Red Hat Single Sign-On 7.0 Server Installation and Configuration Guide

Server Installation and Configuration Guide

Red Hat Customer Content Services

Server Installation and Configuration Guide

Legal Notice

Abstract

This guide consist of information to install and configure Red Hat Single Sign-On 7.0

Table of Contents

CHAPTER 1. GUIDE OVERVIEW

The purpose of this guide is to walk through the steps that need to be completed prior to booting up the Red Hat Single Sign-On server for the first time. If you just want to test drive Red Hat Single Sign-On, it pretty much runs out of the box with its own embedded and local-only database. For actual deployments that are going to be run in production you'll need to decide how you want to manage server configuration at runtime (standalone or domain mode), configure a shared database for Red Hat Single Sign-On storage, set up encryption and HTTPS, and finally set up Red Hat Single Sign-On to run in a cluster. This guide walks through each and every aspect of any pre-boot decisions and setup you must do prior to deploying the server.

One thing to particularly note is that Red Hat Single Sign-On is derived from the JBoss EAP Application Server. Many aspects of configuring Red Hat Single Sign-On revolve around JBoss EAP configuration elements. Often this guide will direct you to documentation outside of the manual if you want to dive into more detail.

1.1. RECOMMENDED ADDITIONAL EXTERNAL DOCUMENTATION

Red Hat Single Sign-On is built on top of the JBoss EAP application server and it's sub-projects like Infinispan (for caching) and Hibernate (for persistence). This guide only covers basics for infrastructure-level configuration. It is highly recommended that you peruse the documentation for JBoss EAP and its sub projects. Here is the link to the documentation:

- JBoss EAP Configuration Guide

CHAPTER 2. INSTALLATION

Installing Red Hat Single Sign-On is as simple as downloading it and unzipping it. This chapter reviews system requirements as well as the directory structure of the distribution.

2.1. SYSTEM REQUIREMENTS

These are the requirements to run the Red Hat Single Sign-On authentication server:

- Can run on any operating system that runs Java
- Java 8 JDK
- zip or gzip and tar
- At least 512M of RAM
- At least 1G of diskspace
- A shared external database like Postgres, MySql, Oracle, etc. Red Hat Single Sign-On requires an external shared database if you want to run in a cluster. Please see the database configuration section of this guide for more information.
- Network multicast support on your machine if you want to run in a cluster. Red Hat Single Sign-On can be clustered without multicast, but this requires a bunch of configuration changes. Please see the clustering section of this guide for more information.

2.2. INSTALLING DISTRIBUTION FILES

The Red Hat Single Sign-On Server is contained in one distribution file:

- 'rh-sso-7.0.0.[zip|tar.gz]'

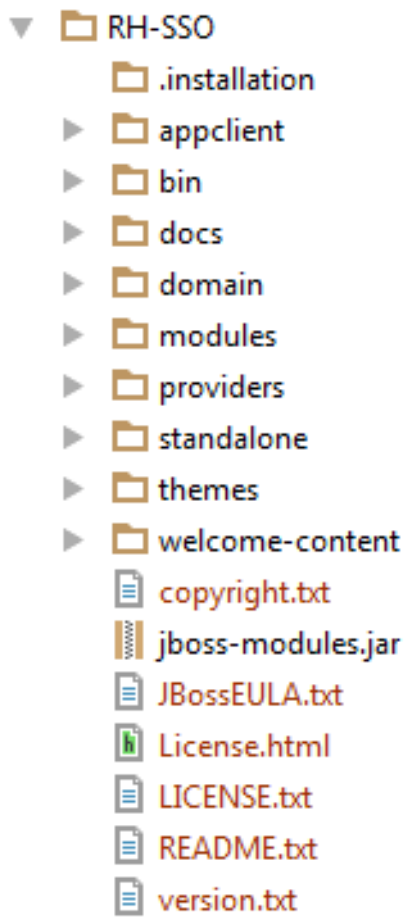
The 'rh-sso-7.0.0.[zip|tar.gz]' file is the server only distribution. It contains nothing other than the scripts and binaries to run the Red Hat Single Sign-On Server.

To unpack of these files run the **`unzip`** or **`gunzip`** and **`tar`** utilities.

2.3. DISTRIBUTION DIRECTORY STRUCTURE

This chapter walks you through the directory structure of the server distribution.

distribution directory structure



Let's examine some of the purposes of each directory

bin/

This contains various scripts to either boot the server or perform some other management action on the server.

domain/

This contains configuration files and working directory when running Red Hat Single Sign-On in domain mode.

modules/

These are all the Java libraries used by the server.

standalone/

This contains configuration files and working directory when running Red Hat Single Sign-On in standalone mode.

themes/

This directory contains all the html, style sheets, javascript files, and images used to display any UI screen displayed by the server. Here you can modify an existing theme or create your own. See the Server Developer Guide for more information on this.

CHAPTER 3. CHOOSING AN OPERATING MODE

Before deploying Red Hat Single Sign-On in a production environment you need to decide which type of operating mode you are going to use. Will you run Red Hat Single Sign-On within a cluster? Do you want a centralized way to manage your server configurations? Your choice of operating mode effects how you configure databases, configure caching and even how you boot the server.

Tip

The Red Hat Single Sign-On is built on top of the JBoss EAP Application Server. This guide will only go over the basics for deployment within a specific mode. If you want specific information on this, a better place to go would be the JBoss EAP Configuration Guide

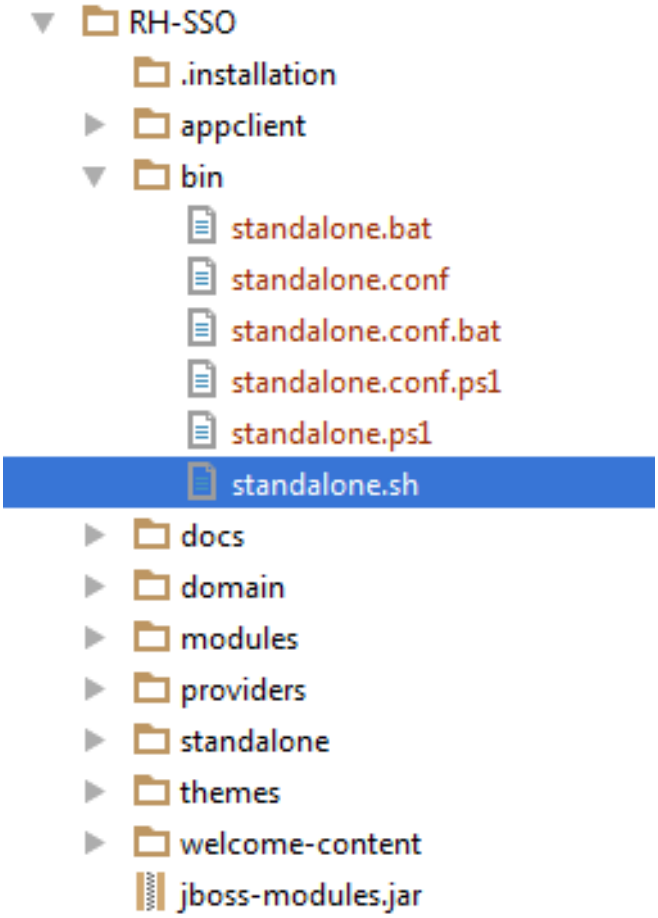
3.1. STANDALONE MODE

Standalone operating mode is only useful when you want to run one, and only one Red Hat Single Sign-On server instance. It is not usable for clustered deployments and all caches are non-distributed and local-only. It is not recommended that you use standalone mode in production as you will have a single point of failure. If your standalone mode server goes down, users will not be able to log in. This mode is really only useful to test drive and play with the features of Red Hat Single Sign-On

3.1.1. Standalone Boot Script

When running the server in standalone mode, there is a specific script you need to run to boot the server depending on your operating system. These scripts live in the *bin/* directory of the server distribution.

Standalone Boot Scripts



To boot the server:

Linux/Unix

```
$ .../bin/standalone.sh
```

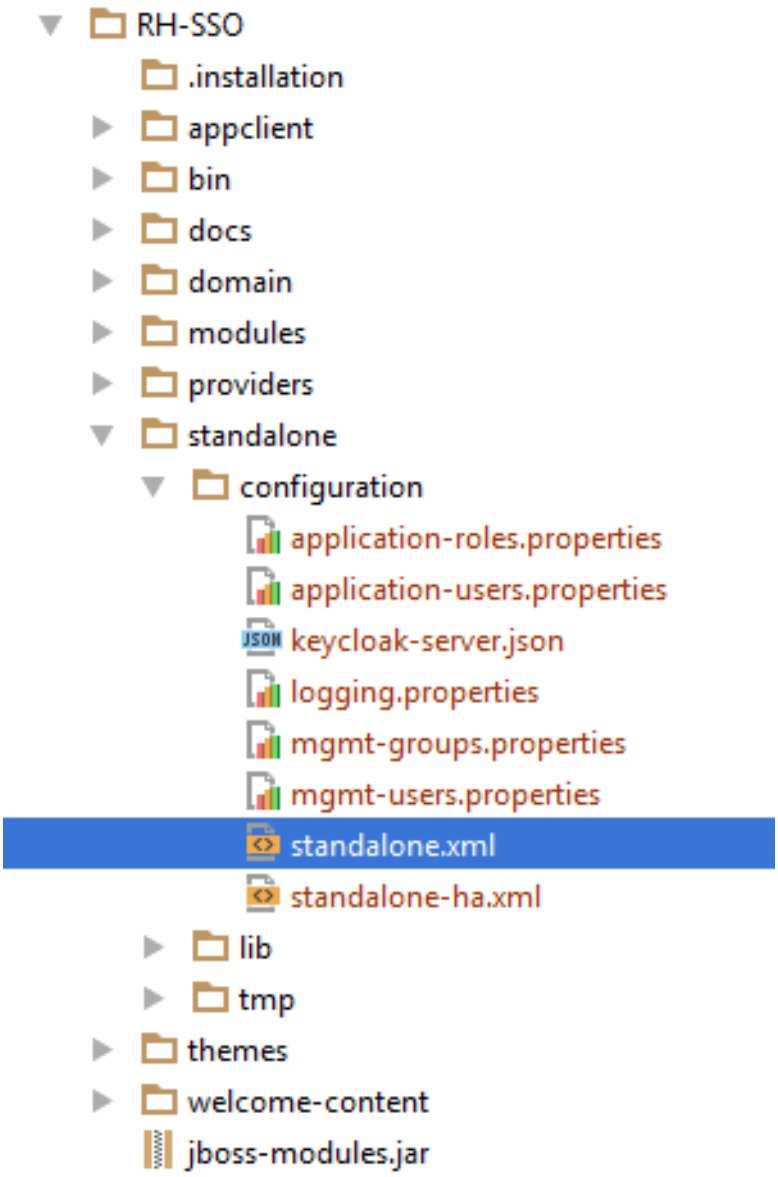
Windows

```
> ...\bin\standalone.bat
```

3.1.2. Standalone Configuration

The bulk of this guide walks you through how to configure infrastructure level aspects of Red Hat Single Sign-On. These aspects are configured in a configuration file that is specific to the application server that Red Hat Single Sign-On is a derivative of. In the standalone operation mode, this file lives in *…/standalone/configuration/standalone.xml*.

