.com</td>
</tr>
<tr>
<td>HOSTNAME_HTTP</td>
<td>sso-sso-app-demo.openshift32.example.com</td>
</tr>
<tr>
<td>HTTPS_KEYSTORE</td>
<td>sso-https.jks</td>
</tr>
<tr>
<td>HTTPS_PASSWORD</td>
<td>password</td>
</tr>
<tr>
<td>HTTPS_SECRET</td>
<td>sso-ssl-secret</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_KEYSTORE</td>
<td>jgroups.jceks</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_PASSWORD</td>
<td>password</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_SECRET</td>
<td>sso-jgroup-secret</td>
</tr>
</tbody>
</table>

Include the following variables to create a new demorealm realm when the RH-SSO server is deployed and a user to manage that realm.

It is **required** to specify **SSO_ADMIN_USERNAME** and **SSO_ADMIN_PASSWORD** credentials here. If not specified, they are auto generated and displayed as an OpenShift instructional message when the template is instantiated.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO_REALM</td>
<td>demorealm</td>
</tr>
<tr>
<td>SSO_SERVICE_USERNAME</td>
<td>sso-mgmtuser</td>
</tr>
<tr>
<td>SSO_SERVICE_PASSWORD</td>
<td>mgmt-password</td>
</tr>
<tr>
<td>SSO_ADMINUSERNAME</td>
<td>admin</td>
</tr>
<tr>
<td>SSO_ADMINPASSWORD</td>
<td>adm-password</td>
</tr>
</tbody>
</table>
This example uses self-signed certificates and must additionally include information on the truststore. These variables are not required with SSL certificates purchased from a verified Certificate Authority:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO_TRUSTSTORE</td>
<td>truststore.jks</td>
</tr>
<tr>
<td>SSO_TRUSTSTORE_SECRET</td>
<td>sso-ssl-secret</td>
</tr>
<tr>
<td>SSO_TRUSTSTORE_PASSWORD</td>
<td>password</td>
</tr>
</tbody>
</table>

5. Click Create to deploy the application template and start pod deployment. This may take a couple of minutes.

Then access the RH-SSO 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 RH-SSO 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.

---

### 4.2. EXAMPLE WORKFLOW: UPDATING EXISTING DATABASE WHEN MIGRATING RH-SSO FOR OPENSHIFT IMAGE TO A NEW VERSION

**IMPORTANT**

- Rolling updates from RH-SSO for OpenShift 7.0 / 7.1 to 7.2 are not supported as databases and caches are not backward compatible.
- Stop all RH-SSO for OpenShift 7.0 / 7.1 instances 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 7.2 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 RH-SSO 7.2 for the first time.

### 4.2.1. Automatic Database Migration
This process assumes that you are running RH-SSO 7.1 image deployed using one of the following templates:

- **sso71-mysql**
- **sso71-postgresql**
- **sso71-mysql-persistent**
- **sso71-postgresql-persistent**

**IMPORTANT**

Stop all RH-SSO 7.1 pods before upgrading to RH-SSO 7.2, as they cannot run concurrently against the same database.

Use the following steps to automatically migrate the database schema:

1. Identify existing deployment config for RH-SSO 7.1 containers.

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

2. Stop all RH-SSO 7.1 containers 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 RH-SSO 7.2 image.

   ```bash
   $ oc patch dc/sso --type=json -p '[["op": "replace", "path": "/spec/triggers/0/imageChangeParams/from/name", "value": "redhat-sso72-openshift:1.0"]]'"sso" patched
   ```