Standalone Config File



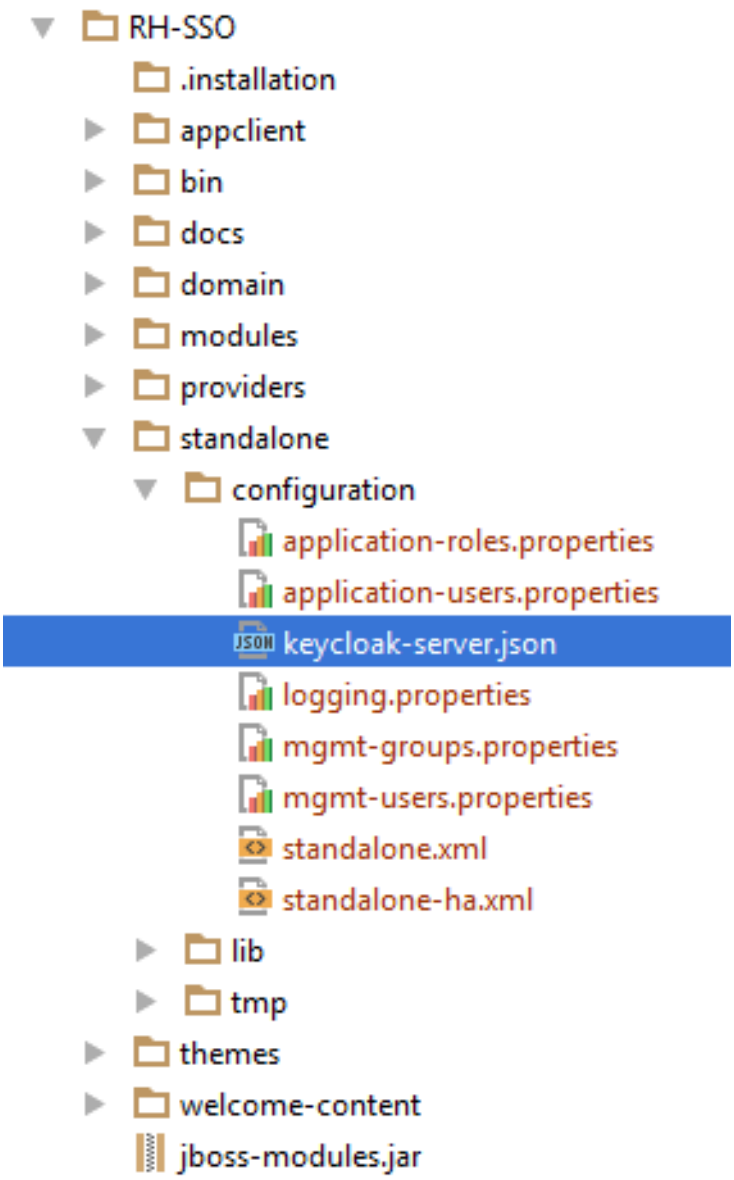
Warning

Any changes you make to this file while the server is running will not take effect and may even be overwritten by the server. Instead use the the command line scripting or the web console of JBoss EAP. See the JBoss EAP Configuration Guide for more information.

3.1.3. Standalone Red Hat Single Sign-On JSON Configuration

Red Hat Single Sign-On has a json configuration file that is specific to Red Hat Single Sign-On components. This configuration is located within the file…/standalone/configuration/keycloak-server.json. This file is used to configure non-infrastructure level things that are only applicable to Red Hat Single Sign-On

Standalone Red Hat Single Sign-On Config File



Warning

Any changes you make to this file while the server is running will not take effect. You'll need to reboot the server.

3.2. STANDALONE CLUSTERED MODE

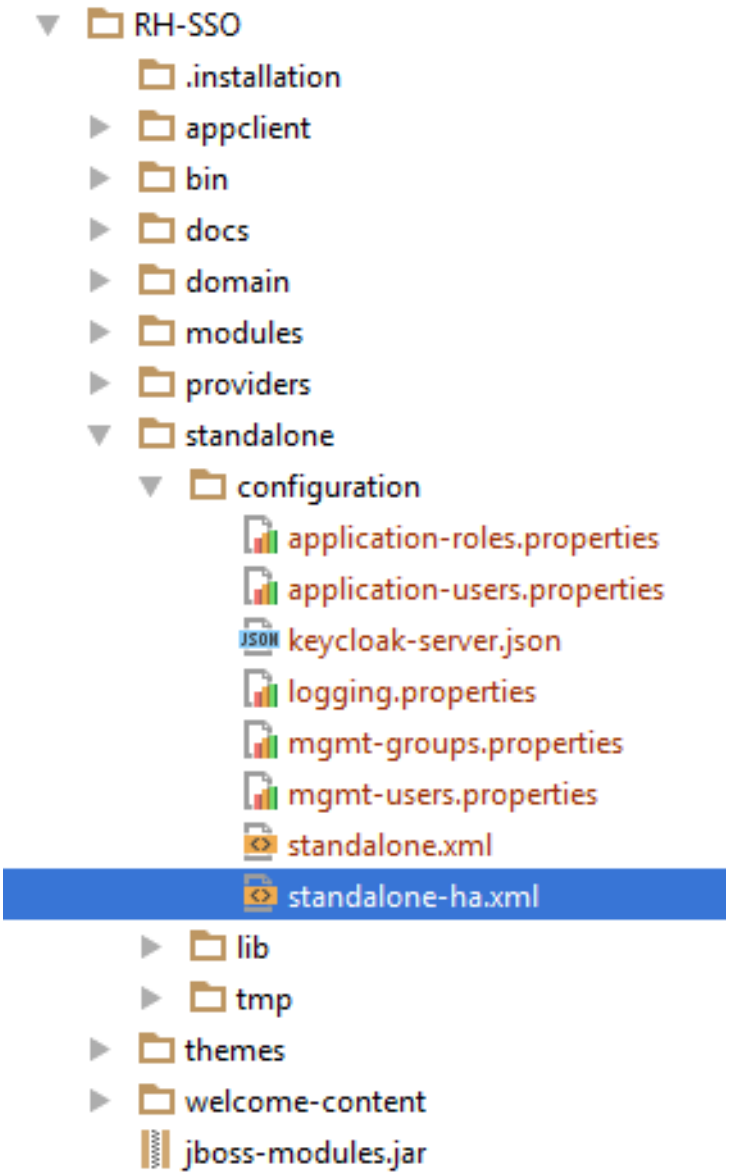
Standalone clustered operation mode is for when you want to run Red Hat Single Sign-On within a cluster. This mode requires that you have a copy of the Red Hat Single Sign-On distribution on each machine you want to run a server instance. This mode can be very easy to deploy initially, but can become quite cumbersome. To make a configuration change you'll have to modify each distribution on each machine. For a large cluster this can become time consuming and error prone.

3.2.1. Standalone Clustered Configuration

The distribution has a mostly pre-configured app server configuration file for running within a cluster. It has all the specific infrastructure settings for networking, databases, caches, and discovery. This file resides in *…/standalone/configuration/standalone-ha.xml*. There's a few things missing from this

configuration. You can't run Red Hat Single Sign-On in a cluster without a configuring a shared database connection. You also need to deploy some type of load balancer in front of the cluster. The clustering and database sections of this guide walk you though these things.

Standalone HA Config



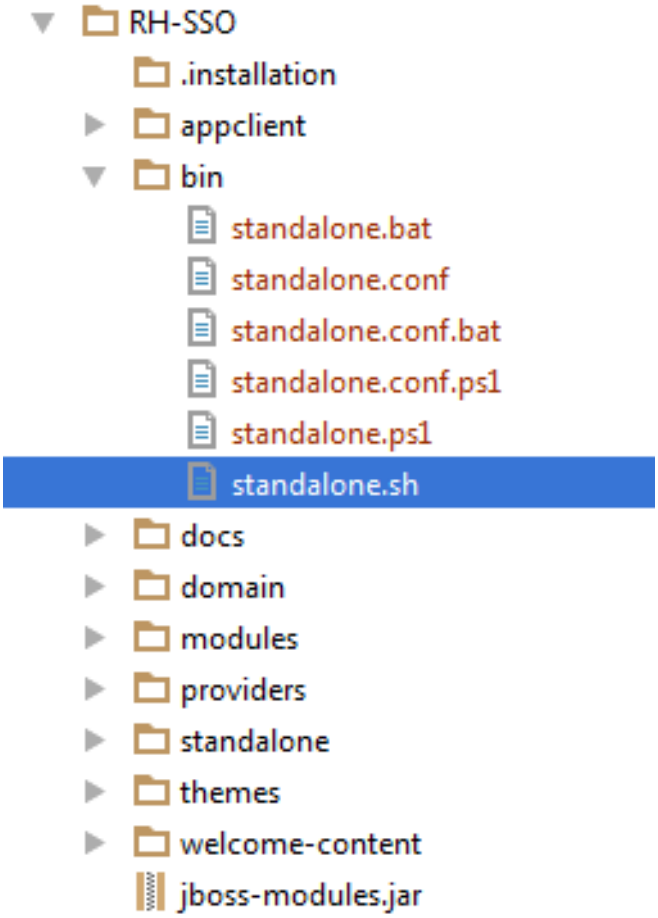
Warning

Any changes you make to this file while the server is running will not take effect and may even be overwritten by the server. Instead use the the command line scripting or the web console of JBoss EAP. See the JBoss EAP Configuration Guide for more information.

3.2.2. Standalone Clustered Boot Script

You use the same boot scripts to start Red Hat Single Sign-On as you do in standalone mode. The difference is that you pass in an additional flag to point to the HA config file.

Standalone Clustered Boot Scripts



To boot the server:

Linux/Unix

```
$ .../bin/standalone.sh --server-config=standalone-ha.xml
```

Windows

```
> ...\bin\standalone.bat --server-config=standalone-ha.xml
```

3.2.3. Standalone Clustered Red Hat Single Sign-On JSON Configuration

The Red Hat Single Sign-On specific json configure file resides in the same place as in standalone mode: *…/standalone/configuration/keycloak.json*. Like *standalone-ha.xml* you have to modify this file in each distribution in your cluster when you want to make a change here.

> **Warning**
>
> Any changes you make to this file while the server is running will not take effect. You'll need to reboot the server.

3.3. DOMAIN CLUSTERED MODE

Domain mode is a way to centrally manage and publish the configuration for your servers.

Running a cluster in standard mode can quickly become aggravating as the cluster grows in size. Every time you need to make a configuration change, you have perform it on each node in the cluster. Domain mode solves this problem by providing a central place to store and publish configuration. It can be quite complex to set up, but it is worth it in the end. This capability is built into the JBoss EAP Application Server which Red Hat Single Sign-On derives from.

Note

The guide will go over the very basics of domain mode. Detailed steps on how to set up domain mode in a cluster should be obtained from the JBoss EAP Configuration Guide.

Here are some of the basic concepts of running in domain mode.

domain controller

The domain controller is a process that is responsible for storing, managing, and publishing the general configuration for each node in the cluster. This process is the central point from which nodes in a cluster obtain their configuration.

host controller

The host controller is responsible for managing server instances on a specific machine. You configure it to run one or more server instances. The domain controller can also interact with the host controllers on each machine to manage the cluster. To reduce the number of running process, a domain controller also acts as a host controller on the machine it runs on.

domain profile

A domain profile is a named set of configuration that can be used by a server to boot from. A domain controller can define multiple domain profiles that are consumed by different servers.

server group

A server group is a collection of servers. They are managed and configured as one. You can assign a domain profile to a server group and every service in that group will use that domain profile as their configuration.

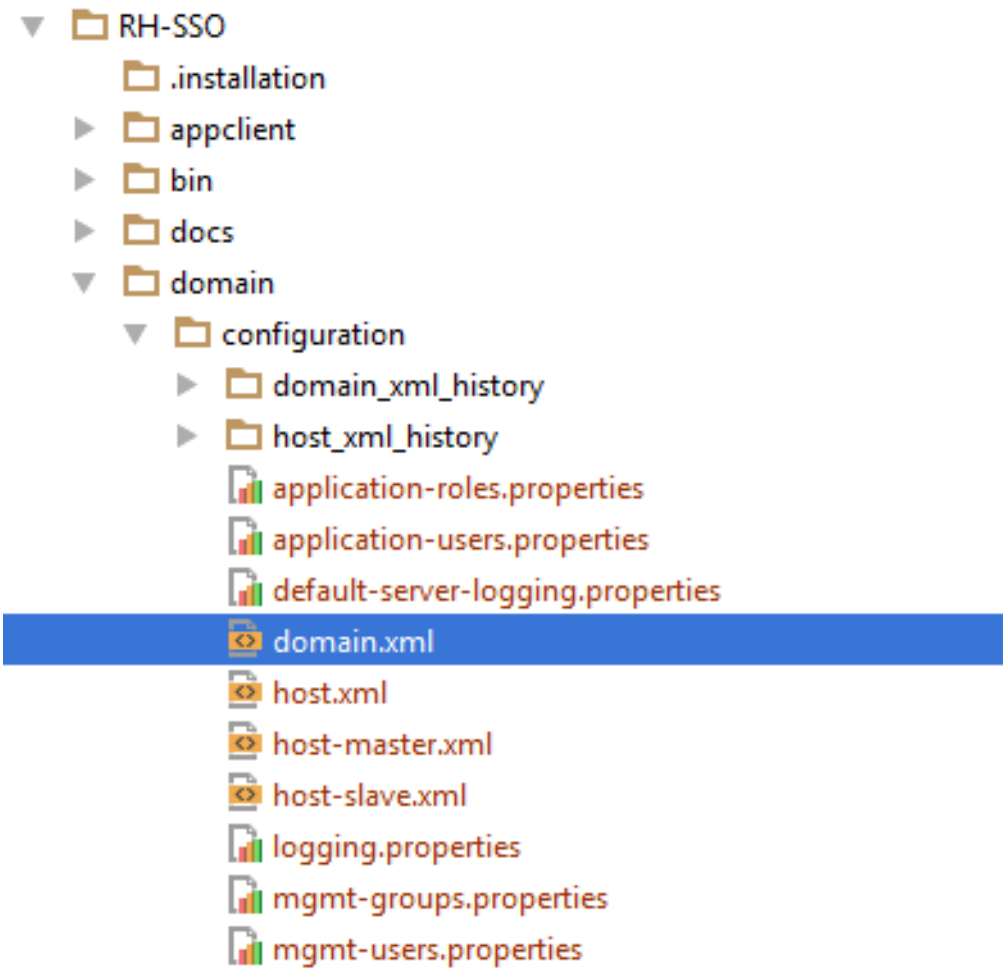
In domain mode, a domain controller is started on a master node. The configuration for the cluster resides in the domain controller. Next a host controller is started on each machine in the cluster. Each host controller deployment configuration specifies how many Red Hat Single Sign-On server instances will be started on that machine. When the host controller boots up, it starts as many Red Hat Single Sign-On server instances as it was configured to do. These server instances pull their configuration from the domain controller.

3.3.1. Domain Configuration

Various other chapters in this guide walk you through configuring various aspects like databases, HTTP network connections, caches, and other infrastructure related things. While standalone mode uses the *standalone.xml* file to configure these things, domain mode uses the…

/domain/configuration/domain.xml configuration file. This is where the domain profile and server group for the Red Hat Single Sign-On server are defined.

domain.xml



Warning

Any changes you make to this file while the domain controller is running will not take effect and may even be overwritten by the server. Instead use the the command line scripting or the web console of JBoss EAP. See the JBoss EAP Configuration Guide for more information.

Let's look at some aspects of this *domain.xml* file. The **`auth-server-standalone`** and **`auth-server-clustered profile`** XML blocks are where you are going to make the bulk of your configuration decisions. You'll be configuring things here like network connections, caches, and database connections.

auth-server profile

```
    <profiles>
        <profile name="auth-server-standalone">
            ...
         </profile>
```

```
        <profile name="auth-server-clustered">
            ...
        </profile>
```

The **auth-server-standalone** profile is a non-clustered setup. The **auth-server-clustered** profile is the clustered setup.

If you scroll down further, you'll see various **socket-binding-groups** defined.

socket-binding-groups

```
    <socket-binding-groups>
        <socket-binding-group name="standard-sockets" default-
interface="public">
            ...
        </socket-binding-group>
        <socket-binding-group name="ha-sockets" default-
interface="public">
            ...
        </socket-binding-group>
        <!-- load-balancer-sockets should be removed in production
systems and replaced with a better softare or hardare based one -->
        <socket-binding-group name="load-balancer-sockets" default-
interface="public">
            ...
        </socket-binding-group>
    </socket-binding-groups>
```

This config defines the default port mappings for various connectors that are opened with each Red Hat Single Sign-On server instance. Any value that contains **${…}** is a value that can be overriden on the command line with the **-D** switch, i.e.

```
$ domain.sh -Djboss.http.port=80
```

The definition of the server group for Red Hat Single Sign-On resides in the **server-groups** XML block. It specifies the domain profile that is used (**default**) and also some default boot arguments for the Java VM when the host controller boots an instance. It also binds a **socket-binding-group** to the server group.

server group

```
    <server-groups>
        <!-- load-balancer-group should be removed in production systems
and replaced with a better softare or hardare based one -->
        <server-group name="load-balancer-group" profile="load-balancer">
            <jvm name="default">
                <heap size="64m" max-size="512m"/>
            </jvm>
            <socket-binding-group ref="load-balancer-sockets"/>
        </server-group>
        <server-group name="auth-server-group" profile="auth-server-
clustered">
            <jvm name="default">
                <heap size="64m" max-size="512m"/>
```

```
            </jvm>
            <socket-binding-group ref="ha-sockets"/>
        </server-group>
    </server-groups>
```

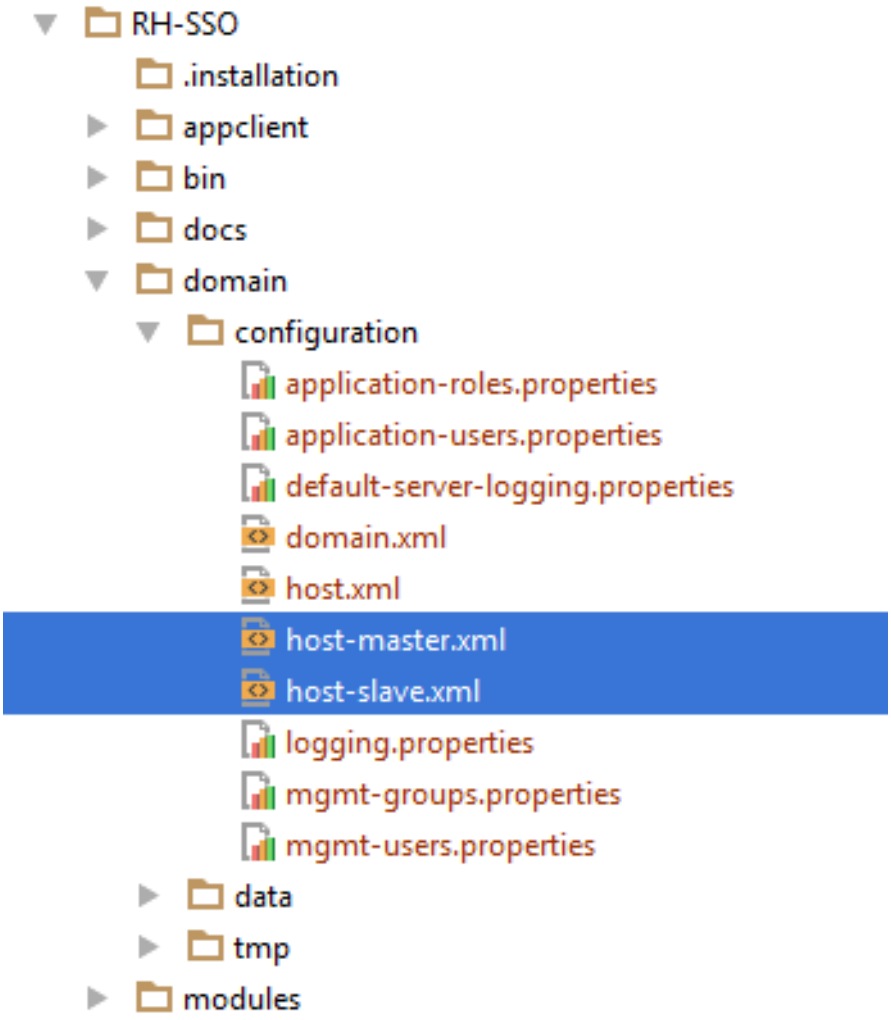
3.3.2. Host Controller Configuration

Red Hat Single Sign-On comes with two host controller configuration files that reside in the *…/domain/configuration/* directory: *host-master.xml* and *host-slave.xml*. *host-master.xml* is configured to boot up a domain controller, a load balancer, and one Red Hat Single Sign-On server instance. *host-slave.xml* is configured to talk to the domain controller and boot up one Red Hat Single Sign-On server instance.

Note

The load balancer is not a required service. It exists so that you can easily test drive clustering on your development machine. While usable in production, you have the option of replacing it if you have a different hardware or software based load balancer you want to use.