4. Start rollout of the new RH-SSO 7.2 images based on the latest image defined in the image change triggers.

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

5. Deploy RH-SSO 7.2 containers using the modified deployment config.

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

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

   ```bash
   $ 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'

4.2.2. Manual Database Migration

**IMPORTANT**

Pre-generated scripts are not available. They are generated dynamically depending on the database. With RH-SSO 7.2 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 RH-SSO 7.2 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 RH-SSO 7.2 for further details.

   The database migration process handles the data schema update and performs manipulation of the data, therefore, stop all RH-SSO 7.1 instances before dynamic generation of the SQL migration file.

This guide assumes the RH-SSO 7.1 for OpenShift image has been previously deployed using one of the following templates:

- `sso71-mysql`
- `sso71-postgresql`
- `sso71-mysql-persistent`
- `sso71-postgresql-persistent`

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.

   ```yaml
   $ cat sso71-to-sso72-db-migrate-job.yaml.orig
   apiVersion: batch/v1
   kind: Job
   metadata:
   ```
name: sso71-to-sso72-db-migrate-job
spec:
  autoSelector: true
  parallelism: 0
  completions: 1
template:
  metadata:
    name: sso71-to-sso72-db-migrate-job
spec:
  containers:
  - env:
    - name: DB_SERVICE_PREFIX_MAPPING
      value: <<DB_SERVICE_PREFIX_MAPPING_VALUE>>
    - name: <<PREFIX>>_JNDI
      value: <<PREFIX_JNDI_VALUE>>
    - name: <<PREFIX>>_USERNAME
      value: <<PREFIX_USERNAME_VALUE>>
    - name: <<PREFIX>>_PASSWORD
      value: <<PREFIX_PASSWORD_VALUE>>
    - name: <<PREFIX>>_DATABASE
      value: <<PREFIX_DATABASE_VALUE>>
    - name: TX_DATABASE_PREFIX_MAPPING
      value: <<TX_DATABASE_PREFIX_MAPPING_VALUE>>
    - name: <<SERVICE_HOST>>
      value: <<SERVICE_HOST_VALUE>>
    - name: <<SERVICE_PORT>>
      value: <<SERVICE_PORT_VALUE>>
  image: <<SSO_IMAGE_VALUE>>
  imagePullPolicy: Always
  name: sso71-to-sso72-db-migrate-job
  command: 
    - /bin/bash
    - "-c","/opt/eap/bin/openshift-launch.sh || sleep 600"
  restartPolicy: Never

$ cp sso71-to-sso72-db-migrate-job.yaml.orig sso71-to-sso72-db-migrate-job.yaml

2. Copy the datasource definition and database access credentials from RH-SSO 7.1
   deployment config to appropriate places in database job migration template.
   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 `sso71-to-sso72-db-migrate-
   job.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.

```bash
$ 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="sso71-to-sso72-db-migrate-job.yaml"

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

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

# Substitute (the order in which replacements are made is
# important):
# * <<PREFIX>> with actual $PREFIX value and
# * <<PREFIX with "<$PREFIX" value
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
debare -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
# 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. 
# if grep -Pq "${SERVICE}_SERVICE_HOST=[^ ]+
## "$SSO_DC_VARS[@]" 
# && 
# grep -Pq "${SERVICE}_SERVICE_PORT=[^ ]+
## "$SSO_DC_VARS[@]"
# then 
# DB_VARS=( "$DB_VARS[@]" ${SERVICE}_SERVICE_HOST 
# ${SERVICE}_SERVICE_PORT ) 
# If they are not defined, delete their placeholder rows in yaml 
# definition file 
# (since if not defined they are not expanded which make the yaml 
# definition invalid). 
# else 
# for KEY in "HOST" "PORT" 
# do 
# sed -i "/SERVICE_${KEY}/d" ${JOB_MIGRATION_YAML} 
# done 
# fi \\
# Substitute (the order in which replacements are made is 
# important): 
# # * <<SERVICE_HOST>> with ${SERVICE}_SERVICE_HOST and 
# # * <<SERVICE_HOST_VALUE>> with "<<SERVICE_HOST_VALUE>>" 
# # Do this for both "HOST" and "PORT" 
# for KEY in "HOST" "PORT" 
# do 
# sed -i "/<SERVICE_${KEY}/d" ${JOB_MIGRATION_YAML} 
# sed -i "/<SERVICE_${KEY}_VALUE>/d" ${JOB_MIGRATION_YAML} 
# done 
# Propagate the values of the datasource array items into yaml 
# definition of the 
# migration job 
# for VAR in "$SSO_DC_VARS[@]"
# do 
# IFS=":=" read KEY VALUE <<< $VAR 
# if grep -q $KEY <<< ${DB_VARS[@]}
# then 
# KEY="_VALUE" 
# # Enwrap integer port value with double quotes 
# if [[ $KEY =~ ${SERVICE}_SERVICE_PORT_VALUE ]];
# then 
# sed -i "s<${KEY}>({VALUE})"{JOB_MIGRATION_YAML} 
# # Character values do not need quotes 
# else 
# sed -i "s<${KEY}>{VALUE}"{JOB_MIGRATION_YAML} 
# fi 
# # Verify that the value has been successfully propagated.
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 RH-SSO 7.2 database migration image using the pre-configured source and wait for the build to finish.

```bash
$ oc get is -n openshift | grep sso72 | cut -d ' ' -f1
redhat-sso72-openshift

$ oc new-build redhat-sso72-openshift:1.0~https://github.com/jboss-openshift/openshift-examples --context-dir=sso-manual-db-migration -name=sso72-db-migration-image
--> Found image bf45ac2 (7 days old) in image stream
"openshift/redhat-sso72-openshift" under tag "1.0" for "redhat-sso72-openshift:1.0"

Red Hat SSO 7.2
--------------
Platform for running Red Hat SSO
Tags: sso, sso7, keycloak

* A source build using source code from
https://github.com/jboss-openshift/openshift-examples will be created
* The resulting image will be pushed to image stream "sso72-db-migration-image:latest"
* Use 'start-build' to trigger a new build

--> Creating resources with label build=sso72-db-migration-image ...
```
4. Update the template of the database migration job (sso71-to-sso72-db-migrate-job.yaml) with reference to the built sso72-db-migration-image image.