Host Controller Config



To disable the load balancer server instance, edit *host-master.xml* and comment out or remove the **`"load-balancer"`** entry.

```
<servers>
    <!-- remove or comment out next line -->
    <server name="load-balancer" group="loadbalancer-group"/>
    ...
</servers>
```

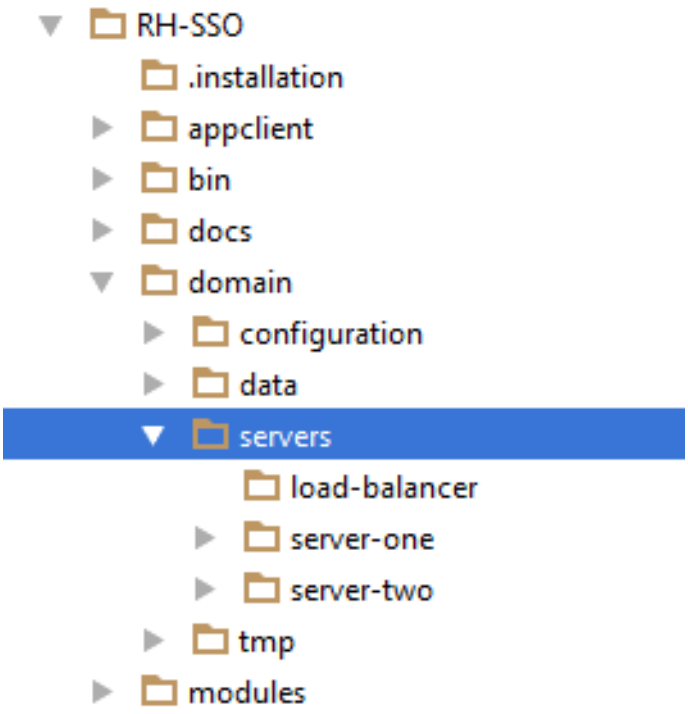
Another interesting thing to note about this file is the declaration of the authentication server instance. It has a **`port-offset`** setting. Any network port defined in the *domain.xml* **`socket-binding-group`** or the server group will have the value of **`port-offset`** added to it. For this example domain setup we do this so that ports opened by the load balancer server don't conflict with the authentication server instance that is started.

```
<servers>
    ...
    <server name="server-one" group="auth-server-group">
        <socket-bindings port-offset="150"/>
    </server>
</servers>
```

3.3.3. Server Instance Working Directories

Each Red Hat Single Sign-On server instance defined in your host files creates a working directory under *…/domain/servers/{SERVER NAME}*. Additional configuration can be put there, and any temporary, log, or data files the server instance needs or creates go there too. The structure of these per server directories ends up looking like any other JBoss EAP booted server.

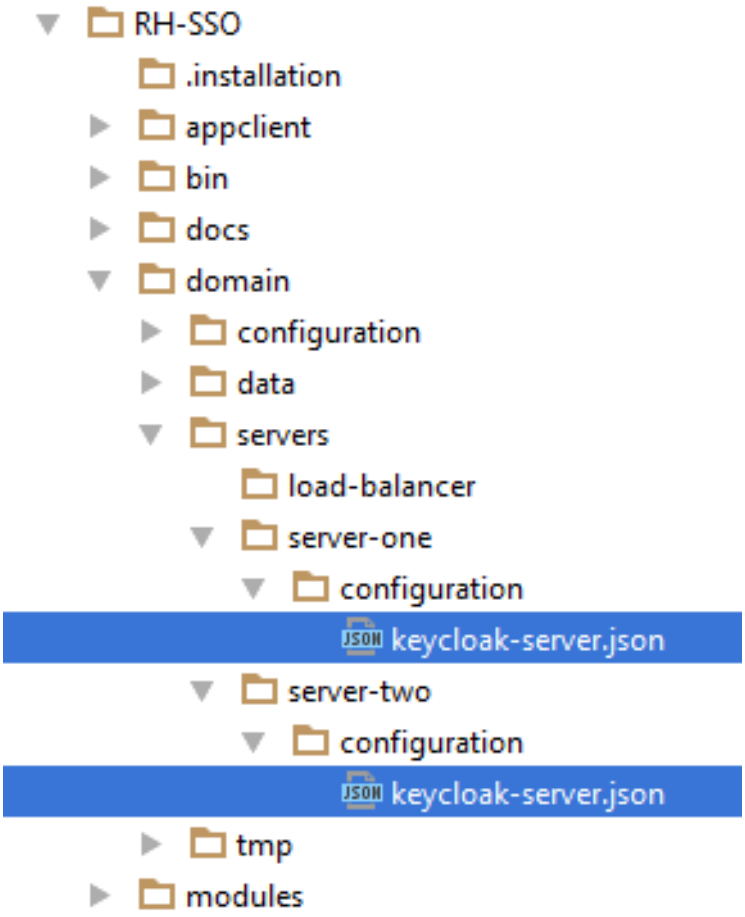
Working Directories



3.3.4. Red Hat Single Sign-On JSON Configuration

Unfortunately, there is no centralized way to manage the *keycloak.json* file. You'll have to manage a copy of this file in every server instance you deploy. This file must exist in the … */domain/servers/{SERVER NAME}/configuration directory.*

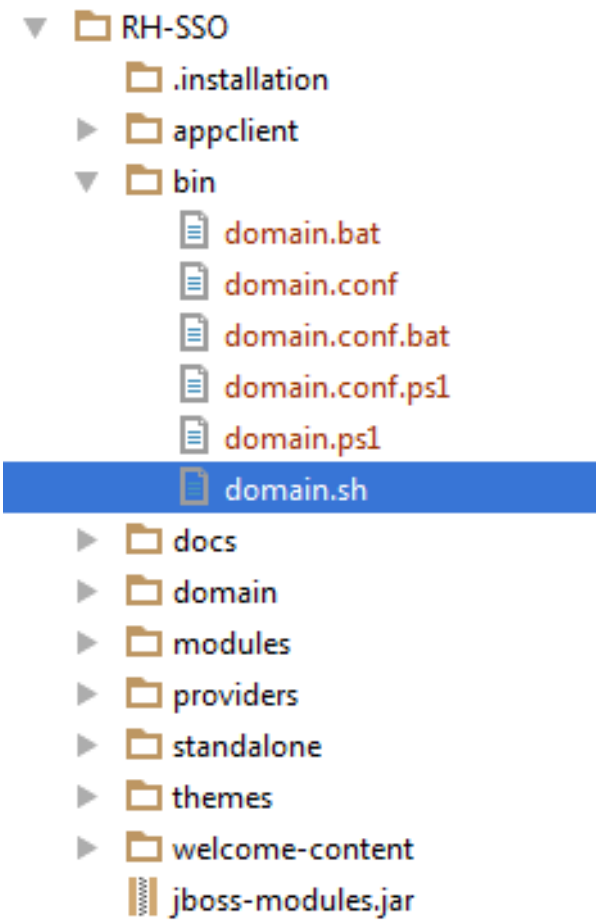
JSON Configuration



3.3.5. Domain Boot Script

When running the server in domain mode, there is a specific script you need to run to boot the server depending on your operating system. These scripts live in the *bin/* directory of the server distribution.

Domain Boot Script



To boot the server:

Linux/Unix

```
$ .../bin/domain.sh --host-config=host-master.xml
```

Windows

```
> ...\bin\domain.bat --host-config=host-slave.xml
```

When running the boot script you will need pass in the host controlling configuration file you are going to use via the **`--host-config`** switch.

3.3.6. Clustered Domain Example

You can test drive clustering using the out-of-the-box *domain.xml* configuration. This example domain is meant to run on one machine and boots up:

- a domain controller
- an HTTP load balancer
- 2 Red Hat Single Sign-On server instances

To simulate running a cluster on two machines, you'll run the **domain.sh** script twice to start two separate host controllers. The first will be the master host controller which will start a domain controller, an HTTP load balancer, and one Red Hat Single Sign-On authentication server instance. The second will be a slave host controller that only starts up an authentication server instance.

3.3.6.1. Setup Slave Connection to Domain Controller

Before you can boot things up though, you have to configure the slave host controller so that it can talk securely to the domain controller. If you do not do this, then the slave host will not be able to obtain the centralized configuration from the domain controller. To set up a secure connection, you have to create a server admin user and a secret that will be shared between the master and the slave. You do this by running the **…/bin/add-user.sh** script.

When you run the script select **Management User** and answer **yes** when it asks you if the new user is going to be used for one AS process to connect to another. This will generate a secret that you'll need to cut and paste into the *…/domain/configuration/host-slave.xml* file.

Add App Server Admin

```
$ add-user.sh
 What type of user do you wish to add?
  a) Management User (mgmt-users.properties)
  b) Application User (application-users.properties)
 (a): a
 Enter the details of the new user to add.
 Using realm 'ManagementRealm' as discovered from the existing property
files.
 Username : admin
 Password recommendations are listed below. To modify these
restrictions edit the add-user.properties configuration file.
  - The password should not be one of the following restricted values
{root, admin, administrator}
  - The password should contain at least 8 characters, 1 alphabetic
character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
  - The password should be different from the username
 Password :
 Re-enter Password :
 What groups do you want this user to belong to? (Please enter a comma
separated list, or leave blank for none)[ ]:
 About to add user 'admin' for realm 'ManagementRealm'
 Is this correct yes/no? yes
 Added user 'admin' to file '/.../standalone/configuration/mgmt-
users.properties'
 Added user 'admin' to file '/.../domain/configuration/mgmt-
users.properties'
 Added user 'admin' with groups to file
'/.../standalone/configuration/mgmt-groups.properties'
 Added user 'admin' with groups to file
'/.../domain/configuration/mgmt-groups.properties'
 Is this new user going to be used for one AS process to connect to
another AS process?
 e.g. for a slave host controller connecting to the master or for a
```

```
Remoting connection for server to server EJB calls.
 yes/no? yes
 To represent the user add the following to the server-identities
definition <secret value="bWdtdDEyMyE=" />
```

Note

The add-user.sh does not add user to Red Hat Single Sign-On server but to the underlying JBoss Enterprise Application Platform. The credentials used and generated in the above script are only for example purpose. Please use the ones generated on your system.

Now cut and paste the secret value into the *…/domain/configuration/host-slave.xml* file as follows:

```
<management>
    <security-realms>
        <security-realm name="ManagementRealm">
            <server-identities>
                <secret value="bWdtdDEyMyE="/>
            </server-identities>
```

3.3.6.2. Run the Boot Scripts

Since we're simulating a two node cluster on one development machine, you'll run the boot script twice:

Boot up master

```
$ domain.sh --host-config=host-master.xml
```

Boot up slave

```
$ domain.sh --host-config=host-slave.xml
```

To try it out, open your browser and go to http://localhost:8080/auth

CHAPTER 4. MANAGE CONFIGURATION AT RUNTIME

In the upcoming chapters, you'll often be provided two options for applying application server configuration changes to your deployment. You'll be shown how to edit the *standalone.xml* or *domain.xml* directly. This must be done when the server (or servers) are offline. Additionally, you may be shown how to apply config changes on a running server using the app server's command line interface (JBoss EAP CLI). This chapter discusses how you will do this.

4.1. START THE JBOSS EAP CLI

To start the JBoss EAP CLI, you need to run the **`jboss-cli`** script.

Linux/Unix

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

Windows

```
> ...\bin\jboss-cli.bat
```

This will bring you to a prompt like this:

Prompt

```
[disconnected /]
```

There's a few commands you can execute without a running standalone server or domain controller, but usually you will have to have those services booted up before you can execute CLI commands. To connect to a running server simply execute the **`connect`** command.

connect

```
[disconnected /] connect
connect
[domain@localhost:9990 /]
```

You may be thinking to yourself, "I didn't enter in any username or password!". If you run **`jboss-cli`** on the same machine as your running standalone server or domain controller and your account has appropriate file permissions, you do not have to setup or enter in a admin username and password. See the JBoss EAP Configuration Guide for more details on how to make things more secure if you are uncomfortable with that setup.

CHAPTER 5. RELATIONAL DATABASE SETUP

Red Hat Single Sign-On comes with its own embedded Java-based relational database called H2. This is the default database that Red Hat Single Sign-On will use to persist data and really only exists so that you can run the authentication server out of the box. We highly recommend that you replace it with a more production ready external database. The H2 database is not very viable in high concurrency situations and should not be used in a cluster either. The purpose of this chapter is to show you how to connect Red Hat Single Sign-On to a more mature database.

Red Hat Single Sign-On uses two layered technologies to persist its relational data. The bottom layered technology is JDBC. JDBC is a Java API that is used to connect to a RDBMS. There are different JDBC drivers per database type that are provided by your database vendor. This chapter discusses how to configure Red Hat Single Sign-On to use one of these vendor-specific drivers.

The top layered technology for persistence is Hibernate JPA. This is a object to relational mapping API that maps Java Objects to relational data. Most deployments of Red Hat Single Sign-On will never have to touch the configuration aspects of Hibernate, but we will discuss how that is done if you run into that rare circumstance.

Note

Datasource configuration is covered much more thoroughly within the the datasource configuration chapter of the JBoss EAP Configuration Guide.

5.1. RDBMS SETUP CHECKLIST

These are the steps you will need to perform to get an RDBMS configured for Red Hat Single Sign-On.

1. Locate and download a JDBC driver for your database
2. Package the driver JAR into a module and install this module into the server
3. Declare the JDBC driver in the configuration profile of the server
4. Modify the datasource configuration to use your database's JDBC driver
5. Modify the datasource configuration to define the connection parameters to your database

This chapter will use PostgresSQL for all its examples. Other databases follow the same steps for installation.

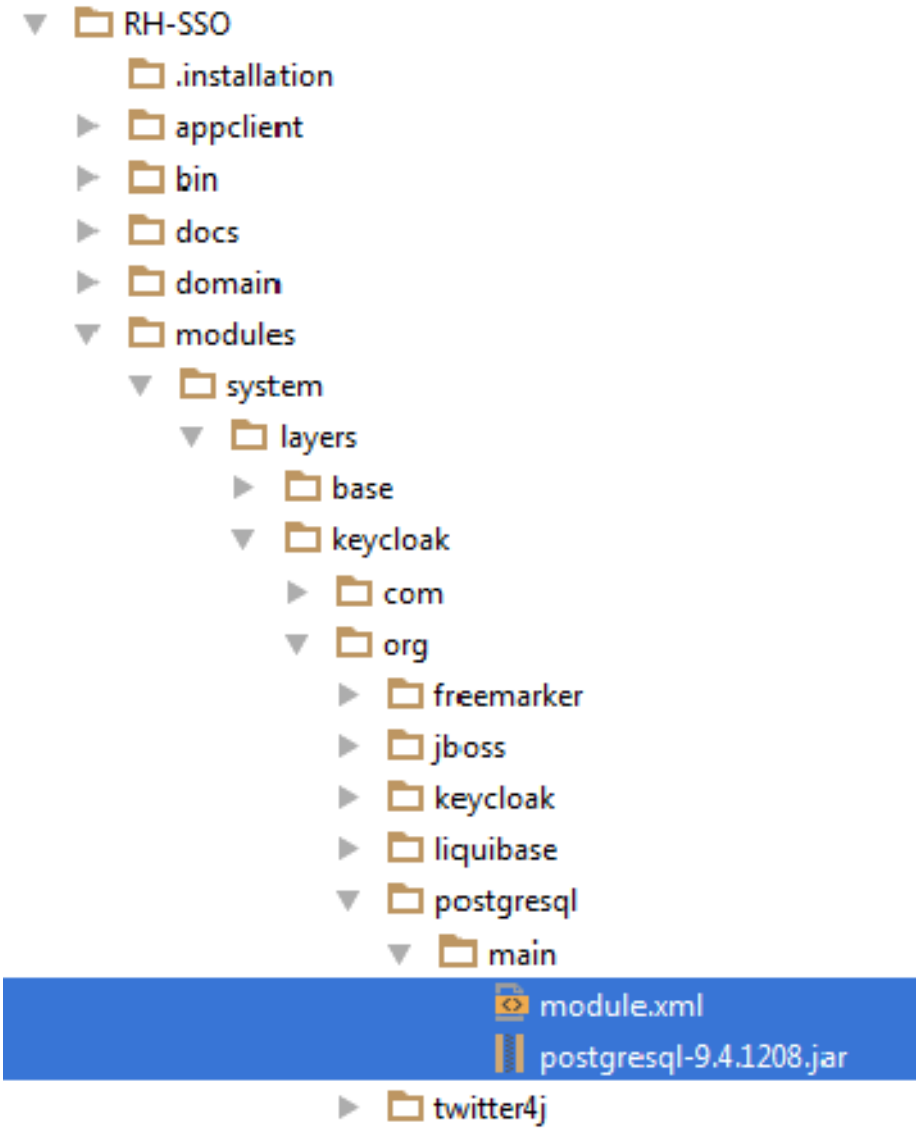
5.2. PACKAGE THE JDBC DRIVER

Find and download the JDBC driver JAR for your RDBMS. Before you can use this driver, you must package it up into a module and install it into the server. Modules define JARs that are loaded into the Red Hat Single Sign-On classpath and the dependencies those JARs have on other modules. They are pretty simple to set up.

Within the *…/modules/* directory of your Red Hat Single Sign-On distribution, you need to create a directory structure to hold your module definition. The convention is use the Java package name of the JDBC driver for the name of the directory structure. For PostgreSQL, create the directory *org/postgresql/main*. Copy your database driver JAR into this directory and create an empty

module.xml file within it too.

Module Directory



After you have done this, open up the *module.xml* file and create the following XML

Module XML

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.3" name="org.postgresql">

    <resources>
        <resource-root path="postgresql-9.4.1208.jar"/>
    </resources>

    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

The module name should match the directory structure of your module. So, *org/postgresql* maps to **`org.postgresql`**. The **`resource-root path`** attribute should specify the JAR filename of the driver. The rest are just the normal dependencies that any JDBC driver JAR would have.

5.3. DECLARE AND LOAD JDBC DRIVER

The next thing you have to do is declare your newly packaged JDBC driver into your deployment profile so that it loads and becomes available when the server boots up. Where you perform this action depends on your operating mode. If you're deploying in standard mode, edit *…/standalone/configuration/standalone.xml*. If you're deploying in standard clustering mode, edit *…/standalone/configuration/standalone-ha.xml*. If you're deploying in domain mode, edit *…/domain/configuration/domain.xml*. In domain mode, you'll need to make sure you edit the profile you are using: either **`auth-server-standalone`** or **`auth-server-clustered`**

Within the profile, search for the **`drivers`** XML block within the **`datasources`** subsystem. You should see a pre-defined driver declared for the H2 JDBC driver. This is where you'll declare the JDBC driver for your external database.

JDBC Drivers

```
  <subsystem xmlns="urn:jboss:domain:datasources:4.0">
     <datasources>
       ...
       <drivers>
          <driver name="h2" module="com.h2database.h2">
              <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
          </driver>
       </drivers>
     </datasources>
  </subsystem>
```

Within the **`drivers`** XML block you'll need to declare an additional JDBC driver. It needs to have a **`name`** which you can choose to be anything you want. You specify the **`module`** attribute which points to the **`module`** package you created earlier for the driver JAR. Finally you have to specify the driver's Java class. Here's an example of installing PostgreSQL driver that lives in the module example defined earlier in this chapter.

Declare Your JDBC Drivers

```
  <subsystem xmlns="urn:jboss:domain:datasources:4.0">
     <datasources>
       ...
       <drivers>
          <driver name="postgresql" module="org.postgresql">
              <xa-datasource-class>org.postgresql.Driver</xa-datasource-
class>
          </driver>
          <driver name="h2" module="com.h2database.h2">
              <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
```

```
          </driver>
        </drivers>
      </datasources>
    </subsystem>
```

5.4. MODIFY THE RED HAT SINGLE SIGN-ON DATASOURCE

After declaring your JDBC driver, you have to modify the existing datasource configuration that Red Hat Single Sign-On uses to connect it to your new external database. You'll do this within the same configuration file and XML block that you registered your JDBC driver in. Here's an example that set's up the connection to your new database:

Declare Your JDBC Drivers

```
  <subsystem xmlns="urn:jboss:domain:datasources:4.0">
     <datasources>
       ...
       <datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-
name="KeycloakDS" enabled="true" use-java-context="true">
           <connection-
url>jdbc:postgresql://localhost/keycloak</connection-url>
           <driver>postgresql</driver>
           <pool>
               <max-pool-size>20</max-pool-size>
           </pool>
           <security>
               <user-name>William</user-name>
               <password>password</password>
           </security>
       </datasource>
        ...
     </datasources>
  </subsystem>
```

Search for the **`datasource`** definition for **`KeycloakDS`**. You'll first need to modify the **`connection-url`**. The documentation for your vendor's JDBC implementation should specify the format for this connection URL value.

Next define the **`driver`** you will use. This is the logical name of the JDBC driver you declared in the previous section of this chapter.

It is expensive to open a new connection to a database ever time you want to perform a transaction. To compensate, the datasource implementation maintains a pool of open connections. The **`max-pool-size`** specifies the maximum number of connections it will pool. You may want to change the value of this depending on the load of your system.

Finally, with PostgreSQL at least, you need to define the database username and password that is needed to connect to the database. You may be worried that this is in clear text in the example. There are methods to obfuscate this, but this is beyond the scope of this guide.

Note

For more information and details features of datasources, please see the the datasource configuration chapter of the JBoss EAP Configuration Guide.

5.5. HIBERNATE CONFIGURATION

The Hibernate persistence API is already pre-configured out of the box and rarely needs to be changed. The configuration for this component lies in the *keycloak-server.json* file. If you are running in standalone mode, this file is in the *…/standalone/configuration* directory. If you are running in domain mode this file will live in the *…/domain/servers/{server name}/configuration* directory.

Hibernate JPA Config

```
"connectionsJpa": {
    "default": {
        "dataSource": "java:jboss/datasources/KeycloakDS",
        "databaseSchema": "update"
    }
},
```

Possible configuration options are:

dataSource

JNDI name of the dataSource

jta

boolean property to specify if datasource is JTA capable

driverDialect

Value of database dialect. In most cases you don't need to specify this property as dialect will be autodetected by Hibernate.

databaseSchema

Specify if schema should be updated or validated. Valid values are **`update`** (default) and **`validate`**. The **`update`** value will set up or update the table definitions that Red Hat Single Sign-On needs. **`validate`** just makes sure that the database has the appropriate table definitions set up.

showSql

Specify whether Hibernate should show all SQL commands in the console (false by default). This is very verbose!

formatSql

Specify whether Hibernate should format SQL commands (true by default)

globalStatsInterval

Will log global statistics from Hibernate about executed DB queries and other things. Statistics are always reported to server log at specified interval (in seconds) and are cleared after each report.

schema

Specify the database schema to use

Note

All these configuration switches and more are described in the JBoss EAP Development Guide.