a. Get the docker pull reference for the image.

   ```bash
   $ PULL_REF=$(oc get istag -n $(oc project -q) --no-headers | grep sso72-db-migration-image | tr -s ' ' | cut -d ' ' -f 2)
   ```

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

   ```bash
   $ sed -i "s/<<SSO_IMAGE_VALUE>>#$PULL_REF#g" sso71-to-sso72-db-migrate-job.yaml
   ```

c. Verify that the field is updated.

5. Instantiate database migration job from the job template.

   ```bash
   $ oc create -f sso71-to-sso72-db-migrate-job.yaml
   job "sso71-to-sso72-db-migrate-job" created
   ```

**IMPORTANT**

The database migration process handles the data schema update and performs manipulation of the data, therefore, stop all RH-SSO 7.1 instances before dynamic generation of the SQL migration file.

6. Identify existing deployment config for RH-SSO 7.1 containers.

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

7. Stop all RH-SSO 7.1 containers in the current namespace.

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

8. Run the database migration job and wait for the pod to be running correctly.
9. Get the dynamically generated SQL database migration file from the pod.

   $ mkdir -p ./db-update
   $ oc rsync sso71-to-sso72-db-migrate-job-b87bb:/opt/eap/keycloak-database-update.sql ./db-update
   receiving incremental file list
   keycloak-database-update.sql
   sent 30 bytes  received 29,726 bytes  59,512.00 bytes/sec
   total size is 29,621  speedup is 1.00

10. Inspect the **keycloak-database-update.sql** file for changes to be performed within manual RH-SSO 7.2 database update.

11. Apply the database update manually.

   - Run the following commands for **sso71-postgresql** and **sso71-postgresql-persistent** templates (PostgreSQL database):

     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.

```bash
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
...
```

**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.

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

- Run the following commands for `sso71-mysql` and `sso71-mysql-persistent` templates (MySQL database):

  i. Given pod situation similar to the following:

  ```bash
  $ oc get pods
  NAME                                  READY     STATUS
  RESTARTS   AGE
  sso-mysql-1-zvhk3                     1/1       Running
  0          1h
  sso71-to-sso72-db-migrate-job-m202t   1/1       Running     0
  11m
  sso72-db-migration-image-1-build      0/1       Completed   0
  13m
  ```

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

  ```bash
  $ oc rsync --no-perms=true ./db-update/ sso-mysql-1-zvhk3:/tmp
  sending incremental file list
  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.

  ```bash
  $ oc rsh sso-mysql-1-zvhk3
  sh-4.2$
  ```
iv. Use the `mysql` tool to apply database update manually.

```bash
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 image change trigger in the existing deployment config of RH-SSO 7.1 to reference the RH-SSO 7.2 image.

```bash
$ oc patch dc/sso --type=json -p '[{"op": "replace", "path": "/spec/triggers/0/imageChangeParams/from/name", "value": "redhat-sso72-openshift:1.0"}]'
"sso" patched
```

13. Start rollout of the new RH-SSO 7.2 images based on the latest image defined in the image change triggers.

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

14. Deploy RH-SSO 7.2 containers using the modified deployment config.

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

### 4.3. EXAMPLE WORKFLOW: MIGRATING ENTIRE RH-SSO 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 RH-SSO Authentication for OpenShift Deployment section have been performed already.

#### 4.3.1. Deploying the RH-SSO 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 `sso72-mysql` RH-SSO 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 7.2 database **is triggered at RH-SSO server boot time and its parameters are passed in via Java system properties**. This means during one RH-SSO 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 RH-SSO for OpenShift image, a database import is performed in order to create the RH-SSO 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 RH-SSO web console at [https://secure-sso-<sso-app-demo>.<openshift32.example.com>/auth/admin](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 RH-SSO 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.

4.3.2. (Optional) Creating additional RH-SSO realm and users to be also exported

When performing **Red Hat Single Sign-On 7.2 server database export** only RH-SSO realms and users currently present in the database will be exported. If the exported JSON file should include also additional RH-SSO 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.

### 4.3.3. Export the RH-SSO database as a JSON file on the OpenShift pod

1. Get the RH-SSO deployment config and scale it down to zero.

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

2. Instruct the RH-SSO 7.2 server deployed on RH-SSO for OpenShift image to perform database export at RH-SSO server boot time.

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

3. Scale the RH-SSO deployment config back up. This will start the RH-SSO 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'
   (ServerService Thread Pool -- 57) Exporting model into file
   /tmp/demorealm-export.json
   09:24:59,998 INFO  [org.keycloak.services] (ServerService Thread
   Pool -- 57) KC-SERVICES0035: Export finished successfully
   ```

### 4.3.4. Retrieve and import the exported JSON file

1. Retrieve the JSON file of the RH-SSO 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 .
   ```
2. (Optional) Import the JSON file of the RH-SSO database into an RH-SSO server running in another environment.

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

Use the administration console of the RH-SSO server to import the resources from previously exported JSON file into the RH-SSO server’s database, when the RH-SSO server is running as a Red Hat Single Sign-On 7.2 container on OpenShift:

a. Log into the master realm’s administration console of the RH-SSO server using the credentials used to create the administrator user. In the browser, navigate to http://sso-<project-name>.<hostname>/auth/admin for the RH-SSO web server, or to https://secure-sso-<project-name>.<hostname>/auth/admin for the encrypted RH-SSO web server.

b. At the top of the sidebar choose the name of the RH-SSO 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 RH-SSO 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:

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 RH-SSO server, on which the export JSON file was created, to the target RH-SSO 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.

IMPORTANT
The administration console import allows you to overwrite resources if you choose (Overwrite strategy). On a production system use this feature with caution.

4.4. EXAMPLE WORKFLOW: CONFIGURING OPENSSHIFT TO USE RH-SSO FOR AUTHENTICATION

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

This example adds RH-SSO 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.

4.4.1. Configuring RH-SSO Credentials
Log in to the encrypted RH-SSO web server at https://secure-sso-sso-app-demo.openshift32.example.com/auth/admin using the administrator account created during the RH-SSO 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 OpenShift) and click Create.

Create a User

Create a test user that can be used to demonstrate the RH-SSO-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 usestestuser) 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.

4.4.2. 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
   
   Add RH-SSO as a secondary identity provider with content similar to the following snippet:

   ```yaml
   - name: rh_sso
     challenge: false
     login: true
     mappingInfo: 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:
     - preferredUsername:
     - name:
     - email:

   a. The RH-SSO Secret hash for the `clientSecret` can be found in the RH-SSO web console: **Clients → openshift-demo → Credentials**

   b. The endpoints for the `urls` can be found by making a request with the RH-SSO application. For example:

   ```bash
   curl -k https://secure-sso-sso-app-demo.example.com```
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:

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

### 4.4.3. 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 RH-S SO. 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:

```bash
$ oadm policy add-role-to-user view testuser -n sso-app-demo
```

### 4.5. EXAMPLE WORKFLOW: AUTOMATICALLY REGISTERING EAP APPLICATION IN RH-SSO WITH OPENID-CONNECT CLIENT

This follows on from Example Workflow: Preparing and Deploying the RH-SSO for OpenShift image, in which RH-SSO was deployed on OpenShift. This example prepares RH-SSO realm, role, and user credentials for an EAP project using an OpenID-Connect client adapter. These credentials are then provided in the EAP for OpenShift template for automatic RH-SSO client registration. Once deployed, the RH-SSO 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 RH-SSO Clients and Automatic and Manual RH-SSO Client Registration Methods for more information on the differences between OpenID-Connect and SAML clients.