CHAPTER 6. NETWORK SETUP

Red Hat Single Sign-On can run out of the box with some networking limitations. For one, all network endpoints bind to **localhost** so the auth server is really only usable on one local machine. For HTTP based connections, it does not use default ports like 80 and 443. HTTPS/SSL is not configured out of the box and without it, Red Hat Single Sign-On has many security vulnerabilities. Finally, Red Hat Single Sign-On may often need to make secure SSL and HTTPS connections to external servers and thus need a trust store set up so that endpoints can be validated correctly. This chapter discusses all of these things.

6.1. BIND ADDRESSES

By default Red Hat Single Sign-On binds to the localhost loopback address **127.0.0.1**. That's not a very useful default if you want the authentication server available on your network. Generally, what we recommend is that you deploy a reverse proxy or load balancer on a public network and route traffic to individual Red Hat Single Sign-On server instances on a private network. In either case though, you still need to set up your network interfaces to bind to something other than **localhost**.

Setting the bind address is quite easy and can be done on the command line with either the *standalone.sh* or *domain.sh* boot scripts discussed in the Choosing an Operating Mode chapter.

```
$ standalone.sh -b 192.168.0.5
```

The **-b** switch sets the IP bind address for any public interfaces.

Alternatively, if you don't want to set the bind address at the command line, you can edit the profile configuration of your deployment. Open up the profile configuration file (*standalone.xml* or *domain.xml* depending on your operating mode) and look for the **interfaces** XML block.

```
    <interfaces>
        <interface name="management">
            <inet-address
value="${jboss.bind.address.management:127.0.0.1}"/>
        </interface>
        <interface name="public">
            <inet-address value="${jboss.bind.address:127.0.0.1}"/>
        </interface>
    </interfaces>
```

The **public** interface corresponds to subsystems creating sockets that are available publicly. An example of one of these subsystems is the web layer which serves up the authentication endpoints of Red Hat Single Sign-On. The **management** interface corresponds to sockets opened up by the management layer of the JBoss EAP. Specifically the sockets which allow you to use the **jboss-cli.sh** command line interface and the JBoss EAP web console.

In looking at the **public** interface you see that it has a special string **${jboss.bind.address:127.0.0.1}**. This string denotes a value **127.0.0.1** that can be overriden on the command line by setting a Java system property, i.e.:

```
$ domain.sh -Djboss.bind.address=192.168.0.5
```

The **-b** is just a shorthand notation for this command. So, you can either change the bind address value directly in the profile config, or change it on the command line when you boot up.

Note

There's a lot more nifty options when setting up **`interface`** definitions. See the the network interface chapter of the JBoss EAP Configuration Guide.

6.2. SOCKET PORT BINDINGS

The ports opened for each socket have a pre-defined default that can be overriden at the command line or within configuration. To illustrate this configuration, let's pretend you are running in standalone mode and open up the *…/standalone/configuration/standalone.xml*. Search for **`socket-binding-group`**.

```
    <socket-binding-group name="standard-sockets" default-
interface="public" port-offset="${jboss.socket.binding.port-offset:0}">
        <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
        <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}"/>
        <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
        <socket-binding name="http" port="${jboss.http.port:8080}"/>
        <socket-binding name="https" port="${jboss.https.port:8443}"/>
        <socket-binding name="txn-recovery-environment" port="4712"/>
        <socket-binding name="txn-status-manager" port="4713"/>
        <outbound-socket-binding name="mail-smtp">
            <remote-destination host="localhost" port="25"/>
        </outbound-socket-binding>
    </socket-binding-group>
```

`socket-bindings` define socket connections that will be opened by the server. These bindings specify the **`interface`** (bind address) they use as well as what port number they will open. The ones you will be most interested in are:

http

Defines the port used for Red Hat Single Sign-On HTTP connections

https

Defines the port used for Red Hat Single Sign-On HTTPS connections

ajp

This socket binding defines the port used for the AJP protocol. This protocol is used by Apache HTTPD server in conjunction **`mod-cluster`** when you are using Apache HTTPD as a load balancer.

management-http

Defines the HTTP connection used by JBoss EAP CLI and web console.

When running in domain mode setting the socket configurations is a bit trickier as the example *domain.xml* file has multiple **`socket-binding-groups`** defined. If you scroll down to the **`server-group`** definitions you can see what **`socket-binding-group`** is used for each **`server-group`**.

domain socket bindings

```
    <server-groups>
        <server-group name="load-balancer-group" profile="load-balancer">
            ...
            <socket-binding-group ref="load-balancer-sockets"/>
        </server-group>
        <server-group name="auth-server-group" profile="auth-server-
clustered">
            ...
            <socket-binding-group ref="ha-sockets"/>
        </server-group>
    </server-groups>
```

Note

There's a lot more nifty options when setting up **`socket-binding-group`** definitions. See the the socket binding group chapter of the JBoss EAP Configuration Guide.

6.3. SETTING UP HTTPS/SSL

Warning

Red Hat Single Sign-On is not set up by default to handle SSL/HTTPS. It is highly recommended that you either enable SSL on the Red Hat Single Sign-On server itself or on a reverse proxy in front of the Red Hat Single Sign-On server.

This default behavior is defined by the SSL/HTTPS mode of each Red Hat Single Sign-On realm. This is discussed in more detail in the Server Administration Guide, but let's give some context and a brief overview of these modes.

external requests

Red Hat Single Sign-On can run out of the box without SSL so long as you stick to private IP addresses like **`localhost`**, **`127.0.0.1`**, **`10.0.x.x`**, **`192.168.x.x`**, and **`172..16.x.x`**. If you don't have SSL/HTTPS configured on the server or you try to access Red Hat Single Sign-On over HTTP from a non-private IP adress you will get an error.

none

Red Hat Single Sign-On does not require SSL. This should really only be used in development when you are playing around with things.

all requests

Red Hat Single Sign-On requires SSL for all IP addresses.

The SSL mode for each realm can be configured in the Red Hat Single Sign-On admin console.

6.3.1. Enabling SSL/HTTPS for the Red Hat Single Sign-On Server

If you are not using a reverse proxy or load balancer to handle HTTPS traffic for you, you'll need to enable HTTPS for the Red Hat Single Sign-On server. This involves

1. Obtaining or generating a keystore that contains the private key and certificate for SSL/HTTP traffic
2. Configuring the Red Hat Single Sign-On server to use this keypair and certificate.

6.3.1.1. Creating the Certificate and Java Keystore

In order to allow HTTPS connections, you need to obtain a self signed or third-party signed certificate and import it into a Java keystore before you can enable HTTPS in the web container you are deploying the Red Hat Single Sign-On Server to.

6.3.1.1.1. Self Signed Certificate

In development, you will probably not have a third party signed certificate available to test a Red Hat Single Sign-On deployment so you'll need to generate a self-signed one using the **`keytool`** utility that comes with the Java JDK.

```
$ keytool -genkey -alias localhost -keyalg RSA -keystore keycloak.jks -
validity 10950
    Enter keystore password: secret
    Re-enter new password: secret
    What is your first and last name?
    [Unknown]:  localhost
    What is the name of your organizational unit?
    [Unknown]:  Keycloak
    What is the name of your organization?
    [Unknown]:  Red Hat
    What is the name of your City or Locality?
    [Unknown]:  Westford
    What is the name of your State or Province?
    [Unknown]:  MA
    What is the two-letter country code for this unit?
    [Unknown]:  US
    Is CN=localhost, OU=Keycloak, O=Test, L=Westford, ST=MA, C=US
correct?
    [no]:  yes
```

You should answer **`What is your first and last name ?`** question with the DNS name of the machine you're installing the server on. For testing purposes, **`localhost`** should be used. After executing this command, the **`keycloak.jks`** file will be generated in the same directory as you executed the **`keytool`** command in.

If you want a third-party signed certificate, but don't have one, you can obtain one for free at cacert.org. You'll have to do a little set up first before doing this though.

The first thing to do is generate a Certificate Request:

```
$ keytool -certreq -alias yourdomain -keystore keycloak.jks >
keycloak.careq
```

Where **`yourdomain`** is a DNS name for which this certificate is generated for. Keytool generates the request:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIC2jCCAcICAQAwZTELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk1BMREwDwYDVQQHEwhXZXN
0Zm9y
ZDEQMA4GA1UEChMHUmVkIEhhdDEQMA4GA1UECxMHUmVkIEhhdDESMBAGA1UEAxMJbG9jYWx
ob3N0
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr7kck2TaavlEOGbcpi9c0rncY4H
hdzmY
Ax2nZfq1eZEaIPqI5aTxwQZzzLDK9qbeAd8Ji79HzSqnRDxNYaZu7mAYhFKHgixsolE3o5Y
fzbw1
29RvyeUVe+WZxv5oo9wolVVpdSINIMEL2LaFhtX/c1dqiqYVpfnvFshZQaIg2nL8juzZcBj
j4as
H98gIS7khql/dkZKsw9NLvyxgJvp7PaXurX29fNf3ihG+oFrL22oFyV54BWWxXCKU/GPn61
EGZGw
Ft2qSIGLdctpMD1aJR2bcnlhEjZKDksjQZoQ5YMXaAGkcYkG6QkgrocDE2YXDbi7GIdf9Me
gVJ35
2DQMpwIDAQABoDAwLgYJKoZIhvcNAQkOMSEwHzAdBgNVHQ4EFgQUQwlZJBA+fjiDdiVzaO9
vrE/i
n2swDQYJKoZIhvcNAQELBQADggEBAC5FRvMkhal3q86tHPBYWBuTtmcSjs4qUm6V6f63frh
veWHf
PzRrI1xH272XUIeBk0gtzWo0nNZnf0mMCtUBbHhhDcG82xolikfqibZijoQZCiGiedVjHJF
tniDQ
9bMDUOXEMQ7gHZg5q6mJfNG9MbMpQaUVEEFvfGEQQxbiFK7hRWU8S23/d80e8nExgQxdJWJ
6vd0X
MzzFK6j4Dj55bJVuM7GFmfdNC52pNOD5vYe47Aqh8oajHX9XTycVtPXl45rrWAH33ftbrS8
SrZ2S
vqIFQeuLL3BaHwpl3t7j2lMWcK1p80laAxEASib/fAwrRHpLHBXRcq6uALUOZl4Alt8=
-----END NEW CERTIFICATE REQUEST-----
```

Send this ca request to your CA. The CA will issue you a signed certificate and send it to you. Before you import your new cert, you must obtain and import the root certificate of the CA. You can download the cert from CA (ie.: root.crt) and import as follows:

```
$ keytool -import -keystore keycloak.jks -file root.crt -alias root
```

Last step is to import your new CA generated certificate to your keystore:

```
$ keytool -import -alias yourdomain -keystore keycloak.jks -file your-
certificate.cer
```

6.3.1.2. Configure Red Hat Single Sign-On to Use the Keystore

Now that you have a Java keystore with the appropriate certificates, you need to configure your Red Hat Single Sign-On installation to use it. First step is to move the keystore file to the *configuration/* directory of your deployment and to edit the *standalone.xml*, *standalone-ha.xml* or *domain.xml* file to use the keystore and enable HTTPS. (See operating mode).

In the standalone or domain configuration file, search for the **`security-realms`** element and add:

```
<security-realm name="UndertowRealm">
    <server-identities>
        <ssl>
            <keystore path="keycloak.jks" relative-
```

```
to="jboss.server.config.dir" keystore-password="secret" />
        </ssl>
    </server-identities>
</security-realm>
```

Find the element **`server name="default-server"`** (it's a child element of **`subsystem xmlns="urn:jboss:domain:undertow:`**) and add:

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
   <buffer-cache name="default"/>
   <server name="default-server">
      <https-listener name="https" socket-binding="https" security-
realm="UndertowRealm"/>
   ...
</subsystem>
```

6.4. OUTGOING HTTP REQUESTS

The Red Hat Single Sign-On server often needs to make non-browser HTTP requests to the applications and services it secures. The auth server manages these outgoing connections by maintaining an HTTP client connection pool. There are some thing you'll need to configure in the *keycloak-server.json*. Where this file lives depends on your operating mode

keycloak-server.json HTTP client Config

```
"connectionsHttpClient": {
    "default": {}
},
```

Possible configuration options are:

establish-connection-timeout-millis

Timeout for establishing a socket connection.

socket-timeout-millis

If an outgoing request does not receive data for this amount of time, timeout the connection.

connection-pool-size

How many connections can be in the pool (128 by default).

max-pooled-per-route

How many connections can be pooled per host (64 by default).

connection-ttl-millis

Maximum connection time to live in milliseconds. Not set by default.

max-connection-idle-time-millis

Maximum time the connection might stay idle in the connection pool (900 seconds by default). Will start background cleaner thread of Apache HTTP client. Set to **-1** to disable this checking and the background thread.

disable-cookies

`true` by default. When set to true, this will disable any cookie caching.

client-keystore

This is the file path to a Java keystore file. This keystore contains client certificate for two-way SSL.

client-keystore-password

Password for the client keystore. This is *REQUIRED* if `client-keystore` is set.

client-key-password

Password for the client's key. This is *REQUIRED* if `client-keystore` is set.

6.4.1. Outgoing HTTPS Request Truststore

When Red Hat Single Sign-On invokes on remote HTTPS endpoints, it has to validate the remote server's certificate in order to ensure it is connecting to a trusted server. This is necessary in order to prevent man-in-the-middle attacks. The certificates of these remote server's or the CA that signed these certificates must be put in a truststore. This truststore is managed by the Red Hat Single Sign-On server.

The truststore is used when connecting securely to identity brokers, LDAP identity providers, when sending emails, and for backchannel communication with client applications.

Warning

By default, a truststore provider is not configured, and any https connections fall back to standard java truststore configuration as described in Java's JSSE Reference Guide. If there is no trust establised, then these outgoing HTTPS requests will fail.

You can use *keytool* to create a new truststore file or add trusted host certificates to an existing one:

```
$ keytool -import -alias HOSTDOMAIN -keystore truststore.jks -file
host-certificate.cer
```

The truststore is configured within the *keycloak-server.json* file. The location of this file depends on your operating mode. You can add your truststore configuration by using the following template:

```
"truststore": {
    "file": {
        "file": "path to your .jks file containing public
certificates",
        "password": "password",
        "hostname-verification-policy": "WILDCARD",
        "disabled": false
    }
}
```

Possible configuration options for this setting are:

file

The path to a Java keystore file. HTTPS requests need a way to verify the host of the server they are talking to. This is what the trustore does. The keystore contains one or more trusted host certificates or certificate authorities. This truststore file should only contain public certificates of your secured hosts. This is *REQUIRED* if **`disabled`** is not true.

password

Password for the truststore. This is *REQUIRED* if **`disabled`** is not true.

hostname-verification-policy

`WILDCARD` by default. For HTTPS requests, this verifies the hostname of the server's certificate. **`ANY`** means that the hostname is not verified.**`WILDCARD`** Allows wildcards in subdomain names i.e. *.foo.com. **`STRICT`** CN must match hostname exactly.

disabled

If true (default value), truststore configuration will be ignored, and certificate checking will fall back to JSSE configuration as described. If set to false, you must configure **`file`**, and **`password`** for the truststore.

CHAPTER 7. CLUSTERING

This section covers configuring Red Hat Single Sign-On to run in a cluster. There's a number of things you have to do when setting up a cluster, specifically:

- Pick an operation mode
- Configure a shared external database
- Set up a load balancer
- Supplying a private network that supports IP multicast

Picking an operation mode and configuring a shared database have been discussed earlier in this guide. In this chapter we'll discuss setting up a load balancer and supplying a private network. We'll also discuss some issues that you need to be aware of when booting up a host in the cluster.

Note: It is possible to cluster Red Hat Single Sign-On without IP Multicast, but this topic is beyond the scope of this guide. Please see the JGroups chapter of the JBoss EAP Configuration Guide.

7.1. RECOMMENDED NETWORK ARCHITECTURE

The recommended network architecture for deploying Red Hat Single Sign-On is to set up an HTTP/HTTPS load balancer on a public IP address that routes requests to Red Hat Single Sign-On servers sitting on a private network. This isolates all clustering connections and provides a nice means of protecting the servers.

Note

By default, there is nothing to prevent unauthorized nodes from joining the cluster and broadcasting multicast messages. This is why cluster nodes should be in a private network, with a firewall protecting them from outside attacks.

7.2. CLUSTERING EXAMPLE

Red Hat Single Sign-On does come with an out of the box clustering demo that leverages domain mode. Review the Clustered Domain Example chapter for more details.

7.3. SETTING UP A LOAD BALANCER OR PROXY

This section discusses a number of things you need to configure before you can put a reverse proxy or load balancer in front of your clustered Red Hat Single Sign-On deployment. It also covers configuring the built in load balancer that was Clustered Domain Example.

7.3.1. Identifying Client IP Addresses

A few features in Red Hat Single Sign-On rely on the fact that the remote address of the HTTP client connecting to the authentication server is the real IP address of the client machine. This can be problematic when you have a reverse proxy or loadbalancer in front of your Red Hat Single Sign-On authentication server. The usual setup is that you have a frontend proxy sitting on a public network

that load balances and forwards requests to backend Red Hat Single Sign-On server instances that are located on a private network. There is some extra configuration you have to do in this scenario so that the actual client IP address is forwarded to and processed by the Red Hat Single Sign-On server instances. Specifically:

- Configure your reverse proxy (loadbalancer) to properly set **`X-Forwarded-For`** and **`X-Forwarded-Proto`** HTTP headers.
- Configure the authentication server to read the client's IP address from **`X-Forwarded-For header`**.

Configuring your proxy to generate the **`X-Forwarded-For`** and **`X-Forwarded-Proto`** HTTP headers is beyond the scope of this guide. Take extra precautions to ensure that the **`X-Forwared-For`** header is set by your proxy. If your proxy isn't configured correctly, then *rogue* clients can set this header themselves and trick Red Hat Single Sign-On into thinking the client is connecting from a different IP address than it actually is. This becomes really important if you are doing any black or white listing of IP addresses.

Beyond the proxy itself, there are a few things you need to configure on the Red Hat Single Sign-On side of things. If your proxy is forwarding requests via the HTTP protocol, then you need to configure Red Hat Single Sign-On to pull the client's IP address from the **`X-Forwarded-For`** header rather than from the network packet. To do this, open up the profile configuration file (*standalone.xml*, *standalone-ha.xml*, or *domain.xml* depending on your operating mode) and look for the **`"urn:jboss:domain:undertow:3.0`** XML block.

`X-Forwarded-For` HTTP Config

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
   <buffer-cache name="default"/>
   <server name="default-server">
      <ajp-listener name="ajp" socket-binding="ajp"/>
      <http-listener name="default" socket-binding="http" redirect-
socket="https"
          proxy-address-forwarding="true"/>
      ...
   </server>
   ...
</subsystem>
```

Add the **`proxy-address-forwarding`** attribute to the **`http-listener`** element. Set the value to **`true`**.

If your proxy is using the AJP protocol instead of HTTP to forward requests (i.e. Apache HTTPD + mod-cluster), then you have to configure things a little differently. Instead of modifying the **`http-listener`**, you need to add a filter to pull this information from the AJP packets.

`X-Forwarded-For` AJP Config

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
     <buffer-cache name="default"/>
     <server name="default-server">
         <ajp-listener name="ajp" socket-binding="ajp"/>
         <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
         <host name="default-host" alias="localhost">
```

```
            ...
            <filter-ref name="proxy-peer"/>
        </host>
    </server>
       ...
    <filters>
        ...
        <filter name="proxy-peer"
                class-
name="io.undertow.server.handlers.ProxyPeerAddressHandler"
                module="io.undertow.core" />
    </filters>
 </subsystem>
```

7.3.2. Enable HTTPS/SSL with a Reverse Proxy

Assuming that your reverse proxy doesn't use port 8443 for SSL you also need to configure what port HTTPS traffic is redirected to.

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
    ...
    <http-listener name="default" socket-binding="http"
        proxy-address-forwarding="true" redirect-socket="proxy-https"/>
    ...
</subsystem>
```

Add the **`redirect-socket`** attribute to the **`http-listener`** element. The value should be **`proxy-https`** which points to a socket binding you also need to define.

Then add a new **`socket-binding`** element to the **`socket-binding-group`** element:

```
<socket-binding-group name="standard-sockets" default-interface="public"
    port-offset="${jboss.socket.binding.port-offset:0}">
    ...
    <socket-binding name="proxy-https" port="443"/>
    ...
</socket-binding-group>
```

7.3.3. Using the Built-In Load Balancer

This section covers configuring the built in load balancer that is discussed in the Clustered Domain Example.

The Clustered Domain Example is only designed to run on one machine. To bring up a slave on another host, you'll need to

1. Edit the *domain.xml* file to point to your new host slave
2. Copy the server distribution. You don't need the *domain.xml*, *host.xml*, or *host-master.xml* files. Nor do you need the *standalone/* directory.
3. Edit the *host-slave.xml* file to change the bind addresses used or override them on the command line

7.3.3.1. Register a New Host With Load Balancer

Let's look first at registering the new host slave with the load balancer configuration in *domain.xml*. Open this file and go to the undertow configuration in the **`load-balancer`** profile. Add a new **`host`** definition called **`remote-host3`** within the **`reverse-proxy`** XML block.

domain.xml reverse-proxy config

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
  ...
  <handlers>
      <reverse-proxy name="lb-handler">
         <host name="host1" outbound-socket-binding="remote-host1"
scheme="ajp" path="/" instance-id="myroute1"/>
         <host name="host2" outbound-socket-binding="remote-host2"
scheme="ajp" path="/" instance-id="myroute2"/>
         <host name="remote-host3" outbound-socket-binding="remote-host3"
scheme="ajp" path="/" instance-id="myroute3"/>
      </reverse-proxy>
  </handlers>
  ...
</subsystem>
```

The **`output-socket-binding`** is a logical name pointing to a **`socket-binding`** configured later in the *domain.xml* file. the **`instance-id`** attribute must also be unique to the new host as this value is used by a cookie to enable sticky sessions when load balancing.