### 4.5.1. Preparing RH-SSO 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

### 4.5.2. Preparing the RH-SSO Credentials

Log in to the encrypted RH-SSO web server at `https://secure-sso-<project-name>.<hostname>/auth/admin` using the administrator account created during the RH-SSO deployment.

**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 RH-SSO-enabled JBoss EAP image.
Create a Role

Create a role in RH-SSO 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 RH-SSO 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 RH-SSO client registration in the RH-SSO 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 RH-SSO 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.

4.5.3. Deploy the RH-SSO-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 RH-SSO parameters to configure the RH-SSO 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><code>&lt;realm-public-key&gt;</code></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.
4.5.4. Log in to the JBoss EAP Server Using RH-SSO

1. Access the JBoss EAP application server and click Login. You are redirected to the RH-SSO login.

2. Log in using the RH-SSO user created in the example. You are authenticated against the RH-SSO server and returned to the JBoss EAP application server.

4.6. EXAMPLE WORKFLOW: MANUALLY REGISTERING EAP APPLICATION IN RH-SSO WITH SAML CLIENT

This follows on from Example Workflow: Preparing and Deploying the RH-SSO for OpenShift image, in which RH-SSO was deployed on OpenShift.

This example prepares RH-SSO realm, role, and user credentials for an EAP project and configures an EAP for OpenShift deployment. Once deployed, the RH-SSO 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 RH-SSO Clients and Automatic and Manual RH-SSO Client Registration Methods for more information on the differences between SAML and OpenID-Connect clients.

4.6.1. Preparing the RH-SSO Credentials

Log in to the encrypted RH-SSO web server at https://secure-sso-<project-name>.<hostname>/auth/admin using the administrator account created during the RH-SSO 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 RH-SSO-enabled JBoss EAP image.

Create a Role

Create a role in RH-SSO 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 an RH-SSO 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 RH-SSO client registration in the RH-SSO 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 userapp-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 RH-SSO 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.
5. Click Save.

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-samldeployments**.

### 4.6.2. Preparing RH-SSO 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
   ```

4.6.3. Modifying the secure-saml-deployments File

The `keycloak-saml-subsystem.xml`, exported from the RH-SSO 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 RH-SSO 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 RH-S SO client.

5. Delete the second <CertificatePem> tag and key and replace it with the the realm certificate information:

```
...<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 RH-S SO client.

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

### 4.6.4. 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>
...
```

### 4.6.5. Deploying the Application

You do not need to include any RH-S SO configuration for the image because that has been configured in the application itself. Navigating to the application login page redirects you to the RH-S SO login. Log in to the application through RH-S SO using the application user user created earlier.
5.1. 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:

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

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

   ```bash
   oc env bc/sso
   MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"
   buildconfig "sso" updated
   ```

3. Verify the setting

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

4. Schedule new build of the application

   ```bash
   ```

   **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.
5.2. ENVIRONMENT VARIABLES

5.2.1. 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 5.1. Information Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB_JOLOKIA_AUTH_OPENS HIFT</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_HTTPS</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_PASSWORD_RANDOM</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>JBOSS_IMAGE_NAME</td>
<td>Image name, same as Name label.</td>
<td>redhat-sso-7/sso71-openshift or redhat-sso-7/sso72-openshift</td>
</tr>
<tr>
<td>JBOSS_IMAGE_RELEASE</td>
<td>Image release, same as Release label.</td>
<td>dev</td>
</tr>
<tr>
<td>JBOSS_IMAGE_VERSION</td>
<td>Image version, same as Version label.</td>
<td>1.0</td>
</tr>
<tr>
<td>JBOSS_MODULES_SYSTEM_PKGS</td>
<td></td>
<td>org.jboss.logmanager,jdk.nashorn.api</td>
</tr>
<tr>
<td>STI_BUILDER</td>
<td>Provides OpenShift S2I support for jee project types.</td>
<td>jee</td>
</tr>
</tbody>
</table>

5.2.2. 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 5.2. Configuration Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></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_AUTH_OPENS HIFT</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>AB_JOLOKIA_CONFIG</td>
<td>If set uses this file (including path) as Jolokia JVM agent properties (as described in Jolokia's reference manual). If not set, the /opt/jolokia/etc/jolokia.properties file will be created using the settings as defined in this document, otherwise the rest of the settings in this document are ignored.</td>
<td>/opt/jolokia/custom.properties</td>
</tr>
<tr>
<td>AB_JOLOKIA_DISCOVERY_ENABED</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 certificates are generated if no serverCert configuration is given in AB_JOLOKIA_OPTS. NOTE: If the values is set to an empty string, https is turned off. If the value is set to a non empty string, https is turned on.</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_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 default, Jolokia is enabled. NOTE: If the values is set to an empty string, https is turned off. If the value is set to a non empty string, https is turned on.</td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_OPTS</td>
<td>Additional options to be appended to the agent configuration. They should be given in the format &quot;key=value, key=value, ...&quot;</td>
<td>backlog=20</td>
</tr>
<tr>
<td>AB_JOLOKIA_PASSWORD</td>
<td>Password for basic authentication. By default authentication is switched off.</td>
<td>mypassword</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>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</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 on the maximal heap memory. The default is 100 which means 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 <code>-Xms</code> option is added.</td>
<td>100</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>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>JAVA_OPTS_APPEND</strong></td>
<td>Server startup options.</td>
<td><code>-Dkeycloak.migration.action=export -Dkeycloak.migration.provider=dir -Dkeycloak.migration.dir=/tmp</code></td>
</tr>
<tr>
<td><strong>MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION</strong></td>
<td>For backwards compatibility, 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><strong>OPENSHEET_KUBE_PING_LABELS</strong></td>
<td>Clustering labels selector.</td>
<td>app=sso-app</td>
</tr>
<tr>
<td><strong>OPENSHEET_KUBE_PING_NAMESPACE</strong></td>
<td>Clustering project namespace.</td>
<td>myproject</td>
</tr>
<tr>
<td><strong>SCRIPT_DEBUG</strong></td>
<td>If set to true, ensures that the bash scripts are executed with the -x option, printing the commands and their arguments as they are executed.</td>
<td>true</td>
</tr>
<tr>
<td><strong>SSO_ADMIN_PASSWORD</strong></td>
<td>Password of the administrator account for the master realm of the RH-SSO server. Required. 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><strong>SSO_ADMIN_USERNAME</strong></td>