Next go down to the **`load-balancer-sockets`** **`socket-binding-group`** and add the **`outbound-socket-binding`** for **`remote-host3`**. This new binding needs to point to the host and port of the new host.

domain.xml outbound-socket-binding

```
<socket-binding-group name="load-balancer-sockets" default-
interface="public">
    ...
    <outbound-socket-binding name="remote-host1">
        <remote-destination host="localhost" port="8159"/>
    </outbound-socket-binding>
    <outbound-socket-binding name="remote-host2">
        <remote-destination host="localhost" port="8259"/>
    </outbound-socket-binding>
    <outbound-socket-binding name="remote-host3">
        <remote-destination host="192.168.0.5" port="8259"/>
    </outbound-socket-binding>
</socket-binding-group>
```

7.3.3.2. Master Bind Addresses

Next thing you'll have to do is to change the **`public`** and **`management`** bind addresses for the master host. Either edit the *domain.xml* file as discussed in the Bind Addresses chapter or specify these bind addresses on the command line as follows:

```
$ domain.sh --host-config=host-master.xml -
Djboss.bind.address=192.168.0.2 -
Djboss.bind.address.management=192.168.0.2
```

7.3.3.3. Host Slave Bind Addresses

Next you'll have to change the **public**, **management**, and domain controller bind addresses (**jboss.domain.master-address**). Either edit the *host-slave.xml* file or specify them on the command line as follows:

```
$ domain.sh --host-config=host-slave.xml
     -Djboss.bind.address=192.168.0.5
      -Djboss.bind.address.management=192.168.0.5
       -Djboss.domain.master.address=192.168.0.2
```

The values of **jboss.bind.address** and **jboss.bind.addres.management** pertain to the host slave's IP address. The value of **jboss.domain.master.address** need to be the IP address of the domain controller which is the management address of the master host.

7.3.4. Configuring Other Load Balancers

The the load balancer chapter of the JBoss EAP Configuration Guide has information on using some other software based load balancers that may help you.

7.4. MULTICAST NETWORK SETUP

Out of the box clustering support has a need to for IP Multicast. Multicast is a network broadcast protocol. This protocol is used at boot time to discover and join the cluster. It is also used to broadcast messages for the replication and invalidation distributed caches used by Red Hat Single Sign-On.

The clustering subsystem for Red Hat Single Sign-On runs on the JGroups stack. Out of the box, the bind addresses for clustering are bound to a private network interface with a default IP address of 127.0.0.1. You'll have to edit your the *standalone-ha.xml* or *domain.xml* sections discussed in the Bind Address chapter.

private network config

```
    <interfaces>
        ...
        <interface name="private">
            <inet-address
value="${jboss.bind.address.private:127.0.0.1}"/>
        </interface>
    </interfaces>
    <socket-binding-group name="standard-sockets" default-
interface="public" port-offset="${jboss.socket.binding.port-offset:0}">
        ...
        <socket-binding name="jgroups-mping" interface="private" port="0"
multicast-address="${jboss.default.multicast.address:230.0.0.4}"
multicast-port="45700"/>
        <socket-binding name="jgroups-tcp" interface="private"
```

```
port="7600"/>
        <socket-binding name="jgroups-tcp-fd" interface="private"
port="57600"/>
        <socket-binding name="jgroups-udp" interface="private"
port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-
port="45688"/>
        <socket-binding name="jgroups-udp-fd" interface="private"
port="54200"/>
        <socket-binding name="modcluster" port="0" multicast-
address="224.0.1.105" multicast-port="23364"/>
        ...
    </socket-binding-group>
```

Things you'll want to configure are the **`jboss.bind.address.private`** and **`jboss.default.multicast.address`** as well as the ports of the services on the clustering stack. Please see the JGroups chapter of the JBoss EAP Configuration Guide for more details.

Note: It is possible to cluster Red Hat Single Sign-On without IP Multicast, but this topic is beyond the scope of this guide. Please see the JGroups chapter of the JBoss EAP Configuration Guide.

7.5. SERIALIZED CLUSTER STARTUP

Red Hat Single Sign-On cluster nodes are allowed to boot concurrenty. When Red Hat Single Sign-On server instance boots up it may do some database migration, importing, or first time initializations. A DB lock is used to prevent start actions from conflicting with one another when cluster nodes boot up concurrently.

By default, the maximum timeout for this lock is 900 seconds. If a node is waiting on this lock for more than the timeout it will fail to boot. Typically you won't need to increase/decrease the default value, but just in case it's possible to configure it in **`keycloak-server.json`**:

```
"dblock": {
    "jpa": {
        "lockWaitTimeout": 900
    }
}
```

7.6. BOOTING THE CLUSTER

Booting Red Hat Single Sign-On in a cluster depends on your operating mode

Standalone Mode

```
$ bin/standalone.sh --server-config=standalone-ha.xml
```

Domain Mode

```
$ bin/domain.sh --host-config=host-master.xml
$ bin/domain.sh --host-config=host-slave.xml
```

7.7. TROUBLESHOOTING

Note that when you run cluster, you should see message similar to this in the log of both cluster nodes:

```
INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-10,shared=udp)
ISPN000094: Received new cluster view: [node1/keycloak|1] (2)
[node1/keycloak, node2/keycloak]
```

If you see just one node mentioned, it's possible that your cluster hosts are not joined together.

Usually it's best practice to have your cluster nodes on private network without firewall for communication among them. Firewall could be enabled just on public access point to your network instead. If for some reason you still need to have firewall enabled on cluster nodes, you will need to open some ports. Default values are UDP port 55200 and multicast port 45688 with multicast address 230.0.0.4. Note that you may need more ports opened if you want to enable additional features like diagnostics for your JGroups stack. Red Hat Single Sign-On delegates most of the clustering work to Infinispan/JGroups. Please consult the JGroups chapter of the JBoss EAP Configuration Guide.

CHAPTER 8. SERVER CACHE CONFIGURATION

Red Hat Single Sign-On has two types of caches. One type of cache sits in front of the database to decrease load on the DB and to increase overall response times by keeping data in memory. Realm, client, role, and user metadata is kept in this type of cache. This cache is an invalidation cache. Invalidation caches do not use replication. Instead, they only keep copies locally and if the entry is updated an invalidation message is sent to the rest of the cluster and the entry is evicted. This greatly reduces network traffic, makes things efficient, and avoids transmitting sensitive metadata over the ire.

The second type of cache handles managing user sessions, offline tokens, and keeping track of login failures so that the server can detect password phishing and other attacks. The data held in these caches is temporary, in memory only, but is possibly replicated across the cluster.

This chapter discusses some configuration options for these caches for both clustered a non-clustered deployments.

Note: More advanced configuration of these caches can be found in JBoss EAP Configuration Guide

8.1. EVICTION AND EXPIRATION

There are multiple different caches configured for Red Hat Single Sign-On. There is a realm cache that holds information about secured applications, general security data, and configuration options. This size of this cache is unbounded and does not have a limit on entries. This might scare you a little bit, but the number of entries in this cache is pretty low compared to the user cache. There is also a user cache that contains user metadata. It defaults to a maximum of 10000 entries and uses a least recently used eviction strategy. There are also separate caches for user sessions, offline tokens, and login failures. These caches are unbounded in size as well.

The eviction policy and max entries for these caches can be configured in the *standalone.xml*, *standalone-ha.xml*, or *domain.xml* depending on your operating mode.

non-clustered

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
   <cache-container name="keycloak" jndi-name="infinispan/Keycloak">
       <local-cache name="realms"/>
       <local-cache name="users">
          <eviction max-entries="10000" strategy="LRU"/>
       </local-cache>
       <local-cache name="sessions"/>
       <local-cache name="offlineSessions"/>
       <local-cache name="loginFailures"/>
       <local-cache name="work"/>
       <local-cache name="realmVersions">
          <transaction mode="BATCH" locking="PESSIMISTIC"/>
       </local-cache>
   </cache-container>
```

clustered

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
```

```
    <cache-container name="keycloak" jndi-name="infinispan/Keycloak">
        <transport lock-timeout="60000"/>
        <invalidation-cache name="realms" mode="SYNC"/>
        <invalidation-cache name="users" mode="SYNC">
            <eviction max-entries="10000" strategy="LRU"/>
        </invalidation-cache>
        <distributed-cache name="sessions" mode="SYNC" owners="1"/>
        <distributed-cache name="offlineSessions" mode="SYNC" owners="1"/>
        <distributed-cache name="loginFailures" mode="SYNC" owners="1"/>
        <replicated-cache name="work" mode="SYNC"/>
        <local-cache name="realmVersions">
            <transaction mode="BATCH" locking="PESSIMISTIC"/>
        </local-cache>
    </cache-container>
```

To limit or expand the number of allowed entries simply add, edit, or remove the **`eviction`** element from the **`invalidation-cache`** or **`distributed-cache`** declaration you want to change.

8.2. REPLICATION AND FAILOVER

The **`sessions`**, **`offlineSessions`** and **`loginFailures`** caches are the only caches that may perform replication. Entries are not replicated to every single node, but instead one or more nodes is chosen as an owner of that data. If a node is not the owner of a specific cache entry it queries the cluster to obtain it. What this means for failover is that if all the nodes that own a piece of data go down, that data is lost forever. By default, Red Hat Single Sign-On only specifies one owner for data. So if that one node goes down that data is lost. This usually means that users will be logged out and will have to login again.

You can change the number of nodes that replicate a piece of data by change the **`owners`** attribute in the **`distributed-cache`** declaration.

owners

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
    <cache-container name="keycloak" jndi-name="infinispan/Keycloak">
        <distributed-cache name="sessions" mode="SYNC" owners="2"/>
...
```

Here's we've changed it so at least two nodes will replicate one specific user login session.

Tip

The number of owners recommended is really dependent on your deployment. If you do not care if users are logged out when a node goes down, then one owner is good enough and you will avoid replication.

8.3. DISABLING CACHING

To disable the realm or user cache, you must edit the **`keycloak-server.json`** file in your distribution. Where this file lives depends on your operating mode Here's what the config looks like initially.

```
    "userCache": {
        "default" : {
            "enabled": true
        }
    },

    "realmCache": {
        "default" : {
            "enabled": true
        }
    },
```

To disable the cache set the **enabled** field to false for the cache you want to disable. You must reboot your server for this change to take effect.

8.4. CLEARING CACHES AT RUNTIME

To clear the realm or user cache, go to the Red Hat Single Sign-On admin console Realm Settings→Cache Config page. On this page you can clear the realm cache or the user cache. This will clear the caches for all realms and not only the selected realm.