<td>Username of the administrator account for the master realm of the RH-SSO server. Required. 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>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>SSO_REALM</strong></td>
<td>Name of the realm to be created in the RH-SSO server if this environment variable is provided.</td>
<td>demo</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_PASSWORD</strong></td>
<td>The password for the RH-SSO service user.</td>
<td>mgmt-password</td>
</tr>
<tr>
<td><strong>SSO_SERVICE_USERNAME</strong></td>
<td>The username used to access the RH-SSO service. This is used by clients to create the application client(s) within the specified RH-SSO 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 RH-SSO 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 RH-SSO-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 RH-SSO-enabled applications, is described below.

### 5.2.3. Template variables for all RH-SSO images

Table 5.3. Configuration Variables Available For All RH-SSO Images
<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>DB_MAX_POOL_SIZE</strong></td>
<td>Sets xa-pool/max-pool-size for the configured datasource.</td>
</tr>
<tr>
<td><strong>DB_TX_ISOLATION</strong></td>
<td>Sets transaction-isolation for the configured datasource.</td>
</tr>
<tr>
<td><strong>DB_USERNAME</strong></td>
<td>Database user name.</td>
</tr>
<tr>
<td><strong>HOSTNAME_HTTP</strong></td>
<td>Custom hostname for http service route. Leave blank for default hostname, e.g.: <code>&lt;application-name&gt;</code>.</td>
</tr>
<tr>
<td><strong>HOSTNAME_HTTPS</strong></td>
<td>Custom hostname for https service route. Leave blank for default hostname, e.g.: <code>&lt;application-name&gt;</code>.</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 $JBOSS_HOME/standalone/configuration.</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_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>
<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>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>JGROUPS_CLUSTER_PASSWORD</code></td>
<td>JGroups cluster password.</td>
</tr>
<tr>
<td><code>JGROUPS_ENCRYPT_KEYSTORE</code></td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td><code>JGROUPS_ENCRYPT_NAME</code></td>
<td>The name associated with the server certificate (e.g. <code>secret-key</code>).</td>
</tr>
<tr>
<td><code>JGROUPS_ENCRYPT_PASSWORD</code></td>
<td>The password for the keystore and certificate (e.g. <code>password</code>).</td>
</tr>
<tr>
<td><code>JGROUPS_ENCRYPT_SECRET</code></td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td><code>SSO_ADMIN_USERNAME</code></td>
<td>Username of the administrator account for the master realm of the RH-SSO 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><code>SSO_ADMIN_PASSWORD</code></td>
<td>Password of the administrator account for the master realm of the RH-SSO 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><code>SSO_REALM</code></td>
<td>Name of the realm to be created in the RH-SSO server if this environment variable is provided.</td>
</tr>
<tr>
<td><code>SSO_SERVICE_USERNAME</code></td>
<td>The username used to access the RH-SSO service. This is used by clients to create the application client(s) within the specified RH-SSO realm. This user is created if this environment variable is provided.</td>
</tr>
<tr>
<td><code>SSO_SERVICE_PASSWORD</code></td>
<td>The password for the RH-SSO service user.</td>
</tr>
<tr>
<td><code>SSO_TRUSTSTORE</code></td>
<td>The name of the truststore file within the secret.</td>
</tr>
<tr>
<td><code>SSO_TRUSTSTORE_SECRET</code></td>
<td>The name of the secret containing the truststore file. Used for <code>sso-truststore-volume</code> volume.</td>
</tr>
</tbody>
</table>
The password for the truststore and certificate.

### 5.2.4. Template variables specific to sso71-mysql, sso72-mysql, sso71-mysql-persistent, and sso72-mysql-persistent

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSO_TRUSTSTORE_PASSWORD</strong></td>
<td>The password for the truststore and certificate.</td>
</tr>
</tbody>
</table>

#### Table 5.4. Configuration Variables Specific To RH-SSO-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.</td>
</tr>
<tr>
<td></td>
<td>java:/jboss/datasources/mysql.</td>
</tr>
<tr>
<td><strong>MYSQL_AIO</strong></td>
<td>Controls the innodb_use_native_aio 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><strong>MYSQL_FT_MIN_WORD_LEN</strong></td>
<td>The minimum length of the word to be included in a FULLTEXT index.</td>
</tr>
<tr>
<td><strong>MYSQL_LOWER_CASE_TABLE_NAMES</strong></td>
<td>Sets how the table names are stored and compared.</td>
</tr>
<tr>
<td><strong>MYSQL_MAX_CONNECTIONS</strong></td>
<td>The maximum permitted number of simultaneous client connections.</td>
</tr>
</tbody>
</table>

### 5.2.5. Template variables specific to sso71-postgresql, sso72-postgresql, sso71-postgresql-persistent, and sso72-postgresql-persistent

<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>
</tbody>
</table>
5.2.6. Template variables specific to sso71-mysql-persistent, sso72-mysql-persistent, sso71-postgresql-persistent, and sso72-postgresql-persistent

Table 5.6. Configuration Variables Specific To RH-SSO-enabled MySQL / PostgreSQL Applications With Persistent Storage

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

5.2.7. Template variables for general eap64, eap70, and eap71 S2I images

Table 5.7. 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>APPLICATION_NAME</td>
<td>The name for the application.</td>
</tr>
<tr>
<td>ARTIFACT_DIR</td>
<td>Artifacts directory.</td>
</tr>
<tr>
<td>AUTO_DEPLOY_EXPLODED</td>
<td>Controls whether exploded deployment content should be automatically deployed.</td>
</tr>
<tr>
<td>CONTEXT_DIR</td>
<td>Path within Git project to build; empty for root project directory.</td>
</tr>
<tr>
<td>GENERIC_WEBHOOK_SECRET</td>
<td>Generic build trigger secret.</td>
</tr>
<tr>
<td>GITHUB_WEBHOOK_SECRET</td>
<td>GitHub trigger secret.</td>
</tr>
<tr>
<td>HORNETQ_CLUSTER_PASSWORD</td>
<td>HornetQ cluster administrator password.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HORNETQ_QUEUES</td>
<td>Queue names.</td>
</tr>
<tr>
<td>HORNETQ_TOPICS</td>
<td>Topic names.</td>
</tr>
<tr>
<td>HOSTNAME_HTTP</td>
<td>Custom host name for http service route. Leave blank for default host name, e.g.: &lt;application-name&gt;.&lt;project&gt;.&lt;default-domain-suffix&gt;.</td>
</tr>
<tr>
<td>HTTPS_KEYSTORE_TYPE</td>
<td>The type of the keystore file (JKS or JCEKS).</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_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 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>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JGROUPS_ENCRYPT_NAME</td>
<td>The name associated with the server certificate (e.g. <code>secret-key</code>).</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_PASSWORD</td>
<td>The password for the keystore and certificate (e.g. <code>password</code>).</td>
</tr>
<tr>
<td>JGROUPS_ENCRYPT_SECRET</td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td>SOURCE_REPOSITORY_REF</td>
<td>Git branch/tag reference.</td>
</tr>
<tr>
<td>SOURCE_REPOSITORY_URL</td>
<td>Git source URI for application.</td>
</tr>
</tbody>
</table>

#### 5.2.8. Template variables specific to eap64-sso-s2i, eap70-sso-s2i, and eap71-sso-s2i for automatic client registration

**Table 5.8. Configuration Variables For EAP 6.4 and EAP 7 RH-SSO-enabled Applications Built Via S2I**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO_URL</td>
<td>RH-SSO server location.</td>
</tr>
<tr>
<td>SSO_REALM</td>
<td>Name of the realm to be created in the RH-SSO server if this environment variable is provided.</td>
</tr>
<tr>
<td>SSO_USERNAME</td>
<td>The username used to access the RH-SSO service. This is used to create the application client(s) within the specified RH-SSO realm. This should match the <code>SSO_SERVICE_USERNAME</code> specified through one of the <code>sso71-</code> or <code>sso72-</code> templates.</td>
</tr>
<tr>
<td>SSO_PASSWORD</td>
<td>The password for the RH-SSO service user.</td>
</tr>
<tr>
<td>SSO_PUBLIC_KEY</td>
<td>RH-SSO 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 RH-SSO client secret for confidential access.</td>
</tr>
<tr>
<td>SSO_SERVICE_URL</td>
<td>RH-SSO service location.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>SSO_TRUSTSTORE_SECRET</code></td>
<td>The name of the secret containing the truststore file. Used for <code>sso-truststore-volume</code> volume.</td>
</tr>
<tr>
<td><code>SSO_TRUSTSTORE</code></td>
<td>The name of the truststore file within the secret.</td>
</tr>
<tr>
<td><code>SSO_TRUSTSTORE_PASSWORD</code></td>
<td>The password for the truststore and certificate.</td>
</tr>
<tr>
<td><code>SSO_BEARER_ONLY</code></td>
<td>RH-SSO client access type.</td>
</tr>
<tr>
<td><code>SSO_DISABLE_SSL_CERTIFICATE_VALIDATION</code></td>
<td>If true SSL communication between EAP and the RH-SSO Server is insecure (i.e. certificate validation is disabled with curl)</td>
</tr>
<tr>
<td><code>SSO_ENABLE_CORS</code></td>
<td>Enable CORS for RH-SSO applications.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SSO_SAML_CERTIFICATE_NAME</code></td>
<td>The name associated with the server certificate.</td>
</tr>
<tr>
<td><code>SSO_SAML_KEYSTORE_PASSWORD</code></td>
<td>The password for the keystore and certificate.</td>
</tr>
<tr>
<td><code>SSO_SAML_KEYSTORE</code></td>
<td>The name of the keystore file within the secret.</td>
</tr>
<tr>
<td><code>SSO_SAML_KEYSTORE_SECRET</code></td>
<td>The name of the secret containing the keystore file.</td>
</tr>
<tr>
<td><code>SSO_SAML_LOGOUT_PAGE</code></td>
<td>RH-SSO logout page for SAML applications.</td>
</tr>
</tbody>
</table>

### 5.2.9. Template variables specific to eap64-sso-s2i, eap70-sso-s2i, and eap71-sso-s2i for automatic client registration with SAML clients

Table 5.9. Configuration Variables For EAP 6.4 and EAP 7 RH-SSO-enabled Applications Built Via S2I Using SAML Protocol

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